Université catholique de Louvain (UCL)
Institute of Information and Communication Technologies,
Electronics and Applied Mathematics (ICTEAM)
Louvain Verification Lab (LVL)

# Symbolic model checking of multi-modal logics: uniform strategies and rich explanations

Simon Busard

*Thesis submitted in partial fulfillment of the requirements for the Degree of Doctor in Engineering Sciences*

July 2017

# ***Abstract***

*Model checking* is a verification technique that performs an *exhaustive search* among the states of safety-critical systems to check whether a given *property* is satisfied. These properties are usually expressed within a *logic* that captures different aspects of the system such as its evolution through time.

*Multi-modal logics* mix several aspects of the system such as the knowledge and strategies of its agents. They usually are *branching logics*, that is, they can express properties about several successors of the states of interest. Nevertheless, logics to reason about the strategies of agents with an imperfect view of a system under fairness constraints have been seldom considered, and model-checking algorithms appeared only recently.

In this thesis, we first define $ATLK_{irF}$, a multi-modal logic reasoning about time, knowledge and uniform strategies of concurrent agents under unconditional fairness constraints. This logic can be used to reason about *multi-agent programs* under the supervision of a *fair scheduler*.

We then describe three approaches to solve its model-checking problem. They are all based on an explicit enumeration of the strategies. The first one simply enumerates and checks all uniform strategies of the agents. The second one limits this enumeration to *partial strategies*, and uses early termination and caching to improve its performances in practice. The last one performs a *backward* exploration of the system to directly build the winning strategies. Furthermore, we present variants of these approaches based on pre-filtering losing moves. Finally, we implement and compare them with state-of-the-art symbolic solutions. The experiments show that the different approaches outperform the others in different situations.

Second, we attack the problem of generating and manipulating *rich explanations* for multi-modal logics. One of the main advantages of model checking is its capability to produce a counter-example showing why the checked property is violated. But multi-modal logics have rich and complex explanations, and state-of-the-art model checkers such as NuSMV provide only partial explanations.

We thus present a µ-calculus based model-checking framework. The *propositional $\mu$-calculus* is a logic integrating modal operators and least and greatest fixpoint operators. The goal of this framework is to help a designer to solve her top-level model-checking problem by translating it into µ-calculus. It integrates a µ-calculus model checker with rich explanations. It also integrates a set of functionalities to help the designer to translate these explanations back into the top-level language, such as *formula aliases*, a *relational graph algebra*, and *choosers* to guide the generation process. The framework is then used on the case of $ATL$, the standard logic to reason about the strategies of the agents of a system, to show its applicability.

# *Acknowledgements*

First of all, I would like to thank Charles for the opportunity to pursue this PhD. I thank him for his guidance and advice through this adventure. I am also grateful to Prof. Franco Raimondi who introduced me to the world of strategic reasoning. I am thankful to the members of my jury for their useful comments and questions.

Many thanks to Antoine, who shared lunchtime, an office, long discussions and many other activities for these last twelve years, from our bachelor degree to today. I also thank my INGI friends, Xavier, Jérôme, Samuel, Jean-Baptiste, and the others, who spent much valuable time with me for these last twelve years as well.

I am grateful to all my colleagues from the INGI pole, and particularly to my colleagues from the Louvain Verification Lab, Sébastien, José, Christophe, Guillaume, Xavier, and Hossein. Our discussions, technical or not, were always a pleasure.

I also thank my family and friends, who supported me during these years of research. And, last but not least, I mostly thank Gaëlle for the support she gave me. We overcame many obstacles together, and will continue to do so.

# *Contents*

# *Chapter 1*

# *Introduction*

Safety-critical systems are everywhere in our daily life. They are embedded in planes, cars and trains, but also lie in health-related systems such as pacemakers and radiotherapy machines. These systems must be safe and bug-free since a fault can put the life of their users in danger, or harm the economical future of their builders.

Techniques are necessary to ensure that such systems really do what they are intended to. Many methods exist, from testing and simulation to formal methods. In particular, verification techniques are designed to make sure that a given system satisfies a particular property. Model checking is one of them. It mainly applies to reactive systems, that is, systems that react to their environment and exhibit infinite behaviors, instead of merely computing an output value in response to their inputs.

In essence, model checking performs an exhaustive search to verify that all the possible behaviors satisfy the property. More precisely, it takes a model of the system, usually represented by a (finite) state-transition graph, and a property to check, usually expressed in a logic. Then it performs an exhaustive search among the states and paths of the graph to check that it satisfies the property [CGP99, BK08]. Compared to other verification techniques such as theorem proving, model checking is fully automatic and can extract an explanation of why the property is satisfied (or not) by the model.

Since the 80s, many logics have been proposed to address different aspects of the system. The first ones were designed to reason about the temporal evolution of the system: *Linear Temporal Logic* (*LTL*) expresses properties about the execution traces of the system [MP92], while *Computation Tree Logic* (*CTL*) expresses properties about its computation tree [CE81]. Starting from these logics, many others have been designed to reason about other aspects of the system, such as (the evolution of) the knowledge of the agents of the system [FHMV95, PL03, vdMS99], or the strategies they can play to achieve their goals [AHK02].

To illustrate the concepts and techniques presented in this thesis, we will use the simple example of a card game derived from [JvdH04]. This game is explained in Example 1.1 and will be referred as the *simple card game* in the sequel; the variant of the game that is repeated infinitely many times will be referred as the *repeated card game*.

**Example 1.1.** *The game is played with three cards $A$, $K$ and $Q$, between a player and a dealer. $A$ wins over $K$, $K$ wins over $Q$, and $Q$ wins over $A$. First, the dealer gives one card to the player, keeps one and leaves the last one on table, face down. Then the player can keep his card or swap it with the one on the table. Finally, the player wins if his card wins over the dealer's.*

*Branching logics* express properties about the branching structure of the system. For instance, $CTL$ allows the user to express the fact that *all plays of the card game will lead to an end*, or that *there exists no path to a state in which all successor states are winning for the player.*

Many extensions of $CTL$ have been proposed to take into account other aspects of the verified systems. The *Computation Tree Logic of Knowledge* ($CTLK$) extends $CTL$ to express properties about time and knowledge altogether [PL03]. Within this logic, we can express properties such as *in every state of the game, the player knows his card*, or *after the first step of the game, the player does not know the card of the dealer.*

Another extension of $CTL$ is the *Alternating-time Temporal Logic* ($ATL$), that reasons about the strategies the agents of the system have to achieve some objectives [AHK02]. For instance, we can say that *the player has a strategy to win the card game*, or that *the dealer can choose to give the $Q$ to the player.* Many variants of ATL have been proposed to reason about the strategies that agents with limited knowledge of the system can actually play [JvdH04]. In particular, $ATL_{ir}$ [Sch04] can be seen as the minimal $ATL$-based logic to reason about uniform memoryless strategies [JD06], that is, strategies of agents that base their choices on the observations they can make on the current state of the system. Such a logic is well-suited for the card game since the player does not know the card of the dealer before the end of the game. It is also well-suited to reason about strategies of multi-agent programs [DJ10], or to check and synthesize security policies, for instance.

Logics such as $CTLK$ and $ATL$ can be translated into the *propositional $\mu$-calculus* [Koz83]. This logic is based on the notion of fixpoints and allows the user to express properties such as *for every game instance, there is always a way for the player to lose or win.*

Different techniques have been proposed to solve the model-checking problem of these logics. The first ones to appear were *explicit* techniques in the sense that they handle the model as a state-transition graph defined by explicit sets of states and transitions [CE81].

*Symbolic* techniques appeared later. Instead of representing the model explicitly, they represent sets of states and transitions with particular data structures such as *binary decision diagrams* (BDDs) [Bry86] or *satisfiability problems* [BCM$^{+}$90, BCCZ99]. Binary decision diagrams work well with branching logics, as shown by their model-checking algorithms. These logics are state-based in the sense that their properties are interpreted over states of the model, and the main advantage of BDDs is that they can compactly represent large sets of states.

The production of an explanation of why a property is satisfied by a given model is one of the claimed advantages of model checking. Nevertheless, only few works attacked the problem of generating and visualizing them in the case of branching logics. Indeed, these explanations can be very rich as, in general, branching logics need branching counter-examples [BEGL01]. For instance, to show that *the player has a strategy to eventually know the card of the dealer*, we need to exhibit a strategy for him, that is, for each state of the system, the action the player must play. Furthermore, we also need to show that this strategy really achieves its goal by showing that any play of the game following the strategy leads to a state in which the player knows the card of the dealer. The generation, as well as the presentation of explanations for branching logics are thus difficult problems that have got little attention in the past.

## 1.1 Research goals

The overall goal of this thesis is to improve the generation and presentation of explanations in model checking of branching logics. More precisely, this improvement is achieved by tackling two more specific problems: (1) the model checking of uniform strategies and (2) the development of methods to generate, visualize and manipulate explanations for μ-calculus based branching logics.

### 1.1.1 Model checking uniform strategies

To illustrate the first problem, let us go back to the example of the card game. We are interested in whether the player has a strategy to win the game, given that he does not see the card of the dealer before the end. This amounts to finding a winning *uniform* strategy for the player, that is, a strategy that defines the same action in states that are indistinguishable from the player. In the present game, the player has no uniform strategy to win.

To go further, we can consider that the game is played infinitely. Even in this situation, the player cannot eventually win a play since, whatever he plans to do, the dealer can give him the wrong hand. Nevertheless, if we consider a fair dealer, that is, a dealer who gives every possible hand infinitely often, then any strategy of the player is winning as he is sure to eventually get a winning hand.

Logics reasoning about uniform strategies have gathered a lot of attention in the last decade (see for instance [JvdH04, Sch04, JÅ07]). Nevertheless, only few works investigated the development of practical algorithms to perform their model checking. More precisely, an explicit algorithm for $ATL_{ir}$ was proposed by Calta et al. a few years ago [CSS10], but symbolic approaches appeared only recently [PBJ14, HvdM14b]. The first goal of this thesis is thus to propose a BDD-based algorithm for model checking uniform strategies.

To this end, we first describe $ATLK_{irF}$, a new extension of $ATL_{ir}$ for reasoning about uniform strategies under unconditional fairness constraints. This logic expresses properties about (branching) time, knowledge, and uniform strategies of agents under fairness constraints. These fairness constraints can be useful, for instance, to model a fair scheduler.

A second contribution is the design of several BDD-based model-checking algorithms for $ATLK_{irF}$. These algorithms are based on an explicit enumeration of uniform strategies represented with BDDs. While the first algorithm simply checks all these strategies to find a winning one, some improvements are made by restricting the search to *partial* uniform strategies, that is, strategies that are not specified for all the states of the model. Other improvements are made by exploring and enumerating the strategies starting from the objective states, and by pre-filtering losing moves.

Third, we describe an implementation of the algorithms and a comparison with other state-of-the-art symbolic approaches for model checking $ATL_{ir}$.

### 1.1.2 Handling rich explanations

To illustrate the second problem, let us consider the property that *in every play of the card game, the player knows the dealer's card before swapping or keeping his cards.* The property is obviously violated as the player does not see the card of the dealer before the end of the game. Showing that the property is false amounts to showing a play of the game ending in a state in which the player cannot distinguish two states with different cards for the dealer.

While linear logics such as $LTL$ reason about single executions of the systems—and thus have explanations consisting in single executions—branching logics have explanations that must show different branches of the execution tree of the system to fully explain the truth value of the property.

Nevertheless, current state-of-the-art tools such as NuSMV only produce single executions of the model when explaining why a property is violated by the model [CCG+02]. The second goal of this thesis is thus to propose techniques and tools to generate, visualize and manipulate explanations for μ-calculus based logics.

To this end, we give a definition of rich explanations for μ-calculus properties, as well as a BDD-based algorithm to generate them. Based on these explanations, a framework is developed to easily generate and modify explanations for logics that can be translated into the μ-calculus, such as $CTL$, $ATL$ and $CTLK$. The framework provides a μ-calculus model checker that generates rich explanations. Furthermore, practical solutions to handle the translation and the richness of such explanations are presented, such as formula aliases, explanation node attributes, or formula markers. Finally, an implementation of the framework in Python is described, as well as a tool to visualize and manipulate the explanations.

This framework takes inspiration from preliminary work on rich explanations for $CTLK$ [BP12]. These explanations are tree-like annotated counter-examples that exhibit the paths of the system explaining why $CTLK$ sub-formulas are satisfied.

## 1.2 Contributions

In summary, the contributions for solving the two problems identified in the previous section are:

- the definition of $ATLK_{irF}$, a new logic mixing time, knowledge and uniform strategies under unconditional fairness constraints;
- three symbolic approaches for solving its model-checking problem, with variants based on pre-filtering losing moves;
- an implementation of these algorithms and a comparison with other state-of-the-art symbolic approaches;

- a definition of explanations for µ-calculus properties, as well as a BDD-based algorithm to generate them;
- a framework to generate and modify explanations for logics that can be translated into the µ-calculus, with features to help the translation;
- an implementation of the framework and a tool to visualize and manipulate the explanations.

The tools have been built with PyNuSMV, a NuSMV-based Python library developed to support the implementation of the ideas presented in this thesis [BP13].

## 1.3 Publications

Most of the work has already been published in

[BP12] Simon Busard and Charles Pecheur. Rich counter-examples for temporal-epistemic logic model checking. In *Proceedings Second International Workshop on Interactions, Games and Protocols, IWIGP 2012, Tallinn, Estonia, 25th March 2012*, pages 39–53, 2012.

[BPQR13] Simon Busard, Charles Pecheur, Hongyang Qu, and Franco Raimondi. Reasoning about strategies under partial observability and fairness constraints. In Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi, editors, *Proceedings 1st International Workshop on Strategic Reasoning, SR 2013, Rome, Italy, March 16-17, 2013*, volume 112 of *EPTCS*, pages 71–79, 2013.

[BP13] Simon Busard and Charles Pecheur. PyNuSMV: NuSMV as a Python library. In Guillaume Brat, Neha Rungta, and Arnaud Venet, editors, *Nasa Formal Methods 2013*, volume 7871 of *LNCS*, pages 453–458. Springer-Verlag, 2013.

[BPQR14] Simon Busard, Charles Pecheur, Hongyang Qu, and Franco Raimondi. Improving the model checking of strategies under partial observability and fairness constraints. In Stephan Merz and Jun Pang, editors, *Formal Methods and Software Engineering*, volume 8829 of *Lecture Notes in Computer Science*, pages 27–42. Springer International Publishing, 2014.

[BPQR15] Simon Busard, Charles Pecheur, Hongyang Qu, and Franco Raimondi. Reasoning about memoryless strategies under partial observability and unconditional fairness constraints. *Information and Computation*, 242:128 – 156, 2015.

The model-checking algorithm for uniform strategies based on backward generation, the comparison of symbolic approaches for model checking uniform strategies and the work on the framework for µ-calculus based logics explanations have not been published yet.

## 1.4 Structure of the thesis

Chapter 2 describes the background material. Part I presents the contributions about model checking uniform strategies under fairness constraints. Chapter 3 presents related work on the subject. Chapter 4 describes the logic $ATLK_{irF}$ itself, its syntax and semantics, and the complexity of its model-checking problem. Chapter 5 proposes several BDD-based algorithms for model checking the logic. Chapter 6 presents other recent symbolic approaches. Chapter 7 experimentally compares all the presented algorithms. Chapter 8 draws some conclusions about this first part.

Part II presents the contributions about the explanations of rich logics. Chapter 9 presents related work on the subject. Chapter 10 describes the framework for µ-calculus based logics explanations and its implementation in Python. Chapter 11 draws some conclusions about the second part.

Finally, Chapter 12 concludes the thesis, shows the link between the two presented parts, and draws some perspectives.

# *Chapter 2*

# *Background*

This chapter presents the background material the rest of the thesis relies on. First, it describes several logics that will be referred to in the text. Then it discusses their symbolic model checking. Finally it presents the tools used to implement the different techniques and to perform the experiments.

## 2.1 Rich logics

This section presents *Computation Tree Logic* and some of its extensions to reason about strategies, as well as the *propositional $\mu$-calculus*, a fixpoint-based logic into which all the previous ones can be translated.

### 2.1.1 Computation tree logic

Computation tree logic ($CTL$) reasons about the possible future executions of a system [CE81]. The syntax of the formulas follows the grammar

$$\begin{aligned}\phi &::= true \mid p \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{E}\ \psi \\ \psi &::= \mathbf{X}\ \phi \mid \phi\ \mathbf{U}\ \phi \mid \phi\ \mathbf{W}\ \phi\end{aligned}$$

where $p$ is an atomic proposition from a fixed set $AP$. Formulas $\phi$ (resp. $\psi$) are called *state* (resp. *path*) formulas.

Other operators can be defined in terms of the previous ones:

$$\begin{aligned}false &\equiv \neg true \\ \phi_1 \wedge \phi_2 &\equiv \neg(\neg\phi_1 \vee \neg\phi_2)\end{aligned}$$

$$\begin{aligned}
\phi_1 \implies \phi_2 &\equiv \neg\phi_1 \vee \phi_2\\
\phi_1 \iff \phi_2 &\equiv (\phi_1 \implies \phi_2) \wedge (\phi_2 \implies \phi_1)\\
\mathbf{A}\ \psi &\equiv \neg\mathbf{E}\ \neg\psi\\
\mathbf{F}\ \phi &\equiv \mathit{true}\ \mathbf{U}\ \phi\\
\mathbf{G}\ \phi &\equiv \phi\ \mathbf{W}\ \mathit{false}
\end{aligned}$$

**E** and **A** operators are called *path quantifiers* and **X**, **U**, **W**, **G** and **F** are *path operators.*

Intuitively, $\mathbf{E}[\phi_1\ \mathbf{W}\ \phi_2]$ means that there exists a path such that all states of the path satisfy $\phi_1$ unless $\phi_2$ has been true before. On the other hand, $\mathbf{A}[\phi_1\ \mathbf{U}\ \phi_2]$ means that $\phi_2$ will eventually be true, for all possible behaviors, and $\phi_1$ will be true in the meantime.

For instance, the fact that *all plays of the card game will lead to an end* can be translated into *CTL* as **AF** *end*. The fact that *there exists no path to a state in which all successor states are winning for the player* can be written as $\neg$**EF AX** *win*.

The **U** and **W** operators can be derived from one another:

$$\begin{aligned}
\neg(\phi_1\ \mathbf{U}\ \phi_2) &\equiv (\neg\phi_2)\ \mathbf{W}\ (\neg\phi_1 \wedge \neg\phi_2),\\
\neg(\phi_1\ \mathbf{W}\ \phi_2) &\equiv (\neg\phi_2)\ \mathbf{U}\ (\neg\phi_1 \wedge \neg\phi_2).
\end{aligned}$$

Nevertheless, both operators are needed if we want to derive *positive normal forms* (PNF) of *CTL* formulas—that is, equivalent formulas in which the negation $\neg$ only applies to atomic propositions—as the formula $\neg\mathbf{E}[\phi_1\ \mathbf{U}\ \phi_2]$ has no PNF without the **W** operator.

*CTL* formulas are interpreted over the states of *Kripke structures* $S = \langle Q, Q_0, R, V\rangle$ where

- $Q$ is a set of states;
- $Q_0 \subseteq Q$ is the set of initial states;
- $R \subseteq Q \times Q$ is a serial transition relation; we write $q \to q'$ for $\langle q, q'\rangle \in R$;
- $V : Q \to 2^{AP}$ is a labeling function determining the set of atomic propositions true in each state.

We call a *path* of $S$ an infinite sequence $\pi = q_0, q_1, \ldots$ such that $q_d \to q_{d+1}$ for all $d \geq 0$; we write $\pi(d)$ for $q_d$. We say that a path $\pi$ starts in $q$ if $\pi(0) = q$.

The Kripke structure for the simple card game is illustrated in Figure 2.1. The states are decorated with the cards of the player and the dealer, respectively, and each row of states belong to the same game step. Atomic propositions include $pcard = \mathcal{C}$ and $dcard = \mathcal{C}$ to refer to the fact that the player (resp. the dealer) has card $\mathcal{C} \in \{A, K, Q\}$, and the propositions *win* and *lose* label the states in which the player wins the game (resp. loses it).

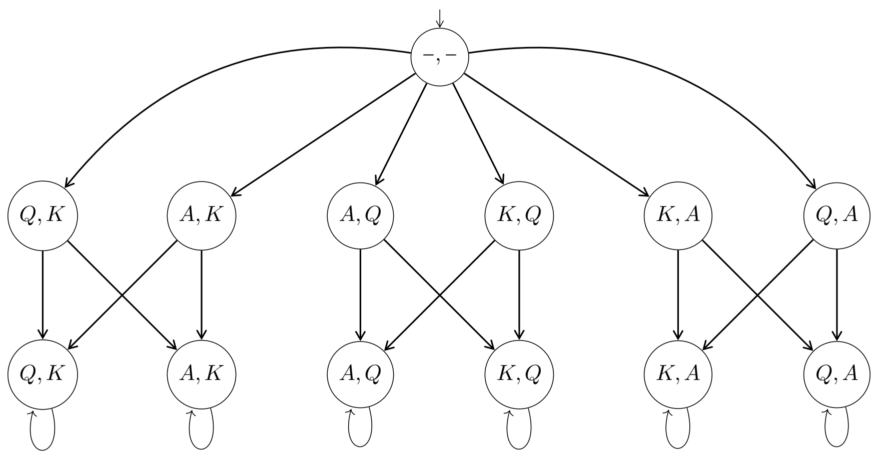


Figure 2.1: The Kripke structure for the card game. Circles are states, with pairs of cards ($K, A$ means player has $K$, dealer has $A$). Arrows are transitions. Atomic propositions are not pictured but are easily inferred.

The semantics of $CTL$ formulas is defined over the states of $S$ by the relation $S, q \vDash_{CTL} \phi$. We say that $q$ satisfies $\phi$ in $S$ when $S, q \vDash_{CTL} \phi$. This relation is defined as

$$\begin{array}{ll} S, q \vDash_{CTL} true, & \\ S, q \vDash_{CTL} p & \Leftrightarrow p \in V(q), \\ S, q \vDash_{CTL} \neg\phi & \Leftrightarrow S, q \nvDash_{CTL} \phi, \\ S, q \vDash_{CTL} \phi_1 \vee \phi_2 & \Leftrightarrow S, q \vDash_{CTL} \phi_1 \text{ or } S, q \vDash_{CTL} \phi_2, \\ S, q \vDash_{CTL} \mathbf{E}\ \psi & \Leftrightarrow \begin{cases} \text{there exists a path } \pi \text{ of } S \text{ starting in } q \\ \text{s.t. } S, \pi \vDash_{CTL} \psi. \end{cases} \end{array}$$

The relation $S, \pi \vDash_{CTL} \psi$ on paths is defined as

$$\begin{array}{ll} S, \pi \vDash_{CTL} \mathbf{X}\ \phi & \Leftrightarrow S, \pi(1) \vDash_{CTL} \phi, \\ S, \pi \vDash_{CTL} \phi_1\ \mathbf{U}\ \phi_2 & \Leftrightarrow \begin{cases} \text{there exists } d \geq 0 \text{ s.t. } S, \pi(d) \vDash_{CTL} \phi_2 \\ \text{and for all } c < d, S, \pi(c) \vDash_{CTL} \phi_1, \end{cases} \\ S, \pi \vDash_{CTL} \phi_1\ \mathbf{W}\ \phi_2 & \Leftrightarrow S, \pi \vDash_{CTL} \phi_1\ \mathbf{U}\ \phi_2 \text{ or } \forall d \geq 0, S, \pi(d) \vDash_{CTL} \phi_1. \end{array}$$

Given a structure $S = \langle Q, Q_0, R, V\rangle$ and a $CTL$ formula $\phi$, we say that $S$ satisfies $\phi$—and write $S \vDash_{CTL} \phi$—if all initial states of $S$ satisfy $\phi$, that is, if

$$\forall q \in Q_0, S, q \vDash_{CTL} \phi.$$

For instance, the Kripke structure of Figure 2.1 satisfies **EF** *win* because the right-most path (among others) leads to the third-step state $\langle Q, A\rangle$ in which the player wins.

$CTL$ restricts the path operators to be immediately preceded by a path quantifier. Removing this restriction gives $CTL^*$, in which we can express properties such as *there exists a path along which the player wins and the dealer never gets an A*, written

$$\mathbf{E}\ (\mathbf{F}\ win \wedge \mathbf{G}\ dcard \neq A).$$

### 2.1.2 Fair computation tree logic

Fair $CTL$ ($FCTL$) is a variant of $CTL$ that quantifies over *fair* paths only [CES86]. More precisely, $FCTL$ formulas follow the same syntax as $CTL$ ones, but are interpreted over states of *fair* Kripke structures $S = \langle Q, Q_0, R, V, FC\rangle$ where $\langle Q, Q_0, R, V\rangle$ is a $CTL$ Kripke structure and $FC \subseteq 2^Q$ is a set of *unconditional fairness constraints.* A *fair* path $\pi$ of $S$ is a path such that, for each fairness constraint $fc \in FC$, there exist infinitely many positions $d$ such that $\pi(d) \in fc$. We call a *fair state* a state from which there exists a fair path.

The semantics of $FCTL$ is the same as for $CTL$, except for the path quantifier **E**:

$$S, q \vDash_{FCTL} \mathbf{E}\ \psi \Leftrightarrow \begin{cases} \text{there exists a } \textit{fair} \text{ path } \pi \text{ starting in } q \\ \text{s.t. } S, \pi \vDash_{FCTL} \psi, \end{cases}$$

where the relation $S, \pi \vDash_{FCTL} \psi$ follows the relation $S, \pi \vDash_{CTL} \psi$.

Figure 2.2 presents the repeated card game with a fair dealer modeled by adding one fairness constraint per second-step state. In this Kripke structure, the initial state $\langle \_, \_\rangle$ satisfies **AF** $dcard = A$ under the semantics of Fair $CTL$ because every fair path of the model must go through all given pairs of cards.

### 2.1.3 Alternating-time temporal logic

Alternating-time Temporal Logic ($ATL$) extends $CTL$ by replacing the path quantifiers **E** and **A** with coalition modalities $\langle\!\langle \Gamma \rangle\!\rangle$ and $[\![\Gamma]\!]$ reasoning

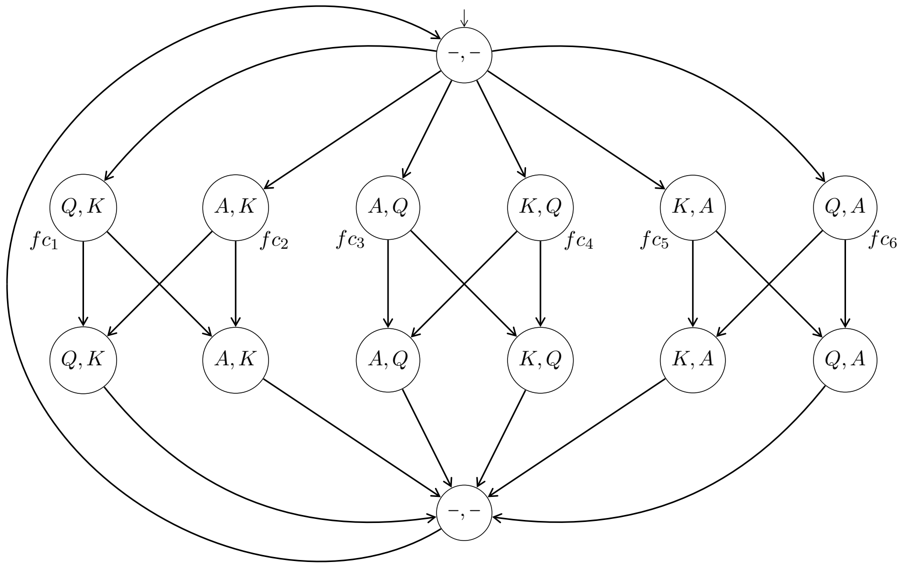


Figure 2.2: The fair Kripke structure of the repeated card game. Fairness constraints $fc_i$ label the intermediate states.

about the strategies of groups of agents $\Gamma$ to enforce an objective [AHK02]. More precisely, $ATL$ formulas follow the grammar

$$\phi ::= true \mid p \mid \neg\phi \mid \phi \vee \phi \mid \langle\!\langle \Gamma \rangle\!\rangle\ \psi$$

where $p$ is an atomic proposition from a fixed set $AP$, $\Gamma$ is a set of agents, and $\psi$ path formulas follow the same grammar as for $CTL$. The logic includes the derived path operators of $CTL$ $\mathbf{F}$ and $\mathbf{G}$, and keeps the duality between the two strategic modalities, that is, $\langle\!\langle \Gamma \rangle\!\rangle\ \psi \equiv \neg[\![\Gamma]\!]\ \neg\psi$. Furthermore, $\langle\!\langle \Gamma \rangle\!\rangle\mathbf{U}$ and $\langle\!\langle \Gamma \rangle\!\rangle\mathbf{W}$ operators can be translated into one another:

$$\begin{aligned}\neg\langle\!\langle \Gamma \rangle\!\rangle[\phi_1\ \mathbf{U}\ \phi_2] &\equiv [\![\Gamma]\!][(\neg\phi_2)\ \mathbf{W}\ (\neg\phi_1 \wedge \neg\phi_2)],\\ \neg\langle\!\langle \Gamma \rangle\!\rangle[\phi_1\ \mathbf{W}\ \phi_2] &\equiv [\![\Gamma]\!][(\neg\phi_2)\ \mathbf{U}\ (\neg\phi_1 \wedge \neg\phi_2)].\end{aligned}$$

Intuitively, $\langle\!\langle \Gamma \rangle\!\rangle\ \psi$ means that the agents of $\Gamma$ have a collective strategy such that all resulting paths satisfy $\psi$, that is, $\Gamma$ can enforce $\psi$. On the other hand, $[\![\Gamma]\!]\ \psi$ means that $\Gamma$ have no collective strategy such that no resulting path satisfies $\psi$, that is, $\Gamma$ cannot avoid $\psi$.

For instance, the fact that *the player has a strategy to win the game* is translated as $\langle\!\langle player \rangle\!\rangle\mathbf{F}\ win$, and *the dealer can give the A to the player* is written $\langle\!\langle dealer \rangle\!\rangle\mathbf{X}\ pcard = A$.

*ATL* formulas are interpreted over the states of *concurrent game structures* (CGS)[1] $S = \langle Ag, Q, Q_0, Act, e, \delta, V \rangle$ where

- $Ag$ is a set of agents;
- $Q$ is a set of states;
- $Q_0 \subseteq Q$ is the set of initial states;
- $Act$ is a set of actions; a *joint action* is a tuple of actions, one for each agent;
- $e : Ag \to (Q \to (2^{Act} \backslash \varnothing))$ defines, for each agent $ag$ and state $q$, the non-empty set of actions $ag$ can choose in $q$, that is, actions *enabled* in $q$; we write $e_{ag}$ for $e(ag)$;
- $\delta : Q \times Act^{Ag} \nrightarrow Q$ is a partial deterministic transition function[2] defined for all states $q \in Q$ and all joint actions enabled in $q$; we write $q \xrightarrow{a} q'$ for $\delta(q, a) = q'$;
- $V : Q \to 2^{AP}$ labels the states of $Q$ with atomic propositions.

A joint action $a \in Act^{Ag}$ *completes* an action $a_\Gamma \in Act^\Gamma$ for agents in $\Gamma$, written $a_\Gamma \sqsubseteq a$, if the actions of $\Gamma$ in $a$ correspond to the ones in $a_\Gamma$. From $e$ we define $E : 2^{Ag} \to (Q \to 2^{Act^{Ag}})$ as $E(\Gamma)(q) = \prod_{ag \in \Gamma} e_{ag}(q)$, that returns the set of actions for $\Gamma$ enabled in $q$. A *path* of $S$ is an infinite sequence $\pi = q_0, a_1, q_1, a_2 ...$ such that $\delta(q_d, a_{d+1}) = q_{d+1}$ for all $d \geq 0$.

A strategy for agent $ag$ is a function $f_{ag} : Q^+ \to Act$ such that $\forall q_0, ..., q_n \in Q^+, f_{ag}(q_0, ..., q_n) \in e_{ag}(q_n)$. We write $F_{ag}$ for the set of strategies of agent $ag$. The *outcomes* of a strategy $f_{ag}$ from a state $q$ are the set of paths resulting in applying the strategy and are defined as

$$out(f_{ag}, q) = \{q_0, a_1, q_1, a_2 ... \mid q_0 = q \land \forall d \geq 0, f_{ag}(q_0, ..., q_d) \sqsubseteq a_{d+1}\}$$

Finally, a strategy $f_\Gamma$ for a set of agents $\Gamma$ is a tuple of strategies, one for each agent of $\Gamma$, and the outcomes of $f_\Gamma$ is the intersection of outcomes of each strategies, that is, $out(f_\Gamma, q) = \bigcap_{f_{ag} \in f_\Gamma} out(f_{ag}, q)$. We write $F_\Gamma$ for the set of strategies of the group $\Gamma$.

[1]Alur et al. originally defined the semantics of *ATL* on states of *alternating transition systems* [AHK98]; the two classes of models are equivalent [GJ04].

[2]Imposing a deterministic transition function is not an actual restriction. A model with a non-deterministic transition relation can be translated into a model with a deteministic transition function by adding a new agent that breaks the non-determinism [BPQR15].

The semantics of $ATL$ is defined over the states of a CGS $S$ by the relation $S, q \vDash_{ATL} \phi$. This relation is the same as the $CTL$ one for atomic propositions and Boolean connectives. For the coalition operator $\langle\!\langle \Gamma \rangle\!\rangle\ \psi$, the relation is defined as

$$S, q \vDash_{ATL} \langle\!\langle \Gamma \rangle\!\rangle\ \psi \Leftrightarrow \begin{cases} \text{there exists a strategy } f_\Gamma \text{ for } \Gamma \\ \text{s.t. } \forall \pi \in out(f_\Gamma, q), S, \pi \vDash_{ATL} \psi, \end{cases}$$

where the relation $S, \pi \vDash_{ATL} \psi$ follows the $CTL$ one. The relation $S \vDash_{ATL} \phi$ is defined in the standard way.

As for $CTL$, we can remove from $ATL$ the restriction that path operators must be immediately preceded by a strategic operator, giving the logic $ATL^*$. In this logic, we can express properties such as *the player and the dealer have a strategy to make the player win and to avoid giving an A to the dealer*, written as

$$\langle\!\langle player, dealer \rangle\!\rangle\ (\mathbf{F}\ win \wedge \mathbf{G}\ dcard \neq A).$$

$ATL$ subsumes $CTL$ in the sense that every $CTL$ formula can be translated into an equivalent $ATL$ formula. Indeed, the **E** modality corresponds to the $\langle\!\langle Ag \rangle\!\rangle$ one, and the **A** modality corresponds to the $[\![Ag]\!]$ one.

### 2.1.4 Alternating-time temporal logic with imperfect information

$ATL_{ir}$ is an extension of $ATL$ restricted to memoryless uniform strategies, that is, strategies that choose an action for an agent based on the observations the agent can make on the current state [Sch04]. The *ir* subscripts mean *imperfect information* ($i$) and *imperfect recall* ($r$), in opposition to perfect information ($I$) and perfect recall ($R$). $ATL$ can be seen as $ATL_{IR}$. $ATL_{ir}$ is used to reason about the strategies of agents with a partial view of the system as the strategies the logic considers are the ones that the agents can actually play [JvdH04].

The syntax of $ATL_{ir}$ is the same as for $ATL$. The models over which $ATL_{ir}$ formulas are interpreted are *imperfect information concurrent game structures* (iCGS) $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V \rangle$ where $\langle Ag, Q, Q_0, Act, e, \delta, V \rangle$ is a CGS and $\sim: Ag \rightarrow 2^{(Q \times Q)}$ defines, for each agent $ag$, an equivalence relation over states. We write $\sim_{ag}$ for $\sim (ag)$. We assume that each agent can choose his action based on his observations of the current state only, that is, $\forall q, q' \in Q$ s.t. $q \sim_{ag} q', e_{ag}(q) = e_{ag}(q')$.

A *memoryless strategy* for agent $ag$ is a strategy $f_{ag}$ such that

$$\forall \pi, \pi' \in Q^+ \text{ s.t. } \pi(|\pi| - 1) = \pi'(|\pi'| - 1), f_{ag}(\pi) = f_{ag}(\pi'),$$

that is, the strategy $f_{ag}$ is memoryless if it chooses the action to play based on the current state only. Such a strategy can be viewed as a function $f_{ag} : Q \to Act$ depending only on the current state. A *memoryless uniform strategy*, also called uniform strategy in the sequel, is a memoryless strategy $f_{ag}$ such that

$$\forall q, q' \in Q \text{ s.t. } q \sim_{ag} q', f_{ag}(q) = f_{ag}(q'),$$

that is, the agent chooses the action to play based on his observations of the current state. We write $F^u_{ag}$ for the set of uniform memoryless strategies of agent $ag$, and $F^u_\Gamma$ for the set of uniform memoryless strategies for a group of agents $\Gamma$.

The semantics of $ATL_{ir}$ formulas is the same as for $ATL$, except for the coalition operator that is restricted to the existence of a uniform memoryless strategy for all indistinguishable states:

$$S, q \vDash_{ATL_{ir}} \langle\!\langle \Gamma \rangle\!\rangle\ \psi \Leftrightarrow \begin{cases} \text{there exists a uniform strategy } f_\Gamma \text{ for } \Gamma \text{ s.t.} \\ \text{for all } ag \in \Gamma, \text{for all } q' \in Q \text{ s.t. } q \sim_{ag} q', \\ \text{for all } \pi \in out(f_\Gamma, q'), S, \pi \vDash_{ATL_{ir}} \psi, \end{cases}$$

where the relation $S, \pi \vDash_{ATL_{ir}} \psi$ follows the one for $ATL$.

For instance, the player has no uniform strategy to win the card game because he must distinguish the state $\langle A, K\rangle$ from the state $\langle A, Q\rangle$, in which he must choose different actions to win. So, in this model, the formula $\langle\!\langle player \rangle\!\rangle \mathbf{F}\ win$ is not satisfied by the initial state.

The model checking problem for $ATL_{ir}$ is $\Delta^P_2$-complete, that is, it needs a polynomial number of calls to an NP oracle to solve the problem [JD06, JD08].

### 2.1.5 Propositional μ-calculus

The propositional μ-calculus (abbreviated in the sequel as the μ-calculus) is a logic based on fixpoints [Koz83]. Its syntax is composed of a set of atomic properties $AP$, the special properties $true$ and $false$, a set of variables $Var$, the standard Boolean combinators $\neg$, $\wedge$ and $\vee$, the least and greatest fixpoint operators $\mu$ and $\nu$, and a set of existential and universal modes $\Diamond_i$ and $\Box_i$. More formally, μ-calculus formulas follow the grammar

$$\phi ::= true \mid p \mid v \mid \neg\phi \mid \phi \vee \phi \mid \Diamond_i\ \phi \mid \mu v.\ \phi$$

where $p \in AP$ are atomic propositions and $v \in Var$ are variables. In the sequel, we write $\mathcal{L}_\mu$ for the set of μ-calculus formulas. As usual, other operators can be defined in terms of the ones above:

$$\begin{aligned}\Box_i\ \phi &\equiv\ \neg \Diamond_i\ \neg\phi,\\ \nu v.\ \phi &\equiv\ \neg\mu v.\ \neg\phi(\neg v).\end{aligned}$$

A variable $v$ is *bound* in $\phi$ if it is enclosed in a sub-formula $\mu v.\ \psi$ or $\nu v.\ \psi$; otherwise, it is *free*. We sometimes note $\mu v.\ \psi(v)$, $\nu v.\ \psi(v)$, and $\psi(v)$ to stress the fact that $\psi$ contains free occurrences of variable $v$. We also write $\psi[\chi/v]$—or equivalently $\psi(\chi)$ when $v$ is clear from the context—for the µ-calculus formula $\psi$ where every occurrence of $v$ is replaced by $\chi$. This notation is extended for any sub-formula, that is, $\phi[\phi_2/\phi_1]$ is the formula $\phi$ where every occurrence of the sub-formula $\phi_1$ is replaced by $\phi_2$. A formula $\phi$ is called *closed* if all variables occurring in $\phi$ are bound in $\phi$; otherwise, $\phi$ is *open*. We write $\psi \leq \phi$ to express the fact that $\psi$ is a sub-formula of $\phi$.

Any formula $\mu v.\ \psi$ or $\nu v.\ \psi$, must be *syntactically monotone*, that is, all occurrences of $v$ in $\psi$ must fall under an even number of negations. A formula is in *positive normal form* if negations are only applied to atomic propositions and variables. Any syntactically monotone formula can be transformed into an equivalent syntactically monotone formula in positive normal form.

We say that a formula $\phi$ is *alternation-free* if, when $\phi$ is in positive normal form, there is no occurrence of the $\nu$ (resp. $\mu$) operator on any syntactic path between a $\mu$ (resp. $\nu$) operator and the occurrences of the variable it binds. We call the *alternation-free $\mu$-calculus* the µ-calculus restricted to alternation-free formulas.

µ-calculus models are Kripke structures $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$ where

- $Q$ is a set of states;
- $R_i \subseteq Q \times Q$ are $|\Sigma|$ transition relations; in the sequel, we write $q \rightarrow_i q'$ for $\langle q, q'\rangle \in R_i$;
- $V : Q \rightarrow 2^{AP}$ is a function labeling the states with atomic propositions of $AP$.

µ-calculus formulas are interpreted as sets of states under a given environment. An environment is a function $e : Var \rightarrow 2^Q$ associating a set of states to all free variables of the formula. The set of environments is noted $\mathcal{E}$. We write $e[Q'/v]$ for $Q' \subseteq Q$ and $v \in Var$ as the function $e'$ such that $e'(v) = Q'$ and $e'$ agrees with $e$ for all other variables. The semantics of formulas is given by the function $[\![\phi]\!]^S e$. It takes a formula $\phi$ and an environment $e$ defined at least for the free variables of $\phi$, and returns the corresponding set of states. $e$ can be omitted when $\phi$ is closed. This function is defined as:

$$\begin{aligned}
[\![true]\!]^S e &= Q,\\
[\![p]\!]^S e &= \{q \in Q \mid p \in V(q)\},\\
[\![v]\!]^S e &= e(v),\\
[\![\neg\phi]\!]^S e &= Q \backslash [\![\phi]\!]^S e,\\
[\![\phi \vee \psi]\!]^S e &= [\![\phi]\!]^S e \cup [\![\psi]\!]^S e,\\
[\![\Diamond_i\ \phi]\!]^S e &= \{q \in Q \mid \exists q' \in Q \text{ s.t. } q \rightarrow_i q' \wedge q' \in [\![\phi]\!]^S e\},\\
[\![\mu v.\ \phi]\!]^S e &= \bigcap\{Q' \subseteq Q \mid [\![\phi]\!]^S e[Q'/v] \subseteq Q'\}.
\end{aligned}$$

## 2.2 Symbolic model checking

This section presents *symbolic model checking*, a set of techniques to solve the model-checking problem of the logics presented above. The model-checking problem for a logic $\mathcal{L}$ is, given a model $S$, a state $q$ of this model and a formula $\phi$ of $\mathcal{L}$, determining whether $S, q \vDash_{\mathcal{L}} \phi$ or not.

These techniques are called *symbolic* because they use binary decision diagrams to compactly represent sets of states and transition relations, instead of representing them in an *explicit* way by enumerating all their elements.

This section describes what are binary decision diagrams (BDDs), how to encode models with BDDs, and how to solve the model-checking problem of the logics with BDDs.

### 2.2.1 Binary decision diagrams

Binary decision diagrams are canonical representations for Boolean formulas [Bry86, MT98]. A Boolean formula $\phi$ on propositional variables $v_1$, ..., $v_n$ can be seen as a function $f_\phi(v_1, ..., v_n)$ such that $f_\phi(b_1, ..., b_n)$ returns $true$ if and only if replacing $v_i$ by $b_i \in \{true, false\}$ in $\phi$ yields a $true$ formula.

Such a Boolean formula $\phi$ can be represented by a binary decision tree where each non-terminal node is linked to a propositional variable and has a left and a right child, and each terminal node is labelled by $true$ or $false$ (or equivalently 1 or 0, respectively). Any path in such a tree meets each propositional variable exactly once, always in the same order, and the leaf node reached through a particular path is labelled by

$f_\phi(b_1, ..., b_n)$, where $b_i = true$ if the path goes through the left child of the node labelled by $v_i$, and $b_i = false$ otherwise. For instance, Figure 2.3 represents the binary decision tree of the formula $v_1 \vee (v_2 \wedge \neg v_3)$.

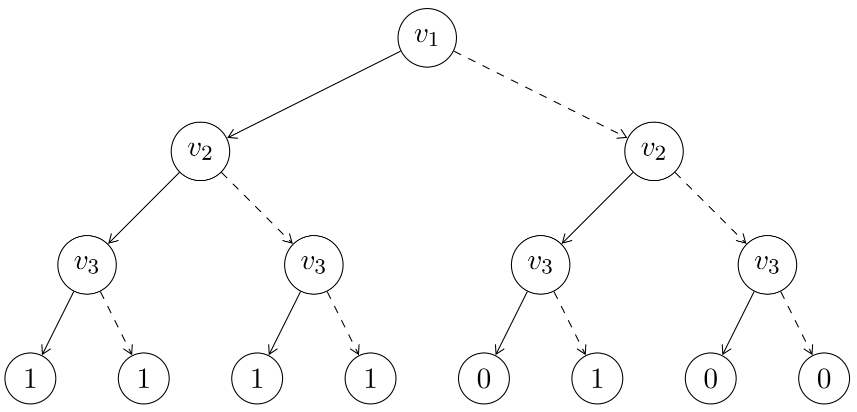


Figure 2.3: The binary decision tree for the formula $v_1 \vee (v_2 \wedge \neg v_3)$. Left ($true$) child is under the plain edge, right ($false$) child under the dashed one.

Binary decision trees contain a lot of redundancy. For instance, the tree of Figure 2.3 contains several identical subtrees, such as the two subtrees rooted at $v_3$ in the left part. Ordered binary decision diagrams—simply called binary decision diagrams in the rest of this thesis—are directed acyclic graphs with non-terminal binary nodes labelled by propositional variables and terminal ones by Boolean values. Furthermore, on any path, all propositional variables are encountered at most once. Also, the propositional variables are totally ordered, such that if $v_i < v_j$, then on any path of the diagram, if two nodes labelled by $v_i$ and $v_j$ are encountered, the first one appears before the second one. Finally, these graphs contain no isomorphic subtrees and no redundant nodes, that is, no nodes with the same subtree on left and right. The binary decision diagram of $v_1 \vee (v_2 \wedge \neg v_3)$ is given in Figure 2.4. BDDs can be viewed as binary decision trees on which we applied the following reduction rules until no rule applies anymore:

- merge identical subtrees;
- remove redundant tests: if a non-terminal node has the same *true* and *false* successors, it can be eliminated.

BDDs are canonical representations of Boolean formulas in the sense that two formulas have isomorphic BDDs if and only if they are equivalent, that is, they have the same truth values for the same assignments

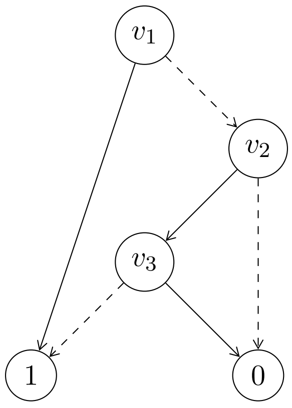


Figure 2.4: The binary decision diagram for the formula $v_1 \vee (v_2 \wedge \neg v_3)$, with variables ordered by $v_1 < v_2 < v_3$. *true* child is under the plain edge, *false* child under the dashed one.

to their variables. Testing the equivalence of two formulas, given their corresponding BDDs, is thus checking that their roots are identical. Furthermore, many standard Boolean operations, such as complementation, conjunction, disjunction and existential quantification, can be performed on BDDs in time polynomial to the number of nodes of the operands.

Nevertheless, BDDs suffer from some disadvantages. The first one is that the number of nodes of the BDD corresponding to a formula $\phi$ is exponential in the number of variables of $\phi$ in the worst case. Furthermore, the order of the variables of a BDD can have a huge impact on its number of nodes, and thus on the time needed to perform Boolean operations on it. Furthermore, finding a good order is difficult, and finding the optimal one is an NP-complete problem. Much effort has been made to develop heuristics that find good orders (see [MT98] for a summary of these techniques).

### 2.2.2 Encoding models with binary decision diagrams

To encode models with binary decision diagrams, we need to be able to encode sets of elements—such as sets of states, sets of actions—and relations—such as transition relations, equivalence relations.

Elements from a set $Z$ are encoded by defining a bijection from Boolean vectors of size $m = \lceil log_2(|Z|) \rceil$ to elements of $Z$. Each element of $Z$ is encoded with $m$ Boolean values. A subset $Z'$ of $Z$ is encoded as the Boolean formula that is true for the Boolean assignments corresponding to states of $Z'$ and false for the others. For instance, let us consider the Kripke structure depicted in Figure 2.5. This structure has four states $Q = \{q_0, q_1, q_2, q_3\}$. We can define the bijection $f : \{0,1\} \to Q$ as $f(v_1, v_2) = q_{(v_1+2*v_2)}$. The subset $Q' = \{q_1, q_3\}$ is thus represented by the BDD corresponding to the Boolean formula

$$(v_1 \wedge \neg v_2) \vee (v_1 \wedge v_2) \equiv v_1.$$

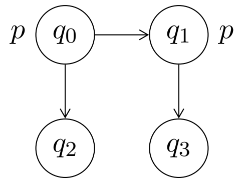


Figure 2.5: A Kripke structure. States are labelled with the atomic proposition $p$.

Relations are represented as sets of tuples using the Boolean variables corresponding to their elements. If a relation is defined over the same set of elements, copies of the variables are used. For instance, to represent the transition relation of the structure of Figure 2.5, we need one copy of the Boolean variables for elements of $Q$, $v'_1$ and $v'_2$. Then, the transition relation is encoded as the BDD corresponding to the formula

$$(\neg v_1 \wedge \neg v_2 \wedge v'_1 \wedge \neg v'_2) \vee (v_1 \wedge \neg v_2 \wedge v'_1 \wedge v'_2) \vee (\neg v_1 \wedge \neg v_2 \wedge \neg v'_1 \wedge v'_2).$$

Finally, the labeling is defined as $|AP|$ subsets of $Q$ encoding, for each atomic proposition of $AP$, the subset of states in which the proposition is true [CGP99]. For instance, the BDD related to the atomic proposition $p$ in Figure 2.5 corresponds to the Boolean formula

$$(\neg v_1 \wedge \neg v_2) \vee (v_1 \wedge \neg v_2) \equiv \neg v_2.$$

### 2.2.3 Model checking rich logics with binary decision diagrams

This section describes how we can solve the model-checking problem of the µ-calculus, $CTL$, $FCTL$ and $ATL$, showing the basic techniques on which many algorithms presented in this thesis are based.

#### Computing fixpoints

Before presenting the model-checking algorithms, let us describe how to compute fixpoints of functions $\tau : 2^Z \to 2^Z$, taking one subset of elements of $Z$ as input and returning another subset. First, if $Z$ is finite, then $\tau$

has least and greatest fixpoints if and only if $\tau$ is monotonic [CGP99]. All functions for which we compute fixpoints in this thesis are monotonic.

Given a function $\tau$ over sets of elements of $Z$ and a subset $Z' \subseteq Z$, $Z'$ is a fixpoint of $\tau$ if $\tau(Z') = Z'$. Furthermore, let $\tau^i(Z')$, $i \geq 0$, be defined as $\tau^0(Z') = Z'$ and $\tau^{i+1}(Z') = \tau(\tau^i(Z'))$ for $i \geq 0$. Given that $\tau$ is a monotonic function, its least fixpoint is given by $\tau^\infty(\varnothing) = \tau^j(\varnothing)$ for some $j \geq 0$, and its greatest fixpoint by $\tau^\infty(Z) = \tau^j(Z)$ for some $j \geq 0$. This gives us a direct way to compute least and greatest fixpoints: by iterating $\tau$ applications from one extreme ($\varnothing$ and $Z$, respectively) and spotting when a fixpoint is found. In the sequel, we use *Lfp* and *Gfp* for the two algorithms that compute the least and greatest fixpoints (resp.) of the function they receive as argument (see, for instance, [CGP99] for more information).

As soon as subsets of $Z$ can be represented with BDDs and $\tau$ can be computed on these BDDs, $\mathit{Lfp}(\tau)$ and $\mathit{Gfp}(\tau)$ are easily implemented with BDDs as they only need the $\varnothing$ BDD (a single 0 node), the $Z$ BDD (a single 1 node), and the possibility to check the equivalence of two BDDs (available in any BDD package, at low cost). These two algorithms are thus well suited to compute least and greatest fixpoints with binary decision diagrams. In the sequel, we use $\mathit{Lfp}(\tau)$ and $\mu Z'.\tau(Z')$ interchangeably, as well as $\mathit{Gfp}(\tau)$ and $\nu Z'.\tau(Z')$.

### µ-calculus model checking

This section describes an algorithm to solve the model-checking problem of the µ-calculus [CGP99]. First, let

$$Pre(S, i, Q') = \{q \in Q \mid \exists q' \in Q' \text{ s.t. } q \rightarrow_i q'\}. \tag{2.1}$$

It takes a Kripke structure $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$, a transition relation identifier $i \in \Sigma$ from $S$ and a subset $Q'$ of $Q$ as arguments, and returns the set of states of $Q$ that have a successor in $Q'$ through the relation $R_i$.

We can define the model-checking algorithm $eval_\mu(S, \phi, e)$ that returns the set of states $S$ satisfying $\phi$ in environment $e$, that is, the set $[\![\phi]\!]^S e$. This algorithm is described as a function defined for all possible operators that $\phi$ can use, and the underlying semantics is that the algorithm evaluates the function depending on the actual operator of $\phi$. $eval_\mu(S, \phi, e)$ is defined as

$$\begin{aligned}
eval_\mu(S, true, e) &= Q,\\
eval_\mu(S, p, e) &= \{q \in Q \mid p \in V(q)\},\\
eval_\mu(S, v, e) &= e(v),\\
eval_\mu(S, \neg\phi, e) &= Q \backslash eval_\mu(S, \phi, e),
\end{aligned}$$

$$
\begin{aligned}
eval_\mu(S, \phi_1 \vee \phi_2, e) &= eval_\mu(S, \phi_1, e) \cup eval_\mu(S, \phi_2, e),\\
eval_\mu(S, \Diamond_i\ \phi, e) &= Pre(S, i, eval_\mu(S, \phi, e)),\\
eval_\mu(S, \mu v.\ \phi, e) &= Lfp(\lambda Q'.eval(S, \phi, e[Q'/v])).
\end{aligned}
$$

## $CTL$ model checking

Like the algorithm for the μ-calculus, the algorithm for solving the model-checking problem for $CTL$ is based on a simple $Pre$ function, standard set manipulations, and least and greatest fixpoint computations [BCM+90]. Let

$$Pre(S, Q') = \{q \in Q \mid \exists q' \in Q' \text{ s.t. } q \to q'\}.$$

It takes a Kripke structure $S = \langle Q, Q_0, R, V\rangle$ and a subset of states $Q' \subseteq Q$ as arguments, and returns the set of states that have a successor in $Q'$ through the transition relation of $S$.

With $Pre$, we can define the function

$$Reach(S, Q_1, Q_2) = \mu Q'.\ Q_2 \cup (Q_1 \cap Pre(S, Q'))$$

taking a Kripke structure $S = \langle Q, Q_0, R, V\rangle$ and two subsets of states $Q_1, Q_2 \subseteq Q$ as arguments, and returning the states of $S$ from which there exists a finite path to a state of $Q_2$ through states of $Q_1$.

Based on the two functions above, we can define the model-checking algorithm $eval_{CTL}(S, \phi)$ that returns the set of states $S$ satisfying $\phi$, that is, the set $\{q \in Q \mid S, q \models_{CTL} \phi\}$. With this algorithm, it is easy to solve the model-checking problem for $CTL$: $S, q \models_{CTL} \phi$ if and only if $q \in eval_{CTL}(S, \phi)$. This algorithm is defined as

$$
\begin{aligned}
eval_{CTL}(S, true) &= Q,\\
eval_{CTL}(S, p) &= \{q \in Q \mid p \in V(q)\},\\
eval_{CTL}(S, \neg\phi) &= Q \backslash Q',\\
eval_{CTL}(S, \phi_1 \vee \phi_2) &= Q_1 \cup Q_2,\\
eval_{CTL}(S, \mathbf{EX}\ \phi) &= Pre(S, Q'),\\
eval_{CTL}(S, \mathbf{E}[\phi_1\ \mathbf{U}\ \phi_2]) &= Reach(S, Q_1, Q_2),\\
eval_{CTL}(S, \mathbf{E}[\phi_1\ \mathbf{W}\ \phi_2]) &= \nu Q'.\ Q_2 \cup (Q_1 \cap Pre(S, Q')),
\end{aligned}
$$

where

$$
\begin{aligned}
Q' &= eval_{CTL}(S, \phi),\\
Q_1 &= eval_{CTL}(S, \phi_1),\\
Q_2 &= eval_{CTL}(S, \phi_2).
\end{aligned}
$$

### Fair $CTL$ model checking

The model-checking algorithm for $FCTL$ is a variant of the one for $CTL$. More precisely, it is based on the same $Pre$ function, lifted to take Kripke structures $S = \langle Q, Q_0, R, V, FC \rangle$ with fairness constraints into account. Let $Reach$ be the same function as before, also lifted for the same structures. Then we can define the function $Fair(S)$ as

$$Fair(S) = \nu Q'.\ \bigcap_{fc \in FC} Pre\big(Reach\big(S, Q, (Q' \cap fc)\big)\big).$$

This function takes a Kripke structure as argument and returns the states of this structure from which there exists a fair path, that is, a path such that all fairness constraints of $FC$ are met infinitely often.

Based on the $Pre$, $Reach$ and $Fair$ functions, we can define the model-checking algorithm $eval_{FCTL}(S, \phi)$ returning the set of states of $S$ satisfying $\phi$:

$$\begin{aligned} eval_{FCTL}(S, \mathbf{EX}\ \phi) &= Pre(S, Q_F), \\ eval_{FCTL}(S, \mathbf{E}[\phi_1\ \mathbf{U}\ \phi_2]) &= Reach(S, Q_1, Q_{2,F}), \end{aligned}$$

and

$$\begin{aligned} &eval_{FCTL}(S, \mathbf{E}[\phi_1\ \mathbf{W}\ \phi_2]) = \\ &\nu Q'.\ Q_{2,F} \cup \left( Q_1 \cap \bigcap_{fc \in FC} Pre\big(Reach\big(S, Q_1, Q_{2,F} \cup (Q' \cap fc)\big)\big) \right), \end{aligned}$$

where

$$\begin{aligned} Q_F &= eval_{FCTL}(S, \phi) \cap Fair(S), \\ Q_1 &= eval_{FCTL}(S, \phi_1), \\ Q_2 &= eval_{FCTL}(S, \phi_2), \\ Q_{2,F} &= Q_2 \cap Fair(S). \end{aligned}$$

Intuitively, $eval_{FCTL}(S, \mathbf{EX}\ \phi)$ returns the states that have a fair successor satisfying $\phi$. Similarly, $eval_{FCTL}(S, \mathbf{E}[\phi_1\ \mathbf{U}\ \phi_2])$ returns the states that can reach a fair state satisfying $\phi_2$ through states satisfying $\phi_1$. Finally, $eval_{FCTL}(S, \mathbf{E}[\phi_1\ \mathbf{W}\ \phi_2])$ returns the states that either satisfy $\phi_2$ and are fair (states of $Q_{2,F}$), or that can repeatedly reach any fairness constraint $fc$ or another state of $Q_{2,F}$ through states satisfying $\phi_1$. Some state $q$ thus belongs to $eval_{FCTL}(S, \mathbf{E}[\phi_1\ \mathbf{W}\ \phi_2])$ iff either there exists, from $q$, a finite path through $Q_1$ to a fair state satisfying $\phi_2$, or there exists, from $q$, an infinite path through $Q_1$ that meets every fairness constraint infinitely often.

### $ATL$ model checking

As $ATL$ is an extension of $CTL$, the model-checking algorithm for $ATL$ extends the one for $CTL$ [AHK02]. Let

$$Pre_{\llbracket\Gamma\rrbracket}(S,Q') = \left\{ q \in Q \;\middle|\; \begin{array}{l} \forall a_\Gamma \in E_\Gamma(q), \exists a \in E_{Ag}(q) \\ \text{s.t. } a_\Gamma \sqsubseteq a \wedge \delta(q,a) \in Q' \end{array} \right\}. \tag{2.2}$$

It takes a CGS $S = \langle Ag, Q, Q_0, Act, e, \delta, V\rangle$ and a set $Q' \subseteq Q$ as arguments and returns the set of states $q$ in which, for each action of $\Gamma$ enabled in $q$, there exists a choice for the other agents such that the reached state is in $Q'$. In other words, $Pre_{\llbracket\Gamma\rrbracket}(S,Q')$ computes the set of states in which $\Gamma$ cannot avoid to reach $Q'$ in one step.

With this $Pre$ function, we can define

$$Reach_{\llbracket\Gamma\rrbracket}(S,Q_1,Q_2) = \mu Q'.\ Q_2 \cup (Q_1 \cap Pre_{\llbracket\Gamma\rrbracket}(S,Q')) \tag{2.3}$$

that returns the set of states of $S$ from which $\Gamma$ cannot avoid to reach a state of $Q_2$ through states of $Q_1$.

Based on these two functions, we can define a model-checking algorithm $eval_{ATL}(S,\phi)$ returning the set of states of the CGS $S$ satisfying the $ATL$ formula $\phi$:

$$\begin{aligned}
eval_{ATL}(S, true) &= Q,\\
eval_{ATL}(S, p) &= \{q \in Q \mid p \in V(q)\},\\
eval_{ATL}(S, \neg\phi) &= Q \backslash Q',\\
eval_{ATL}(S, \phi_1 \vee \phi_2) &= Q_1 \cup Q_2,\\
eval_{ATL}(S, \llbracket\Gamma\rrbracket \mathbf{X}\ \phi) &= Pre_{\llbracket\Gamma\rrbracket}(S,Q'),\\
eval_{ATL}(S, \llbracket\Gamma\rrbracket[\phi_1\ \mathbf{U}\ \phi_2]) &= Reach_{\llbracket\Gamma\rrbracket}(S,Q_1,Q_2),\\
eval_{ATL}(S, \llbracket\Gamma\rrbracket[\phi_1\ \mathbf{W}\ \phi_2]) &= \nu Q'.\ Q_2 \cup (Q_1 \cap Pre_{\llbracket\Gamma\rrbracket}(S,Q')),
\end{aligned}$$

where

$$\begin{aligned}
Q' &= eval_{ATL}(S,\phi),\\
Q_1 &= eval_{ATL}(S,\phi_1),\\
Q_2 &= eval_{ATL}(S,\phi_2).
\end{aligned}$$

This algorithm is expressed in terms of the $\llbracket\rrbracket$ operator. To compute the states satisfying a $\langle\!\langle\rangle\!\rangle$ operator, we can simply rely on the equivalence $\llbracket\Gamma\rrbracket\ \psi \equiv \neg\langle\!\langle\Gamma\rangle\!\rangle\ \neg\psi$.

The time complexity of this algorithm is in $\mathcal{O}(m * l)$ where $m$ is the number of transitions of the CGS $S = \langle Ag, Q, Q_0, Act, e, \delta, V\rangle$, that is, $m = |\{\langle q, a, q'\rangle \in Q \times Act^{Ag} \times Q \mid \delta(q,a) = q'\}|$, and $l$ is the number of sub-formulas of the formula $\phi$ [AHK02].

## 2.3 Tools

This section presents the tools used in this thesis to implement and test the presented approaches: NuSMV, a state-of-the-art symbolic model checker, and PyNuSMV, a Python library based on NuSMV.

### 2.3.1 NuSMV

NuSMV is a state-of-the-art symbolic model checker [CCG+02]. It supports the modeling of synchronous finite-state reactive systems through a high-level modeling language. The tool can perform model checking for several kinds of temporal logics including $CTL$, $FCTL$ and $LTL$. It also includes the functionalities to extract trace-based counter-examples, to replay these execution traces and to simulate the model step by step.

The model-checking algorithms it implements are based on binary decision diagrams as well as on SAT solvers. More precisely, NuSMV supports BDD-based model checking for $CTL$, Fair $CTL$ and $LTL$, and SAT-based model checking for $LTL$ [BCCZ99].

The modeling language of NuSMV is based on the concepts of typed state (`VAR`) and input (`IVAR`) variables that take a value among a finite range. The available types are Boolean, finite-range integers, enumerations, bit vectors, and arrays. Furthermore, the evolution of the system, as well as the initial states and fairness constraints, are described with expressions over the declared variables, and their `next` counterparts when defining transition relations. Finally, these variables and expressions are gathered in modules that can be instantiated.

For instance, the Kripke structure of the card game presented in Figure 2.1 (page 11) can be modeled with the NuSMV modeling language as in Figure 2.6. The NuSMV model is composed of three modules `Player`, `Dealer` and `main`—the first two being instantiated in the third one—, several state and input variables—`step`, `pcard`, `dcard`, `player.action`, `dealer.to_player`, `dealer.to_dealer`—, and `INIT` and `TRANS` clauses to define initial states and transition relations, respectively. The `DEFINE` clause defines a macro for the given expression instead of declaring a new variable.

### 2.3.2 PyNuSMV

NuSMV is a well-optimized tool, with several functionalities. Nevertheless, its hundreds of thousands lines of C code makes it difficult to implement new logics or model-checking algorithms and approaches that are not supported yet. On the other hand, this is the primary goal of PyNuSMV [BP13].

```
MODULE Player(step)
  IVAR action : {none, keep, swap};
  TRANS action in ( step = 1 ? {keep, swap} : {none} )
MODULE Dealer(step)
  IVAR to_player : {none, Ac, K, Q};
  IVAR to_dealer : {none, Ac, K, Q};
  TRANS step = 0 -> (to_player != to_dealer &
                     to_player != none &
                     to_dealer != none)
  TRANS step != 0 -> (to_player = none &
                      to_dealer = none)
MODULE main
  VAR step   : 0..2;
      pcard  : {none, Ac, K, Q};
      dcard  : {none, Ac, K, Q};
      dealer : Dealer(step);
      player : Player(step);
  INIT step = 0 & pcard = none & dcard = none

  TRANS next(step) = (step < 2 ? (step + 1) : 2)

  TRANS step = 0 -> next(pcard) = dealer.to_player
  TRANS step = 1 -> case player.action = keep :
                           next(pcard) = pcard;
                         TRUE :
                           next(pcard) != pcard &
                           next(pcard) != dcard &
                           next(pcard) != none;
                    esac
  TRANS step = 2 -> next(pcard) = pcard

  TRANS step = 0  -> next(dcard) = dealer.to_dealer
  TRANS step != 0 -> next(dcard) = dcard

  DEFINE
  win := step = 2 & ( (pcard = Ac & dcard = K) |
                      (pcard = K & dcard = Q)  |
                      (pcard = Q & dcard = Ac) );
```

Figure 2.6: A NuSMV model of the structure of Figure 2.1.

PyNuSMV is a Python framework for prototyping and experimenting with BDD-based model-checking algorithms based on NuSMV. It gives access to NuSMV's functionalities, such as source model parsing and BDD manipulation, while hiding NuSMV's implementation details by providing wrappers to NuSMV functions and data structures. In particular, NuSMV models can be read, parsed, compiled and constructed, giving full access to SMV's rich modeling language and vast collection of existing models.

PyNuSMV uses SWIG [Bea96], a wrapper generator for C code, to expose the API of NuSMV in Python. On top of this wrapper, PyNuSMV provides a library of classes and modules reflecting NuSMV's main data structures (BDDs, expressions) at the Python level. Thanks to these classes and modules, it is easy to use NuSMV functionalities in Python, without struggling with implementation details such as memory management.

The main functionalities of PyNuSMV give access to:

- BDDs, states and inputs (i.e. actions) of the model and standard operations on BDDs provided as built-in operators: `&` (conjunction), `|` (disjunction), `~` (negation);
- the model itself, encoded with BDDs, and basic functionalities such as computing the pre- or post-image of a set of states through the transition relation of the model;
- $CTL$ formulas expressing properties of the model;
- functions acting on the global environment of NuSMV: initializing and finalizing NuSMV, reading the model and encoding it into BDDs;
- the parser of NuSMV to get, for instance, the AST of a simple expression;
- the $CTL$ and $FCTL$ model-checking algorithms implemented in NuSMV.

The abstraction level of the Python language and the performances of NuSMV make PyNuSMV the right tool for prototyping and experimenting with BDD-based model-checking algorithms. For instance, the following Python code loads the model of Figure 2.6 (stored in the `cardgame.smv` file), builds the corresponding BDD-based finite-state machine, and prints the number of reachable states and whether the model satisfies the formula **EF** *win*.

```
import pynusmv as pn
with pn.init.init_nusmv():
    pn.glob.load("cardgame.smv")
    pn.glob.compute_model()
    fsm = pn.glob.prop_database().master.bddFsm
    print(fsm.count_states(fsm.reachable_states))
    formula = pn.prop.ef(pn.prop.atom("win"))
    print(pn.mc.check_ctl_spec(fsm, formula))
```

# Part I

# Symbolic model checking of uniform strategies under fairness constraints

# *Chapter 3*

# *Logics and algorithms for model checking strategies*

Since the introduction of $ATL$ at the end of the 90's, a lot of research has been conducted on logics reasoning about the strategies of the agents of a system. This chapter presents work in the domain of model-checking logics that consider the uniform strategies of these agents. It starts by presenting the main logics that have been proposed since the introduction of $ATL$. Then it discusses their extensions with knowledge operators and uniform strategies, and with fairness constraints. Finally, it presents the existing approaches for solving the model-checking problem of logics dealing with memoryless uniform strategies.

This summary shows that, while a lot of logics for strategic reasoning exist, only few can deal with uniform strategies and fairness constraints together. Furthermore, none of them focus on the particular problem of reasoning about uniform memoryless strategies under unconditional fairness constraints.

## 3.1 Reasoning about strategies

The introduction of $ATL$ led to a large body of research and literature. A lot of research was conducted to correctly understand the logic and the associated problems such as the satisfiability and the model-checking problems (see for instance [LMO08] and [BJ10]). Furthermore, many other logics related to $ATL$ appeared since the seminal work. This section presents the main ones.

### 3.1.1 Alternating µ-calculus

Alur et al. proposed the *Alternating µ-calculus* (*AMC*), in addition to the Alternating-time Temporal Logic [AHK98, AHK02]. This logic extends the Propositional µ-calculus presented in Section 2.1.5 by replacing the possibility operators $\Diamond_i \phi$ with strategic operators $\langle\langle\Gamma\rangle\rangle \mathbf{X}\ \phi$, where $\Gamma$ is a subset of agents. With this logic, it is possible to express properties such as *agent ag has a strategy to ensure that, on all paths, the proposition p holds in all the even positions*, written $\nu Z.\ p \land \langle\langle ag\rangle\rangle \mathbf{X}\ \langle\langle ag\rangle\rangle \mathbf{X}\ Z$.

*AMC* subsumes $ATL^*$, that is, there exists a translation for $ATL^*$ formulas into *AMC* such that a given $ATL^*$ formula is satisfied by a state of a given model if and only if the corresponding translated *AMC* formula is satisfied by the same state of the given model. Furthermore, the alternation-free fragment of *AMC* (written *af-AMC* in the sequel) subsumes *ATL*.

In terms of complexity, model checking the alternation-free *AMC* is not more difficult than model checking *ATL*. Both problems can be solved in time $\mathcal{O}(m * l)$, where $m$ is the number of transitions of the model and $l$ is the number of sub-formulas of the formula. On the other hand, the model-checking problem for the full *AMC* can be solved in time $\mathcal{O}((m * l)^{d+1})$, where $d$ is the alternation depth of the checked formula.

A model-checking algorithm for *AMC* can be obtained by applying a small modification to the algorithm for the µ-calculus presented in Section 2.2.3. Instead of using the *Pre* function of Equation 2.1, we can use the complement of the $Pre_{[\![\Gamma]\!]}$ function of Equation 2.2.

### 3.1.2 Strategy logic

Another logic reasoning about the strategies of the agents of a system is the *Strategy Logic* (*SL*). It has been first proposed by Chatterjee et al. in [CHP10] for the case of two-player games, and has been later extended to multi-agent concurrent systems by Mogavero et al. [MMV10].

While *ATL* and its extensions such as $ATL^*$ and *AMC* quantify over the strategies implicitly through the strategic operators, the Strategy Logic includes an explicit quantification over strategies. This explicit quantification allows to express properties of nonzero-sum games—that is, games in which the objectives of a coalition and its complement are not complement objectives—such as strategy domination and Nash equilibria. For instance, *SL* allows us to express the fact that *there exists a strategy for the dealer such that there exists a strategy for the player to win, and there exists another one for the player to lose*, written—using the notations of Mogavero et al.—as

$$\langle\!\langle x \rangle\!\rangle\ (dealer, x)(\langle\!\langle y \rangle\!\rangle\ (player, y)\mathbf{F}\ wins \wedge \langle\!\langle z \rangle\!\rangle\ (player, z)\mathbf{F}\ loses).$$

The $\langle\!\langle x \rangle\!\rangle\ \phi$ operator means that *there exists a strategy $x$ such that $\phi$ is true.* Furthermore, $(ag, x)\phi$ means that *if agent $ag$ plays the strategy $x$, then $\phi$ is true.*

In terms of expressiveness, Strategy Logic subsumes $ATL$ and $ATL^*$: the $ATL^*$ strategic quantifier $\langle\!\langle \Gamma \rangle\!\rangle\ \psi$ can be translated into $SL$ as

$$\langle\!\langle x \rangle\!\rangle\ (\Gamma, x)[\![ y ]\!]\ (Ag \backslash \Gamma, y)\psi.$$

Finally, the model-checking problem for $SL$ is PTIME with regard to the size of the model, but 2EXPTIME w.r.t. the size of the formula [MMV10]. This means that it needs time in $\mathcal{O}(p(m) * 2^{2^{p(l)}})$ to solve the problem, where $p(x)$ is a polynomial depending on $x$, $m$ is the size of the model, and $l$ the number of sub-formulas of the checked formula. On the other hand, the satisfiability problem is undecidable.

### 3.1.3 Alternating-time temporal logic with strategy contexts

$ATL^*_{sc}$ extends $ATL^*$ with *strategy contexts* [BCLM09]. The main concern this logic addresses is the fact that, in $ATL$, when some agents $\Gamma$ choose to play a particular strategy, the other agents do not use the fact that this particular strategy is played to inspect their own strategies. For instance, given a system composed of one server and several clients, it is not possible to express in $ATL^*$ the fact that the server has a strategy such that each client, knowing that this strategy is played, can eventually access the service served by the server.

$ATL^*_{sc}$ replaces the $\langle\!\langle\rangle\!\rangle$ operator with two new operators: $\langle\cdot\Gamma\cdot\rangle\phi$ means that $\Gamma$ have a collective strategy such that $\phi$ is true in the context of this strategy, and $\cdot\rangle\Gamma\langle\cdot\phi$ means that $\phi$ is true if $\Gamma$ forget their strategies and consider the whole system executions as possible. For instance, the property that the server has a strategy such that all the $n$ clients can, individually, access the resource, can be written as

$$\langle\cdot server\cdot\rangle\mathbf{G} \bigwedge_{i\in 1..n} \langle\cdot client_i\cdot\rangle\mathbf{F}\ access.$$

In terms of expressiveness, $ATL^*_{sc}$ subsumes $ATL^*$ because the $\langle\!\langle \Gamma \rangle\!\rangle\ \phi$ operator can be translated into $\cdot\rangle Ag\langle\cdot\langle\cdot\Gamma\cdot\rangle\phi$. Furthermore, the model-checking problem for the full logic is decidable but non-elementary, that is, it does not belong to the class of problems solvable in $2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}$ time. By considering only memoryless strategies, the problem becomes PSPACE-complete (we call this variant $ATL_{sc,0}$ in the sequel).

### 3.1.4 Coalition logic

While the logics presented above subsume $ATL$, are more expressive but have more complex model-checking problems, *Coalition Logic* ($CL$) is simpler and has a more attractive model-checking complexity. $CL$ combines the propositional operators with a single operator $[\Gamma]$ to reason about the effectivity of agents of $\Gamma$, that is, what the agents can enforce in one step [Pau02]. For instance, in $CL$, we can express the fact that *the dealer is effective to bring a state in which the player can win*, written as

$$[dealer][player]win.$$

$CL$ is subsumed by $ATL$ [GJ04]. Indeed, $[\Gamma]\phi$ corresponds to $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi$ and the logic can be viewed as the fragment of $ATL$ in which only the $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}$ operator is used.

### 3.1.5 Expressiveness comparison

Most of the logics presented above are incomparable in terms of expressiveness. Indeed, there exist $AMC$ formulas that cannot be translated into $SL$, and vice versa. For instance, the formula

$$\langle\!\langle x\rangle\!\rangle\ (a_1, x)(\langle\!\langle y_1\rangle\!\rangle\ (a_2, y_1)(\mathbf{G}\ p) \wedge \langle\!\langle y_2\rangle\!\rangle\ (a_2, y_2)(\mathbf{G}\ q)),$$

interpreted over a two-agent concurrent system, means that there exists a strategy $x$ for agent $a_1$ and two strategies $y_1$ and $y_2$ for $a_2$, such that, when $x$ is combined with $y_1$ (resp. $y_2$), $p$ is maintained (resp. $q$). This $SL$ formula cannot be expressed in $AMC$ [CHP10]. Furthermore, the formula

$$\nu Z.\ p \wedge \langle\!\langle\varnothing\rangle\!\rangle\mathbf{X}\ \langle\!\langle\varnothing\rangle\!\rangle\mathbf{X}\ Z$$

represents the set of states $s$ such that in all paths from $s$, all even positions satisfy $p$. This formula cannot be translated into $SL$ [CHP10].

$AMC$ is incomparable to $ATL^*_{sc}$, and $SL$ is conjectured to be incomparable with $ATL^*_{sc}$ either [BCLM09].

On the other hand, as $ATL$ subsumes $CL$ and the three logics above subsume $ATL$, they also all subsume $CL$, and every $CL$ formula can be translated into an equivalent formula in the three logics above.

## 3.2 Strategies and knowledge

All the logics described in the previous section agree on the notion of coalition abilities: a coalition of agents can enforce some objective if they have a collective strategy such that all resulting outcomes satisfy the objective. The Alternating µ-calculus and the Coalition Logic embed this in their single-step operators $\langle\!\langle \Gamma \rangle\!\rangle \mathbf{X}$ and $[\Gamma]$, respectively, and the Strategy Logic and $ATL^*_{sc}$ embed this notion in the strategies they consider. All these logics reason about agents with *perfect information*, that is, about agents that know everything about the current state of the system. In this setting, they can use the complete information of the current state to perform their choices, and they have no uncertainty about where they are.

A lot of research has been conducted on the relation between the knowledge of the agents and their abilities. Nevertheless, the proposed logics do not agree with the notion of abilities of the agents, and the relation between what they know and observe, and what they can do. For instance, the *Alternating Temporal Epistemic Logic* ($ATEL$) mixes $ATL$ operators with $ATL$ semantics and knowledge operators, interpreted over states of multi-agent concurrent systems in which the agents have imperfect information about the states [vdHW02, vdHW03]. The semantics of $ATEL$ leads to counterintuitive properties of the systems, in which an agent has a strategy to win, knows he has one, but cannot play it because he lacks some information about the states of the system [JvdH04].

$ATL_{ir}$ is a logic that tries to solve this problem by considering the uniform strategies of the agents, that is, the strategies they can play based on the observations they make [Sch04]. But all the researchers do not agree on the most intuitive semantics, and many have been proposed, such as considering that the agents can communicate during the execution of the strategy, or only when they choose it.

This section first describes and discusses $ATEL$ and its extensions. Then it speaks about $ATL_{ir}$ and the related logics that consider strategies that the agents can effectively play. Finally, it presents extensions of the Alternating µ-calculus and the Strategy Logic with imperfect information, and discusses other related logics.

### 3.2.1 Alternating temporal epistemic logic

The Alternating Temporal Epistemic Logic extends $ATL$ by introducing knowledge operators [vdHW02, vdHW03]. For instance, one can express in $ATEL$ the fact that, in a variant of the card game in which the player never sees the card of the dealer, *the player has a strategy to eventually know the card of the dealer*, written as

$$\langle\!\langle player \rangle\!\rangle \mathbf{F}\ (\mathbf{K}_{player}\ dc = A \vee \mathbf{K}_{player}\ dc = K \vee \mathbf{K}_{player}\ dc = Q),$$

where $dc = X$ is true in states in which the dealer has card $X$. $ATEL$ works with a memoryless notion of knowledge in which the agents only base their knowledge on their observations of the current state.

Model checking $ATEL$ formulas can be made in polynomial time in terms of the size of the model and of the formula. The model-checking algorithm presented in Section 2.2.3 can be easily extended to handle knowledge operators [vdHW02]. As the logic extends $ATL$ with knowledge operators, it subsumes $ATL$. Furthermore, $ATEL$ formulas and models can be re-encoded back into $ATL$ formulas and (more complex) models [GJ04].

Another semantics for $ATEL$ was proposed by van Otterloo and Jonker in [vOJ05]. Their idea is that agents do not have to know a specific strategy for a certain objective to have a capability; they propose instead to quantify over undominated strategies. Intuitively, a strategy is undominated if there is no other strategy that strictly does better for achieving a certain goal. Then, $\langle\!\langle \Gamma \rangle\!\rangle\ \psi$ is true if all undominated strategies are successful for the objective $\psi$. This means that $\Gamma$ can enforce $\psi$ if, by playing any of their best strategies, the objective is fulfilled.

### 3.2.2 Alternating-time temporal logic with imperfect information

To address the problem of $ATEL$ that quantifies over strategies that the agent cannot play because they do not have the necessary knowledge, the notion of uniform strategies have been proposed. In [Sch04], Schobbens considers different semantics for strategies along two axes: imperfect and perfect information—identified with $i$ and $I$ subscripts, respectively—, and imperfect and perfect recall—with $r$ and $R$ subscripts, respectively. This leads to four logics, $ATL_{ir}$, $ATL_{iR}$, $ATL_{Ir}$, and $ATL_{IR}$, the two last ones being equivalent and corresponding to standard $ATL$.

#### $ATL_{ir}$: Memoryless uniform strategies

$ATL_{ir}$ and can be seen as the core logic to reason about uniform memoryless strategies [JD06]. The logic quantifies on uniform memoryless strategies. More precisely, a formula $\langle\!\langle \Gamma \rangle\!\rangle\ \psi$ is true in a state $q$ of a given model if there exists a *memoryless uniform strategy* $f_\Gamma$ for $\Gamma$ such that, for *all states* $q'$ *indistinguishable from* $q$ by some agent of $\Gamma$, all outcomes of $f_\Gamma$ from $q'$ satisfy $\psi$. Its semantics considers that a uniform strategy for a group of agents $\Gamma$ is a tuple of uniform strategies, one for each agent, as opposed to, for instance, a single strategy for whole group, viewing them as a single agent. Furthermore, when looking at strategies in $q$, the states that at least one agent cannot distinguish from $q$ are considered as possible, that is, the group knowledge relation is used.

### $ATL_{iR}$: Memory-full strategies

$ATL_{iR}$ has a semantics similar to $ATL_{ir}$ but considers memory-full strategies, that is, strategies that use the entire history of observations to choose the next action. For instance, let us consider a variant of the repeated card game in which the dealer has to always give the cards in the same order but can choose the order at the very beginning—for example, he can choose to loop through $A$, $K$ and $Q$ when giving the cards, or through $Q$, $K$ and $A$. In this variant, the formula $\langle\langle player\rangle\rangle \mathbf{F}\ wins$ is true in the initial state under $ATL_{iR}$ because the player can play a first game to discover the chosen order and play accordingly the second game. On the other hand, the formula is false under $ATL_{ir}$ because he lacks the memory to do so.

$ATL_{iR}$ considers more powerful agents than $ATL_{ir}$ and the latter is subsumed by the former. Furthermore, while the model-checking problem for $ATL_{ir}$ is $\Delta_2^P$-complete [JD06, JD08], the $ATL_{iR}$ model-checking problem is undecidable [DT11]. The two logics, as well as their relation to $ATL$, have been compared in [JB11].

Bulling et al. proposed a variant of $ATL^*_{iR}$, the extension of $ATL_{iR}$ in which any $LTL$ formula can be used under the scope of a coalition modality. Their variant tackles the problem that $ATL^*_{iR}$ does not use the events that occurred before choosing the strategy when playing it [BJP14a, BJP14b]. An example given in [BJP14a] is the following: the formula $\langle\langle Bob, Charles\rangle\rangle \mathbf{F}\ \langle\langle Alice, Bob\rangle\rangle \mathbf{X}\ married$ means that $Bob$ and $Charles$ have a strategy to ensure that $Alice$ and $Bob$ will be able to get married. In the context of $ATL^*_{iR}$, the agents use their knowledge of the past events to choose their actions. Nevertheless, by the semantics of the logic, $Bob$ forgets everything he learned from the execution of the top-level strategy when executing his strategy for the $\langle\langle Bob, Alice\rangle\rangle \mathbf{X}\ married$ sub-formula.

Bulling et al. propose to modify the semantics of $ATL^*_{iR}$ to use the full history of past events since the initial state when playing a chosen strategy, giving new powers to the agents. Nevertheless, the expressiveness of the new variant is incomparable to the expressiveness of the original $ATL^*_{iR}$ semantics, and the authors did not study the complexity of the model-checking problem.

### Distributed knowledge

Dima et al. proposed another logic, called $ATL_{iR}^{D}$, that considers that a (memory-full) collective strategy for a group of agents is a single function that chooses the joint action of these agents for a given sequence of past observations [DEG10]. These past observations are based on the distributed knowledge of the agents. Furthermore, to distinguish the considered histories, the logic also uses the distributed knowledge.

Intuitively, these choices can be interpreted as the group of agents being under the supervision of a virtual supervisor. This supervisor can ask them about their knowledge of the current history and can point them the action to play based on the knowledge of all agents [DEG10]. Another interpretation of this approach is that it sees a coalition of agents as a single (though composite) agent [KÅJ14].

For instance, in a variant of the simple card game with two players $player_1$ and $player_2$, in which the dealer gives the third card to $player_2$ (who does nothing during the game), the coalition composed of the two players has no strategy to make $player_1$ win the game under $ATL_{ir}$. Indeed, they have no way to pass the knowledge of the third card from $player_2$ to $player_1$. On the other hand, under $ATL_{iR}^{D}$, the agents in the coalition can share their knowledge, and $player_1$ can distinguish the states in which the dealer has the $A$ from the states in which he has the $Q$. Thus $player_1$, thanks to the shared knowledge of $player_2$, can win the game. This difference comes from the shared knowledge, and the perfect recall of past observations has no impact in this particular case.

In opposition to $ATL_{iR}$, the $ATL_{iR}^{D}$ model-checking problem is decidable [DEG10]. Also, any $ATL_{iR}^{D}$ model and formula can be translated into equivalent $ATL_{iR}$ model and formula, respectively [KÅJ14]. This translation transforms the formula into the fragment of $ATL_{iR}$ in which coalition modalities contain only one agent, instead of a group of agents.

Jiang et al. proposed another variant based on the distributed knowledge relation [JZP15]. This logic quantifies on collective strategies that are tuples of memory-full uniform strategies for individual agents. When selecting the strategies to play, the agents share their knowledge of the current state, that is, the considered states are the ones indistinguishable from the current state through the distributed knowledge of the group. Under this semantics, the agents have the memory of the past composed of observations as well as their own actions, instead of the observations alone, as in the standard semantics.

Finally, Huang proposed a third variant based on a bounded semantics and distributed knowledge [Hua15]. In this logic, we can express the fact that *an agent $ag$ can enforce $\phi$ within $k$ steps*—written $\langle\!\langle ag \rangle\!\rangle\, \mathbf{F}^k\, \phi$—, or that *a group of agents* $\Gamma$ *can maintain* $\phi'$ *for at least* $k'$ *steps*—written $\langle\!\langle\Gamma\rangle\!\rangle\ \mathbf{G}^{k'}\ \phi'$. As for $ATL_{iR}^{D}$, the strategies are single functions for the whole group, and their distributed knowledge relation is used to distinguish the whole history of observations and actions, and to choose the actions to play. Thanks to the bounded aspect, the model-checking problem is in PSPACE.

### 3.2.3 Alternating epistemic µ-calculus

The Alternating µ-calculus has been extended to take the knowledge of the agents into account. Bulling and Jamroga proposed $AMC_i$, a variant of $AMC$ where the $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi$ operator is restricted to the observations of the agents [BJ11]. More precisely, $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi$ is true in all states $q$ in which there exists a tuple of uniform memoryless strategies for agents in $\Gamma$ that enforce $\phi$ in the successors of all states indistinguishable from $q$ by some agent of $\Gamma$. That is, the $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi$ operator of $AMC$ is replaced by the $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi$ of $ATL_{ir}$. They limit their logic to the alternation-free fragment of $AMC_i$—called $af\text{-}AMC_i$—and show that, while the alternation-free fragment of $AMC$ subsumes $ATL$, $af\text{-}AMC_i$ does not subsume $ATL_{ir}$.

The particularity of $AMC_i$ is the capability to express the fact that the agents of $\Gamma$ have a strategy that they know how to play *along all the game*. The semantics of $ATL_{ir}$ implies that the agents have a strategy to achieve their objective if they can write it down—or memorize it when choosing it—and then follow it blindly. On the other hand, the fixpoints of $AMC_i$ imply that the agents have a strategy to achieve their objective if they can recompute this winning strategy in every step of the game that they play. They do not have to memorize the strategy when they choose it since they can, at each step, look at what they observe and infer the correct way to play the next step. For instance, the formula $\mu Z.\ p \vee \langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ Z$ is true in a state if the agents of $\Gamma$ know a strategy that they can play and recompute at every step, to eventually make $p$ true. This formula also implies that $\Gamma$ must be certain that the goal $p$ is really approaching [BJ11].

While $ATL_{ir}$ and $af\text{-}AMC_i$ have incomparable expressive power, they seem to share similar model-checking complexities: the $af\text{-}AMC_i$ model-checking problem is NP-hard and in $\Delta_2^P$, while $ATL_{ir}$ model checking is $\Delta_2^P$-complete.

### 3.2.4 Epistemic strategy logic

Huang and van der Meyden proposed an extension of Strategy Logic to reason about knowledge and strategies [HvdM14a]. They started from Epistemic Temporal Logic ($ETLK$), that reasons about time and knowledge, to which they add two new operators: $\exists x.\phi$ is true in a state $q$ if there exists another global state $q'$ such that $\phi$ (that depends on $x$) holds at $q$ when $q'$ is linked to $x$. Furthermore, $e_{ag}(x)$ is true if agent $ag$ cannot distinguish the current state from the state linked to variable $x$.

The semantics of this extension is based on *interpreted systems.* Such a system is composed of a set of agents $Ag$, each agent $ag$ having a set of local states $L_{ag}$ and a set of actions $Act_{ag}$. These actions are chosen according to a protocol $P_{ag} : L_{ag} \to 2^{Act_{ag}}$. The environment of the agents is modelled with a special agent $E$. A global state of an interpreted system is a tuple $g \in \prod_{ag \in Ag} L_{ag} \times L_E$ and the evolution of the local state of the agent $ag$ is defined by the function $t_{ag} : L_{ag} \times L_E \times \prod_{ag \in Ag} Act_{ag} \times Act_E \to L_{ag}$. The evolution of the local states is synchronous, and the evolution of the whole system can be described by a function $t : G \times Act \to G$ where $G \subseteq \prod_{ag \in Ag} L_{ag} \times L_E$ is the set of states reachable from a subset of states $I$, and $Act = \prod_{ag \in Ag} Act_{ag} \times Act_E$ is the set of joint actions. Finally, global states are labelled with atomic propositions of $AP$ through a valuation function $V : AP \to 2^G$ giving, for a proposition $p \in AP$, the set of global states in which $p$ is true.

Huang and van der Meyden propose to use $ETLK$ for reasoning about the strategies of agents in *strategic environments.* These structures are similar to imperfect information concurrent game structures, but every action is enabled in every state of the system. The authors define an interpreted system that encodes the behaviors of the agents that stick to a strategy under a given strategic environment. This definition introduces a new agent $\sigma(ag)$ for each agent $ag \in Ag$ that observes the strategy $ag$ is currently following.

In this context, $ESL$ is the instance of $ETLK$ that works with the interpreted systems corresponding to strategic environments. The language can be restricted to particular types of strategies, such as the uniform strategies of the agents. With the restriction to the set of uniform strategies, we can, for instance, express the fact that *the player does not have a uniform strategy to win the game* as

$$\neg\exists x.D_{ag}(e_{\sigma(ag)}(x) \implies wins).$$

Finally, supposing that the set of considered strategies is a PTIME presented class of strategies, the complexity of the model-checking problem for $ESL$ is EXPSPACE-complete [HvdM14a].

Belardinelli also proposed an extension of Strategy Logic with knowledge operators also called *Epistemic Strategy Logic* (noted $ESL_{Bel}$ in the sequel) [Bel14]. In this logic, we can express facts such as *the player* *knows that he has a strategy to win the game, whatever the dealer does*, written as

$$\mathbf{K}_{player}\ \exists x_{player}.\forall y_{dealer}.\mathbf{F}\ wins,$$

or that *there exist two strategies for the player and the dealer such that the player eventually knows that dealer has the A*, written as

$$\exists x_{player}.\exists y_{dealer}.\mathbf{F}\ \mathbf{K}_{player}\ dealer = A.$$

Similarly to $ATEL$, $ESL$ considers perfect information strategies for the agents, even if they do not have full information about the system. So, adding knowledge operators does not increase the complexity of the model checking problem.

### 3.2.5 Strategy contexts and imperfect information

Laroussinie et al. studied extensions of $ATL_{sc}$ with imperfect information [LMS15]. Instead of assuming that the strategy contexts are based on general strategies, they assume that the strategies are *memory-full* and *uniform*. As plain $ATL_{sc}$ subsumes $ATL$, $ATL_{sc}$ with uniform strategies (called $ATL_{sc,iR}$ in the sequel) subsumes $ATL_{iR}$. The model-checking problem is thus undecidable.

Nevertheless, if we consider the subset of iCGS that are so-called *uniform*, the model-checking problem becomes decidable. An iCGS is *uniform* if all the agents observe the same things, that is, the relations $\sim_{ag}$ are the same for all the agents. In the case of uniform iCGS, the model checking is Tower-complete, that is, solving the problem for a formula with $k$ nested strategic operators is $k$-EXPTIME-complete [LMS15]. Furthermore, if we consider only memoryless uniform strategies in a (not necessarily uniform) iCGS, the logic subsumes $ATL_{ir}$ and the model-checking problem is PSPACE-complete.

### 3.2.6 Epistemic coalition logic

Ågotnes and Alechina proposed an extension of Coalition Logic with epistemic operators [ÅA12]. In this logic, it is possible to express the fact that *the dealer knows he can give the A to the player*, written $\mathbf{K}_{dealer}[dealer]player = A$. The semantics of the strategic operator is not modified and still reasons about what agents can do with perfect information. Furthermore, the authors do not discuss the model-checking problem and its complexity, but show that the satisfiability stay decidable.

### 3.2.7 Other logics

Other logics have been proposed to deal with imperfect information and strategic abilities. This section present the main ones.

#### $ATOL$ and $ATEL\text{-}R^*$

Jamroga and van der Hoek proposed the *Alternating-time Temporal Observational Logic* ($ATOL$) and the *Alternating-time Temporal Epistemic Logic with Recall* ($ATEL\text{-}R^*$) [JvdH04]. The first one is a logic of strategic abilities and knowledge based on the observations of the agents, with no recall of the past events. The second one removes the restriction to memoryless semantics and gives the agents the ability to recall the past events. Instead of giving a particular semantics to the $ATL$ operators like $ATL_{ir}$ or $ATL_{iR}^{D}$, the two logics introduce many operators for reasoning about the different knowledge and strategic modalities of the agents. The syntax of $ATOL$ includes

- the standard propositional operators,
- the memoryless knowledge operator $Obs_{ag}$ to reason about what agent $ag$ observes in the current state,
- the standard memoryless operators $CO_\Gamma$, $EO_\Gamma$ and $DO_\Gamma$ to reason about common observations, group observations and distributed observations of groups of agents $\Gamma$,
- the operators $\langle\langle\Gamma\rangle\rangle^{\bullet}_{Obs(ag)}\mathbf{X}$, $\langle\langle\Gamma\rangle\rangle^{\bullet}_{Obs(ag)}\mathbf{G}$ and $\langle\langle\Gamma\rangle\rangle^{\bullet}_{Obs(ag)}\mathbf{U}$ to reason about the strategies of $\Gamma$ to enforce objectives that agent $ag$ sees,
- the operators $\langle\langle\Gamma\rangle\rangle^{\bullet}_{\Theta(\Gamma')}\mathbf{X}$, $\langle\langle\Gamma\rangle\rangle^{\bullet}_{\Theta(\Gamma')}\mathbf{G}$ and $\langle\langle\Gamma\rangle\rangle^{\bullet}_{\Theta(\Gamma')}\mathbf{U}$, where $\Theta(\Gamma')$ is an element of $\{CO(\Gamma'), DO(\Gamma'), EO(\Gamma')\}$, to reason about the strategies of $\Gamma$ to enforce objectives that the agents of $\Gamma'$ commonly, distributively, or together see.

$ATOL$ formulas are interpreted over states of iCGS and quantify over collective uniform memoryless strategies of groups of agents. For instance, we can express in $ATOL$ that, when the dealer gave the cards, *the player has a strategy to win the game that the dealer sees*, written $\langle\langle player\rangle\rangle^{\bullet}_{Obs(dealer)}\mathbf{X}\ wins$. Nevertheless, in the same states, *the player has no strategy to win that he sees*, written $\neg\langle\langle player\rangle\rangle^{\bullet}_{Obs(player)}\mathbf{X}\ wins$.

$ATOL$ subsumes $ATL_{ir}$: the $ATL_{ir}$ operators $\langle\langle\Gamma\rangle\rangle\ \psi$ can be written as $\langle\langle\Gamma\rangle\rangle^{\bullet}_{EO(\Gamma)}\psi$. Furthermore, the model-checking problem for $ATOL$ is NP-hard and $\Delta_2^P$-easy.

The logic $ATEL\text{-}R^*$ removes the restriction of $ATOL$ to memoryless agents, and the restriction of coalition modalities being directly followed by a path operator. Its syntax shares the strategic operators of $ATOL$ $\langle\!\langle\Gamma\rangle\!\rangle^{\bullet}_{Obs(ag)}$ and $\langle\!\langle\Gamma\rangle\!\rangle^{\bullet}_{\Theta(\Gamma')}$, that can be followed by any $ATEL\text{-}R^*$ formula. Furthermore, the following operators are added:

- $\mathbf{C}_\Gamma$, $\mathbf{D}_\Gamma$ and $\mathbf{E}_\Gamma$ to reason about the common, distributed and group knowledge of the group of agents $\Gamma$, given that they remember their observations of the past events;
- $\langle\!\langle\Gamma\rangle\!\rangle_{\mathcal{K}(\Gamma')}$ for $\mathcal{K}(\Gamma')$ being an element of $\{\mathbf{C}_{\Gamma'}, \mathbf{D}_{\Gamma'}, \mathbf{E}_{\Gamma'}\}$, to reason about the memory-full uniform strategies of $\Gamma$ that $\Gamma'$ know they have, given their observations of the past events;
- $\langle\!\langle\Gamma\rangle\!\rangle_{\Theta(\Gamma')}$, for $\Theta(\Gamma') \in \{CO(\Gamma'), DO(\Gamma'), EO(\Gamma')\}$, to reason about the memory-full uniform strategies of $\Gamma$ that $\Gamma'$ see they have, given their observations of the current state;
- the past $LTL$ operators $\mathbf{X}^{-1}$ and $\mathbf{S}$.

The logic is very expressive and it subsumes $ATL^*$ and $ATEL$, $ATL^*_{ir}$ and $ATL^*_{iR}$. So its model-checking problem is undecidable.

### Constructive strategic logic

Another logic proposed by Jamroga and Ågotnes is the *Constructive Strategic Logic* ($CSL$) [JÅ07]. $CSL$ formulas are composed of the standard propositional operators, the strategic operators $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi$, $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{G}\ \phi$ and $\langle\!\langle\Gamma\rangle\!\rangle[\phi_1\ \mathbf{U}\ \phi_2]$, the standard knowledge operators $\mathbf{C}_\Gamma\ \phi$, $\mathbf{E}_\Gamma\ \phi$ and $\mathbf{D}_\Gamma\ \phi$, and three new *constructive knowledge operators* $\mathbb{C}_\Gamma\ \phi$, $\mathbb{E}_\Gamma\ \phi$ and $\mathbb{D}_\Gamma\ \phi$.

These formulas are interpreted over *sets of states* of an iCGS. Let $img(Q', R) = \{q \mid \exists q' \in Q' \text{ s.t. } qRq'\}$. The relation $S, Q' \models \phi$, meaning that the set of states $Q'$ of the iCGS $S$ satisfies $\phi$, is defined as

$$\begin{array}{lcl}
S, Q' \models p & \Leftrightarrow & \forall q \in Q', p \in V(q), \\
S, Q' \models \neg\phi & \Leftrightarrow & S, Q' \not\models \phi, \\
S, Q' \models \phi_1 \vee \phi_2 & \Leftrightarrow & S, Q' \models \phi_1 \text{ or } S, Q' \models \phi_2, \\
S, Q' \models \langle\!\langle\Gamma\rangle\!\rangle\ \psi & \Leftrightarrow & \left\{ \begin{array}{l} \text{there exists a uniform memoryless strategy } f_\Gamma \\ \text{s.t. } \forall q \in Q', \forall \pi \in out(f_\Gamma, q), S, \pi \models \psi, \end{array} \right. \\
S, Q' \models \mathcal{K}_\Gamma\ \phi & \Leftrightarrow & \forall q \in img(Q', \sim^{\mathcal{K}}_{\Gamma}), S, \{q\} \models \phi, \text{where } \mathcal{K} \in \{\mathbf{C}, \mathbf{E}, \mathbf{D}\}, \\
S, Q' \models \hat{\mathcal{K}}_\Gamma\ \phi & \Leftrightarrow & \left\{ \begin{array}{l} S, img(Q', \sim^{\mathcal{K}}_{\Gamma}) \models \phi, \\ \text{where } \langle\hat{\mathcal{K}}, \mathcal{K}\rangle \in \{\langle\mathbb{C}, \mathbf{C}\rangle, \langle\mathbb{E}, \mathbf{E}\rangle, \langle\mathbb{D}, \mathbf{D}\rangle\} \end{array} \right.
\end{array}$$

The semantics of path operators is the standard $LTL$ one. Combined with the $\langle\!\langle \Gamma \rangle\!\rangle$ operator, the $\mathcal{K}$ and $\hat{\mathcal{K}}$ operators can express that *there exists a winning strategy for all indistinguishable states* ($\mathcal{K}_\Gamma \langle\!\langle \Gamma' \rangle\!\rangle\ \psi$), or that *there exists a strategy that is winning for all indistinguishable states* ($\hat{\mathcal{K}}_\Gamma \langle\!\langle \Gamma' \rangle\!\rangle\ \psi$). The difference is the same as between $ATEL$ and $ATL_{ir}$.

This logic allows to express other modalities such as the ones of $ATOL$. More precisely, there exists a linear translation of $ATOL$ and $ATL_{ir}$ formulas into $CSL$, thus the latter subsumes the former. Finally, the model-checking problem for $CSL$ is $\Delta_2^P$-complete.

*uATEL*

Van Ditmarsch and Knight proposed the logic $uATEL$ to reason about uniform strategies with perfect recall [vDK14]. Its particularity is that it includes different strategic operators: (1) $\langle\!\langle \Gamma \rangle\!\rangle_a\ \psi$ has an active strategy semantics in which all the agents are active, (2) $\langle\!\langle \Gamma \rangle\!\rangle_p\ \psi$ has a passive strategy semantics in which some agents do nothing, and (3) $\langle\!\langle \Gamma \rangle\!\rangle_c\ \psi$ has a communication strategy semantics in which the agents share their knowledge to choose the strategy to play.

The considered individual strategies are perfect-recall uniform strategies in the sense that they use the history of all the observations of the agent, as well as the actions he played. A collective strategy is then a tuple of individual strategies. So, the semantics of the three classes of strategic operators differ only by the set of states they consider as possible when they have to choose their strategies: the active semantics uses the common knowledge of all agents, the passive one uses the common knowledge and strategies of a subset of the agents, the other remaining passive, and the communication semantics uses the distributed knowledge of all the agents.

## 3.3 Strategies and fairness

Reasoning about fairness in multi-agent systems can be useful. For instance, Dastani and Jamroga showed that it is interesting to reason about a fair scheduler in the context of multi-agent programs [DJ10]. These programs define the actions, beliefs and goals of agents with imperfect information about their environment. They are usually executed in an asynchronous fashion, and need a scheduler to decide which program runs at each step. Jamroga and Dastani stress the fact that it is necessary to consider a fair scheduler, that is, a scheduler that will run all programs infinitely often. Without such a fair scheduler, the programs could fail to achieve their goal simply because a necessary action of a particular program is enabled, but the scheduler ignores it forever.

Such kind of fairness is easily expressed in logics such as $ATL^*$ and $ATL^*_{ir}$, in which the user has the full expressiveness of all combinations of path and Boolean operators to describe the objectives. For instance, in a multi-agent program with two programs $pr_1$ and $pr_2$, the fact that the programs have a strategy to achieve the objective $\psi$ under a fair scheduler can be written in $ATL^*_{ir}$ as

$$\langle\!\langle pr_1, pr_2 \rangle\!\rangle \; (( \bigwedge_{i \in \{1,2\}} \mathbf{G}\ \mathbf{F}\ run_i) \implies \psi), \tag{3.1}$$

where $run_i$ is true whenever $pr_i$ is chosen by the scheduler. Nevertheless, the complexity of the model-checking problem for these logics is higher than their non-starred counterpart: $ATL$ model checking is PTIME-complete while $ATL^*$ is 2EXPTIME-complete [AHK02]; $ATL_{ir}$ model checking is $\Delta_2^P$-complete while $ATL^*_{ir}$ is in PSPACE [Sch04].

Another solution is to consider a logic that is more expressive than $ATL_{ir}$ and can express the assumption that the scheduler is fair, but for which the model-checking problem is easier than the one for $ATL^*_{ir}$. Dastani and Jamroga propose to use $EATL^+_p$ to express these properties. It extends $ATL_{ir}$ with operators to reason about beliefs and goals of multi-agent programs, and allows, under a coalition modality $\langle\!\langle \Gamma \rangle\!\rangle$, the usage of Boolean combinations of the path operators **G**, **X** and **U**, and the special path formula **G F**. The formula of Equation 3.1 clearly falls in this logic. Furthermore, the logic has a $\Delta_3^P$-complete model-checking problem, thus easier than the full $ATL^*_{ir}$ problem. A third solution is to include the fairness assumptions directly in the semantics, but this will be discussed later in Chapter 4.

The problem of the fair scheduler for multi-agent programs raises the need for *unconditional fairness*, that is, constraints saying that something (here, that each program runs) happens *infinitely often.* Other kinds of fairness exist and have been investigated in the framework of strategic reasoning. For instance, Alur et al. proposed two types of fairness [AHK02]. They consider that a fairness constraint is a couple $\langle ag, \gamma \rangle$ where $ag$ is an agent of the system and $\gamma$ is a function mapping each state $q$ to a (possibly empty) subset of actions enabled for $ag$ in $q$. We say that $\langle ag, \gamma \rangle$ is enabled at position $d$ of a path $\pi$ if $\gamma(\pi(d)) \neq \varnothing$; we say also that $\langle ag, \gamma \rangle$ is taken at position $d$ if there exists a joint action $a$ such that there exists $a_{ag} \in \gamma(\pi(d))$ such that $a_{ag} \sqsubseteq a$. Then, Alur et al. define two interpretations for these constraints:

- a path is *weakly $\langle ag, \gamma \rangle$-fair* when, if $\langle ag, \gamma \rangle$ is eventually always enabled, then the constraint is taken infinitely often;

- a path is *strongly* $\langle ag, \gamma\rangle$*-fair* when, if $\langle ag, \gamma\rangle$ is enabled infinitely often, then the constraint is taken infinitely often.

Given a set of fairness constraints $FC$, a strategy $f_{ag}$ for agent $ag$ is strongly (resp. weakly) $FC$-fair for $q$ if for every fairness constraint of the form $\langle ag, \gamma\rangle$ in $FC$ and every path $\pi \in out(f_{ag}, q)$, $\pi$ is strongly (resp. weakly) $\langle ag, \gamma\rangle$-fair. So, the semantics of Fair $ATL$ is:

$$S, q \models \langle\!\langle \Gamma \rangle\!\rangle\ \psi \quad \Leftrightarrow \left\{ \begin{array}{l} \text{there exists an } FC\text{-fair strategy } f_\Gamma \text{ for } \Gamma \text{ s.t.} \\ \forall\ FC\text{-fair paths } \pi \in out(f_\Gamma, q), S, \pi \models \psi. \end{array} \right.$$

These fairness constraints can be used to express the assumptions of a fair scheduler (see [AHK02]), but can also express more complex fairness assumptions. Alur et al. propose a model-checking algorithm for the case of the fair scheduler, but simply reduce the problem to $ATL^*$ model checking for the general case of weak and strong fairness. Furthermore, they show that the model-checking problem for the weak version of Fair $ATL$ is PTIME-complete, while the one for the strong version is PSPACE-complete.

Another logic that includes fairness constraints in its semantics is the *Alternating-time Stream Logic* ($ASL$) proposed by Klüppelholz and Baier [KB08]. This logic is an extension of $ATL$ for reasoning about the capabilities of the agents of multi-agent systems enriched with I/O-operations, data dependencies and different kinds of channel-based communications. The notion of fairness they consider is similar to the strong fairness of Fair $ATL$: a path is fair if, whenever a channel can be written infinitely often, it is effectively written infinitely often.

## 3.4 Complexities and expressiveness

This section summarizes the complexity of the model-checking problem for the logics discussed in this Chapter. Table 3.1 gives the complexity of the model-checking problems of the logics that reason about strategies with perfect information. All these problems are decidable, but some are far from solvable in practice, the worst case being $ATL^*_{sc}$ with its NONELEMENTARY problem.

Table 3.2 presents the complexities for logics that mix strategies and knowledge. When dealing with strategies and knowledge, the problems stay easy as long as we do not consider uniform strategies. Indeed, $ATEL$, $ESL_{Bel}$ and $ECL$ model-checking problems can be solved in polynomial time because the strategies they consider still use perfect information. Nevertheless, when considering memoryless uniform strategies, the problems become more complex, ranging from $\Delta_2^P$ to $EXPSPACE$. Finally,

| Logic | Complexity |
|---|---|
| $ATL$ | PTIME |
| $af\text{-}AMC$ | PTIME |
| $CL$ | PTIME |
| $ATL^*_{sc,0}$ | PSPACE |
| $AMC$ | EXPTIME |
| $SL$ | 2EXPTIME |
| $SL$ | 2EXPTIME |
| $ATL^*_{sc}$ | NONELEMENTARY |

Table 3.1: Complexities of the model-checking problem for logics dealing with perfect information strategies.

| Logic | Complexity |
|---|---|
| $ATEL$ | PTIME |
| $ESL_{Bel}$ | PTIME |
| $ECL$ | PTIME |
| $ATL_{ir}$ | $\Delta_2^P$ |
| $af\text{-}AMC_i$ | $\Delta_2^P$ |
| $ATOL$ | $\Delta_2^P$ |
| $CSL$ | $\Delta_2^P$ |
| $ESL$ | $EXPSPACE$ |
| $ATL^D_{iR}$ | decidable |
| $ATL_{iR}$ | undecidable |
| $ATEL\text{-}R^*$ | undecidable |
| $ATL^*_{sc,iR}$ | undecidable |

Table 3.2: Complexities of the model-checking problem for logics dealing with strategies and knowledge.

when the strategies are memory-full and uniform, the problem becomes undecidable: the only exception is the case of $ATL^D_{iR}$, because it sees a group of agents as a single one, simplifying the problem to keep it decidable.

Table 3.3 lists the complexity of the model-checking problems for logics with fairness constraints. While Fair $ATL$ with weak fairness constraints remains an easy problem, $EATL^+_p$ and Fair $ATL$ with strong fairness constraints become way more complex than standard $ATL$.

Finally, Figure 3.1 shows how the different logics compare to each others in terms of expressiveness.

| Logic | Complexity |
|---|---|
| Fair $ATL$ (weak) | PTIME |
| $EATL_p^+$ | $\Delta_3^P$ |
| Fair $ATL$ (strong) | PSPACE |

Table 3.3: Complexities of the model-checking problem for logics dealing with fairness constraints.

$$\begin{tikzcd}
ECL \arrow[r] & ATEL \arrow[r] \arrow[d, leftrightarrow] & ESL_{Bel} & & & \\
& & & af\text{-}AMC \arrow[drr] & & SL \arrow[d, leftrightarrow, "/" marking] \\
CL \arrow[r] & ATL \arrow[r] \arrow[urr] \arrow[ddd] & \text{Fair } ATL \arrow[rr] & & ATL^* \arrow[r] \arrow[ur] \arrow[dr] & AMC \arrow[d, leftrightarrow, "/" marking] \\
& & & & AMC_i & ATL^*_{sc} \\
& & af\text{-}AMC_i \arrow[urr] \arrow[drr, leftrightarrow, "/" marking] & & ATL^D_{iR} \arrow[d] & \\
& ATL_{ir} \arrow[rrr] \arrow[ur, leftrightarrow, "/" marking] \arrow[dr] \arrow[ddr] \arrow[dddr] & & & ATL_{iR} \arrow[r] \arrow[d] & ATL_{sc,iR} \\
& & ATOL \arrow[rr] \arrow[drr] & & ATEL\text{-}R^* & \\
& & ESL & & CSL & \\
& & EATL^+_p \arrow[rr] & & ATL^*_{ir} &
\end{tikzcd}$$

Figure 3.1: Comparison of the expressiveness of the logics. An arrow from logic $\mathcal{L}_1$ to $\mathcal{L}_2$ means that $\mathcal{L}_2$ is more expressive than $\mathcal{L}_1$. A crossed double arrow means that the two logics have incomparable expressiveness.

## 3.5 Model checking uniform strategies

While many logics were proposed to reason about the strategies and knowledge of the agents of a system, only few effort has been brought to practically solve their model-checking problem. $ATL$ and the like (such as $ATEL$) already have a fixpoint-based model-checking algorithm (see Section 2.2.3), but it has been shown that uniform strategies based logics such as $ATL_{ir}$ do not share the fixpoint characterization needed to be able to use similar constructs [JB11].

This section first presents model-checking algorithms for $ATL_{ir}$, then it describes an algorithm proposed by Huang and van der Meyden for their $ESL$ logic. Finally, it discusses related algorithms from the game theory field. The approaches of Pilecki et al. and Huang and van der Meyden are explained in more detail because Chapter 6 adapts and implements them in a BDD-based framework.

### 3.5.1 Model-checking algorithms for alternating-time temporal logic with imperfect information

Three algorithms have been proposed to solve the model-checking problem for $ATL_{ir}$. This section presents them.

#### Lomuscio and Raimondi

Lomuscio and Raimondi proposed an algorithm to perform the model checking of uniform strategies for interpreted systems [LR06b]. In this context, a $\Gamma$-uniform interpreted system, for a subset of agents $\Gamma \in Ag$, is an interpreted system in which the protocols of agents of $\Gamma$ are restricted to propose only one action to play in every local state. A $\Gamma$-uniform interpreted system is compatible with another (not necessarily $\Gamma$-uniform) interpreted system if they share the same agents, states and actions, the same evolution and valuation functions, and the same initial states, and the protocols of the first one are restrictions of the corresponding protocols of the second one. From a given interpreted system and a subset of agents $\Gamma$, we can build the set of compatible $\Gamma$-uniform interpreted systems compatible with it. Lomuscio and Raimondi propose to check the existence of uniform strategies that win the objectives in $\phi$ by checking $\phi$ on the compatible $\Gamma$-uniform interpreted systems. The formula is then true in the original interpreted system if there exists a compatible $\Gamma$-uniform interpreted system in which $\phi$ is true.

This semantics is different from $ATL_{ir}$. In the case of Lomuscio and Raimondi, strategic formulas are interpreted globally, in the sense that a formula $\phi$ is true in an interpreted system if there exists a compatible uniform interpreted system in which the formula is true. Thus, all strategic sub-formulas of $\phi$ must share the same strategy. For instance, let $\phi = \langle\langle\Gamma\rangle\rangle\mathbf{F}\ p \wedge \langle\langle\Gamma\rangle\rangle\mathbf{G}\ q$; $\phi$ is true if and only if there exists a uniform strategy such that all enforced paths satisfy $\mathbf{F}\ p$ and $\mathbf{G}\ q$.

This characteristics is also applied to different states. Let us consider that the model has two initial states; then the formula $\phi = \langle\langle\Gamma\rangle\rangle\mathbf{F}\ p$ is true if there exists a compatible $\Gamma$-uniform interpreted system in which $\phi$ is true. In terms of strategies, this means that the same uniform strategy must be winning for both states, even if the two can be distinguished by $\Gamma$. This also means that, if $\Gamma$ must play two different uniform strategies in the two states to win their objective in both states, the formula is false.

On the other hand, $ATL_{ir}$ semantics is local to the sub-formula, that is, the formula $\phi = \langle\!\langle\Gamma\rangle\!\rangle\mathbf{F}\ p \wedge \langle\!\langle\Gamma\rangle\!\rangle\mathbf{G}\ q$ is true if there exists a strategy such that all paths satisfy $\mathbf{F}\ p$, and there exists another strategy such that all paths satisfy $\mathbf{G}\ q$. Furthermore, for different states, different strategies can be winning, as soon as the states are distinguishable by $\Gamma$.

## Calta et al.

Another algorithm was proposed by Calta et al. [CSS10]. They propose to solve the problem of model checking $ATL_u$ formulas, a new logic corresponding to $ATL_{ir}$ interpreted over sets of states of iCGS. Their algorithm works by computing, for a formula $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$, the union of all strategies for $\Gamma$ that are winning for $\psi$, represented as a set of moves, that is, a set of pairs composed of a state and a joint action for $\Gamma$. Then, from this set of moves, they extract the list of maximal uniform strategies and remove the ones that are not winning for $\psi$. In particular, they use a sub-algorithm to find all maximal cliques of a graph derived from the set of moves to find all the maximal uniform strategies.

## Pilecki et al.

A last algorithm was recently proposed by Pilecki et al. [PBJ14]. This algorithm solves the model-checking problem for a variant of $ATL_{ir}$ that considers $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$ true in a state if there exists a collective uniform strategy for $\Gamma$ that wins for $\psi$ in the current state only (instead of in all indistinguishable states). Furthermore, the algorithm is limited to formulas of the form $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$ where $\psi$ contains no strategic operator.

Their algorithm for checking whether a state $q$ satisfies a formula $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$ first guesses nondeterministically a collective strategy $f_\Gamma$, then performs $CTL$ model checking of $\mathbf{A}\ \psi$ on $q$ in the model restricted to the behaviors allowed by $f_\Gamma$. To reduce the number of strategies to guess, they explore some equivalences between strategies and use a representation that allows to try simple strategies first. They also propose to explore so-called *path-based* strategies instead of standard ones, but the technique is incomplete in the sense that the algorithm could say that the formula is not satisfied while it actually is; this idea is not discussed here.

An *incomplete* strategy $f_{ag}$ is a strategy defined by a partial function $f_{ag} : Q \nrightarrow Act$ instead of a total one. The *domain* of $f_{ag}$, written

$dom(f_{ag})$, is the set states in which the strategy is defined. Given an iCGS $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V\rangle$ and a (possibly incomplete) strategy $f_{ag}$, the *trimmed model* is the model $S\dagger f_{ag} = \langle Ag, Q, Q_0, Act, e', \delta, \sim, V\rangle$ where $e'(ag') = e(ag')$ for all $ag' \neq ag$ and $e'(ag)(q) = \{f_{ag}(q)\}$ if $q \in dom(f_{ag})$ and $e'(ag)(q) = e(ag)(q)$ otherwise. Given a collective strategy $f_\Gamma$ for a group of agents $\Gamma$, $S\dagger f_\Gamma$ is defined in the same way. Finally, we call the *proper domain* $Q_{f_\Gamma}$ of $f_\Gamma$ the set of states reachable from the initial states in the model $S\dagger f_\Gamma$, and we say that an incomplete strategy $f_\Gamma$ is *proper* if $dom(f_\Gamma) = Q_{f_\Gamma}$. It is easy to see that the choices a strategy $f_\Gamma$ makes for states in $Q\backslash Q_{f_\Gamma}$ has no incidence on the fact that the strategy is winning for any objective since these states are never reached and the choices are never actually used. Thus, it is not necessary to check all strategies, but only one per class of strategies that make the same choices on their proper domains.

A *partial* strategy for agent $ag$ is a nondeterministic possibly incomplete strategy $f_{ag} : Q \nrightarrow 2^{Act}$ such that for every state $q \in dom(f_{ag})$, we have $f_{ag}(q) = e_{ag}(q)$ or $f_{ag}(q)$ is a singleton. That is, a partial strategy either makes a choice in state $q$, or makes no restriction, meaning that the choice in this particular state is not relevant. The *implicit* part of a partial strategy is the set of states for which the strategy chooses one action, the *explicit* part is the rest of its domain. A strategy is *empty* if its implicit part is empty. We can easily lift the notion of partial strategy to collective strategies for a group of agents.

The idea of the algorithm is to restrict the search for a winning strategy to the proper ones, and to start the synthesis of these strategies from the empty partial strategy. Algorithm 3.1 presents the algorithm proposed by Pilecki et al. It works with a list of strategy tasks $\langle F, U, f_\Gamma\rangle$ where $F$ is the *fixed* set of states for which $f_\Gamma$ already defines an action for all agents, $U$ is the *unfixed* rest of the states of the domain of $f_\Gamma$, and $f_\Gamma$ is a partial strategy. Starting from the empty partial strategy, the algorithm takes one strategy task at a time, tries to extend it by making a choice in some unfixed state, and checks whether the resulting strategy satisfies $\mathbf{A}\ \psi$ or not in the restricted model. If this is the case, the algorithm found a winning strategy, otherwise another strategy task is analyzed, until no strategy tasks remain. One of the advantages of this algorithm is that it can produce—by returning the winning strategy beside the answer *yes*—relatively small descriptions of winning strategies. Indeed, when a strategy is found, it only describes what are the choices of the agents in the first steps of the game while keeping the unimportant further moves unspecified.

**Algorithm 3.1:**

```
Data: $S$ an iCGS, $\phi = \langle\langle \Gamma \rangle\rangle\ \psi$ an $ATL_{ir}$ formula, $q$ a state of $S$.
Result: Whether $S, q \models \phi$.
$STL = \langle \langle \varnothing, \{q\}, emptyPartialStrategy \rangle \rangle$
while $STL \neq \langle\rangle$ do
    pick and remove one $ST = \langle F, U, f_\Gamma \rangle$ from $STL$
    pick one state $q' \in U$
    $F = F \cup \{q'\}$, $U = U \backslash \{q'\}$
    if there exists an agent $ag \in \Gamma$ with no fixed action in $q'$ by $f_\Gamma$
    then
        pick one agent $ag \in \Gamma$ with no fixed action in $q'$ by $f_\Gamma$
        for all actions $a$ enabled in $q'$ for $ag$ compatible with $f_\Gamma$ do
            fix $a$ as the choice in $q'$ for $ag$ in $f_\Gamma$
            add to $U$ the successors of $q'$ that are not in $F \cup U$
            $STL = STL + \langle F, U, f_\Gamma \rangle$
        set $f_\Gamma$ as the strategy of the first task added to $STL$
    else
        if $U \neq \varnothing$ then
            add to $U$ the successors of $q'$ that are not in $F \cup U$
            $STL = STL + \langle F, U, f_\Gamma \rangle$
    if $S \dagger f_\Gamma, q' \models \boldsymbol{A}\ \psi$ then
        return yes
return no
```

### 3.5.2 Epistemic strategy logic

Huang and van der Meyden proposed a BDD-based algorithm to solve the model-checking problem for $ESL$ [HvdM14b]. While their description of $ESL$ does not integrate fairness constraints, their algorithm supports them and can be tuned to work for $ATLK_{irF}$, the logic described in this thesis. Chapter 6 explains how this can be done.

The idea of the algorithm is to encode the strategies of the agents into BDD variables and to solve the model-checking problem with BDD manipulations and fixpoints computations. More precisely, given a set of agents $Ag$ and a strategic environment $E = \langle Q, Q_0, Act, R, \sim, V, FC \rangle$ with fairness constraints $FC \subseteq 2^Q$, the interpreted system defined over the uniform deterministic strategies for the agents of $Ag$ can be encoded with BDDs as follows:

- Each state $q \in Q$ is represented as a Boolean assignment to the set

$AP$ of atomic propositions, that is, $Q$ is represented as a Boolean formula over $AP$. The initial states $Q_0$ and fairness constraints $fc \in FC$ are represented similarly.

- The actions $Act_{ag}$ of each agent are represented using a set of Boolean variables $BAct_{ag} = \{b_{ag,1}, ..., b_{ag,m_{ag}}\}$ where $m_{ag}$ is the number of Boolean variables needed to encode all actions. A joint action is represented as an assignment to $BAct = \bigcup_{ag \in Ag} BAct_{ag}$.
- The transition relation $R$ is encoded using copies of the atomic propositions $AP' = \{p' \mid p \in AP\}$. The relation is thus represented as a Boolean formula over variables of $AP \cup BAct \cup AP'$.
- The observation functions $\sim_{ag}$ can be represented by giving a subset $AP_{ag} \subseteq AP$ of atomic propositions observed by agent $ag$. We use the formula $eq_{ag} \equiv \bigwedge_{p \in AP_{ag}} p \iff p'$ to represent the fact that agent agent $ag$ has the same observations in the assignments of variables $AP$ and $AP'$ respectively.

Let $\chi$ be an assignment of Boolean values to variables of a set $X$. We write $\chi[X \to X']$ for the assignment that gives to variable $x' \in X'$ the value of $x \in X$ from assignment $\chi$.

Let $\mathcal{O}_{ag} = \{\sim(ag)(q) \mid q \in Q\}$ be the observations of agent $ag$. A uniform deterministic strategy for agent $ag$ can be seen as a function $f_{ag} : \mathcal{O}_{ag} \to Act_{ag}$. To represent the strategies, we introduce, for each agent $ag$, a set of variables $X_{ag}$ containing the variables $x_{ag,o,j}$ for $o \in \mathcal{O}_{ag}$ and $j = 1..m_{ag}$. In this context, a strategy $f_{ag}$ is an assignment $\chi_{f_{ag}}$ to variables of $X_{ag}$ such that for $o \in O_{ag}$ and $j = 1..m_{ag}$, $\chi_{f_{ag}}(x_{ag,o,j}) = 1$ if $f_{ag}(o)(b_{ag,j}) = 1$. Let $X = \bigcup_{ag \in Ag} X_{ag}$. Using these variables, the formula

$$f_{strat} \equiv \bigwedge_{ag \in Ag} \bigwedge_{o \in \mathcal{O}_{ag}} \left( \hat{o} \implies \bigwedge_{j=1..m_{ag}} (x_{ag,o,j} \iff b_{ag,j}) \right),$$

where $\hat{o}$ is the conjunction of the literals corresponding to the assignment $o$, says that the agents select the action corresponding to their current strategy. Finally, to encode a context $C$, we introduce a set of variables $X^y = \{x^y \mid x \in AP \cup X\}$ for each variable $y \in Var$ in the formula to be checked.

The model-checking algorithm is defined as the Boolean formula representing the set of states satisfying a given formula. It is defined as follows:

$$eval_{ESL}(E, p) = p,$$
$$eval_{ESL}(E, \neg\phi) = \neg\Phi,$$

$$\begin{aligned}
eval_{ESL}(E, \phi_1 \vee \phi_2) &= \Phi_1 \vee \Phi_2,\\
eval_{ESL}(E, \mathbf{EX}\ \phi) &= ex(fair \wedge \Phi),\\
eval_{ESL}(E, \mathbf{EG}\ \phi) &= \nu Z.\ \Phi \wedge \bigwedge_{fc \in FC} ex(eu(\Phi, Z \wedge fc)),\\
eval_{ESL}(E, \mathbf{E}[\phi_1\ \mathbf{U}\ \phi_2]) &= \mu Z.\ (\Phi_2 \wedge fair) \vee (\Phi_1 \wedge ex(Z)),\\
eval_{ESL}(E, \exists y.\phi) &= \exists X^y \text{ s.t. } ((fair \wedge reach)[X \to X^y] \wedge \Phi),\\
eval_{ESL}(E, e_\Gamma(y)) &= \bigwedge_{ag \in \Gamma} \bigwedge_{p \in AP_{ag}} p \iff p^y,
\end{aligned}$$

$$eval_{ESL}(E, \mathbf{D}_\Gamma\ \phi) = \forall AP' \cup X', (\bigwedge_{ag \in \Gamma} eq_{ag} \wedge reach' \wedge fair' \implies \Phi'),$$

$$eval_{ESL}(E, \mathbf{C}_\Gamma\ \phi) =$$
$$\nu Z.\ \bigwedge_{ag \in \Gamma} \forall AP' \cup X', (eq_{ag} \wedge reach' \wedge fair' \implies (\Phi \wedge Z')),$$

where

$$\begin{aligned}
ex(Z) &= \exists AP' \cup X' \text{ s.t. } (R \wedge f_{strat} \wedge Z'),\\
eu(Z_1, Z_2) &= \mu Z.\ Z_2 \vee (Z_1 \wedge ex(Z)),\\
reach &= \mu Z.\ Q_0 \vee (\exists AP \cup BAct \text{ s.t. } Z \wedge R \wedge f_{strat})[AP' \to AP],\\
fair &= \nu Z.\ \bigwedge_{fc \in FC} ex(eu(true, Z \wedge fc)),\\
\Phi &= eval_{ESL}(E, \phi),\\
\Phi_1 &= eval_{ESL}(E, \phi_1),\\
\Phi_2 &= eval_{ESL}(E, \phi_2).
\end{aligned}$$

$eval_{ESL}(E, \phi)$ returns the Boolean formula representing the set of states of $E$ satisfying $\phi$ [HvdM14b].

### 3.5.3 Other algorithms

Algorithms for model checking other logics reasoning about imperfect information strategies have also been proposed. For instance, Raskin et al. proposed an algorithm to check the existence of observation-based strategies for two-player turn-based games on graphs with $\omega$-regular objectives [RCDH07]. These games are composed of two players, each state of the game being owned by one of them. The objectives they try to achieve are specified as infinite plays, and they specifically consider reachability, safety, Büchi and coBüchi, and parity objectives, that are all $\omega$-regular. They are interested in the existence of winning observation-based strategies, that is, strategies with imperfect information and perfect recall. One limitation of their approach is the fact that the objective must be expressed in terms of the observations of the agent that tries to achieve them. They propose a fixpoint-based algorithm that computes the set of states in which a player has a strategy to win his objective. The fixpoint is computed in the lattice of antichains of state sets, that is, the lattice of downward-closed subsets of states.

Another algorithm was recently proposed by Bozianu et al. [BDF14]. Their algorithm deals with the synthesis of a strategy with imperfect information and perfect recall for a single agent. The objectives of the agent are expressed in an extension of $LTL$ with a knowledge operator to reason about what the agent knows. As above, their algorithm works with antichains.

# *Chapter 4*

# *Reasoning about uniform strategies under fairness constraints*

The example of the repeated card game presented in the Introduction showed that, by assuming that the dealer will distribute all pairs of cards infinitely often—that is, by assuming a *fair* dealer—, the player has a strategy to win the game, even if he does not see the card of the dealer. Reasoning about fairness assumptions can be useful in other scenarios, too. For instance, in a system with agents communicating through a lossy channel, it is reasonable to assume that the channel will not lose messages forever. Finally, Section 3.3 showed that it is interesting to reason about the strategies of multi-agent programs under the assumption that the scheduler will not ignore a program forever. This chapter presents the syntax and semantics of $ATLK_{irF}$, an extension of $ATL_{ir}$ with *unconditional* fairness constraints on *states*. This logic is adequate to reason about all the cases above. Section 4.1 presents the syntax of the logic, Section 4.2 the models it is interpreted over, and Section 4.3 describes its semantics. Section 4.4 discusses some choices and properties of the logic, and Section 4.5 compares it with existing logics presented in the previous chapter.

## 4.1 Syntax

$ATLK_{irF}$ formulas are composed of atomic propositions, the standard Boolean operators, the $CTL$ operators, the knowledge operators **K**, **D**, **E** and **C**, and the $ATL$ strategic operators. More precisely, $ATLK_{irF}$

formulas follow this grammar:

$$\phi ::= true \mid p \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{E}\ \psi \mid \langle\!\langle \Gamma \rangle\!\rangle\ \psi \mid \mathbf{K}_{ag}\ \phi \mid \mathbf{E}_\Gamma\ \phi \mid \mathbf{D}_\Gamma\ \phi \mid \mathbf{C}_\Gamma\ \phi$$
$$\psi ::= \mathbf{X}\ \phi \mid \phi\ \mathbf{U}\ \phi \mid \phi\ \mathbf{W}\ \phi$$

where $p$ is an atomic proposition of a set $AP$, $\Gamma$ is a subset of a set of agents $Ag$, $ag$ is an agent of $Ag$.

As for $CTL$ and $ATL$, the other standard Boolean, $CTL$ and $ATL$ operators can be defined in terms of these ones. Nevertheless, unlike for $CTL$, the path operator $\mathbf{W}$ is needed for expressing the dual operator for $\langle\!\langle \Gamma \rangle\!\rangle[\phi_1\ \mathbf{U}\ \phi_2]$ [LMO08]. Furthermore, the standard $\mathbf{G}\ \phi$ path operator can be expressed as $\phi\ \mathbf{W}\ false$. We can thus limit ourselves to the minimal set of path operators composed of $\mathbf{X}$, $\mathbf{U}$ and $\mathbf{W}$.

## 4.2 Models

$ATLK_{irF}$ formulas are interpreted over the states of *imperfect information concurrent game structures with fairness constraints* (iCGSf). These structures extend imperfect information concurrent game structures with unconditional fairness constraints on states. More precisely, an iCGSf is a structure $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC \rangle$ such that

- $Ag$ is a finite set of *agents*;
- $Q$ is a finite set of *states*;
- $Q_0 \subseteq Q$ is the set of *initial* states;
- $Act$ is a finite set of *actions*; a *joint action* is a tuple $a \in Act^{Ag}$ of actions, one for each agent of $Ag$;
- $e : Ag \to (Q \to (2^{Act} \backslash \varnothing))$ defines, for each agent $ag$ and state $q$, the set of actions $ag$ can choose in $q$, that is, the actions *enabled* in $q$; we write $e_{ag}$ for the function $e(ag)$ giving the non-empty set of actions $ag$ can choose in any state;
- $\delta : Q \times Act^{Ag} \nrightarrow Q$ is a partial deterministic *transition function* defined for each state $q \in Q$ and each joint action enabled in $q$; we write $q \xrightarrow{a} q'$ for $\delta(q, a) = q'$;
- $\sim: Ag \to 2^{Q \times Q}$ defines a set of *equivalence classes* representing the observability of agents; we write $\sim_{ag}$ for $\sim (ag)$ (and call it the *epistemic relation* of $ag$) and we assume that each agent can choose his actions based on his own knowledge of the system, that is,

$$\forall q, q' \in Q, q \sim_{ag} q' \implies e_{ag}(q) = e_{ag}(q')$$

for any agent $ag \in Ag$;

- $V : Q \rightarrow 2^{AP}$ is a function *labeling* states with atomic propositions from a given set $AP$;
- $FC \subseteq 2^{Q}$ is a set of *fairness constraints*.

An iCGSf is a standard iCGS to which we add some unconditional fairness constraints on states, similarly to Fair $CTL$.

Given a set of agents $\Gamma \in Ag$, the relations $\sim^{\mathbf{D}}_{\Gamma}$, $\sim^{\mathbf{E}}_{\Gamma}$ and $\sim^{\mathbf{C}}_{\Gamma}$ are defined in the standard way, that is, $\sim^{\mathbf{D}}_{\Gamma} = \bigcap_{ag \in \Gamma} \sim_{ag}$ defines the *distributed knowledge* relation of group $\Gamma$, $\sim^{\mathbf{E}}_{\Gamma} = \bigcup_{ag \in \Gamma} \sim_{ag}$ is the *group knowledge* relation of $\Gamma$, and $\sim^{\mathbf{C}}_{\Gamma}$, defined as the reflexive transitive closure of the relation $\sim^{\mathbf{E}}_{\Gamma}$, is the *common knowledge* relation of $\Gamma$. We write

$$[Q']_{ag} = \{q' \in Q \mid \exists q \in Q' \text{ s.t. } q \sim_{ag} q'\}$$

for the set of states indistinguishable by agent $ag$ from a state of $Q'$, and

$$[Q']^{\mathcal{K}}_{\Gamma} = \{q' \in Q \mid \exists q \in Q' \text{ s.t. } q \sim^{\mathcal{K}}_{\Gamma} q'\}$$

for the set of states indistinguishable by group $\Gamma$ through knowledge relation $\sim^{\mathcal{K}}_{\Gamma}$ from a state of $Q'$, where $\mathcal{K} \in \{\mathbf{E}, \mathbf{D}, \mathbf{C}\}$.

The different notions of (i)CGS are lifted to iCGSf. More precisely, a joint action $a \in Act^{Ag}$ *completes* an action $a_{\Gamma} \in Act^{\Gamma}$ for a set of agents $\Gamma$, written $a_{\Gamma} \sqsubseteq a$, if the action for each agent of $\Gamma$ in $a$ corresponds to the action of the same agent in $a_{\Gamma}$. Given a joint action $a \in Act^{Ag}$ and a set of agents $\Gamma \subseteq Ag$, we write $a(\Gamma)$ for the tuple of actions of agents of $\Gamma$ in $a$; when $\Gamma = \{ag\}$ is a singleton, we write $a(ag)$ instead of $a(\{ag\})$. The function $E : 2^{Ag} \rightarrow (Q \rightarrow 2^{Act^{Ag}})$ is defined as $E(\Gamma)(q) = \prod_{ag \in \Gamma} e_{ag}(q)$ and returns the set of actions for $\Gamma$ enabled in $q$; we write $E_{\Gamma}$ for $E(\Gamma)$. Finally, we call a $\Gamma$-*move* (or a *move* if $\Gamma$ is clear from the context) an element $\langle q, a_{\Gamma}\rangle \in Q \times Act^{\Gamma}$ such that $a_{\Gamma} \in E_{\Gamma}(q)$, that is, a pair composed of a state and an action for $\Gamma$ enabled in the state.

A *path* in an iCGSf $S$ is a sequence $\pi = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} ...$ such that $\delta(q_d, a_{d+1}) = q_{d+1}$ for all $d \geq 0$. We write $\pi(d)$ for $q_d$, and $|\pi|$ for the number of states of $\pi$. A path in $S$ is *fair* if it meets all fairness constraints of $S$ infinitely often, that is, $\pi$ is fair if, for each fairness constraint $fc \in FC$, there exist infinitely many indices $d$ such that $\pi(d) \in fc$. A state $q$ is *reachable* in $S$ if there exists a path $\pi$ in $S$ such that $\pi(0) \in Q_0$ and there exists $d \geq 0$ such that $\pi(d) = q$. A state $q$ is *fair* if there exists a fair path starting at $q$. A fair reachable state is thus a state belonging to a fair path starting at an initial state of $S$.

A *memoryless strategy* for agent $ag$ is a function $f_{ag} : Q \rightarrow Act$ such that $\forall q \in Q, f_{ag}(q) \in e_{ag}(q)$. A (memoryless) *uniform* strategy for agent

$ag$ is a strategy $f_{ag}$ such that $\forall q, q' \in Q, q \sim_{ag} q' \implies f_{ag}(q) = f_{ag}(q')$. We call *outcomes* of a strategy the set of paths of the structure that are coherent with the strategy; more precisely, the outcomes of a strategy $f_{ag}$ for agent $ag$ from a state $q$ are defined as

$$out(f_{ag}, q) = \{\pi = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \mid q_0 = q \wedge \forall d \in \mathbb{N}, f_{ag}(q_d) \sqsubseteq a_{d+1}\}. \quad (4.1)$$

Finally, a (uniform) strategy for a group of agents $\Gamma \subseteq Ag$ is a tuple of (uniform) strategies, one for each agent of $\Gamma$. The outcomes of a strategy $f_\Gamma$ for a group of agents $\Gamma$ from a state $q$ are defined as

$$out(f_\Gamma, q) = \bigcap_{f_{ag} \in f_\Gamma} out(f_{ag}, q); \quad (4.2)$$

these outcomes are the paths that are coherent with every strategy of the set $f_\Gamma$.

In the sequel, we mainly speak about uniform strategies and call them strategies; when speaking about strategies that are not necessarily uniform, we speak about *general* strategies. A strategy $f_\Gamma$ can be represented as the set of $\Gamma$-moves

$$\{\langle q, a_\Gamma \rangle \in Q \times Act^\Gamma \mid a_\Gamma = f_\Gamma(q)\}, \quad (4.3)$$

that is, the set of moves such that the actions are the ones specified by the strategy. In the sequel, the notation $f_\Gamma$ is interchangeably used for a set of $\Gamma$-moves and the strategy they represent. Furthermore, we say that a set of $\Gamma$-moves $M_\Gamma$ *covers* a set of states $Q' \subseteq Q$ if

$$\forall q \in Q', \exists \langle q', a'_\Gamma \rangle \in M_\Gamma \text{ s.t. } q' = q;$$

in other words $M_\Gamma$ covers $Q'$ if $M_\Gamma$ proposes an action for all states of $Q'$. We write $M_\Gamma|_Q$ for the set of states $M_\Gamma$ covers. We also interchangeably write $E_\Gamma$ for the original function taking a state $q$ and returning the set of actions $\Gamma$ can play in $q$, and for the set of $\Gamma$-moves it represents, that is, the set $\{\langle q, a_\Gamma \rangle \in Q \times Act^\Gamma \mid a_\Gamma \in E_\Gamma(q)\}$.

Finally, the outcomes function *out* is lifted for any subset of $\Gamma$-moves $M_\Gamma$ as follows:

$$out(M_\Gamma, q) = \left\{ \pi = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \;\middle|\; \begin{array}{l} q_0 = q \wedge \forall i, 0 \le i < |\pi| - 1, \\ \exists \langle q', a'_\Gamma \rangle \in M_\Gamma \text{ s.t.} \\ q' = q_i \wedge a'_\Gamma \sqsubseteq a_{i+1} \end{array} \right\},$$

that is, $out(M_\Gamma, q)$ is the set of (finite or infinite) paths that follow some actions for $\Gamma$ proposed by $M_\Gamma$.

## 4.3 Semantics

The semantics of $ATLK_{irF}$ formulas is defined over states of an iCGSf $S$ by the relation $S, q \models \phi$; $S$ is omitted when clear from the context. This relation meets the standard semantics for propositional, branching-time and knowledge operators. For strategic operators, the semantics is similar to $ATL_{ir}$ semantics except that it considers only fair paths in the outcomes of strategies. More precisely, the $q \models \phi$ relation is defined as

$$\begin{array}{lll}
q \models true, & & \\
q \models p & \Leftrightarrow & p \in V(q), \\
q \models \neg\phi & \Leftrightarrow & q \not\models \phi, \\
q \models \phi_1 \vee \phi_2 & \Leftrightarrow & q \models \phi_1 \text{ or } q \models \phi_2, \\
q \models \mathbf{E}\ \psi & \Leftrightarrow & \text{there exists a } \textit{fair path } \pi \text{ s.t. } \pi(0) = q \text{ and } \pi \models \psi, \\
q \models \mathbf{K}_{ag}\ \phi & \Leftrightarrow & \begin{cases} q' \models \phi \text{ for all } q' \in Q \text{ s.t.} \\ q \sim_{ag} q' \text{ and } q' \text{ is a } \textit{fair reachable} \text{ state,} \end{cases} \\
q \models \mathcal{K}_\Gamma \phi & \Leftrightarrow & \begin{cases} q' \models \phi \text{ for all } q' \in Q \text{ s.t.} \\ q \sim_\Gamma^{\mathcal{K}} q' \text{ and } q' \text{ is a } \textit{fair reachable} \text{ state,} \\ \text{where } \mathcal{K} \in \{\mathbf{D}, \mathbf{E}, \mathbf{C}\}, \end{cases} \\
q \models \langle\!\langle \Gamma \rangle\!\rangle\ \psi & \Leftrightarrow & \begin{cases} \text{there exists a } \textit{uniform strategy } f_\Gamma \text{ for } \Gamma \text{ s.t.} \\ \forall ag \in \Gamma, \forall q' \sim_{ag} q, \text{ for all } \textit{fair paths } \pi \in out(f_\Gamma, q'), \\ \pi \models \psi. \end{cases}
\end{array}$$

The relation $\pi \models \psi$ over paths $\pi$ of the structure $S$ is defined as

$$\begin{array}{lll}
\pi \models \mathbf{X}\ \phi & \Leftrightarrow & \pi(1) \models \phi, \\
\pi \models \phi_1\ \mathbf{U}\ \phi_2 & \Leftrightarrow & \exists d \geq 0 \text{ s.t. } \pi(d) \models \phi_2 \text{ and } \forall e < d, \pi(e) \models \phi_1, \\
\pi \models \phi_1\ \mathbf{W}\ \phi_2 & \Leftrightarrow & \begin{cases} \exists d \geq 0 \text{ s.t. } \pi(d) \models \phi_2 \text{ and } \forall e < d, \pi(e) \models \phi_1, \\ \text{or } \forall d \geq 0, \pi(d) \models \phi_1. \end{cases}
\end{array}$$

As usual, we write $S \models \phi$ if all initial states of $S$ satisfy $\phi$, that is, if

$$\forall q \in Q_0, S, q \models \phi.$$

Intuitively, this semantics say that $q$ satisfies $\langle\!\langle \Gamma \rangle\!\rangle\ \psi$ if agents in $\Gamma$ have a collective strategy such that, whatever the action of the other agents is, the objective $\psi$ will be satisfied by all the resulting fair paths from all indistinguishable states. The main difference with the standard $ATL_{ir}$ semantics is the fact that only fair paths are considered: $q$ satisfies $\langle\!\langle \Gamma \rangle\!\rangle\ \psi$ if $\Gamma$ have a strategy to enforce $\psi$, assuming all other agents will act fairly, that is, the other agents only follow fair paths. For instance, in the case of multi-agent programs, $q$ satisfies $\langle\langle\Gamma\rangle\rangle\ \psi$ if the programs in $\Gamma$ have a strategy to enforce $\psi$, assuming that the scheduler—modeled as another agent of the system—is fair, that is, each program will run infinitely often.

The restriction to memoryless strategies is strong. Nevertheless, it is necessary in the case of imperfect information as the problem of the existence of a memory-full uniform strategy under imperfect information is undecidable [DT11].

We call $ATL_{IrF}$ the restriction of $ATLK_{irF}$ that only reasons about iCGSf in which the epistemic relation of all agents are the identity relation. This means that $ATL_{IrF}$ deals with structures in which all agents observe everything, thus can base their strategies on the current state instead of on their observations of this state. In this restricted logic, knowledge operators are useless as all agents know every true fact. Some properties of this sub-logic will be discussed in the next section.

There are redundancies between the temporal and the strategic operators. The formula $\mathbf{E}\ \psi$ can be expressed in vanilla $ATL$ as $[\![\varnothing]\!]\ \psi$: there is a path satisfying $\psi$ if the empty set of agents cannot avoid $\psi$, that is, all the agents can cooperate to lead to at least one path satisfying $\psi$ [AHK02]. Similarly, $\mathbf{A}\ \psi$ can be expressed as $\langle\langle\varnothing\rangle\rangle\ \psi$, that is, all paths satisfy $\psi$ if, whatever all agents do, $\psi$ is enforced. The same equivalences apply in $ATLK_{irF}$. Nevertheless, both kinds of operators (temporal and strategic) are kept in the logic to clearly separate operators about the pure execution of the model and operators reasoning about strategies. Furthermore, the model-checking algorithms for $ATLK_{irF}$ presented in the next chapter are clearly more efficient when dealing with temporal operators than with strategic ones.

Because we deal with models with fairness constraints, some equivalences that exist in $ATL$ are not kept: $\mathbf{A}\ \psi \not\equiv [\![Ag]\!]\ \psi$ and $\mathbf{E}\ \psi \not\equiv \langle\langle Ag\rangle\rangle\ \psi$. These equivalences are broken because of fairness constraints. Let us illustrate this with an example, given in Figure 4.1. In this case, the model contains no fair path since the right state can be visited at most once. In the top state of this model, the formula $\mathbf{EX}\ p$ is trivially false because there is no fair path (at all) satisfying $\mathbf{X}\ p$; on the other hand, the formula $\langle\langle Ag\rangle\rangle\mathbf{X}\ p$ is vacuously true because $Ag$ has a strategy such that all fair paths (there are none) satisfy $\mathbf{X}\ p$. Dually, the top state satisfies $\mathbf{AX}\ p$, but it does not satisfy $[\![Ag]\!]\mathbf{X}\ p$. Note that this case is due to an ill-defined model containing no fair path, and this should be avoided in meaningful models. The problem of vacuous strategies will be further discussed in Section 4.4.2.

Finally, $ATLK_{irF}$ subsumes $ATL_{ir}$, that is, for any iCGS $S$ and $ATL_{ir}$ formula $\phi$, there exist corresponding iCGSf $T(S)$ and $ATLK_{irF}$

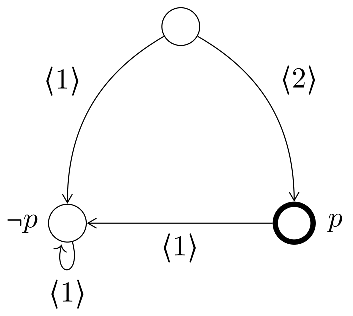


Figure 4.1: A model with one agent where the equivalences $\mathbf{A}\ \psi \not\equiv [\![Ag]\!]\ \psi$ and $\mathbf{E}\ \psi \not\equiv \langle\!\langle Ag \rangle\!\rangle\ \psi$ are broken due to presence of fairness constraints. Vertices are states, labelled with propositions true in this state; edges are transitions, labelled with joint actions. The bold state belongs to the (single) fairness constraint of the model. The agent observes everything and his epistemic relation is the identity one.

formula $T(\phi)$ such that, for any state $q$ of $S$, $S, q \vDash \phi$ if and only if $T(S), T(q) \vDash T(\phi)$. Intuitively, any iCGS can be transformed into an iCGSf with only one fairness constraint composed of the full state-space; the corresponding iCGSf shares the same labeled graph and the fairness constraint says that any path of the model is a fair path. The translation of formulas is the identity function, that is, every Boolean and strategic operator of $ATL_{ir}$ is translated into its corresponding operator of $ATLK_{irF}$.

It is obvious that there exists a uniform strategy for a group of agents $\Gamma$ in an iCGS such that all enforced paths satisfy $\psi$ if and only if there exists a uniform strategy for the same group in the corresponding iCGSf such that all enforced fair paths satisfy $\psi$, since every path of the model is fair. This implies that the model-checking problem of $ATLK_{irF}$ is $\Delta_2^P$-hard since the problem is $\Delta_2^P$-complete for $ATL_{ir}$ [Sch04]. The model-checking problem for $ATLK_{irF}$ is in fact $\Delta_2^P$-complete, but this will be discussed later in Chapter 5 after presenting model-checking algorithms for the logic that are shown to be in $\Delta_2^P$.

## 4.4 Discussion

This section discusses the choice of fairness constraints that have been made when designing $ATLK_{irF}$ semantics. Furthermore, it discusses the issue of vacuous strategies and presents possible solutions. Finally, it discusses the unimportance of memory for the $ATL_{IrF}$ sub-logic.

### 4.4.1 Fairness constraints

This section compares the fairness constraints chosen for $ATLK_{irF}$ with alternatives, and discusses the link between fairness and strategies.

$ATLK_{irF}$ considers only fair paths, that is, paths that meet each fairness constraint $fc \in FC$ infinitely often. This kind of fairness definition is called *unconditional* fairness; it is standard in the framework of Fair $CTL$ and is used in the family of SMV model checkers (e.g. NuSMV) [CGP99]. There exist other kinds of fairness such as *strong* and *weak* fairness, expressed on actions or on states, such as the ones discussed in Section 3.3. While fairness constraints can be expressed on actions or on states, fairness constraints on actions can be reduced to fairness constraints on states, thus it is sufficient to limit ourselves to the latter [BK08, Chapter 3].

The three kinds of fairness constraints can be useful to reason about the strategies of agents. For instance, unconditional fairness constraints can be used to reason about the strategies of multi-agent programs under a fair scheduler, as previously discussed. In the same vein, weak and strong fairness constraints can be used to reason about strategies of agents under particular circumstances. For instance, suppose a multi-agent program where one particular agent controls a mutex in the system. If an agent needs to lock this mutex to achieve his task, he cannot win his objective without the cooperation of the controlling agent. On the other hand, by adding strong fairness constraints to ensure that if the agent asks infinitely often for the mutex then the controlling one will grant it, the agent has a strategy to win his objective.

One advantage of unconditional fairness, compared to strong and weak fairness, is that $ATL_{IrF}$ does not differentiate between memoryless and memory-full strategies [BPQR15]. Thus we can focus on memoryless ones without losing any expressive power. On the other hand, for strong and weak fairness with perfect information, memory is necessary to win the objectives: two variants on ATL have been proposed, both needing memory [AHK02]. Nevertheless, the amount of memory needed to win the objectives under weak and strong fairness is finite. Furthermore, strategies that are allowed to use a bounded amount of memory can be reduced to memoryless strategies in a derived model where the memory is encoded in the states [JvdH04, Sch04], allowing the reasoning to be restricted to memoryless strategies.

In the full case of $ATLK_{irF}$, memory has an impact on the capabilities of the agents as $ATL_{ir}$ and $ATL_{iR}$ have different expressiveness. We propose to use unconditional fairness constraints on states to match the framework of Fair $CTL$.

Another concern about fairness is the set of agents that must enforce fair paths. Under $ATLK_{irF}$, $\Gamma$ win if they have a strategy such that *all fair paths satisfy the objective*; in this case, $\overline{\Gamma}$ have to produce fair paths that violate the objective to prevent $\Gamma$ to win. Let us call this semantics the *weak-strategy semantics.* Another choice is: $\Gamma$ win if they have a strategy such that *all enforced paths are fair and satisfy the objective.* In this case, $\Gamma$ have to enforce only fair paths to win, and $\overline{\Gamma}$ can prevent them to win by avoiding fair paths (regardless of the objective). Let us call this new semantics the *strong-strategy semantics.*

While weak-strategy objectives do not need memory to be won in the case of $ATL_{IrF}$, strong-strategy ones need memory to be won. For instance, let us consider the model presented in Figure 4.2 and the property $\langle\!\langle Ag \rangle\!\rangle \mathbf{F}\ p$. The model contains only one agent that can play two different actions in the top state. For a path to be fair, both the left and right states must be visited infinitely often. Under the weak-strategy semantics, the property is true: by playing action 0 in the top state, the agent will enforce no fair path (the only enforced path never meets the right state), and the property is vacuously true. If the agent can use memory, the result is the same, the agent still has a strategy to win. On the other hand, under the strong-strategy semantics, the agent has a memory-full strategy to win the objective—play 0 and 1 alternatively—but no memoryless one. He has to stick to the same action in the top state, and will never meet the third state. Thus, in the case of the strong-strategy semantics, memory makes a difference.

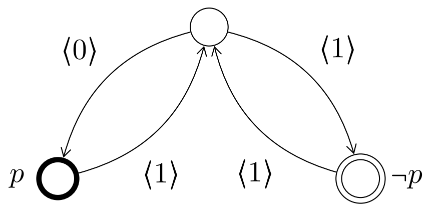


Figure 4.2: A model with one agent where the agent needs memory to win under the strong-strategy semantics. The bold state belongs to one fairness constraint, the double-lined one to another. The agent observes everything and his epistemic relation is the identity one.

When designing $ATLK_{irF}$ semantics, we kept the weak-strategy semantics because it corresponds to the usual situation in which fairness is an assumption about the environment. Furthermore, this semantics is useful, for example, to reason about multi-agent programs [DJ10]. In this case, the weak-strategy semantics allows the user to reason about the strategies of the programs while assuming a fair scheduler.

### 4.4.2 Vacuous strategies

This section discusses the problem of vacuous strategies and presents possible solutions. Let us illustrate this problem with the example of the repeated card game presented in the Introduction. We already discussed the case of the player who can eventually win the game, relying on the fair behavior of the dealer: because he knows the dealer will eventually give a winning hand, he can always keep his card and finally win the game. On the other hand, the dealer can also always win the game: he has a strategy such that the player never wins. This strategy is to avoid any fair path: if he avoids fair paths, all enforced fair paths vacuously satisfy any objective and, in particular, the objective consisting in never letting the player win.

More generally, the problem of vacuous strategies is that if a group of agents have a strategy to avoid fair paths, then they can win over any objective, even an unsatisfiable one such as $\mathbf{F}\ false$: all the fair paths (there are none) trivially satisfy any temporal formula. This problem is already present in $CTL$, where a state $q$ satisfies any $\mathbf{A}\ \psi$ property if $q$ is not fair, that is, no fair path starts in $q$. In the case of $CTL$, the usual approach is to inform the user whenever the model accepts no fair paths. This solution can be used in the present case, and we can inform the user whenever $\Gamma$ can avoid fair paths.

In the setting of $ATL_{IrF}$, either there exists a coalition that can prevent fairness, or all paths are fair and fairness constraints are useless. Indeed, the transition relation is deterministic, thus any strategy of the grand coalition of agents $Ag$ defines one single outcome. So, either all paths are fair, or there exists a strategy for $Ag$ that avoids fair paths, and $Ag$ can enforce any temporal formula by using this strategy. Nevertheless, in some models, some smaller coalitions also have vacuous strategies. For instance, in the case of the repeated card game, any coalition including the dealer can enforce unfair paths only. In the case of the lossy channel, the channel itself has a vacuous strategy as it can prevent the messages to be transmitted forever.

This property is true in the perfect information setting, but not in the general case of $ATLK_{irF}$ in which the agents have imperfect information about the system. This is shown by the model of Figure 4.3, composed of one agent that cannot distinguish the three upper states. This agent has two different uniform strategies: either play action 0 in the upper states, or play action 1. The grand coalition of all agents (there is only one) does not have a uniform strategy to avoid fair paths as both strategies will lead to the bottom left state. Nevertheless, the model contains unfair paths as the paths that end up in the bottom right states are unfair.

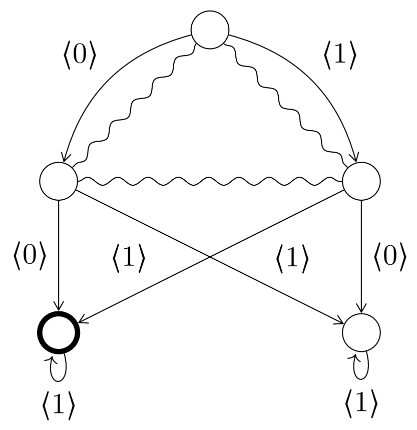


Figure 4.3: A model with one agent where the grand coalition $Ag$ cannot avoid fair paths, even with the presence of unfair paths. The bold state belongs to the single fairness constraint. Waved edges represent the epistemic relation of the agent (self loops are not pictured).

It is possible to address the problem of vacuous strategies by changing the semantics. For instance, we can change the semantics such that only strategies enforcing at least one fair path are considered. This is different from the strong-strategy semantics discussed above in the sense that, in the present case, the agents still do not need to enforce only fair paths. The only requirement is that their strategy enforce at least one fair path. In the case of the card game, this would solve the concern above: if we only consider strategies that contain at least one fair path, the dealer has no strategy to prevent the player to win, since he will need to deal each pair of cards infinitely often.

### 4.4.3 Memory and perfect information

In the case of strategies under imperfect information, the memory of the agents has a huge impact on what they can perform and on the related model-checking problems. Indeed, the model-checking problem for $ATL_{ir}$—and thus for $ATLK_{irF}$— is $\Delta_2^P$-hard, while $ATL_{iR}$ has an undecidable one [Sch04]. This difference is already visible for the simple structure of Figure 4.4 and the reachability objective $\mathbf{F}\ p$. In this structure, the single agent has a memory-full uniform strategy to reach the bottom state (play 0 then 1, for instance), but no memoryless uniform one since he must play the same action in all three top states.

On the other hand, $ATL_{IR}$ and $ATL_{Ir}$ are equivalent: when the agents have perfect information about the states of the system, they do

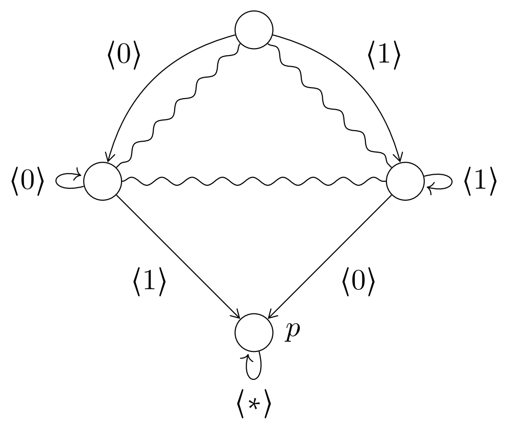


Figure 4.4: A model with one agent where the grand coalition $Ag$ has no memoryless strategy to reach the bottom state, but has a memory-full one.

not need memory to achieve their goals [AHK02]. In other words, agents with perfect information and no memory are as powerful as agents with perfect information and perfect recall, for the objectives considered by $ATL$. Nevertheless, memory has an impact on $ATL^*$ objectives, that is, there exist $ATL^*$ formulas for which memoryless agents have no winning strategy while memory-full ones have a winning strategy [AHK02].

In the case of $ATL_{IrF}$, the agents do not need memory to enforce that all fair paths satisfy a given temporal objective. The complete proof of this property is out of the scope of this thesis, but the remainder of this section presents a proof sketch. The full proof uses results from the work of E. Grädel [Grä04] and W. Thomas [Tho95], and gets inspiration from results presented in [AG11]. It is based on the fact that, when all the agents have perfect information, checking whether $S, q \models \langle\langle\Gamma\rangle\rangle\ \psi$ can be reduced to finding a winning strategy in a two-player game.

First, given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$ where

$$\forall ag \in Ag, \sim_{ag} = \{(q,q) \mid q \in Q\},$$

and a strategic $ATL_{IrF}$ formula $\langle\langle\Gamma\rangle\rangle\ \psi$, we can build a two-player turn-based game such that a state of the structure satisfies the formula if and only if the corresponding state of the game is winning for the player corresponding to $\Gamma$. Intuitively, $\Gamma$ is mapped to the first player 0 and $\overline{\Gamma}$ is mapped to the second player 1. For each action $a_\Gamma$ of $\Gamma$ enabled in a state $q$, a new state $q_{a_\Gamma}$ for 1 is created, representing the fact that $\Gamma$ chose the corresponding action; in these states for 1, $\overline{\Gamma}$ can then choose the next state.

Furthermore, the $ATLK_{IrF}$ formula is translated into an objective for the corresponding two-player turn-based game, and we can show that, to a strategy for 0 winning the objective on the game, there corresponds a strategy for $\Gamma$ in the original structure. Whenever the player 0 chooses a successor in the game, there exists a corresponding action in the original structure, and every path enforced in the game by the winning strategy can be mapped to a path in the original model, enforced by the corresponding strategy.

Finally, using standard results in game theory about memoryless strategies of particular objectives, we can show that our objectives do not need memory to be won in such games. For this, the proof proposes custom algorithms to find all the winning states of the game. By the way the algorithms compute these states, we can show that there exists a winning memoryless strategy in these states, and so the player does not need memory to win the game. Thus, since there exists a correspondence between winning strategies in the game and winning strategies in the original structure, agents do not need memory either in the case of $ATL_{IrF}$ objectives on iCGSf. The full proof is given in [BPQR15].

## 4.5 Comparison with related work

This section briefly discusses the relation between $ATLK_{irF}$ and its restriction $ATL_{IrF}$, and different logics presented in Chapter 3.

First, it is clear that $ATLK_{irF}$ extends and subsumes Fair $CTL$. A fair Kripke structure can be translated into an iCGSf with the same states, one agent, and the same labeling and fairness constraints. Then, the Fair $CTL$ operators are the same as the $ATLK_{irF}$ temporal operators as they have the same semantics. Nevertheless, while Fair $CTL$ model-checking problem is polynomial [CGP99], the $ATLK_{irF}$ one is more complex. This is due to the strategic operators, not to temporal ones. $ATLK_{irF}$ can be viewed as an extension of Fair $CTL$ with knowledge and strategic operators.

On the other hand, there is no direct comparison of $ATLK_{irF}$ with Fair $ATL$. The former reasons about uniform memoryless strategies while the latter reasons about general memory-full strategies. Furthermore, the fairness constraints are not the same as $ATLK_{irF}$ uses *unconditional* fairness constraints on *states* while Fair $ATL$ works with *weak* and *strong* fairness constraints on *actions*. Nevertheless, a version of Fair $ATL$ with state-based fairness constraints would subsume $ATL_{IrF}$.

Second, the strategic fragment of $ATLK_{irF}$—the logic from which we remove the knowledge operators—is subsumed by $ATL^*_{ir}$. Indeed, given an iCGSf, we can translate it into an iCGS in which the fairness constraints are replaced by fresh atomic propositions. More precisely, given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC \rangle$, $S$ can be translated into an iCGS $T(S) = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V' \rangle$ in which the labeling function is updated as

$$V'(q) = V(q) \bigcup_{fc \in FC} \{fc \in AP' \mid q \in fc\},$$

where $AP' = AP \cup \{fc \mid fc \in FC\}$. Then, an $ATLK_{irF}$ strategic formula $\phi = \langle\!\langle \Gamma \rangle\!\rangle\ \psi$ can be translated into an $ATL^*_{ir}$ formula

$$T(\phi) = \langle\!\langle \Gamma \rangle\!\rangle \left( \left( \bigwedge_{fc \in FC} \mathbf{G}\ \mathbf{F}\ fc \right) \implies \psi \right).$$

It is easy to show that $S, q \models \phi$ if and only if $T(S), q \models T(\phi)$. Indeed, $T(\phi)$ is true if there exists a uniform strategy for $\Gamma$ such that all paths satisfy $((\bigwedge_{fc \in FC} \mathbf{G}\ \mathbf{F}\ fc) \implies \psi)$, that is, all paths meeting all $fc \in FC$ infinitely often satisfy $\psi$, and that is exactly the semantics of $\phi$. One advantage of $ATLK_{irF}$ compared to $ATL^*_{ir}$ is the complexity of its model-checking problem; the former is $\Delta_2^P$-complete while the latter is PSPACE.

Dastani and Jamroga proposed $EATL^+_p$ to reason about goals, beliefs and strategies of multi-agent programs [DJ10]. This logic also subsumes the strategic fragment of $ATLK_{irF}$ as the translation above also applies to $EATL^+_p$. But, again, the complexity of the model-checking problem is lower for $ATLK_{irF}$ as the problem is $\Delta_3^P$-complete for $EATL^+_p$.

Finally, Huang and van der Meyden proposed a variant of $ESL$ with unconditional fairness constraints. $ATLK_{irF}$ is subsumed by this logic as $ATL_{ir}$ is subsumed by their original proposal for $ESL$ [HvdM14b]. As for the previous logics, the gain of defining a less expressive logic is the gain in complexity as $ESL$ model-checking problem is PSPACE-complete.

# *Chapter 5*

# *Model Checking uniform strategies under fairness constraints*

In the previous chapter, we presented $ATLK_{irF}$, a logic for reasoning about the temporal evolution, the knowledge and the strategies of agents with imperfect information in the context of a system with fairness constraints. This chapter proposes BDD-based symbolic model-checking algorithms to check whether a given iCGSf satisfies or not a given $ATLK_{irF}$ formula.

In opposition to the algorithms presented in Chapter 2 based on fixpoint computations, $ATL_{ir}$—and thus $ATLK_{irF}$—does not have the fixpoint characterization needed for such algorithms [JB11]. For instance, the following $CTL$ scheme is a valid scheme, that is, whatever $\phi$ is, the resulting formula is satisfied by any state of any Kripke structure:

$$\mathbf{EF}\ \phi \iff (\phi \vee \mathbf{EX}\ \mathbf{EF}\ \phi). \tag{5.1}$$

This valid scheme gives us a way to compute the set of states satisfying a formula $\mathbf{EF}\ \phi$: these states are the ones satisfying $\phi$ or from which there exists a successor satisfying $\mathbf{EF}\ \phi$. From this, we can derive the fixpoint-based algorithm of Section 2.2.3:

$$eval_{CTL}(S, \mathbf{EF}\ \phi) = \mu Q'.\ eval_{CTL}(S, \phi) \cup Pre(S, Q').$$

On the other hand, $ATLK_{irF}$ does not have similar valid schemes; in particular, the strategic scheme similar to the one of Equation 5.1,

$$\langle\!\langle \Gamma \rangle\!\rangle \mathbf{F}\ \phi \iff (\phi \vee \langle\!\langle \Gamma \rangle\!\rangle \mathbf{X}\ \langle\!\langle \Gamma \rangle\!\rangle \mathbf{F}\ \phi), \tag{5.2}$$

is not valid under $ATLK_{irF}$. Indeed, the top left state of the structure in Figure 5.1 satifies $\langle\langle ag \rangle\rangle \mathbf{F}\ p$ because choosing action 0 in all states will eventually lead to the bottom left state. On the other hand, the middle left state does not satisfy $\langle\langle ag \rangle\rangle \mathbf{F}\ p$ because a winning strategy should choose action 0 in the middle left state and action 1 in the middle right one, and violate the constraint of uniformity. Thus, while the top left state satisfies $\langle\langle ag \rangle\rangle \mathbf{F}\ p$, it does not satisfy $p \vee \langle\langle ag \rangle\rangle \mathbf{X}\ \langle\langle ag \rangle\rangle \mathbf{F}\ p$, showing that the scheme of Equation 5.2 is not valid.

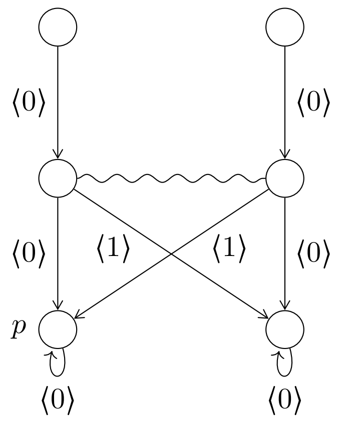


Figure 5.1: An iCGS with a single agent $ag$ in which the formula $\langle\langle ag \rangle\rangle \mathbf{F}\ p \iff (p \vee \langle\langle ag \rangle\rangle \mathbf{X}\ \langle\langle ag \rangle\rangle \mathbf{F}\ p)$ is not valid.

As model-checking algorithms for $ATLK_{irF}$ cannot use these kinds of fixpoint characterization, it is necessary to find other ways to solve the problem. This chapter presents several BDD-based model-checking algorithms. The main idea used to compute the set of states of an iCGSf $S$ satisfying a given strategic formula $\langle\langle \Gamma \rangle\rangle\ \psi$ is the following: (1) enumerate all uniform strategies for $\Gamma$ in $S$, represented with BDDs, and (2) for each of them, check whether the strategy is winning for $\psi$ or not.

The remainder of this chapter is structured as follows: Section 5.1 presents the fixpoint computations used in the algorithms to check if a strategy is winning; Section 5.2 explains how to enumerate the uniform strategies of a group of agents and describes a naive algorithm that checks all strategies to determine whether the formula is true. Section 5.3 presents the notion of partial strategies and how to use them to solve the model-checking problem; it also presents some optimizations to this partial strategies-based algorithm. Section 5.4 discusses how we can ignore some strategies that cannot be winning. Section 5.5 presents another approach that generates the winning strategies from the objective states. Finally Section 5.6 studies the complexity of the presented approaches, and Section 5.7 presents their implementation with PyNuSMV, as well as the modeling language designed for describing iCGSf.

Most functions and algorithms described in this chapter assume the existence of an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$ from which their arguments come.

## 5.1 Checking individual strategies

This section presents a set of fixpoint-based algorithms, called the $filter$ algorithms, for computing the set of states for which there exists a winning general strategy for a given objective. They can then be used to check whether a given uniform strategy is winning. This section presents the algorithms; their usage for checking strategies is presented in the next sections.

This section mainly speaks about *general* strategies. Thus, when speaking about *strategies*, we speak about general ones instead of uniform ones. This change of terminology is only applicable to this section, and the next ones will call *strategies* the uniform ones, and will speak about *general* strategies when needed.

The $filter$ algorithms are based on several functions. The first one is a modified version of $Pre_{[\![\Gamma]\!]}$ of Equation 2.2. This function is defined as

$$Pre_{[\![\Gamma]\!]}(Q', M_\Gamma) = \left\{ q' \in Q \;\middle|\; \begin{array}{l} \forall \langle q, a_\Gamma\rangle \in M_\Gamma, \\ q' = q \implies \exists a \in E_{Ag}(q) \\ \text{s.t. } a_\Gamma \sqsubseteq a \wedge \delta(q,a) \in Q' \end{array} \right\}.$$

Given a subset of states $Q' \subseteq Q$ and a set of $\Gamma$-moves $M_\Gamma$, $Pre_{[\![\Gamma]\!]}(Q', M_\Gamma)$ returns the subset of states $q'$ such that for all actions proposed by $M_\Gamma$ in $q'$, there exists a completing action leading to a state of $Q'$. In other words, it returns the subset of states $q'$ such that $\Gamma$ cannot prevent to reach $Q'$ by choosing an action proposed for $q'$ in $M_\Gamma$.

In the sequel, we say that a set $M_\Gamma$ of $\Gamma$-moves is *closed* if

$$\forall \langle q, a_\Gamma\rangle \in M_\Gamma, \forall a \in E_{Ag}(q), a_\Gamma \sqsubseteq a \implies \delta(q,a) \in M_\Gamma|_Q,$$

that is, if all states reachable from a move of $M_\Gamma$ also have a move in $M_\Gamma$. Furthermore, we say that a strategy $f_\Gamma$ for $\Gamma$ is *compatible with a set of* $\Gamma$*-moves* $M_\Gamma$ if

$$\forall q \in M_\Gamma|_Q, \langle q, f_\Gamma(q)\rangle \in M_\Gamma,$$

that is, if all the decisions made by $f_\Gamma$ in states of $M_\Gamma$ correspond to some moves in $M_\Gamma$. If $M_\Gamma$ is closed, then all paths enforced by a compatible strategy $f_\Gamma$ from a state of $M_\Gamma|_Q$ stay in states (and actions) of $M_\Gamma$. Indeed, for such a path to quit $M_\Gamma|_Q$, it must reach a state successor not in $M_\Gamma|_Q$ accessible through one of the actions of $M_\Gamma$, but this is not possible as $M_\Gamma$ is closed.

Finally, we say that $f_\Gamma$ *cannot avoid a path from a state $q \in Q$ satisfying $X$*, for a given path condition $X$, iff $\exists \pi \in out(f_\Gamma, q)$ such that $\pi$ satisfies $X$. For instance, a strategy $f_\Gamma$ cannot avoid a fair path from $q$ iff $\exists \pi \in out(f_\Gamma, q)$ such that $\pi$ is fair.

From the new $Pre_{[\![\Gamma]\!]}$ function, we can define a function derived from the $Reach_{[\![\Gamma]\!]}$ one of Equation 2.3:

$$Reach_{[\![\Gamma]\!]}(Q_1, Q_2, M_\Gamma) = \mu Q'.\ Q_2 \cup \left(Q_1 \cap Pre_{[\![\Gamma]\!]}(Q', M_\Gamma)\right).$$

Given two subsets of states $Q_1, Q_2 \subseteq Q$ and a closed subset of $\Gamma$-moves $M_\Gamma$, $Reach_{[\![\Gamma]\!]}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ returns the states of $M_\Gamma|_Q$ from which $\Gamma$ cannot avoid to reach a state of $Q_2$ through states of $Q_1$, by exclusively using actions in $M_\Gamma$.

From the two functions above, we can define the function

$$Fair_{[\![\Gamma]\!]}(M_\Gamma) = \nu Q'.\ \bigcap_{fc \in FC} Pre_{[\![\Gamma]\!]}\left(Reach_{[\![\Gamma]\!]}(Q, Q' \cap fc, M_\Gamma), M_\Gamma\right).$$

Given a closed set of $\Gamma$-moves $M_\Gamma$, $Fair_{[\![\Gamma]\!]}(M_\Gamma) \cap M_\Gamma|_Q$ returns the set of states of $M_\Gamma|_Q$ from which $\Gamma$ cannot avoid a *fair* path by using a strategy compatible with $M_\Gamma$.

Thanks to these functions, we can define three $filter$ algorithms. The first one is defined as

$$filter_{[\![\Gamma]\!]\mathbf{X}}(Q', M_\Gamma) = Pre_{[\![\Gamma]\!]}(Q' \cap Fair_{[\![\Gamma]\!]}(M_\Gamma), M_\Gamma).$$

Given a group of agents $\Gamma$, a subset of states $Q'$ and a closed set of $\Gamma$-moves $M_\Gamma$, $filter_{[\![\Gamma]\!]\mathbf{X}}(Q', M_\Gamma) \cap M_\Gamma|_Q$ returns the subset of states $q \in M_\Gamma|_Q$ such that all strategies compatible with $M_\Gamma$ cannot avoid a fair path starting in $q$ with its second state in $Q'$.

The second $filter$ algorithm is defined as

$$filter_{[\![\Gamma]\!]\mathbf{U}}(Q_1, Q_2, M_\Gamma) = Reach_{[\![\Gamma]\!]}(Q_1, Q_2 \cap Fair_{[\![\Gamma]\!]}(M_\Gamma), M_\Gamma).$$

Given a group of agents $\Gamma$, two subsets of states $Q_1$ and $Q_2$, and a closed set of $\Gamma$-moves $M_\Gamma$, $filter_{[\![\Gamma]\!]\mathbf{U}}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ returns the subset of states $q \in M_\Gamma|_Q$ such that all strategies compatible with $M_\Gamma$ cannot avoid a fair path starting in $q$ and reaching a state of $Q_2$ through states of $Q_1$.

Finally, the third $filter$ algorithm is defined as

$$\begin{aligned}&filter_{[\![\Gamma]\!]\mathbf{W}}(Q_1, Q_2, M_\Gamma) = \\ &\nu Q'.\ Q_{2,F} \cup (Q_1 \cap \\ &\bigcap_{fc \in FC} Pre_{[\![\Gamma]\!]}\left(Reach_{[\![\Gamma]\!]}(Q_1, Q_{2,F} \cup (Q' \cap fc), M_\Gamma\right), M_\Gamma)),\end{aligned}$$

where

$$Q_{2,F} = Q_2 \cap Fair_{[\![\Gamma]\!]}(M_\Gamma).$$

Given a group of agents $\Gamma$, two subsets of states $Q_1$ and $Q_2$, and a closed set of $\Gamma$-moves $M_\Gamma$, $filter_{[\![\Gamma]\!]\mathbf{W}}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ returns the subset of states $q \in M_\Gamma|_Q$ such that all strategies compatible with $M_\Gamma$ cannot avoid a fair path starting in $q$, that reaches a state of $Q_2$ through states of $Q_1$, or that stays in states of $Q_1$ forever.

Thanks to these $filter$ algorithms, it is possible to compute the states for which there exists a winning strategy compatible with a given subset of $\Gamma$-moves $M_\Gamma$. For instance, $filter_{[\![\Gamma]\!]\mathbf{X}}(Q', M_\Gamma) \cap M_\Gamma|_Q$ returns the set of states $q \in M_\Gamma|_Q$ such that all strategies compatible with $M_\Gamma$ cannot avoid a fair path starting in $q$ with second state in $Q'$. Thus, $M_\Gamma|_Q \backslash filter_{[\![\Gamma]\!]\mathbf{X}}(Q', M_\Gamma)$ is the set of states such that there exists a strategy compatible with $M_\Gamma$ that can avoid a fair path starting in $q$ with second state in $Q'$.

It is possible to define the algorithms corresponding to

$$\begin{aligned}&M_\Gamma|_Q \backslash filter_{[\![\Gamma]\!]\mathbf{X}}(Q', M_\Gamma),\\ &M_\Gamma|_Q \backslash filter_{[\![\Gamma]\!]\mathbf{U}}(Q_1, Q_2, M_\Gamma),\\ &M_\Gamma|_Q \backslash filter_{[\![\Gamma]\!]\mathbf{W}}(Q_1, Q_2, M_\Gamma),\end{aligned}$$

by using the duality of the coalition modalities: $\langle\!\langle\Gamma\rangle\!\rangle\ \psi = \neg[\![\Gamma]\!]\ \neg\psi$. Indeed, let

$$\begin{aligned}Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(Q', M_\Gamma) &= \overline{Pre_{[\![\Gamma]\!]}(\overline{Q'}, M_\Gamma)}\\ &= \left\{ q' \in Q \;\middle|\; \begin{array}{l}\exists\langle q, a_\Gamma\rangle \in M_\Gamma \text{ s.t.}\\ q' = q \wedge \forall a \in E_{Ag}(q),\\ a_\Gamma \sqsubseteq a \implies \delta(q,a) \in Q'\end{array} \right\}. \qquad (5.3)\end{aligned}$$

Intuitively, $Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(Q', M_\Gamma)$ returns the states $q \in M_\Gamma|_Q$ such that there exists an action for $q$ in $M_\Gamma$ that surely leads to a state of $Q'$ in one step.

Also, let

$$Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(Q_1, Q_2, M_\Gamma) = \nu Q'.\ Q_2 \cup \left(Q_1 \cap Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(Q', M_\Gamma)\right).$$

Intuitively, $Stay_{\langle\langle\Gamma\rangle\rangle}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ is the states $q \in M_\Gamma|_Q$ such that there exists a strategy compatible with $f_\Gamma$ that forces, from $q$, to reach $Q_2$ through $Q_1$, or to stay in $Q_1$ forever.

Finally, let

$$\begin{aligned} NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma) &= \overline{Fair_{\llbracket\Gamma\rrbracket}(M_\Gamma)} \\ &= \mu Q'. \bigcup_{fc \in FC} Pre_{\langle\langle\Gamma\rangle\rangle}\left(Stay_{\langle\langle\Gamma\rangle\rangle}(Q' \cup \overline{fc}, \varnothing, M_\Gamma), M_\Gamma\right). \end{aligned}$$

Intuitively, $NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma)$ is the set of states $q \in M_\Gamma|_Q$ such that there exists a strategy compatible with $M_\Gamma$ that forces unfair paths from $q$.

Thanks to these three functions, we can define the dual algorithms of the previous *filter* ones. First, let

$$\begin{aligned} filter_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(Q', M_\Gamma) &= Q \backslash filter_{\llbracket\Gamma\rrbracket\mathbf{X}}(\overline{Q'}, M_\Gamma) \\ &= Pre_{\langle\langle\Gamma\rangle\rangle}(Q' \cup NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma), M_\Gamma). \end{aligned}$$

$filter_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(Q', M_\Gamma) \cap M_\Gamma|_Q$ computes the set of states $q \in M_\Gamma|_Q$ such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ such that all fair paths enforced by $f_\Gamma$ from $q$ have their second state in $Q'$.

Second, let

$$\begin{aligned} filter_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(Q_1, Q_2, M_\Gamma) &= Q \backslash filter_{\llbracket\Gamma\rrbracket\mathbf{W}}(\overline{Q_2}, \overline{Q_1} \cap \overline{Q_2}, M_\Gamma) \\ &= \begin{array}{l} \mu Q'.\ Q_{1,2,N} \cap \\ (Q_2 \cup \bigcup\limits_{fc \in FC} Pre_{\langle\langle\Gamma\rangle\rangle}(Stay_{\langle\langle\Gamma\rangle\rangle}( \begin{array}{c} Q_{1,2,N} \cap (Q' \cup \overline{fc}), \\ Q_2 \cap (Q' \cup \overline{fc}), M_\Gamma \end{array} ), M_\Gamma)), \end{array} \end{aligned}$$

where

$$Q_{1,2,N} = Q_1 \cup Q_2 \cup NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma).$$

$filter_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ computes the set of states $q \in M_\Gamma|_Q$ such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ such that all fair paths enforced by $f_\Gamma$ from $q$ reach a state of $Q_2$ through states of $Q_1$.

Third, let

$$\begin{aligned} filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, M_\Gamma) &= Q \backslash filter_{\llbracket\Gamma\rrbracket\mathbf{U}}(\overline{Q_2}, \overline{Q_1} \cap \overline{Q_2}, M_\Gamma) \\ &= Stay_{\langle\langle\Gamma\rangle\rangle}(Q_1 \cup Q_2 \cup NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma), Q_2, M_\Gamma). \end{aligned}$$

$filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ computes the set of states $q \in M_\Gamma|_Q$ such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ such that all fair paths from $q$ enforced by $f_\Gamma$ from $q$ reach $Q_2$ through $Q_1$, or stay in $Q_1$ forever.

Finally, the fixpoint computations presented in this section can be used to compute the states for which a given strategy is winning. If $M_\Gamma$ represents a single strategy—that is, if there exists only one move for each state in $M_\Gamma$—then the last three $filter$ algorithms compute the states for which the strategy is winning. These algorithms will be used to perform model checking of $ATLK_{irF}$ formulas, and this usage is described in the following sections. The correctness of the $filter_{\langle\langle\Gamma\rangle\rangle}$ algorithms is proved in Section A.1 of Appendix A.

## 5.2 Enumerating all strategies

Thanks to the fixpoint-based algorithms presented in the previous section, it is possible to check whether a given strategy is winning for a given objective. They can then be used to verify whether a given state satisfies an $ATLK_{irF}$ strategic formula $\langle\langle\Gamma\rangle\rangle\ \psi$. The naive approach is to simply enumerate all uniform strategies of the given structure, compute the states for which each strategy is winning, and return the set of states for which there exists a winning uniform strategy for all indistinguishable states. This section presents the *Split* algorithm to generate all uniform strategies of a given structure, and describes such a naive model-checking algorithm for $ATLK_{irF}$ strategic formulas.

To generate all uniform strategies of a given structure, the *Split* algorithm presented in Algorithm 5.1 can be used. It takes a subset of agents $\Gamma \subseteq Ag$ and a subset of $\Gamma$-moves $M_\Gamma$ as arguments and returns the set of all the largest subsets of non-$\Gamma$-conflicting moves of $M_\Gamma$. We say that two $\Gamma$-moves $\langle q, a_\Gamma\rangle$ and $\langle q', a'_\Gamma\rangle$ are $\Gamma'$-*conflicting*, where $\Gamma' \subseteq \Gamma$, if

$$\exists ag \in \Gamma' \text{ s.t. } q \sim_{ag} q' \text{ and } a_\Gamma(ag) \neq a'_\Gamma(ag).$$

In other words, $\langle q, a_\Gamma\rangle$ and $\langle q', a'_\Gamma\rangle$ are $\Gamma'$-conflicting if the states are indistinguishable for some agent $ag \in \Gamma'$ and the proposed actions for $ag$ are different. We write $ag$-conflicting instead of $\{ag\}$-conflicting. Furthermore, we say that a set of $\Gamma$-moves $M_\Gamma$ is $\Gamma'$-conflicting if there exist two $\Gamma'$-conflicting moves in $M_\Gamma$.

If $M_\Gamma$ is non-$\Gamma$-conflicting, then it represents (a part of) a uniform strategy. Indeed, $M_\Gamma$ proposes joint actions for $\Gamma$ such that, for any agent $ag \in \Gamma$, for two states indistinguishable by $ag$, $M_\Gamma$ gives the same action for $ag$.

Algorithm 5.1 computes the set of largest subsets of non-$\Gamma$-conflicting moves of $M_\Gamma$ by using Algorithm 5.2 to split subsets of moves into subsets that are not conflicting for a given agent. By recursively splitting all non-conflicting subsets for each agent, *Split* is able to compute non-$\Gamma$-conflicting subsets of moves.

**Algorithm 5.1:** *Split*$(\Gamma, M_\Gamma)$

```
Data: $\Gamma \subseteq Ag$ a group of agents, $M_\Gamma \subseteq E_\Gamma$ a set of moves.
Result: The set of largest subsets of non-$\Gamma$-conflicting moves of $M_\Gamma$.
$subsets = \{M_\Gamma\}$
for $ag \in \Gamma$ do
    $subsets' = \{\}$
    for $subset \in subsets$ do
        $subsets' = subsets' \cup SplitAgent(ag, \Gamma, subset)$
    $subsets = subsets'$
return $subsets$
```

To split the set of moves $M_\Gamma$ into non-conflicting subsets for the agent $ag$, Algorithm 5.2 first gets all the conflicting moves of $M_\Gamma$. If there are no such conflicts, $M_\Gamma$ is its own largest non-$ag$-conflicting subset. Otherwise, *SplitAgent* picks one set of conflicting moves *equivalent*, gets all the possible actions *actions* in this set and, for each of these actions $a_{ag}$, creates a new non-conflicting subset by computing the cross product of the moves playing $a_{ag}$ in the conflicting set, and the subsets *subsubset* for other conflicting sets recursively computed by *SplitAgent*. In other words, *Split* iteratively splits conflicting equivalence classes and computes the cross product of all the splittings to build all possible non-$ag$-conflicting subset of $M_\Gamma$.

Finally, Algorithm 5.3 uses the *Split* algorithm and the $filter$ ones to perform the model checking of $ATLK_{irF}$ strategic formulas. More precisely, when handling a strategic formula, $eval_{ATLK_{irF}}$ iterates over each uniform strategy $f_\Gamma$ generated by $Split(\Gamma, E_\Gamma)$, computes the set of states for which the strategy is winning thanks to the $filter$ algorithms, and keeps only the ones such that the strategy is winning in all indistinguishable states. At the end, all strategies are checked, and all states for which there exists a winning uniform strategy for all indistinguishable states are returned. In the rest of the thesis, this approach is called the *naive* approach.

Algorithm 5.3 only presents the strategic cases. The model checking for the propositional cases (*true*, $p$, $\neg$ and $\vee$), the temporal ones (**EX**, **EU** and **EW**) and the knowledge ones (**K**, **E**, **D** and **C**) is performed in the standard way, as exposed in Section 2.2.3. The correctness of the $eval_{ATLK_{irF}}$ algorithm is proved in Section A.2 of Appendix A.

**Algorithm 5.2:** *SplitAgent*$(ag, \Gamma, M_\Gamma)$

```
**Data**: $ag \in \Gamma$ an agent of $\Gamma$, $\Gamma \subseteq Ag$ a group of agents, $M_\Gamma \subseteq E_\Gamma$ a set of $\Gamma$-moves.
**Result**: The set of largest subsets of non-$ag$-conflicting moves of $M_\Gamma$.
$conflicting = \left\{ \langle q, a_\Gamma \rangle \in M_\Gamma \middle| \begin{array}{l} \exists \langle q', a'_\Gamma \rangle \in M_\Gamma \text{ s.t.} \\ q' \sim_{ag} q \wedge a_\Gamma(ag) \neq a'_\Gamma(ag) \end{array} \right\}$
if $conflicting = \varnothing$ then return $\{M_\Gamma\}$
else
    $\langle q, a_\Gamma \rangle$ = pick one element in $conflicting$
    $equivalent = \{\langle q', a'_\Gamma \rangle \in M_\Gamma \mid q' \sim_{ag} q\}$
    $actions = \{a_{ag} \in Act \mid \exists \langle q', a'_\Gamma \rangle \in equivalent \text{ s.t. } a'_\Gamma(ag) = a_{ag}\}$
    $ncsubsets = SplitAgent(ag, \Gamma, M_\Gamma \backslash equivalent)$
    $subsets = \{\}$
    for $a_{ag} \in actions$ do
        $equivsubset = \{\langle q', a'_\Gamma \rangle \in equivalent \mid a'_\Gamma(ag) = a_{ag}\}$
        $subsets = subsets \cup \left\{ \begin{array}{l} equivsubset \cup ncsubset \mid \\ ncsubset \in ncsubsets \end{array} \right\}$
    return $subsets$
```

## 5.3 Partial strategies

The previous section described a naive algorithm to perform the model checking of $ATLK_{irF}$ strategic formulas $\langle\!\langle \Gamma \rangle\!\rangle\ \psi$. It generates all possible uniform strategies for $\Gamma$ and computes, for each of them, the states for which the strategy is winning. In many cases, this naive algorithm checks more strategies than necessary. For instance, let us consider a variant of the three-cards game. In this variant, the player can cheat and rearrange the cards before the dealer gives them. If he cheats, he gets the $K$ and the dealer the $Q$, and the player just has to keep his card to win the game. The corresponding iCGSf is given in Figure 5.2. In this structure, a uniform strategy for the player chooses whether the player cheats or not in the initial state, and whether he changes his card or not afterwards. There are thus $2^5 = 32$ uniform strategies:

1. two choices in the initial state: cheating or not;
2. two choices when he has cheated: swapping or not;
3. two choices when he has the $A$;

**Algorithm 5.3:** $eval_{ATLK_{irF}}(S, \phi)$

```
Data: $S$ an iCGSf, $\phi$ an $ATLK_{irF}$ formula.
Result: The states of $S$ satisfying $\phi$.
case $\phi \in \{\langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi', \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2], \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]\}$
    $sat = \{\}$
    for $f_\Gamma \in Split(\Gamma, E_\Gamma)$ do
        case $\phi = \langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi'$
            $Q' = eval_{ATLK_{irF}}(S, \phi')$
            $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(Q', f_\Gamma)$
        case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2]$
            $Q_1 = eval_{ATLK_{irF}}(S, \phi_1)$
            $Q_2 = eval_{ATLK_{irF}}(S, \phi_2)$
            $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(Q_1, Q_2, f_\Gamma)$
        case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]$
            $Q_1 = eval_{ATLK_{irF}}(S, \phi_1)$
            $Q_2 = eval_{ATLK_{irF}}(S, \phi_2)$
            $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, f_\Gamma)$
        $sat = sat \cup \{q \in winning \mid \forall ag \in \Gamma, \forall q' \sim_{ag} q, q' \in winning\}$
    return $sat$
case $\phi = ...$ // Cases for the other operators are standard
```

4. two choices when he has the $K$;

5. two choices when he has the $Q$.

Furthermore, the formula $\langle\langle player\rangle\rangle[\neg player\ loses\ \mathbf{U}\ player\ wins]$ is true in the initial state because the player can simply cheat and automatically win the current game by keeping his card. But if the player chooses to cheat in the initial state, his choices in the other case are not relevant anymore for the outcomes of the strategy. Indeed, if his strategy is to cheat, the executions will never reach a state in which he has not cheated, and thus the choices he makes in these states do not impact the outcomes of his strategy. In fact, there are only 10 relevant uniform strategies:

1. two when he cheats: keeping his card or not;

2. eight when he does not cheat: whether keeping his card when he has the $A$ or not, when he has the $K$, and when he has the $Q$.

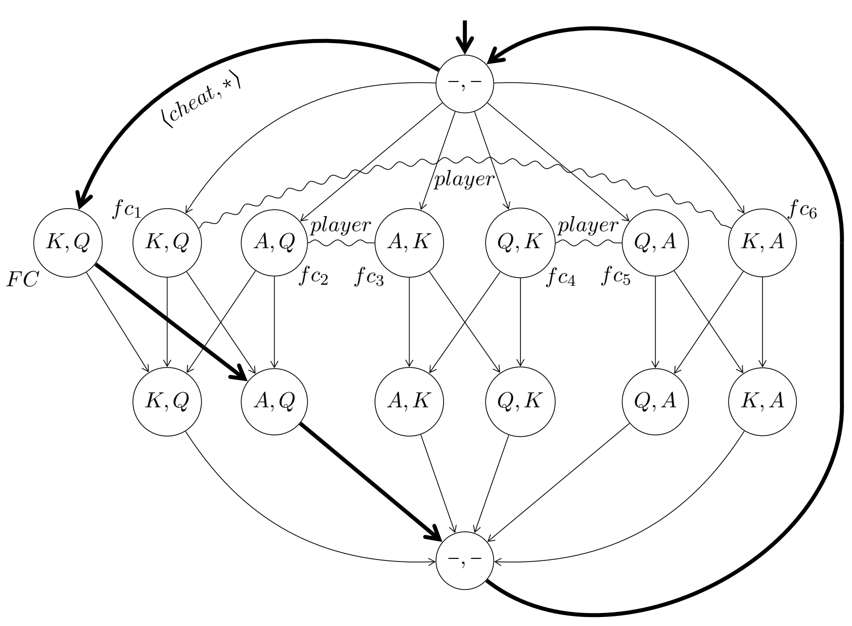


Figure 5.2: The graph of the card game with a cheating player. The $\langle cheat, * \rangle$ labelled transition means that the player cheats and the dealer chooses any action. The bold arrows show a partial strategy.

The characteristics of these strategies are that they do not make a choice in states that are not reachable through the strategy itself, while being sufficient to decide whether the player has a strategy to win the game in the initial state. In the sequel, we call these strategies *partial* strategies because they do not propose an action for all states of the structure.

This section describes an algorithm to perform the model checking of $ATLK_{irF}$ strategic formulas that exploits the idea of partial strategies and the fact that it is enough to check the existence of a winning partial uniform strategy to conclude whether there exists a winning (complete) uniform strategy. This section first formally defines partial strategies, then shows how to generate them, describes a model-checking algorithm, and presents some optimizations for increasing its performance in practice.

A *partial* strategy for agent *ag* is a partial function $f_{ag} : Q \nrightarrow Act$ such that $\forall q \in dom(f_{ag}), f_{ag}(q) \in E_{ag}(q)$. That is, a partial strategy is a strategy that gives a choice for *ag* for *some* states of the model, instead of for *all* states. A partial strategy $f_\Gamma$ for a group of agents $\Gamma \subseteq Ag$ is a tuple of partial strategies, one for each agent of the group, such that all individual strategies share the same domain—that is, are defined for the same subset of states—and

$$\forall q \in dom(f_\Gamma), \forall a \in Act^{Ag}, f_\Gamma(q) \sqsubseteq a \implies \delta(q,a) \in dom(f_\Gamma).$$

That is, a partial strategy for $\Gamma$ gives a choice for any state reachable through a state it is defined for. A partial strategy $f_\Gamma$ for a group $\Gamma$ can thus be represented by a set of $\Gamma$-moves since in each state of the system, either the strategy gives an action for all agents of $\Gamma$, or it gives no action for anyone. The set of $\Gamma$-moves corresponding to a partial strategy $f_\Gamma$ is closed, by definition of these strategies.

Furthermore, let $Q' \subseteq Q$ be a set of states, we say that a partial strategy $f_\Gamma$ is *adequate* for $Q'$ if $Q' \subseteq dom(f_\Gamma)$, that is, $f_\Gamma$ defines an action for all states of $Q'$, and thus for all states reachable from $Q'$ through $f_\Gamma$. A partial strategy $f_\Gamma$ is thus adequate for all states of $dom(f_\Gamma)$. Also, a partial strategy $f_\Gamma$ is uniform if it gives the same action for an agent $ag \in \Gamma$ in states indistinguishable by $ag$, that is, if

$$\forall ag \in \Gamma, \forall q, q' \in dom(f_\Gamma), q \sim_{ag} q' \implies f_\Gamma(q)(ag) = f_\Gamma(q')(ag).$$

Finally, we say that a partial strategy $f_\Gamma$ *extends* a set of $\Gamma$-moves $M_\Gamma$ if

$$\forall q' \in M_\Gamma|_Q, \exists \langle q, a_\Gamma \rangle \in M_\Gamma \text{ s.t. } , q' = q \text{ and } f_\Gamma(q) = a_\Gamma,$$

that is, for all states $q$ for which $M_\Gamma$ proposes an action, $f_\Gamma$ proposes an action from the possible actions defined by $M_\Gamma$ in $q$. The outcomes of a partial strategy is only defined for the state it is adequate for. Given a partial strategy $f_\Gamma$ and a state $q \in dom(f_\Gamma)$, the outcomes of $f_\Gamma$ from $q$ is defined as

$$out(f_\Gamma, q) = \{\pi = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \mid q_0 = q \wedge \forall d \in \mathbb{N}, f_\Gamma(q_d) \sqsubseteq a_{d+1}\}.$$

This set of outcomes is well-defined as, if $q \in dom(f_\Gamma)$, $f_\Gamma$ is defined for all states reachable from $q$ through $f_\Gamma$ itself.

When determining whether a given state $q \in Q$ satisfies a strategic formula $\langle\!\langle \Gamma \rangle\!\rangle\, \psi$, it is enough to look for winning uniform partial strategies instead of complete ones. Indeed, there exists a winning uniform strategy for $\psi$ in $q$ if there exists a partial one. This property is captured by the following theorem.

**Theorem 5.1.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC \rangle$, a state $q \in Q$, a set of agents $\Gamma \subseteq Ag$, and an $ATLK_{irF}$ path formula $\psi$, there exists a uniform strategy $f_\Gamma$ such that*

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \models \psi,$$

*if and only if there exists a partial uniform strategy $f'_\Gamma$ adequate for $[q]^E_\Gamma$ such that*

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f'_\Gamma, q'), \pi \models \psi.$$

*Proof.* Proving the left-to-right part is trivial: a uniform strategy $f_\Gamma$ is a partial uniform strategy adequate for $[q]^E_\Gamma$ as it is adequate for $Q$.

For the right-to-left part, let us assume the existence of a partial uniform strategy $f'_\Gamma$ adequate for $[q]^E_\Gamma$. Let $f_\Gamma$ be a (complete) uniform strategy such that

$$\forall q \in dom(f'_\Gamma), f_\Gamma(q) = f'_\Gamma(q).$$

That is, $f_\Gamma$ makes the same choices as $f'_\Gamma$ in states for which $f'_\Gamma$ is defined, and chooses any enabled action in the other states. The outcomes of $f_\Gamma$ from $q' \in [q]^E_\Gamma$ are the same as the outcomes of $f'_\Gamma$ from the same states. Indeed, let assume that $q' \in [q]^E_\Gamma$ and $out(f_\Gamma, q') \neq out(f'_\Gamma, q)$. Thus there exists a path $\pi = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} ...$ such that $\pi \in out(f_\Gamma, q') \backslash out(f'_\Gamma, q')$, or such that $\pi \in out(f'_\Gamma, q') \backslash out(f_\Gamma, q')$.

Let us suppose that $\pi \in out(f_\Gamma, q') \backslash out(f'_\Gamma, q')$. In this case, there exists a position $d \in \mathbb{N}$ such that $f_\Gamma(q_d) \neq f'_\Gamma(q_d)$, otherwise $\pi$ would be in $out(f'_\Gamma, q')$, too. Let $d$ be the first position such that $f_\Gamma(q_d) \neq f'_\Gamma(q_d)$. If $f_\Gamma(q_d) \neq f'_\Gamma(q_d)$, then $q_d \notin dom(f'_\Gamma)$; indeed, if $q_d \in dom(f'_\Gamma)$, then $f'_\Gamma(q_d) = f_\Gamma(q_d)$, by definition of $f_\Gamma$. But, as $f'_\Gamma$ is a partial strategy, if $q_d \notin dom(f'_\Gamma)$, then $q_{d-1} \notin dom(f'_\Gamma)$. Thus $d$ is not the first position such that $f_\Gamma(q_d) \neq f'_\Gamma(q_d)$, and we have a contradiction. So there is no such $d$, and $\pi$ belongs to $out(f'_\Gamma, q')$, too. The other case, where $\pi \in out(f'_\Gamma, q') \backslash out(f_\Gamma, q')$ is similar, and the proof is done since there cannot be any path $\pi$ in the outcomes of $f'_\Gamma$ but not in the ones of $f_\Gamma$, or vice versa. □

Checking the existence of partial strategies instead of complete ones is correct for determining whether some state satisfies a given $ATLK_{irF}$ strategic formula. We can thus improve the algorithms of the previous section to take partial strategies into account. Given a set of agents $\Gamma$, and a set of non-$\Gamma$-conflicting $\Gamma$-moves $M_\Gamma$, *ReachSplit*$(\Gamma, M_\Gamma)$, given in Algorithm 5.4, returns the set of uniform partial strategies extending $M_\Gamma$. All these partial strategies are adequate for $M_\Gamma|_Q$ as they extend $M_\Gamma$.

Algorithm 5.4 depends on several functions. First, it depends on the *Split* algorithm presented in the previous section. Furthermore, it depends on the functions *Post* and *Compatible*. The first one takes a set of states $Q' \subseteq Q$ and a set of $\Gamma$-moves $M_\Gamma$ as arguments and returns the successor states of a state of $Q'$ through an action given in $M_\Gamma$. More formally,

$$Post(Q', M_\Gamma) = \left\{ q \in Q \,\middle|\, \begin{array}{l} \exists q' \in Q', \exists \langle q', a'_\Gamma \rangle \in M_\Gamma, \exists a' \in E_{Ag}(q') \\ \text{s.t. } a'_\Gamma \sqsubseteq a' \wedge \delta(q', a') = q \end{array} \right\}.$$

*Compatible* takes a set of states $Q' \subseteq Q$ and a set of non-$\Gamma$-conflicting $\Gamma$-moves $M_\Gamma \subseteq E_\Gamma$ as arguments and returns the moves of $E_\Gamma$ such that there exists no conflicting move in $M_\Gamma$. More formally,

$$Compatible(Q', M_\Gamma) = \left\{ \langle q, a_\Gamma \rangle \in E_\Gamma \,\middle|\, \begin{array}{l} q \in Q' \wedge \nexists \langle q', a' \rangle \in M_\Gamma, ag \in \Gamma \\ \text{s.t. } q \sim_{ag} q' \wedge a_\Gamma(ag) \neq a'_\Gamma(ag) \end{array} \right\}.$$

Based on these functions, Algorithm 5.4 computes the smallest uniform partial strategies extending a given set of non-$\Gamma$-conflicting $\Gamma$-moves. First it computes the set *new* of states reachable, in one step, from $M_\Gamma$ that are not in $M_\Gamma$ yet. If *new* is empty, this means that all states reachable from $M_\Gamma$ are already in $M_\Gamma$, and thus $M_\Gamma$ is already a uniform partial strategy. Otherwise, we can extend $M_\Gamma$ with different choices for states of *new*. For this, *ReachSplit* computes the moves of *new* compatible with $M_\Gamma$ and splits these moves to get uniform choices. Then, *ReachSplit* extends $M_\Gamma$ with each largest non-$\Gamma$-conflicting subset of *compatible* and recursively generates all uniform partial strategies extending the extension.

**Algorithm 5.4:** *ReachSplit*$(\Gamma, M_\Gamma)$

```
Data: $\Gamma$ a subset of agents, $M_\Gamma \subseteq E_\Gamma$ a set of non-$\Gamma$-conflicting
      $\Gamma$-moves.
Result: The set of smallest uniform partial strategies extending
        $M_\Gamma$.
1 $new = Post(Q, M_\Gamma) \backslash M_\Gamma|_Q$
  if $new = \varnothing$ then return $\{M_\Gamma\}$
  else
4     $compatible = Compatible(new, M_\Gamma)$
5     $newstrats = Split(\Gamma, compatible)$
      $strats = \{\}$
7     for $M'_\Gamma \in newstrats$ do
          $strats = strats \cup ReachSplit(\Gamma, M_\Gamma \cup M'_\Gamma)$
      return $strats$
```

Thanks to this algorithm, we can generate all partial strategies adequate for a set of states $Q'$. First, let

$$Moves_\Gamma(Q') = \{\langle q', a'_\Gamma \rangle \in E_\Gamma \mid q' \in Q'\}$$

be the set of Γ-moves enabled in states of $Q'$. The function

$$PartialStrats(\Gamma, Q') = \bigcup \left\{ \begin{array}{l} ReachSplit(\Gamma, M_\Gamma) \mid \\ M_\Gamma \in Split(\Gamma, Moves_\Gamma(Q')) \end{array} \right\}$$

returns the set of all smallest uniform partial strategies for Γ that are adequate for $Q'$.

From the functions and algorithms above, we can design a model-checking algorithm for $ATLK_{irF}$ strategic formulas. Precisely, Algorithm 5.5 accumulates in *sat* the states of $Q'$ for which there exists a winning uniform partial strategy by iterating over all these strategies and computing the states for which the strategy is winning. First it computes the set of uniform partial strategies adequate for $[Q']^E_\Gamma$. Then, for each such strategy $f_\Gamma$, it computes the states *winning* for which $f_\Gamma$ wins the strategic objective thanks to the corresponding $filter$ algorithm. Finally, it accumulates in *sat* the states of $Q'$ for which all indistinguishable states are in *winning*. In the rest of the thesis, this approach is called the *partial* approach.

As for $eval_{ATLK_{irF}}$, Algorithm 5.5 only presents the strategic cases. The model checking for the propositional cases (*true*, $p$, $\neg$ and $\vee$), the temporal ones (**EX**, **EU** and **EW**) and the knowledge ones (**K**, **E**, **D** and **C**) is performed in the standard way, as exposed in Section 2.2.3.

Algorithm 5.5 is correct, in the sense that it effectively returns the states of $Q'$ satisfying $\phi$. Its correctness is proved in Section A.3 of Appendix A.

### 5.3.1 Optimizations

[BPQR14] showed through practical experiments that $eval^{Partial}_{ATLK_{irF}}$ can increase the performance of model checking compared to the naive approach. Nevertheless, it is still possible to increase the performances of the former through other practical optimizations.

The first optimization is *caching*. Algorithm 5.5 needs to compute, for each uniform partial strategy $f_\Gamma$, the states that satisfy the sub-formula(s) of $\phi$. There can be a large number of such strategies, and these strategies can share a lot of common states. Thus, the algorithm potentially evaluates many times the same formula in the same states. When the sub-formula is an atomic proposition, the effort is not high, but when it is a strategic formula, the effort to evaluate the formula once is already very high.

Caching can then be used to avoid re-evaluating the same formula in the same states several times. More precisely, given a set of states $Q'$

**Algorithm 5.5:** $eval^{Partial}_{ATLK_{irF}}(S, Q', \phi)$

```
Data: $S$ an iCGSf, $Q' \subseteq Q$ a subset of states, $\phi$ an $ATLK_{irF}$
      formula.
Result: The states of $Q'$ satisfying $\phi$.
case $\phi \in \{\langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi', \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2], \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]\}$
    $sat = \{\}$
3   for $f_\Gamma \in PartialStrats(\Gamma, [Q']^E_\Gamma)$ do
4       case $\phi = \langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi'$
            $Q'' = eval^{Partial}_{ATLK_{irF}}(S, Post([Q']^E_\Gamma, f_\Gamma), \phi')$
6           $winning = filter_{\langle\langle\Gamma\rangle\rangle \mathbf{X}}(Q'', f_\Gamma)$
        case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2]$
            $Q_1 = eval^{Partial}_{ATLK_{irF}}(S, dom(f_\Gamma), \phi_1)$
            $Q_2 = eval^{Partial}_{ATLK_{irF}}(S, dom(f_\Gamma), \phi_2)$
10          $winning = filter_{\langle\langle\Gamma\rangle\rangle \mathbf{U}}(Q_1, Q_2, f_\Gamma)$
        case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]$
            $Q_1 = eval^{Partial}_{ATLK_{irF}}(S, dom(f_\Gamma), \phi_1)$
            $Q_2 = eval^{Partial}_{ATLK_{irF}}(S, dom(f_\Gamma), \phi_2)$
14          $winning = filter_{\langle\langle\Gamma\rangle\rangle \mathbf{W}}(Q_1, Q_2, f_\Gamma)$
15      $sat = sat \cup \left\{ q \in winning \cap Q' \middle| \begin{array}{l} \forall ag \in \Gamma, \forall q' \sim_{ag} q, \\ q' \in winning \end{array} \right\}$
    return $sat$
case $\phi = ...$ // Cases for the other operators are standard
```

and a formula $\phi$, it is not necessary to evaluate $\phi$ in the states of $Q'$ for which we already know whether $\phi$ is satisfied or not.

Algorithm 5.6 implements this idea. It assumes the existence of two caches, $cache_{sat}$ and $cache_{unsat}$, that it can use to store and retrieve data. It also assumes that, for each $\langle S, \phi\rangle$ they cover, $cache_{sat}[\langle S, \phi\rangle]$ is a subset of the states of $S$ that satisfy $\phi$, and $cache_{unsat}[\langle S, \phi\rangle]$ is a subset of the states of $S$ that satisfy $\neg\phi$.

First, Algorithm 5.6 gets in *sat* and *unsat* the set of states satisfying (resp. violating) $\phi$ stored in the caches. Then, it computes the set *unknown* of states of $Q'$ for which we do not know the truth value of $\phi$ yet. Using the $eval^{Partial}_{ATLK_{irF}}$ algorithm, it computes the subset *newsat* of states of *unknown* satisfying $\phi$, and updates the two caches to add *newsat* to the states satisfying $\phi$ (in $cache_{sat}$), and the others to the states violating $\phi$ (in $cache_{unsat}$). Finally, it returns the subset of $Q'$

by which it knows the formula is satisfied. $eval^{Partial}_{ATLK_{irF}}$ must then call $eval^{Cached}_{ATLK_{irF}}$ instead of itself when evaluating the sub-formula(s) of $\phi$.

**Algorithm 5.6:** $eval^{Cached}_{ATLK_{irF}}(S, Q', \phi)$

```
Data: $S$ an iCGSf, $Q' \subseteq Q$ a subset of states, $\phi$ an $ATLK_{irF}$
      formula.
Result: The states of $Q'$ satisfying $\phi$.
$sat = cache_{sat}[\langle S, \phi \rangle]$
$unsat = cache_{unsat}[\langle S, \phi \rangle]$
$unknown = Q' \backslash (sat \cup unsat)$
$newsat = eval^{Partial}_{ATLK_{irF}}(S, unknown, \phi)$
$cache_{sat}[\langle S, \phi \rangle] = sat \cup newsat$
$cache_{unsat}[\langle S, \phi \rangle] = unsat \cup (unknown \backslash newsat)$
return $(sat \cup newsat) \cap Q'$
```

A second optimization is called *early termination*. When evaluating a strategic formula $\langle\langle \Gamma \rangle\rangle\ \psi$ in a set of states $Q'$, we can stop looking for winning strategies as soon as we found a winning one for each state of $Q'$.

Algorithm 5.7 implements this idea. The only difference between this algorithm and the $eval^{Partial}_{ATLK_{irF}}$ one is the **if** statement at Line 4. Before a strategy $f_\Gamma$ is evaluated, this algorithm checks whether all states of interest in $Q'$ already satisfy the formula. If it is the case, the algorithm can safely stop and return that all states satisfy the formula. Otherwise it has to check other strategies. Note that if some state in $Q'$ does not satisfy $\phi$, then the algorithm has to check all uniform partial strategies returned by *PartialStrats* before concluding, and the early termination is never triggered.

[BPQR14] proposed other optimizations related to early termination. The technique above is quite simple: we stop searching for strategies when we found a winning one for each state of interest. Following this idea, we can reconsider smaller strategies when *sat* grows. Indeed, whenever we find a strategy in $PartialStrats(\Gamma, [Q']^E_\Gamma)$ that is winning for some states added in *sat*, we can recompute the smaller strategies reachable from $[Q']^E_\Gamma \backslash sat$, ignoring the part of these strategies taking *sat* states into account. This can be done by recomputing a new set of strategies whenever *sat* grows. We can also perform fewer re-computations of the strategies by recomputing them when the number of states of $[Q']^E_\Gamma$ that are not in *sat* decreases under a certain threshold.

The main drawback of these two approaches is that parts of some

**Algorithm 5.7:** $eval^{Term}_{ATLK_{irF}}(S, Q', \phi)$

```
Data: $S$ an iCGSf, $Q' \subseteq Q$ a subset of states, $\phi$ an $ATLK_{irF}$ formula.
Result: The states of $Q'$ satisfying $\phi$.
case $\phi \in \{\langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi', \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2], \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]\}$
    $sat = \{\}$
    for $f_\Gamma \in PartialStrats(\Gamma, [Q']^E_\Gamma)$ do
4       if $sat = Q'$ then
            return $sat$
        case $\psi = \boldsymbol{X}\ \phi'$
            $Q'' = eval^{Partial}_{ATLK_{irF}}(S, Post([Q']^E_\Gamma, f_\Gamma), \phi')$
            $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(Q'', f_\Gamma)$
        case $\psi = \phi_1\ \boldsymbol{U}\ \phi_2$
            $Q_1 = eval^{Partial}_{ATLK_{irF}}(S, dom(f_\Gamma), \phi_1)$
            $Q_2 = eval^{Partial}_{ATLK_{irF}}(S, dom(f_\Gamma), \phi_2)$
            $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(Q_1, Q_2, f_\Gamma)$
        case $\psi = \phi_1\ \boldsymbol{W}\ \phi_2$
            $Q_1 = eval^{Partial}_{ATLK_{irF}}(S, dom(f_\Gamma), \phi_1)$
            $Q_2 = eval^{Partial}_{ATLK_{irF}}(S, dom(f_\Gamma), \phi_2)$
            $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, f_\Gamma)$
        $sat = sat \cup \left\{ q \in winning \cap Q' \,\middle|\, \begin{array}{l} \forall ag \in \Gamma, \forall q' \sim_{ag} q, \\ q' \in winning \end{array} \right\}$
    return $sat$
case $\phi = ...$ // Cases for the other operators are standard
```

strategies will be checked again, while we know they are not winning for the remaining states. [BPQR14] presents an implementation of these three early termination optimizations, and experimentally shows that they are equivalent in performance. The remainder of this thesis thus considers only the first solution.

## 5.4 Pre-filtering

Compared to the *naive* approach, the *partial* approach improves the performance of model checking strategic formulas by enumerating and checking fewer strategies. We can still decrease the number of strategies to check by removing, from the model, the states and moves that cannot be part of a winning general strategy.

More precisely, it is costly to compute the states and the *uniform* strategies that are winning for a given objective, but it is far less costly to compute the states and the *general* strategies that are winning for the same objective by using the fixpoint computations of Section 5.1. We can thus use the result of the latter to ignore the states and actions that cannot be part of a winning *general* strategy, and reduce the number of *uniform* strategies to consider.

For instance, Figure 5.3 shows the variant of the card game with the cheating player. The moves that do not belong to a general strategy for the player to win the current game are dashed. Indeed, in $A, Q$ at the last step, the player loses the game; furthermore, in $A, Q$ at the previous step, keeping his card does not allow the player to win. On the other hand, in $K, Q$ at the last step, the player already wins. Furthermore, in $A, Q$ at the previous step, changing his card allows the player to win the game.

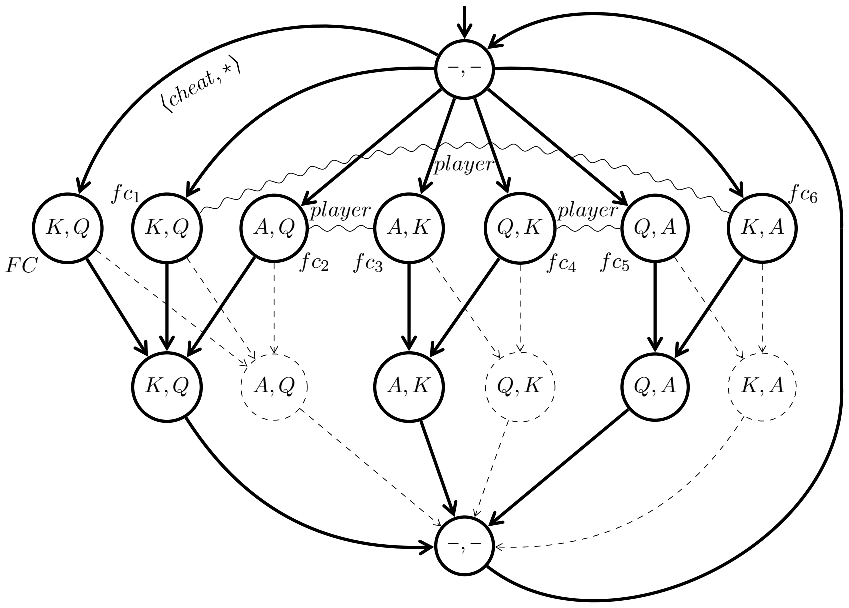


Figure 5.3: The graph of the card game with a cheating player. The dashed states and transitions do not belong to a strategy of the player to win the current game, the bold ones are the remaining ones.

To compute the set of Γ-moves belonging to a general strategy for Γ winning a given objective, we can use a modified version of the $filter$ algorithms that returns moves instead of states.

This section first describes how to modify the $filter$ algorithms to compute the winning moves instead of the winning states. Then it presents an extension of the naive approach taking filtered moves into account. Finally, it applies the same idea to the partial approach.

### 5.4.1 Computing the winning moves

Let $Pre^M_{\langle\!\langle\Gamma\rangle\!\rangle}$ be a variant of the $Pre_{\langle\!\langle\Gamma\rangle\!\rangle}$ function of Equation 5.3, defined as

$$Pre^M_{\langle\!\langle\Gamma\rangle\!\rangle}(M'_\Gamma, M_\Gamma) = \left\{ \langle q, a_\Gamma\rangle \in M_\Gamma \;\middle|\; \begin{array}{l} \forall a \in E_{Ag}(q), \\ a_\Gamma \sqsubseteq a \implies \delta(q,a) \in M'_\Gamma|_Q \end{array} \right\}.$$

This variant takes two sets of $\Gamma$-moves $M'_\Gamma$ and $M_\Gamma$ and returns the set of $\Gamma$-*moves* of $M_\Gamma$ reaching only states of moves of $M'_\Gamma$. From this new $Pre^M_{\langle\!\langle\Gamma\rangle\!\rangle}$ function, we can define variants of the *Stay* and *NFair* algorithms as

$$Stay^M_{\langle\!\langle\Gamma\rangle\!\rangle}(M_1, M_2, M_\Gamma) = \nu M'.\ M_2 \cup \left(M_1 \cap Pre^M_{\langle\!\langle\Gamma\rangle\!\rangle}(M', M_\Gamma)\right),$$

and

$$\begin{array}{l} NFair^M_{\langle\!\langle\Gamma\rangle\!\rangle}(M_\Gamma) = \\ \mu M'.\ \bigcup_{fc \in FC} Pre^M_{\langle\!\langle\Gamma\rangle\!\rangle}\left(Stay^M_{\langle\!\langle\Gamma\rangle\!\rangle}(M' \cup (Moves_\Gamma(\overline{fc}) \cap M_\Gamma), \varnothing, M_\Gamma), M_\Gamma\right), \end{array}$$

From these functions, we can finally define variants of the *filter* algorithms returning the moves instead of just the states belonging to a general strategy winning a given objective. More precisely, let

$$filter^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q', M_\Gamma) = Pre^M_{\langle\!\langle\Gamma\rangle\!\rangle}(Moves_\Gamma(Q') \cup NFair^M_{\langle\!\langle\Gamma\rangle\!\rangle}(M_\Gamma), M_\Gamma).$$

Intuitively, $filter^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q', M_\Gamma)$ returns the moves of $M_\Gamma$ belonging to a general strategy for which all enforced fair paths have their second state in $Q'$.

Furthermore, let

$$\begin{array}{l} filter^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, M_\Gamma) = \\ \mu M'.\ MQ_{1,2,N} \cap (MQ_2 \cup \\ \bigcup_{fc \in FC} Pre^M_{\langle\!\langle\Gamma\rangle\!\rangle}(Stay^M_{\langle\!\langle\Gamma\rangle\!\rangle}( \begin{array}{l} MQ_{1,2,N} \cap M'nfc, \\ MQ_2 \cap M'nfc, M_\Gamma \end{array} ), M_\Gamma)), \end{array}$$

where

$$\begin{aligned} MQ_{1,2,N} &= (Moves_\Gamma(Q_1) \cup Moves_\Gamma(Q_2) \cup NFair^M_{\langle\!\langle\Gamma\rangle\!\rangle}(M_\Gamma)) \cap M_\Gamma, \\ M'nfc &= M' \cup (Moves_\Gamma(\overline{fc}) \cap M_\Gamma), \\ MQ_2 &= Moves_\Gamma(Q_2) \cap M_\Gamma. \end{aligned}$$

Intuitively, $filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(Q_1, Q_2, M_\Gamma)$ returns the moves of $M_\Gamma$ belonging to a general strategy for which all enforced fair paths reach a state of $Q_2$ through states of $Q_1$.

Finally, let

$$filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, M_\Gamma) = Stay^M_{\langle\langle\Gamma\rangle\rangle}(MQ_{1,2,N}, MQ_2, M_\Gamma),$$

where

$$\begin{aligned}MQ_{1,2,N} &= (Moves_\Gamma(Q_1) \cup Moves_\Gamma(Q_2) \cup NFair^M_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma)) \cap M_\Gamma,\\ MQ_2 &= Moves_\Gamma(Q_2) \cap M_\Gamma.\end{aligned}$$

Intuitively, $filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, M_\Gamma)$ returns the moves of $M_\Gamma$ belonging to a general strategy for which all enforced fair paths reach a state of $Q_2$ through states of $Q_1$, or stay in $Q_1$ forever.

In the sequel, we sometimes abbreviate the $filter^M$ ones with the notation $filter^M_{op}(Q_1, Q_2, M_\Gamma)$ that corresponds to $filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(Q_1, M_\Gamma)$, $filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(Q_1, Q_2, M_\Gamma)$, or $filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, M_\Gamma)$, depending on the value of the strategic operator *op*.

### 5.4.2 The naive approach with pre-filtering

With the new $filter^M$ algorithms, we can modify $eval_{ATLK_{irF}}$ to pre-filter out of the computation the moves that cannot be part of a winning strategy. Algorithm 5.8 is a variant of $eval_{ALTK_{irF}}$ that uses pre-filtering. The differences are:

1. Lines 3 to 13 compute the set $filtered$ of moves belonging to general strategies that are winning for the objective.

2. The **if** statement of Line 14 checks whether there are still some states that could satisfy the formula. If no moves belong to winning general strategies, there cannot be a winning uniform strategy.

3. Line 16 enumerates the uniform strategies that use the remaining moves instead of the whole possible ones of $E_\Gamma$, reducing the number of strategies to check.

The proof of correctness of this approach is given in Section A.4.2 of Appendix A.

**Algorithm 5.8:** $eval^{PF}_{ATLK_{irF}}(S, \phi)$

```
Data: $S$ an iCGSf, $\phi$ an $ATLK_{irF}$ formula.
Result: The set of states of $S$ satisfying $\phi$.
case $\phi \in \{\langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi', \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2], \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]\}$
    $sat = \{\}$
3   case $\phi = \langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi'$
        $Q' = eval^{PF}_{ATLK_{irF}}(S, \phi')$
        $filtered = filter^{M}_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(Q', E_\Gamma)$
    case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2]$
        $Q_1 = eval^{PF}_{ATLK_{irF}}(S, \phi_1)$
        $Q_2 = eval^{PF}_{ATLK_{irF}}(S, \phi_2)$
        $filtered = filter^{M}_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(Q_1, Q_2, E_\Gamma)$
    case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]$
        $Q_1 = eval^{PF}_{ATLK_{irF}}(S, \phi_1)$
        $Q_2 = eval^{PF}_{ATLK_{irF}}(S, \phi_2)$
13      $filtered = filter^{M}_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)$
14  if $filtered = \varnothing$ then
        return $\varnothing$
16  for $M_\Gamma \in Split(\Gamma, filtered)$ do
        case $\phi = \langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi'$
            $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(Q'', M_\Gamma)$
        case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2]$
            $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(Q_1, Q_2, M_\Gamma)$
        case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]$
            $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, M_\Gamma)$
        $sat = sat \cup \{q \in winning \mid \forall ag \in \Gamma, \forall q' \sim_{ag} q, q' \in winning\}$
    return $sat$
case $\phi = ...$ // Cases for the other operators are standard
```

### 5.4.3 The partial approach with pre-filtering

We can also modify the $eval^{Partial}_{ATLK_{irF}}$ to take pre-filtered moves into account. The modifications are more involved in this case because the generation of uniform partial strategies is buried in the *ReachSplit* algorithm and the functions it depends on. The modified *ReachSplit* is given in Algorithm 5.9. It uses a modified version of the *Compatible* function defined as

$$Compatible^M(M'_\Gamma, M_\Gamma) = \left\{ \langle q', a'_\Gamma \rangle \in M'_\Gamma \middle| \begin{array}{l} \nexists \langle q, a \rangle \in M_\Gamma, ag \in \Gamma \text{ s.t.} \\ q \sim_{ag} q' \wedge a_\Gamma(ag) \neq a'_\Gamma(ag) \end{array} \right\}.$$

Instead of taking a subset of states $Q'$ and a set of $\Gamma$-moves $M_\Gamma$, it takes two subsets of $\Gamma$-moves. It returns the moves of $M'_\Gamma$ that are compatible with moves of $M_\Gamma$. The difference between this version and the original *Compatible* function is that only moves of $M'_\Gamma$ are returned, instead of any compatible move of the whole system.

Algorithm 5.9 first computes the new states reachable from some move of $M_\Gamma$. Then, it gets the moves of these states in $filtered$. Finally, it computes the subset of these new moves compatible with $M_\Gamma$. If there are no such compatible moves, this means that there are no new moves of $filtered$ reachable from $M_\Gamma$. Otherwise, $ReachSplit^{PF}$ splits these compatible moves into non-conflicting subsets $M'_\Gamma$ and recursively extends $M_\Gamma \cup M'_\Gamma$ with reachable moves of $filtered$.

```
Algorithm 5.9: $ReachSplit^{PF}(\Gamma, M_\Gamma, filtered)$
Data: $\Gamma$ a subset of agents, $M_\Gamma \subseteq E_\Gamma$ a set of non-$\Gamma$-conflicting
      $\Gamma$-moves, $filtered$ a set of $\Gamma$-moves.
Result: The set of largest non-$\Gamma$-conflicting extensions of $M_\Gamma$ with
        moves of $filtered$ reachable from $M_\Gamma$.
$new\_states = Post(Q, M_\Gamma) \setminus M_\Gamma|_Q$
$new\_moves = \{\langle q, a_\Gamma \rangle \in filtered \mid q \in new\_states\}$
$compatible = Compatible^M(new\_moves, M_\Gamma)$
if $compatible = \emptyset$ then return $\{M_\Gamma\}$
else
    $new\_strats = Split(\Gamma, compatible)$
    $strats = \{\}$
    for $M'_\Gamma \in new\_strats$ do
        $strats = strats \cup ReachSplit^{PF}(\Gamma, M_\Gamma \cup M'_\Gamma, filtered)$
    return $strats$
```

Thanks to this new $ReachSplit^{PF}$ algorithm, we can define the $PartialStrats^{PF}$ function as

$$PartialStrats^{PF}(\Gamma, Q', M_\Gamma) =$$
$$\bigcup \{ReachSplit^{PF}(\Gamma, M'_\Gamma, M_\Gamma) \mid M'_\Gamma \in Split(\Gamma, Moves_\Gamma(Q') \cap M_\Gamma)\}.$$

It takes a group $\Gamma$ of agents, a set $Q'$ of states, and a set $M_\Gamma$ of $\Gamma$-moves, and returns the set of non-$\Gamma$-conflicting subsets of $M_\Gamma$ reachable from states of $Q'$.

Finally, we can modify the $eval^{Partial}_{ATLK_{irF}}$ to take pre-filtered moves into account. Algorithm 5.10 extends Algorithm 5.5 by computing the set of moves $filtered$ belonging to a winning general strategy thanks to the new $filter^M$ algorithms (Lines 3 to 13). If $filtered$ covers no states of $Q'$, there cannot be a winning uniform strategy and the algorithm is done (Line 14). Otherwise, it generates all strategies to check thanks to the new $PartialStrats^{PF}$ function (Line 17), and checks them one by one (Lines 18 to 23), accumulating in *sat* the set of states for which the strategy is winning (Line 24).

The correctness of this $eval^{Partial,PF}_{ATLK_{irF}}$ algorithm is proved in Section A.4.3 of Appendix A.

## 5.5 Backward generation of strategies

The *partial* approaches are based on *ReachSplit* algorithms that perform a *forward* exploration of the structure under investigation to generate the partial strategies, and on $filter$ algorithms that perform a *backward* exploration of the same structure to evaluate a given strategy. Instead of performing the former with a forward exploration, we can generate the strategies with a backward exploration.

As an example, let us consider the simple card game of the Introduction. Because the player does not see the card on table nor the card of the dealer before making a decision, he has no uniform strategy to win the game.

In this example, to check whether there exists a strategy to win the game with the $A$, the naive and partial approaches test all possible strategies, that is, they make a choice for the player in all states of the structure and check whether one of them is winning in the initial state.

Another approach is to start by looking at the states in which the player already wins the game with the $A$, and look at the non-conflicting moves that can reach these states. By iterating this procedure, we can explore the parts of the uniform strategies that surely reach the winning states.

Figure 5.4 shows the graph of the simple card game with the winning parts of a uniform strategy in bold. This strategy chooses to swap the card when the player has $Q$. Note that this set of non-conflicting moves cannot be extended with non-conflicting moves that would surely reach the set. Thus no uniform strategy that makes these choices is winning for the initial state, because the initial state has no move in the set. There exists another subset of moves that make the player surely reach the state in which she wins with the $A$: keeping her card when she has the $A$.

**Algorithm 5.10:** $eval^{Partial,PF}_{ATLK_{irF}}(S, Q', \phi)$

```
Data: $S$ an iCGSf, $Q' \subseteq Q$ a subset of states, $\phi$ an $ATLK_{irF}$
      formula.
Result: The set of states of $Q'$ satisfying $\phi$.
case $\phi \in \{\langle\langle\Gamma\rangle\rangle \boldsymbol{X}\, \phi', \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2], \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]\}$
    $sat = \{\}$
    case $\phi = \langle\langle\Gamma\rangle\rangle \boldsymbol{X}\, \phi'$
        $Q'' = eval^{Partial,PF}_{ATLK_{irF}}(S, Post([Q']^E_\Gamma, E_\Gamma), \phi')$
        $filtered = filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(Q'', E_\Gamma)$
    case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2]$
        $Q_1 = eval^{Partial,PF}_{ATLK_{irF}}(S, Q, \phi_1)$
        $Q_2 = eval^{Partial,PF}_{ATLK_{irF}}(S, Q, \phi_2)$
        $filtered = filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(Q_1, Q_2, E_\Gamma)$
    case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]$
        $Q_1 = eval^{Partial,PF}_{ATLK_{irF}}(S, Q, \phi_1)$
        $Q_2 = eval^{Partial,PF}_{ATLK_{irF}}(S, Q, \phi_2)$
        $filtered = filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)$
    $Q' = Q' \cap filtered|_Q$
    if $Q' = \emptyset$ then
        return $\emptyset$
    for $f_\Gamma \in PartialStrats^{PF}(\Gamma, [Q']^E_\Gamma, filtered)$ do
        case $\phi = \langle\langle\Gamma\rangle\rangle \boldsymbol{X}\, \phi'$
            $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(Q'', f_\Gamma)$
        case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2]$
            $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(Q_1, Q_2, f_\Gamma)$
        case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]$
            $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, f_\Gamma)$
        $sat = sat \cup \left\{ q \in winning \cap Q' \,\middle|\, \begin{array}{l} \forall ag \in \Gamma, \forall q' \sim_{ag} q, \\ q' \in winning \end{array} \right\}$
    return $sat$
case $\phi = ...$ // Cases for the other operators are standard
```

This cannot be winning in the initial state either. There are thus only two strategies to check before concluding, while the naive and partial approaches need to check 8 strategies.

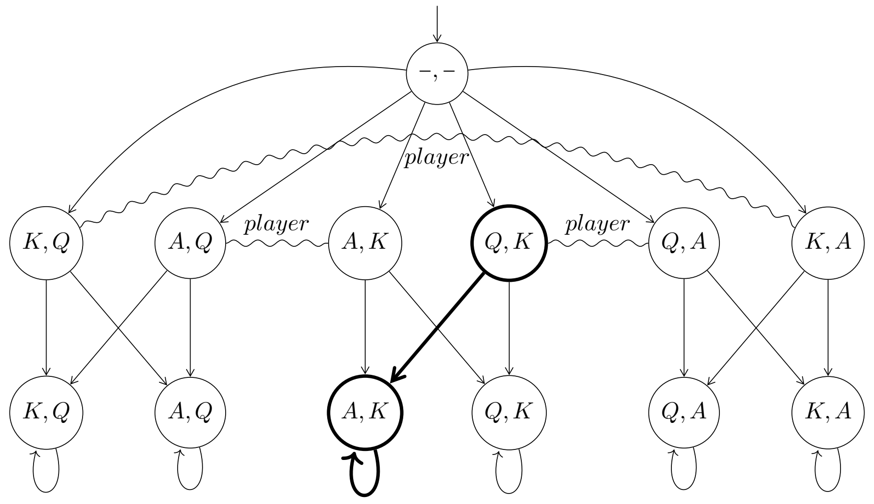


Figure 5.4: The graph of the simple card game. In bold, the winning part of a strategy that chooses to swap when the player has the $Q$.

This section presents an approach based on this idea to generate the winning parts of the uniform strategies from the target states. In the sequel, it is called the *backward* approach because it generates the strategies with a backward exploration of the structure.

This approach has some serious limitations: it cannot handle greatest fixpoints-based objectives because, in this case, we cannot build the winning strategies from the ground up. The approach thus cannot handle $\langle\langle\Gamma\rangle\rangle\mathbf{W}$ and $\langle\langle\Gamma\rangle\rangle\mathbf{G}$ objectives, nor fairness constraints. In the sequel, the backward approach assumes that the iCGSf has only one fairness constraint that covers the whole set of states of the structure—and thus corresponds to an iCGS without fairness constraints—, and only $\langle\langle\Gamma\rangle\rangle\mathbf{X}$ and $\langle\langle\Gamma\rangle\rangle\mathbf{U}$ formulas are considered; the other formulas can be evaluated with any other approach of this thesis.

The approach, presented in Algorithm 5.12, uses the $eval^{Backward}_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}$ algorithm of Algorithm 5.11 to compute the states for which there exists a strategy to win a $\langle\langle\Gamma\rangle\rangle\mathbf{U}$ objective.

Let $Q_1, Q_2 \subseteq Q$ be two subsets of states. We say that a non-$\Gamma$-conflicting subset of $\Gamma$-moves $M_\Gamma$ *enforces to reach* $Q_2$ *through* $Q_1$ if $Q_2 \subseteq M_\Gamma|_Q$, and for all states $q \in M_\Gamma|_Q$, for all paths $\pi \in out(M_\Gamma, q)$, $\pi$ is finite and

$$\pi(|\pi|) \in Q_2 \wedge \forall i, 0 \leq i \leq |\pi| - 1, \pi(i) \in Q_1 \backslash Q_2.$$

In other words, $M_\Gamma$ enforces to reach $Q_2$ through $Q_1$ if all the paths enforced by $M_\Gamma$ reach a state of $Q_2$ through states of $Q_1 \backslash Q_2$.

Given two formulas $\phi_1$ and $\phi_2$, there exists a strategy $f_\Gamma$ such that all outcomes from some state $q$ satisfy $\phi_1$ **U** $\phi_2$ iff there exists a subset of moves $M'_\Gamma$ containing a move for $q$ that *enforces to reach states satisfying* $\phi_2$ *through states satisfying* $\phi_1$. $eval^{Backward}_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}$ uses this property to compute the states for which there exists a winning strategy for a $\langle\langle\Gamma\rangle\rangle\mathbf{U}$ objective.

More precisely, it takes as arguments a subset $Q' \subseteq Q$ such that $Q' = [Q']^E_\Gamma$, $M_\Gamma \subseteq E_\Gamma$ a non-conflicting set of moves, and two subsets of states $Q_1, Q_2 \subseteq Q$ such that $M_\Gamma$ *enforces to reach* $Q_2$ *through* $Q_1$. From these arguments, it computes the set of states $q$ such that there exists a uniform strategy $f'_\Gamma$ that shares the same choices as $M_\Gamma$ and such that all outcomes of $f'_\Gamma$ from all states indistinguishable from $q$ reach a state of $Q_2$ through states of $Q_1$.

To compute this set of states, $eval^{Backward}_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}$ first computes the set of states for which there surely cannot exist such a strategy (in *lose*) and for which there surely exists such a strategy (in *win*). If *lose* and *win* cover all states of interest $Q'$, then the job is done. Otherwise, it computes the moves *compatible* from states of $Q_1$ that can surely reach $M_\Gamma$ and are compatible with it, and recursively calls itself with $M_\Gamma$ extended with the non-conflicting subsets of *compatible*, accumulating in *sat* the results.

$eval^{Backward}_{ATLK_{irF}}$ can handle $\langle\langle\Gamma\rangle\rangle\mathbf{X}$ and $\langle\langle\Gamma\rangle\rangle\mathbf{U}$ formulas. For $\langle\langle\Gamma\rangle\rangle\mathbf{X}$, it recursively computes the states of $S$ satisfying the sub-formula $\phi'$ and then computes the states for which there exists a move for all indistinguishable states. More precisely, it splits the set of moves that $\Gamma$ can use to enforce to reach the states satisfying $\phi'$ into non-conflicting greatest subsets. There exists a strategy that wins the objective in $q$ iff there exists an action that enforces to reach states of $Q'''$ in one step in all states indistinguishable from $q$, and that is exactly what is computed by the algorithm and accumulated in *sat*.

For $\langle\langle\Gamma\rangle\rangle\mathbf{U}$, it recursively computes the states of $S$ satisfying the sub-formulas $\phi_1$ and $\phi_2$. Then is uses the $eval^{Backward}_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}$ algorithm with the greatest non-conflicting subsets of the moves of the states satisfying $\phi_2$ to accumulate in *sat* the states $q$ such that there exists a strategy to win the objective in all states indistinguishable from $q$.

The correctness of the $eval^{Backward}_{ATLK_{irF}}$ algorithm is given in Section A.5 of Appendix A.

Section 5.3 proposed to use early termination and caching for the partial approach. Early termination is already embedded in the algorithms of this section. They keep track of the states of interest for which no decision has been made yet, and stop whenever there are no such remaining states. Caching is useless in the present case. It is useful for the partial approach because the sub-formulas are evaluated again and

**Algorithm 5.11:** $eval^{Backward}_{\langle\langle\Gamma\rangle\rangle \mathbf{U}}(Q', M_\Gamma, Q_1, Q_2)$

```
Data: $Q' \subseteq Q$ a subset of states such that $Q' = [Q']^E_\Gamma$, $M_\Gamma \subseteq E_\Gamma$ a
      non-$\Gamma$-conflicting set of $\Gamma$-moves, $Q_1, Q_2 \subseteq Q$ two subsets of
      states such that $M_\Gamma$ enforces to reach $Q_2$ through $Q_1$.
Result: The set of states $q \in Q'$ such that there exists a uniform
        strategy $f'_\Gamma \supseteq M_\Gamma$ such that all fair outcomes of $f'_\Gamma$ from
        all states indistinguishable from $q$ reach a state of $Q_2$
        through states of $Q_1$.
  $notlose = filter_{\langle\langle\Gamma\rangle\rangle \mathbf{U}}(\Gamma, Q_1, M_\Gamma|_Q, E_\Gamma)$
  $lose = \{q \in Q' \mid \exists ag \in \Gamma$ s.t. $\exists q' \in Q$ s.t. $q' \sim_{ag} q \wedge q' \notin notlose\}$
3 $win = \{q \in Q' \mid \forall ag \in \Gamma, \forall q' \in Q, q' \sim_{ag} q \implies q' \in M_\Gamma|_Q\}$
  if $Q' \backslash (lose \cup win) = \varnothing$ then return $win$
  else
6     $Q' = Q' \backslash (lose \cup win)$
      $new\_moves = (Pre^M_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma, E_\Gamma) \cap Moves_\Gamma(Q_1)) \backslash M_\Gamma$
      $compatible = Compatible^M(new\_moves, M_\Gamma)$
      if $compatible = \varnothing$ then return $win$
      else
          for $M'_\Gamma \in Split(\Gamma, compatible)$ do
              $win = win \cup eval^{Backward}_{\langle\langle\Gamma\rangle\rangle \mathbf{U}}(Q', M_\Gamma \cup M'_\Gamma, Q_1, Q_2)$
              $Q' = Q' \backslash win$
              if $Q' = \varnothing$ then return $win$
          return $win$
```

again, on different subsets of states. On the other hand, the backward approach evaluates the sub-formulas only once, and builds the winning strategies from them. It is thus not necessary to cache the results as they would never be reused.

Finally, Section 5.4 proposed to apply pre-filtering for the naive and partial approaches. The idea is to pre-compute the sub-part of the structure that can be part of a winning strategy. The search for winning strategies can then be restricted to this sub-part. Pre-filtering is not useful for the backward approach. Indeed, by construction, the approach explores the parts of the strategies that are winning, and ignores the parts that are losing. Pre-filtering would thus bring no additional information.

**Algorithm 5.12:** $eval^{Backward}_{ATLK_{irF}}(S, Q', \phi)$

```
Data: $S$ an iCGSf, $Q' \subseteq Q$ a subset of states, $\phi$ an $ATLK_{irF}$
      formula.
Result: The set of states of $Q'$ satisfying $\phi$.
case $\phi \in \{\langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi', \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2]\}$
    $Q'' = [Q']^E_\Gamma$
    $sat = \emptyset$
    case $\phi = \langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi'$
        $Q''' = eval^{Backward}_{ATLK_{irF}}(S, Post([Q'']^E_\Gamma, E_\Gamma), \phi')$
        for $M_\Gamma \in Split(\Gamma, Pre^M_{\langle\langle\Gamma\rangle\rangle}(Moves_\Gamma(Q'''), E_\Gamma))$ do
            $sat = sat \cup \left\{ q \in Q'' \;\middle|\; \begin{array}{l} \forall ag \in \Gamma, \forall q' \in Q, \\ q' \sim_{ag} q \implies q' \in M_\Gamma|_Q \end{array} \right\}$
            $Q'' = Q'' \backslash sat$
            if $Q'' = \emptyset$ then return $sat \cap Q'$
        return $sat \cap Q'$
    case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2]$
        $Q_1 = eval^{Backward}_{ATLK_{irF}}(S, Q, \phi_1)$
        $Q_2 = eval^{Backward}_{ATLK_{irF}}(S, Q, \phi_2)$
14      $sat = \{q \in Q'' \mid \forall ag \in \Gamma, \forall q' \in Q, q' \sim_{ag} q \implies q' \in Q_2\}$
        if $sat = Q''$ then
            return $sat \cap Q'$
        $Q'' = Q'' \backslash sat$
        for $M_\Gamma \in Split(\Gamma, Moves_\Gamma(Q_2))$ do
            $sat = sat \cup eval^{Backward}_{\langle\langle\Gamma\rangle\rangle \mathbf{U}}(Q'', M_\Gamma, Q_1, Q_2)$
            $Q'' = Q'' \backslash sat$
            if $Q'' = \emptyset$ then return $sat \cap Q'$
        return $sat \cap Q'$
case $\phi = ...$ // Cases for the other operators are standard
// $\langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]$ is not supported
```

## 5.6 Complexity analysis

This section discusses the time complexity of the $ATLK_{irF}$ model-checking problem. The end of Section 4.3 already showed that the $ATLK_{irF}$ model checking problem is $\Delta^P_2$-hard as the $ATL_{ir}$ model checking problem can be reduced to $ATLK_{irF}$. This section shows that the $ATLK_{irF}$ model-checking problem is in fact $\Delta^P_2$-complete by showing it is in $\Delta_2^P$. First it shows that the $filter$ algorithms run in polynomial time in terms of the size of the model and of the formula. Then it presents a non-deterministic polynomial algorithm for checking a single strategic formula. Thanks to this algorithm, we can conclude that the whole model-checking problem is in $\Delta_2^P$ as it needs a polynomial number of calls to the non-deterministic polynomial algorithm.

Let us show that the $filter$ algorithms are in PTIME, that is, they are polynomial in terms of the size of their arguments. First, the computation of the $Pre_{\langle\langle\Gamma\rangle\rangle}$ function is polynomial: given any set of states $Q'$ and $\Gamma$-moves $M_\Gamma$, it is necessary to check each state and transition of the system at most once to get all the states belonging to $Pre_{\langle\langle\Gamma\rangle\rangle}(Q', M_\Gamma)$. Second, $Stay_{\langle\langle\Gamma\rangle\rangle}(Q_1, Q_2, M_\Gamma)$ can be computed in polynomial time: the function $\tau(Z) = Q_2 \cup (Q_1 \cap Pre_{\langle\langle\Gamma\rangle\rangle}(Q'))$ is monotonic, thus the least fixpoint of $\tau(Z)$ is reached in at most $|Q|$ steps. Each step uses the $Pre_{\langle\langle\Gamma\rangle\rangle}$ function, thus this function is evaluated at most $|Q|$ times, hence the polynomial time complexity of $Stay_{\langle\langle\Gamma\rangle\rangle}$. Third, $NFair_{\langle\langle\Gamma\rangle\rangle}$ can also be computed in polynomial time: it is the greatest fixpoint of a $\tau$ function evaluating $Stay_{\langle\langle\Gamma\rangle\rangle}$ for each fairness constraint $fc \in FC$. The $\tau$ function is monotonic, thus only a polynomial number of calls to $Stay_{\langle\langle\Gamma\rangle\rangle}$ are needed, hence a polynomial time complexity. Finally, the three $filter_{\langle\langle\Gamma\rangle\rangle}$ algorithms are compositions of the above three functions, thus they have a polynomial time complexity.

Let us now show that there is a $\Delta_2^P$ algorithm for model checking $ATLK_{irF}$ strategic formulas, thus the problem is in $\Delta_2^P$. The proposed algorithm is a non-deterministic variant of the naive algorithm where the uniform strategies are non-deterministically chosen. More precisely, the algorithm uses Algorithm 5.13 to compute the states of the model satisfying a given strategic formula.

Algorithm 5.13 is effectively non-deterministic: the **choose** operation at Line 7 is a non-deterministic choice among the enabled actions of $ag$ in $q$. Furthermore, supposing that $eval^{ND}_{ATLK_{irF}}$ is already computed for the sub-formulas of $\phi$, this algorithm is polynomial: the body of the **while** loop that non-deterministically chooses a uniform strategy is repeated at most $|Q|$ times since the size of $states$ is decreased by at least one at each step. This loop is itself repeated $|\Gamma|$ times. Furthermore, the $filter$ algorithms run in polynomial time. Thus, $eval^{ND}_{ATLK_{irF}}$ must be called for the polynomially many (indirect) sub-formulas of $\phi$, leading to a polynomial number of calls to an NP algorithm. We can then conclude that Algorithm 5.13 is in $\Delta_2^P$, and that the model-checking problem for $ATLK_{irF}$ is $\Delta_2^P$-complete.

**Algorithm 5.13:** $eval^{ND}_{ATLK_{irF}}(S, \phi)$

```
Data: $S$ a given iCGSf, $\langle\langle\Gamma\rangle\rangle\ \psi$ an $ATLK_{irF}$ strategic formula.
Result: The set of states of $S$ satisfying $\langle\langle\Gamma\rangle\rangle\ \psi$.
$f_\Gamma = \langle\rangle$
for $ag \in \Gamma$ do
    $f_{ag} = \varnothing$; $states = Q$
    while $states \neq \varnothing$ do
        $q =$ pick one element in $states$
        $states = states \backslash [q]_{ag}$
7       $a_{ag} =$ choose one action in $E_{ag}(q)$
        $f_{ag} = f_{ag} \cup \{\langle q', a_{ag}\rangle \in E_{ag} \mid q' \in [q]_{ag}\}$
    $f_\Gamma = f_\Gamma + \langle f_{ag}\rangle$
case $\psi = \boldsymbol{X}\ \phi'$
    $Q' = eval^{ND}_{ATLK_{irF}}(S, \phi')$
    $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(Q', f_\Gamma)$
case $\psi = \phi_1\ \boldsymbol{U}\ \phi_2$
    $Q_1 = eval^{ND}_{ATLK_{irF}}(S, \phi_1)$; $Q_2 = eval^{ND}_{ATLK_{irF}}(S, \phi_2)$
    $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(Q_1, Q_2, f_\Gamma)$
case $\psi = \phi_1\ \boldsymbol{W}\ \phi_2$
    $Q_1 = eval^{ND}_{ATLK_{irF}}(S, \phi_1)$; $Q_2 = eval^{ND}_{ATLK_{irF}}(S, \phi_2)$
    $winning = filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, f_\Gamma)$
return $\{q \in Q \mid \forall ag \in \Gamma, \forall q' \sim_{ag} q, q' \in winning\}$
```

## 5.7 Implementation

The five algorithms presented in this chapter have been implemented in the BDD-based framework of PyNuSMV [BP13]. This section discusses this implementation, focusing on parts that have been implemented slightly differently to fit the BDD framework. Second, this section briefly presents the modeling language supported by the implementation, and how a given iCGSf can be modelled with it.

### 5.7.1 Implementing the algorithms with binary decision diagrams

In the BDD-based model-checking framework, sets of moves and sets of states are represented with BDDs. Furthermore, the whole transition relation of the system, as well as the different knowledge relations for each agent, are also represented with BDDs (see for instance [CGP99, Chapter 6]). In this framework, any operation over sets of states or moves is thus performed as an operation on BDDs.

While most of the algorithm parts can be directly implemented through BDD operations and simple loops, such as unions, intersections and fixpoint computations, some operations need more work to be implemented. This section focuses on the $Pre_{\langle\langle\Gamma\rangle\rangle}$ operator and its variations, on the *Split* algorithm, and on the *Compatible* functions. In the sequel, we sometimes write $B[v]$ for the BDD $B$ with support $v$, that is, the BDD with $v$ as the set of significant variables. For instance, $M_\Gamma[q, a_\Gamma]$ is a BDD $M_\Gamma$ defined on variables representing pairs of state $q$ and joint action $a_\Gamma$ for $\Gamma$.

Given a BDD representing the transition relation of the system, we can easily define the operator $Pre^M(Q')[q, a]$ returning the BDD of the moves leading to at least one state of the BDD of the set $Q'[q]$, that is, the function

$$Pre^M(Q') = \{\langle q, a\rangle \in E_{Ag} \mid \exists q' \in Q' \text{ s.t. } \delta(q, a) = q'\}.$$

This operator is present in BDD-based model checking tools such as NuSMV because it is the basis operator for BDD-based $CTL$ model checking. The implementation of this operator relies on the BDD representation of the transition relation and the existential quantification on BDDs.

Given this $Pre^M$ operator, a BDD representing a set of states $Q'[q]$, and a BDD representing a set of $\Gamma$-moves $M_\Gamma[q, a_\Gamma]$, we can implement the $Pre_{\langle\langle\Gamma\rangle\rangle}$ function with BDD operations, as captured by the following theorem.

**Theorem 5.2.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a subset of agents $\Gamma \subseteq Ag$, a subset of states $Q' \subseteq Q$, and a closed set of $\Gamma$-moves $M_\Gamma$,*

$$Pre_{\langle\langle\Gamma\rangle\rangle}(Q', M_\Gamma) = \exists a((\overline{\exists a_{\overline{\Gamma}}.Pre^M(\overline{Q'})}) \cap (\exists a_{\overline{\Gamma}}.Pre^M(Q')) \cap M_\Gamma),$$

*where $\exists$ is the existential quantification over BDDs, $\cap$ is the conjunct of BDDs, and $\overline{\phantom{x}}$ is the negation of BDDs.*

*Proof.* The result of $(\exists a_{\overline{\Gamma}}.Pre^M(\overline{Q'}))[q, a_\Gamma]$ is the set of moves of $\Gamma$ such that there exists a completing action leading to $\overline{Q'}[q]$. Thus, $\overline{\exists a_{\overline{\Gamma}}.Pre^M(\overline{Q'})}$ is the set of $\Gamma$-moves such that all completing actions surely lead to $Q'$. Then, $(\overline{\exists a_{\overline{\Gamma}}.Pre^M(\overline{Q'})}) \cap (\exists a_{\overline{\Gamma}}.Pre^M(Q'))$ is the set of $\Gamma$-moves surely leading to $Q'$. $\overline{\exists a_{\overline{\Gamma}}.Pre^M(\overline{Q'})}$ must be restricted to

$(\exists a_{\overline{\Gamma}}.Pre^M(Q'))[q, a_\Gamma]$ to be sure to only keep actions that are actually enabled. Furthermore, $(\overline{\exists a_{\overline{\Gamma}}.Pre^M(\overline{Q'})}) \cap (\exists a_{\overline{\Gamma}}.Pre^M(Q')) \cap M_\Gamma$ is the set of moves of $M_\Gamma$ that surely lead to $Q'$. Thus, the whole $Pre_{\langle\langle\Gamma\rangle\rangle}(Q', M_\Gamma)[q]$ BDD is effectively the set of states for which there exists a move in $M_\Gamma$ surely leading to $Q'$. □

The $Pre^M_{\langle\langle\Gamma\rangle\rangle}(M'_\Gamma, M_\Gamma)[q, a_\Gamma]$ function is implemented in a similar way, as captured by the following theorem.

**Theorem 5.3.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$*, a subset of agents* $\Gamma \subseteq Ag$*, a subset of* $\Gamma$*-moves* $M'_\Gamma$*, and a closed set of* $\Gamma$*-moves* $M_\Gamma$*,*

$$Pre^M_{\langle\langle\Gamma\rangle\rangle}(M'_\Gamma, M_\Gamma) = (\overline{\exists a_{\overline{\Gamma}}.Pre^M(\overline{\exists a.M'_\Gamma})}) \cap (\exists a_{\overline{\Gamma}}.Pre^M(\exists a.M'_\Gamma)) \cap M_\Gamma.$$

*Proof.* The proof is similar to $Pre_{\langle\langle\Gamma\rangle\rangle}$. The differences are that $Pre^M_{\langle\langle\Gamma\rangle\rangle}$ first gets the states of $M'_\Gamma$ with $\exists a.M'_\Gamma$, and does not abstract away the actions of $\Gamma$ from the computed subset of moves of $M_\Gamma$. □

The *SplitAgent* algorithm is, on the other hand, not easily implemented as described in Algorithm 5.2. More precisely, the computation of the set *conflicting* of conflicting $\Gamma$-moves of $M_\Gamma$ needs some modifications to be implemented with BDDs. In fact, to directly compute this set of conflicting moves, we need to quantify over a relation between two moves, that is, we need to quantify over a BDD ranging over two copies of state and action variables. Such a BDD can become really huge and is not necessarily available in a BDD-based model-checking framework such as NuSMV.

One way to implement the *SplitAgent* algorithm is to pick one move in $M_\Gamma$, get all equivalent states—by using a BDD ranging over two copies of state variables, instead of two copies of state and action variables—then all moves of $M_\Gamma$ restricted to these states, and finally check that there is only one available action for these states in $M_\Gamma$. If this is not the case, this particular equivalence class of states is conflicting and needs to be split. Otherwise, we can ignore this class and pick another move from $M_\Gamma$.

This implementation is less efficient than Algorithm 5.2 because it has to enumerate all equivalence classes (through picking moves) before concluding that $M_\Gamma$ is non-$\Gamma$-conflicting. On the other hand, it does not need to build a large BDD representing the relation between two moves.

The *Compatible* functions suffer from the same problem as, to be able to find compatible moves, we need to compare pairs of moves. The same solution is applied: by picking states and isolating, for each equivalence class, the compatible actions, we can compute the interesting moves.

### 5.7.2 Modeling language

The modeling language supported by the implementation is based on the NuSMV language. To model an iCGSf, we first provide a standard NuSMV model. This model describes the states and actions of the system, the transition function, the labeling function and the fairness constraints. In addition to this NuSMV model, we provide the set of agents of the system, defined by their name, the set of state variables they observe and the set of input variables representing their actions. These agents thus define the actions that are enabled for them in each state—through the transition function of the NuSMV model—and the equivalence classes of indistinguishable states. Additional conditions must be met by the NuSMV model to correctly represent an iCGSf:

- the sets of actions of agents must be disjoint, that is, they control different actions of the model;
- for each agent and each state of the system, the enabled actions are not constrained by the actions of another agent;
- for each agent and each equivalence class—defined by the state variables he observes—the enabled actions are the same in all the states of the equivalence class.

If these additional conditions are met by the NuSMV model, it correctly represents an iCGSf.

The model of the repeated card game is given in Figures 5.6, 5.5 and 5.7. Figure 5.5 defines the `main` module. It is composed of

- a `step` variable to keep track of the steps of the game;
- the cards of the player (`pcard`) and the dealer (`dcard`);
- the dealer's and the player's protocols, implemented by dedicated modules;
- a `DEFINE` clause for winning states, that is, final states (`step = 2`) in which the player's card wins over the dealer's.

The rest of the module defines the initial state with the `INIT` clause, how the value of `step` is incremented, and how the cards of both agents are computed:

- the card of the player is the one chosen by the dealer at the first step, and is the third one at second step if the player swaps it;

```
MODULE main
    VAR step   : 0..2;
        pcard  : {none, Ac, K, Q};
        dcard  : {none, Ac, K, Q};
        dealer : Dealer(step);
        player : Player(step);

    DEFINE
    win := step = 2 & ( (pcard = Ac & dcard = K) |
                        (pcard = K & dcard = Q)  |
                        (pcard = Q & dcard = Ac) );

    INIT step = 0 & pcard = none & dcard = none

    TRANS next(step) = (step + 1) mod 3

    TRANS step = 0 -> next(pcard) = dealer.to_player
    TRANS step = 1 -> case player.action = keep :
                               next(pcard) = pcard;
                           TRUE :
                               next(pcard) != pcard &
                               next(pcard) != dcard &
                               next(pcard) != none;
                      esac
    TRANS step = 2 -> next(pcard) = none

    TRANS step = 0 -> next(dcard) = dealer.to_dealer
    TRANS step = 1 -> next(dcard) = dcard
    TRANS step = 2 -> next(dcard) = none
```

Figure 5.5: The NuSMV model of the iCGSf of the repeated card game: the `main` module.

- the card of the dealer is the one he chose at the first step, and stays the same until the end of the game.

The `Player` and `Dealer` modules are presented in Figure 5.6. These modules simply define the protocols of the agents, that is, what they can do based on what they observe:

- the player can `keep` or `swap` his card when the game is at the second step, otherwise he can do nothing (`none`);

- the dealer gives a card to the player (`to_player`) and to himself (`to_dealer`) when the game starts, and he can do nothing otherwise.

Finally, the six fairness constraints of Figure 5.7 tell that the dealer must give all pairs of cards infinitely often.

```
MODULE Player(step)
    IVAR action : {none, keep, swap};
    TRANS action in ( step = 1 ? {keep, swap} : {none} )

MODULE Dealer(step)
    IVAR to_player : {none, Ac, K, Q};
         to_dealer : {none, Ac, K, Q};
    TRANS step = 0 -> (to_player != to_dealer &
                       to_player != none &
                       to_dealer != none)
    TRANS step != 0 -> (to_player = none &
                        to_dealer = none)
```

Figure 5.6: The NuSMV model of the iCGSf of the repeated card game: the definition of the agents' modules.

```
FAIRNESS step = 1 & pcard = Ac & dcard = K
FAIRNESS step = 1 & pcard = Ac & dcard = Q
FAIRNESS step = 1 & pcard = K  & dcard = Ac
FAIRNESS step = 1 & pcard = K  & dcard = Q
FAIRNESS step = 1 & pcard = Q  & dcard = Ac
FAIRNESS step = 1 & pcard = Q  & dcard = K
```

Figure 5.7: The NuSMV model of the iCGSf of the repeated card game: the fairness constraints.

This NuSMV model is not sufficient to define a complete iCGSf. It only defines the temporal aspects of the system, but misses the definition of the agents, what they can do and what they observe. In the case of the card game, the agents and their actions are already defined in two separated modules. More precisely,

- the player is defined as an agent named `player`, who controls the input variable `player.action`, and who observes the value of the `step` and `pcard` variables;

- the dealer is defined as an agent named `dealer`, who controls the input variables `to_player` and `to_dealer`, and who observes the `step`, `pcard`, and `dcard` variables, too.

It is easy to check that this model and these agents satisfy the additional conditions above, and correctly define an iCGSf.

# *Chapter 6*

# *Existing symbolic approaches*

Other solutions have been proposed for the $ATL_{ir}$ model-checking problem. Some of them are well-suited for a BDD-based framework. This chapter adapts the ideas presented by Pilecki et al. [PBJ14] and Huang and van der Meyden [HvdM14b] to fit $ATLK_{irF}$.

The approach of Pilecki et al. is limited to one initial state and to one strategic operator, and does not support fairness constraints. This chapter adapts and improves their idea to remove these limitations and to go a bit further.

On the other hand, the logic supported by the approach of Huang and van der Meyden already subsumes $ATLK_{irF}$. So this chapter simply restricts their algorithm to fit this logic. Also, none of these approaches consider the idea of pre-filtering moves, so this chapter defines and presents such variants.

## 6.1 Interleaving strategy generation and verification

This approach is based on the work presented by Pilecki et al. [PBJ14]. The main idea of Pilecki et al. is that we do not need to build a complete partial strategy before deciding whether it can be winning for some states of interest. Indeed, if we can check that all extensions of an incomplete partial strategy are winning, we do not need to build all these extensions. Fortunately, it is easy to check whether all extensions of such a strategy are winning or not. In the sequel, this approach is called the *early* approach, because it tries to decide whether there exists a winning extension of an incomplete strategy as soon as possible.

To give some intuition, let us go back to the card game with a cheating player, but consider now that $A$ wins over $K$ and $Q$, $K$ wins over $Q$, and $Q$ always loses. In this case, if the dealer has the $Q$, then he cannot win, whatever the player does. Figure 6.1 illustrates the idea: if we consider the strategy where the only specified move says that the player cheats (in bold), any completion of this strategy will go through the two bottom-left-most states, in which the player wins (the moves of all possible extensions are in dashed lines). Thus, it is not necessary to investigate any extension of this particular strategy since we know that it will be surely winning.

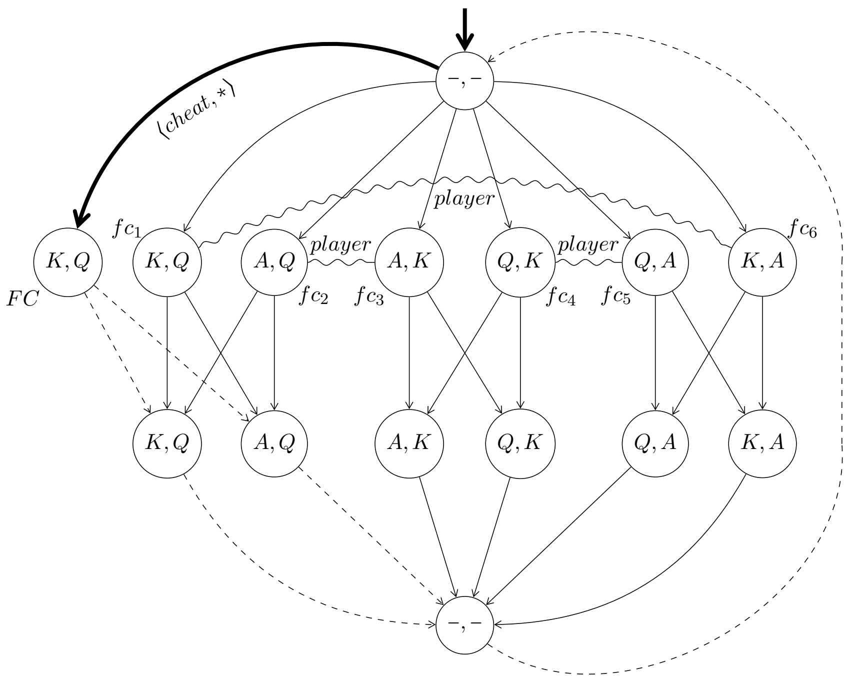


Figure 6.1: The graph of the repeated card game with a cheating player. The bold arrow shows an incomplete partial strategy, the dashed one its extensions.

This section presents the approach and how to extend it with pre-filtering. Section 6.1.1 presents algorithms to check that any extension of a given strategy is winning. Section 6.1.2 describes the actual model-checking algorithm that alternates between building and checking strategies. Section 6.1.3 extends it to take pre-filtered moves into account. Section 6.1.4 discusses some optimizations, and Section 6.1.5 describes the implementation with PyNuSMV.

### 6.1.1 Checking all extensions of partial strategies

Given a closed set of Γ-moves $M_\Gamma$, it is possible to compute the set of states from which all outcomes through $M_\Gamma$ satisfy a given objective. These new algorithms are similar to the *filter* ones of Section 5.1.

First, let

$$Pre_{\mathbf{E}}(Q', M_\Gamma) = \left\{ q' \in Q \;\middle|\; \begin{array}{l} \exists \langle q, a_\Gamma \rangle \in M_\Gamma \text{ s.t. } \exists a \in E_{Ag}(q) \text{ s.t.} \\ q = q' \wedge a_\Gamma \sqsubseteq a \wedge \delta(q,a) \in Q' \end{array} \right\}.$$

Intuitively, given a subset of states $Q'$ and a set of Γ-moves $M_\Gamma$, $Pre_{\mathbf{E}}(Q', M_\Gamma)$ returns the set of states $q$ such that there exists a move in $M_\Gamma$ for $q$ that can reach a state of $Q'$ in one step. The difference between $Pre_{\mathbf{E}}(Q', M_\Gamma)$ and $Pre_{\langle\!\langle \Gamma \rangle\!\rangle}(Q', M_\Gamma)$ is that a state $q$ is in the former if it has *one* successor in $Q'$ through some move of $M_\Gamma$, while $q$ is in the latter if *all* its successors through some move of $M_\Gamma$ are in $Q'$.

From the $Pre_{\mathbf{E}}$ function, we can define the function

$$Reach_{\mathbf{E}}(Q_1, Q_2, M_\Gamma) = \mu Q'.\ Q_2 \cup \left(Q_1 \cap Pre_{\mathbf{E}}(Q', M_\Gamma)\right).$$

Given two subsets of states $Q_1$ and $Q_2$, and a closed subset of Γ-moves $M_\Gamma$, $Reach_{\mathbf{E}}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ returns the states of $M_\Gamma|_Q$ from which there is, in the outcomes of $M_\Gamma$, a path that reaches a state of $Q_2$ through states of $Q_1$.

From the two functions above, we can define the function

$$Fair_{\mathbf{E}}(M_\Gamma) = \nu Q'.\ \bigcap_{fc \in FC} Pre_{\mathbf{E}}\left(Reach_{\mathbf{E}}(Q, Q' \cap fc, M_\Gamma), M_\Gamma\right).$$

Given a closed set of Γ-moves $M_\Gamma$, $Fair_{\mathbf{E}}(M_\Gamma) \cap M_\Gamma|_Q$ returns the set of states of $M_\Gamma|_Q$ from which there is a *fair* path in the outcomes of $M_\Gamma$.

Thanks to these functions, we can define three new $filter_{\mathbf{E}}$ algorithms. The first one is defined as

$$filter_{\mathbf{EX}}(Q', M_\Gamma) = Pre_{\mathbf{E}}(Q' \cap Fair_{\mathbf{E}}(M_\Gamma), M_\Gamma).$$

Given a group of agents Γ, a subset of states $Q'$ and a closed set of Γ-moves $M_\Gamma$, $filter_{\mathbf{EX}}(Q', M_\Gamma) \cap M_\Gamma|_Q$ returns the states $q \in M_\Gamma|_Q$ such that there is, in the outcomes of $M_\Gamma$, a fair path starting in $q$ with its second state in $Q'$.

The second $filter_{\mathbf{E}}$ algorithm is defined as

$$filter_{\mathbf{EU}}(Q_1, Q_2, M_\Gamma) = Reach_{\mathbf{E}}(Q_1, Q_{2,F}, M_\Gamma),$$

where

$$Q_{2,F} = Q_2 \cap Fair_{\mathbf{E}}(M_\Gamma).$$

Given a group of agents $\Gamma$, two subsets of states $Q_1$ and $Q_2$, and a closed set of $\Gamma$-moves $M_\Gamma$, $filter_{\mathbf{EU}}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ returns the subset of states $q \in M_\Gamma|_Q$ such that there is, in the outcomes of $M_\Gamma$, a fair path starting in $q$ and reaching a state of $Q_2$ through states of $Q_1$.

Finally, the third $filter_{\mathbf{E}}$ algorithm is defined as

$$\begin{aligned}&filter_{\mathbf{EW}}(Q_1, Q_2, M_\Gamma) = \\ &\nu Q'.\ Q_{2,F} \cup (Q_1 \cap \\ &\bigcap_{fc \in FC} Pre_{\mathbf{E}}(Reach_{\mathbf{E}}(Q_1, Q_{2,F} \cup (Q' \cap fc), M_\Gamma), M_\Gamma)),\end{aligned}$$

where

$$Q_{2,F} = Q_2 \cap Fair_{\mathbf{E}}(M_\Gamma).$$

Given a group of agents $\Gamma$, two subsets of states $Q_1$ and $Q_2$, and a closed set of $\Gamma$-moves $M_\Gamma$, $filter_{\mathbf{EW}}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ returns the subset of states $q \in M_\Gamma|_Q$ such that there is, in the outcomes of $M_\Gamma$, a fair path starting in $q$, that reaches a state of $Q_2$ through states of $Q_1$, or that stays in states of $Q_1$ forever. These algorithms directly derive from the Fair $CTL$ ones presented in Section 2.2.3, as $Pre_{\mathbf{E}}$ corresponds to $Pre$. They are only slightly adapted to take $M_\Gamma$ into account instead of the whole system.

Thanks to these $filter_{\mathbf{E}}$ algorithms, we can compute the states for which all paths in the outcomes of $M_\Gamma$ satisfy a given objective. Indeed, let

$$\begin{aligned}Pre_{\mathbf{A}}(Q', M_\Gamma) &= \overline{Pre_{\mathbf{E}}(\overline{Q'}, M_\Gamma)} \\ &= \left\{ q' \in Q \,\middle|\, \begin{array}{l} \forall \langle q, a_\Gamma \rangle \in M_\Gamma, \forall a \in E_{Ag}(q), \\ q = q' \wedge a_\Gamma \sqsubseteq a \implies \delta(q,a) \in Q' \end{array} \right\}.\end{aligned} \tag{6.1}$$

Intuitively, $Pre_{\mathbf{A}}(Q', M_\Gamma) \cap M_\Gamma|_Q$ are the states of $M_\Gamma|_Q$ such that all successors through a move of $M_\Gamma$ are in $Q'$.

Also, let

$$Stay_{\mathbf{A}}(Q_1, Q_2, M_\Gamma) = \nu Q'.\ Q_2 \cup \big(Q_1 \cap Pre_{\mathbf{A}}(Q', M_\Gamma)\big).$$

$Stay_{\mathbf{A}}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ are the states $q$ of $M_\Gamma|_Q$ such that all paths through $M_\Gamma$ from $q$ stay in $Q_1$ or reach $Q_2$ through $Q_1$.

Finally, let

$$\begin{aligned} NFair_{\mathbf{A}}(M_\Gamma) &= \overline{Fair_{\mathbf{E}}(M_\Gamma)} \\ &= \mu Q'.\ \bigcup_{fc \in FC} Pre_{\mathbf{A}}\left(Stay_{\mathbf{A}}(Q' \cup \overline{fc}, \varnothing, M_\Gamma), M_\Gamma\right). \end{aligned}$$

$NFair_{\mathbf{A}}(M_\Gamma) \cap M_\Gamma|_Q$ is the set of states $q$ of $M_\Gamma|_Q$ such that all paths through $M_\Gamma$ from $q$ are unfair.

With these three functions, we can define

$$\begin{aligned} filter_{\mathbf{AX}}(Q', M_\Gamma) &= Q \backslash filter_{\mathbf{EX}}(\overline{Q'}, M_\Gamma) \\ &= Pre_{\mathbf{A}}(Q' \cup NFair_{\mathbf{A}}(M_\Gamma), M_\Gamma). \end{aligned}$$

$filter_{\mathbf{AX}}(Q', M_\Gamma) \cap M_\Gamma|_Q$ computes the set of states $q \in M_\Gamma|_Q$ such that all fair paths in the outcomes of $M_\Gamma$ from $q$ have their second state in $Q'$.

Second, let

$$\begin{aligned} & filter_{\mathbf{AU}}(Q_1, Q_2, M_\Gamma) = Q \backslash filter_{\mathbf{EW}}(\overline{Q_2}, \overline{Q_1} \cap \overline{Q_2}, M_\Gamma) \\ & = \mu Q'.\ Q_{1,2,N} \cap (Q_2 \cup \\ & \bigcup_{fc \in FC} Pre_{\mathbf{A}}(Stay_{\mathbf{A}}\left(Q_{1,2,N} \cap (Q' \cup \overline{fc}), Q_2 \cap (Q' \cup \overline{fc}), M_\Gamma\right), M_\Gamma)), \end{aligned}$$

where

$$Q_{1,2,N} = Q_1 \cup Q_2 \cup NFair_{\mathbf{A}}(M_\Gamma).$$

$filter_{\mathbf{AU}}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ computes the set of states $q \in M_\Gamma|_Q$ such that all fair paths in the outcomes of $M_\Gamma$ from $q$ reach a state of $Q_2$ through states of $Q_1$.

Third, let

$$\begin{aligned} filter_{\mathbf{AW}}(Q_1, Q_2, M_\Gamma) &= Q \backslash filter_{\mathbf{EU}}(\overline{Q_2}, \overline{Q_1} \cap \overline{Q_2}, M_\Gamma) \\ &= Stay_{\mathbf{A}}(Q_1 \cup Q_2 \cup NFair_{\mathbf{A}}(M_\Gamma), Q_2, M_\Gamma). \end{aligned}$$

$filter_{\mathbf{AW}}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ is the set of states $q \in M_\Gamma|_Q$ such that all fair paths from $q$ in the outcomes of $M_\Gamma$ reach $Q_2$ through $Q_1$, or stay in $Q_1$ forever.

These three functions can be used to check whether all extensions of one partial strategy are winning. Given a closed set of moves $M_\Gamma$, these three functions return the states for which all outcomes in $M_\Gamma$ satisfy the corresponding objective.

### 6.1.2 The model-checking algorithm

To check that any extension of an incomplete partial strategy is winning for a particular objective, we need to complete this strategy up to a closed set of moves. This can be done with Algorithm 6.1. It computes the set of moves reachable from $M_\Gamma$ that are compatible with it by accumulating in $M'_\Gamma$ the moves reachable in one step from $M'_\Gamma$, and compatible with $M_\Gamma$.

**Algorithm 6.1:** $Complete(M_\Gamma)$

```
**Data**: $M_\Gamma$ a set of non-$\Gamma$-conflicting $\Gamma$-moves.
**Result**: The set of $\Gamma$-moves reachable from some move of $M_\Gamma$ and
          compatible with $M_\Gamma$.
$M'_\Gamma = M_\Gamma$
$new\_states = Post(Q, M'_\Gamma) \backslash M'_\Gamma|_Q$
$new\_moves = Compatible(new\_states, M_\Gamma)$
**while** $new\_moves \neq \varnothing$ **do**
    $M'_\Gamma = M'_\Gamma \cup new\_moves$
    $new\_states = Post(Q, M'_\Gamma) \backslash M'_\Gamma|_Q$
    $new\_moves = Compatible(new\_states, M_\Gamma)$
**return** $M'_\Gamma$
```

By completing a partial strategy $f_\Gamma$ and by using the three new $filter_{\mathbf{A}}$ functions, we can compute the set of states for which all the extensions of $f_\Gamma$ are winning for the corresponding objective. Indeed, any partial strategy defined from the completed moves would have as outcomes a subset of the paths enforced by these completed moves. Thus, if all paths of the completed moves satisfy the objective, all outcomes of a particular partial strategy extending $f_\Gamma$ satisfy the objective and the strategy is winning.

The $eval^{alt}_{ATLK_{irF}}$ algorithm (divided into three parts, Algorithms 6.2, 6.3 and 6.4) uses $filter_{\mathbf{A}}$, $filter$, and $Complete$ to compute the set of states for which an incomplete partial strategy can be extended into a winning partial strategy. More precisely, given a set of states $Q'$, a strategic formula $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$ and a (not necessarily complete) partial strategy $f_\Gamma$, $eval^{alt}_{irF}(Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi, f_\Gamma)$ returns the set of states $q \in Q'$ such that there exists an extension of $f_\Gamma$ that is winning for $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$ in all states indistinguishable from $q$.

First, Algorithm 6.2 completes $f_\Gamma$ (Line 1). This results in a set of moves $cf_\Gamma$ that does not represent a particular uniform partial strategy, but that rather defines a part of the checked structure. Second, it

**Algorithm 6.2:** $eval^{alt}_{ATLK_{irF}}(Q', \langle\langle\Gamma\rangle\rangle\ \psi, f_\Gamma)$ (part 1)

```
Data: $Q' \subseteq Q$ a subset of states such that $[Q']^E_\Gamma = Q'$, $\langle\langle\Gamma\rangle\rangle\ \psi$ an
      $ATLK_{irF}$ formula, $f_\Gamma$ an incomplete partial strategy such
      that $Q' \subseteq f_\Gamma|_Q$.
Result: The set of states $q \in Q'$ such that there exists an
        extension of $f_\Gamma$ winning for $\langle\langle\Gamma\rangle\rangle\ \psi$ in all states
        indistinguishable from $q$.
1  $cf_\Gamma = Complete(f_\Gamma)$
2  case $\psi = \boldsymbol{X}\ \phi'$
       $Q'' = eval^{Early}_{ATLK_{irF}}(S, Post([Q']^E_\Gamma, cf_\Gamma), \phi')$
       $notlose = Q' \cap filter_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(\Gamma, Q'', cf_\Gamma)$
   case $\psi = \phi_1\ \boldsymbol{U}\ \phi_2$
       $Q_1 = eval^{Early}_{ATLK_{irF}}(S, cf_\Gamma|_Q, \phi_1)$
       $Q_2 = eval^{Early}_{ATLK_{irF}}(S, cf_\Gamma|_Q, \phi_2)$
       $notlose = Q' \cap filter_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(\Gamma, Q_1, Q_2, cf_\Gamma)$
   case $\psi = \phi_1\ \boldsymbol{W}\ \phi_2$
       $Q_1 = eval^{Early}_{ATLK_{irF}}(S, cf_\Gamma|_Q, \phi_1)$
       $Q_2 = eval^{Early}_{ATLK_{irF}}(S, cf_\Gamma|_Q, \phi_2)$
12     $notlose = Q' \cap filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(\Gamma, Q_1, Q_2, cf_\Gamma)$
13 $lose = Q' \backslash \{q \in notlose \mid \forall ag \in \Gamma, \forall q' \in Q, q' \sim_{ag} q \implies q' \in notlose\}$
```

**Algorithm 6.3:** $eval^{alt}_{ATLK_{irF}}(Q', \langle\langle\Gamma\rangle\rangle\ \psi, f_\Gamma)$ (part 2)

```
12 case $\psi = \boldsymbol{X}\ \phi'$
       $win = Q' \cap filter_{\mathbf{AX}}(Q'', cf_\Gamma)$
   case $\psi = \phi_1\ \boldsymbol{U}\ \phi_2$
       $win = Q' \cap filter_{\mathbf{AU}}(Q_1, Q_2, cf_\Gamma)$
   case $\psi = \phi_1\ \boldsymbol{W}\ \phi_2$
17     $win = Q' \cap filter_{\mathbf{AW}}(Q_1, Q_2, cf_\Gamma)$
18 $win = \{q \in win \mid \forall ag \in \Gamma, \forall q' \in Q, q' \sim_{ag} q \implies q' \in win\}$
```

computes the set of states *notlose* for which there exists a general strategy in the completion of $f_\Gamma$ to win the objective (Lines 2 to 12). This is done by using the $filter$ algorithms. So, the set of states *lose* is the set of states for which there is no general strategy in the completion of $f_\Gamma$ to win the objective from some equivalent state (Line 13); this implies that, in these states, there exists no winning extension of $f_\Gamma$ for all their

**Algorithm 6.4:** $eval^{alt}_{ATLK_{irF}}(Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi, f_\Gamma)$ (part 3)

```
22 if $Q' \backslash (lose \cup win) = \varnothing$ then return $win$
23 else
       $new = Post(Q, f_\Gamma) \backslash f_\Gamma|_Q$
       $compatible = Compatible(new, f_\Gamma)$
       $newstrats = Split(\Gamma, compatible)$
       for $f'_\Gamma \in newstrats$ do
           $win = win \cup eval^{alt}_{irF}(Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi, f_\Gamma \cup f'_\Gamma)$
29     return $win$
```

indistinguishable states.

Then, Algorithm 6.3 computes the set of states for which all the outcomes in the completion of $f_\Gamma$ satisfy the objective (Lines 12 to 17), that is, the states for which any extension of $f_\Gamma$ would be winning. After Line 18, *win* is thus the set of states for which all outcomes from all indistinguishable states satisfy the objective. So the algorithm computed at Line 18 the set of states that are surely losing (in *lose*), or surely winning (in *win*), considering all compatible extensions of $f_\Gamma$. If these two sets cover the whole set $Q'$, the search is done and the algorithm can return the set of winning states (Line 22 of Algorithm 6.4). Otherwise, there are some states in $Q'$ for which $cf_\Gamma$ is not winning nor losing. Thus $f_\Gamma$ cannot be closed (and cannot be a complete partial strategy adequate for $Q'$). As $f_\Gamma$ is not closed, new states can be reached, leading to extensions of $f_\Gamma$ that the algorithm can recursively check with $eval^{alt}_{ATLK_{irF}}$ (Lines 23 to 29).

Given a set of states $Q'$ and a formula $\phi$, the $eval^{alt}_{ATLK_{irF}}$ algorithm can be used to compute the set of states of $Q'$ satisfying $\phi$. We can start with small incomplete partial strategies and check whether their extensions can be winning with $eval^{alt}_{ATLK_{irF}}$. More precisely, Algorithm 6.5 uses $eval^{alt}_{ATLK_{irF}}$ to compute the set of states of $Q'$ satisfying $\phi$. It splits the moves enabled in the states indistinguishable from $Q'$ and calls $eval^{alt}_{ATLK_{irF}}$ on these partial strategies (Lines 4 and 5). At the end, *sat* contains the set of states of $Q'$ satisfying $\phi$.

Algorithm 6.5 only presents the strategic cases. The model checking for the propositional cases (*true*, $p$, $\neg$ and $\vee$), the temporal ones (**EX**, **EU** and **EW**) and the knowledge ones (**K**, **E**, **D** and **C**) is performed in the standard way, as exposed in Section 2.2.3. The correctness of this approach is proved in Section A.6.1 of Appendix A.

**Algorithm 6.5:** $eval^{Early}_{ATLK_{irF}}(S, Q', \phi)$

```
Data: $S$ an iCGSf, $Q' \subseteq Q$ a subset of states, $\phi$ an $ATLK_{irF}$
      formula.
Result: The states of $Q'$ satisfying $\phi$.
case $\phi \in \{\langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi', \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2], \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]\}$
    $Q'' = [Q']^E_\Gamma$
    $sat = \{\}$
4   for $f_\Gamma \in Split(\Gamma, Moves_\Gamma(Q''))$ do
5       $win = eval^{alt}_{ATLK_{irF}}(Q'', \langle\langle\Gamma\rangle\rangle\ \psi, f_\Gamma)$
        $sat = sat \cup (win \cap Q')$
    return $sat$
case $\phi = ...$ // Cases for the other operators are standard
```

### 6.1.3 Pre-filtering

As for the naive and partial approaches, we can pre-filter the original structure to only consider the moves that can be part of a winning strategy. The $eval^{alt,PF}_{ATLK_{irF}}$ algorithm—presented in Algorithms 6.6, 6.7 and 6.8—takes an additional $filtered$ argument to limit its search to filtered parts of partial strategies.

More precisely, the first two parts correspond to the first two parts of $eval^{alt}_{ATLK_{irF}}$ algorithm: first they complete the given strategy, then compute the sets *lose* and *win* corresponding to states for which the completion is either surely losing, or surely winning, respectively (in Algorithms 6.6 and 6.7, respectively).

The third part of $eval^{alt,PF}_{ATLK_{irF}}$ differs from the $eval^{alt}_{ATLK_{irF}}$ one: instead of computing the moves reached in one step from $f_\Gamma$, it limits them to the ones of its $filtered$ argument. This is similar to the $ReachSplit^{PF}$ algorithm, and allows to consider filtered partial strategies only. The rest of the algorithm behaves as $eval^{alt}_{ATLK_{irF}}$: it recursively calls itself with $f_\Gamma$ extended with each newly reached subset of non-conflicting moves, and accumulates in *win* the states for which there exists a winning extension.

The $eval^{Early}_{ATLK_{irF}}$ is also extended to take pre-filtered moves into account. Algorithm 6.9 presents the extended version, $eval^{Early,PF}_{ATLK_{irF}}$. It first computes the set of pre-filtered moves $filtered$ (Lines 4 to 14), then it checks if there are still some states for which there could be some winning strategies. If there are some, then it explores, with the $eval^{alt,PF}_{ATLK_{irF}}$, the partial strategies in the remaining states, limited to the pre-filtered moves.

**Algorithm 6.6:** $eval^{alt,PF}_{ATLK_{irF}}(Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi, f_\Gamma, filtered)$ (part 1)

```
Data: $Q' \subseteq Q$ a subset of states such that $[Q']^E_\Gamma = Q'$, $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$ an
      $ATLK_{irF}$ formula with top-level operator $op$, $filtered$ a
      set of $\Gamma$-moves such that $filtered = filter^M_{op}(Q_1, Q_2, E_\Gamma)$,
      with $Q_i$ the states satisfying the $i$th sub-formula of $\psi$, and
      $f_\Gamma$ an incomplete partial strategy such that $Q' \subseteq f_\Gamma|_Q$ and
      $f_\Gamma \subseteq filtered$.
Result: The set of states $q \in Q'$ for which there exists an
        extension of $f_\Gamma$ winning for $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$ in all states
        indistinguishable from $q$.
$cf_\Gamma = Complete(f_\Gamma)$
case $\psi = \boldsymbol{X}\ \phi'$
    $Q'' = eval^{Early,PF}_{ATLK_{irF}}(S, Post([Q']^E_\Gamma, cf_\Gamma), \phi')$
    $notlose = Q' \cap filter_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(\Gamma, Q'', cf_\Gamma)$
case $\psi = \phi_1\ \boldsymbol{U}\ \phi_2$
    $Q_1 = eval^{Early,PF}_{ATLK_{irF}}(S, cf_\Gamma|_Q, \phi_1)$
    $Q_2 = eval^{Early,PF}_{ATLK_{irF}}(S, cf_\Gamma|_Q, \phi_2)$
    $notlose = Q' \cap filter_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(\Gamma, Q_1, Q_2, cf_\Gamma)$
case $\psi = \phi_1\ \boldsymbol{W}\ \phi_2$
    $Q_1 = eval^{Early,PF}_{ATLK_{irF}}(S, cf_\Gamma|_Q, \phi_1)$
    $Q_2 = eval^{Early,PF}_{ATLK_{irF}}(S, cf_\Gamma|_Q, \phi_2)$
    $notlose = Q' \cap filter_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}(\Gamma, Q_1, Q_2, cf_\Gamma)$
$lose = Q' \setminus \{q \in notlose \mid \forall ag \in \Gamma, \forall q' \in Q, q' \sim_{ag} q \implies q' \in notlose\}$
```

**Algorithm 6.7:** $eval^{alt,PF}_{ATLK_{irF}}(Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi, f_\Gamma, filtered)$ (part 2)

```
   case $\psi = \boldsymbol{X}\ \phi'$
       $win = Q' \cap filter_{\mathbf{AX}}(Q'', cf_\Gamma)$
   case $\psi = \phi_1\ \boldsymbol{U}\ \phi_2$
       $win = Q' \cap filter_{\mathbf{AU}}(Q_1, Q_2, cf_\Gamma)$
   case $\psi = \phi_1\ \boldsymbol{W}\ \phi_2$
       $win = Q' \cap filter_{\mathbf{AW}}(Q_1, Q_2, cf_\Gamma)$
18 $win = \{q \in win \mid \forall ag \in \Gamma, \forall q' \in Q, q' \sim_{ag} q \implies q' \in win\}$
```

Algorithm 6.9 only presents the strategic cases. The model checking for the propositional cases (*true*, $p$, $\neg$ and $\vee$), the temporal ones (**EX**, **EU** and **EW**) and the knowledge ones (**K**, **E**, **D** and **C**) is performed

**Algorithm 6.8:** $eval^{alt,PF}_{ATLK_{irF}}(Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi, f_\Gamma, filtered)$ (part 3)

```
if $Q' \backslash (lose \cup win) = \varnothing$ then return $win$
else
    $new\_states = Post(Q, f_\Gamma) \backslash f_\Gamma|_Q$
    $new\_moves = \{\langle q, a_\Gamma\rangle \in filtered \mid q \in new\_states\}$
    $compatible = Compatible^M(new\_moves, f_\Gamma)$
    if $compatible = \varnothing$ then return $Q' \backslash lose$
    else
        $newstrats = Split(\Gamma, compatible)$
        for $f'_\Gamma \in newstrats$ do
            $win = win \cup eval^{alt,PF}_{irF}(Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi, f_\Gamma \cup f'_\Gamma, filtered)$
        return $win$
```

in the standard way, as exposed in Section 2.2.3. The correctness of this approach is proved in Section A.6.2 of Appendix A.

### 6.1.4 Optimizations

Similarly to the *partial* approaches, the *early* approaches can be extended to integrate early termination and caching. Early termination can be embedded in the $eval^{alt}_{ATLK_{irF}}$ and $eval^{Early}_{ATLK_{irF}}$ algorithms. There are several places where we can check whether we already found a winning strategy for all states of interest. The modifications are given in Algorithms 6.10 and 6.11: Algorithm 6.10 shows the third part of $eval^{alt}_{ATLK_{irF}}$ that reduces the set of states of interest to the states for which no decision can be made (Line 27), still reduces this set of states of interest when extended strategies are found to be winning (Line 30), and stops as soon as all states are covered (Line 31).

$eval^{Early}_{ATLK_{irF}}$ can also be extended to use early termination. Algorithm 6.11 keeps track of states for which a winning strategy has been found (Line 7), and stops as soon as all states are covered (Line 8).

$eval^{Early}_{ATLK_{irF}}$ can also be extended to cache already known results. This is achieved with the same extension as for the $eval^{Partial}_{ATLK_{irF}}$ algorithm, as shown in Algorithm 6.12. The $eval^{alt}_{ATLK_{irF}}$ algorithm must call the $eval^{Early,Cached}_{ATLK_{irF}}$ algorithm on sub-formulas, instead of vanilla $eval^{Early}_{ATLK_{irF}}$.

**Algorithm 6.9:** $eval^{Early,PF}_{ATLK_{irF}}(S, Q', \phi)$

```
Data: $S$ an iCGSf, $Q' \subseteq Q$ a subset of states, $\phi$ an $ATLK_{irF}$ formula.
Result: The states of $Q'$ satisfying $\phi$.
case $\phi \in \{\langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi', \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2], \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]\}$
    $Q'' = [Q']^E_\Gamma$
    $sat = \{\}$
4   case $\phi = \langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi'$
        $Q''' = eval^{Early,PF}_{ATLK_{irF}}(S, Post(Q'', E_\Gamma), \phi')$
        $filtered = filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(\Gamma, Q''', E_\Gamma)$
    case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2]$
        $Q_1 = eval^{Early,PF}_{ATLK_{irF}}(S, Q, \phi_1)$
        $Q_2 = eval^{Early,PF}_{ATLK_{irF}}(S, Q, \phi_2)$
        $filtered = filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(\Gamma, Q_1, Q_2, E_\Gamma)$
    case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]$
        $Q_1 = eval^{Early,PF}_{ATLK_{irF}}(S, Q, \phi_1)$
        $Q_2 = eval^{Early,PF}_{ATLK_{irF}}(S, Q, \phi_2)$
14      $filtered = filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(\Gamma, Q_1, Q_2, E_\Gamma)$
15  $Q'' = Q'' \cap filtered|_Q$
    if $Q'' = \varnothing$ then
17      return $\varnothing$
    for $f_\Gamma \in Split(\Gamma, Moves_\Gamma(Q'') \cap filtered)$ do
        $win = eval^{alt,PF}_{ATLK_{irF}}(Q'', \langle\langle\Gamma\rangle\rangle\ \psi, f_\Gamma, filtered)$
        $sat = sat \cup (win \cap Q')$
    return $sat$
case $\phi = ...$ // Cases for the other operators are standard
```

### 6.1.5 Implementation

The *early* approaches mostly reuse the basic algorithms that the *partial* ones use. The implementation thus suffers from the same problem (see Section 5.7). The only additional function that needs to be implemented is the $Pre_{\mathbf{A}}$ function of Equation 6.1:

$$Pre_{\mathbf{A}}(Q', M_\Gamma) = \left\{ q' \in Q \;\middle|\; \begin{array}{l} \forall \langle q, a_\Gamma\rangle \in M_\Gamma, \forall a \in E_{Ag}(q), \\ q = q' \wedge a_\Gamma \sqsubseteq a \implies \delta(q, a) \in Q' \end{array} \right\}.$$

**Algorithm 6.10:** $eval^{alt,Term}_{ATLK_{irF}}(Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi, f_\Gamma)$ (part 3)

```
   if $Q' \backslash (lose \cup win) = \varnothing$ then return $win$
   else
       $new = Post(Q, f_\Gamma) \backslash f_\Gamma|_Q$
       $compatible = Compatible(new, f_\Gamma)$
       $newstrats = Split(\Gamma, compatible)$
27     $Q' = Q' \backslash (lose \cup win)$
       for $f'_\Gamma \in newstrats$ do
           $win = win \cup eval^{alt}_{irF}(Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi, f_\Gamma \cup f'_\Gamma)$
30         $Q' = Q' \backslash win$
31         if $Q' = \varnothing$ then break
       return $win$
```

**Algorithm 6.11:** $eval^{Early,Term}_{ATLK_{irF}}(S, Q', \phi)$

```
   Data: $Q' \subseteq Q$ a subset of states, $\phi$ an $ATLK_{irF}$ formula.
   Result: The states of $Q'$ satisfying $\phi$.
   case $\phi \in \{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{X}\ \phi', \langle\!\langle\Gamma\rangle\!\rangle[\phi_1\ \boldsymbol{U}\ \phi_2], \langle\!\langle\Gamma\rangle\!\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]\}$
       $Q'' = [Q']^E_\Gamma$
       $sat = \{\}$
       for $f_\Gamma \in Split(\Gamma, Moves(Q''))$ do
           $win = eval^{alt}_{ATLK_{irF}}(Q'', \langle\!\langle\Gamma\rangle\!\rangle\ \psi, f_\Gamma)$
           $sat = sat \cup (win \cap Q')$
7          $Q'' = Q'' \backslash win$
8          if $Q'' = \varnothing$ then break
       return $sat$
   case $\phi = ...$ // Cases for the other operators are standard
```

It is implemented in the same way as the $Pre_{\langle\!\langle\Gamma\rangle\!\rangle}$ function:

$$Pre_{\mathbf{A}}(Q', M_\Gamma) = \overline{\exists a.(Pre^M(\overline{Q'}) \cap M_\Gamma)}.$$

$Pre^M(\overline{Q'})$ is the set of moves that can reach a state of $\overline{Q'}$ in one step; $Pre^M(\overline{Q'}) \cap M_\Gamma$ is thus the set of moves of $M_\Gamma$ that can reach a state of $\overline{Q'}$ in one step, and $\exists a.(Pre^M(\overline{Q'}) \cap M_\Gamma)$ is the set of states for which there exists a move in $M_\Gamma$ to reach a step of $\overline{Q'}$ in one step. $Pre_{\mathbf{A}}(Q', M_\Gamma)$ is the set of states for which all moves of $M_\Gamma$ reach only states of $Q'$ in one step.

```
Algorithm 6.12: $eval^{Early,Cached}_{ATLK_{irF}}(S, Q', \phi)$
Data: $S$ an iCGSf, $Q' \subseteq Q$ a subset of states, $\phi$ an $ATLK_{irF}$
      formula.
Result: The set of states of $Q'$ satisfying $\phi$.
$sat = cache_{sat}[\langle S, \phi \rangle]$
$unsat = cache_{unsat}[\langle S, \phi \rangle]$
$unknown = Q' \backslash (sat \cup unsat)$
$newsat = eval^{Early}_{ATLK_{irF}}(S, unknown, \phi)$
$cache_{sat}[\langle S, \phi \rangle] = sat \cup newsat$
$cache_{unsat}[\langle S, \phi \rangle] = unsat \cup (unknown \backslash newsat)$
return $(sat \cup newsat) \cap Q'$
```

## 6.2 A fully symbolic approach

This approach is based on the work of Huang and van der Meyden presented in [HvdM14b]. In the sequel, it is called the *symbolic* approach. The main idea of Huang and van der Meyden is to design a fully symbolic model checking algorithm by encoding the uniform strategies of the agents of the system into a derived structure and to perform custom fixpoint computations on it. More precisely, let $S = \langle Ag, Q, Q_0, Act, e, \delta, V, \sim, FC \rangle$ be an iCGSf, we can build a derived structure

$$EncStrats(S) = \langle Ag, Q^{ES}, Q_0^{ES}, Act, e^{ES}, \delta^{ES}, V^{ES}, \sim^{ES}, FC^{ES} \rangle$$

such that

- $Q^{ES} = Q \times \prod_{ag \in Ag} F^u_{ag}$, where $F^u_{ag}$ is the set of uniform strategies of agent $ag$; given a state $q^{ES} = \langle q, f_{ag_1}, ..., f_{ag_{|Ag|}} \rangle \in Q^{ES}$, we write $state(q^{ES})$ for $q$, and $strategy(ag, q^{ES})$ for $f_{ag}$, for any agent $ag$ of $Ag$;
- $Q_0^{ES} = \{q^{ES} \in Q^{ES} \mid state(q^{ES}) \in Q_0\}$;
- $e^{ES}_{ag}(q^{ES}) = e_{ag}(state(q^{ES}))$;
- $\delta^{ES}(q^{ES}, a) =$ $\langle \delta(state(q^{ES}), a), strategy(ag_1, q^{ES}), ..., strategy(ag_{|Ag|}, q^{ES}) \rangle$;
- $V^{ES}(q^{ES}) = V(state(q^{ES}))$;
- $q_1^{ES} \sim^{ES}_{ag} q_2^{ES}$ iff $state(q_1^{ES}) \sim_{ag} state(q_2^{ES})$;

- $FC^{ES} = \{\{q^{ES} \in Q^{ES} \mid state(q^{ES}) \in fc\} \mid fc \in FC\}$.

Intuitively, we encode in the states of $EncStrats(S)$ the uniform strategies of all agents of $S$. More precisely, the states of the derived structure are tuples composed of one original state and a strategy for each agent, the transition function $\delta$ is defined such that strategies stay the same in states and their successors, and the other elements of $EncStrats(S)$ are lifted for the derived states.

Given a set of states $Q' \subseteq Q$ and a set of strategies $F'_\Gamma$ for a group of agents $\Gamma$, we write $Q' \times F'_\Gamma$ for the set of tuples composed of elements of $Q'$, and elements of $F'_\Gamma$, that is, the set

$$\{\langle q, f_{ag_1}, ..., f_{ag_{|\Gamma|}}\rangle \mid q \in Q' \wedge \langle f_{ag_1}, ..., f_{ag_{|\Gamma|}}\rangle \in F'_\Gamma\}.$$

Furthermore, given a state $q^{ES} \in Q^{ES}$, we write $strategy(\Gamma, q^{ES})$ for the strategies for the group $\Gamma$ stored in $q^{ES}$.

Thanks to the derived structure $EncStrats(S)$, it is possible to compute the set of states of $S$ satisfying a given strategic formula. Let

$$Pre_{str}(\Gamma, Q') = \left\{ q \in Q^{ES} \;\middle|\; \begin{array}{l} \forall a \in E^{ES}_{Ag}(q), \\ (\forall ag \in \Gamma, \\ q \in dom(strategy(ag, q)) \implies \\ a(ag) = strategy(ag, q)(state(q))) \\ \implies \delta^{ES}(q, a) \in Q' \end{array} \right\}.$$

Intuitively, given a set of agents $\Gamma \subseteq Ag$ and a set of states $Q' \subseteq Q^{ES}$, $Pre_{str}(\Gamma, Q')$ returns the set of states $q$ of $Q^{ES}$ such that $\Gamma$ can force to reach states of $Q'$ by playing the action specified by their strategies enclosed in $q$, if such an action is specified, or by playing any action otherwise.

This general definition of the $Pre_{str}$ function will be useful to define the extension of the approach with pre-filtering in the next section. But if the strategies are defined for all states of the original structure, as it is the case for $EncStrats(S)$, then the $Pre_{str}$ function is equivalent to the following:

$$Pre_{str}(\Gamma, Q') = \left\{ q \in Q^{ES} \;\middle|\; \begin{array}{l} \forall a \in E^{ES}_{Ag}(q), (\forall ag \in \Gamma, \\ a(ag) = strategy(ag, q)(state(q))) \\ \implies \delta^{ES}(q, a) \in Q' \end{array} \right\}.$$

Furthermore, let

$$EqQ(Q') = \{q_1^{ES} \in Q^{ES} \mid \exists q_2^{ES} \in Q' \text{ s.t. } state(q_1^{ES}) = state(q_2^{ES})\},$$

and

$$EquivQEqStr(\Gamma, Q') =$$
$$\left\{ q_1^{ES} \in Q^{ES} \middle| \forall q_2^{ES} \in Q^{ES}, \begin{pmatrix} \text{if } \exists ag \in \Gamma, q_1^{ES} \sim_{ag} q_2^{ES} \wedge \forall ag \in Ag, \\ strategy(ag, q_1^{ES}) = strategy(ag, q_2^{ES}) \\ \text{then } q_2^{ES} \in Q' \end{pmatrix} \right\}.$$

Intuitively, $EqQ(Q')$, $Q'$ being a subset of $Q^{ES}$, returns the set of states of $Q^{ES}$ for which there exists a state in $Q'$ sharing the same original state. Furthermore, $EquivQEqStr(\Gamma, Q')$ returns the set of states for which all derived states from which the original states are indistinguishable for some agent in $\Gamma$, and in which the strategies are the same, are in $Q'$.

From these functions, we can define a new $Stay_{str}$ function as

$$Stay_{str}(\Gamma, Q_1, Q_2) = \nu Q'.\ Q_2 \cup (Q_1 \cap Pre_{str}(\Gamma, Q')),$$

and a new $NFair_{str}$ function as

$$NFair_{str}(\Gamma) = \mu Q'.\ \bigcup_{fc \in FC} Pre_{str}(\Gamma, Stay_{str}(\Gamma, Q' \cup \overline{fc}, \varnothing)).$$

Intuitively, $NFair_{str}(\Gamma)$ returns the set of states from which $\Gamma$ can enforce only unfair paths by following the strategies encoded in these states. Using these functions, we can define the following ones, given $ESS = EncStrats(S)$, and given two subsets of states $Q_1^{ES}, Q_2^{ES} \subseteq Q^{ES}$:

$$filter^{ES}_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(ESS, Q_1^{ES}) = Pre_{str}(\Gamma, Q_1^{ES} \cup NFair_{str}(\Gamma))$$
$$filter^{ES}_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(ESS, Q_1^{ES}, Q_2^{ES}) =$$
$$\mu Q'.\ Q_{1,2,N} \cap (Q_2^{ES} \cup$$
$$\bigcup_{fc \in FC} Pre_{str}(\Gamma, Stay_{str}(\Gamma, Q_{1,2,N}^{ES} \cap (Q' \cup \overline{fc}), Q_2^{ES} \cap (Q' \cup \overline{fc}))))$$
$$filter^{ES}_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(ESS, Q_1^{ES}, Q_2^{ES}) = Stay_{str}(Q_{1,2,N}^{ES}, Q_2^{ES})$$

where

$$Q_{1,2,N} = Q_1^{ES} \cup Q_2^{ES} \cup NFair_{str}(\Gamma).$$

Assuming that, for $i \in \{1, 2\}$,

$$Q_i^{ES} = Q_i \times \prod_{ag \in Ag} F_{ag}^u,$$

then, intuitively, $filter^{ES}_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(ESS, Q_1^{ES})$ is the set of states in which $\Gamma$ can enforce that all fair paths have their second state in $Q_1^{ES}$ by following the strategies in these states. The result of $filter^{ES}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(ESS, Q_1^{ES}, Q_2^{ES})$ is the set of states in which $\Gamma$ can enforce that all fair paths reach a state of $Q_2^{ES}$ through states of $Q_1^{ES}$ by following their strategies encoded in the states. $filter^{ES}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}(ESS, Q_1^{ES}, Q_2^{ES})$ is the set of states in which $\Gamma$ can enforce that all fair paths reach a state of $Q_2^{ES}$ through states of $Q_1^{ES}$, or stay forever in $Q_1^{ES}$, by following their strategies encoded in the states.

Thanks to these new $filter^{ES}$ algorithms, we can define $eval^{Symbolic}_{ATLK_{irF}}$ as Algorithm 6.13.

```
Algorithm 6.13: $eval^{Symbolic}_{ATLK_{irF}}(ESS, \phi)$
Data: $ESS$ an iCGSf such that $ESS = EncStrats(S)$ for some
      iCGSf $S$, $\phi$ an $ATLK_{irF}$ formula.
Result: The states $Q_1^{ES} = Q_1 \times \prod_{ag \in Ag} F^u_{ag}$ of $ESS$, where $Q_1$ is
        the set of states of $S$ satisfying $\phi$.

case $\phi \in \{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{X}\ \phi', \langle\!\langle\Gamma\rangle\!\rangle[\phi_1\ \boldsymbol{U}\ \phi_2], \langle\!\langle\Gamma\rangle\!\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]\}$
    case $\phi = \langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{X}\ \phi_1$
        $Q_1^{ES} = eval^{Symbolic}_{ATLK_{irF}}(ESS, \phi_1)$
        $winning = filter^{ES}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(ESS, Q_1^{ES})$
    case $\phi = \langle\!\langle\Gamma\rangle\!\rangle[\phi_1\ \boldsymbol{U}\ \phi_2]$
        $Q_1^{ES} = eval^{Symbolic}_{ATLK_{irF}}(ESS, \phi_1)$
        $Q_2^{ES} = eval^{Symbolic}_{ATLK_{irF}}(ESS, \phi_2)$
        $winning = filter^{ES}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(ESS, Q_1^{ES}, Q_2^{ES})$
    case $\phi = \langle\!\langle\Gamma\rangle\!\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]$
        $Q_1^{ES} = eval^{Symbolic}_{ATLK_{irF}}(ESS, \phi_1)$
        $Q_2^{ES} = eval^{Symbolic}_{ATLK_{irF}}(ESS, \phi_2)$
        $winning = filter^{ES}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}(ESS, Q_1^{ES}, Q_2^{ES})$
13  return $EqQ(EquivQEqStr(\Gamma, winning))$
case $\phi = ...$ // Cases for the other operators are standard
```

$eval^{Symbolic}_{ATLK_{irF}}(ESS, \langle\!\langle\Gamma\rangle\!\rangle\ \psi)$ returns the set of states for which there exists a state with other strategies ($EqQ$), for which, in all equivalent states sharing the same strategies ($EquivQEqStr$), $\Gamma$ can enforce that all fair paths satisfy $\psi$. In other words, it computes the set of states satisfying $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$.

Algorithm 6.13 only presents the strategic cases. The model checking for the propositional cases ($true$, $p$, $\neg$ and $\vee$), the temporal ones (**EX**, **EU** and **EW**) and the knowledge ones (**K**, **E**, **D** and **C**) is performed in the standard way, as exposed in Section 2.2.3. The correctness of this algorithm is proved in Section A.7 of Appendix A.

### 6.2.1 Pre-filtering

Pre-filtering can be applied in the context of the symbolic approach. Before encoding the original structure $S$ into $EncStrats(S)$, we can rule out the set of moves that cannot be part of a winning strategy—thanks to the $filter^M$ algorithms—and only encode the sub-structure defined by these moves.

By analyzing the $eval^{Symbolic}_{ATLK_{irF}}$ algorithm and the functions it uses, we can see that, for a a given strategic formula $\langle\langle\Gamma\rangle\rangle\ \psi$, it is not necessary to encode the strategies of agents outside $\Gamma$; these strategies are never used and do not change the states $q$ for which there exists a derived state in $eval^{Symbolic}_{ATLK_{irF}}(\langle\langle\Gamma\rangle\rangle\ \psi)$. On the other hand, keeping only pre-filtered moves leads to different remaining strategies for different strategic formulas.

The solution we thus propose for extending the symbolic approach with pre-filtering is, given a structure $S$ and an $ATLK_{irF}$ formula $\phi$, to encode one new derived structure per strategic sub-formula $\phi'$ of $\phi$. The derived structure encodes, in its states, the strategies that remain after applying pre-filtering on the original structure for the formula $\phi'$. This solution needs a bottom-up procedure to evaluate the sub-formulas as we need to know which states of the original structure satisfy a given sub-formula to be able to perform the pre-filtering before encoding the remaining strategies in a new derived structure.

More precisely, let $S = \langle Ag, Q, Q_0, Act, e, \delta, V, \sim, FC\rangle$ be an iCGSf, and $\langle\langle\Gamma\rangle\rangle\ \psi$ an $ATLK_{irF}$ strategic formula with top-level operator $op$ such that all sub-formulas are atomic propositions. Let $filtered \subseteq E_\Gamma$ be a set of $\Gamma$-moves. We can build a derived structure

$$\begin{aligned}&EncStrats^{PF}(S, filtered) = \\ &\langle Ag, Q^{ES}, Q_0^{ES}, Act, e^{ES}, \delta^{ES}, V^{ES}, \sim^{ES}, FC^{ES}\rangle\end{aligned}$$

such that $Q^{ES} = Q \times Split(\Gamma, filtered)$, and all other elements of the structure $EncStrats^{PF}(S, filtered)$ are defined in the same way as for $EncStrats(S)$. In other words, $EncStrats^{PF}(S, filtered)$ is defined as $EncStrats(S)$, except that the strategies that are encoded in the derived states are reduced to the ones defined by $filtered$.

Then, the $eval^{Symbolic,PF}_{ATLK_{irF}}$ algorithm is defined as the $eval^{Symbolic}_{ATLK_{irF}}$ one, but uses $EncStrats^{PF}(S, filtered)$ instead of $EncStrats(S)$. The algorithm is given in Algorithm 6.14. It uses the function $States(Q_1^{ES})$

defined as

$$States(Q_1^{ES}) = \{q \in Q \mid \exists q^{ES} \in Q_1^{ES} \text{ s.t. } state(q^{ES}) = q\},$$

and returning the set of original states stored by states of $Q_1^{ES}$.

Algorithm 6.14 only presents the strategic cases. The model checking for the propositional cases ($true$, $p$, $\neg$ and $\vee$), the temporal ones (**EX**, **EU** and **EW**) and the knowledge ones (**K**, **E**, **D** and **C**) is performed in the standard way, as exposed in Section 2.2.3. The correctness of this algorithm is proved in Section A.7.1 of Appendix A.

Note that, for both symbolic approaches, early termination and caching have no meaning. Indeed, the former is used in the other approaches to stop looking for a strategy when it is not necessary anymore. This idea is useful when strategies are analyzed one by one; on the other hand, the symbolic approaches consider all the strategies at the same time, thus there is no way to stop the process as soon as some winning strategies are found for all states of interest.

Caching has no meaning either. Indeed, in the other approaches, the sub-formulas are evaluated again and again, for each checked strategy. On the other hand, the symbolic approaches do not enumerate the strategies one by one, and thus need to evaluate the sub-formulas only once. Caching is not necessary in this case.

### 6.2.2 Implementation

This section discusses how the symbolic approaches have been implemented with PyNuSMV. PyNuSMV is able to dynamically declare new variables in a model. This mechanism is used to encode, for each agent, for each equivalence class of this agent, and each action, a new variable defining the choice the agent should make in this class.

As the declaration can be performed at any time, this allows the tool to lazily encode the strategies of the agents, depending on the checked strategic formula. This avoids the need to encode the strategies of all the agents of the model—leading to larger models—, and also allows to implement the symbolic approach with pre-filtering, in which the formula is important, too. Pre-filtering forces the tool to declare new variables, for each equivalence class and action, for every strategic sub-formula in which an agent is involved. So, while pre-filtering allows to reduce the number of strategies to encode, it suffers from more BDD variable declarations when dealing with nested strategic sub-formulas involving the same agents.

The declaration of one variable per equivalence class encoding the possible choices for the agent makes the approach with pre-filtering less

**Algorithm 6.14:** $eval^{Symbolic,PF}_{ATLK_{irF}}(S, \phi)$

```
Data: $S$ an iCGSf, $\phi$ an $ATLK_{irF}$ formula.
Result: The states $Q_1^{ES}$, where $States(Q_1^{ES})$ is the set of states
        of $S$ satisfying $\phi$.
case $\phi \in \{\langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi', \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2], \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]\}$
    case $\phi = \langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi_1$
        $Q_1 = eval^{Symbolic,PF}_{ATLK_{irF}}(S, \phi_1)$
        $filtered = filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(States(Q_1), E_\Gamma)$
    case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2]$
        $Q_1 = eval^{Symbolic,PF}_{ATLK_{irF}}(S, \phi_1)$
        $Q_2 = eval^{Symbolic,PF}_{ATLK_{irF}}(S, \phi_2)$
        $filtered = filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(States(Q_1), States(Q_2), E_\Gamma)$
    case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]$
        $Q_1 = eval^{Symbolic,PF}_{ATLK_{irF}}(S, \phi_1)$
        $Q_2 = eval^{Symbolic,PF}_{ATLK_{irF}}(S, \phi_2)$
        $filtered = filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(States(Q_1), States(Q_2), E_\Gamma)$
    if $filtered = \varnothing$ then return $\varnothing$
    $ESS = EncStrats^{PF}(S, filtered)$
    case $\phi = \langle\langle\Gamma\rangle\rangle \boldsymbol{X}\ \phi_1$
        $Q_1^{ES} = States(Q_1) \times Split(filtered)$
        $winning = filter^{ES}_{\langle\langle\Gamma\rangle\rangle\mathbf{X}}(ESS, Q_1^{ES})$
    case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{U}\ \phi_2]$
        $Q_1^{ES} = States(Q_1) \times Split(filtered)$
        $Q_2^{ES} = States(Q_2) \times Split(filtered)$
        $winning = filter^{ES}_{\langle\langle\Gamma\rangle\rangle\mathbf{U}}(ESS, Q_1^{ES}, Q_2^{ES})$
    case $\phi = \langle\langle\Gamma\rangle\rangle[\phi_1\ \boldsymbol{W}\ \phi_2]$
        $Q_1^{ES} = States(Q_1) \times Split(filtered)$
        $Q_2^{ES} = States(Q_2) \times Split(filtered)$
        $winning = filter^{ES}_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(ESS, Q_1^{ES}, Q_2^{ES})$
    return $EqQ(EquivQEqStr(\Gamma, winning))$
case $\phi = ...$ // Cases for the other operators are standard
```

powerful than the partial and early ones. In the latter, the strategies are limited to the moves that really matter; in the former, even if pre-filtering removes an action from a particular state, this action must be encoded in the derived model if there exists another indistinguishable state in which the action has been kept.

PyNuSMV is also able to dynamically declare several transition relations, encoded with BDDs. This mechanism is used to define the different functions used by the two approaches. More precisely, several transition relations are defined per group of agents encountered in strategic formulas:

- one transition relation is defined to implement the $\delta^{ES}$ and $Pre_{str}$ functions. This transition relation is defined as the original transition relation, constrained such that (1) the strategies of the group are kept in the successor states, and (2) the agents of the group follow the encoded strategies, that is, they play the action that the encoded strategies prescribe. The $Pre_{str}$ function is the pre-image of this transition relation.

- one transition relation is defined to implement the $EqQ$ function. This relation forces the state to stay the same—by forcing the original state variables to keep the same value—and lets the other variables encoding the strategies vary.

- one transition relation is defined to implement the $EquivQEqStr$ function. This relation forces the strategies to stay the same, and allows the original state variables to vary, except the ones that are observed by the agents.

Thanks to PyNuSMV, these transition relations can be built lazily, when a new group or strategic formula is encountered. They are defined either for each encountered group in the symbolic approach, or for each strategic sub-formula in the case of the approach with pre-filtering.

# *Chapter 7*

# *Experimental comparison*

This chapter presents the experiments performed with the implementation of the nine approaches presented in thesis. First, the section describes the models and properties used for the tests. Then it analyses and compares the results for each approach. As the implementation is based on a BDD framework, it also compares the different BDD variable reordering techniques offered by PyNuSMV. Finally, it draws some conclusions based on the experiments.

## 7.1 Models and properties

To compare the relative performances of the different approaches, we used three models with strategic formulas. This section describes the three models, how they are encoded as iCGSf, and the formulas we verified.

### 7.1.1 Tian Ji and the king

The first model is based on the ancient Chinese tale of Tian Ji [LQR15]. The model contains two agents, the king and his general Tian Ji, playing a horse racing game. Each agent has $N$ horses $h_1, ..., h_N$ such that if the king plays with horse $h_i$ against Tian Ji with horse $h_j$, the winner is the one with the highest index; if $i = j$, then the winner is chosen non-deterministically. The game is composed of $N$ races and one horse can play only once. The winner of the game is the player with the most won races. The game is replayed infinitely.

The formal iCGSf of the game counts two agents *king* and *Tian Ji*. The states of the model are composed of the remaining horses for Tian Ji, the remaining ones for the king, and the score of both players. In the initial state, all horses are available for both players and both scores are null. There is one action per horse, and each player can play any action corresponding to the remaining horses. Furthermore, each player observes only his remaining horses, not the other's, that is, the equivalence classes for a player are such that two states are indistinguishable if they share the same remaining horses for the player. Finally, fairness constraints are specified such that the king chooses his horses in any order, infinitely many times each. This is achieved by asking the king to choose the order of his horses before starting the game, and specifying one fairness constraint per such an initial order. The number of reachable states evolves exponentially in terms of the number of horses.

The number of strategies of Tian Ji, depending on the number of horses $N$, is given by the equation

$$\prod_{i=1..N} i^{C_N^i},$$

where $C_N^i$ is the number of combinations of $i$ elements among $N$. Indeed, in each state where Tian Ji has $i$ remaining horses, he can choose one of them. Furthermore, there are $C_N^i$ different sets of $i$ remaining horses that Tian Ji can have, giving the equation above. For example, Tian Ji has 24 strategies with 3 horses and 20736 ones with 4 horses. The naive and symbolic approaches have to consider all these strategies, while the others can reduce their number by restricting the search for partial strategies and by pre-filtering out losing moves.

The first formula checked on this model is

$$\phi_1^T = \langle\langle \mathit{Tian\ Ji} \rangle\rangle \mathbf{F}\ \mathit{Tian\ Ji\ wins},$$

where *Tian Ji wins* is true in all states where there are no remaining horses and Tian Ji has a higher score than the king. This formula says that Tian Ji can eventually win a game. It is true in the initial state of the model. Indeed, since the king is fair and will choose any possible order infinitely often, Tian Ji can play the same combination over and over again, and this combination will be winning at some point. In fact, any memoryless uniform strategy for Tian Ji is winning.

Two other formulas have been checked on this model to highlight the differences of behaviors of the tested approaches. The second formula is

$$\phi_2^T = \langle\langle \mathit{Tian\ Ji} \rangle\rangle \mathbf{F}\ \langle\langle \mathit{Tian\ Ji} \rangle\rangle [\neg \mathit{King\ wins}\ \mathbf{U}\ \mathit{Tian\ Ji\ wins}];$$

similarly to *Tian Ji wins*, *King wins* is true in all states where there are no remaining horses and the king has a higher score than Tian Ji. This formula says that Tian Ji has a strategy to reach states in which he can surely win the current game, that is, a state in which he can win before the king wins. The formula is not true in the model because there exists no state in which Tian Ji is sure to win the current game. Indeed, he does not observe the current score, so even in states in which he wins the current game, he does not see it. Furthermore, he does not see the current order of the king's horses, and cannot adapt his choices accordingly.

The last formula checked on the model of Tian Ji is

$$\phi_3^T = \langle\!\langle Tian\ Ji \rangle\!\rangle \mathbf{X}\ Tian\ Ji\ null\ score,$$

where *Tian Ji null score* is true in all states where the score of Tian Ji is 0. This formula is not true in the initial state of the model because Tian Ji has no horse that he is sure will lose the race. Indeed, even the slowest horse $h_1$ could possibly win the race if the king also plays with $h_1$.

### 7.1.2 The three castles

The second model is the one described in [PBJ14]. It is composed of three castles with their corresponding health points ranging from 0 to 3, 0 health points meaning that the castle is defeated. Each castle is defended by a set of workers. At each turn, a worker can attack another castle, defend his own castle or do nothing, but a worker cannot defend her castle twice in a row. The number of damages a castle receives is the number of attackers against this castle minus the number of defenders of this castle, if this number is greater than 0. The health points of the castles are not reset at each turn, thus the game is played in several turns. Finally, the workers only observe whether they can defend their castle or not, and, for each castle, whether it is defeated or not. The model is parametrized with the number of workers of each castle.

The formal model contains one agent per worker. The states are composed of the health points of each castle and whether or not each worker can defend her castle at the next turn. In the initial state, all castles have 3 health points and all workers can defend their castle. The actions of each worker are (1) doing nothing—the only one enabled when her castle is defeated—(2) defending her castle—only possible when she did not defend it in the previous turn—(3) for each other castle, one action to attack it. The equivalence classes are such that for a given worker, she can only observe whether she can defend her castle and, for each castle, whether it is defeated or not. Finally, a unique fairness constraint is defined, containing all states of the model. This ensures that all paths of the model are fair. The initial state is specially marked such that all workers can distinguish it from any other state. This is necessary to be able to specify that some workers have a strategy in the initial state for a given objective. As for Tian Ji's model, the number of reachable states of the model evolves exponentially in terms of the number of workers.

Each worker has 82944 possible strategies. These strategies can be decomposed as her choices in

- the initial state, in which she can choose one action among four: doing nothing, defending her castle or attacking one of the two others. This amounts to 4 possible combinations of choices.
- the non-initial states in which her castle is not defeated and she did not defend it in the previous turn. There are four equivalence classes matching this case, depending on whether the other castles are defeated or not. In these cases, the worker can choose between the four actions. This amounts to $4^4 = 256$ possible combinations of choices.
- the non-initial states in which her castle is not defeated but she defended it in the previous turn. There are four equivalence classes matching this case. The worker can choose between three actions as she cannot defend her castle. This amounts to $3^4 = 81$ possible combinations of choices.
- the non-initial states in which her castle is defeated. There are 8 equivalence classes matching this case, depending on whether the other castles are defeated and whether she defended her castle just before or not. The worker can choose only one action, giving $1^8 = 1$ possible choice.

All these choices can be combined in any way since they are exclusive in the equivalence classes they consider, giving $4 * 256 * 81 * 1 = 82944$ possible uniform strategies for one worker.

The depth of the model—that is, the number of steps needed to reach all the reachable states from the initial one—does not change with the number of workers since it depends only on the health points of the castles. An exception is when there are few workers, that is, with one worker in each castle. In this case, the depth is a bit higher because there are too few workers to ensure to quickly reach a final state. The partial, backward and early approaches really depend on this depth since it dictates how far the partial strategies are.

We are interested in two formulas. The first one is

$$\phi_1^C = \langle\!\langle Castle_1, Castle_2 \rangle\!\rangle \mathbf{F}\ Castle_3\ defeated, \tag{7.1}$$

where $Castle_i$ groups the workers of the $i$th castle and $Castle_3$ *defeated* is true in all states in which the third castle has 0 health points. This formula is true in all tested models, but is not true in general. Indeed, if the third castle has enough workers, they are able to defend the castle and prevent the other workers to damage it. More precisely, if the third castle has more workers than the addition of the two others, the formula is false, even if the workers have perfect information. The tested models always have enough workers in the first two castles to make the formula satisfied.

The second formula is

$$\phi_2^C = \langle\!\langle Worker_1, Worker_2 \rangle\!\rangle \mathbf{F}\ all\ defeated, \tag{7.2}$$

where $Worker_1$ (resp. $Worker_2$) is a worker of the first castle (resp. second castle), and *all defeated* is true in the states where all castles have 0 health points. This formula is false in all tested models because, even if they can defeat the third castle, the workers have not enough information to ensure that the other two castles will be defeated at the same time.

### 7.1.3 The prisoners and the light bulb

The third model is based on the problem of the 100 prisoners and the light bulb [vDK15]:

> "A group of 100 prisoners, all together in the prison dining area, are told that they will be all put in isolation cells and then will be interrogated one by one in a room containing a light with an on/off switch. The prisoners may communicate with one another by toggling the light switch (and that is the only way in which they can communicate). The light is initially switched off. There is no fixed order of interrogation, or interval between interrogations, and the same prisoner may be interrogated again at any stage. When interrogated, a prisoner can either do nothing or toggle the light switch, or announce that all the prisoners have been interrogated. If that announcement is true, the prisoners will (all) be set free, but if it is false, they will be executed. While still in the dining room, and before the prisoners go to their isolation cells (forever), can the prisoners agree on a protocol that will set them free?"

One strategy to guarantee their freedom is to designate a *counter* among the prisoners. This counting prisoner starts at 0 and each time he enters the room and the light bulb is switched on, he switches it off and increments his counter. Every time another prisoner enters the room, if the light bulb is switched off and he has never switched it on, he switches it on before leaving the room. When the prisoner with counter enters the room and his counter is at 99, he is sure that all prisoners have entered the room at least once and he can safely announce that all prisoners have visited the room. This strategy is winning if the warden fairly chooses the prisoners each day because the counter will enter infinitely often, thus will be able to switch the light off as many times as he wants, and the other prisoners will also enter the room infinitely often and be able to switch the light on once.

The formal model encodes this problem of the prisoners and the light bulb, designates a special prisoner that can keep track of a counter and gives the ability to the other prisoners to remember whether they already switched the light on or not. The idea behind the model is to verify that the prisoners effectively have a strategy to be released with these limited capabilities.

More precisely, the formal model is composed of the warden and $N$ prisoners; one of them is the counting one. The warden keeps track of which prisoners have been interrogated at least once—to be able to release or execute them when the counting prisoner makes an announce—and chooses the next prisoner to interrogate. Furthermore, he keeps track of whether the prisoners should be released (if the counting prisoner correctly announces that all prisoners have been interrogated) or executed (if the counting prisoner makes an incorrect announce). The counting prisoner keeps track of his counter and each prisoner keeps track of whether he has already switched the light bulb or not. The states of the model are thus composed of the state of the light bulb (on or off), who has already been interrogated, should the prisoners be released or not, and executed or not, the counter of the counting prisoner, whether each prisoner has already switched the light bulb or not, and finally the prisoner that is being currently interrogated. In the initial state, the light bulb is off, nobody has been interrogated, the prisoners should not be released or executed, the counter is at 0, no prisoner has already switched the light bulb, and nobody is currently interrogated. When nobody is interrogated, the warden can choose any prisoner, and when a prisoner is interrogated, he can choose to switch the light bulb or do nothing, and the counting prisoner can additionally choose to make the announcement, as well as to increment his counter. Each prisoner sees the light bulb when he is currently interrogated, knows whether he already switched the light or not, and knows whether he is currently interrogated. In addition, the counting prisoner knows the value of his counter. All prisoners (including the counting one) can distinguish the initial state from the others. Finally, fairness constraints are specified such that each prisoner is interrogated infinitely often. Again, the number of reachable states of the model evolves exponentially in terms of the number of prisoners.

In this model, the counting prisoner has $6^{2N}$ uniform strategies, where $N$ is the number of prisoners (including himself). Indeed, he can do nothing when he is not interrogated, so only his choices when he is interrogated can lead to different strategies. The model contains $2N$ equivalence classes for the counting prisoner when he is currently interrogated. Indeed, he observes the value of his counter (from 0 to $N - 1$) and the state of the light bulb. In each class, he can perform $3 * 2 = 6$ different actions: doing nothing, switching the light bulb, or making an announcement, and furthermore incrementing his counter or not. This gives us $6^{2N}$ possible strategies for the counting prisoner: 1296 strategies with two prisoners, 46656 ones with three prisoners. The other prisoners have 16 different strategies. A prisoner can only do something when he is interrogated. In this case, the model contains 4 equivalence classes as he observes the state of the light bulb and whether or not he already switched it on. In each of these classes, the prisoner can switch the bulb or not, giving us $2^4$ possible strategies. So there are at most $16^{N-1} * 6^{2N}$ strategies to consider when checking whether the coalition of the $N$ prisoners (including the counting one) can enforce a given objective.

We are interested in the following formula

$$\phi^P = \langle\!\langle \mathit{prisoners} \rangle\!\rangle [\neg \mathit{executed}\ \mathbf{U}\ \mathit{released}], \tag{7.3}$$

saying that the prisoners have a collective strategy to be released before being executed. This formula is true in the model, showing that the prisoners effectively have a counting strategy to be free.

## 7.2 Measures and comparisons

The formulas of the previous section have been checked using the approaches on models of increasing size. This section presents and compares the results. All the experiments have been performed on a MacBook Pro with a 2.6GHz processor and 16GB RAM, and under a time limit of 1800 seconds. This time limit is indicated by a horizontal line in the graphs, and data points reaching this time limit are depicted above the line. Each data point is the average of 20 runs; the observed variability was very low for all measurements. Furthermore, these experiments usually consumed less than 1GB of memory, but some consumed up to several GBs. They nevertheless never consumed all the available memory.

In all the figures, short names are used to refer to the tested approaches:

- *Naive* refers to the naive approach without pre-filtering;
- *Naive/filt* refers to the naive approach with pre-filtering;
- *Partial* refers to the partial approach with caching and early termination, but without pre-filtering;
- *Partial/filt* refers to the partial approach with caching, early termination, and pre-filtering;
- *Backward* refers to the backward approach;
- *Early* refers to the early approach with caching and early termination, but without pre-filtering;
- *Early/filt* refers to the early approach with caching, early termination, and pre-filtering;
- *Symbolic* refers to the symbolic approach without pre-filtering;
- *Symbolic/filt* refers to the symbolic approach with pre-filtering.

The partial and early approaches use caching and early termination. For the former, [BPQR14] showed that using both always increases performances, and early experiments not provided in this thesis showed same results for the latter.

For each approach, observations are given, then the differences of performances are explained based on these observations. As the *backward* approach does not handle fairness constraints and $\langle\langle\Gamma\rangle\rangle\mathbf{W}$ formulas, it is only tested on the model of the castles.

The Python implementation used for the following experiments is a prototype showing the applicability of the approaches. It would not compete with dedicated tools performing the same kind of tasks. These experiments are not meant to show the absolute performances of the implementation but the relative gain of the different approaches.

### 7.2.1 Tian Ji and the king

This section presents the results for the properties of Tian Ji's model.

$\phi_1^T = \langle\langle Tian\ Ji \rangle\rangle \mathbf{F}\ Tian\ Ji\ wins$

Figure 7.1 shows the evolution of the verification time in terms of the number of horses, for the eight approaches checking the formula $\phi_1^T$ on the model of Tian Ji. As explained before, any strategy of Tian Ji is winning for this objective because the fair king will ensure that all configurations will eventually happen.

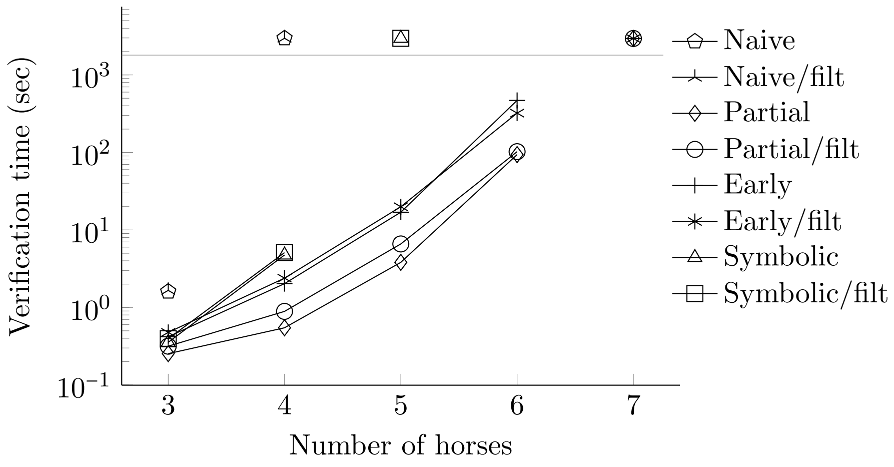


Figure 7.1: Evolution of the verification time of the approaches for the formula $\phi_1^T = \langle\langle Tian\ Ji \rangle\rangle \mathbf{F}\ Tian\ Ji\ wins$.

**Pre-filtering** Pre-filtering removes no move because any general strategy is winning.

**Naive approaches** The Naive approach checks all possible strategies. Furthermore, as pre-filtering removes no move, the Naive/filt approach has to check the same strategies as the Naive one. The time needed for pre-filtering is negligible compared to the time needed for checking all strategies.

**Partial** The Partial approach checks only one strategy. Since all of them are winning, early termination allows the approach to stop its processing after one strategy.

**Partial/filt** The Partial/filt approach only checks one strategy. Pre-filtering removes no move, so the extra effort is useless. Nevertheless, the time needed for pre-filtering is not negligible anymore. But dynamic reordering of BDD variables accelerates the process of checking the single strategy, thanks to the better variable order computed during pre-filtering.

**Early** The Early approach extends the first strategy to $\lfloor \frac{N}{2} \rfloor + 1$ steps before concluding it is a winning strategy. At smaller steps, the approach cannot decide whether any extension is winning because Tian Ji did not win enough races to be sure to win the game.

**Early/filt** The Early/filt approach also extends the first strategy to $\lfloor \frac{N}{2} \rfloor + 1$ steps before concluding it is a winning strategy, as pre-filtering is useless. Pre-filtering is not negligible, but the dynamic reordering of variables accelerates the rest of the process.

**Symbolic approaches** The Symbolic approach encodes and tests all strategies at once. The Symbolic/filt approach behaves exactly like the Symbolic one since pre-filtering removes no move. In this case, the time needed for pre-filtering is negligible.

**Comparison** First, pre-filtering removes no moves, thus is useless in reducing the number of strategies to consider. The time needed to perform pre-filtering in the Naive and Symbolic approaches is negligible compared to the time needed for checking the strategies. This explains why the version with pre-filtering is the same as the other one for these approaches.

On the other hand, pre-filtering takes a significant amount of time in the partial and early cases, as the number of checked strategies is very small. Nevertheless, the time needed by both versions (with and without pre-filtering) are very similar. This is explained, for both approaches, by the fact that pre-filtering, while removing no moves, triggers the dynamic reordering of the BDD variables; the new order computed after pre-filtering is way better than the initial one and allows the verification of strategies to be performed faster. This can be shown by disabling dynamic reordering. In this case, the approaches with pre-filtering take more time to check the formula, and the difference is the time needed to perform the useless pre-filtering. The fact that the approaches with and without pre-filtering perform similarly is thus purely coincidental.

Second, the Partial approach is better than the Early approach because the Partial one checks only one adequate partial strategy while the Early one needs to check $\lfloor \frac{N}{2} \rfloor + 1$ sub-models before concluding that there is a winning strategy.

Finally, the symbolic approaches have worse performances than the others (except the naive ones). This is explained by the fact that, as all strategies are winning and the other approaches stop as soon as a winning strategy is found, symbolic approaches have to perform more work to check all strategies at the same time. The symbolic approaches are, however, better than the naive ones because the former check all strategies symbolically, while the latter have to check them individually.

$\phi_2^T = \langle\langle Tian\ Ji \rangle\rangle \mathbf{F}\ \langle\langle Tian\ Ji \rangle\rangle [\neg King\ wins\ \mathbf{U}\ Tian\ Ji\ wins]$

Figure 7.2 shows the evolution of verification time for the eight approaches on the formula $\phi_2^T$ checked on the model of Tian Ji. As said before, this formula is not satisfied by the initial state of the model because the inner strategic formula $\langle\langle Tian\ Ji \rangle\rangle [\neg King\ wins\ \mathbf{U}\ Tian\ Ji\ wins]$ is false in all states. On the other hand, Tian Ji has a winning strategy if he can observe the whole system because he knows which order the king will play for the current game.

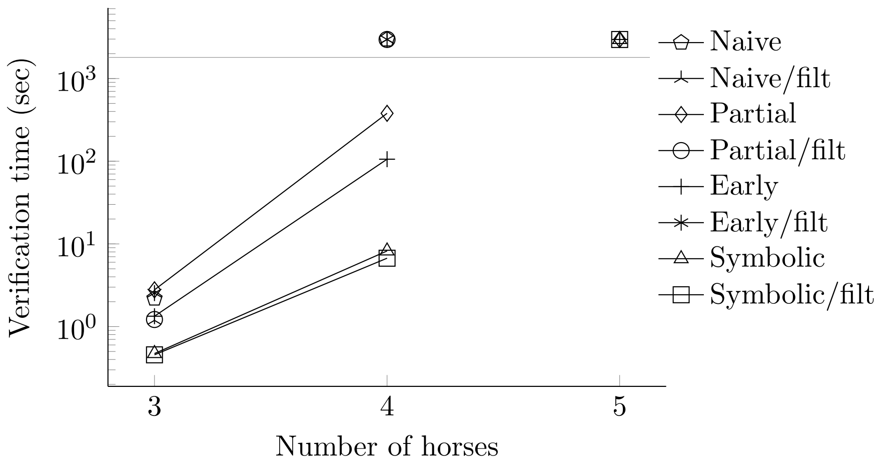


Figure 7.2: Evolution of the verification time of the approaches for the formula $\phi_2^T = \langle\langle Tian\ Ji \rangle\rangle \mathbf{F}\ \langle\langle Tian\ Ji \rangle\rangle [\neg King\ wins\ \mathbf{U}\ Tian\ Ji\ wins]$.

**Pre-filtering** Empirically, we can observe that pre-filtering removes about 50% of the moves for the strategic sub-formula. Tian Ji has a winning general strategy in many states, but not all actions lead to winning the game. Furthermore, pre-filtering really helps for the top-level strategic formula because no state satisfies the sub-formula, making the top-level one trivially unsatisfiable, even with perfect information.

**Naive** The Naive approach checks all strategies, for both strategic operators. The number of strategies quickly becomes too large to handle in 30 minutes: there are more than 20000 strategies to check for 4 horses.

**Naive/filt** The Naive/filt approach checks all strategies for the inner strategic operator, but can directly conclude that the top formula is not satisfied when the pre-filtering is done.

This approach still has to check the same number of strategies for the sub-formula as the Naive one: after pre-filtering, all equivalence classes are still present, and all actions are still possible in some states in each of them. Indeed, for any set of remaining horses for Tian Ji, he could be currently winning the game, and any choice for the next horse could still win the next race, as the king could run with any horse (since Tian Ji does not know the king's remaining horses). This leads to the same number of strategies as for the Naive approach, but the strategies contain fewer moves.

**Partial** The Partial approach checks all adequate partial strategies for the sub-formula

$$\langle\!\langle \mathit{Tian\ Ji} \rangle\!\rangle[\neg \mathit{King\ wins}\ \mathbf{U}\ \mathit{Tian\ Ji\ wins}]$$

in all reachable states as the adequate partial strategies for the initial states lead to all reachable states. Furthermore, since the top formula is false in the initial state, the approach checks all adequate partial strategies for the initial state for the top formula. Thus, it has to check all adequate strategies, for both strategic sub-formulas, to conclude that the formula is false.

Nevertheless, this approach has to check the sub-formula for relatively small subsets of the reachable states each time, since partial strategies reach relatively small subsets of the reachable states. Indeed, as the top-level strategies do not cover the complete set of reachable states—for instance, if Tian Ji chooses his best horse, he will not consider the states in which he still can play this horse—, the number of strategies to consider for the sub-formula (for this particular top-level strategy) is not large.

**Partial/filt** Pre-filtering in the Partial/filt approach triggers the evaluation of the sub-formula in all the reachable states at once. This represents a large number of adequate partial strategies, compared to computing strategies for different separate subsets of states as for the Partial approach. More precisely, as all reachable states are considered at the same time, the number of strategies to consider is the same as for the Naive approach: in any equivalence class, all choices remain, thus the number of strategies is not reduced.

Nevertheless, when the evaluation of sub-formula is done, the approach detects that there are no general winning strategies in the initial states, thus it can directly conclude that the formula is violated.

**Early** The Early approach evaluates $N$ strategies for the top formula because there are $N$ possible initial moves for Tian Ji. It needs to check this small amount of strategies (for the top formula) because, for each of them, it evaluates the sub-formula in the reached states and immediately determines that there is no extending winning strategy. On the other hand, it checks the sub-formula on a large subset of the reachable states because the top formula-related strategies reach a lot of states.

The sub-formula is evaluated on many states at once. This means that a large partial strategy is already reached when splitting the moves in these states. This large partial strategy is sufficient to determine that the formula is false in these states. Thus, the Early approach has to check several strategies, but these strategies are determined as losing without extending them.

**Early/filt** The Early/filt approach triggers pre-filtering for the top formula, that evaluates the sub-formula on all the reachable states. This means that the approach has to check the sub-formula for all these states at once. Evaluating the sub-formula on all the reachable states also triggers pre-filtering for the sub-formula removing about half the moves.

Furthermore, the number of strategies (splitting the other half of the moves) in the whole set of reachable states is large. Nevertheless, the Early/filt approach can directly determine that each strategy is losing, avoiding to extend them to adequate ones.

The number of strategies to check in the whole set of reachable states is way larger than the strategies checked by the Early approach. Nevertheless, when the sub-formula has been evaluated on all the reachable states, pre-filtering for the top formula determines that no state is winning and no strategy must be checked for the top formula.

**Symbolic** The Symbolic approach checks all the strategies at once. First, it computes the set of states satisfying the sub-formula, then the set of states satisfying the top formula. It encodes strategies for Tian Ji once, even if there are two strategic formulas, because it can reuse the encoded strategies for both formulas.

**Symbolic/filt** The Symbolic/filt approach first triggers pre-filtering. This leads to checking the sub-formula. This sub-formula is first pre-filtered, but nothing is gained because any action that is ruled out in a state can be winning in another indistinguishable state. Nevertheless, the approach does not have to encode the strategies for the top formula because pre-filtering evaluates that there are no possibly winning moves, directly determining that the top formula cannot be true.

**Comparison** The Naive approach takes more time that the Naive/filt one for 3 horses because the former has to check the 24 strategies for both strategic sub-formulas, while the latter checks these 24 strategies for the inner strategic operator only. Furthermore, they both cannot check the formula for 4 horses because the number of strategies to check is too large: there are more than 20000 strategies to consider.

Second, pre-filtering in the Partial/filt approach evaluates the sub-formula for all reachable states at once, representing a large number of strategies. On the other hand, the Partial approach evaluates the sub-formula on smaller subsets of states. These states reach only a subset of the states of the model, reducing the number of strategies to consider. Overall, this lazy evaluation allows the Partial approach to compute less strategies than the Partial/filt one to evaluate the sub-formula. Nevertheless, after evaluating the sub-formula, the Partial/filt approach detects that there are no general winning strategies in the initial states and can directly conclude. All in all, the Partial/filt approach performs worse than the Partial one because it considers all reachable states at once, instead of different subsets of the reachable states separately, leading to way more strategies to check. In the case of 4 horses, the Partial/filt approach does not succeed in checking all the strategies for the sub-formula within 30 minutes, while the Partial one does.

Regarding the early approaches, the number of strategies to check in the whole set of reachable states (as done by the Early/filt approach) is larger than the strategies checked by the Early approach, since the Early strategies already made a choice in the initial states. This allows the Early approach to conclude for 4 horses within 30 minutes while the Early/filt one does not.

Regarding the Symbolic approaches, it is a coincidence that both approaches take a similar amount of time. The Symbolic approach has to evaluate the two strategic sub-formulas. On the other hand, the Symbolic/filt one has to perform pre-filtering on the sub-formula and then evaluate it, but does not have to evaluate the top formula.

Regarding all the approaches, the Partial approach takes much time because it has to check a lot of strategies to determine that none of them are winning. The Early approach has less work to do as it does not extend the strategy. Finally, the symbolic ones are better because they can evaluate all the strategies at once and determine that they are not winning.

Pre-filtering does not work for the Partial/filt and Early/filt approaches because, in both cases, it triggers the evaluation of the sub-formula on all reachable states, leading to way more strategies to consider, as these strategies have to take all states into account at the same time.

$\phi_3^T = \langle\langle Tian\ Ji \rangle\rangle \mathbf{X}\ Tian\ Ji\ null\ score$

Figure 7.3 shows the evolution of verification time of the eight approaches for the formula $\phi_3^T$ on the model of Tian Ji. This formula says that Tian Ji can enforce to lose the first race, and is false because even if he chooses his slowest horse, the king could use his slowest one, too.

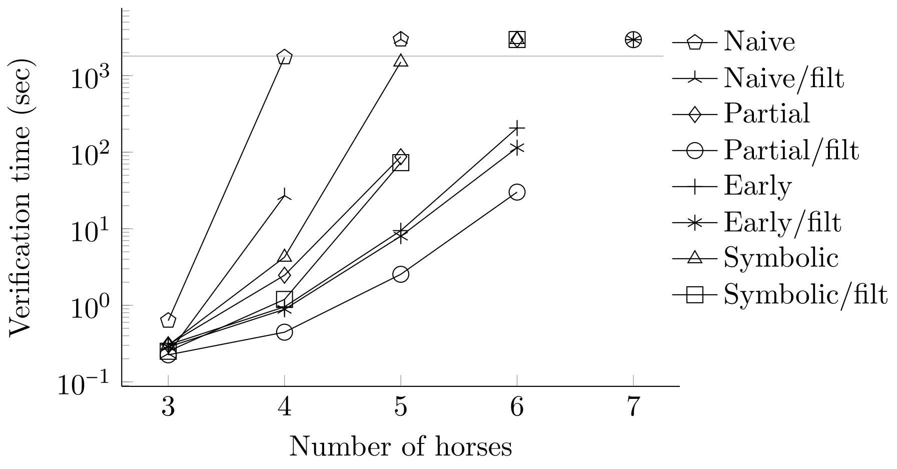


Figure 7.3: Evolution of the verification time of the approaches for the formula $\phi_3^T = \langle\langle Tian\ Ji \rangle\rangle \mathbf{X}\ Tian\ Ji\ null\ score$.

**Pre-filtering** Pre-filtering removes a lot of moves. Indeed, there are only few states in which Tian Ji can keep his score at 0. In these states, he has to have a score of 0, and he has to still have a horse that surely loses the next race. Furthermore, pre-filtering for the Partial/filt and Early/filt approach is even more efficient as they restrict the sub-formula *Tian Ji null score* to the successors of the initial states.

**Naive** The Naive approach has to check all strategies to conclude that the formula is false. It is more efficient in this case, compared to the previous formula, because the evaluation of a single strategy is faster when dealing with $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}$ operators than with $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}$ ones.

**Naive/filt** This approach has fewer strategies to check than the Naive one, thanks to pre-filtering. But, as some reachable states have a 0 score for Tian Ji and a losing horse, the number of strategies to check is still large.

**Partial** Thanks to the restriction to adequate partial strategies, the Partial approach has fewer strategies to check before concluding that the formula is false.

**Partial/filt** Thanks to the very efficient pre-filtering, the Partial/filt approach has very few strategies to check: there remains only $N-1$ strategies, as only the initial moves of the game are kept (except the one playing the best horse, as it cannot be used to win the first race). In this case, most of the time (50 – 70%) is spent to perform pre-filtering.

**Early** The Early approach has to check the $N$ initial strategies to conclude that the formula is false. Indeed, it is not necessary to extend them as the approach can directly conclude that there is no adequate winning strategy. Nevertheless, it has to complete each strategy with the compatible reachable moves to check that they are not winning, leading to extra work compared to the Partial/filt approach.

**Early/filt** The Early/filt approach computes the same pre-filtering as the Partial/filt approach. It thus still has to check $N-1$ strategies before concluding that the formula is false. As for the Early approach, it has to complete each strategy with the compatible reachable moves to check that they are not winning.

**Symbolic approaches** The Symbolic approach encodes all strategies for Tian Ji, and checks them all at the same time, while the Symbolic/filt one benefits from pre-filtering and encodes fewer strategies.

**Comparison** The approaches with pre-filtering perform better than their counterpart without pre-filtering. This is due to the fact that the number of remaining strategies is way smaller than the initial number of strategies.

Furthermore, the Early/filt approach performs worse than the Partial/filt one because it has more work to do: for each initial move, it has to complete the strategy with compatible reachable moves before concluding that the strategy cannot win. On the other hand, the Partial/filt approach completes the strategy with pre-filtered moves only, and there are no such moves as pre-filtering is limited to the initial states.

The Partial strategy performs worse than the Early one because it has to check adequate partial strategies, while the Early approach only has to check the $N$ initial moves (and complete them before checking them).

Regarding the naive and symbolic approaches, the former perform worse than the latter because the former check all strategies (modulo pre-filtering) one by one, while the latter can check them all at the same time.

### 7.2.2 The three castles

This section presents the results for the properties of the three castles. It includes the results for the backward approach as it is applicable in this case.

$\phi_1^C = \langle\!\langle Castle_1, Castle_2 \rangle\!\rangle \mathbf{F}\ Castle_3\ defeated$

Figure 7.4 shows the evolution of verification time of the nine approaches for checking the formula $\phi_1^C$ on the model of the castles. The size of the model (Number of workers) is given as a triplet 1 2 3, meaning that the first castle is defended by one worker, the second one by two and the third one by three workers. The tests were performed on instances in which there are at least as many workers in the first two castles as in the third castle. Cases in which there are more workers in the third castle than in the other two have not been considered because, in this case, the formula is false even with perfect information.

**Pre-filtering** Depending on the size of the model, pre-filtering removes from 18% (1 1 1 case) to 77% (1 1 2 case) of the moves. For the other sizes, the gain of pre-filtering is between these two bounds.

The huge gain of 77% in the 1 1 2 is explained by the fact that the power of the workers of the first two castles is comparable to the power of the workers of the third castle. This means that the first two castles workers do not have many winning moves, even if they know whether their opponents defended their castle before and the actual health points of the castles. On the other hand, in the 1 1 1 case, the first two castles have more power compared to the third one and can more easily win if they have perfect information. The other cases are between these two

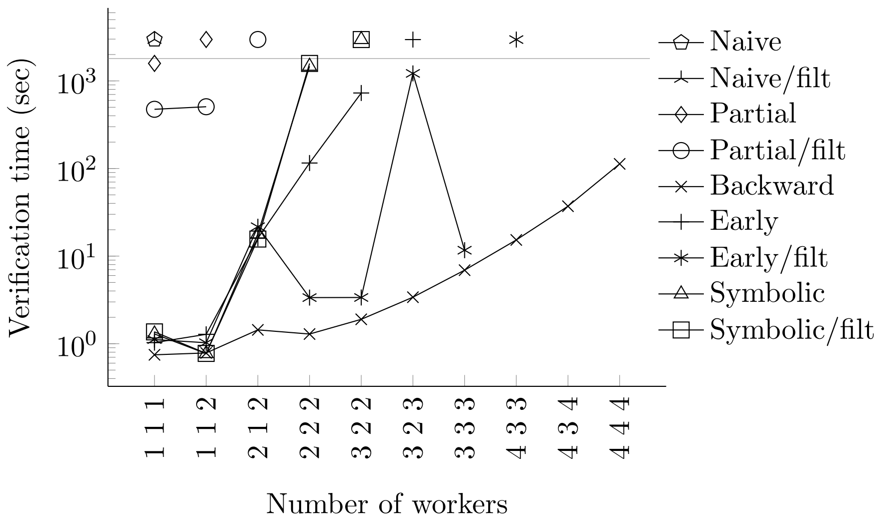


Figure 7.4: Evolution of the verification time of the approaches for the formula $\phi_1^C = \langle\!\langle Castle_1, Castle_2 \rangle\!\rangle \mathbf{F}\ Castle_3\ defeated$.

extremes. In all cases, the time needed to perform pre-filtering is always negligible compared to the search for a winning strategy.

**Naive approaches** The number of strategies to check ($6.9 \times 10^9$ for the 1 1 1 case) is way too large for the Naive approach to find a winning one within 30 minutes. Furthermore, the gain from pre-filtering for the 1 1 1 case is low, so the Naive/filt cannot find a winning strategy within 30 minutes, either.

**Partial approaches** The Partial approach succeeds in finding a winning strategy within 30 minutes for the 1 1 1 case. Nevertheless, for the 1 1 2 case, it cannot find a winning one. In this case, there is the same number of strategies, but it is more costly to check each strategy as the model is bigger. On the other hand, the Partial/filt approach benefits from pre-filtering and finds a winning strategy more quickly than the Partial one. Nevertheless, it fails at finding a winning one in the 2 1 2 case.

**Backward** The Backward approach needs to reach at most half the depth of the model to determine whether a strategy is losing or not in the initial state. This is especially true in the smaller models in which the workers of the first two castles have power comparable to those of the third castle. For larger models, the first two castles workers have more power than those of the third castle, and the approach needs only one or two steps, and no backtracking, to find a winning strategy. The increase of time is simply due to the fact that the model becomes larger and larger, and evaluating a single strategy—with the $filter$ algorithms—takes more and more time.

**Early** In the 1 1 1 case, the Early approach needs to reach up to half the depth of the model to determine the strategies to be losing. This allows the approach to find a winning strategy easily.

In the 1 1 2 case, it needs to split only a few steps to determine that the strategies are losing. This is because the group of agents of the first two castles have relatively less power than in the previous case.

In the larger cases, the Early approach needs to reach about half way, again. The number of strategies increases with the number of workers to consider, as well as the time needed to check larger models.

**Early/filt** In the 1 1 1 and 1 1 2 cases, the Early/filt approach as fewer strategies to check than the Early approach because it benefits from pre-filtered moves.

In the 2 1 2 case, the approach finds a winning strategy within the same time as the Early approach.

In the 2 2 2, 3 2 2 and 3 3 3 cases, it very quickly finds a winning strategy (after resp. 18, 37 and 52 strategies). It really benefits from pre-filtering and finds a good strategy after a few steps. It still has to reach about half way to find this winning strategy. In the 3 2 3 case, it needs to consider many more strategies before finding a good one.

The Early/filt approach really benefits from pre-filtering and decreases the number of strategies to check, compared to the Early approach, but still struggles to find winning strategies for some cases such as the 3 2 3 and 4 3 3 cases.

**Symbolic approaches** The Symbolic approach has to encode and check all strategies at the same time. As the number of workers increases, there are more and more strategies for the group. Furthermore, the Symbolic/filt approach cannot benefit from pre-filtering because all equivalence classes are still present and all actions are still possible in each of them. Thus, it performs exactly as the Symbolic one, as the time to perform pre-filtering is negligible.

**Comparison** The naive approaches are not efficient as they have to check all strategies before concluding. As the number of strategies is already huge for the first case, they cannot check them all within 30 minutes.

The partial approaches succeed in decreasing the number of strategies to consider, and thanks to early termination, can stop as soon as a winning strategy is found. Nevertheless, the number of adequate partial strategies is still large, and the approaches quickly fail to find a winning one.

The symbolic approaches are better. Nevertheless, pre-filtering does not benefit to the Symbolic/filt approach, thus both approaches do the same work.

The early approaches are better in the present scenario because they can quickly determine that a partial strategy and all its extensions cannot be winning. In particular, the Early/filt approach really benefits from pre-filtering and drastically reduces the number of strategies it checks for the larger models. The Early/filt approach shows some irregularities in performances because it sometimes makes the right choices of actions, and sometimes not.

The backward approach is the best in this scenario because it concentrates on the strategies that can effectively reach the target states. It does not need to backtrack a lot before finding a winning strategy in the initial state.

$\phi_2^C = \langle\!\langle Worker_1, Worker_2 \rangle\!\rangle \mathbf{F}\ all\ \ defeated$

Figure 7.5 shows the evolution of verification time of the nine approaches for checking the formula $\phi_2^C$ on the model of the castles. This formula is false for all checked sizes.

**Pre-filtering** A major difference between the 1 1 1 case and the others is that, in the former case, the two workers have a strategy to achieve their goal when they have perfect information, while it is not the case for the greater sizes. Thus, pre-filtering, in the cases of larger models, allows the Naive/filt, Partial/filt and Early/filt approaches to directly determine that the formula is false, without checking any strategy.

**Naive approaches** The Naive approach cannot check the huge amount of strategies to conclude that the formula is false within the 30 minutes limit, even for the smallest model. On the other hand, the Naive/filt one benefits from pre-filtering. For the first two models, pre-filtering drastically reduces the number of strategies, and for the three last ones, it leaves only one strategy to check.

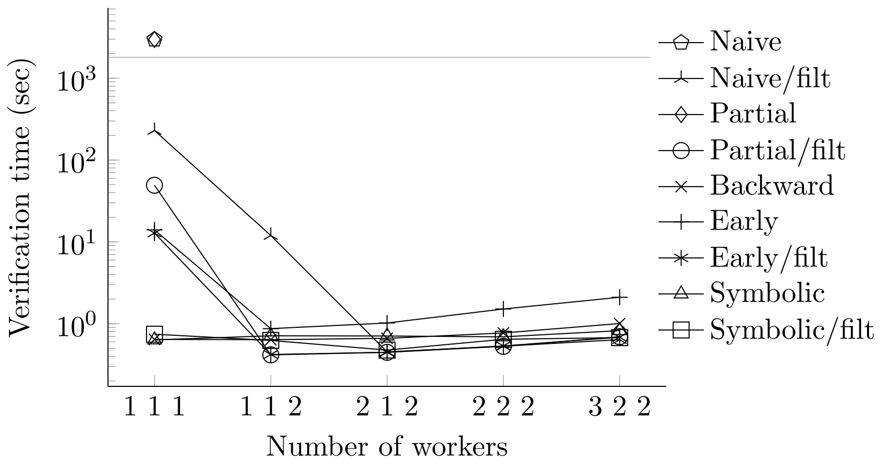


Figure 7.5: Evolution of the verification time of the approaches for the formula $\phi_2^C = \langle\langle Worker_1, Worker_2 \rangle\rangle \mathbf{F}\ \mathit{all\ defeated}$.

For the 1 1 2 case, pre-filtering directly concludes that the initial state is not winning, but some other states of the model are still winning. This is why the Naive/filt approach still has some work to do in the 1 1 2, but not after, while the Partial/filt and Early/filt approaches can directly conclude, even in the 1 1 2 case.

**Partial** The Partial approach reaches the timeout even for the smallest model size. Given the number of possible strategies ($6.9 \times 10^9$) and the fact that the approach must check them all to determine that the formula is false, this result is not surprising.

**Partial/filt** On the 1 1 1 case, pre-filtering drastically reduces the number of moves to consider, and thus the number of strategies the Partial/filt approach needs to check before stating that the formula is false. The approach shows that the remaining strategies are losing within the time limit, while the Partial fails to do so. For the other cases, pre-filtering performs all the work.

**Backward** On the 1 1 1 case, the Backward approach only needs to reach about half way—that is, to fix actions in states up to half of the depth of the model from the target states—to determine that there is no winning extension of the strategy that reaches the *all defeated* states from the initial state.

For the other cases, it directly determines that there is no extension of the strategy that is winning in the initial state, thanks to its evaluation of the losing states. It does not need to extend the first considered strategy.

**Early** For the 1 1 1 case, the Early approach also needs to reach about half way—from the initial state—to determine strategies to be losing, as for the previous formula. This allows the approach to check all strategies more easily.

For the other cases, the approach only needs to check the 16 initial actions of the two workers to conclude that there can be no winning strategy. Indeed there are no winning strategy in these cases, even with perfect information, and the approach can determine it directly.

The increasing of time for the Early approach only comes from the fact that the model is bigger and bigger, making the verification of these 16 strategies longer and longer.

**Early/filt** For the 1 1 1 case, the Early/filt approach does not gain from pre-filtering. In fact, the moves that are filtered out are never reached by the Early/filt approach because it can determine that the strategies are losing before reaching them. Thus, it behaves like the Early one on this case. For the other cases, the Early/filt approach does not check any strategy since pre-filtering directly determines that there can be no winning strategy.

**Symbolic approaches** The Symbolic approach behaves in the same way for all model sizes. The only differences come from building a model of increasing size. The actual fixpoint computation to determine the winning strategies is exactly the same in all cases. On the other hand, the Symbolic/filt approach gains from pre-filtering. It drastically reduces the number of strategies to encode for the first two cases. For the three last ones, there remains only one strategy to encode and check.

**Comparison** The Naive and Partial approaches do not handle the smallest model because they have to check the huge number of strategies to determine that there are no winning ones. On the other hand, the Naive/filt, Partial/filt and Early/filt approaches only need pre-filtering to conclude. The Early approach can also quickly determine that the formula is false because it just needs to check all possible actions in the initial state.

The Backward approach is also really quick because there is only one possible strategy for the two workers in states satisfying *all defeated*—doing nothing—,and there is no general strategy reaching these states from the initial one. The approach can thus directly conclude that the formula is false. Finally, the symbolic approaches also perform well because the BDDs they compute remain very small.

In conclusion, all approaches are comparable for the case 2 1 2 and after because it is easy to show that the formula is false, except for the Naive and Partial approaches that must check all possible strategies to reach this conclusion.

### 7.2.3 The prisoners and the light bulb

This section presents the results for the properties of the prisoners model.

$\phi^P = \langle\!\langle prisoners \rangle\!\rangle[\neg executed\ \mathbf{U}\ released]$

Figure 7.6 shows the evolution of verification time of the eight approaches for checking the formula $\phi^P$ on the model of the prisoners. The formula is true in all tested models, showing that the prisoners effectively have a strategy to release all the prisoners without being executed. The number of partial strategies adequate for the initial state grows exponentially in terms of the number of prisoners. Furthermore, the number of strategies is already huge for the smallest model, compared to the model of Tian Ji: the 2 prisoners have ≈ 20000 strategies while Tian Ji has only 24 strategies with 3 horses.

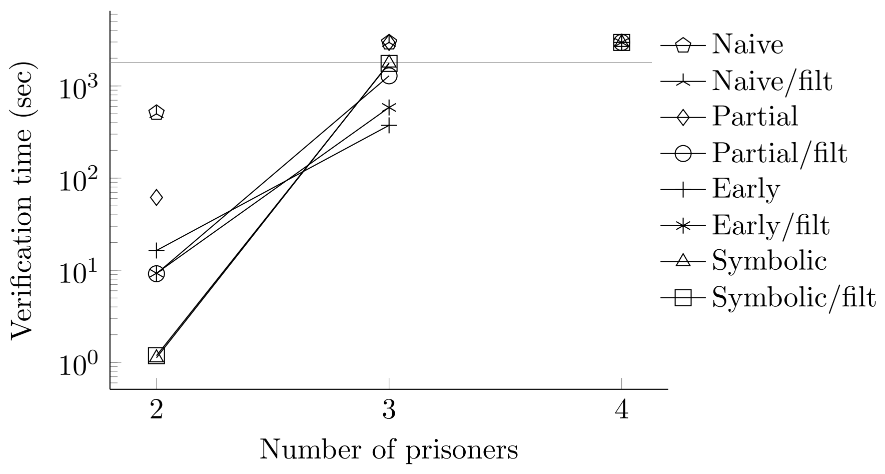


Figure 7.6: Evolution of the verification time of the approaches for the formula $\phi^P = \langle\!\langle prisoners \rangle\!\rangle[\neg executed\ \mathbf{U}\ released]$.

**Pre-filtering** Pre-filtering does not remove a lot of moves. Indeed, when the prisoners have perfect information, the counting one knows who has already been interrogated and can make a correct announcement as soon as possible. The only moves that are removed are the ones that lead to an incorrect announcement and the execution of all prisoners.

**Naive approaches** For the 2 prisoners case, the Naive approach finds a winning strategy among the $\approx 20000$ ones within 30 minutes, but fails to find one for 3 prisoners. The Naive/filt approach acts like the Naive one because pre-filtering removes some moves, but does not remove equivalence classes, nor actions in these classes. The number of strategies to check thus remains the same. So the approach finds a winning strategy for 2 prisoners but not for 3.

**Partial approaches** The Partial approach is lucky to find a winning strategy for 2 prisoners. For 3 prisoners, it does not find a good one within 30 minutes. The Partial/filt one benefits from pre-filtering and even achieves to find a winning strategy for 3 prisoners, but fails for 4 prisoners.

**Early approaches** The Early approach is quicker than the previous approaches to find a winning strategy for 2 prisoners. It is explained by the fact that as soon as the current strategy considers an incorrect announcement, the approach stops extending it because it is surely losing. The Early/filt one performs even better for 2 prisoners than the Early approach, but it is the opposite for 3 prisoners. As the time needed to perform pre-filtering is negligible, this simply means that the Early approach makes better decisions than the Early/filt one for 3 prisoners.

**Symbolic approaches** The Symbolic approach is really efficient for 2 prisoners, but is less for 3. This is because there are sufficiently few strategies to encode in the former case, but too many for the latter. The other approaches that succeed in finding winning strategies for 3 prisoners simply made the right choices, as they are not able to check all possible strategies within 30 minutes, while the symbolic approaches have to check them all. The Symbolic/filt approach behaves like the Symbolic one because pre-filtering does not remove any equivalence classes, nor any actions in these classes.

**Comparison** As for the previous model, the number of strategies to consider is huge. The naive approaches consider the same number of strategies, and have to check them all. The Partial approach is quicker for 2 prisoners because there are fewer partial strategies that are adequate for the initial state. These three approaches fail at finding a winning strategy for 3 prisoners.

The number of partial strategies to check by the Partial/filt approach is lower than for the Partial one. This means that pre-filtering is useful here. On the other hand, the partial approaches are less efficient than the early ones because the latter can more easily rule out strategies. Both early approaches behave similarly.

Finally, the symbolic approaches both behave in the same way, and are really efficient when considering the strategies for 2 prisoners. Nevertheless, for 3 prisoners, the number of strategies is too large and the other approaches make the right choices and find a winning strategy within the time limit, while the symbolic approaches have to consider all strategies at once to conclude.

### 7.2.4 BDD variable reordering techniques

The implementation of the approaches is based on BDDs. It is well-known that ordered binary diagrams performances are highly correlated to the order of their variables. The implementation being based on NuSMV, it inherits all the functionalities of the tool to dynamically reorder the BDD variables. NuSMV proposes 19 different heuristics for dynamic reordering. To assess their performances, 18 of these heuristics have been tested with the approaches (see the NuSMV Manual for more information about the different reordering heuristics [CCJ+]). The omitted one is the *exact* heuristics that computes the optimal order. It can take a lot of time and is not advised with more than 16 Boolean variables [CCJ+]. The 18 heuristics have been tested on different formulas, one for each approach. These formulas have been chosen such that the time needed to solve the model-checking problem is not too low—when it is too low, other quick computations such as building the model can have a significant impact on the overall model-checking time—, and not too high—hitting the 1800 second timeout would yield no useful information for comparing the heuristics. As for the previous tests, each formula has been checked 20 times. The formulas are:

- $\phi^P$ with 2 prisoners for the *Naive* approach;
- $\phi^T_3$ with 4 horses for the *Naive/filt* approach;
- $\phi^T_1$ with 6 horses for the *Partial* and *Partial/filt* approaches;
- $\phi^C_1$ with 4, 4 and 4 workers for the *Backward* approach;

- $\phi_2^T$ with 4 horses for the *Early* approach;
- $\phi_3^T$ with 4 horses for the *Early/filt* approach;
- $\phi_1^C$ with 2, 1 and 2 workers for the *Symbolic* and *Symbolic/filt* approaches.

Figure 7.7 shows the time needed for the Partial approach with pre-filtering to verify the formula $\langle\!\langle \mathit{Tian\ Ji} \rangle\!\rangle \mathbf{F}\ \mathit{Tian\ Ji\ wins}$ on the model of Tian Ji with 6 horses.

First, these tests show that no heuristic is better than another. The approach takes about 110 seconds to verify the formula, regardless of the reordering technique. Second, these tests show the variability stays small: the approach usually needs from 100 to 120 seconds to perform the verification.

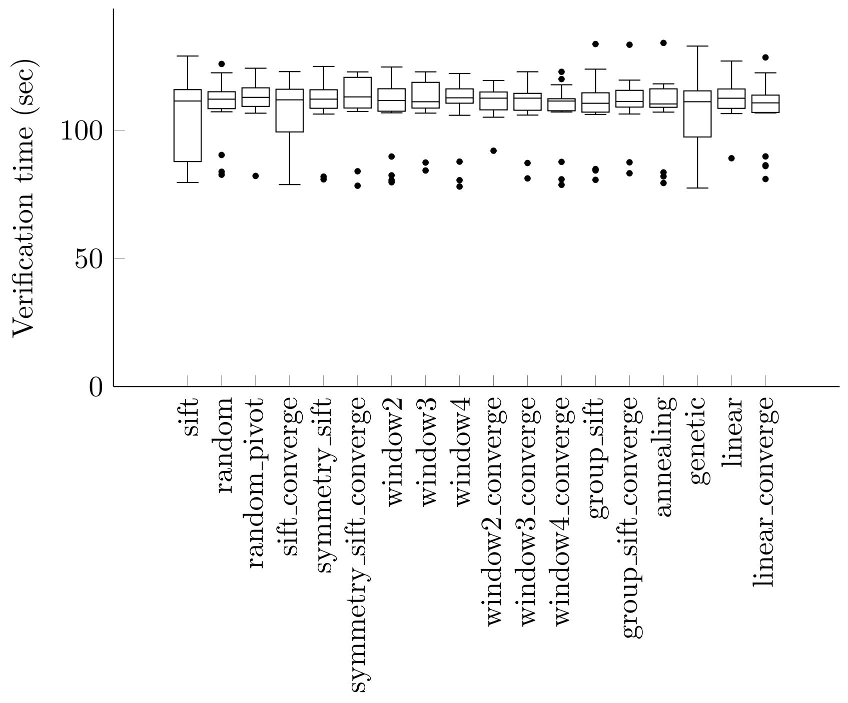


Figure 7.7: Time taken with all variable reordering heuristics with the *Partial/filt* approach for checking formula $\phi_1^T$ on the model of Tian Ji with 6 horses.

Figure 7.8 shows the time needed for the Early approach to verify the formula $\langle\!\langle \mathit{Tian\ Ji} \rangle\!\rangle \mathbf{F}\ \langle\!\langle \mathit{Tian\ Ji} \rangle\!\rangle [\neg \mathit{King\ wins}\ \mathbf{U}\ \mathit{Tian\ Ji\ wins}]$ on the model of Tian Ji with 4 horses. Again, these results show that no heuristic is better than another and that the variability stays small.

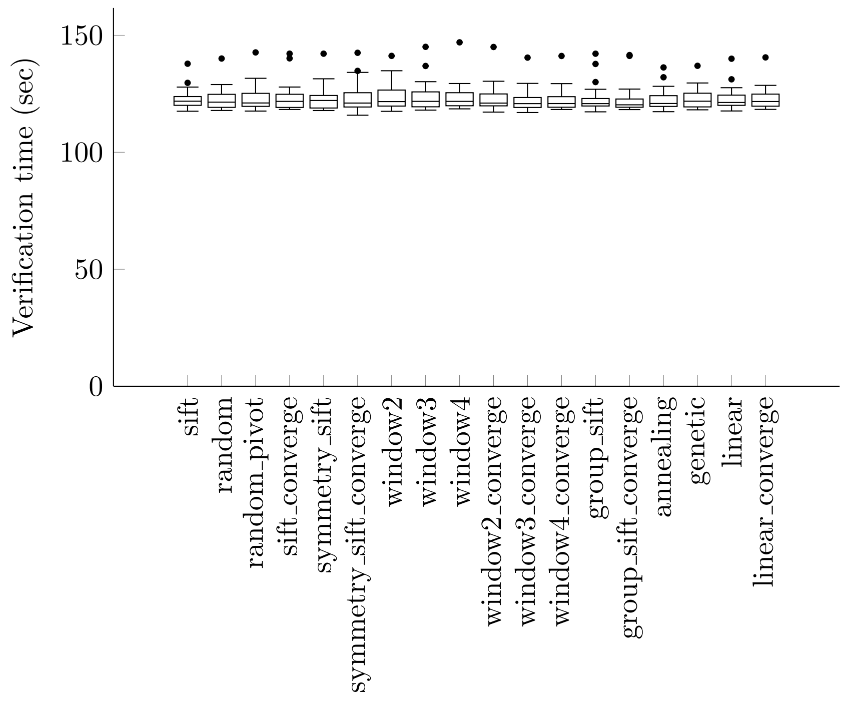


Figure 7.8: Time taken with all variable reordering heuristics with the *Early* approach for checking formula $\phi_2^T$ on the model of Tian Ji with 4 horses.

The other tested approaches showed similar results. For all reordering heuristics, the *Naive* approach took about 500 seconds to verify the formula, the *Naive/filt* one took about 200 seconds, the *Partial*, *Backward*, *Early/filt* and symbolic ones took about 100 seconds. No heuristic showed better performances in verifying the formulas, with all tested model sizes and approaches. This leads to the conclusion that the chosen heuristic is not important in the present cases and that choosing the default *sift* one is a sensible choice. This heuristic has been used for all the other tests presented in this chapter.

### 7.2.5 Conclusions on the experiments

Based on the observations made on the different experiments presented above, we can draw some general conclusions.

The best approach to check that there exists a winning strategy when most of them are winning is the Partial approach. Nevertheless, it performs really poorly when showing that there are no or few winning strategies.

The early approaches present a better trade-off since they take more time to show that there is a winning strategy if most of them are winning, but can more easily find one when there are only few winning ones, or even show that there are no winning strategies. Nevertheless, in the case in which only complete partial strategies are winning, and not some incomplete ones that can be extended in winning ones, the early approaches tend to perform extra useless work. Indeed, they have to extend a strategy completely to find a winning one, and the intermediate computations of losing and winning states do not yield any gain.

The symbolic approaches work better when there is a huge number of strategies to consider because they can represent them in a compact way. On the other hand, the other approaches cannot handle a huge amount of strategies since they need to enumerate them. Furthermore, the symbolic approaches also work well with nested strategic formulas.

Pre-filtering may or may not help. Either it removes a lot of losing moves when there are big parts of the model in which the agents have no winning strategies at all (even non-uniform ones), or it removes only a few of them, producing extra useless work. In some cases, it can even directly conclude that there are no winning strategies, avoiding the need to check any strategy.

The symbolic approaches are more stable than the others, in the sense that the time needed to perform the model checking is less dependent on the checked property. This can be seen on the first two formulas of the model of Tian Ji, that the Symbolic approach can evaluate for 4 but never for 5 horses in the available time. On the third formula, the $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}$ operator is easier to deal with because the fixpoint computation is simpler, allowing the approach to evaluate the formula for 5 horses. On the other hand, the other approaches have very variable performances depending on the formula.

While the approaches can be efficient in finding winning strategies when there are a lot of them, or to check that there are no winning strategies when there are not even non-uniform ones, they can be very unpredictable when there are only few winning strategies. Partial and early approaches can take a long time to find the winning combinations of moves, and symbolic approaches can have more complicated BDD manipulations to perform in this case.

The Backward approach has been really efficient in finding strategies that reach a given objective. Nevertheless, the limitations of the approach—no fairness constraints, only $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}$ and $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}$ objectives—limited the applicable cases.

While NuSMV and PyNuSMV propose some heuristics to dynamically reorder the BDD variables, the tests showed that none of them is better than the others.

In summary, there is no best approach for all cases. The early approaches present a good trade-off between the cases where there are winning strategies and the cases where there are not. The symbolic approaches beat the others when the number of uniform strategies is really huge. And the backward approach is really good on cases it can handle. Finally, pre-filtering is useful in some cases but produces extra useless work in others.

# *Chapter 8*

# *Part I: Conclusion*

The first part of this thesis describes a new logic, $ATLK_{irF}$, to reason about time, knowledge and uniform strategies of agents under fairness constraints. It also presents algorithms to check $ATLK_{irF}$ formulas on imperfect information concurrent game structures.

The logic combines the $CTL$ operators **EX**, **EU** and **EW** (as well as their dual operators), the knowledge operators **K**, **D**, **E** and **C**, and the $ATL$ strategic operators $\langle\!\langle \Gamma \rangle\!\rangle \mathbf{X}$, $\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}$ and $\langle\!\langle \Gamma \rangle\!\rangle \mathbf{W}$.

The temporal and epistemic operators are interpreted in the usual way. The strategic operators reason about the existence of uniform strategies of a group of agents with imperfect information for a given objective. The structures on which the logic is interpreted, called iCGSf, embed fairness constraints to rule out unfair behaviors, and a uniform strategy of a group $\Gamma$ is said to be winning if all the *fair* paths enforced by this strategy satisfy the objective.

Such a logic is useful, for instance, to reason about *multi-agent programs* that describe the actions, beliefs and goals of agents with imperfect information [DJ10]. They are usually executed asynchronously and need a scheduler to choose which program will act next. In this context, it is interesting to reason about the strategies and the knowledge of these agents under the assumption of a fair scheduler, and $ATLK_{irF}$ can be used to this end.

Model checking $ATLK_{irF}$ formulas over the states of iCGSf is a difficult problem. Chapters 4 and 5 showed that this problem is $\Delta_P^2$-complete, that is, it needs a polynomial number of calls to an NP oracle to be solved.

To solve this model-checking problem, this thesis proposes three different approaches. The first approach is called the *naive* one. It consists in enumerating all uniform strategies of the group $\Gamma$ and, for each strategy, computing the states for which it is winning by using fixpoint computations adapted to the objective $\psi$. This approach is simple to understand and to implement, but experiments showed that it is highly ineffective.

To overcome the ineffective naive approach, this thesis proposes the *partial* approach. It is based on the notion of partial strategies that are adequate for some states of interest. The key idea is that it is sufficient to check all adequate partial strategies to conclude, instead of all possible strategies, and there are fewer partial strategies. Furthermore, thanks to optimizations such as early termination and caching, the partial approach can be way more efficient than the naive one.

Another proposed improvement is pre-filtering. The idea is that, while it is costly to compute the winning *uniform* strategies, it is way cheaper to compute the moves that belong to winning *general* strategies. Ignoring the moves that are not part of a winning general strategy allows the approaches to decrease the number of strategies to check before concluding. Nevertheless, practical experiments showed that pre-filtering can be useless in some cases.

A third approach is the *backward* one. The idea is to generate the winning strategies from the target states, instead of generating them from the states of interest such as the initial states. Nevertheless, this approach cannot handle fairness constraints and $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}$ formulas.

Two existing approaches have been adapted to the case of $ATLK_{irF}$. The first adapted approach is the *early* approach. It is based on the work of Pilecki et al. [PBJ14]. The main idea is, instead of computing adequate partial strategies, to generate incomplete ones, and to check whether they can be extended into winning adequate partial strategies. More precisely, this approach alternates between extending a partial strategy and checking whether (1) all extensions are surely winning, or (2) no extension can be winning. In the first case, the process can be stopped there, with a winning uniform strategy. In the second case, the process can restart with another partial strategy. The early approach has also been extended to take pre-filtered moves into account.

The early approach improves the original algorithm of Pilecki et al. in several ways. Their approach is limited to one initial state and to one top-level strategic operator. Furthermore, it does not support fairness constraints. The early approach removes all these limitations. Finally, Pilecki et al. only check whether *all* extensions of one partial strategy are winning while the early approach also checks whether no extension can be winning, and can thus ignore losing strategies more quickly.

The early approach shares some ideas with the backward one. They both start with small partial strategies and try to extend them until they are either surely winning or surely losing. The difference between them is that the early one starts with the states of interest while the backward one starts with the target states. This difference is why the early approach can handle any $ATLK_{irF}$ model and formula, while the backward one cannot handle fairness constraints and $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}$ formulas.

The second adapted approach is the *symbolic* one. It is based on the work of Huang and van der Meyden [HvdM14a, HvdM14b]. The main idea is to build, from the original structure, a derived structure in which the strategies are encoded in the states. Then the model-checking problem is solved with fixpoint computations on this derived structure. In other words, while the previous approaches explicitly enumerate the strategies, this one proposes to encode them in a derived structure and to manipulate them symbolically. The approach has also been extended to take pre-filtered moves into account.

The symbolic approach is a restriction of the algorithm of Huang and van der Meyden. The logic they support subsumes $ATLK_{irF}$ and can express more properties. Nevertheless, restricting the idea to $ATLK_{irF}$ leads to simplifications of the algorithm, such as the abandon of the additional strategic agents. Furthermore, Huang and van der Meyden did not considered pre-filtering losing moves.

BDD-based model-checking approaches are called *symbolic* because they deal with sets of states—defined by BDDs—instead of individual states. In this sense, the naive, partial, backward and early approaches are symbolic ones, as they deal with sets of states defined by BDDs. Nevertheless, they deal with strategies in an explicit way: instead of working with sets of strategies, they work with individual ones defined with BDDs. They can thus be seen as *semi-symbolic* approaches. On the other hand, the *symbolic* approaches are *fully symbolic* ones: they deal with sets of strategies by encoding them directly into BDD variables.

The advantage of enumerating the strategies is the fact that the BDDs remain relatively small. Furthermore, the approaches can be very efficient for finding winning strategies: as soon as they found one, they can stop the process thanks to early termination. On the other hand, as the symbolic approaches deal with all strategies at once, they cannot stop as soon as a winning strategy is found. Nevertheless, enumerating all strategies can become really time consuming when their number is very large. This has been shown with the second formula on the Tian Ji's model, for which all strategies must be checked to determine that the formula is false. On the other hand, the symbolic approaches handle all strategies at once. This leads to larger BDDs as the derived structures encode more information into BDDs. Nevertheless, these approaches can benefit from the full power of BDDs to represent sets of strategies efficiently.

The approaches have been compared through practical experiments using three models and six formulas. These experiments compared the time needed to solve the model-checking problem. As expected, they showed that the naive approach is not efficient at all as there is no mechanism in this approach to make it efficient. On the other hand, some formulas have been checked faster with the partial approaches, others with the early ones, or with the symbolic ones. Furthermore, pre-filtering has been shown to be useful in some cases, but a waste of time in others. Based on these results, we can conclude that the three approaches have comparable performances. Finally, the backward approach showed very good performances compared to the other approaches on the small subset of formulas it could handle.

## 8.1 Comparison with related work

This section discusses the link between the approaches presented in Section 3.5 and the ones described in this thesis.

**Lomuscio and Raimondi** Lomuscio and Raimondi propose to perform the model checking of uniform strategies by enumerating all Γ-uniform systems compatible with a given interpreted system, and by concluding that the original system satisfies the formula if there exists a Γ-uniform compatible system satisfying the formula [LR06b].

This approach is similar to the naive one without pre-filtering: compatible Γ-uniform interpreted systems are closely related to uniform strategies, as both define sub-models in which the agents act uniformly. Furthermore, both approaches work by enumerating the strategies or compatible Γ-uniform interpreted systems, and by finding winning ones to conclude that the formula is satisfied.

Nevertheless, as already explained in Section 3.5, the semantics handled by the two approaches are different. In the case of Lomuscio and Raimondi, the strategies must be winning for the whole formula, that is, the same uniform strategy must be winning for all strategic sub-formulas, and for all states of interest. On the other hand, $ATLK_{irF}$ semantics is local to the sub-formula, and two different formulas or distinguishable states can have a different winning strategy. $ATLK_{irF}$ semantics follows the ideas of $ATL$ and $ATL_{ir}$, where two strategic sub-formulas can be satisfied because of two different strategies, while the semantics of Lomuscio and Raimondi diverges from them.

**Calta et al.** Calta et al. propose an algorithm to model check $ATL_{ir}$ formulas over sets of states of iCGS [CSS10]. The problem they solve is closely related to the model checking of $ATLK_{irF}$. Nevertheless, this approach is tailored to the case of $ATL_{ir}$ and it is difficult to adapt it to the case of $ATLK_{irF}$.

**Pilecki et al.** The *early* approach described in this chapter is based on the one presented by Pilecki et al. [PBJ14]. Nevertheless, the original approach has been heavily extended, as explained above.

First, the approach described by Pilecki et al. is limited to one initial state and one top-level strategic operator. Those limitations introduce simplifications to the problem they solve: they can stop the process as soon as they find a winning strategy for this single initial state, instead of keeping track of the states for which they already know the truth value, as proposed with early termination. Furthermore, considering only one state prevents the technique to be applied to sub-formulas for which we need to considerate the same strategy for different states. The *early* approach of this thesis handles any subset of states of interest, making it usable to check strategic sub-formulas in subsets of states.

Second, the approach of Pilecki et al. only uses the idea that we can stop looking for winning strategies as soon as all extensions of the current partial strategy are winning (corresponding to Lines 12 to 17 of Algorithm 6.2). The approach described in this thesis extends the original idea of Pilecki et al. by also trying to filter out states for which we know no winning extension exists.

Third, the *early* approach has been extended with pre-filtering, and the experiments of the previous chapter showed that this extension can be useful to make the process more efficient.

Pilecki et al. do not describe in which order the strategies are selected by their algorithm. They can be explored in a depth-first search manner, taking one strategy and extending it as long as necessary, or they could be explored in any other manners. On the other hand, the *early* approach is tailored to a depth-first search by design. It would be interesting to adapt the latter to support other exploration orders, and to measure their impact on experimental results.

Fourth, the approach of Pilecki et al. does not handle fairness constraints. Finally, while the *early* approach has been designed to be implemented with binary decision diagrams, the algorithm of Pilecki et al. has been implemented in an explicit framework.

**Epistemic Strategy Logic** The *symbolic* approach presented in this thesis is a particular case of the approach of Huang and van der Meyden [HvdM14b]. Their work describes a more general logic than $ATL_{ir}$ that explicitly quantifies over strategies of agents [HvdM14a], and explains how to encode the model-checking problem of $ATL_{ir}$ [HvdM14b]. Thanks to their logic, it is, for instance, possible to express the fact that an agent has a strategy to ensure two objectives at the same time.

On the other hand, the symbolic approach presented in this thesis is tailored to the verification of $ATLK_{irF}$. More precisely, the original approach of Huang and van der Meyden is tailored by restricting the syntax of the logic to $ATLK_{irF}$, that is, asking every strategy quantification being followed by only one objective. Furthermore, the original work of Huang and van der Meyden defines so-called *strategic agents* to be able to reason about states of $EncStrats(S)$ that share the same strategy. In our setting, $EquivQEqStr$ is defined such that these strategic agents are not necessary. They are thus omitted.

Nevertheless, the approach of Huang and van der Meyden is slightly different from the *symbolic* approach. By construction, Huang and van der Meyden force every agent to act uniformly. Indeed, they encode the uniform strategies of all agents in the derived structure, and the transition relation is defined such that all agents follow the strategy encoded in the current state. In the context of $ATLK_{irF}$, the agents outside $\Gamma$ can act freely and they do not have to make uniform choices. This is achieved by encoding different transition relations for different groups of agents. In other words, the semantics supported by the work of Huang and van der Meyden puts more limitations on the capabilities of the opponents than $ATLK_{irF}$ semantics. Finally, the *symbolic* approach has been extended with pre-filtering, while Huang and van der Meyden do not consider this extension.

**Observation-based two-player games** Raskin et al. propose an approach to check the existence of winning observation-based strategies with perfect recall in the context of two-player games [RCDH07]. Their algorithm is based on fixpoint computations on sets of states and works with antichains. The objectives of their strategies are $\omega$-regular ones, such as Büchi and coBüchi objectives.

Similarly, Bozianu et al. propose another antichain-based algorithm for checking the existence and synthesizing strategies with imperfect information and perfect recall in the context of two-player games [BDF14]. The objectives of their strategies are defined using an extension of $LTL$ with knowledge operators.

These problems are related to the problem of $ATLK_{irF}$ model checking. Nevertheless, they put different limitations on the problem to make it decidable. Remember that $ATL^*_{iR}$ model checking is undecidable. To solve this problem of undecidability, $ATL_{ir}$ and $ATLK_{irF}$ are limited to memoryless strategies. The approaches of Raskin et al. and Bozianu et al. limit the games to be two-player games. In this context, checking the existence of winning memory-full uniform strategies becomes decidable. Furthermore, the approaches they propose, based on antichains, are far from the BDD-based algorithms proposed in this thesis.

## 8.2 Future work

Many improvements can be made to the approaches presented in the first part of this thesis. This section discusses some of them as future work.

**Experiments with real-life cases** Chapter 7 performed some experiments. But the tested models and formulas are toy models based on the ones found in the literature. They do not necessarily reflect the structure, size and complexity of real-life cases. It would be interesting to find real-life cases of iCGSf and to test the approaches on them. Nevertheless, one of the main obstacles with such cases would be their size as the experiments showed that the approaches do not scale well.

**Investigating the initial order of BDD variables** Some experiments of Chapter 7 showed that, while NuSMV and PyNuSMV propose several heuristics to dynamically reorder the BDD variables during the process, none of these heuristics had a significant impact on the performance. Nevertheless, there exist other ways to improve the performance of BDD-based algorithms by manipulating this variable order. In particular, the initial order of the variables plays a big role in the successive orders computed by the heuristics. For the experiments of this thesis, the standard initial order defined by the order in which the variables have been declared in the description of the model has been used. It would be interesting to evaluate the impact of this initial order and to define, if possible, general guidelines for specifying efficient initial orders.

**Fine-tuning pre-filtering** The idea of pre-filtering is to ignore the moves that cannot be winning for the group of agents when dealing with a particular objective. By ignoring some moves, the number of strategies to consider is lowered, reducing the effort to find a winning one.

As shown by the experiments, pre-filtering can drastically reduce the number of strategies to consider, but can also lead to extra useless work, when all moves belong to some general strategy to win the objective. As pre-filtering can help or not, depending on the model and the strategic formula, it would be interesting to extend the approaches to allow the user to specify which strategic sub-formulas should be handled with pre-filtering, and which should not. This would give her a finer control on how and when pre-filtering is applied. An even better solution would be to design heuristics to automatically determine whether pre-filtering should be applied or not for each sub-formula.

Furthermore, pre-filtering triggers the evaluation of sub-formulas on a large part of the model. When the sub-formulas include strategic ones, the effort to perform the model checking on such a large subset of states—usually the whole set of reachable states—is large. Some approximations could be used to evaluate these sub-formulas. For instance, the strategic sub-formulas could be evaluated under perfect information, needing the application of a polynomial algorithm instead of a $\Delta_P^2$ one. This would lead to less ignored moves, as the approximation would keep some moves that should be excluded, but pre-filtering would be way less costly when dealing with strategic sub-formulas.

**Investigating other traversal strategies** It has already been mentioned that the original algorithm proposed by Pilecki et al. does not define an order in which the strategies are traversed, while the adapted *early* approach makes the choice to enumerate them in a depth-first search manner (DFS). In fact, all the three semi-symbolic approaches—the naive, partial, and early ones—enumerate the strategies in a DFS manner. It would be interesting to modify them to be able to use different orders such as breath-first search, and to evaluate their relative performances on different models and formulas.

**Fully forward exploration** The semi-symbolic approaches presented in this thesis are based on $filter$ algorithms that perform a *backward* exploration of the structure under investigation to evaluate a given strategy, and on algorithms that perform a *forward* exploration —or a *backward* one, for the backward approach—of the same structure to generate the strategies. It would be interesting to develop algorithms that perform both explorations in a forward manner. Nevertheless, some limitations have been highlighted when performing a forward exploration for the model checking of $CTL$ formulas (see for instance [INH96] and [HKQ98] for more information). It would be necessary to study these limitations to determine whether they also apply to $ATLK_{irF}$ model checking.

**Reducing the number of choices of actions** The number of possible uniform strategies is directly linked to the number of actions an agent can choose in equivalence classes of indistinguishable states. Finding ways to reduce this number of actions can thus lead to performance improvements. Given a state $q$ and two actions $a_1$ and $a_2$ for agent $ag$, if playing $a_1$ or $a_2$ leads to the same set of states—that is, if the opponents of $ag$ can force the next state to be in the same set of successors, for both actions—then we can consider that these two actions are equivalent. By grouping actions into equivalent classes, we could thus reduce the number of choices that matter, and reduce the number of strategies to consider. The gain could be even higher by considering groups of agents instead of individual ones.

**$CTL^*$-like objectives** $ATLK_{irF}$ has been limited to $CTL$-like objectives, that is, $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}$, $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}$ and $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}$ ones. It would be interesting to investigate the introduction of more general objectives consisting in any combination of the propositional and path operators, à la $CTL^*$. Two major problems should then be tackled: (1) how to check a particular (partial, incomplete) strategy for such an objective, and (2) how to adapt pre-filtering. One advantage would be that, in this case, handling fairness constraints could directly be done within the logic itself.

**Mixing approaches** The previous chapter showed that the partial, early and symbolic approaches are comparable in efficiency, and that they are efficient in different situations. It would be interesting to mix them, to evaluate different sub-formulas using different approaches. For instance, given the formula $\langle\!\langle\Gamma_1\rangle\!\rangle\mathbf{F}\ p \wedge \langle\!\langle\Gamma_2\rangle\!\rangle\mathbf{G}\ q$, the first sub-formula $\langle\!\langle\Gamma_1\rangle\!\rangle\mathbf{F}\ p$ could be evaluated with the backward approach while the second one $\langle\!\langle\Gamma_2\rangle\!\rangle\mathbf{G}\ q$ could be evaluated with the early approach. The approach to use for each sub-formula could be defined by the user, but it would also be interesting to investigate criteria to automatically select the best approach.

# Part II

# Rich diagnostics for multi-modal logics

# *Chapter 9*

# *Explanations for modal logics*

A major benefit of model checking is the capability to generate a counter-example when a property is not satisfied. For linear temporal logics such as $LTL$, such a counter-example is a single path of the system. On the other hand, the counter-examples of $CTL$ are not linear in general [BEGL01], and their branching feature is even more critical when dealing with multi-modal logics such as $CTLK$, or with the µ-calculus.

To illustrate the need for tools and functionalities to handle complex counter-examples, let us consider the $CTL$ formula **AF AG** *initialized* saying that *the system always reaches a state from which it stays initialized forever.* A counter-example for this formula is not necessarily linear as it has to exhibit an infinite path such that, from each state, there exists another path reaching a non-initialized state. Such a counter-example is illustrated in Figure 9.1.

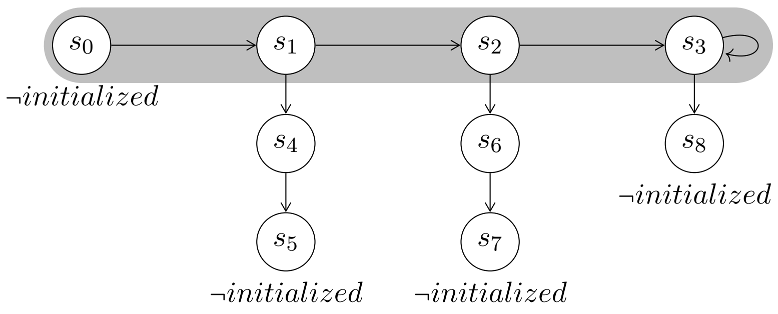


Figure 9.1: A branching counter-example for the formula **AF AG** *initialized*. The main path is highlighted.

Unfortunately, most of the current state-of-the-art model checkers such as NuSMV only return linear counter-examples for $CTL$ formulas [CGMZ95]. In particular, for the formula above, NuSMV would simply return the main path, without including the secondary paths. On the other hand, there exists a lot of research on defining, generating and presenting such kinds of rich explanations. This chapter describes the existing work on the subject. Section 9.1 first summarizes the solutions brought for solving the problem of representing, generating and visualizing explanations for $CTL$. Section 9.2 describes the solutions proposed for extensions of $CTL$ such as temporal epistemic logics. Section 9.3 presents tree-like annotated counter-examples for temporal-epistemic logic. Section 9.4 discusses the solutions for the µ-calculus.

## 9.1 Explanations for $CTL$ model checking

$CTL$ has branching counter-examples. Solutions are thus needed to represent them, generate them, and visualize and manipulate them. Many solutions to these problems have been proposed for the past 25 years, starting from the work of Rasse [Ras92].

### 9.1.1 Branching explanations for $CTL$ model checking

**Explanations based on explicative sequences** Rasse describes explanations for a variant of $CTL$ without the *next* operator ($\mathbf{X}$) interpreted over Kripke structures with non-serial transition relations labelled with actions [Ras92]. These explanations are composed of lists of paths of the systems—called *explicative sequences*—with states and paths labelled with the sub-formula they explain, and transitions labelled with actions of the system. They provide a reason for why existential quantifiers are satisfied by particular states of the system.

The explicative sequences are paths of the structure. Their shape depends on the formula they explain. For instance, an explicative sequence for the formula $\mathbf{E}[\phi_1 \mathbf{U} \phi_2]$ is a finite path of states satisfying $\phi_1$ reaching a state satisfying $\phi_2$.

In addition to the structure of explanations, Rasse discusses about minimal explicative sequences, from which minimal explanations are built. Intuitively, an explicative sequence is minimal if either it is finite and has no cycle (no repeating state), or it is infinite and has only one so-called elementary cycle. Minimal explanations have the same structure as standard ones but are composed of minimal explicative sequences.

Rasse also describes a way to simplify explanations according to a given set of visible actions, and briefly presents the implementation of the technique in the tool Cléo.

**Justifications for tabled logic programming** Roychoudhury et al. propose to produce justifications from tableau-based proof systems executed by a tabled logic programming system [RRR00]. Such a logic programming system uses tables to store explored proof steps to avoid infinite inference loops. When a query is proved true or false, it is difficult to understand why. Indeed, standard debugging techniques for logic programming allow the user to inspect the search for a proof—with break points, inspection, assertions, etc.—but are not suited to get the part of the search tree that is responsible for the final outcome.

Roychoudhury et al. propose an algorithm to extract, from the tables used to store intermediate proof steps, the smallest part of the proof tree responsible for the final outcome, that they call *justifications*. More precisely, a justification for explaining why a query $Q$ is true of false in a given proof system $R$ is a directed acyclic graph (DAG) of proof steps—with leafs being `fail` or `fact` nodes indicating false and true ground results, respectively—, rooted at the node $Q$, and where edges are applications of the rules of the proof system. Furthermore, justifications use the keyword `ancestor` to pinpoint looping branches—with no indication of the exact ancestor along the history. Justifications contain the necessary steps needed to show the final outcome of the proof. Finally, the DAG can be reduced to match the high-level proof system instead of its encoding into the logic programming system.

Roychoudhury et al. apply the technique to $CTL$ model checking performed by encoding the model along the rules expressing the semantics of $CTL$ operators. Model checking is then performed by writing a query encoding whether the model satisfies the given formula or not. For instance, Figure 9.2 presents a simple Kripke structure and Figure 9.3 shows the justification for why $q_0$ does not satisfy the **AG** $p$ formula.

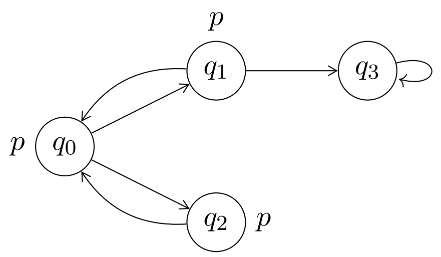


Figure 9.2: A simple Kripke structure [RRR00].

The same research group later improved the algorithm generating justifications by taking into account non-tabled proof steps [GRR01]. These steps are not present in the memo tables, making the original

$$\begin{tikzcd}
 & \texttt{models(q0, ag(p))} \arrow[d] & & \\
 & \texttt{tnot(models(q0, ef(not(p))))} \arrow[d] & & \\
 & \texttt{models(q0, ef(not(p)))} \arrow[dl] \arrow[dr] & & \\
\texttt{trans(q0, q1)} & & \texttt{models(q1, ef(not(p)))} \arrow[dl] \arrow[dr] & \\
 & \texttt{trans(q1, q3)} & & \texttt{models(q3, ef(not(p)))} \arrow[d] \\
 & & & \texttt{models(q3, not(p))} \arrow[d] \\
 & & & \texttt{fact}
\end{tikzcd}$$

Figure 9.3: A justification for why $q_0$ violates $\mathbf{AG}\ p$ [RRR00].

algorithm evaluate them repeatedly. The improved algorithm uses *lazy generation* of justification and *sharing* of common sub-justifications to reduce both the time and the memory needed to generate justifications.

A second algorithm has been proposed to further improve the performances of generating justification [PGD+04]. Instead of using the data stored in memo tables a posteriori to generate justifications, this new algorithm generates them during the evaluation process. This is done by deriving, from the original proof system, a new proof system that computes the justification at evaluation time. Experimental results with XMC, a µ-calculus model checker [RRS+00], showed that generating the justification on-the-fly instead of after the evaluation really improves the generation process with little overhead to the evaluation time.

*CTL* **multi-paths** Buccafurri et al. got interested in the subset of *ACTL* formulas that have linear counterexamples. These formulas $\phi$ are such that, whenever a state $q$ of a structure $S$ violates $\phi$, there exists an infinite path completely showing why the violation occurs [BEGL01]. In this context, they make several contributions:

- they precisely capture the notion of counterexample and linear counterexample through their definition of *multi-paths*;
- they show that whenever $S, q \not\models \phi$, it is NP-hard to determine whether there exists a linear counterexample;
- they show that it is PSPACE-hard to decide whether a given *ACTL* formula always admits a linear counterexample in case of violation;

- they define a set of *templates* of $ACTL$ formulas such that any instantiation of such a template yields a formula for which there exists a linear counterexample, whatever the structure under interest;
- they provide a polynomial time algorithm to produce linear counterexamples for instantiations of the templates they defined.

Buccafurri et al. capture the notion of counterexample with *multi-paths*. A multi-path is either a single state $q$ or an infinite sequence of multi-paths. A multi-path represents an infinite tree with the main path serving as a backbone. For instance, $\langle\langle q_0, q_1, q_1, ...\rangle, q_2, q_2, ...\rangle$ is a multi-path representing a tree with two branches: the main branch is $q_0, q_2, q_2, ...$, and the other branch is $q_0, q_1, q_1, ...$. This multi-path is also illustrated in Figure 9.4.

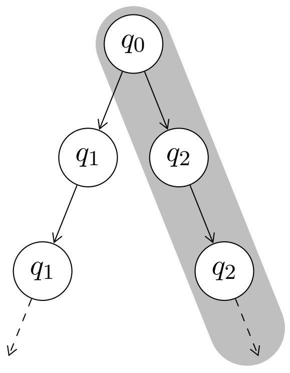


Figure 9.4: A simple multi-path. The main path is highlighted [BEGL01].

The proposed generating algorithm takes advantage of the fact that the given formula must have a linear counterexample by its structure, and works inductively on the structure of the formula to produce one. For instance, if $\phi = \mathbf{A}[p\ \mathbf{V}\ \phi']$ is violated[1]—that is, $\mathbf{E}[\neg p\ \mathbf{U}\ \neg\phi']$ is satisfied—, then the algorithm produces a path of states violating $p$ and reaching a state violating $\phi'$, makes it the backbone of the linear counterexample, and expands it by appending a counterexample for why the last state of the path violates $\phi'$.

**Tree-like counter-examples** Clarke et al. propose *tree-like counter-examples* for the universal fragment of an extended branching time logic based on $\omega$-regular operators called $A\Omega$ [CJLV02]. This fragment subsumes $ACTL$, the universal fragment of $CTL$. The logic $A\Omega$ is based

[1]The $\phi_1\ \mathbf{V}\ \phi_2$ path operator is a variant of the $\mathbf{U}$ and $\mathbf{W}$ operators such that $\phi_1\ \mathbf{V}\ \phi_2 \equiv \neg(\neg\phi_1\ \mathbf{U}\ \neg\phi_2)$.

on atomic propositions, the propositional operators $\wedge$ and $\vee$, the universal path quantifier **A**, and so-called $\omega$-regular linear time operators. These operators describe paths that can be checked against regular expressions composed of the special symbol $\perp$ (don't care), and one symbol per sub-formula. For instance, the regular expressions corresponding to the path formulas $\mathbf{G}\ \phi_1$ and $\phi_1\ \mathbf{U}\ \phi_2$ are given by $(M_1)^\omega$ and $(M_1)^* M_2$ respectively, where $M_i$ is the symbol corresponding to states satisfying $\phi_i$. The key characteristics of such path operators is their *monotonicity*, ensuring the fact that there exists a tree-like counter-example for any violated $A\Omega$ formula.

A tree-like counter-example is a Kripke structure with a tree-like underlying graph. A directed graph is tree-like if every strongly connected component (SCC) is a simple cycle, and the graph where every SCC is replaced by a single node is a directed acyclic graph. For instance, Figure 9.5 presents a tree-like counter-example for the $ACTL$ formula $\mathbf{AG}\ \neg x \vee \mathbf{AF}\ \neg y$—that is, a witness for $\mathbf{EF}\ x \wedge \mathbf{EG}\ y$.

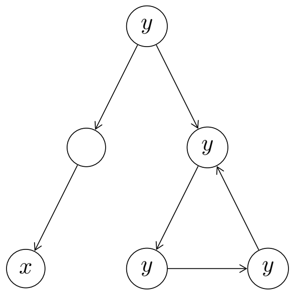


Figure 9.5: A tree-like counter-example for $\mathbf{AG}\ \neg x \vee \mathbf{AF}\ \neg y$ [CJLV02].

The paper also presents a symbolic algorithm to build these counter-examples. It supposes the existence of a sub-algorithm that can produce a path matching the regular expression of any $\omega$-regular operator. Such a sub-algorithm can be implemented by building the Büchi automaton corresponding to the regular expression of the operator and finding an accepted path in the product of the automaton and the Kripke structure violating the formula. With this sub-algorithm, the generating algorithm extracts a path showing why the top-level $\omega$-regular operator is violated and recursively calls itself to generate the sub-branches of the counter-example explaining why sub-formulas are violated.

Finally, the paper illustrates the applicability of the presented technique to two use cases: $ACTL$ model checking and counter-example guided abstraction refinement.

### 9.1.2 Proof-like counter-examples

Gurfinkel and Chechik propose the idea of proof-like counter-examples for $CTL$ [GC03a] and for a multi-valued extension of $CTL$ [GC03b]. These counter-examples are derived from the set of rules of a proof system for $CTL$, and show every proof step explaining why a formula is satisfied by some state of some Kripke structure. For instance, Figure 9.6 shows a simple proof (with notations adapted from the original paper) showing why some state $q_1$ of some Kripke structure satisfies the $CTL$ formula $\mathbf{EX}\ p$, by exhibiting a successor $q_2$ satisfying $p$.

$$\cfrac{\cfrac{q_1 \rightarrow q_2 \qquad \cfrac{p \in L(q_2)}{q_2 \vDash p}\ \mathit{atomic}}{q_1 \rightarrow q_2 \wedge q_2 \vDash p}\ \wedge}{\cfrac{\exists q \in Q \text{ s.t. } q_1 \rightarrow q \wedge q \vDash p}{q_1 \vDash \mathbf{EX}\ p}\ \mathbf{EX}}\ \mathit{one-point}$$

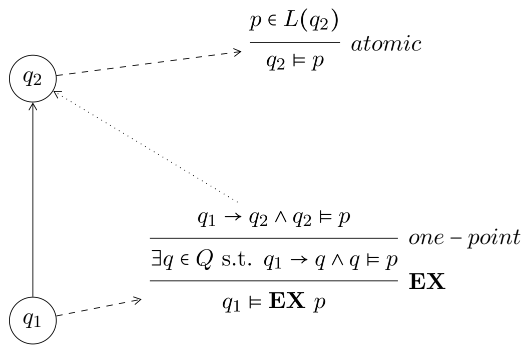
Figure 9.6: A simple proof of why $q_1 \vDash \mathbf{EX}\ p$.

In particular, these proofs—and thus the corresponding proof-like counter-examples—completely explain why a $CTL$ formula is satisfied, that is, branches are provided when needed, and universal operators are explained by expanding all successors. These counter-examples are thus very detailed, as every step of the proof is given.

Gurfinkel and Chechik give an algorithm to generate these proofs. It is similar to proof-based local model checkers [SW91] that incrementally build proofs to check whether a formula is satisfied in a given model, but instead of testing all rules, it uses a global model checker to find applicable ones. For instance, when dealing with the $\vee$ rules $\dfrac{q \vDash \phi_1}{q \vDash \phi_1 \vee \phi_2}\ \vee$ and $\dfrac{q \vDash \phi_2}{q \vDash \phi_1 \vee \phi_2}\ \vee$, a local model checker must try a rule and backtrack if it is not conclusive to try the other. On the other hand, the algorithm proposed by Gurfinkel and Chechik uses a global model checker to determine whether $q \vDash \phi_1$ or $q \vDash \phi_2$, and to apply the corresponding rule, avoiding the need for backtracking.

From these proofs, they build proof-like counter-examples. These counter-examples are parts of the Kripke structure, where states are linked to the proof steps that apply to them. For instance, the proof-like counter-example corresponding to the proof of Figure 9.6 is given in

Figure 9.7. These counter-examples allow the user to get a part of the structure that is responsible for the satisfaction (instead of a single path), and to get more details about why a particular state is part of this explanation thanks to the proof part.



$\dfrac{p \in L(q_2)}{q_2 \vDash p}$ *atomic*
$q_2$
$\dfrac{q_1 \rightarrow q_2 \wedge q_2 \vDash p}{\exists q \in Q \text{ s.t. } q_1 \rightarrow q \wedge q \vDash p}$ *one − point*
$\dfrac{\exists q \in Q \text{ s.t. } q_1 \rightarrow q \wedge q \vDash p}{q_1 \vDash \mathbf{EX}\ p}$ **EX**
$q_1$

Figure 9.7: A proof-like counter-example for $q_1 \vDash \mathbf{EX}\ p$.

To control the aspects of the proof, Gurfinkel and Chechik provide two mechanisms [CG05, CG07]. First, the user can define *visualization strategies* that control the way the counter-example is displayed. For instance, the user can define starting and stopping conditions for the visualization: the part of the counter-example between a state that satisfies the starting conditions and a state satisfying the stopping ones is displayed. These strategies can also control how the states are displayed and what information is retained in the visualization, allowing the user to control the verbosity of the proof annotations. Finally, the resulting counter-example can be examined in a forward fashion—from the state violating the formula to its successors—, or in a backward fashion, from faulty states to their source.

The second mechanism allows the user to control the *generation* of the proof. The user-specified strategies can inspect the current state of the proof, add rules and choose the one that must be used for the next step, and react to rules applications. Thanks to these strategies, the user can, for instance, choose the smallest subgoal—that is, choose the smallest sub-formula for $\phi_1 \vee \phi_2$, if both are satisfied—prefer exploring some pre-defined parts of the model, or impose sequential constraints on the paths embedded in the proof. This mechanism also allows the user to generate the proof interactively, the user choosing the rule to apply next, or the successor to exhibit for explaining **EX** formulas.

Finally, Gurfinkel and Chechik describe KEGVis, a graphical tool for browsing counter-examples. The tool provides several views of the counter-example—a high-level view of the proof, the proof itself, the

Kripke structure part—and implements some pre-defined visualization and generation strategies that the user can directly apply.

### 9.1.3 $CTL$ and Boolean equation systems

Boolean Equation Systems (BES) are sets of equations enriched with fixpoint operators. They are used to perform model checking for $CTL$. More precisely, a BES is defined as a list of *blocks*, that are composed of a *parity indicator* $\mu$ or $\nu$ and a set of *Boolean equations*. A Boolean equation is an identity $X_i = f_i$, where $X_i$ is a Boolean variable and $f_i$ a disjunction or a conjunction of other Boolean variables $X_j$. It is possible to construct, from given $CTL$ formula $\phi$, Kripke structure $S$ and state $q$ of $S$, a BES such that the top-level variable $X$ is true if and only if $S, q \vDash \phi$.

**Diagnostic for Boolean equation systems** Mateescu proposes a way to generate examples and counter-examples for Boolean equation systems (BES) [Mat00]. More precisely, given a BES, we can derive an *extended Boolean graph* (EBG), that is, a graph where vertices are the variables of the BES, the edges denote the (direct) dependencies between variables, and vertices are labelled with $\vee$ or $\wedge$, depending on the operator used in the corresponding equation of the system. Furthermore, an EBG is extended with a *frontier* composed of a subset of the vertices.

Let $M$ be a closed BES—that is, a BES where all variables are the left-hand side of some equation—and $X_i$ a variable. A witness for why $X_i$ is true—called an *example* in [Mat00]—is the part of $G$ effectively showing why $X_i$ is true. More precisely, an example for $X_i$ is a sub-graph $G'$ of $G$ such that $X_i$ is still true in $G'$. Furthermore, as there are several such sub-graphs—including $G$ itself—the paper defines a partial order on such sub-graphs. There are thus minimal $G'$, and the paper describes an algorithm to generate a minimal example for why $X_i$ is true in $G$. Examples and counter-examples are tightly linked, and the paper also describes witnesses for why $X_i$ is false. These counter-examples are also sub-graphs of $G$, on which the same partial order applies. Finally, the paper provides an algorithm to generate counter-examples.

**Evidence-based model checking** Tan and Cleaveland propose the notion of *support set*, a data structure based on Boolean equation systems and storing the reasons for a model checker result [TC02]. The data structure is independent from the actual model-checking technology, and the paper describes how a support set can be extracted from an automata-based model checker for $CTL^*$ [KVW00].

A support set for a given BES is a rooted directed graph where nodes are variables of the BES, with the top-level variable $X$ as the root. Furthermore, any cycle in the graph is such that the shallowest variable—the variable belonging to the first encountered block of the BES—is in a $\nu$-block if the value of $X$ is true, or in a µ-block otherwise. Intuitively, the transition relation $X_i \rightarrow X_j$ reflects the fact that $X_i$ has its value (true or false) because $X_j$ has its value. The graph thus encodes the dependency relation between variables.

Tan and Cleaveland show that the support set extracted from an automata-based $CTL^*$ model checker can be used to extract the part of the system responsible for the property being violated, that is, they show how to extract a counter-example. Indeed, there is a bijection between variables $X_i$ of a support set and state-formula pairs $\langle q, \phi \rangle$, such that $X_i$ is true if and only if $q$ satisfies $\phi$. Thus an edge between $X_i$ and $X_j$ in a support set encodes the fact that some $q_i$ satisfies some $\phi_i$ because some $q_j$ satisfies some $\phi_j$. From this information, it is possible to extract the underlying part of the system explaining why the property is violated or satisfied, depending on the outcome of the model checker.

Tan and Cleaveland also use support sets to *certify* the result of a model checker, by showing how to check the essential conditions that lead to a correct model-checking result. Finally, based on the notion of support sets, Tan propose a generic interface called *PlayGame* on top of the CWB tool for playing diagnostic games in which the user can play a game against the system to understand why a formula is satisfied (or violated) [Tan04].

### 9.1.4 Other solutions

Other authors proposed solutions to explain why a $CTL$ formula is satisfied (or not) by a given state of a system.

**Witness and counter-example automata** Meolic et al. are interested in witnesses and counter-examples for Action-based $CTL$—that they call $ACTL$, not to be confused with the universal fragment of $CTL$—interpreted over labelled transition systems (LTSes) [MFG04].

Labelled transition systems are structures $S = \langle Q, Act, T, q_0 \rangle$ where $Q$ and $Act$ are sets of states and actions, respectively, $T \subseteq Q \times Act \times Q$ is a transition relation labelled with action, and $q_0$ is the initial state. Action-based $CTL$ is a variant of $CTL$ where path operators are annotated with propositional formulas over actions. For instance, the formula $\mathbf{E}[true\ {}_a\mathbf{U}_{(b \vee c)}\ true]$ expresses the fact that there exists a path of actions $a$ ending with an action $b$ or $c$.

Their primary concern is about generating test cases, so they are only interested in a fragment of the logic that accepts finite linear witnesses and counter-examples, that they capture through a particular grammar. Given an LTS $S$, a state $q$ of $S$, and an $ACTL$ formula $\phi$, they propose an algorithm to generate an automaton accepting all finite linear witnesses (or counter-examples) explaining why $q$ satisfies (or not) $\phi$ in $S$. For instance, Figure 9.8 presents a small LTS, and Figure 9.9 shows a witness automaton for the formula $\mathbf{EF}\ \mathbf{EX}_b\ \mathit{true}$. The automaton effectively captures all finite linear witnesses for the formula as such a witness can stay in $wc_0$ (through $a$) for a while, then must go to $wc_1$ or $wc_3$ to show that $b$ can happen once or twice, respectively.

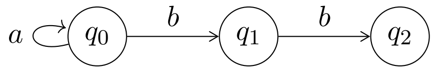

$$\begin{tikzcd} q_0 \arrow[loop left, "a"] \arrow[r, "b"] & q_1 \arrow[r, "b"] & q_2 \end{tikzcd}$$

Figure 9.8: A simple labelled transition system [MFG04].

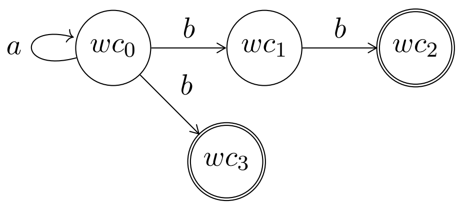

$$\begin{tikzcd} wc_0 \arrow[loop left, "a"] \arrow[r, "b"] \arrow[dr, "b"] & wc_1 \arrow[r, "b"] & wc_2 \\ & wc_3 & \end{tikzcd}$$

Figure 9.9: A witness automaton for $q_0 \vDash \mathbf{EF}\ \mathbf{EX}_b\ \mathit{true}$ [MFG04].

**Game-based counter-examples** Shoham and Grumberg propose a game-based framework for $CTL$ counter-examples and 3-valued abstraction refinement [SG07]. In this framework, a concrete Kripke structure is abstracted into a *Kripke Modal Transition System* (KMTS) with two transition relations: the *may* transition relation links two abstract states $q_1^a$ and $q_2^a$ if there exists two concrete states $q_1$ and $q_2$—behind $q_1^a$ and $q_2^a$, respectively—such that $q_2$ is a successor of $q_1$. The *must* transition relation links $q_1^a$ and $q_2^a$ if, for all concrete state $q_1$ behind $q_1^a$, there exists a successor $q_2$ behind $q_2^a$.

Using these abstract systems, Shoham and Grumberg define a 3-valued semantics for $CTL$ in which a $CTL$ formula is true, false or indefinite in a given abstract state. The idea behind this semantics is that, if the given $CTL$ formula is true (resp. false) in some abstract state $q^a$, then it is true (resp. false) in all states behind $q^a$, otherwise, the formula is indefinite in $q^a$ and nothing can be concluded about the concrete states.

To evaluate a given $CTL$ formula over a given KMTS, Shoham and Grumberg propose a game-based coloring algorithm. The idea of the algorithm is to derive, from the KMTS $S$ and the formula $\phi$, a game between two players Abelard and Eloise. Abelard tries to refute the formula while Eloise tries to satisfy it. The game is composed of pairs of states of $S$ and sub-formulas of $\phi$. At each step, the player that chooses the next step is based on the formula of the current node: if the formula is an existential one—such as **EX** or $\vee$—Eloise chooses the next step, while if the formula is a universal one—**AX** and $\wedge$ nodes—Abelard makes the choice. The formula $\phi$ is true in the initial node of the game if Eloise has a strategy to win the game, $\phi$ is false if Abelard has a winning strategy, and the formula is indefinite otherwise.

The proposed algorithm determines the nodes of the game in which each player has a winning strategy, and the ones in which none of them can win. It colors each node, starting from the leaves of the game, and determines the color of a node—$T$ for true, $F$ for false, and ? for indefinite—based on its successors and its formula. In this context, Shoham and Grumberg define *annotated counter-examples* as the sub-part of the game graph that is sufficient to show why the root node has the $F$ color. For instance, Figure 9.10 shows a (concrete) KMTS, and Figure 9.11 the game graph for the formula $\mathbf{A}[p \;\mathbf{V}\; q]$—equivalent to $\neg\mathbf{E}[\neg p \;\mathbf{U}\; \neg q]$—evaluated on $q_0$, where gray nodes are colored with $F$ and white nodes with $T$. As the KMTS is concrete and not abstract, the game graph contains no node colored with ?. Figure 9.12 shows the corresponding counter-example.

The algorithm and the annotated counter-example it generates are tightly coupled to the coloring algorithm. To allow the generating algorithm to build the counter-example, the coloring algorithm needs to remember, for each node, the reason for its color—for instance, the responsible successor. From these causes, the generating algorithm can determine which part of the game graph must be exhibited. The drawback of this approach is that the coloring algorithm determines *one* cause of the coloring instead of *all of them*, resulting in the generation of one predetermined counter-example. Generating another counter-example would imply recoloring the game graph and making other choices when determining the causes of coloring.

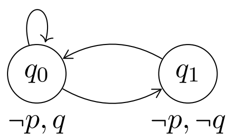
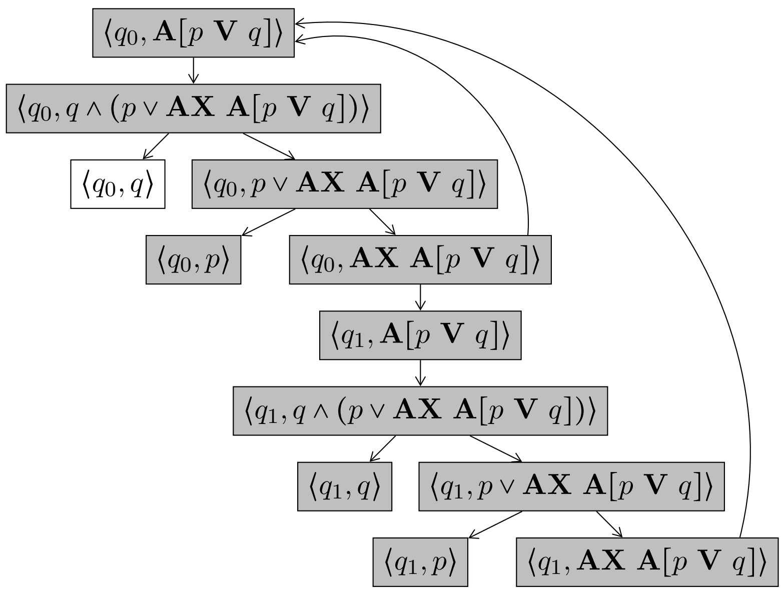


Figure 9.10: A simple concrete KMTS [SG07].



$\langle q_0, \mathbf{A}[p\ \mathbf{V}\ q]\rangle$
$\langle q_0, q \wedge (p \vee \mathbf{AX}\ \mathbf{A}[p\ \mathbf{V}\ q])\rangle$
$\langle q_0, q\rangle$
$\langle q_0, p \vee \mathbf{AX}\ \mathbf{A}[p\ \mathbf{V}\ q]\rangle$
$\langle q_0, p\rangle$
$\langle q_0, \mathbf{AX}\ \mathbf{A}[p\ \mathbf{V}\ q]\rangle$
$\langle q_1, \mathbf{A}[p\ \mathbf{V}\ q]\rangle$
$\langle q_1, q \wedge (p \vee \mathbf{AX}\ \mathbf{A}[p\ \mathbf{V}\ q])\rangle$
$\langle q_1, q\rangle$
$\langle q_1, p \vee \mathbf{AX}\ \mathbf{A}[p\ \mathbf{V}\ q]\rangle$
$\langle q_1, p\rangle$
$\langle q_1, \mathbf{AX}\ \mathbf{A}[p\ \mathbf{V}\ q]\rangle$

Figure 9.11: The colored game graph for $q_0 \models \mathbf{A}[p\ \mathbf{V}\ q]$. Gray nodes are colored with $F$ and the white one with $T$ [SG07].

## 9.2 Explanations for multi-modal logics

Some authors adapted the solutions proposed for bare $CTL$ to support extensions of the logic, including epistemic-temporal logics such as $CTLK$.

**MCMAS** MCMAS is an open-source model checker for multi-agent systems [LR06a, LQR09, LQR15]. Its main functionality is the BDD-based verification of $CTL$, $ATL$ and epistemic properties over multi-agent systems that are described in the framework of interpreted systems [FHMV95]. This framework describes each agent separately, defining her local states, her protocol—that is, which actions she can play in each local state—, and her local evolution. The whole system is the

$$\begin{tikzcd}
\boxed{\langle q_0, \mathbf{A}[p \mathbf{~V~} q]\rangle} \arrow[d] & & \\
\boxed{\langle q_0, q \wedge (p \vee \mathbf{AX~A}[p \mathbf{~V~} q])\rangle} \arrow[dr] & & \\
& \boxed{\langle q_0, p \vee \mathbf{AX~A}[p \mathbf{~V~} q]\rangle} \arrow[dl] \arrow[dr] & \\
\boxed{\langle q_0, p\rangle} & & \boxed{\langle q_0, \mathbf{AX~A}[p \mathbf{~V~} q]\rangle} \arrow[d] \\
& & \boxed{\langle q_1, \mathbf{A}[p \mathbf{~V~} q]\rangle} \arrow[d] \\
& & \boxed{\langle q_1, q \wedge (p \vee \mathbf{AX~A}[p \mathbf{~V~} q])\rangle} \arrow[dl] \\
& \boxed{\langle q_1, q\rangle} &
\end{tikzcd}$$

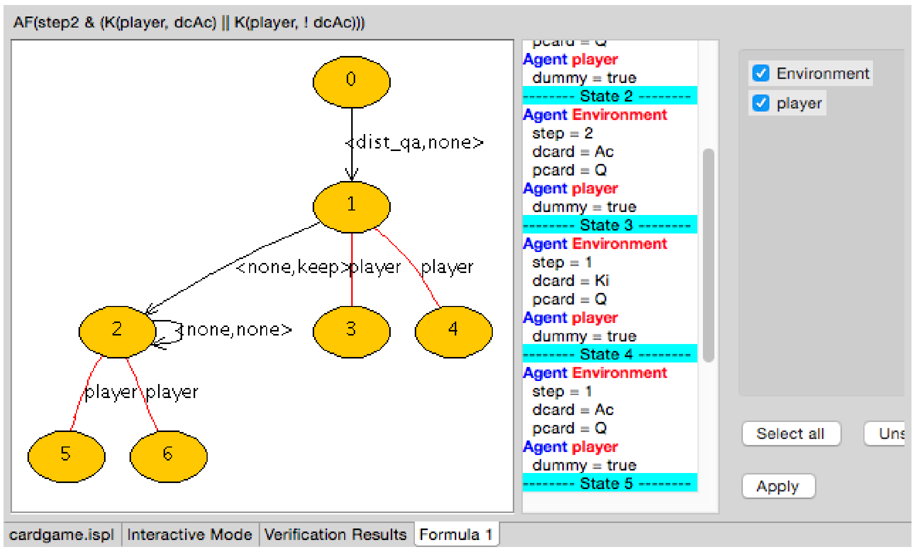
Figure 9.12: An annotated counter-example for $q_0 \vDash \mathbf{A}[p \mathbf{~V~} q]$ [SG07].

synchronous composition of all the agents.

In addition to the command-line functionalities, MCMAS proposes a graphical interface based on an Eclipse plug-in. Through this interface, the user can define new interpreted systems, simulate them, and verify associated formulas. Furthermore, the tool can generate witnesses for existential formulas and counter-examples for universal ones. These witnesses and counter-examples are tree-like ones, as proposed by Clarke et al. [CJLV02], adapted for the case of epistemic and $ATL$ operators. More precisely, the tool can generate counter-examples for universal operators—including the knowledge ones—and witnesses for existential operators, but does no generate the counter-parts. Nevertheless, it generates as much of the explanation as possible. For instance, if the formula **AG EF** *init* is violated by an interpreted system, MCMAS will provide a trace to a state violating **EF** *init*, but will not explain why this state satisfies **AG** $\neg$*init*.

The explanations are displayed in a dedicated window, laid out thanks to the Graphviz tool [GN00], and state information are displayed next to the graph representing the counter-example. Figure 9.13 presents a snapshot of the counter-example window of MCMAS. State information about particular agents can be selectively hidden or shown, by unchecking or checking the corresponding box in the right panel.

**MCK** MCK is a model checker for temporal-epistemic logics over multi-agent systems [GvdM04]. The tool supports a large panel of logics, such


AF(step2 & (K(player, dcAc) || K(player, ! dcAc)))

```
pcard = Q
Agent player
 dummy = true
-------- State 2 --------
Agent Environment
 step = 2
 dcard = Ac
 pcard = Q
Agent player
 dummy = true
-------- State 3 --------
Agent Environment
 step = 1
 dcard = Ki
 pcard = Q
Agent player
 dummy = true
-------- State 4 --------
Agent Environment
 step = 1
 dcard = Ac
 pcard = Q
Agent player
 dummy = true
-------- State 5 --------
```

Figure 9.13: A snapshot of the counter-example window of MCMAS, with the graph on the left and the details of the states on the right.

as $CTL$, $CTL^*$, knowledge and common knowledge operators, as well as more restricted variants such as $\mathbf{X}^n$ operators reasoning about facts that become true after a fixed number of $n$ steps of the model. Furthermore, it supports several semantics for the epistemic operators, such as the *observational* one—the agents base their knowledge on what they can observe from the current state—, the *clock* semantics—based on the current state and the number of steps from the initial state—, and the *perfect recall* one—based on the whole history of observations.

The tool implements several algorithms to tackle the model-checking problems, using either BDDs, explicit traversal or bounded model-checking techniques to verify whether a given formula is satisfied by a given model. Nevertheless, all combinations are not available. For instance, it is not possible to check $CTLK$ formulas under the *perfect recall* semantics.

MCK proposes several debugging facilities. The first one is the capability to export the whole finite state machine of the model in a Graphviz-compliant format. This functionality is however only useful for very small models. When dealing with bounded model-checking algorithms, the tool can also export the trace or the tree-like counter-example showing why the given formula is violated, depending on the checked formula.

The last proposed debugging facility is the capability to play a *game* against the model checker when a formula is violated [HvdM09]. This game is played between a *verifier*, that tries to show that the formula is satisfied is some state, and a *refuter* trying to show it is violated. Initially, the game starts with an initial state and the violated formula, the model checker being the refuter and the user being the verifier. Depending on the current sub-formula, either one player or the other must make a step. For instance, if the formula is $\phi_1 \wedge \phi_2$, then the refuter must choose one of the two sub-formulas, and the game goes on with the chosen sub-formula. If the formula is $\mathbf{EX}\ \phi$, then the verifier must choose a successor of the current state, and the game goes on with the new state and $\phi$.

The idea of this game is to give some insight to the user into why the formula is violated. For instance, when showing that the formula $\mathbf{E}[\phi_1\ \mathbf{U}\ \phi_2]$ is violated, the user must exhibit a path of states satisfying $\phi_1$ and reaching a state satisfying $\phi_2$. Since the formula is not satisfied by the current state, the user cannot exhibit such a path. The idea is to let the player explore the parts of the model she thinks contain a witness path, and let her understand why there is no such path. The game is incremental, and the tool allows the user to backtrack to previous game steps, allowing her to make other choices and get more insight into the model.

**Counter-examples for temporal description logic** Weitl et al. propose structured explanations—they call *evidences*—for the temporal description logic $ALCCTL$, a combination of the description logic $ALC$ with $CTL$ [WNF10]. Within this logic, it is possible to express facts about a graph of webpages where edges represent hyperlinks, such as $\mathbf{AG}\ (defined \sqsubseteq explained \sqcap \mathbf{EX}\ examplified)$, meaning that *it holds globally that the set of elements defined on a page is a subset of the set of elements that are explained on the same page and exemplified in another hyperlinked page.*

The evidences are tree-like structures where nodes are annotated with sequences of formulas, and where children of one node explain why the formulas of the parent node are satisfied or violated. For instance, Figure 9.14 presents a (incomplete) evidence for the formula

$$\phi = \mathbf{E}[(Task \sqsubseteq \mathbf{EX}\ Solution)\ \mathbf{U}\ \neg(Test \sqsubseteq \bot)]$$

satisfied in some state $q_0$ of some model, expressing the fact that *there exists a path of hyperlinks along which every task has a solution in some directly linked page, to a page with a test.*

Weitl et al. also propose an algorithm to generate evidences. It builds the evidence top-down, by using a model checker to decide which

$$\begin{tikzcd}
 & \boxed{q_0 \vDash \mathbf{E}[(Task \sqsubseteq \mathbf{EX}\ Solution)\ \mathbf{U}\ \neg(Test \sqsubseteq \bot)]} \arrow[d] & & \\
 & \boxed{q_0 \vDash Task \sqsubseteq \mathbf{EX}\ Solution, q_1 \vDash Task \sqsubseteq \mathbf{EX}\ Solution, q_2 \vDash \neg(Test \sqsubseteq \bot)} \arrow[dl] \arrow[d] \arrow[dr] & & \\
\cdots & \cdots & \boxed{q_2 \vDash \neg(Test \sqsubseteq \bot)} \arrow[d] & \\
 & & \boxed{q_2 \vDash Test(tree) \wedge \neg\bot(tree)} \arrow[d] & \\
 & & \boxed{q_2 \vDash Test(tree), q_2 \vDash \neg\bot(tree)} \arrow[d] \arrow[dr] & \\
 & & \boxed{\top} & \boxed{q_2 \nvDash \bot(tree)} \arrow[d] \\
 & & & \boxed{\top}
\end{tikzcd}$$

Figure 9.14: An evidence for $q_0 \vDash \phi$ [WNF10].

successors are true, and which nodes must be added. In particular, it generates all evidences for some parts of the graph, instead of one: when dealing with *concept formulas*—that speaks about the concepts such as $Task$ and *Solution*—, the algorithm generates all elements for which there exists a task in the current page, instead of just showing one example.

For the cases of **AX** and **EX** operators, it exposes all successors, limited to the ones satisfying the sub-formula for **EX**. Nevertheless, it generates only one (shortest) path for the witnesses of existential temporal operators (such as **EU**) and the counterexamples of universal temporal ones (such as **AF**). Furthermore, witnesses of universal temporal operators (such as **AF**) and counter-examples of existential ones (such as **EU**) are not given at all; a simple $\top$ evidence is returned, giving no additional information to the user.

Weitl and Nakajima further extend the algorithm in [WN10] to include incremental and interactive generation. Instead of giving all evidences for the parts it gives all evidences, the user is asked to choose one of them. Furthermore, the evidence is lazily generated, one node at a time, and the user is asked about which node must be expanded. This allows the approach to scale better when dealing with large models, and to allow the user to explore the parts of the evidence she is interested in.

## 9.3 Tree-like annotated counter-examples for temporal-epistemic logic

In [BP12], we present tree-like annotated counter-examples for the logic *CTLK* (TLACEs in short). These counter-examples have a tree-like structure, that is, there are composed of a hierarchy of single paths. Each path—or *branch*—is annotated with the sub-formula that it explains. These counter-examples are defined in the framework of $ARCTL$, an extension of $CTL$ that restricts path quantifiers [PR06], but they are targeted to $CTLK$ through a translation of its model-checking problem [LPR07].

For instance, let us consider a variant of the simple card game. In this variant, the game is played with four cards—$A$, $K$, $Q$ and $J$—such that $A$ wins over $K$ and $Q$, $K$ over $Q$ and $J$, $Q$ over $J$ and $J$ over $A$. Furthermore, the dealer and the player never see their opponent's card, nor the cards on table, but they both know who is the winner at the end. In this game, we can ask whether *the player always finally knows whether the dealer has the* $A$ *or not*, expressed as

$$\phi = \mathbf{AF}\ (dc \neq \_ \wedge (\mathbf{K}_{player}\ dc = A \vee \mathbf{K}_{player}\ dc \neq A)),$$

where $dc = C$ is true in states in which the dealer has card $C$.

The formula is violated, and TLACE for explaining why is given in Figure 9.15. It corresponds to a scenario where the dealer receives the $A$ and the player the $Q$, and the player chooses to keep his card. The states contain the values of the cards of both agents—$C_1, C_2$ means that the player has $C_1$ and the dealer $C_2$. The wavy transitions link together states that are undistinguishable by the player and arrowed transitions are temporal ones. States are annotated with the formulas they satisfy and transitions are annotated with formulas they explain.

The main branch in bold explains how the agents receive their cards and how the player keeps his. For the two last states of this main path, two states that are undistinguishable by the player from the main state are given to show that he does not know that the dealer has the $A$ (right state) nor that he has not (left state). Furthermore, dashed states show that each of these states are reachable from the initial one.

A TLACE is an adequate explanation for explaining why an $ARCTL$ formula $\phi$ is satisfied by a system $S$ in the sense that it matches $S$ and explains $\phi$. A TLACE $n$ matches $S$ if the states and paths of $n$ are states and paths of $S$. Furthermore, $n$ explains $\phi$ if $n$ correctly explains each sub-formula of $\phi$. In practice, these counter-examples integrate the complete information to explain why the existential fragment of the logic is satisfied, but they do not provide explanations for universal operators.

The paper also presents an algorithm to generate adequate TLACEs and an implementation of this algorithm in NuSMV. This implementation is complemented with a tool to display and manipulate TLACEs. This tool is an interactive graphical interface application for displaying and browsing tree-like annotated counter-examples. The counter-examples

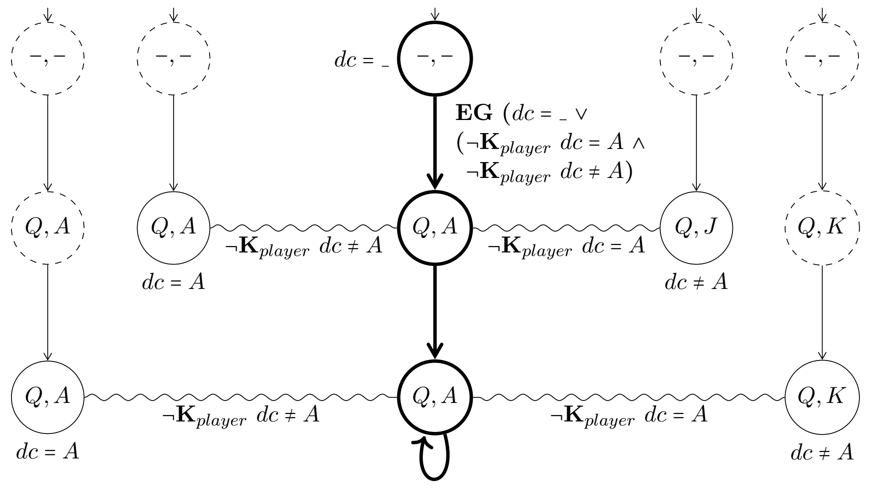

$-,-$
$-,-$
$dc = \_$
$-,-$
$-,-$
$-,-$
$\mathbf{EG}\ (dc = \_ \vee$
$(\neg\mathbf{K}_{player}\ dc = A \wedge$
$\neg\mathbf{K}_{player}\ dc \neq A)$
$Q, A$
$Q, A$
$\neg\mathbf{K}_{player}\ dc \neq A$
$Q, A$
$\neg\mathbf{K}_{player}\ dc = A$
$Q, J$
$Q, K$
$dc = A$
$dc \neq A$
$Q, A$
$\neg\mathbf{K}_{player}\ dc \neq A$
$Q, A$
$\neg\mathbf{K}_{player}\ dc = A$
$Q, K$
$dc = A$
$dc \neq A$

Figure 9.15: A tree-like annotated counter-example for the formula $\mathbf{AF}\ (dc \neq \_ \wedge (\mathbf{K}_{player}\ dc = A \vee \mathbf{K}_{player}\ dc \neq A))$.

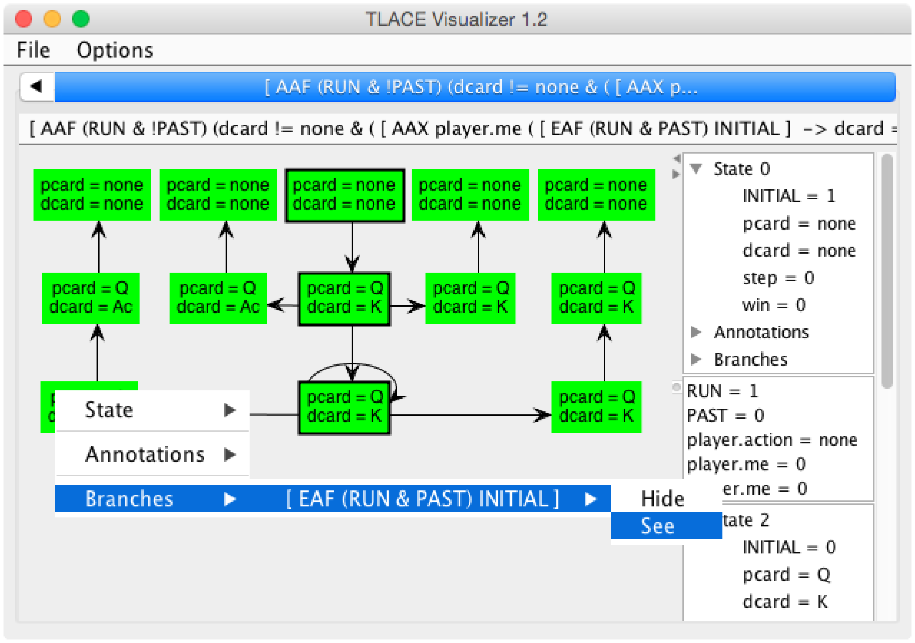
are loaded from XML files produced by the modified NuSMV and pictured as a graph in the main area of the interface. The tool also provides different means to arrange the layout of the graph and explore the detailed information associated to each node. A snapshot of the interface is given in Figure 9.16.

The tool automatically lays out the counter-example upon loading, according to a custom layout algorithm that takes into account the semantic structure of the counter-example. This representation presents the general structure of the TLACE, showing branches and loops. Single states or entire subtrees can be dragged around for better readability. To support browsing of larger graphs, branches can be folded and unfolded to reduce clutter and selectively show relevant information.

A side panel displays the values of all variables and annotations along a selected path in the graph, in a collapsible hierarchical presentation. All variable values can also be accessed as a pop-up menu on each node in the main panel, and variables can be selected for display as part of the node label, giving immediate visibility for a few variables of interest.

The main advantage of TLACEs is the fact that they give the complete information to understand why an existential *ARCTL* formula is satisfied. Furthermore, the visualization tool can help the user to understand complex explanations thanks to its functionalities to move parts of the graph and to inspect particular paths.

Nevertheless, tree-like annotated counter-examples can be very large: the number of nodes of a TLACE is exponential in the size of the checked



[ AAF (RUN & !PAST) (dcard != none & ( [ AAX player.me ( [ EAF (RUN & PAST) INITIAL ] -> dcard =

State 0
INITIAL = 1
pcard = none
dcard = none
step = 0
win = 0
Annotations
Branches

RUN = 1
PAST = 0
player.action = none
player.me = 0
er.me = 0

tate 2
INITIAL = 0
pcard = Q
dcard = K

Figure 9.16: A snapshot of the interface of the graphical tool illustrating its different features.

formula. Furthermore, they also lack some information as they do not explain why a universal **A** operator is satisfied. The explanation for such a formula is not a single (potentially lasso-shaped) path of the system, so TLACEs are not an adequate structure to explain them.

## 9.4 Explanations for the µ-calculus

The µ-calculus and *CTL* share the same problems with counter-examples because they have the same branching characteristics. Several authors proposed solutions to solve these problems for the µ-calculus, starting from the (unpublished) work of Kick [Kic95b, Kic95a].

### 9.4.1 Explanation graphs for µ-calculus

In two technical reports, Kick proposes witnesses for global µ-calculus model checking. First, he defines witnesses as parts of the checked Kripke structure $S$ with states labelled with sets of sub-formulas of the checked formula $\phi$ [Kic95b]. Furthermore, he proposes an algorithm to generate such witnesses. The algorithm uses information generated during the execution of a modified model-checking algorithm that keeps track of the successive iterations used in least fixpoint evaluations.

Second, Kick refines these witnesses by separating in different nodes the labels of the states [Kic95a]. In effect, these new witnesses are derived from tableaux for the µ-calculus: a witness is a part of a proof tree based on tableau rules, where isomorphic sub-trees are merged. They can be viewed as graphs where each node is annotated with $q \vDash \phi$, with successors showing why $q$ effectively satisfies $\phi$. Kick also describes an algorithm that uses the same modified model-checking algorithm.

### 9.4.2 Using games to explain formula violation

Stirling et al. proposed a proof system for the µ-calculus over states of labelled transition systems (LTS) [SW91]. From this proof system, Stirling derives a game between two players such that one player has a strategy for winning the game if and only if the formula is satisfied [Sti95].

This work leads to the definition of a debugging game in which the user plays against the computer to understand why a given µ-calculus formula is satisfied or violated [SS98]. The idea of the game is to show to the user why what she thinks might be true is false, or vice versa. These games can give more information than standard counter-examples (and witnesses) because they allow the user to explore the parts of the model she thinks should be a counter-example (or a witness), and allow her to understand why it is not the case. She can thus explore and expose all witnesses or counter-examples she wants.

For instance, if a state $q$ of some LTS violates the µ-calculus formula $\phi = \mu v.\ p \vee \Diamond\ v$—meaning that there exists a path from $q$ eventually reaching a state satisfying $p$—, then the user can play a game with the system in which she will try to exhibit such a path, without succeeding (as the formula is not satisfied).

Lange and Stirling later defined model-checking games for $CTL^*$ and some of its fragments such as $CTL$ and $LTL$, using the same ideas [LS02]. These games have inspired the work of Tan [Tan04] presented in Section 9.1.3, and the work of Huang and van der Meyden [HvdM09] presented in Section 9.2.

### 9.4.3 Boolean equation systems and µ-calculus

Boolean equation systems are also used to perform model checking for µ-calculus formulas.

**A relational graph algebra for evidence exploration** Dong et al. propose a framework and a tool to explore what they call *evidences* in the case of model checking µ-calculus formulas [DRS03a, DRS03b]. An evidence is a graph with nodes composed of state-formula pairs that shows why the state of the root of the graph satisfies its formula in a given model. The problem they try to solve is to provide the user with tools to explore and understand evidences, that can become very large and complex when dealing with large models and formulas.

The size of an evidence is $O(|S| \times |\phi|)$, thus can be very large for large systems. To overcome this complexity, Dong et al. propose a *relational graph algebra*. A *relational graph* is a graph where nodes and edges are labelled with values from some domains $D_V$ and $D_E$, respectively. A domain is either a primitive domain such as integers, or the Cartesian product of other domains $D_1, ..., D_n$. Furthermore, we assume that any domain contains the special *null value* $\epsilon$. An element of a domain $D$ is thus a tuple of values from the domains composing $D$.

An evidence can be viewed as a relational graph with the vertex domain $Q \times \mathcal{L}_\mu$, where $\mathcal{L}_\mu$ is the set of µ-calculus formulas, and with the empty domain for edges.

Similarly to standard relational algebra, Dong et al. propose a set of basic operators to manipulate relational graphs. Given two graphs $G_1$, $G_2$ with nodes and edges on domains $D_{V,i}$, $D_{E,i}$ for $i = 1, 2$,

- $G_1 \cup G_2$ is the union of the graphs, containing nodes and edges of both graphs, assuming that they are defined on the same nodes and edges domains;

- $G_1 \cap G_2$ is the intersection of the graphs, containing nodes and edges present in both graphs, assuming that they are defined on the same nodes and edges domains;

- $G_1 -_V G_2$ is the node-difference of $G_1$ and $G_2$, containing nodes of $G_1$ not in $G_2$;

- $G_1 -_E G_2$ is the edge-difference of $G_1$ and $G_2$, containing nodes of $G_1$, and edges of $G_1$ not in $G_2$;

- given two Boolean functions $f_v$ and $f_e$ over nodes and edges domains of $G_1$, $\sigma_{f_v,f_e}(G_1)$ is the *selection of nodes and edges satisfying $f_v$ and $f_e$, respectively*, that is, the graph composed of nodes $v$ and edges $e$ of $G_1$ such that $f_v(v)$ and $f_e(e)$ are true, respectively;

- given two sub-domains $d_v$ and $d_e$ of $D_{V,1}$ and $D_{E,1}$, respectively, $\pi_{d_v,d_e}(G_1)$ is the *projection of nodes and edges onto sub-domains $d_v$*

*and $d_e$ respectively*, that is, the graph composed of the projection of nodes and edges of $G_1$ on sub-domains $d_v$ and $d_e$, respectively;

- $G_1 \times G_2$ is the graph composed of cross-product of nodes and edges of $G_1$ and $G_2$.

Other operators can be derived from the basic ones, such as the *natural join* of two graphs, or the *extension* and *grouping* of nodes and edges of a graph. We will reuse this algebra in Chapter 10 to manipulate explanations.

Using this relational graph algebra, Dong et al. designed and implemented the Evidence Explorer, a tool to visualize and navigate through an evidence thanks to smaller views [DRS03b]. The tool proposes several views of the same evidence, synchronized on a so-called *focus node*, such as the *formula window* displaying the top-level formula in a graph-based fashion, the *overview window* displaying the whole evidence as a spanning tree, and the *path window* displaying a path from the state of the root node to the state of the focus node. Using these windows, the user can inspect the evidence and update the focus node—within any window—to display information about other parts of the evidence.

**Diagnostics for Model Checking** Linssen proposes the notion of *generic diagnostic graph* for µ-calculus model checking, a graph where vertices are labelled with pairs of state and formula, and where edges follow a well-defined dependency relation, based on the states and formulas labelling the extremities [Lin11]. For instance, a vertex labelled with $\langle q_0, \Diamond\ \phi\rangle$ must have, as only successor, a vertex labelled with $\langle q_1, \phi\rangle$, where $q_1$ is a successor of $q_0$ in the checked model. Linssen then shows how these generic diagnostic graphs are related to different model-checking frameworks such as parity game-based model checking, and Boolean Equation System-based model checking.

Linssen applies and extends the notion of generic diagnostic graph to BES, and shows how to generate them by extending the BES with state-formula annotations on BES variables. Furthermore, he provides algorithms to play an interactive game against the system based on the annotated BES solution to understand why a formula is true (or false) in some state, and uses this algorithm to generate simple paths and reduced systems, that is, the part of the model responsible for the model-checking outcome.

**Proof graphs for parameterised Boolean Equation Systems and fixpoint logic** Cranen et al. extend the notion of support set for *parameterised Boolean equation systems* (PBES) [CLW13]. These PBES extend Boolean equation systems by introducing data types to variables of the system. This allows a user to define infinite state space systems, and to reason about such systems in a more compact way. Cranen et al. define the notion of *proof graph*, an extension of support sets where the vertices of the underlying graph are not variables anymore, but so-called *instantiations*, that is, variables coupled with a particular data element from the corresponding data type.

Cranen et al. show that there exists a proof graph explaining that some instantiation satisfies a given PBES if and only if this instantiation effectively satisfies the PBES. Furthermore, they discuss about minimality of proof graphs and show that minimizing a proof graph—that is, finding the smallest sub-graph that still fully explains the outcome—is an NP-hard problem. Finally, they apply the idea of proof graphs to the verification of strong bisimilarity of two systems, and show that it is possible to extract, from the proof graph showing that the two systems are not bisimilar, a distinguishing formula, that is, a formula that is satisfied by the first system but not by the second.

Cranen et al. further extend their notion of proof graphs for the case of least fixpoint logic ($LFP$) [CLW15]. $LFP$ is an extension of first-order logic with least fixpoints. $LFP$ formulas are based on first-order variables, function and relation symbols (over a given domain of discourse $A$), first-order logic operators, and the least fixpoint operator $\mu$ on second-order variables. For instance, the following formula (using the notations of Cranen et al.) expresses the fact that only finitely many consecutive $a$-transitions can be taken from the state $s_1$:

$$\phi = [\mathbf{lfp} X s. \forall s', s \xrightarrow{a} s' \implies X s'] s_1$$

$X$ is a second-order variable ranging over relations.

In this context, a *proof graph* is a graph where nodes are triplets composed of (1) a truth value, (2) a second-order variable or a function symbol, and (3) a sequence of elements of the domain $A$. A node $\langle true, X, s\rangle$ means that $s \in X$, and the descendants of the node explain why this is the case. Thanks to the introduction of the truth value, the framework can handle negations, as a node $\langle false, X, s\rangle$ says that $s \notin X$. Proof graphs are restricted through several dependency conditions. In particular, any cycle in the proof graph must be about a greatest fixpoint if the truth value within this cycle is true, or about a least fixpoint otherwise. These conditions ensure that there exists a proof graph for $q \models \phi$ if and only if $q$ effectively satisfies $\phi$.

From these proof graphs, Cranen et al. explain how to extract what they call an *evidence*, that is, the part of the system responsible for the validity or invalidity of the formula. While the proof graph still retains information about variables, relation and function symbols of the formula, as well as elements of the domains linked to them, the evidence is expressed in the language of the model under verification. They finally apply the idea of evidences to generate counter-examples for stuttering bisimulation checking, and for providing linear and tree-like counter-examples for $LTL$ and $ACTL^*$ formulas.

### 9.4.4 Model-checking certificates

Namjoshi proposes a deductive proof system to certify the result of a model checker in the framework of automata-based verification of the µ-calculus [Nam01]. The idea behind the proof system is to be able to check the result given by a model checker independently from the model checker itself. In the considered automata-based framework, a µ-calculus formula is translated into an alternating automaton on which a parity game is played between two players to decide whether the given Kripke structure satisfies the given formula or not. If the first player has a winning strategy, then the formula is satisfied, otherwise the formula is violated.

The proposed deductive proof system is composed of predicates and partial rank functions for each state of the alternating automaton satisfying some predefined rules to effectively explain the satisfaction of the underlying µ-calculus formula.

Namjoshi then explains how to generate deductive proofs from the automaton used for verification, and discusses how such a proof is related to counter-examples: a proof embeds all possible counter-examples, that is, all reasons for why the formula is satisfied by the given model.

Similarly to Namjoshi, Hofmann and Rueß provide *certificates* for µ-calculus formulas—that is, proofs of validity that can be checked independently from the model checker result—by providing winning strategies for parity games [HR14]. *Parity games* are two-player games with ranks over states, in which player 0 (resp. 1) wins a play if and only if the parity of the highest rank met infinitely often during the play is even (resp. odd). The players of such games have positional strategies—that is, memoryless strategies—, and µ-calculus model checking can be reduced to solving a derived parity game. Hofmann and Rueß propose an algorithm to generate a positional strategy to show why player 0 can win the game.

## 9.5 Summary

Even if they apply to different frameworks—such as game-based, BDD-based, or BES-based model checking—or logics—such as $CTL$, epistemic logics, or the µ-calculus—, the ideas presented in this chapter share some common ideas.

They represent an explanation as a directed graph. This graph can be a part of the system, such as the tree-like counter-examples of Clarke et al. But they can be more detailed, including the sub-formulas of the formula of interest, in the cases, for instance, of TLACEs, Kick's witnesses, or proof-like counter-examples.

As they can be complex, some ideas have been proposed to project them on the system, such as the evidences of Cranen et al. Other ideas to deal with this complexity include interactive and incremental generation, with the framework of Weitl et al., and the generation and visualization strategies of Gurfinkel and Chechik.

Another solution is the debugging game of Stirling, in which the user tries to show that some formula is true in a system that violates it. This idea has been later adapted by Huang and van der Meyden, and by Tan. Finally, the relational graph algebra of Dong et al. can also be used to reduce the richness by providing different views of the explanation.

# Chapter 10

# A framework for $\mu$-calculus based logic explanations

This chapter presents a framework for generating, manipulating and visualizing explanations for logics that can be translated into the µ-calculus. More precisely, let us suppose that someone—called the *designer* in the sequel—defines a new logic—called the *top-level logic* in the sequel—to express and verify new facts about some system, and wants to develop a model checker for it. She can either develop the tool from scratch, or she can translate the models and formulas of the logic into another logic models and formulas—the *base logic*—and use existing tools to solve the model-checking problem.

Many logics can be translated into the µ-calculus, making it a good candidate for a base logic. Nevertheless, when translating her model-checking problem into a µ-calculus one, the designer has no help to facilitate this translation, in particular, the counter-examples returned by the model checker (if any) are expressed in terms of µ-calculus primitives instead of top-level logic ones. To overcome this limitation and to help designers to quickly develop a model checker with rich counter-examples, this chapter proposes a µ-calculus based framework with rich explanations. The framework provides

- a BDD-based µ-calculus model checker with generation of rich explanations,
- functionalities to define how top-level logic formulas are translated into µ-calculus formulas,

- functionalities to control how the explanations for the µ-calculus are generated,

- functionalities to translate µ-calculus explanations into top-level logic explanations.

To illustrate the presented concepts, this chapter uses the case of $ATL$ model checking. Given a CGS $S$, a state $q$ of $S$, and an $ATL$ formula $\phi$, it is possible to translate $S$ into a Kripke structure $S'$, $q$ into a state $q'$ of $S'$, and $\phi$ into a µ-calculus formula $\phi'$ such that $S, q \vDash \phi$ if and only if $S', q' \vDash \phi'$. To avoid technical details, this introduction only presents the intuition of the translation, and focuses on one $ATL$ operator only. The full translation and the application of the framework to the full $ATL$ logic are differed to Section 10.4.

The idea of the translation from a CGS $S = \langle Ag, Q, Q_0, Act, e, \delta, V\rangle$ to a µ-calculus Kripke structure $S' = \langle Q', \{R'_i \mid i \in \Sigma\}, V'\rangle$ is to derive, from each state $q \in Q$, each group of agents $\Gamma \subseteq Ag$, and each joint action $a_\Gamma$ of $\Gamma$, a new state $q_{a_\Gamma}$ representing the fact that $\Gamma$ chose to play $a_\Gamma$ in $q$. In the sequel, we write $original(q_{a_\Gamma})$ for the original state $q$ from which $q_{a_\Gamma}$ is derived. For each group $\Gamma$, two transition relations are derived from $\delta$: $R_{\Gamma choose}$ links any state $q \in Q$ to the derived states $q_{a_\Gamma}$ for all possible actions of $\Gamma$; $R_{\Gamma follow}$ links any derived state $q_{a_\Gamma}$ to the original successors of $q$ through $a_\Gamma$, that is, the successors of $q$ restricted to the ones reached if $\Gamma$ choose $a_\Gamma$. Intuitively, the derived Kripke structure $S'$ encodes in two steps ($q \to q_{a_\Gamma} \to q'$) the one-step transitions of $S$ ($q \xrightarrow{a} q'$).

For instance, Figure 10.1 presents a CGS for a simple one-bit transmission problem in which a *sender* tries to send a value through a unreliable *link*. The sender can *send* the value or *wait*, and the link can *transmit* the message (if any), or *block* the transmission. In this context, we ask, for instance, whether the transmitter has a strategy to never transmit the value.

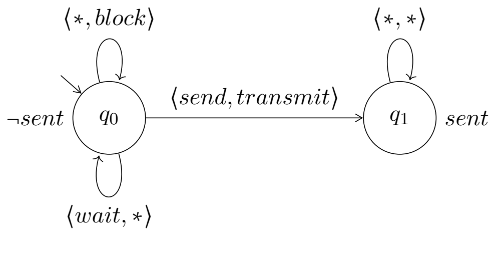


Figure 10.1: The CGS of the bit transmission problem. The action couples are the action of the sender, and the transmitter, respectively. $*$ means *any action of the agent.*

The CGS of the bit transmission problem can be translated into a µ-calculus Kripke structure. Figure 10.2 presents a part of the translation, focusing on the states derived from $q_0$; the part about $q_1$ is not shown.

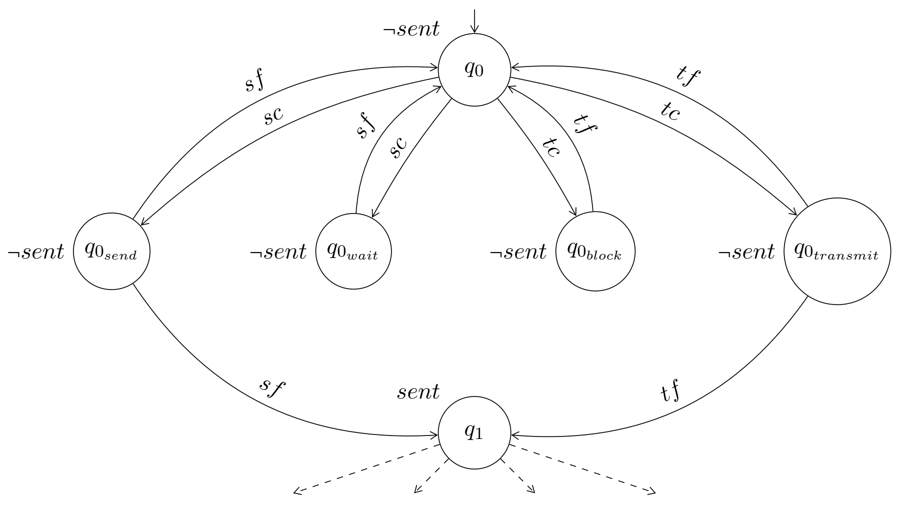


Figure 10.2: A part of the Kripke structure corresponding to the CGS of the bit transmission problem. Edges are labelled with (a shortcut of) the name of the transition relation they belong to: *sc* means *sender chooses*, *sf* means *sender follows*, *tc* and *tf* means *transmitter chooses* and *transmitter follows*, respectively. Transition relations for the two other groups of agents (no agent, and both agents) are not shown.

*ATL* formulas can also be translated into µ-calculus formulas. The formula $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{G}\ \phi$ is translated into

$$\nu v.\ \phi' \wedge \Diamond_{\Gamma choose}\ (\Diamond_{\Gamma follow}\ true \wedge \Box_{\Gamma follow}\ v),$$

where $\phi'$ is the translation of $\phi$. For instance, $\langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent$, saying that the transmitter has a strategy to never transmit the value, is translated as

$$\nu v.\ \neg sent \wedge \Diamond_{trans\ chooses}\ (\Diamond_{trans\ follows}\ true \wedge \Box_{trans\ follows}\ v),$$

where the relations *trans chooses* and *trans follows* are shortcuts for *transmitter chooses* and *transmitter follows*, respectively. The idea behind this translation can be understood as follows. First, a state satisfies

$$\Diamond_{trans\ chooses}\ (\Diamond_{trans\ follows}\ true \wedge \Box_{trans\ follows}\ v)$$

if there exists an action for *transmitter* ($\Diamond_{trans\ chooses}$), such that the action is enabled ($\Diamond_{trans\ follows}\ true$), and all choices of the other agents lead to $v$ ($\Box_{trans\ follows}\ v$). In other words, $q$ satisfies the formula $\Diamond_{trans\ chooses}\ (\Diamond_{trans\ follows}\ true \wedge \Box_{trans\ follows}\ v)$ if the transmitter has an action to enforce to reach $v$ in one step. Second, a state satisfies

$$\nu v.\ \neg sent \wedge \Diamond_{trans\ chooses}\ (\Diamond_{trans\ follows}\ true \wedge \Box_{trans\ follows}\ v)$$

if the transmitter can enforce to stay in states satisfying $\neg sent$ forever, that is, if the transmitter has a strategy to enforce $\mathbf{G}\ \neg sent$.

To explain why a given $ATL$ formula $\phi$ is satisfied by a given state $q$ of some CGS $S$, we want to extract the part of the model starting at $q$ that is responsible for the satisfaction. Furthermore, as such part can be complex and difficult to understand, we want to annotate each state with the sub-formulas of $\phi$ that are true in these states.

For instance, an explanation for why the state $q_0$ satisfies the formula $\langle\langle transmitter\rangle\rangle\mathbf{G}\ \neg sent$ in the bit transmission problem is given in Figure 10.3. The explanation shows that, in $q_0$, the *block* action of the transmitter allows it to prevent the message to be sent. The goal of the µ-calculus based framework presented in this chapter is to help the designer of a new logic to implement a model checker with explanations for her logic, by translating her models and formulas into the µ-calculus, and translating the generated explanations back into the original language.

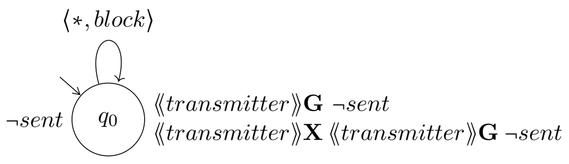

$\langle *, block\rangle$
$\neg sent$
$q_0$
$\langle\langle transmitter\rangle\rangle\mathbf{G}\ \neg sent$
$\langle\langle transmitter\rangle\rangle\mathbf{X}\ \langle\langle transmitter\rangle\rangle\mathbf{G}\ \neg sent$

Figure 10.3: An explanation for why the transmitter has a strategy to never transmit the value.

The remainder of this chapter is structured as follows: Section 10.1 defines µ-calculus explanations and describes an algorithm to generate them. Section 10.2 presents the functionalities to facilitate the translation of formulas and explanations. Section 10.3 briefly describes an implementation of the framework based on PyNuSMV. Finally, Section 10.4 extends the running example to the full $ATL$ logic, showing the applicability of the framework.

## 10.1 µ-calculus explanations

This section defines the notion of explanations for µ-calculus formulas and an algorithm to generate them. Given a structure $S$, a state $q$ of $S$, a µ-calculus formula $\phi$ with free variables $FVar$, and an environment $e$ defined for all variables of $FVar$—that is, a function associating a subset of states to each variable of $FVar$—, an explanation gives the reasons for why $q \in [\![\phi]\!]^S e$, that is, it explains why $q$ satisfies $\phi$ in the environment $e$.

An explanation is a graph where nodes are triplets—called *obligations*—composed of a state $q$ of $S$, a µ-calculus formula $\phi$, and an environment $e$. An edge $\langle\langle q, \phi, e\rangle, \langle q', \phi', e'\rangle\rangle$ of the graph encodes the fact that $q \in [\![\phi]\!]^S e$ *because* $q' \in [\![\phi']\!]^S e'$. In this section, all µ-calculus formulas are considered in *positive normal form*, that is, all negations are applied to atomic propositions or variables only.

More formally, given a Kripke structure $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$ labelled with atomic propositions from the set $AP$, an explanation is a graph $E = \langle O, T\rangle$ such that

- the nodes of the graph $O \subseteq Q \times \mathcal{L}_\mu \times \mathcal{E}$ are composed of triplets of states, µ-formulas and environments. $\mathcal{L}_\mu$ is the set of µ-calculus formulas over atomic propositions from $AP$ and variables from the set $Var$, $\mathcal{E}$ the set of environments defined for variables in $Var$.
- The edges $T \subseteq O \times O$ of the graph link obligations together. We note $succ(o) = \{o' \mid \langle o, o'\rangle \in T\}$ for the set of successors of $o$.

We are interested in explanations that effectively show why $q$ belongs to the interpretation of $\phi$ over $S$ in $e$ (that is, why $q \in [\![\phi]\!]^S e$). Such an explanation $E$ is *consistent* and composed of elements of $S$, $\phi$ and $e$. We call these explanations *adequate* explanations.

An explanation is *consistent* if it exhibits the different parts needed to explain its elements. More formally, let $E = \langle O, T\rangle$ be an explanation and let $o = \langle q, \phi, e\rangle \in O$. $o$ is said to be *locally consistent in $E$* iff

- $\phi \neq false$;
- if $\phi = true$, then $succ(o) = \emptyset$;
- if $\phi = p$ or $\phi = \neg p$, for an atomic proposition $p$, then $succ(o) = \emptyset$;
- if $\phi = v$, for a variable $v$, then $q \in e(v)$ and $succ(o) = \emptyset$;
- if $\phi = \neg v$, then $q \notin e(v)$ and $succ(o) = \emptyset$;
- if $\phi = \phi_1 \wedge \phi_2$ then $succ(o) = \{o_1, o_2\}$, where $o_1 = \langle q, \phi_1, e\rangle \in O$ and $o_2 = \langle q, \phi_2, e\rangle \in O$;

- if $\phi = \phi_1 \vee \phi_2$ then $succ(o) = \{o_j\}$, where $o_j = \langle q, \phi_j, e\rangle \in O$ for some $j \in \{1,2\}$;
- if $\phi = \Diamond_i \phi'$ then $succ(o) = \{o'\}$, where $o' = \langle q', \phi', e\rangle \in O$ for some state $q'$;
- if $\phi = \Box_i \phi'$ then for all $o' \in succ(o)$, $o' = \langle q', \phi', e\rangle \in O$ for some state $q'$;
- if $\phi = \mu v.\psi(v)$, then $succ(o) = \{o'\}$, where $o' = \langle q, \psi^k(false), e\rangle \in O$ for some $k \geq 0$;
- if $\phi = \nu v.\psi(v)$, then $succ(o) = \{o'\}$, where $o' = \langle q, \psi(\phi), e\rangle \in O$.

The notations $\mu v.\ \psi(v)$ and $\nu v.\ \psi(v)$ are used to stress the fact that $\psi$ depends on variable $v$. This emphasis is usually dropped in the sequel when the variable $v$ is clear from the context, in particular when it comes from a least or greatest fixpoint formula. Given a formula $\psi(v)$ depending on variable $v$, $\psi^k(\phi)$ is defined as $\psi^0(\phi) = \phi$, and $\psi^{k+1}(\phi) = \psi(\psi^k(\phi))$. Also, $\psi(\phi)$ is a shortcut for $\psi^1(\phi)$. $E$ is then *consistent* iff all obligations $o \in O$ are locally consistent in $E$.

Intuitively, if $\phi = \mu v.\ \psi$, then $q \in [\![\phi]\!]^S e$ because $q$ belongs to a finite number of applications of $\psi$ on $false$, that is, $q \in [\![\psi^k(false)]\!]^S e$ for some $k \geq 0$. On the other hand, this idea cannot be applied for $\phi = \nu v.\ \psi$. In this case, $q \in [\![\phi]\!]^S e$ because it belongs to any number of applications of $\psi$ on *true*. Thus, to explain it, $E$ simply shows that $q \in [\![\psi(\phi)]\!]^S e$ and relies on the fact that the structure has a finite number of states to ensure that the explanation is finite as well.

An explanation $E = \langle O, T\rangle$ is *adequate* for explaining why $q$ belongs to the interpretation of $\phi$ over $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$ in environment $e$ iff $E$ is consistent, $E$ *matches* $S$ and $\langle q, \phi, e\rangle \in O$. $E$ matches $S$ iff

1. for all $\langle q', \phi', e'\rangle \in O$, $q' \in Q$;
2. for all $\langle q', p, e'\rangle \in O$, $p \in V(q')$ and for all $\langle q', \neg p, e'\rangle \in O$, $p \notin V(q')$ for an atomic proposition $p$;
3. for all $\langle\langle q', \phi', e'\rangle, \langle q'', \phi'', e''\rangle\rangle \in T$, either $q' = q''$, or $\phi'$ belongs to $\{\Diamond_i \phi'', \Box_i \phi''\}$ and $\langle q', q''\rangle \in R_i$;
4. for all $o' = \langle q', \Box_i \phi', e'\rangle \in O$,

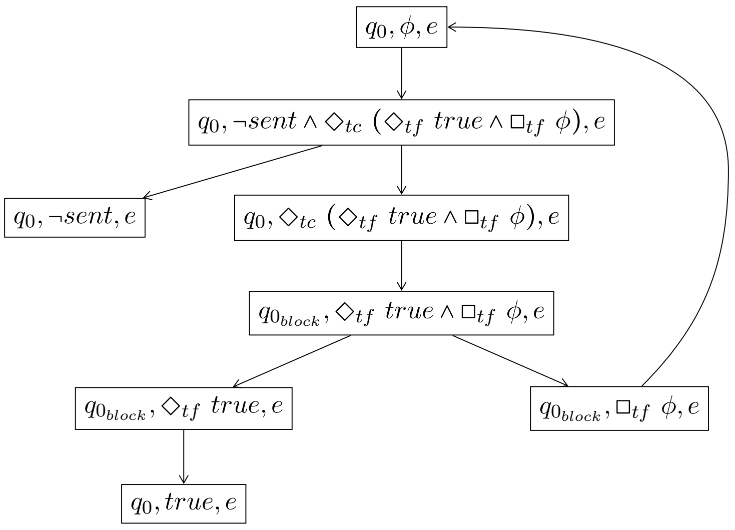
$$\begin{aligned}&\exists o'' \in succ(o') \text{ s.t. } o'' = \langle q'', \phi'', e''\rangle \text{ for some } \phi'', e''\\ &\iff \langle q', q''\rangle \in R_i.\end{aligned}$$

$E$ matches $S$ if $E$ is part of $S$: Point 1 says that the states of $E$ are states of $S$; Point 2 says that atomic propositions of $E$ are coherent with labels of $S$; Point 3 says that successor states in $E$ are successors in $S$; Point 4 is an additional condition that says that the explanation for the $\Box_i$ operator exhibits all successors through $R_i$.

For instance, an adequate explanation for why $q_0$ of the Kripke structure of Figure 10.2—the µ-calculus structure for the bit transmission problem—belongs to the interpretation of

$$\phi = \nu v.\ \neg sent \wedge \Diamond_{trans\ chooses}\ (\Diamond_{trans\ follows}\ true \wedge \Box_{trans\ follows}\ v)$$

is given in Figure 10.4.



$q_0, \phi, e$
$q_0, \neg sent \wedge \Diamond_{tc}\ (\Diamond_{tf}\ true \wedge \Box_{tf}\ \phi), e$
$q_0, \neg sent, e$
$q_0, \Diamond_{tc}\ (\Diamond_{tf}\ true \wedge \Box_{tf}\ \phi), e$
$q0_{block}, \Diamond_{tf}\ true \wedge \Box_{tf}\ \phi, e$
$q0_{block}, \Diamond_{tf}\ true, e$
$q0_{block}, \Box_{tf}\ \phi, e$
$q_0, true, e$

Figure 10.4: An explanation for why $q_0 \in [\![\phi]\!]^S e$ in the bit transmission problem.

Adequate explanations are necessary and sufficient proofs for why $q \in [\![\phi]\!]^S e$, captured by the following property. Its proof is given in Appendix B.

**Property 10.1.** *Given a Kripke structure $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$, a state $q \in Q$, a $\mu$-calculus formula $\phi$ and an environment $e$, $q \in [\![\phi]\!]^S e$ if and only if there exists an adequate explanation $E$ for $q \in [\![\phi]\!]^S e$.*

Furthermore, we can view adequate explanations as patterns. Let us consider an explanation $E$. $E$ defines an entire set of Kripke structures $\mathcal{K}(E)$ that $E$ matches. We can see these Kripke structures as the set of models that are coherent with $E$. $E$ is thus an explanation for why all structures of $\mathcal{K}(E)$ satisfy any formula $\phi$ that $E$ contains. This intuition is formally captured by the following property.

**Property 10.2.** *Given a consistent explanation $E = \langle O, T \rangle$, for all $\langle q, \phi, e \rangle \in O$, $q \in [\![\phi]\!]^S e$ for all $S$ such that $E$ matches $S$.*

*Proof.* This property is directly derived from Property 10.1. If $E$ is consistent, $E$ matches $S$ and $\langle q, \phi, e \rangle \in O$, $E$ is adequate for $q \in [\![\phi]\!]^S e$. By Property 10.1, since there exists an adequate explanation for $q \in [\![\phi]\!]^S e$, $q \in [\![\phi]\!]^S e$ is true. □

Finally, we can define an algorithm to generate adequate explanations for µ-calculus formulas, presented in Algorithm 10.1. It takes as arguments a Kripke structure $S$, a state $q$ of $S$, a µ-calculus formula $\phi$, and an environment $e$ such that $q \in [\![\phi]\!]^S e$. It then returns an adequate explanation for $q \in [\![\phi]\!]^S e$.

Intuitively, the algorithm starts with an empty explanation and adds the $\langle q, \phi, e \rangle$ obligation into the *pending* set. Then it considers each obligation $o' \in pending$, adding to $O$ and $T$ the necessary obligations and edges to make $o'$ locally consistent, and adding to *pending* the newly discovered obligations. The algorithm thus stops the process when all pending obligations have been made locally consistent in $\langle O, T \rangle$.

This algorithm supposes that, for each state $q'$ of $S$ and each sub-formula $\phi'$ of $\phi$, it is determined whether $q' \in [\![\phi']\!]^S e$. It is necessary for cases such as $\phi_1 \vee \phi_2$ or $\Diamond_i\ \phi_1$ because the algorithm must be able to choose the right sub-formula of $\phi'$ or the right successor of $q'$.

The correctness of this *explain* algorithm is proved in Appendix B.

## 10.2 Translating µ-calculus explanations

The previous section proposed a structure to explain why a µ-calculus formula is satisfied by a state of some Kripke structure. Nevertheless, as the µ-calculus model checker and explanations are used to solve the model-checking problem of some other top-level logic, the usefulness of such explanations is limited.

This section presents a set of functionalities to help the top-level logic designer to translate these µ-calculus explanations back into the top-level logic. They are generic to allow her to easily translate the explanations for many top-level logics such as $CTL$, $CTLK$, $ATL$ or $PDL$, as well as their fair variants such as $FCTL$ and $ATLK_{IrF}$.

The functionalities provided by the framework are presented in the following six sections:

**Algorithm 10.1:** *explain*$(S, q, \phi, e)$

```
Data: $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$ a Kripke structure, $q \in Q$ a state of
      $S$, $\phi$ a μ-calculus formula, and $e$ an environment such that
      $q \in [\![\phi]\!]^S e$.
Result: An adequate explanation for $q \in [\![\phi]\!]^S e$.
$O = \varnothing$; $T = \varnothing$
$pending = \{\langle q, \phi, e\rangle\}$
while $pending \neq \varnothing$ do
    pick $o' = \langle q', \phi', e'\rangle \in pending$
    $pending = pending \backslash \{o'\}$
    $O = O \cup \{o'\}$
    case $\phi' \in \{true, p, \neg p, v, \neg v\}$
        $O' = \varnothing$
    case $\phi' = \phi_1 \wedge \phi_2$
        $O' = \{\langle q', \phi_1, e'\rangle, \langle q', \phi_2, e'\rangle\}$
    case $\phi' = \phi_1 \vee \phi_2$
        if $q' \in [\![\phi_1]\!]^S e'$ then $O' = \{\langle q', \phi_1, e'\rangle\}$
        else $O' = \{\langle q', \phi_2, e'\rangle\}$
    case $\phi' = \Diamond_i\ \phi''$
        pick $q'' \in \{q'' \in Q \mid \langle q', q''\rangle \in R_i \wedge q'' \in [\![\phi'']\!]^S e'\}$
        $O' = \{\langle q'', \phi'', e'\rangle\}$
    case $\phi' = \Box_i\ \phi''$
        $O' = \{\langle q'', \phi'', e'\rangle \mid \langle q', q''\rangle \in R_i\}$
    case $\phi' = \mu v.\ \psi$
        $\phi'' = false$; $sat = [\![\phi'']\!]^S e'$
        while $q' \notin sat$ do
            $\phi'' = \psi(\phi'')$; $sat = [\![\phi'']\!]^S e'$
        $O' = \{\langle q', \phi'', e'\rangle\}$
    case $\phi' = \nu v.\ \psi$
        $O' = \{\langle q', \psi(\phi'), e'\rangle\}$
    $T = T \cup \{\langle o', o''\rangle \mid o'' \in O'\}$
    $pending = pending \cup (O' \backslash O)$
return $\langle O, T\rangle$
```

1. Section 10.2.1 presents the notion of formula aliases. They link the formulas stored in the obligations to the top-level logic formulas they represent.

2. Section 10.2.2 shows how the relational graph algebra of Dong et al. can be used to manipulate explanations and derive new graphs from them. This algebra allows the designer to transform the original μ-calculus explanation into the part of the original model responsible for the model checking outcome.

3. The relational graph algebra treats the explanation *as a whole*. Section 10.2.3 describes obligation and edge attributes. They add information to *individual* nodes and edges of the explanation graph.

4. Attributes allow the designer to add information to obligations and edges, and the relational graph algebra allows her to manipulate the explanation as a whole. Section 10.2.4 presents local translation, a way to focus on the small part that explains a given alias without having to deal with the whole graph at once.

5. The previous functionalities help the designer to manipulate and transform the generated explanation. Nevertheless, the generating algorithm produces one arbitrary explanation among the possible ones. Section 10.2.5 presents the notion of choosers and explains how they can be used to perform interactive or guided generation of explanations. These choosers introduce the notion of *partial explanations* that are also described in this section.

6. Finally, Section 10.2.6 describes the notion of formula markers. They are generic tags on formulas, and two instantiations are presented: *points of interest* mark the formulas that the user would have an interest in—for instance, formulas corresponding to top-level logic operators—, and *points of decision* mark the formulas that should remain unexplained to produce partial explanations.

All these functionalities work together to help the designer to produce useful explanations. Figure 10.5 illustrates the structure of the framework; in gray, the parts that the designer has to define; in white, the elements provided by the framework.

The designer first translates the original model into a μ-calculus one. She also translates the formula. She can enrich the translated formulas with aliases and markers, and she can also attach attributors, local translators and choosers. The aliases and markers will be present in the obligations in the generated enriched μ-calculus explanation to help the designer with the translation of explanations.

The attributors and local translators are used by the model checker to add extra information to the generated explanations. The choosers allow the generation process to make the right choices. Finally, the designer

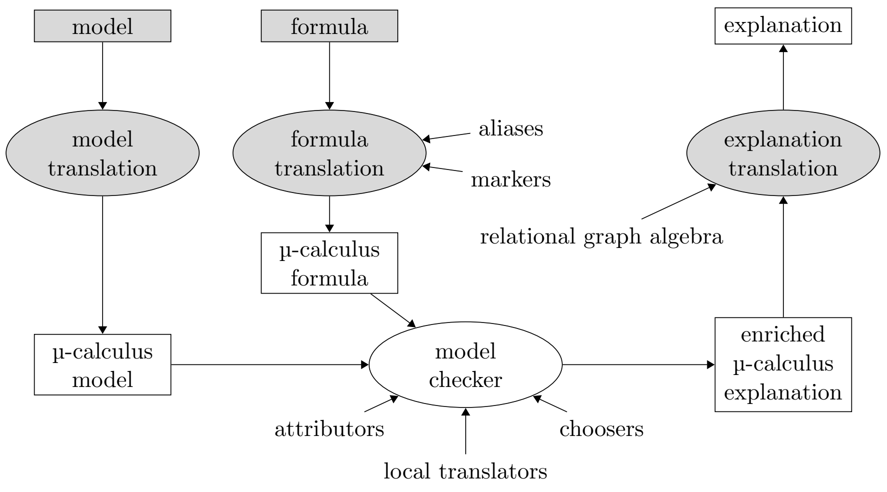


Figure 10.5: The structure of the framework.

translates the enriched explanation back into the top-level logic language thanks to the relational graph algebra.

### 10.2.1 Aliases

Aliases allow the designer to hide µ-calculus translations behind top-level logic formulas. An alias is a syntactic function. It takes a set of arguments and returns an *aliased* µ-calculus formula. Such an aliased formula is a standard µ-calculus formula $\phi$ to which is attached the alias $\alpha$ and the correspondence between $\phi$ sub-elements and $\alpha$ arguments. In the sequel, we say that $\phi$ is *decorated with* $\alpha(a_1, ..., a_n)$, where $a_1$, ..., $a_n$ are the sub-elements of $\phi$ corresponding the the arguments of $\alpha$.

For instance, the alias $\langle\langle\rangle\rangle\mathbf{G}$, defined as

$$\langle\langle\rangle\rangle\mathbf{G}(\Gamma, \phi) = \nu v.\ \phi \wedge \Diamond_{\Gamma choose}\ (\Diamond_{\Gamma follow}\ true \wedge \Box_{\Gamma follow}\ v),$$

takes a group of agents $\Gamma$ and a µ-calculus formula $\phi$ as arguments and returns the aliased formula corresponding to the greatest fixpoint above. In the sequel, we usually write the arguments of the aliases in place, such as $\langle\langle\Gamma\rangle\rangle\mathbf{G}\ \phi$ instead of $\langle\langle\rangle\rangle\mathbf{G}(\Gamma, \phi)$, for better presentation.

Let $\phi$ be an aliased formula. Let alias $\alpha(a_1, ..., a_n)$ be the alias decorating $\phi$. In the sequel, both $\phi$ and $\alpha(\phi_1, ..., \phi_n)$ are interchangeably used to represent the same aliased formula. For instance, the aliased formula $\phi$, decorated with $\langle\langle\rangle\rangle\mathbf{G}(\{transmitter\}, \neg sent)$ and defined as

$$\begin{aligned}&\langle\langle\rangle\rangle\mathbf{G}(\{transmitter\}, \neg sent) =\\ &\nu v.\ \neg sent \wedge \Diamond_{trans\ chooses}\ (\Diamond_{trans\ follows}\ true \wedge \Box_{trans\ follows}\ v)\end{aligned}$$

is usually written $\langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent$ in the sequel. This illustrates the usefulness of aliases: they hide µ-calculus formulas behind something more intelligible.

Aliases and aliased formulas support substitution. More precisely, when substituting a sub-formula in an aliased formula, the alias decorates the resulting new formula, but only if the actual substitutions happen in arguments of the alias, keeping the alias integrity. For instance, let $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi$ be an alias for $\Diamond_{\Gamma choose}\ (\Diamond_{\Gamma follow}\ true \wedge \Box_{\Gamma follow}\ \phi)$.

Let $\psi = \neg sent \wedge \langle\!\langle transmitter\rangle\!\rangle\mathbf{X}\ v$. When substituting $v$ for another formula, the $\langle\!\langle\rangle\!\rangle\mathbf{X}$ alias is kept:

$$\begin{aligned}&\psi[false/v] =\\ &\neg sent \wedge \Diamond_{trans\ chooses}\ (\Diamond_{trans\ follows}\ true \wedge \Box_{trans\ follows}\ false) =\\ &\neg sent \wedge \langle\!\langle transmitter\rangle\!\rangle\mathbf{X}\ false.\end{aligned}$$

On the other hand, when substituting $\Diamond_{trans\ follows}\ true$ with $true$, for instance, the $\langle\!\langle\rangle\!\rangle\mathbf{X}$ alias is lost, as intended:

$$\begin{aligned}&\psi[true/\ \Diamond_{trans\ follows}\ true] =\\ &\neg sent \wedge \Diamond_{trans\ chooses}\ (true \wedge \Box_{trans\ follows}\ v).\end{aligned}$$

Finally, aliases support negation, in the sense that the designer can define two aliases and tell that the first one is the negation of the second. For instance, let $[\![\Gamma]\!]\mathbf{X}\ \phi$ be an alias for

$$\Box_{\Gamma choose}\ (\Box_{\Gamma follow}\ false \vee \Diamond_{\Gamma follow}\ \phi).$$

By telling that $\neg\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi = [\![\Gamma]\!]\mathbf{X}\ \neg\phi$, and that $\neg[\![\Gamma]\!]\mathbf{X}\ \phi = \langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \neg\phi$, the designer informs the framework that when deriving positive normal forms of formulas, the negation of the alias $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi$ is the alias $[\![\Gamma]\!]\mathbf{X}\ \neg\phi$, and vice versa.

To illustrate the usefulness of aliases, Figure 10.6 presents the explanation of Figure 10.4 for why $q_0$ of the structure of the bit transmission problem satisfies the formula $\langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent$, with the $\langle\!\langle\rangle\!\rangle\mathbf{X}$ alias defined above and the $\langle\!\langle\rangle\!\rangle\mathbf{G}$ alias redefined as $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{G}\ \phi = \nu v.\ \phi \wedge \langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ v$. Some formulas are unchanged, but the ones that are aliased are more understandable by the end user.

The main goal of aliases is to give intelligible names to whole µ-calculus formulas. As most of the formulas appearing in an explanation come from the top-level logic designer, she has the freedom to define aliases for some sub-formulas that have a meaning in the top-level logic while keeping the others in plain µ-calculus.

$$\begin{tikzcd}
& & \boxed{q_0, \langle\!\langle transmitter \rangle\!\rangle \mathbf{G}\ \neg sent, e} \arrow[d] & \\
& & \boxed{q_0, \neg sent \wedge \langle\!\langle transmitter \rangle\!\rangle \mathbf{X}\ \langle\!\langle transmitter \rangle\!\rangle \mathbf{G}\ \neg sent, e} \arrow[dll] \arrow[d] & \\
\boxed{q_0, \neg sent, e} & & \boxed{q_0, \langle\!\langle transmitter \rangle\!\rangle \mathbf{X}\ \langle\!\langle transmitter \rangle\!\rangle \mathbf{G}\ \neg sent, e} \arrow[d] & \\
& & \boxed{q_{0_{block}}, \Diamond_{tf}\ true \wedge \Box_{tf}\ \langle\!\langle transmitter \rangle\!\rangle \mathbf{G}\ \neg sent, e} \arrow[dl] \arrow[dr] & \\
& \boxed{q_{0_{block}}, \Diamond_{tf}\ true, e} \arrow[d] & & \boxed{q_{0_{block}}, \Box_{tf}\ \langle\!\langle transmitter \rangle\!\rangle \mathbf{G}\ \neg sent, e} \arrow[uuuul, bend right] \\
& \boxed{q_0, true, e} & &
\end{tikzcd}$$

Figure 10.6: An explanation for why $q_0 \in [\![ \langle\!\langle transmitter \rangle\!\rangle \mathbf{G}\ \neg sent ]\!]^S e$ in the bit transmission problem, with aliases for the important formulas.

The *explain* algorithm nevertheless derives two sets of formulas not written by the designer: $\mu v.\ \psi$ formulas are extended as $\psi^k(false)$ for some $k$, and $\nu v.\ \psi$ ones are extended as $\psi(\nu v.\ \psi)$. While the latter is usually not problematic as it expands the $\psi$ body only once, the former results in a rather large formula when the value of $k$ is high.

For instance, let $\langle\!\langle \Gamma \rangle\!\rangle \mathbf{F}\ \phi'$ be an alias for

$$\mu v.\ \phi' \vee \langle\!\langle \Gamma \rangle\!\rangle \mathbf{X}\ v,$$

and let us consider the formula $\phi = \langle\!\langle sender \rangle\!\rangle \mathbf{F}\ sent$. If some state $q$ satisfies $\phi$, then an explanation will contain, for instance, an obligation with the formula

$$sent \vee \langle\!\langle sender \rangle\!\rangle \mathbf{X}\ (sent \vee \langle\!\langle sender \rangle\!\rangle \mathbf{X}\ (sent \vee \langle\!\langle sender \rangle\!\rangle \mathbf{X}\ false)), \tag{10.1}$$

if *sender* can force to reach *sent* within 3 steps. Such a formula is already long to read and understand, even with a low value of $k$ ($k = 3$).

To solve this problem, an alias of the form $(\psi)^k(v = false)$ is attached to the $\phi' = \psi^k(false)$ formula. More precisely, for every $k'$ from 1 to $k$, the sub-formula $\psi^{k'}(false)$ is linked to the corresponding alias $(\psi)^{k'}(v = false)$. For instance, the formula of Equation 10.1 is rendered as

$$(sent \vee \langle\!\langle sender \rangle\!\rangle \mathbf{X}\ v)^3(v = false).$$

Furthermore, its sub-formula

$$sent \vee \langle\langle sender\rangle\rangle\mathbf{X}\ (sent \vee \langle\langle sender\rangle\rangle\mathbf{X}\ false)$$

is aliased by

$$(sent \vee \langle\langle sender\rangle\rangle\mathbf{X}\ v)^2(v = false),$$

and so on. The goal of these aliases is to reduce the size of the aliased formulas, making them more readable for the end user.

### 10.2.2 Relational graph algebra

Aliases allow the designer to give intelligible names to µ-calculus formulas with meaningful correspondences to the top-level language. Nevertheless, the explanation of Figure 10.6 is still far from the original model of the bit transmission problem.

To ease the translation of explanations back into the original model language, the framework integrates the relational graph algebra of Dong et al. (see Section 9.4.3 and [DRS03a]) to manipulate and transform explanations.

Relational graphs are graphs where nodes are labelled with elements of some domain $D_V$, and edges with elements from $D_E$. A µ-calculus explanation can thus be viewed as a relational graph where nodes are labelled with triplets of values from the domain $Q \times \mathcal{L}_\mu \times \mathcal{E}$, and where edges are labelled with elements of the empty domain, as the edges carry no value apart from their extremities.

The relational operators include the simple union $G_1 \cup G_2$ and intersection $G_1 \cap G_2$ of two relational graphs $G_1$ and $G_2$, the selection $\sigma_{f_v,f_e}(G)$ of some nodes and edges of a graph $G$ satisfying $f_v$ and $f_e$ resp., the projection $\pi_{d_v,d_e}(G)$ of nodes and edges on sub-domains $d_v$ and $d_e$ resp., and the cross-product $G_1 \times G_2$ of two graphs. From these operators, other ones can be defined such as the extension $\epsilon_{x_v,x_e}(G)$ of one graph with new domains on nodes and edges for which elements are obtained through $x_v$ and $x_e$ resp., the mapping $\rho_{x_v,x_e}(G)$ of a graph replacing nodes and edges with results of $x_v$ and $x_e$ resp., or the grouping $\gamma_{d_v,d_e}(G)$ of nodes and edges with the same values for domains $d_v$ and $d_e$ resp.

Thanks to this algebra, the designer can more easily translate explanations back into the original model language. For instance, let $E$ be the explanation of Figure 10.7. This explanation is derived from the explanation of Figure 10.6 to which we added an edge decorated with an action *block*. This graph can thus be viewed as a relational graph

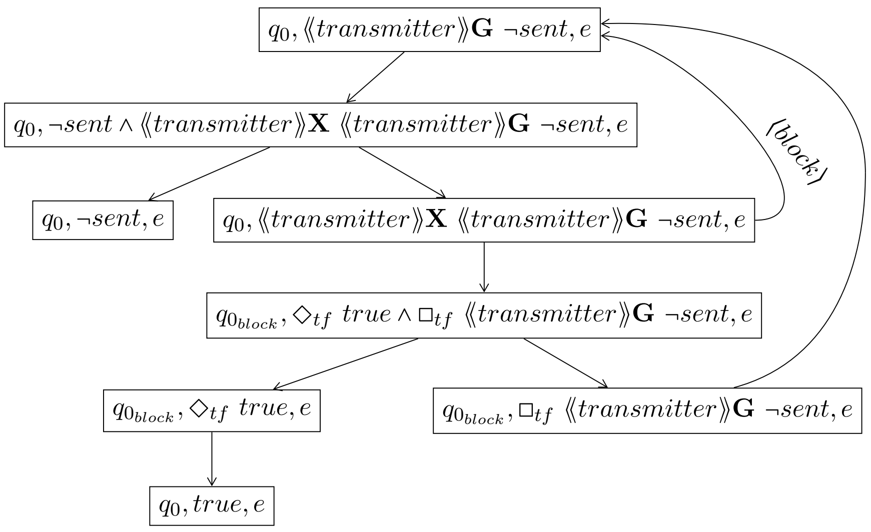

$q_0, \langle\!\langle transmitter \rangle\!\rangle \mathbf{G}\ \neg sent, e$
$q_0, \neg sent \wedge \langle\!\langle transmitter \rangle\!\rangle \mathbf{X}\ \langle\!\langle transmitter \rangle\!\rangle \mathbf{G}\ \neg sent, e$
$q_0, \neg sent, e$
$q_0, \langle\!\langle transmitter \rangle\!\rangle \mathbf{X}\ \langle\!\langle transmitter \rangle\!\rangle \mathbf{G}\ \neg sent, e$
$\langle block \rangle$
$q0_{block}, \Diamond_{tf}\ true \wedge \Box_{tf}\ \langle\!\langle transmitter \rangle\!\rangle \mathbf{G}\ \neg sent, e$
$q0_{block}, \Diamond_{tf}\ true, e$
$q0_{block}, \Box_{tf}\ \langle\!\langle transmitter \rangle\!\rangle \mathbf{G}\ \neg sent, e$
$q_0, true, e$

Figure 10.7: An explanation for why $q_0 \in [\![\langle\!\langle transmitter \rangle\!\rangle \mathbf{G}\ \neg sent]\!]^S e$ in the bit transmission problem, with aliased formulas and an extra edge.

with $Q' \times \mathcal{L}_\mu \times \mathcal{E}$ as the node domain, where $Q'$ is the set of states of the µ-calculus model, and with $Act$ as the edge domain.

The designer can extract, from the obligations, the original state using the algebra extension operator $\epsilon$:

$$E' = \epsilon_{state,id}(E),$$

where $id$ is the identity function—so edges are not extended with any new value—, and $state(\langle q', \phi', e' \rangle) = original(q')$, that is, nodes are extended with the original state corresponding to $q'$. $E'$ is thus defined on the node domain $Q' \times \mathcal{L}_\mu \times \mathcal{E} \times Q$—where $Q$ are the states of the original model—, and on the edge domain $Act$.

The designer can then project the result on the formulas and original states with the projection operator $\pi$:

$$E'' = \pi_{\mathcal{L}_\mu \times Q, Act}(E').$$

$E''$ is thus defined on the node domain $\mathcal{L}_\mu \times Q$ and on the edge domain $Act$. Intuitively, $E''$ is $E'$ but we removed the derived state and the environment from every node.

From $E''$, the designer can then group obligations according to their original state, accumulating all the formulas in a new domain, filter these accumulated formulas to keep the important ones only, and finally keep the edges that are labelled with some action. That is, the final result $E^f$

is given by

$$E^f = \sigma_{true,actions}\left(\rho_{important,id}\left(\gamma_{Q,empty}(E'')\right)\right), \qquad (10.2)$$

where

- *empty* is the empty domain, $\gamma_{Q,empty}(E'')$ is thus defined on the node domain $Q \times 2^{\mathcal{L}_\mu}$ and on the edge domain $Act$.
- *important* takes a tuple $\langle q, \{\phi\}\rangle \in Q \times 2^{\mathcal{L}_\mu}$ and returns the tuple $\langle q, \{\phi \mid \phi \text{ is important}\}\rangle$ composed of the same state and the formulas that are important (this notion is kept vague here, and will be developed further in Section 10.2.6).
- *true* is the function selecting all nodes.
- *actions* is the function selecting all edges with an attached action.

The $E^f$ explanation is illustrated in Figure 10.8. It is defined on the node domain $Q \times 2^{\mathcal{L}_\mu}$ and on the edge domain $Act$. This final graph is similar to the explanation of Figure 10.3.

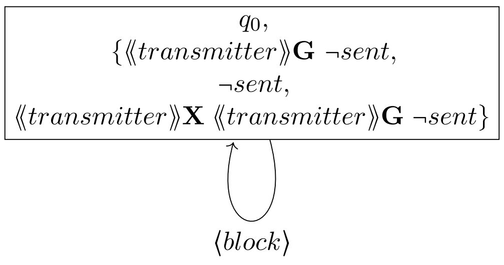

$q_0$,
$\{\langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent$,
$\neg sent$,
$\langle\!\langle transmitter\rangle\!\rangle\mathbf{X}\ \langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent\}$
$\langle block\rangle$

Figure 10.8: An explanation for why $q_0 \in [\![\langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent]\!]^S e$ in the bit transmission problem, translated using the relational graph algebra.

### 10.2.3 Obligation and edge attributes

The relational graph algebra allows the designer to translate the explanation back into the original model language, but it treats the explanation as a whole. In the given example, the translation first extends the information of obligations and edges before transforming the graph. To ease the addition of information to obligations and edges, the framework provides the notion of attributes and attributors. An *attribute* is an element of a relational graph domain, and an *attributor* is a function taking an obligation or an edge and returning attributes to add to this element. If the function works on obligations, we call it an *obligation attributor*, if it takes an edge as argument, we call it an *edge attributor*.

Attributors work as local decorators, in the sense that they deal with obligations and edges one at a time. They can be given to the generating algorithm to be run on every obligation or edge after generating the explanation, or they can be attached to individual aliases to be run only on the obligations with instantiations of the aliases, or outgoing edges of these obligations. This improves the performances of decorating the graph when only a few elements must be decorated. For instance, the designer of the $ATL$ model checker can decorate every outgoing edge of an obligation labelled with a $\langle\langle\Gamma\rangle\rangle\mathbf{X}$ formula with the action chosen by the agents. As there are only few such edges, using an attributor would make the decoration process more efficient than extending the edges with a relational graph operation.

Updating the generating algorithm of Section 10.1 to take attributors into account is not difficult. The main idea is to take additional attributors as arguments, and to post-process the explanation graph to add attributes when needed. More precisely, Algorithm 10.2 computes the original explanation using the *explain* algorithm, then runs every obligation through the obligation attributors, that is, the ones given in argument as well as the ones attached to the obligation formula ($\phi'.attributors$).

Then it runs every edge through the edge attributors. The attributors run on an edge $e$ are the ones attached to the formula of the origin of $e$ ($origin.formula.attributors$). This allows the designer to easily attach edge attributors. Furthermore, edge attributors take the decorated obligations at both ends of the edge instead of the original ones. This allows obligation attributors to add to obligations information that could be useful for edge attributors. To this end, the algorithm keeps in the $new\_O$ dictionary the correspondence between original and decorated obligations. More precisely, $new\_O[o] = o'$ makes the decorated obligation $o'$ correspond to the original obligation $o$, and $new\_O[o]$ returns the decorated obligation corresponding to $o$. Furthermore, the attributors linked to formulas as well as the ones given as argument are ordered into a list. This allows, again, the last attributors to rely on information added by the previous ones.

Obligation and edge attributors can be replaced by the relational graph algebra. Indeed, an attributor adds information to a graph element, and this can be done with the extension operator $\epsilon$. We nevertheless keep both because attributors can be attached to particular formulas, restricting the attributor to a small set of graph elements, and simplifying its definition.

**Algorithm 10.2:** $explain\_with\_attr(S, q, \phi, e, attributors)$

```
Data: $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$ a Kripke structure, $q \in Q$ a state of
      $S$, $\phi$ a µ-calculus formula, and $e$ an environment such that
      $q \in [\![\phi]\!]^S e$, and $attributors$ a list of attributors.
Result: An adequate explanation for $q \in [\![\phi]\!]^S e$ augmented with
        attributes.
$\langle O, T\rangle = explain(S, q, \phi, e)$
$O' = \varnothing$; $T' = \varnothing$
$new\_O$ = empty dictionary
for $o = \langle q', \phi', e'\rangle \in O$ do
    $o' = o$
    $attrs = \phi'.attributors + attributors$
    $obl\_attrs = \langle attr \in attrs \mid attr$ is an obligation attributor$\rangle$
    for $attributor \in obl\_attrs$ do
        $o' = o' + attributor(o')$
    $O' = O' \cup \{o'\}$
    $new\_O[o] = o'$
for $\langle origin, end\rangle \in T$ do
    $e' = \langle\rangle$
    $attrs = origin.formula.attributors + attributors$
    $edge\_attrs = \langle attr \in attrs \mid attr$ is an edge attributor$\rangle$
    for $attributor \in edge\_attrs$ do
        $e' = e' + attributor(new\_O[origin], e', new\_O[end])$
    $T' = T' \cup \{\langle new\_[origin], e', new\_O[end]\rangle\}$
return $\langle O', T'\rangle$
```

### 10.2.4 Local translation

The relational graph algebra treats the explanation as a whole, and the attributors only see one element at a time. The framework also provides local translators to treat a part of the given graph. A local translator receives a sub-graph of the explanation and can update it. It is useful, for instance, for adding the edge with the action shown in Figure 10.7: instead of having to search for obligations with a $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi$ formula through the whole explanation, a local translator receives the sub-graph explaining the aliased formula, and can manipulate it.

More precisely, a local translator is a function taking a relational graph and a particular node as arguments, and that returns a new relational graph. To translate a small part of the explanation in an isolated manner, it is necessary to define what part of the explanation to isolate. The framework uses aliases to define this part: a local translator is attached to an alias, and the graph it receives is the part of the explanation that explains a formula aliased by the alias. Aliases can have sub-formulas, such as the formula $\phi$ in the alias $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi$. In this case, the local translator receives the part of the explanation that starts at the obligation labelled with the aliased formula, and that stops at obligations labelled with sub-formulas of the alias.

For instance, the part of the explanation of Figure 10.6 explaining the aliased formula $\langle\!\langle transmitter\rangle\!\rangle\mathbf{X}\ \langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent$ is given in Figure 10.9. The whole explanation is given for clarity, and the part explaining the $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}$ alias is in bold. This is the sub-graph a local translator attached to the $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}$ alias would receive as argument.

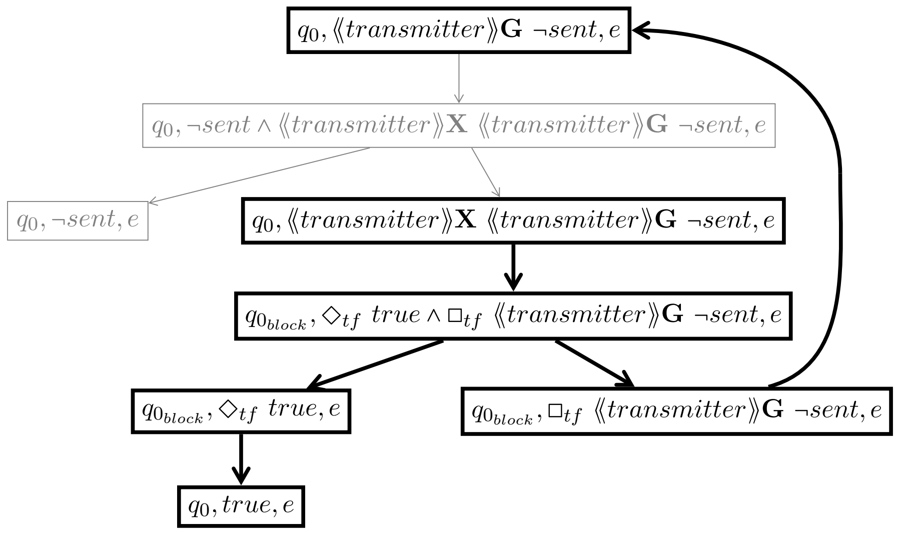

$q_0, \langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent, e$
$q_0, \neg sent \wedge \langle\!\langle transmitter\rangle\!\rangle\mathbf{X}\ \langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent, e$
$q_0, \neg sent, e$
$q_0, \langle\!\langle transmitter\rangle\!\rangle\mathbf{X}\ \langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent, e$
$q0_{block}, \Diamond_{tf}\ true \wedge \Box_{tf}\ \langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent, e$
$q0_{block}, \Diamond_{tf}\ true, e$
$q0_{block}, \Box_{tf}\ \langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent, e$
$q_0, true, e$

Figure 10.9: An explanation for why $q_0$ satisfies $\langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent$ in the bit transmission problem. The bold part explains the $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}$ alias.

More precisely, let $\phi$ be a µ-calculus formula with alias $\alpha(\phi_1, ..., \phi_n)$. In other words, $\phi$ is a formula to which is attached the alias $\alpha$, and sub-formulas $\phi_1$, ..., $\phi_n$ are sub-formulas of $\phi$ and arguments of $\alpha$. For instance, the formula

$$\nu v.\ \neg sent \wedge \Diamond_{trans\ chooses}\ (\Diamond_{trans\ follows}\ true \wedge \Box_{trans\ follows}\ v)$$

is aliased as $\langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent$, the $\langle\!\langle\rangle\!\rangle\mathbf{G}(\Gamma, \phi)$ alias is attached to it, and the only sub-formula of this alias is $\neg sent$.

Given an explanation $E = \langle O, T\rangle$ and an obligation $o$ with an aliased formula—noted $o.formula$—with alias $\alpha(\phi_1, ..., \phi_n)$, the part of the explanation explaining $\alpha(\phi_1, ..., \phi_n)$ is the graph containing all the obligations and edges encountered on any prefix of a path from $o$ to the first obligation with $\phi_i$ attached, for some $i \in \{1, ..., n\}$. For this definition to make sense, $o$ must be connected to the obligations with $\phi_i$. This is effectively the case for the original adequate explanations as they are consistent, and it is still the case for the explanations augmented with attributes, as attributes do not change the connectivity of obligations.

Algorithm 10.3 extracts, from an explanation $E = \langle O, T\rangle$ and an obligation $o \in O$, the part of the explanation explaining $o.formula$, assuming that $o.formula$ is aliased with $\alpha(\phi_1, ..., \phi_n)$. Starting from empty $O'$ and $T'$, it accumulates in them the obligations and edges encountered during a traversal of $E$ that stops at obligations with formulas in $\{\phi_1, ..., \phi_n\}$.

**Algorithm 10.3:** *extract_local*$(E, o)$

```
Data: $E = \langle O, T\rangle$ an explanation, and $o \in O$ an obligation such
      that $o.formula$ is aliased with $\alpha(\phi_1, ..., \phi_n)$.
Result: The part of $E$ explaining $\alpha(\phi_1, ..., \phi_n)$.
$O' = \emptyset$; $T' = \emptyset$
$pending = \{o\}$
while $pending \neq \emptyset$ do
    pick $o' \in pending$
    $pending = pending \setminus \{o'\}$
    $O' = O' \cup \{o'\}$
    if $o'.formula \notin \{\phi_1, ..., \phi_n\}$ then
        $T' = T' \cup \{\langle origin, edge, end\rangle \in T \mid origin = o'\}$
        $pending = pending \cup (succ(o') \setminus O')$
return $\langle O', T'\rangle$
```

The *extract_local* algorithm can be integrated in the generating algorithm. Indeed, after generating the original explanation and running attributors on its elements, we can locally translate all parts that can be translated. The local explanation of some alias can be contained in the local explanation of some other alias. For instance, the local explanation of the $\langle\!\langle\rangle\!\rangle\mathbf{X}$ alias is contained in the local explanation of the $\langle\!\langle\rangle\!\rangle\mathbf{G}$ alias. In this case, *extract_local* can only work with the explanation in which the part for $\langle\!\langle\rangle\!\rangle\mathbf{X}$ has already been translated if this part maintains the connectivity property of the graph. To ensure that it is always the case, the proposed algorithm *augments* the explanation with the translated graph instead of *replacing* it.

More precisely, Algorithm 10.4 performs the local translation of parts of the explanation with translators attached to aliased formulas. It first gets the explanation augmented with attributes, and then locally translates the parts that can be locally translated. For this, it computes in *to_translate* the list of obligations with an attached local translator. Then it accumulates in the *subgraphs* dictionary the sub-graphs that must be translated. The *to_translate* list is then sorted by the *Sort* algorithm that reorders obligations according to the graph inclusion of the corresponding sub-graphs—that is, for two obligations $o_1$ and $o_2$ from *to_translate*, $o_1 \leq o_2$ if and only if $subgraphs[o_1] \leq subgraphs[o_2]$, and $\langle O_1, T_1 \rangle \leq \langle O_2, T_2 \rangle$ if and only if $O_1 \subseteq O_2$ and $T_1 \subseteq T_2$. Finally, it locally translates each sub-graph according to the order of *to_translate*. After translating each sub-graph, the translated graph is *added* to the explanation. The final result is then the explanation where every sub-graph has been translated.

**Algorithm 10.4:** $explain\_translate(S, q, \phi, e, attributors)$

```
Data: $S = \langle Q, \{R_i \mid i \in \Sigma\}, V \rangle$ a Kripke structure, $q \in Q$ a state of
      $S$, $\phi$ a µ-calculus formula, and $e$ an environment such that
      $q \in [\![\phi]\!]^S e$, and $attributors$ a list of attributors.
Result: A locally translated adequate explanation for $q \in [\![\phi]\!]^S e$
        augmented with attributes.
$\langle O, T \rangle = explain\_with\_attr(S, q, \phi, e, attributors)$
$to\_translate = \langle o \in O \mid o.formula$ has a local translator$\rangle$
$subgraphs$ = empty dictionary
for $o \in to\_translate$ do
    $subgraphs[o] = extract\_local(\langle O, T \rangle, o)$
$Sort(to\_translate, subgraphs)$
for $o \in to\_translate$ do
    $\langle O', T' \rangle = o.formula.translator(extract\_local(\langle O, T \rangle, o), o)$
    $O = O \cup O'$
    $T = T \cup T'$
return $\langle O, T \rangle$
```

The *explain_translate* algorithm sorts the obligations of *to_translate* according to the sub-graph relation to ensure that a graph $E_1$ included into another graph $E_2$ is translated before $E_2$. Thanks to this sorting, the designer, that defines local translators, is ensured that the local translator attached to some formula will be run after the local translators that translate parts of its sub-graphs, and thus can rely on this translation.

For instance, if we attach a local translator to the $\langle\!\langle\rangle\!\rangle\mathbf{X}$ alias to add an edge in the graph, such as the one added in Figure 10.7, a local translator attached to the $\langle\!\langle\rangle\!\rangle\mathbf{G}$ alias is sure that this extra edge will be present in the sub-graph it will receive, as the algorithm ensures that the $\langle\!\langle\rangle\!\rangle\mathbf{X}$ local translator will be run first.

Nevertheless, this mechanism must be used with caution because, if two sub-graphs are not included in one another, then the algorithm does not guarantee any order of their translation. Furthermore, because the algorithm extracts the sub-graph to give to a local translator from the current translated explanation—and not from the original one—the second translator could receive a sub-graph with more elements than expected, if the first one has already been run and the corresponding sub-graphs share some elements. The designer must thus be aware of this particularity when defining the local translators.

### 10.2.5 Choosers and partial explanations

The previous functionalities help the designer to translate the µ-calculus explanation into another graph that is closer to the initial model language. The aliases replace µ-calculus formulas with their corresponding top-level logic formulas, the attributes attach more information to elements of the graph, and the algebra and local translators derive new graphs from the initial one. Nevertheless, the designer has no control on the initial explanation the algorithm produces. For instance, when explaining why a $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi$ formula is true is some state $q$, it can be the case that there are multiple viable actions for $\Gamma$. Nevertheless, the *explain* algorithm will make a choice among the successors $q_{a_\Gamma}$ of $q$, with no intervention from the designer.

To allow the designer to interfere into these choices, the framework provides the notion of *choosers*. A chooser is a function that takes as arguments an obligation, a set of choices—that is, a set of possible successors of this obligation—, and a type of choice, that depends on the top operator of the obligation formula. The set of choices given to a chooser are the choices that can effectively lead to an adequate explanation. For instance, for a $\Diamond_i\ \phi$ formula, the choices are restricted to the successors of the state that actually satisfy $\phi$. The chooser must then return a subset of the choices, depending on the choice type:

- *exclusive* choices are for $\vee$ and $\Diamond_i$ operators. In these cases, only *one* successor must be chosen among the possible ones, to ensure a consistent explanation of the satisfaction. The chooser still has the right to return no successor.

- *inclusive* choices are for $\wedge$ and $\Box_i$ operators. In these cases, the full explanation needs to show all successors. Nevertheless, the chooser can return only a *subset* of the successors.

- *meaningless* choices are for the other operators. In these cases, there is no meaningful choice: there is no successor for *true* formulas, atomic propositions or variables, and there is only one successor for least and greatest fixpoint formulas. When there is only one successor, the chooser can still ignore it.

Choosers can guide the explanation generation by choosing particular successors, but also limit the size of the generated explanation by only exploring parts of it. This introduces the notion of *partial explanations*, that is, explanations where some obligations are not fully explained because they lack some successors. The advantage of partial explanations is that the complete explanation can be too large to be generated. For instance, when explaining why the initial state of some Kripke structure satisfies the $CTL$ formula **AG** $p$, the explanation is the complete graph of reachable states. So getting a part of it is better than an explanation too large to be useful.

More precisely, a partial explanation is a couple $E^p = \langle E, U\rangle$ where $E = \langle O, T\rangle$ is a (non-consistent) explanation and $U \subseteq O$ is the set of unexplained obligations of $E$. Let $E = \langle O, T\rangle$ be an explanation, we say that $o = \langle q, \phi, e\rangle \in O$ is *partially consistent in $E$* if and only if

- $\phi \neq false$,
- if $\phi = true$, then $succ(o) = \varnothing$;
- if $\phi = p$ or $\phi = \neg p$, for an atomic proposition $p$, then $succ(o) = \varnothing$;
- if $\phi = v$, for a variable $v$, then $q \in e(v)$ and $succ(o) = \varnothing$;
- if $\phi = \neg v$, then $q \notin e(v)$ and $succ(o) = \varnothing$;
- if $\phi = \phi_1 \wedge \phi_2$ then $succ(o) \subseteq \{o_1, o_2\}$, where $o_1 = \langle q, \phi_1, e\rangle$ and $o_2 = \langle q, \phi_2, e\rangle$;
- if $\phi = \phi_1 \vee \phi_2$ then $succ(o) \subseteq \{o_j\}$, where $o_j = \langle q, \phi_j, e\rangle$ for some $j \in \{1, 2\}$;
- if $\phi = \Diamond_i \phi'$ then $succ(o) \subseteq \{o'\}$, where $o' = \langle q', \phi', e\rangle$ for some state $q'$;
- if $\phi = \Box_i \phi'$ then for all $o' \in succ(o)$, $o' = \langle q', \phi', e\rangle$ for some state $q'$;
- if $\phi = \mu v.\psi$, then $succ(o) \subseteq \{o'\}$ where $o' = \langle q, \psi^k(false), e\rangle$ for some $k \geq 0$;
- if $\phi = \nu v.\psi$, then $succ(o) \subseteq \{o'\}$ where $o' = \langle q, \psi(\phi), e\rangle$.

Intuitively, $o$ is partially consistent in $E$ if it has at most the successors it should have to be locally consistent in $E$. A *consistent partial explanation* is a partial explanation $E^p = \langle\langle O, T\rangle, U\rangle$ such that all obligations $o \in U$ are *partially consistent* in $\langle O, T\rangle$, and all obligations in $O\backslash U$ are *locally consistent* in $\langle O, T\rangle$.

Partial explanations are linked to Kripke structures as standard explanations are. Given a structure $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$, a partial explanation $\langle\langle O, T, \rangle, U\rangle$ *partially matches* $S$ if and only if

1. for all $\langle q', \phi', e'\rangle \in O$, $q' \in Q$;
2. for all $\langle q', p, e'\rangle \in O$, $p \in V(q)$ and for all $\langle q', \neg p, e'\rangle \in O$, $p \notin V(q)$, for an atomic proposition $p$;
3. for all $\langle\langle q', \phi', e'\rangle, \langle q'', \phi'', e''\rangle\rangle \in T$, $q' = q''$ or $\phi' \in \{\Diamond_i\phi'', \Box_i\phi''\}$ and $\langle q', q''\rangle \in R_i$;
4. for all $o' = \langle q', \Box_i\phi', e'\rangle \in O$,
$$\begin{aligned}&\exists o'' \in \mathit{succ}(o') \text{ s.t. } o'' = \langle q'', \phi'', e''\rangle \text{ for some } \phi'', e''\\ &\implies \langle q', q''\rangle \in R_i.\end{aligned}$$

The only difference with the *matches* relation is that not all successors of $\Box_i$ obligations have to appear.

Finally, given a Kripke structure $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$, a state $q \in Q$, a µ-calculus formula $\phi$, and an environment $e$, we say that a *partial explanation* $E^p = \langle\langle O, T\rangle, U\rangle$ *is adequate for* $q \in [\![\phi]\!]^S e$ if and only if $E^p$ is consistent, $E^p$ partially matches $S$, and $\langle q, \phi, e\rangle \in O$.

As for attributors, choosers can be attached to formulas. Attaching a chooser to a formula $\phi$ allows the algorithms below to use them only when choosing the successors of the obligations with $\phi$. For instance, it is possible to choose the successors of $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}$ formulas without running the chooser on all the other cases.

Finally, choosers are allowed to avoid making a choice: by returning a special *none* value, a chooser tells that it does not want to make that particular choice, and lets the next chooser to make the choice instead. This allows choosers to be chained.

The algorithms of the previous sections can be updated to take choosers and partial explanations into account. The *explain* algorithm must be updated to (1) ask choosers to make choices, and (2) keep track of the unexplained obligations. Algorithm 10.5 presents this new version. It behaves similarly to the original one, but it computes all valid successors, instead of picking one, and runs them through the available choosers—the ones attached to the formula, and the ones given as argument (Lines 22 to 25). Furthermore, it keeps track of unexplained obligations in an additional $U$ local variable (Line 29). The algorithm uses the $ChoiceType$ function returning $inclusive$, $exclusive$ or $meaningless$, depending on the choice type of the top operator of the given formula.

The *explain_with_attr* algorithm stays almost the same. The only difference is that it calls the *explain*$^p$ algorithm instead of the *explain* one, and returns a partial explanation instead of a standard one. The new version is not given here, but is called *explain*$^p$*_with_attr* in the sequel.

Local translation, on the other hand, handles several obligations and edges of the explanation at the same time. Thus, it needs to be sure that the part that explains a given aliased formula is completely generated before translating it. For achieving this, the *extract_local* algorithm is updated to return the *frontier* of the sub-graph, that is, the nodes that contain a sub-formula of the alias under consideration. Thanks to this frontier, the *explain_translate* algorithm can translate only the sub-graphs for which there are no unexplained obligations outside their frontier.

More precisely, the updated *extract_local* algorithm is given in Algorithm 10.6. The main difference with the original algorithm is given at Line 11, where it accumulates in the $frontier$ set the obligations with a sub-formula of the alias of the main obligation $o$.

The updated *explain_translate* algorithm uses the frontier of sub-graphs explaining aliases to be sure that the alias is completely explained before translating it. More precisely, Algorithm 10.7 presents the new version. The main difference is at Line 7, where it ignores the obligations for which the sub-graph still contains some unexplained obligations outside their frontier.

Finally, providing partial explanations allows the designer to build smaller but incomplete explanations. This also allows the designer to provide interactive generation of explanations, in which the user chooses which part of the explanation must be generated at the next step. This functionality needs thus a way to expand a partial explanation with new obligations explaining the parts left unexplained.

Algorithm 10.8 presents the *expand*$^p$ algorithm that takes a partial explanation $E^p = \langle\langle O, T\rangle, U\rangle$ and an unexplained obligation $o \in U$ in addition to the arguments of the *explain*$^p$ algorithm, and returns a partial explanation extending $E^p$. The difference between the *expand*$^p$ and *explain*$^p$ algorithms is that the latter starts with an empty partial explanation—$O = T = U = \emptyset$—,while the former starts with the partial explanation given as argument.

**Algorithm 10.5:** *explain*$^p(S, q, \phi, e, choosers)$

```
Data: $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$ a Kripke structure, $q \in Q$ a state of
      $S$, $\phi$ a µ-calculus formula, and $e$ an environment such that
      $q \in [\![\phi]\!]^S e$, and $choosers$ a list of choosers.
Result: An adequate partial explanation for $q \in [\![\phi]\!]^S e$.

$O = \varnothing$; $T = \varnothing$; $U = \varnothing$
$pending = \{\langle q, \phi, e\rangle\}$
while $pending \neq \varnothing$ do
    pick $o' = \langle q', \phi', e'\rangle \in pending$
    $pending = pending \backslash \{o'\}$
    $O = O \cup \{o'\}$
    case $\phi' \in \{true, p, \neg p, v, \neg v\}$:  $O' = \varnothing$
    case $\phi' = \phi_1 \wedge \phi_2$:  $O' = \{\langle q', \phi_1, e'\rangle, \langle q', \phi_2, e'\rangle\}$
    case $\phi' = \phi_1 \vee \phi_2$
        $O' = \{\}$
        if $q' \in [\![\phi_1]\!]^S e'$ then  $O' = O' \cup \{\langle q', \phi_1, e'\rangle\}$
        if $q' \in [\![\phi_2]\!]^S e'$ then  $O' = O' \cup \{\langle q', \phi_2, e'\rangle\}$
    case $\phi' = \Diamond_i\ \phi''$
        $O' = \{\langle q'', \phi'', e'\rangle \mid \langle q', q''\rangle \in R_i \wedge q'' \in [\![\phi'']\!]^S e'\}$
    case $\phi' = \Box_i\ \phi''$:  $O' = \{\langle q'', \phi'', e'\rangle \mid \langle q', q''\rangle \in R_i\}$
    case $\phi' = \mu v.\ \psi$
        $\phi'' = false$; $sat = [\![\phi'']\!]^S e'$
        while $q' \notin sat$ do
            $\phi'' = \psi(\phi'')$; $sat = [\![\phi'']\!]^S e'$
        $O' = \{\langle q', \phi'', e'\rangle\}$
    case $\phi' = \nu v.\ \psi$:  $O' = \{\langle q', \psi(\phi'), e'\rangle\}$
22  $chrs = \phi'.choosers + choosers$
    for $chooser \in chrs$ do
        $new\_O' = chooser(o', O', ChoiceType(\phi'))$
25      if $new\_O' \neq none$ then  break
    if $new\_O' = none$ then $new\_O' = O'$
    $T = T \cup \{\langle o', o''\rangle \mid o'' \in new\_O'\}$
    $pending = pending \cup (new\_O' \backslash O)$
29  if $new\_O' \subsetneq O'$ then  $U = U \cup \{o'\}$
return $\langle\langle O, T\rangle, U\rangle$
```

The *expand*$^p$ algorithm needs the original partial explanation to work, that is, the partial explanation before running attributors and local

**Algorithm 10.6:** *extract_local*$^p(E, o)$

```
Data: $E = \langle O, T\rangle$ an explanation, and $o \in O$ an obligation such
      that $o.formula$ is aliased with $\alpha(\phi_1, ..., \phi_n)$.
Result: The part of $E$ explaining $\alpha(\phi_1, ..., \phi_n)$, and the frontier of
        this part.
$O' = \varnothing$; $T' = \varnothing$
$frontier = \varnothing$
$pending = \{o\}$
while $pending \neq \varnothing$ do
    pick $o' \in pending$
    $pending = pending \backslash \{o'\}$
    $O' = O' \cup \{o'\}$
    if $o'.formula \notin \{\phi_1, ..., \phi_n\}$ then
        $T' = T' \cup \{\langle origin, edge, end\rangle \in T \mid origin = o'\}$
        $pending = pending \cup (succ(o') \backslash O')$
11  else $frontier = frontier \cup \{o'\}$
return $\langle\langle O', T'\rangle, frontier\rangle$
```

translators on it. This means that the algorithms *explain*$^p$*_with_attr* and *explain*$^p$*_translate* have be updated to keep track and return the original partial explanation beside the translated one. This is a technicality not developed here, but it is not difficult to isolate the new elements of the partial explanation, to run attributors onto them only, and to isolate newly fully explained parts of the extended explanation that must be locally translated.

### 10.2.6 Markers

Section 10.2.2 showed the translation of µ-calculus explanations into *ATL* explanations using the relational graph algebra. This translation includes the step of keeping, in the explanation, the formulas that are interesting for the designer. Furthermore, the previous section discussed the notion of choosers, and how they can be used to produce partial explanations, with obligations that are kept unexplained.

To further facilitate these two cases, the framework provides the notion of *markers*. A marker is attached to a formula to add extra information to it. The framework provides two types of markers, *points of interest*, and *points of decision*, but new types can be defined by the designer.

Points of interest are intended to mark the formulas that are im-

**Algorithm 10.7:** $explain^p\_translate(S, q, \phi, e, attributors)$

```
Data: $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$ a Kripke structure, $q \in Q$ a state of
      $S$, $\phi$ a µ-calculus formula, and $e$ an environment such that
      $q \in [\![\phi]\!]^S e$, and $attributors$ a list of attributors.
Result: An adequate locally translated partial explanation for
      $q \in [\![\phi]\!]^S e$ augmented with attributes.
$\langle\langle O, T\rangle, U\rangle = explain^p\_with\_attr(S, q, \phi, e, attributors)$
$to\_translate = \langle o \in O \mid o.formula$ has a local translator$\rangle$
$subgraphs$ = empty dictionary
$explained = \langle\rangle$
for $o \in to\_translate$ do
    $\langle\langle O', T'\rangle, frontier\rangle = extract\_local^p(\langle O, T\rangle, o)$
7   if $U \cap (O' \setminus frontier) = \varnothing$ then
        $subgraphs[o] = \langle O', T'\rangle$
        $explained = explained + \langle o\rangle$
$Sort(explained, subgraphs)$
for $o \in explained$ do
    $\langle O', T'\rangle = o.formula.translator(extract\_local^p(\langle O, T\rangle, o), o)$
    $O = O \cup O'$
    $T = T \cup T'$
return $\langle O, T\rangle$
```

portant for the designer. The *important* function of Equation 10.2 in Section 10.2.2 can thus be defined as

$$important(o) = POI \in o.formula.markers,$$

where $o.formula.markers$ is the set of markers attached to the formula of the $o$ obligation, and $POI$ is the point of interest marker.

While points of interest simply give some flexibility and simplicity to the designer, points of decision (PODs, in short) take part to the generation process itself. The purpose of these markers is to avoid explaining an obligation if its formula is marked with a POD. So the $explain^p$ and $expand^p$ algorithms have to take them into account.

Algorithm 10.9 updates the $explain^p$ algorithm to take PODs into account. The difference is at Line 21, where the algorithm exposes no successor of the $o'$ obligation if it contains a formula with a POD.

The $expand^p$ algorithm is updated similarly. In this case, the $o$ obligation given as argument has to be explained—that is, run through the choosers—, even if its formula is marked with a POD. Otherwise, the obligations with POD formulas could not be explained at all.

**Algorithm 10.8:** $expand^p(S, q, \phi, e, E^p, o, choosers)$

**Data**: $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$ a Kripke structure, $q \in Q$ a state of $S$, $\phi$ a μ-calculus formula, and $e$ an environment such that $q \in [\![\phi]\!]^S e$, $E^p = \langle\langle O, T\rangle, U\rangle$ a partial explanation adequate for $q \in [\![\phi]\!]^S e$, $o \in U$ an unexplained obligation, and $choosers$ a list of choosers.

**Result**: An adequate partial explanation for $q \in [\![\phi]\!]^S e$ extending $E^p$.

```
$pending = \{o\}$
while $pending \neq \varnothing$ do
    pick $o' = \langle q', \phi', e'\rangle \in pending$
    $pending = pending \setminus \{o'\}$
    $O = O \cup \{o'\}$
    case $\phi' \in \{true, p, \neg p, v, \neg v\}$: $O' = \varnothing$
    case $\phi' = \phi_1 \wedge \phi_2$: $O' = \{\langle q', \phi_1, e'\rangle, \langle q', \phi_2, e'\rangle\}$
    case $\phi' = \phi_1 \vee \phi_2$
        $O' = \{\}$
        if $q' \in [\![\phi_1]\!]^S e'$ then $O' = O' \cup \{\langle q', \phi_1, e'\rangle\}$
        if $q' \in [\![\phi_2]\!]^S e'$ then $O' = O' \cup \{\langle q', \phi_2, e'\rangle\}$
    case $\phi' = \Diamond_i\ \phi''$
        $O' = \{\langle q'', \phi'', e'\rangle \mid \langle q', q''\rangle \in R_i \wedge q'' \in [\![\phi'']\!]^S e'\}$
    case $\phi' = \Box_i\ \phi''$: $O' = \{\langle q'', \phi'', e'\rangle \mid \langle q', q''\rangle \in R_i\}$
    case $\phi' = \mu v.\ \psi$
        $\phi'' = false$; $sat = [\![\phi'']\!]^S e'$
        while $q' \notin sat$ do
            $\phi'' = \psi(\phi'')$; $sat = [\![\phi'']\!]^S e'$
        $O' = \{\langle q', \phi'', e'\rangle\}$
    case $\phi' = \nu v.\ \psi$: $O' = \{\langle q', \psi(\phi'), e'\rangle\}$
    $chrs = \phi'.choosers + choosers$
    for $chooser \in chrs$ do
        $new\_O' = chooser(o', O', ChoiceType(\phi'))$
        if $new\_O' \neq none$ then break
    if $new\_O' = none$ then $new\_O' = O'$
    $T = T \cup \{\langle o', o''\rangle \mid o' \in new\_O'\}$
    $pending = pending \cup (new\_O' \setminus O)$
    if $new\_O' \subsetneq O'$ then $U = U \cup \{o'\}$
return $\langle\langle O, T\rangle, U\rangle$
```

**Algorithm 10.9:** $explain^{pod}(S, q, \phi, e, choosers)$

**Data**: $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$ a Kripke structure, $q \in Q$ a state of $S$, $\phi$ a µ-calculus formula, and $e$ an environment such that $q \in [\![\phi]\!]^S e$, and *choosers* a list of choosers.

**Result**: An adequate partial explanation for $q \in [\![\phi]\!]^S e$.

```
$O = \varnothing$; $T = \varnothing$; $U = \varnothing$
$pending = \{\langle q, \phi, e\rangle\}$
while $pending \neq \varnothing$ do
    pick $o' = \langle q', \phi', e'\rangle \in pending$
    $pending = pending \backslash \{o'\}$
    $O = O \cup \{o'\}$
    case $\phi' \in \{true, p, \neg p, v, \neg v\}$:  $O' = \varnothing$
    case $\phi' = \phi_1 \wedge \phi_2$:  $O' = \{\langle q', \phi_1, e'\rangle, \langle q', \phi_2, e'\rangle\}$
    case $\phi' = \phi_1 \vee \phi_2$
        $O' = \{\}$
        if $q' \in [\![\phi_1]\!]^S e'$ then  $O' = O' \cup \{\langle q', \phi_1, e'\rangle\}$
        if $q' \in [\![\phi_2]\!]^S e'$ then  $O' = O' \cup \{\langle q', \phi_2, e'\rangle\}$
    case $\phi' = \Diamond_i\ \phi''$
        $O' = \{\langle q'', \phi'', e'\rangle \mid \langle q', q''\rangle \in R_i \wedge q'' \in [\![\phi'']\!]^S e'\}$
    case $\phi' = \Box_i\ \phi''$:  $O' = \{\langle q'', \phi'', e'\rangle \mid \langle q', q''\rangle \in R_i\}$
    case $\phi' = \mu v.\ \psi$
        $\phi'' = false$; $sat = [\![\phi'']\!]^S e'$
        while $q' \notin sat$ do  $\phi'' = \psi(\phi'')$; $sat = [\![\phi'']\!]^S e'$
        $O' = \{\langle q', \phi'', e'\rangle\}$
    case $\phi' = \nu v.\ \psi$:  $O' = \{\langle q', \psi(\phi'), e'\rangle\}$
21  if $POD \in \phi'.markers$ then  $new\_O' = \varnothing$
    else
        $chrs = \phi'.choosers + choosers$
        for $chooser \in chrs$ do
            $new\_O' = chooser(o', O', ChoiceType(\phi'))$
            if $new\_O' \neq none$ then  break
    if $new\_O' = none$ then $new\_O' = O'$
    $T = T \cup \{\langle o', o''\rangle \mid o'' \in new\_O'\}$
    $pending = pending \cup (new\_O' \backslash O)$
    if $new\_O' \subsetneq O'$ then  $U = U \cup \{o'\}$
return $\langle\langle O, T\rangle, U\rangle$
```

Figure 10.10 gives a summary of the call graph for the transformation of a (potentially partial) µ-calculus explanation into a locally translated explanation with attributes. The explanation is generated with $\mathit{explain}^{pod}$, then attributes are added by $\mathit{explain}^{p}\_\mathit{with\_attr}$, and subgraphs are eventually extracted with $\mathit{extract}^{p}\_\mathit{local}$ and locally translated by $\mathit{explain}^{p}\_\mathit{translate}$. The last one is thus the entry point to generate such enriched explanations.

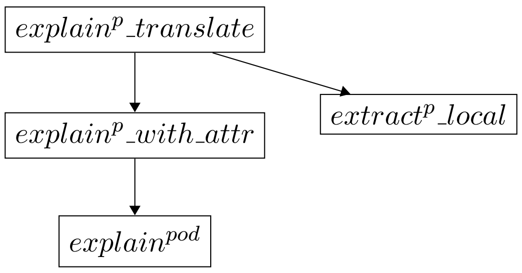


Figure 10.10: The translation call graph.

Finally, markers are attached to formulas. A drawback appears when dealing with substitution: when explaining an obligation $o$ with a greatest fixpoint formula $\phi$, some descendants of $o$ are labelled with $\phi$, too. This means that, if $\phi$ is marked with a point of decision, the generation will stop at all the descendants of $o$ with $\phi$. This can be useful to precisely control the generation, but it also can be annoying, for instance, when explaining why $\mathbf{EG}\ \phi'$ is true: the generation is stopped at every step of the path showing $\mathbf{G}\ \phi'$.

To solve this problem, the framework provides two kinds of markers: the standard markers are attached to all occurrences of the same greatest fixpoint formula, and the *simple markers* are not attached to the copy of the formula in the substitution. The framework provides standard and simple POIs, as well as standard and simple PODs. Thanks to simple PODs, the problem above is avoided: as the marker is not attached to the descendants of $o$, the generation is not stopped, and the explanation of $\mathbf{EG}\ \phi'$ directly gives the full path.

## 10.3 Implementation

The framework has been implemented in Python, using PyNuSMV for solving the model-checking problem. This section briefly describes how all the parts of the framework are implemented. It also presents a tool to visualize and manipulate translated explanations.

### 10.3.1 Encoding the model

To be able to use the framework, the designer has to derive, from the original model, a µ-calculus Kripke structure $S = \langle Q, \{R_i \mid i \in \Sigma\}, V \rangle$. Such a structure is implemented with PyNuSMV as a standard SMV model to which several transition relations $R_i$ are attached.

The framework provides two functionalities for defining and attaching several transition relations. The first functionality relies on the SMV model itself: if the SMV model contains an input variable called `transition`, its values are taken as the different transition relation names, and when dealing with a transition relation named `trans`, the framework uses the standard model transition relation restricted by the `transition = trans` constraint.

For instance, a toy µ-calculus model representing a simple counter ranging from 0 to 3 is given in Figure 10.11. The corresponding µ-calculus model contains the two transition relations named `inc` and `dec`.

```
MODULE main
  VAR counter: 0..3;
  IVAR transition: {inc, dec};

  INIT counter = 0
  TRANS transition = inc ->
        (next(counter) = (counter + 1) mod 4)
  TRANS transition = dec ->
        (next(counter) = (counter + 3) mod 4)
```

Figure 10.11: A NuSMV model encoding the µ-calculus structure of a simple counter.

The second functionality allows the designer to provide, when building the µ-calculus model, an additional set of named transition relations defined using PyNuSMV features. For instance, she can remove the two `TRANS` clauses of Figure 10.11 and declare them in Python (assuming that the model is stored in `model.smv`):

```
pynusmv.glob.load("model.smv")
transitions = {
  'inc': pynusmv.fsm.BddTrans.from_string(
          "next(counter) = (counter + 1) mod 4"),
  'dec': pynusmv.fsm.BddTrans.from_string(
          "next(counter) = (counter + 3) mod 4")
}
muModel = bddModel(transitions)
```

### 10.3.2 Defining µ-calculus formulas

The framework provides Python classes to define µ-calculus formulas, one for each µ-calculus operator: `MTrue`, `MFalse`, `Atom`, `Variable`, `Not`, `And`, `Or`, `Diamond`, `Box`, `Mu`, and `Nu`. With this implementation µ-calculus formulas do not have to be declared in positive normal form. Instead, the framework lazily derives positive normal forms when needed. This allows the formulas that annotate the obligations to stay as close to the main formula as possible.

### 10.3.3 Implementation of translation features

This section briefly discusses how the different features to translate explanations back into the top-level language are provided by the implementation.

Most of the features are implemented with Python *decorators*, that is, function annotations that change the function behavior. Aliases are defined as Python functions returning the corresponding µ-calculus formula and decorated with the `@alias` decorator. Alias negations are defined similarly, by redefining the aliased function `f` and decorating it with the `@f.negation` decorator. Aliases $\alpha(a_1, ..., a_n)$ are printed, by default, as `alpha(a_1, ..., a_n)`, where `a_i` are replaced by the actual arguments of the alias. This can be overridden by providing a Python format string as an argument to the alias.

For instance, we can define the alias $IFF(\phi_1, \phi_2)$ as

```
@alias("{f1} <==> {f2}")
def IFF(f1, f2):
  return Or(And(f1, f2), And(Not(f1), Not(f2)))
```

and define its negation as

```
@IFF.negation
def IFF(f1, f2):
  return And(Or(Not(f1), f2), Or(f1, Not(f2)))
```

The `@alias` argument `"{f1} <==> {f2}"` tells the framework to print `IFF(a, b)` as `"a <==> b"` instead of `"IFF(a, b)"`.

Given an alias, the designer can define a local translator in the same way as she defines the negation of the alias: she decorates the function `f` taking the sub-graph and obligation as arguments with the `@f.translation` decorator. The decorated function must return a new graph representing the translated sub-graph.

Finally, choosers are also defined as functions decorated with the `@chooser` decorator. They take an obligation, a set of successor obligations, and a choice type as arguments, and return a subset of the successors, depending on the choice type.

Markers are instances of the `Marker` class. The four markers provided by the framework are implemented by the `POD` and `SPOD` instances for standard and simple points of decision, and by the `POI` and `SPOI` instances for the points of interest.

Relational graphs, and generated explanations in particular, are implemented with the `Graph` class. Nodes and edges of these graphs are implemented with the `domaintuple` class. This class defines a dictionary-like structure where domains of the elements are identified by a name. For instance, bare explanation nodes are instances of `domaintuple` with three keys: `"state"` contains the state of the obligation, `"formula"` is the formula of the obligation, and `"context"` is its context.

The designer can manipulate `Graph` instances through their methods: the `extension` method derives a new graph extending the instance—it implements the $\epsilon$ operator of the algebra—, the `projection` method returns a new graph with projected elements, etc. Each operator of the relational graph algebra is implemented by a method of the `Graph` class.

Finally, attributors are also defined with decorated Python functions. These functions take either an obligation as argument and are decorated with the `@obligation_attributor` decorator, or take an edge as argument—a triplet composed of an obligation, one edge information, and another obligation—, and are decorated with the `@edge_attributor` decorator. They then return a Python dictionary of new attributes to add to the obligation or edge. These attributors can then be attached to formulas, or directly given to the *explain* algorithm.

For instance, we can define an obligation attributor that extends obligations with the left sub-formula of its formula as

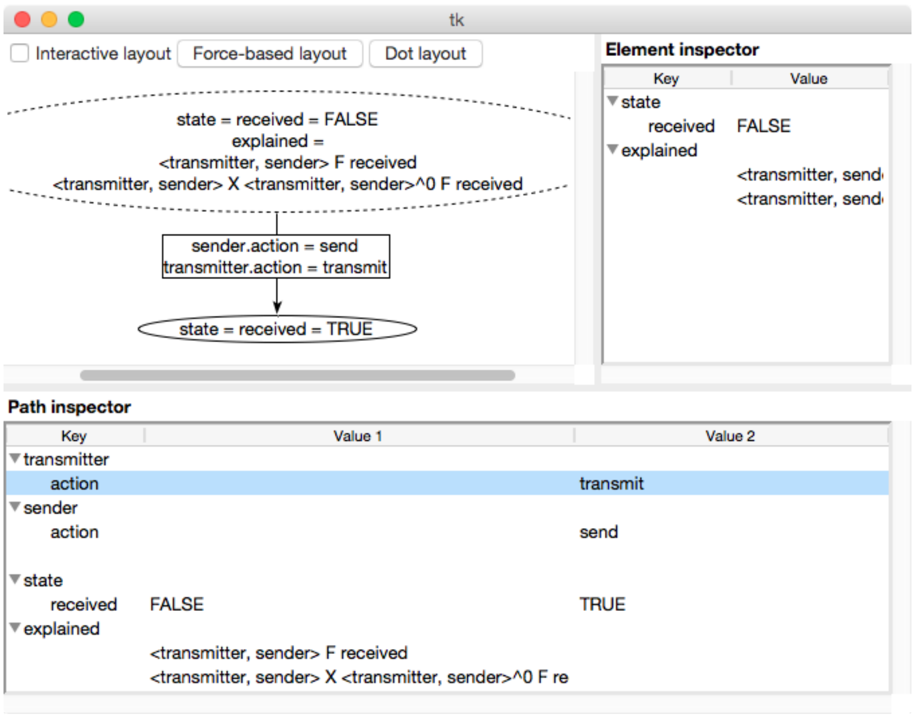
```
@obligation_attributor
left_sub(obligation):
  return {"left": obligation["formula"].left}
```

and attach it to the `IFF` alias as

```
@alias("{f1} <==> {f2}")
def IFF(f1, f2):
  return left_sub(Or(And(f1, f2),
                     And(Not(f1), Not(f2))))
```

### 10.3.4 Visualization tool

The framework allows the designer to efficiently translate an explanation back into the top-level language. Nevertheless, these explanations remain complex and difficult to understand. To help the user in understanding these complex explanations, the implementation also provides a graphical visualization tool. A snapshot of the tool is given in Figure 10.12.



Element inspector

| Key | Value |
|---|---|
| ▼state | |
| received | FALSE |
| ▼explained | |
| | <transmitter, send |
| | <transmitter, send |

Path inspector

| Key | Value 1 | Value 2 |
|---|---|---|
| ▼transmitter | | |
| action | | transmit |
| ▼sender | | |
| action | | send |
| ▼state | | |
| received | FALSE | TRUE |
| ▼explained | | |
| | <transmitter, sender> F received | |
| | <transmitter, sender> X <transmitter, sender>^0 F re | |

Figure 10.12: A snapshot of the visualization tool.

The main part of the tool (top left part) presents the explanation as a graph. Nodes are depicted in ovals, and edges are depicted as arrows decorated with information in a box. This graph can be moved with the mouse, allowing the user to re-arrange it as she wishes. Additionally, the tool provides two ways to automatically re-arrange the graph through the *Force-based layout* and *Dot layout* buttons.

The information displayed in nodes and edge labels come from the explanation elements themselves. These elements are `domaintuple` instances, that is, dictionary-like structures, so the tool displays in nodes and edges a representation of these dictionaries as `key = value` strings. More precisely, the tool displays only the keys not starting with `_`, allowing the designer to add extra information to the explanation without polluting the displayed graph. This behavior can be overridden: if the graph element contains a `"_label_"` key, its value is used instead of the whole dictionary. This mechanism allows the designer to freely choose what will be rendered.

If a `"_label_"` key is not provided, the tool also allows the user to select which keys of the graph elements are displayed, through a right-click menu on the graph area. This functionality is useful when the complete information is too large to be displayed in nodes or edges, or when some keys are useless for the user.

To enable interactivity, the designer can define, for any node or edge, the `"_menu_"` key. If this key exists for some element, it is assumed to be a graphical menu that is displayed whenever the user right-clicks on the element. This can be used, for instance, to expand partial explanations.

The top right part of the tool displays the so-called *element inspector*. When the user clicks on a particular graph element (the dashed one on Figure 10.12), the tool displays in this window the complete information of the element, as a table, where keys are displayed in the left column and values in the right one. This window allows the user to quickly get the complete information of some graph element without having to re-display everything on the whole graph.

Again, the designer can decide what is effectively displayed in the element inspector. By default, all keys not starting with `_` are displayed but, if the element contains the `"_view_"` key, it is used instead to fill the window.

Finally, the bottom part of the tool can display one particular path of the graph. More precisely, the user can select a path in the graph by dragging and dropping the cursor from one node to another. In this case, the shortest path from the first node to the second one is displayed. The user can also add or remove elements from the selection to include or exclude them from the displayed path. When the set of selected elements effectively represents a finite path through the graph, the *path inspector* displays the information of each node and edge in a table. As for the element inspector, the keys of the dictionaries are given in the left column, and the other columns display the information of the edges and nodes along the path. The path inspector uses the same mechanism as the element inspector, so the displayed information can be overridden by the designer by providing a `"_view_"` key.

## 10.4 Application to $ATL$

The objective of this section is to show the usefulness of the framework by applying it to the full $ATL$ logic. It describes how explanations for $ATL$ can be obtained, displayed and manipulated thanks to the framework implementation.

First, the full translation of $ATL$ structures and formulas into μ-calculus ones is given, detailing the intuition given at the beginning of the chapter. This formal translation is completed with a brief description of its Python implementation. Second, the section presents the functionalities used to translate μ-calculus explanations back into $ATL$ ones: aliases, attributors, choosers, markers and local translators, as well as relational graphs manipulations. The section also describes how the visualization tool presented in the previous section can be used to display, manipulate and extend partial $ATL$ explanations. Finally, the section draws some conclusions about the framework based on the presented $ATL$ model checker.

### 10.4.1 Translating $ATL$ structures

The intuition behind the translation of an $ATL$ CGS into a μ-calculus Kripke structure is to split the concurrent choices of the agents of the system into two steps: at the first step, the agents of Γ choose their action; at the second step, the other agents react to this choice, making the system evolve according to the chosen actions.

More precisely, let $S = \langle Ag, Q, Q_0, Act, e, \delta, V\rangle$ be a CGS. From $S$, we can derive a μ-calculus Kripke structure $T_{ATL}(S) = \langle Q', \{R'_i \mid i \in \Sigma\}, V'\rangle$ where

- $Q' = Q \cup \{q_{a_\Gamma} \mid \Gamma \subseteq Ag \wedge q \in Q \wedge a_\Gamma \in Act^\Gamma\}$, that is, $Q'$ is the set of states $Q$ augmented with states $q_{a_\Gamma}$ representing the fact that the group of agents Γ chose action $a_\Gamma$ in state $q$;
- the set of transition relations $\{R'_i \mid i \in \Sigma\}$ is the set

  $$\{R_{\Gamma choose} \mid \Gamma \subseteq Ag\} \cup \{R_{\Gamma follow} \mid \Gamma \subseteq Ag\},$$

  where

  $$R_{\Gamma choose} = \{\langle q, q_{a_\Gamma}\rangle \in Q' \times Q' \mid a_\Gamma \in Act^\Gamma\},$$

  and

  $$R_{\Gamma follow} = \left\{ \langle q_{a_\Gamma}, q'\rangle \in Q' \times Q' \,\middle|\, \begin{array}{l} a_\Gamma \in Act^\Gamma \wedge \exists a \in E_{Ag}(q) \\ \text{s.t. } a_\Gamma \sqsubseteq a \wedge q' = \delta(q, a) \end{array} \right\}.$$

  Intuitively, $R_{\Gamma choose}$ encodes the fact that Γ choose an action, and $R_{\Gamma follow}$ encodes the fact that the system evolves according to the choice of Γ;
- $V'(q) = V(q)$ for all $q \in Q$, and $V'(q_{a_\Gamma}) = \varnothing$ for all $q_{a_\Gamma} \in Q' \backslash Q$.

The implementation represents a CGS with a standard SMV model to which is attached a set of agents. Each agent is defined by a name and a set of SMV input variables corresponding to its actions. For instance, the CGS for the bit transmission problem is given in Figure 10.13. The two agents are the `sender`, controlling the `sender.action` input variable, and the `transmitter`, controlling the `transmitter.action` input variable. The transition relation of the SMV model defines the enabled actions of each agent in each state—in this case, all actions are always enabled—, as well as how the state of the system evolves according to the actions of the agents.

```
MODULE Sender()
  IVAR action : {send, wait};

MODULE Transmitter()
  IVAR action : {block, transmit};

MODULE main
  VAR sender : Sender();
      transmitter : Transmitter();
      sent : boolean;
  INIT !sent
  TRANS (next(sent) = (sender.action = send &
                       transmitter.action = transmit) ?
                      true : sent)
```

Figure 10.13: A NuSMV model encoding the CGS for the bit transmission problem.

As discussed in Section 10.3, the framework implementation provides two ways for defining μ-calculus models: the SMV model can include a `transition` input variable, or the designer can provide additional external transition relations. The implementation of the CGS translation uses the second option and provides, with the SMV model defining the CGS, a Python instance acting like a dictionary of transition relations, and lazily building these relations when needed. The advantage of this mechanism is that, even if the CGS contains a lot of agents—resulting into an exponentially large set of μ-calculus transition relations, since two transition relations must be defined for each subset of agents—its implementation builds the transition relations that are actually needed, that is, for the groups of agents appearing in the checked formula.

### 10.4.2 Translating $ATL$ formulas

$ATL$ formulas can be translated into µ-calculus ones such that the $ATL$ formula $\phi$ is satisfied by state $q$ from some CGS $S$ if and only if the µ-calculus formula corresponding to $\phi$ is satisfied by the same state in $T_{ATL}(S)$. The intuition behind the translation is that the $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}$ and $[\![\Gamma]\!]\mathbf{X}$ operators can be translated into µ-calculus formulas involving $\Box_i$ and $\Diamond_i$ operators, and the other strategic operators are translated into greatest and least fixpoints over formulas involving these operators.

More formally, let $\phi$ be an $ATL$ formula. $T_{ATL}(\phi)$ is defined as

$$
\begin{aligned}
&T_{ATL}(p) = p,\\
&T_{ATL}(\neg\phi) = \neg T_{ATL}(\phi),\\
&T_{ATL}(\phi_1 \vee \phi_2) = T_{ATL}(\phi_1) \vee T_{ATL}(\phi_2),\\
&T_{ATL}(\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi) = \Diamond_{\Gamma choose}\ (\Diamond_{\Gamma follow}\ true \wedge \Box_{\Gamma follow}\ T_{ATL}(\phi)),\\
&T_{ATL}(\langle\!\langle\Gamma\rangle\!\rangle[\phi_1\ \mathbf{U}\ \phi_2]) =\\
&\mu v.\ T_{ATL}(\phi_2) \vee (T_{ATL}(\phi_1) \wedge \Diamond_{\Gamma choose}\ (\Diamond_{\Gamma follow}\ true \wedge \Box_{\Gamma follow}\ v)),\\
&T_{ATL}(\langle\!\langle\Gamma\rangle\!\rangle[\phi_1\ \mathbf{W}\ \phi_2]) =\\
&\nu v.\ T_{ATL}(\phi_2) \vee (T_{ATL}(\phi_1) \wedge \Diamond_{\Gamma choose}\ (\Diamond_{\Gamma follow}\ true \wedge \Box_{\Gamma follow}\ v)).
\end{aligned}
$$

The translation for the other propositional operators—$\wedge$, $\implies$, $\iff$—and strategic ones—$\langle\!\langle\Gamma\rangle\!\rangle\mathbf{F}$, $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{G}$, and their $[\![\Gamma]\!]$ counterparts—is easily derived by syntactic reduction to the ones above.

The implementation of $ATL$ formula translation simply uses the Python classes provided by the framework to define µ-calculus formulas. For instance, the µ-calculus formula

$$\nu v.\ \neg sent \wedge \Diamond_{trans\ chooses}\ (\Diamond_{trans\ follows}\ true \wedge \Box_{trans\ follows}\ v)$$

corresponding to the $ATL$ formula $\langle\!\langle transmitter\rangle\!\rangle\mathbf{G}\ \neg sent$ is defined by

```
Nu(Variable("v"),
   And(Not(Atom("sent")),
       Diamond("trans chooses",
               And(Diamond("trans follows", MTrue()),
                   Box("trans follows",
                       Variable("v"))))))
```

The translation from CGS and $ATL$ formulas to µ-calculus structures and formulas is correct, in the sense that the model-checking outcome for one is the same as for the other. This is formally captured by the following property.

**Property 10.3.** *Given a CGS $S = \langle Ag, Q, Q_0, Act, e, \delta, V\rangle$, a state $q \in Q$, and an $ATL$ formula $\phi$, $S, q \models \phi$ if and only if $q \in [\![T_{ATL}(\phi)]\!]^{T_{ATL}(S)}\varnothing$.*

*Proof.* First, $T_{ATL}(\phi)$ is a closed formula by construction. Thus, the result of $[\![T_{ATL}(\phi)]\!]^{T_{ATL}(S)}e$ is the same for all environments $e$, and so for the empty environment $\varnothing$. Second, we can prove this property by showing that the function

$$Pre_{[\![\Gamma]\!]}(S, Q') = \left\{ q \in Q \;\middle|\; \begin{array}{l} \forall a_\Gamma \in E_\Gamma(q), \exists a \in E_{Ag}(q) \\ \text{s.t. } a_\Gamma \sqsubseteq a \wedge \delta(q,a) \in Q' \end{array} \right\}$$

defined in Equation 2.2 of Section 2.2.3 is equivalent to the µ-calculus formula $\Box_{\Gamma choose}\,(\Box_{\Gamma follow}\; false \vee \Diamond_{\Gamma follow}\; T_{ATL}(\phi'))$. Then the $eval_{ATL}$ algorithm of Section 2.2.3 corresponds to the evaluation of $T_{ATL}(\phi)$, and the translation is correct.

Let $Q' \subseteq Q$, we can show that

$$\begin{aligned} &Pre_{[\![\Gamma]\!]}(S, Q') = \\ &Q \cap [\![\Box_{\Gamma choose}\,(\Box_{\Gamma follow}\; false \vee \Diamond_{\Gamma follow}\; v)]\!]^{T_{ATL}(S)}\{v \to Q'\}. \end{aligned}$$

Let us first assume that $q$ belongs to the righthand side of this equality. In this case, $q$ belongs to $Q$ and for all successors $q_{a_\Gamma}$ of $q$ through $\Gamma_{choose}$, either $q_{a_\Gamma}$ belongs to $[\![\Box_{\Gamma follow}\; false]\!]^{T_{ATL}(S)}\{v \to Q'\}$, or $q$ belongs to $[\![\Diamond_{\Gamma follow}\; v]\!]^{T_{ATL}(S)}\{v \to Q'\}$. In the first case, $a_\Gamma$ is not enabled in $q$. Indeed, if $q_{a_\Gamma}$ has no successor through $R_{\Gamma follow}$, this means that there are no action $a \in E_{Ag}(q)$ and no state $q'$ such that $a_\Gamma \sqsubseteq a$ and $q' = \delta(q,a)$. In the second case, there exists a way to reach a state of $Q'$ through $a_\Gamma$. Indeed, as $q_{a_\Gamma}$ has a successor $q'$ in $Q'$, this means that there exists an action $a \in Act^{Ag}$ completing $a_\Gamma$ and leading to $q'$. Thus, $q$ belongs to the righthand side because for any action $a_\Gamma \in Act^\Gamma$, either $a_\Gamma$ is not enabled in $q$ or there exists a completing action leading to some state of $Q'$, that is, $q$ belongs to $Pre_{[\![\Gamma]\!]}(S, Q')$.

The other direction is similar. Let us assume that $q \in Pre_{[\![\Gamma]\!]}(S, Q')$. For all actions $a_\Gamma \in Act^\Gamma$, either $a_\Gamma$ is not enabled in $q$, or there exists a completing action $a \in Act^{Ag}$ leading to $Q'$. Thus, for all $a_\Gamma \in Act^\Gamma$, either $a_\Gamma$ is not enabled and $q_{a_\Gamma}$ belongs to

$$[\![\Box_{\Gamma follow}\; false]\!]^{T_{ATL}(S)}\{v \to Q'\},$$

or $a_\Gamma$ is enabled and there exists a successor of $q_{a_\Gamma}$ belonging to $Q'$, that is, $q_{a_\Gamma}$ belongs to $[\![\Diamond_{\Gamma follow}\; v]\!]^{T_{ATL}(S)}\{v \to Q'\}$. Thus, $q$ belongs to the righthand side of the equality, and the proof is done. □

### 10.4.3 Translating explanations

The previous sections showed that we can translate an *ATL* model-checking problem into an equivalent µ-calculus one. The µ-calculus framework can thus be used to solve the equivalent µ-calculus model-checking problem and produce rich adequate explanations. This section presents how the framework functionalities are used to translate µ-calculus explanations back into *ATL* ones.

## Aliases

First, aliases are declared for *ATL* operators by defining Python functions that return the corresponding µ-calculus formulas and are decorated with the `@alias` decorator. The implementation takes advantage of the format string we can give as argument to `@alias` to override its string representation. For instance, the alias for the $\langle\langle\Gamma\rangle\rangle\mathbf{X}\ \phi$ formula is defined with

```
@alias("<{agents}> X {formula}")
def CEX(agents, formula):
    return Diamond(agents + "_choose",
                   And(Diamond(agents + "_follow",
                               MTrue()),
                       Box(agents + "_follow",
                           formula)))
```

Furthermore, alias negations are defined to make the link between each *ATL* operator and its dual. For instance, the negation of the `CEX` alias is defined with

```
@CEX.negation
def CEX(agents, formula):
    return CAX(agents, Not(formula))
```

where `CAX` is the alias for the $[\![\Gamma]\!]\mathbf{X}\ \phi$ operator.

Finally, the `@alias` decorator also accepts the `bounded` argument, to override the way least fixpoint formulas are expanded. For instance, the formula pattern $\langle\langle\Gamma\rangle\rangle\mathbf{F}$ is defined with

```
@alias("<{agents}> F {formula}",
       "<{agents}>^{bound} F {formula}")
def CEF(agents, formula):
    var = Variable("CEF")
    return Mu(var, Or(formula, CEX(agents, var)))
```

where the second argument tells the framework that, when the least fixpoint operator is expanded, the string pattern used for the aliased formula should be the given one instead of the standard one.

### Markers

All top-level formulas returned by the defined aliases are marked as points of interest, and both `CEX` and `CAX` aliases are also marked as points of decision. The reason for the latter is to be able to generate small partial explanations and to allow the user to expand them as she wishes. For instance, the `CEG` alias marks the top-level formula as a point of interest:

```
@alias("<{agents}> G {formula}")
def CEG(agents, formula):
    var = Variable("CEG")
    return POI(Nu(var,And(formula, CEX(agents, var))))
```

We do not use a simple point of interest (`SPOI`) for the `CEG` and `CAG` aliases because we try to give as much information as possible, thus explaining that some state of the checked system satisfies $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{G}\ \phi$ because it satisfies both $\phi$ and $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \langle\!\langle\Gamma\rangle\!\rangle\mathbf{G}\ \phi$. With an `SPOI` instead of a `POI`, successors of a $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{G}\ \phi$ formula would not be annotated with $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{G}\ \phi$ in the final translated explanation.

### Attributors

Two attributors are defined to add information to edges and obligations of the explanation. The first one attaches, to each obligation, the original state its state derives from:

```
@obligation_attributor
def original_state(node):
    return {"original":
                node["state"].get_str_values()}
```

It relies on PyNuSMV functionalities to extract, from the BDD representing the state of the obligation—`node["state"]`—,a dictionary with SMV state variables and their corresponding value. This attributor is given to the *explain* algorithm to enrich all obligations.

The other attributor is an edge attributor. It stores the actions chosen by the group in the outgoing edge of the obligations labelled with a `CEX` or `CAX` aliased formula. This way, the information is more easily accessed by local translators. The technical extraction of the actions from the model is hidden behind the `actions_for_group` function.

```
@edge_attributor
def chosen_action(edge):
    origin, label, end = edge
    group = origin["formula"]["agents"]
    actions = actions_for_group(group, end)
    return {"action": actions}
```

This attributor is then attached to the CEX and CAX aliases:

```
@alias("[{agents}] X {formula}")
def CAX(agents, formula):
    return POD(POI(chosen_action(
           Box(agents + "_choose",
               Or(Box(agents + "_follow",
                      MFalse()),
                  Diamond(agents + "_follow",
                          formula)))
           )))
```

## Local translators

The implementation defines two local translators for $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}$ and $[\![\Gamma]\!]\mathbf{X}$. They extract, from the two steps of the µ-calculus model, the original one-step transitions of the CGS. The first one extracts from the sub-graph the action chosen by the group $\Gamma$ and links it to all its successors in the sub-graph, while the second one links the top-level node to all its successors through the different actions chosen by $\Gamma$. For instance, the local translator for the CAX alias is defined with

```
 1  @CAX.translation
 2  def CAX(graph, node):
 3      new_graph = {node: set()}
 4      agents = node["formula"]["agents"]
 5      for (edge, or_node) in graph[node]:
 6          actions = edge.action
 7          _, dia_node = next(iter(graph[or_node]))
 8          if isinstance(dia_node.formula, Diamond):
 9              _, succ = next(iter(graph[dia_node]))
10              new_graph[node] |= {(domaintuple(
11                                      inputs=actions)),
12                                      succ)}
13              new_graph[succ] = set()
14      return Graph(new_graph)
```

This translator gets the sub-graph explaining the alias and its root node as arguments, and, for each successor of this node, gets the action chosen by the group—actions, at Line 6. Then, for all the successors that have a $\Diamond_i$ labelled formula (Line 8), it adds a new edge to the graph with the actions as label (Lines 9 to 13). The translator for CEX works similarly: it adds edges from the root node to the frontier of the sub-graph, labelled with the action chosen by the agents.

### Using the relational graph algebra to translate explanations

The relational graph algebra is used to translate µ-calculus explanations back into $ATL$ ones. More precisely, the `translate_explanation` function below manipulates the µ-calculus explanation to:

1. project the explanation nodes on formulas and original states coming from the `original_state` attributor (Lines 3 and 4);
2. group nodes by their original state (Lines 5 and 6);
3. compute the pairs `unexplained_formulas` of original states and formulas still unexplained (Lines 7 to 10);
4. extend the nodes through the `extract_formulas` function (Lines 11 to 13) that takes the set `unexplained_formulas` of pairs of original states and unexplained formulas, and return a dictionary with two new keys: (1) `"explained"` contains the explained formulas of the node, (2) `"unexplained"` contains the unexplained ones;
5. select edges that are labelled with some actions (Lines 14 and 15);
6. keep only the useful information about nodes and edges (Lines 16 to 18); the `node_values` function keeps the original state and the explained and unexplained formulas, and `edge_values` keeps the actions of the edges.

```
1  def translate_explanation(explanation):
2      graph = explanation.graph
3      graph = graph.projection(
4              node_domain={"formula", "original"})
5      graph = graph.grouping(
6              node_group=("formulas", {"original"}))
7      unexplained_formulas = {
8          (node["original"], node["formula"])
9          for node in explanation.unexplained
10     }
11     graph = graph.extension(
12             node_extension=
13             extract_formulas(unexplained_formulas))
14     graph = graph.selection(edge_selector=
15                             lambda e: e[1].inputs)
16     graph = graph.mapping(
17             node_mapping=node_values,
18             edge_mapping=edge_values)
19     return graph
```

**Choosers**

Finally, a chooser is defined to expand partial explanations. When dealing with a `CEX` alias, this chooser gets, from the given choices, the original actions of the group—through the `actions_from_choices` function—, and displays a window showing one button per possible choice and asking the user to choose one of them—through the `window_for_choices` function and `window.display()`. The chosen action is then returned. The unimportant details of the two sub-functions are not shown here. They use PyNuSMV functionalities to extract possible action choices, and Python GUI functionalities to build and display the window.

```
@chooser
def single_action_chooser(obligation, choices, type_):
    if isinstance(obligation.formula, CEX):
        action_choices = actions_from_choices(choices)
        window = window_for_choices(action_choices)
        window.display()
        return {window.chosen_action}
```

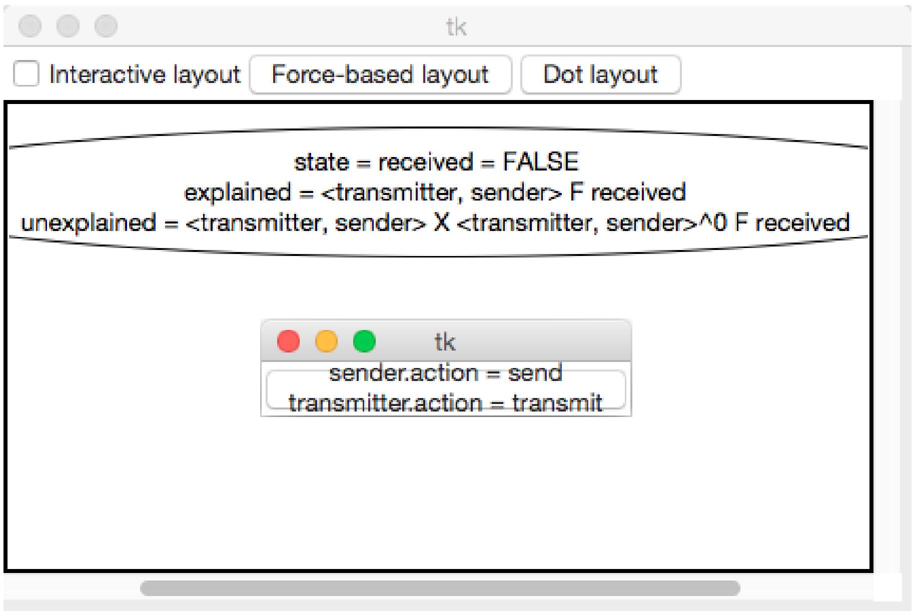
The chooser is then given to the `explain` function.

### 10.4.4 Visualizing explanations

Thanks to the visualization tool, it is possible to display and manipulate the translated explanations. In particular, the *ATL* implementation defines three new attributes for the nodes of the explanation to tweak what the visualization tool displays:

1. the `"_label_"` key is defined to display, as the label of the nodes, a sorted list of key, value pairs, instead of an unsorted one. This minor detail allows the user to understand node labels more easily as the information is displayed in the same order on every node;
2. the `"_view_"` key is similarly defined to order the key, value pairs displayed in the element and path inspectors;
3. the `"_menu_"` key is also defined to display, through a right-click menu, the list of unexplained formulas that can be clicked to explain the formula. This menu triggers the expansion of the currently displayed partial explanation, running through the chooser defined in the previous section to select the action to play.

Figure 10.14 shows the tool displaying the initial node of an explanation for why the formula $\langle\langle \mathit{transmitter}, \mathit{sender}\rangle\rangle \mathbf{F}\ \mathit{sent}$ is satisfied by the bit transmission problem, as well as the window displayed by the chooser to ask the user which action should be chosen to expand the explanation. In this case, there is only one action as both agents must agree to send and transmit the bit. The generating algorithm is limited to adequate explanations, so the other actions are not proposed as they do not lead to winning strategies. The full explanation has already been shown in Figure 10.12.



state = received = FALSE
explained = <transmitter, sender> F received
unexplained = <transmitter, sender> X <transmitter, sender>^0 F received

sender.action = send
transmitter.action = transmit

Figure 10.14: A snapshot of the explanation visualization tool for *ATL* explanations, with the window asking the user for the action to play.

### 10.4.5 Conclusion

The previous sections showed that the framework can be used to solve the model-checking problem of *ATL* formulas and to produce rich explanations. This section analyses the benefits and drawbacks of the framework, based on the *ATL* model-checking case.

The main advantage of the framework is the fact that the designer does not have to worry about solving the model-checking problem itself, nor about generating adequate explanations. Nevertheless, the effort that she would put into developing a model checker from scratch is transferred into translating models and formulas into the µ-calculus, as well as into translating explanations back into the top-level language. For instance, the translation from CGS to µ-calculus structures is not trivial to implement, and the framework gives no help for that.

Using the propositional µ-calculus as the base logic allows many existing logics to be translated. This section showed the case of $ATL$, but similar constructs can be implemented to solve the model-checking problems of $CTL$, $FCTL$, $CTLK$, $ARCTL$, $PDL$, or $ATLK_{IrF}$.

The framework features allow the designer to divide the concerns into smaller parts, first dealing with formula translations (with aliases and markers), then with single elements (with attributors), small subgraphs (with local translation), and with the whole explanation (with the algebra).

Furthermore, all the features are useful, as illustrated by the $ATL$ case. In particular, local translation is useless for cases such as $CTL$, but for $ATL$, where the model translation is difficult, local translators can help treating small parts of the explanation separately, instead of having to deal with the whole explanation graph at once.

Finally, the framework supports interactive and guided generation of the explanations through choosers. This can lead to (1) smaller manageable partial explanations that can be interactively expanded, as illustrated by the $ATL$ case, and (2) to guided generation, by embedding some particular generation strategies in choosers. This second use case will be discussed in the next chapter when comparing the framework to other existing approaches.

Nevertheless, the designer still needs to translate the top-level model into a µ-calculus structure by herself. This should be simpler than implementing the model checking algorithm, as well as generating explanations, but it can be difficult, especially with logics such as $ATL$. In simpler cases such as $(F)CTL(K)$, $ARCTL$, and $PDL$, the model stays the same.

Finally, the framework produces one single (partial) explanation at a time, for given state and formula. While the user or the designer can control the generation, the final result still represents one witness for the satisfaction. The framework cannot generate several explanations at the same time for the same pair of state and formula, and, if the user wants to get different explanations for the same satisfaction, the designer has to define choosers that remember the choices made before to produce different explanations whenever the *explain* algorithm is called.

# Chapter 11

# Part II: Conclusion

In the second part of this thesis, we describe a solution for generating and understanding rich explanations for multi-modal logics. Because of the branching characteristics of these logics, their explanations have a complex structure that makes them difficult to understand.

The proposed solution is a framework for µ-calculus based logics explanations. It integrates a µ-calculus model checker that generates rich explanations and provides several functionalities to translate them into explanations for a top-level logic such as $ATL$.

The framework produces adequate µ-explanations showing why a given µ-calculus formula $\phi$ is satisfied by a given µ-calculus Kripke structure $S$. In this context, an explanation is adequate if it matches $S$—that is, it is composed of elements of $S$—and it has the right structure to explain $\phi$.

In addition to the µ-calculus model checker with rich explanations, the framework provides several functionalities to translate them back into top-level logic ones:

- *aliases* allow the designer to link µ-calculus formulas to their corresponding top-level logic counterparts. Aliases can be viewed as other names for the µ-calculus formulas, but they also support other features such as negation—to derive positive normal forms—and substitution.

- the framework integrates the *relational graph algebra* of Dong et al. (see Section 9.4.3 and [DRS03a]). This algebra allows the designer to derive new explanations from the µ-calculus ones through relational operators such as selection, projection, grouping, etc.

- While the algebra allows the designer to manipulate the explanation as a whole, *attributors* take one explanation node or edge at a time and add information to them.

- *local translators* are attached to aliases and have access to the sub-graph explaining why the formula they are attached to is satisfied. They can then update this sub-graph to facilitate the translation of the explanation.
- The preceding features allow the designer to translate and produce rich explanations, embedding the information to understand them. On the other hand, *choosers* allow her to interfere in the generation process. They are functions that can be passed to the generating algorithm and that resolve the choices to produce particular explanations. These choosers can thus be used to perform guided generation of explanations, or to perform interactive generation where the user resolves the choices.
- Choosers introduce the notion of *partial explanations*. Such explanations lack some parts that are not fully explained, making them smaller, more manageable and more understandable. They can be later expanded, to explain the parts left unexplained so far.
- *markers* add information to formulas themselves. The framework provides two kinds of markers: points of interest and points of decision. The former can be used by the designer to mark formulas that are meaningful for the user. The latter are attached to formulas that should stay unexplained. The generating algorithm take points of decision into account and stops the generation when it encounters them. This leads to partial explanations as well.

The framework has been implemented with PyNuSMV, taking advantage of Python functionalities such as function decorators to easily describe aliases, attributors, local translators and the other features. The implementation also integrates a graphical tool to visualize, manipulate and explore relational graphs.

Finally, the whole framework has been validated on the case of $ATL$ model checking. $ATL$ can be translated into the µ-calculus, and this application showed that all the features provided by the framework are useful to translate µ-explanations into $ATL$ ones.

One of the main advantages of the framework is that many logics can be translated into the µ-calculus, such as $CTL$, $FCTL$, $ARCTL$, $CTLK$, $ATL$, $ATLK_{IrF}$, and $PDL$. It is thus generic enough to provide model-checking functionalities for all of them. Furthermore, thanks to the framework, the designer does not have to worry about designing and implementing a model checker, nor to worry about generating rich explanations. Nevertheless, she has to translate the top-level models and formulas into µ-calculus ones. Model translation can be difficult—for instance, the translation from an *ATL* CGS to a µ-calculus structure is not trivial—, and the framework gives no help to complete this task.

The framework features are generic and complement each other:

- the relational algebra, attributors and local translators manipulate the explanation at different scales;
- points of decision and choosers work together to produce smaller partial strategies and to select the explanations of interest;
- points of interest and aliases add information to important formulas.

Finally, the visualization tool provided by the framework complements the translation features. The latter help the designer to produce useful explanations while the former helps to user in visualizing, manipulating and exploring it.

On the other hand, one of the main drawbacks of the framework is the fact that it produces one single explanation at a time. Representing several explanations at once could help the user to extract the reasons for the satisfaction of the formula more easily. This idea is briefly discussed in Section 11.2 presenting some future work.

## 11.1 Comparison with related work

This section compares the µ-calculus framework of this thesis with the related work of Chapter 9. It discusses explanations for *CTL* model checking, then for multi-modal logics, and finally for the µ-calculus.

### 11.1.1 Explanations for *CTL* model checking

The explanations of Rasse [Ras92] and the *tree-like counter-examples* of Clarke et al. [CJLV02] are similar structures. They are hierarchies of paths, and Rasse additionally annotates them with the formulas they explain. They are composed of paths of the model because *CTL* path formulas can be witnessed by single paths. This is not the case for our framework as µ-calculus explanations cannot be defined as a hierarchy of looping paths in general. Nevertheless, the explanations of Rasse and Clarke et al. are designed for *CTL* and can be produced with the µ-calculus framework of this thesis. Furthermore, the *multi-paths* of Buccafurri et al. [BEGL01] are very similar to tree-like counter-examples, but the goal of Buccafurri et al. is to study the existence of linear counter-examples, while the goal of the other solutions is to help the user understand why the formula is satisfied.

The *justifications* of Roychoudhury et al. [RRR00], the *proof-like counter-examples* of Gurfinkel and Chechik [GC03a], and the *game-based counter-examples* of Shoham and Grumberg [SG07] share the same level of details. They are more detailed than explanations such as tree-like counter-examples as they expose logical steps in addition to transition steps. For instance, they explain that state $q$ satisfies $\phi_1 \wedge \phi_2$ because $q$ satisfies both $\phi_1$ and $\phi_2$. In addition, justifications, proof-like counter-examples and game-based counter-examples explain why universal operators are satisfied. Again, explanations with such details can be generated with the µ-calculus framework. A major difference between justifications, game-based counter-examples, and the µ-calculus framework is that the first ones are developed on a logic-programming-based model-checking framework, the second ones on a game-based model-checking one, and the third is tailored to work with BDD-based model checking.

The visualization and generation strategies proposed by the proof-like counter-examples framework can be easily implemented within the µ-calculus framework by defining adequate choosers. The designer can implement strategies to produce $CTL$ counter-examples, but also for any other logic that can be translated into the µ-calculus. Finally, the visualization tool KEGVis is similar to visualization tool of our framework: the main idea of these tools is to present the explanation as an annotated graph, with additional windows displaying more precise information. The difference between KEGVis and our tool is that the former is centered around the notion of proof while the latter is centered around the checked model itself.

Mateescu's *extended Boolean graphs* (EBGs) capture the part of the checked structure as well as the sub-formulas of the checked formula responsible for the model-checking outcome [Mat00]. EBGs nodes are linked to Boolean equation systems variables, and a BES variable says whether some state of the structure satisfies a formula or not. EBGs are thus full explanations for BES, and thus for the full $CTL$, including universal operators. Nevertheless, the translation of EBG to $CTL$ explanations is not detailed in Mateescu's paper, making it difficult to compare it with the features provided by the µ-calculus framework and the associated visualization tool.

Tan and Cleaveland's support sets are similar to EBGs [TC02]. A translation of support sets to $CTL^*$ explanations is given, but the paper concentrates on linear witnesses while support sets could be used to extract tree-like counter-examples. Their work is more theoretical than the framework presented in this thesis and does not provide functionalities to translate support sets into actual explanations. Also, Mateescu's and Tan and Cleaveland's solutions are based on the framework of Boolean equation systems while the solution of this thesis is based on BDD-based model-checking techniques.

Finally, Meolic et al. witness automata do not solve the same problem as µ-calculus explanations. The former concentrate on linear witnesses for test case generation while the latter are used to give insight on the model-checking outcome. One advantage of these automata is that they represent several explanations at once.

### 11.1.2 Explanations for multi-modal logics

The visualization tool of MCMAS displays tree-like counter-examples for $CTLK$ and $ATL$ formulas [LR06a, LQR09, LQR15]. It is similar to the visualization tool of our µ-calculus framework but it is a bit more limited: the inspection capabilities are limited to one state at a time—thus, no full path inspection—, and the graph is fixed, that is, the user cannot move its nodes and edges. Furthermore, the nodes and edges are not annotated with the formulas they explain.

On the other hand, MCK provides several debugging functionalities [GvdM04]. First, it can export the part of the model resulting from SAT-based model checking, but it is not annotated with subformulas. Furthermore, it provides a debugging game inspired by Stirling's games [Sti95] in which the user can try to show why the model-checking outcome is wrong while the system shows her why it is actually right. Such debugging game can be implemented with the choosers provided by the µ-calculus framework. In this case, the user would choose the successor for inclusive choices ($\wedge$ and $\Box_i$ formulas) while the system would choose the successor for exclusive ones ($\vee$ and $\Diamond_i$ formulas).

Tree-like annotated counter-examples (TLACEs) take direct inspiration from the work of Rasse and from the tree-like counter-examples of Clarke et al. [CJLV02]. They can be viewed as a hierarchy of annotated paths. While these counter-examples allow the user to better understand violations of $CTLK$ formulas, they still suffer from some limitations. They do not explain why a universal operator—such as $\mathbf{AF}\ \phi$ formulas—is satisfied by a given structure, as such an explanation cannot be expressed as a single path. Furthermore, a TLACE represents a single arbitrarily chosen explanation. The user has no control on the way the counter-example is generated. Finally, it can be very large. The µ-calculus framework overcomes all these limitations: it can explain universal operators by extracting the sub-model responsible for the satisfaction, it allows the designer to generate particular counter-examples through choosers, and partial explanations allow her to manage the size of the witness.

Finally, the *ALCCTL* counter-examples of Weitl et al. [WNF10] are similar to TLACEs as they are graphs with paths and states annotated with formulas. Nevertheless, they are designed for another logic—*ALCCTL*—and give more information for some cases: they provide all witnesses for **AX** and **EX** formulas, as well as all instantiations of *ALC* expressions. Nevertheless, like TLACEs, they do not explain other **A** operators such as **AF**, **AG**, **AU** and **AW**. Furthermore, they provide a lazy interactive generation algorithm. This feature can be implemented with the µ-calculus framework by defining the adequate choosers, but it is not clear whether the *ALC* part of the logic can be translated into the µ-calculus.

### 11.1.3 Explanations for the µ-calculus

The relational graph algebra of Dong et al. is designed to manipulate explanations [DRS03a]. It is reused by the µ-calculus framework. Nevertheless, Dong's goals are not the same as the goals of the µ-calculus framework. The former uses the algebra to derive different views of the evidence to present them to the user. On the other hand, the µ-calculus framework uses the algebra to produce new explanations that can be very different from the original one.

The explanations from the work of Kick [Kic95b, Kic95a] are similar to the µ-calculus explanations of this thesis. The main difference between the two solutions is the generation algorithm. While Kick's solution updates the model-checking algorithm to compute and store the additional information needed to explain why least fixpoint formulas are satisfied, the solution of this thesis relies on BDD caching mechanisms to retrieve the information. The advantage of the latter is that the algorithm is simpler, but it has no control on the caching mechanism to make sure that the intermediate computations are still accessible when building the explanation.

Linssen's generic diagnostic graphs [Lin11] are similar to Kick's explanations and to the adequate µ-calculus explanations of this thesis. They are graphs where nodes are couples of state and formula. The proof graphs of Cranen et al. [CLW13] share the same idea as Linssen's graphs, but are adapted to richer formalisms. The difference with the µ-calculus framework of this thesis is that these graphs are generated from BES instead of BDDs.

Finally, model-checking certificates of Namjoshi [Nam01] and Hofmann and Rueß [HR14] are closely related to adequate explanations as the latter can also be used to check that the model-checking outcome is correct. Nevertheless, the approaches of Namjoshi, Hoffmann and Rueß work within a game-based framework.

All solutions discussed in this section work for particular logics such as $CTL$, $CTLK$, the µ-calculus, $ALCCTL$, or are generic solutions with some application to one use case such as BES and their extensions, games, proofs. But no work proposes a solution to produce explanations and to translate them back into the original language, as the µ-calculus framework of this thesis. They either limit themselves to one logic, or they provide generic structures without giving explicit help for applying and translating it into something useful for the end user.

## 11.2 Future work

Several improvements and evolutions can be made on the µ-calculus framework. This section lists and explains some of them.

**Providing more than one explanation at a time** One of the main disadvantages of the µ-calculus explanations is that they represent one single witness of the satisfaction. If the user wants another witness, she has to re-run the generating algorithm with tuned choosers to get another explanation. It would be interesting to explore how we could represent several explanations at once. A set of states can be represented with a single BDD, so such an explanation could be a graph with BDDs in nodes instead of single states. Nevertheless, this raises some problems such as (1) how to display a set of states to the user, and (2) how to treat paths with different lengths. This idea is related to the one of Shen et al. [SQL05b, SQL05a]. They propose to post-process an $ACTL$ linear counter-example, represented as a list of BDDs, to build a list of BDD cubes that captures more executions of the system that violate the formula. The difference with the idea above is that the one of Shen et al. post-processes the counter-example to derive several traces while we propose to directly produce several explanations during the generation process.

**Using the alternating-time µ-calculus as base logic** Translating a CGS and an $ATL$ formula into µ-calculus model and formula is not an easy task compared to other logics such as $CTL$ and $CTLK$. The difficulty comes from the fact that the CGS transition relation is translated into several µ-calculus transition relations, and one step of the original model corresponds to two steps of the translated one. One solution to make this particular translation easier is to use the alternating-time µ-calculus as base logic instead of the propositional µ-calculus. In this case, the notion of adequate explanations must be slightly reworked as the $\Box_i$ and $\Diamond_i$ operators are replaced by their strategic counter-part. On the other hand, this new framework would still be adequate for all the logics already mentioned.

**Helping the designer to translate the model** One of the drawbacks of the µ-calculus framework is that it gives no help to the designer for translating the original model into a µ-calculus one. It would be interesting to explore solutions to provide translation functionalities for the model itself. With such translation functionalities, the translation of explanations back into the original language could become automatic.

# *Chapter 12*

# *Conclusion*

In the first part of this thesis, we present algorithms to solve the model-checking problem of $ATLK_{irF}$, a multi-modal logic that mixes temporal, knowledge and strategic operators for reasoning about systems with fairness constraints and agents with imperfect information. These algorithms are based on an enumeration of the strategies of the agents, with several improvements such as the restriction to partial strategies—with the *partial* approach—, the usage of pre-filtering—for the *naive* and *partial* algorithms—, and the construction of the strategies from the target states with the *backward* approach. They are experimentally compared to other symbolic algorithms—the *early* and *symbolic* ones—, and the results show that each approach outperforms the others on some cases, and works worse on others. More precisely, the experiments showed that:

- The naive approach is not efficient at all; this is expected as it blindly enumerates and checks all uniform strategies before concluding.
- The partial approach is really good when most of the strategies are winning, but performs poorly when showing that there are no winning strategies.
- The early approach presents a better trade-off and can efficiently handle cases with and without winning strategies.
- The symbolic approach works better on models with a huge number of strategies as it handles them all at the same time.
- The backward approach is limited to reachability objectives but works better than any other approach on these cases.

- Pre-filtering can improve the process when the model contains a lot of losing moves, but can make it worse otherwise.

In the second part, we present a solution for providing rich explanations for multi-modal logics. This solution is a µ-calculus-based model-checking framework with rich explanations. As many logics can be translated into the µ-calculus, the framework has a large range of applications. This framework provides a µ-calculus model checker that produces graphs as adequate explanations, and a set of features to translate them into the original modeling language. These features are:

- formula aliases, to attach top-level formulas to µ-calculus ones;
- the relational algebra of Dong et al. [DRS03a] to manipulate and transform the µ-calculus explanations;
- the notion of explanation attributes, and the usage of attributors to attach these attributes to explanation elements;
- local translators, to translate small parts of the explanation;
- choosers that can produce partial explanations;
- markers to attach extra information to formulas.

To show that the framework is useful, it is applied to the case of $ATL$ model checking. The resulting explanations represent the parts of the original model responsible for the model-checking outcome, annotated with the sub-formulas of interest. Finally, we present a graphical tool to display, manipulate and explore the translated explanations.

The µ-calculus framework cannot be used to implement the semi-symbolic approaches described in the first part, because the strategies are explicitly enumerated by the model-checking approaches while the µ-calculus framework offers a fully symbolic solution only. Nevertheless, extracting an explanation for why an $ATLK_{irF}$ formula is satisfied is not very difficult. Indeed, as these semi-symbolic algorithms enumerate and check each strategy, they can keep track of the ones that are winning for the states of interest. An explanation for the satisfaction is then the execution of a winning strategy from the state of interest. Annotations are also easily added by keeping track of the checked sub-formulas in addition to the strategies themselves.

Furthermore, *interactive exploration* of one particular strategy can be achieved by asking the user which successor must be explored. Nevertheless, *interactive generation* of such explanations would be way less efficient as, in this case, the algorithms would have to check *all* strategies to keep the winning ones instead of stopping at the first one.

On the other hand, the *symbolic* approach presented in Chapter 6 can be directly encoded into the μ-calculus framework. Given a concurrent game structure $S$, the approach derives a new structure $EncStrats(S)$ by encoding the uniform strategies of the agents in its states. The winning strategies and the states satisfying the formula are then computed with fixpoints on the derived structure. This $EncStrats(S)$ translation can be easily adapted to produce a μ-calculus Kripke structure. Furthermore, the functions the evaluation relies on can be translated into μ-calculus formulas with $\Box_i$ and $\Diamond_i$ operators on adequate translation relations of the μ-calculus structure, and the whole model-checking algorithm can be translated into μ-calculus formulas, too.

This translation would bring the full set of features to control the generation of the explanations. Thanks to these features, the user could choose the winning strategy to explore, and would be able to do it interactively. Nevertheless, this way of translating the $ATLK_{irF}$ model-checking problem into the μ-calculus introduces a major limitation: because the $eval^{Symbolic}_{ATLK_{irF}}$ algorithm first fixes the strategy before showing why it is winning, the user would need to choose the whole strategy to play through the initial μ-calculus state. In other words, implementing the $eval^{Symbolic}_{ATLK_{irF}}$ algorithm within the μ-calculus framework enforces the μ-calculus explanation to start by choosing the strategy to play, then showing that it is effectively winning. On the other hand, the $ATL$ translation given in Chapter 10 can be used to explore and build the winning strategies incrementally because the translation does not force the framework to choose the strategy *a priori*.

A solution to this limitation is to extend the μ-calculus framework to produce multiple explanations at once: instead of explaining why one state $q^{ES}$ of $EncStrats(S)$ satisfies some μ-calculus formula $\phi$, the framework could explain why a set of states $Q'$ of $EncStrats(S)$ satisfy $\phi$. In this case, $Q'$ could be all the states of $EncStrats(S)$ corresponding to one state $q$ of $S$. These states would thus represent all the strategies that are winning in $q$. Then the framework could interactively explore these strategies by asking the user which action should be played, and by restricting the successors to the ones that fit the user choices. This solution shows the need for the framework to support explanations for sets of states instead of single states.

Finally, this thesis concentrates on two problems: the $ATLK_{irF}$ model-checking problem—and more largely, the problem of checking the existence of uniform memoryless strategies—, and the problem of producing and exploring rich explanations for multi-modal logics. Nevertheless, several related problems remain unsolved. The rest of this chapter discusses three of them.

**Approximations for the existence of uniform strategies** The complexity analysis of Section 5.6 and the experiments of Chapter 7 showed that model checking the existence of uniform memoryless strategies is a difficult problem, both in theory and in practice. The complexity analysis showed that the problem is $\Delta_P^2$-complete—that is, the problem for *one* strategic operator is NP-complete—, and the experiments showed that the problem cannot be solved in reasonable time for models with more than tens of thousands of states.

One way to tackle this complexity is to work with *approximations*. One approximation is already extensively used by this thesis through the pre-filtering feature. Indeed, pre-filtering works because, if there is no winning general strategy in some state $q$ of the checked iCGSf, then there is no winning uniform one in $q$. Another approximation is used by the *early* approach in which a given partial strategy is run through the $filter_{\mathbf{A}}$ algorithms to check whether all its extensions are winning.

To solve the model-checking problem more efficiently, we can look for tighter approximations. Jamroga and other researchers already attacked this idea by proposing some variants of the alternating epistemic μ-calculus that lead to tighter approximations [JKK15, JKK16].

**Generating minimal explanations** The algorithm for generating μ-calculus explanations provides an adequate explanation, without any guarantee about its size. In particular, it does not try to find the smallest explanation that effectively illustrates the satisfaction of the formula.

For the simpler case of trace-based explanations, Clarke et al. already showed that finding the smallest linear counter-example is an NP-complete problem [CGMZ95], and some authors already attacked the problem of generating minimal linear counter-examples [GMZ04, HG08, ZJC11]. Nevertheless, new solutions are needed for the case of μ-calculus explanations, as they are not based on individual paths. Also, it is not clear what would be the best minimality criterion for the size of such complex branching explanations.

**Finding the cause of error** Even if the generated explanation is the smallest possible one, it still can be too large to be correctly explored and understood by the user. In the second part of this thesis, we proposed a graphical tool to explore and inspect the explanations but, instead of letting the user use this tool to find the cause of the error by herself, we could develop techniques to find and isolate this cause.

Many ideas have already been proposed for finding and isolating the possible causes of a formula violation in the domain of trace-based explanations [BNR03], [RS04, JRS04], [GV03, GKL04, GK05, GCKS06]. But how these ideas can be adapted for the case of branching explanations remains an open question.

# References

[ÅA12] Thomas Ågotnes and Natasha Alechina. Epistemic coalition logic: completeness and complexity. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012*, pages 1099–1106, 2012.

[AG11] Krzysztof R. Apt and Erich Grädel. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

[AHK98] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. In Amir Pnueli Willem-Paul de Roever, Hans Langmaack, editor, *Compositionality: The Significant Difference*, volume 1536 of *Lecture Notes in Computer Science*, pages 23–60. Springer Berlin Heidelberg, 1998.

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, September 2002.

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Amsterdam, The Netherlands, March 22-28, 1999*, pages 193–207, 1999.

[BCLM09] Thomas Brihaye, Arnaud Da Costa, François Laroussinie, and Nicolas Markey. ATL with strategy contexts and bounded memory. In *Logical Foundations of Computer Science, International Symposium, LFCS 2009, Deerfield Beach, FL, USA, January 3-6, 2009*, pages 92–106, 2009.

[BCM$^+$90] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill, and Lain-Jinn Hwang. Symbolic model checking: $10^{20}$ states and beyond. In *Logic in Computer Science, 1990. LICS'90*, pages 428–439. IEEE, 1990.

[BDF14] Rodica Bozianu, Cătălin Dima, and Emmanuel Filiot. Safraless synthesis for epistemic temporal specifications. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*, pages 441–456. Springer International Publishing, 2014.

[Bea96] David M. Beazley. SWIG: an easy to use tool for integrating scripting languages with C and C++. In *Proceedings of the 4th conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, TCLTK'96, pages 15–15, Berkeley, CA, USA, 1996. USENIX Association.

[BEGL01] F. Buccafurri, T. Eiter, G. Gottlob, and N. Leone. On ACTL formulas having linear counterexamples. *Journal of Computer and System Sciences*, 62(3):463 – 515, 2001.

[Bel14] Francesco Belardinelli. Reasoning about knowledge and strategies: Epistemic strategy logic. In *Proceedings 2nd International Workshop on Strategic Reasoning, SR 2014, Grenoble, France, April 5-6, 2014.*, pages 27–33, 2014.

[BJ10] Nils Bulling and Wojciech Jamroga. Verifying agents with memory is harder than it seemed. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, pages 699–706, 2010.

[BJ11] Nils Bulling and Wojciech Jamroga. Alternating epistemic mu-calculus. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 109–114, 2011.

[BJP14a] Nils Bulling, Wojciech Jamroga, and Matei Popovici. Agents with truly perfect recall in alternating-time temporal logic. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, pages 1561–1562, 2014.

[BJP14b] Nils Bulling, Wojciech Jamroga, and Matei Popovici. ATL* with truly perfect recall: Expressivity and validities. In *ECAI*

*2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic*, pages 177–182, 2014.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* The MIT Press, 2008.

[BNR03] Thomas Ball, Mayur Naik, and Sriram K Rajamani. From symptom to cause: localizing errors in counterexample traces. In *ACM SIGPLAN Notices*, volume 38, pages 97–105. ACM, 2003.

[BP12] Simon Busard and Charles Pecheur. Rich counter-examples for temporal-epistemic logic model checking. In *Proceedings Second International Workshop on Interactions, Games and Protocols, IWIGP 2012, Tallinn, Estonia, 25th March 2012*, pages 39–53, 2012.

[BP13] Simon Busard and Charles Pecheur. PyNuSMV: NuSMV as a Python library. In Guillaume Brat, Neha Rungta, and Arnaud Venet, editors, *Nasa Formal Methods 2013*, volume 7871 of *LNCS*, pages 453–458. Springer-Verlag, 2013.

[BPQR13] Simon Busard, Charles Pecheur, Hongyang Qu, and Franco Raimondi. Reasoning about strategies under partial observability and fairness constraints. In Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi, editors, *Proceedings 1st International Workshop on Strategic Reasoning, SR 2013, Rome, Italy, March 16-17, 2013*, volume 112 of *EPTCS*, pages 71–79, 2013.

[BPQR14] Simon Busard, Charles Pecheur, Hongyang Qu, and Franco Raimondi. Improving the model checking of strategies under partial observability and fairness constraints. In Stephan Merz and Jun Pang, editors, *Formal Methods and Software Engineering*, volume 8829 of *Lecture Notes in Computer Science*, pages 27–42. Springer International Publishing, 2014.

[BPQR15] Simon Busard, Charles Pecheur, Hongyang Qu, and Franco Raimondi. Reasoning about memoryless strategies under partial observability and unconditional fairness constraints. *Information and Computation*, 242:128 – 156, 2015.

[Bry86] Randal E Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 100(8):677–691, 1986.

[CCG$^+$02] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An open-source tool for symbolic model checking. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer Berlin Heidelberg, 2002.

[CCJ$^+$] Roberto Cavada, Alessandro Cimatti, Charles Arthur Jochim, Gavin Keighren, Emanuele Olivetti, Marco Pistore, Marco Roveri, and Andrei Tchaltsev. NuSMV 2.5 user manual.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs, Workshop, Yorktown Heights, New York, May 1981*, pages 52–71, 1981.

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[CG05] Marsha Chechik and Arie Gurfinkel. A framework for counterexample generation and exploration. In *International Conference on Fundamental Approaches to Software Engineering*, pages 220–236. Springer, 2005.

[CG07] Marsha Chechik and Arie Gurfinkel. A framework for counterexample generation and exploration. *International Journal on Software Tools for Technology Transfer*, 9(5-6):429–445, 2007.

[CGMZ95] E. M. Clarke, O. Grumberg, K. L. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proceedings of the 32Nd Annual ACM/IEEE Design Automation Conference*, DAC '95, pages 427–432, New York, NY, USA, 1995. ACM.

[CGP99] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

[CHP10] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Information and Computation*, 208(6):677 – 693, 2010.

[CJLV02] E. M. Clarke, S. Jha, Y. Lu, and H. Veith. Tree-like counterexamples in model checking. In *Proc. of the 17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, pages 19–29, 2002.

[CLW13] Sjoerd Cranen, Bas Luttik, and Tim AC Willemse. Proof graphs for parameterised boolean equation systems. In *International Conference on Concurrency Theory*, pages 470–484. Springer, 2013.

[CLW15] Sjoerd Cranen, Bas Luttik, and Tim AC Willemse. Evidence for fixpoint logic. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 41. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[CSS10] Jan Calta, Dmitry Shkatov, and Holger Schlingloff. Finding uniform strategies for multi-agent systems. In Jürgen Dix, João Leite, Guido Governatori, and Wojtek Jamroga, editors, *Computational Logic in Multi-Agent Systems*, volume 6245 of *Lecture Notes in Computer Science*, pages 135–152. Springer Berlin / Heidelberg, 2010.

[DEG10] Catalin Dima, Constantin Enea, and Dimitar P. Guelev. Model-checking an alternating-time temporal logic with knowledge, imperfect information, perfect recall and communicating coalitions. In *Proceedings First Symposium on Games, Automata, Logic, and Formal Verification, GANDALF 2010, Minori (Amalfi Coast), Italy, 17-18th June 2010*, pages 103–117, 2010.

[DJ10] Mehdi Dastani and Wojciech Jamroga. Reasoning about strategies of multi-agent programs. In *Proceedings of AAMAS 10*, pages 997–1004, 2010.

[DRS03a] Y. Dong, C. R. Ramakrishnan, and S. A. Smolka. Model checking and evidence exploration. In *Proc. of the 10th IEEE International Conference on Engineering of Computer-Based Systems (ECBS 2003)*, pages 214–223, 2003.

[DRS03b] Yifei Dong, CR Ramakrishnan, and Scott A Smolka. Evidence explorer: A tool for exploring model-checking proofs. In *International Conference on Computer Aided Verification*, pages 215–218. Springer, 2003.

[DT11] Catalin Dima and Ferucio Laurentiu Tiplea. Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoRR*, abs/1102.4225, 2011.

[FHMV95] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge.* MIT Press, Cambridge, 1995.

[GC03a] A. Gurfinkel and M. Chechik. Proof-like counter-examples. In Hubert Garavel and John Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2619 of *Lecture Notes in Computer Science*, pages 160–175. Springer Berlin / Heidelberg, 2003.

[GC03b] Arie Gurfinkel and Marsha Chechik. Generating counterexamples for multi-valued model-checking. In *International Symposium of Formal Methods Europe*, pages 503–521. Springer, 2003.

[GCKS06] Alex Groce, Sagar Chaki, Daniel Kroening, and Ofer Strichman. Error explanation with distance metrics. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(3):229–247, 2006.

[GJ04] Valentin Goranko and Wojciech Jamroga. Comparing semantics of logics for multi-agent systems. *Synthese*, 139(2):241–280, 2004.

[GK05] Alex Groce and Daniel Kroening. Making the most of BMC counterexamples. *Electronic Notes in Theoretical Computer Science*, 119(2):67–81, 2005.

[GKL04] Alex Groce, Daniel Kroening, and Flavio Lerda. Understanding counterexamples with explain. In *International Conference on Computer Aided Verification*, pages 453–456. Springer, 2004.

[GMZ04] Paul Gastin, Pierre Moro, and Marc Zeitoun. Minimization of counterexamples in SPIN. In *International SPIN Workshop on Model Checking of Software*, pages 92–108. Springer, 2004.

[GN00] Emden R Gansner and Stephen C North. An open graph visualization system and its applications to software engineering. *Software Practice and Experience*, 30(11):1203–1233, 2000.

[Grä04] Erich Grädel. Positional determinacy of infinite games. In Volker Diekert and Michel Habib, editors, *STACS 2004*, volume 2996 of *Lecture Notes in Computer Science*, pages 4–18. Springer Berlin Heidelberg, 2004.

[GRR01] Hai-Feng Guo, CR Ramakrishnan, and IV Ramakrishnan. Speculative beats conservative justification. In *International Conference on Logic Programming*, pages 150–165. Springer, 2001.

[GV03] Alex Groce and Willem Visser. What went wrong: Explaining counterexamples. In *International SPIN Workshop on Model Checking of Software*, pages 121–136. Springer, 2003.

[GvdM04] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *LNCS*, pages 479–483. Springer-Verlag, 2004.

[HG08] Henri Hansen and Jaco Geldenhuys. Cheap and small counterexamples. In *Software Engineering and Formal Methods, 2008. SEFM'08. Sixth IEEE International Conference on*, pages 53–62. IEEE, 2008.

[HKQ98] Thomas A. Henzinger, Orna Kupferman, and Shaz Qadeer. From pre-historic to post-modern symbolic model checking. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification: 10th International Conference, CAV'98 Vancouver, BC, Canada, June 28 – July 2, 1998*, pages 195–206, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[HR14] Martin Hofmann and Harald Ruess. Certification for $\mu$-calculus with winning strategies. *arXiv preprint arXiv:1401.1693*, 2014.

[Hua15] Xiaowei Huang. Bounded model checking of strategy ability with perfect recall. *Artif. Intell.*, 222:182–200, 2015.

[HvdM09] Xiaowei Huang and Ron van der Meyden. Model checking games for a fair branching-time temporal epistemic logic. In *Australasian Joint Conference on Artificial Intelligence*, pages 11–20. Springer, 2009.

[HvdM14a] Xiaowei Huang and Ron van der Meyden. An epistemic strategy logic (extended abstract). In Fabio Mogavero, Aniello

Murano, and Moshe Y. Vardi, editors, Proceedings 2nd International Workshop on *Strategic Reasoning,* Grenoble, France, April 5-6, 2014, volume 146 of *Electronic Proceedings in Theoretical Computer Science*, pages 35–41. Open Publishing Association, 2014.

[HvdM14b] Xiaowei Huang and Ron van der Meyden. Symbolic model checking epistemic strategy logic. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 1426–1432, 2014.

[INH96] H. Iwashita, T. Nakata, and F. Hirose. CTL model checking based on forward state traversal. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 82–87, Nov 1996.

[JÅ07] W. Jamroga and T. Ågotnes. Constructive knowledge: what agents can achieve under imperfect information. *Journal of Applied Non-Classical Logics*, 17(4):423–475, 2007.

[JB11] Wojciech Jamroga and Nils Bulling. Comparing variants of strategic ability. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, IJCAI'11, pages 252–257. AAAI Press, 2011.

[JD06] Wojciech Jamroga and Jürgen Dix. Model checking abilities under incomplete information is indeed $\Delta_2^P$-complete. In *EUMAS'06*, 2006.

[JD08] Wojciech Jamroga and Jürgen Dix. Model checking abilities of agents: A closer look. *Theory of Computing Systems*, 42(3):366–410, 2008.

[JKK15] Wojciech Jamroga, Michal Knapik, and Damian Kurpiewski. Approximating strategic abilities under imperfect information: a naive approach. *CoRR*, abs/1510.06587, 2015.

[JKK16] Wojciech Jamroga, Michal Knapik, and Damian Kurpiewski. An approach to model checking ATLir. *CoRR*, abs/1612.02684, 2016.

[JRS04] HoonSang Jin, Kavita Ravi, and Fabio Somenzi. Fate and free will in error traces. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(2):102–116, 2004.

[JvdH04] Wojciech Jamroga and Wiebe van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, Volume 63(2):185–219, 2004.

[JZP15] Guifei Jiang, Dongmo Zhang, and Laurent Perrussel. Knowledge sharing in coalitions. In *AI 2015: Advances in Artificial Intelligence - 28th Australasian Joint Conference, Canberra, ACT, Australia, November 30 - December 4, 2015*, pages 249–262, 2015.

[KÅJ14] Piotr Kaźmierczak, Thomas Ågotnes, and Wojciech Jamroga. Multi-agency is coordination and (limited) communication. In HoaKhanh Dam, Jeremy Pitt, Yang Xu, Guido Governatori, and Takayuki Ito, editors, *PRIMA 2014: Principles and Practice of Multi-Agent Systems*, volume 8861 of *Lecture Notes in Computer Science*, pages 91–106. Springer International Publishing, 2014.

[KB08] Sascha Klüppelholz and Christel Baier. Alternating-time stream logic for multi-agent systems. In *Coordination Models and Languages*, LNCS 5052, pages 184–198. Springer, 2008.

[Kic95a] Alexander Kick. Generation of witnesses for global $\mu$-calculus model checking. Technical report, Universität Karlsruhe, Germany, 1995.

[Kic95b] Alexander Kick. Tableaux and witnesses for the $\mu$-calculus. Technical report, Universität Karlsruhe, Germany, 1995.

[Koz83] Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[KVW00] Orna Kupferman, Moshe Y Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM (JACM)*, 47(2):312–360, 2000.

[Lin11] Charl A.P. Linssen. Diagnostics for model checking. Master's thesis, Eindhoven University of Technology, 2011.

[LMO08] François Laroussinie, Nicolas Markey, and Ghassan Oreiby. On the expressiveness and complexity of ATL. *CoRR*, abs/0804.2435, 2008.

[LMS15] François Laroussinie, Nicolas Markey, and Arnaud Sangnier. ATLsc with partial observation. In *Proceedings Sixth International Symposium on Games, Automata, Logics and*

*Formal Verification, GandALF 2015, Genoa, Italy, 21-22nd September 2015*, pages 43–57, 2015.

[LPR07] Alessio Lomuscio, Charles Pecheur, and Franco Raimondi. Automatic verification of knowledge and time with NuSMV. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1384–1389, 2007.

[LQR09] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Proceedings of CAV 2009*, volume 5643 of *LNCS*, pages 682–688. Springer, 2009.

[LQR15] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. MCMAS: an open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer*, pages 1–22, 2015.

[LR06a] Alessio Lomuscio and Franco Raimondi. MCMAS: A model checker for multi-agent systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 450–454. Springer, 2006.

[LR06b] Alessio Lomuscio and Franco Raimondi. Model checking knowledge, strategies, and games in multi-agent systems. In *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, pages 161–168, 2006.

[LS02] Martin Lange and Colin Stirling. Model checking games for branching time logics. *Journal of Logic and Computation*, 12(4):623–639, 2002.

[Mat00] Radu Mateescu. Efficient diagnostic generation for boolean equation systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 251–265. Springer, 2000.

[MFG04] R. Meolic, A. Fantechi, and S. Gnesi. Witness and counterexample automata for ACTL. In David de Frutos-Escrig and Manuel Núñez, editors, *Formal Techniques for Networked and Distributed Systems – FORTE 2004*, volume 3235 of *Lecture Notes in Computer Science*, pages 259–275. Springer Berlin / Heidelberg, 2004.

[MMV10] Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi. Reasoning about strategies. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, pages 133–144, 2010.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[MT98] Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*. Springer, 1998.

[Nam01] Kedar S Namjoshi. Certifying model checkers. In *International Conference on Computer Aided Verification*, pages 2–13. Springer, 2001.

[Pau02] Marc Pauly. A modal logic for coalitional power in games. *Journal of Logic and Computation*, 12(1):149–166, 2002.

[PBJ14] Jerzy Pilecki, Marek A. Bednarczyk, and Wojciech Jamroga. Synthesis and verification of uniform strategies for multi-agent systems. In Nils Bulling, Leendert van der Torre, Serena Villata, Wojtek Jamroga, and Wamberto Vasconcelos, editors, *Computational Logic in Multi-Agent Systems*, volume 8624 of *Lecture Notes in Computer Science*, pages 166–182. Springer International Publishing, 2014.

[PGD$^+$04] Giridhar Pemmasani, Hai-Feng Guo, Yifei Dong, CR Ramakrishnan, and IV Ramakrishnan. Online justification for tabled logic programs. In *International Symposium on Functional and Logic Programming*, pages 24–38. Springer, 2004.

[PL03] W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, pages 209–216, New York, NY, USA, 2003. ACM.

[PR06] Charles Pecheur and Franco Raimondi. Symbolic model checking of logics with actions. In *Model Checking and Artificial Intelligence, 4th Workshop, MoChArt IV, Riva del Garda, Italy, August 29, 2006*, pages 113–128, 2006.

[Ras92] A. Rasse. Error diagnosis in finite communicating systems. In Kim Larsen and Arne Skou, editors, *Computer Aided Verification*, volume 575 of *Lecture Notes in Computer Science*, pages 114–124. Springer Berlin / Heidelberg, 1992.

[RCDH07] Jean-François Raskin, Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Algorithms for omega-regular games with imperfect information. *Logical Methods in Computer Science*, 3(3), 2007.

[RRR00] Abhik Roychoudhury, CR Ramakrishnan, and IV Ramakrishnan. Justifying proofs using memo tables. In *International Conference on Principles and Practice of Declarative Programming: Proceedings of the 2 nd ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 178–189, 2000.

[RRS$^+$00] CR Ramakrishnan, IV Ramakrishnan, Scott A Smolka, Yifei Dong, Xiaoqun Du, Abhik Roychoudhury, and VN Venkatakrishnan. XMC: A logic-programming-based verification toolset. In *International Conference on Computer Aided Verification*, pages 576–580. Springer, 2000.

[RS04] Kavita Ravi and Fabio Somenzi. Minimal assignments for bounded model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 31–45. Springer, 2004.

[Sch04] Pierre-Yves Schobbens. Alternating-time logic with imperfect recall. *Electronic Notes in Theoretical Computer Science*, 85(2):82 – 93, 2004.

[SG07] Sharon Shoham and Orna Grumberg. A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. *ACM Transactions on Computational Logic (TOCL)*, 9(1):1, 2007.

[SQL05a] Shengyu Shen, Ying Qin, and Sikun Li. Counterexample minimization for ACTL. In *CHARME*, volume 5, pages 393–397, 2005.

[SQL05b] ShengYu Shen, Ying Qin, and SiKun Li. Minimizing counterexample of ACTL property. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 393–397. Springer, 2005.

[SS98] Perdita Stevens and Colin Stirling. Practical model-checking using games. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 85–101. Springer, 1998.

[Sti95] Colin Stirling. Local model checking games. In *International Conference on Concurrency Theory*, pages 1–11. Springer, 1995.

[SW91] Colin Stirling and David Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, 1991.

[Tan04] Li Tan. PlayGame: A platform for diagnostic games. In *International Conference on Computer Aided Verification*, pages 492–495. Springer, 2004.

[TC02] Li Tan and Rance Cleaveland. Evidence-based model checking. In *International Conference on Computer Aided Verification*, pages 455–470. Springer, 2002.

[Tho95] Wolfgang Thomas. On the synthesis of strategies in infinite games. In Ernst W. Mayr and Claude Puech, editors, *STACS 95*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer Berlin Heidelberg, 1995.

[vdHW02] Wiebe van der Hoek and Michael Wooldridge. Tractable multiagent planning for epistemic goals. In *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*, pages 1167–1174, 2002.

[vdHW03] Wiebe van der Hoek and Michael Wooldridge. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75:125–157, 2003.

[vDK14] Hans van Ditmarsch and Sophia Knight. Partial information and uniform strategies. In Nils Bulling, Leendert van der Torre, Serena Villata, Wojtek Jamroga, and Wamberto Vasconcelos, editors, *Computational Logic in Multi-Agent Systems*, volume 8624 of *Lecture Notes in Computer Science*, pages 183–198. Springer International Publishing, 2014.

[vDK15] Hans van Ditmarsch and Barteld Kooi. One hundred prisoners and a light bulb. In *One Hundred Prisoners and a Light Bulb*, pages 83–94. Springer International Publishing, 2015.

[vdMS99] Ron van der Meyden and Nikolay V Shilov. Model checking knowledge and time in systems with perfect recall. In *Foundations of Software Technology and Theoretical Computer Science*, pages 432–445. Springer, 1999.

[vOJ05] Sieuwert van Otterloo and Geert Jonker. On epistemic temporal strategic logic. *Electr. Notes Theor. Comput. Sci.*, 126:77–92, 2005.

[WN10] Franz Weitl and Shin Nakajima. Incremental construction of counterexamples in model checking web documents. In *WWV*, pages 34–50, 2010.

[WNF10] Franz Weitl, Shin Nakajima, and Burkhard Freitag. Structured counterexamples for the temporal description logic alcctl. In *2010 8th IEEE International Conference on Software Engineering and Formal Methods*, pages 232–243. IEEE, 2010.

[ZJC11] Yang Zhao, Xiaoqing Jin, and Gianfranco Ciardo. A symbolic algorithm for shortest EG witness generation. In *Fifth International Symposium on Theoretical Aspects of Software Engineering (TASE), 2011*, pages 68–75. IEEE, 2011.

# *Appendix A*

# *Model checking uniform strategies: correctness of the approaches*

This appendix proves the correctness of the model-checking approaches for $ATLK_{irF}$ presented in Chapters 5 and 6. Its structure follows the structure of the two chapters.

## A.1 Checking individual strategies

This section proves the correctness of the $filter_{⟪\Gamma⟫}$ algorithms described in Section 5.1. First, let

$$Pre_{⟪\Gamma⟫}(Q', M_\Gamma) = \left\{ q' \in Q \;\middle|\; \begin{array}{l} \exists \langle q, a_\Gamma \rangle \in M_\Gamma \text{ s.t.} \\ q' = q \wedge \forall a \in E_{Ag}(q), \\ a_\Gamma \sqsubseteq a \implies \delta(q,a) \in Q' \end{array} \right\}.$$

$Pre_{⟪\Gamma⟫}(Q', M_\Gamma) \cap M_\Gamma|_Q$ computes the states $q \in M_\Gamma|_Q$ such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ s.t. $\forall \pi \in out(f_\Gamma, q), \pi(1) \in Q'$. In the sequel, we say that *a strategy $f_\Gamma$ forces $X$ from a state $q \in Q$*, for a given path condition $X$, iff $\forall \pi \in out(f_\Gamma, q)$, $\pi$ satisfies $X$. Thus, $Pre_{⟪\Gamma⟫}(Q', M_\Gamma) \cap M_\Gamma|_Q$ computes the states of $M_\Gamma|_Q$ such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ forcing to reach $Q'$ in one step. This is captured by the following lemma.

**Lemma A.1.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC \rangle$, a subset of agents $\Gamma \subseteq Ag$, a subset of states $Q' \subseteq Q$, and a closed set of $\Gamma$-moves $M_\Gamma$, $Pre_{⟪\Gamma⟫}(Q', M_\Gamma) \cap M_\Gamma|_Q$ is the subset of states $q \in M_\Gamma|_Q$*

*such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ forcing to reach $Q'$ in one step from $q$.*

*Proof.* First, suppose that $q \in Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(Q', M_\Gamma) \cap M_\Gamma|_Q$. So there exists $\langle q, a_\Gamma\rangle \in M_\Gamma$ such that $\forall a \in E_{Ag}(q), a_\Gamma \sqsubseteq a \implies \delta(q,a) \in Q'$. So there exists $f_\Gamma$ such that $f_\Gamma(q) = a_\Gamma$ and that forces to reach $Q'$ in one step. So there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$—$f_\Gamma(q) = a_\Gamma$ and it can make any compatible decision in other states—that forces to reach $Q'$ in one step from $q$, and this direction is proved.

Second, suppose that $q \in M_\Gamma|_Q$ is such that there exists $f_\Gamma$ compatible with $M_\Gamma$ that forces to reach $Q'$ in one step from $q$. Then $f_\Gamma$ is such that $\forall a \in E_{Ag}(q), f_\Gamma(q) \sqsubseteq a \implies \delta(q,a) \in Q'$ by definition of *out*, and $\langle q, f_\Gamma(q)\rangle \in M_\Gamma$ as $q \in M_\Gamma|_Q$ and $f_\Gamma$ is compatible with $M_\Gamma$. So there exists $\langle q, a_\Gamma\rangle \in M_\Gamma$ s.t. $q' = q \wedge \forall a \in E_{Ag}(q), a_\Gamma \sqsubseteq a \implies \delta(q,a) \in Q'$, thus $q \in Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(Q', M_\Gamma) \cap M_\Gamma|_Q$, and the proof is done. □

Second, let

$$Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(Q_1, Q_2, M_\Gamma) = \nu Q'.\ Q_2 \cup \left(Q_1 \cap Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(Q', M_\Gamma)\right).$$

$Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ returns the states $q \in M_\Gamma|_Q$ such that there exists $f_\Gamma$ compatible with $M_\Gamma$ forcing, from $q$, to reach $Q_2$ through $Q_1$ or staying in $Q_1$ forever. Its correctness is captured by the following lemma.

**Lemma A.2.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a subset of agents $\Gamma \subseteq Ag$, two subsets of states $Q_1, Q_2 \subseteq Q$, and a closed set of $\Gamma$-moves $M_\Gamma$, $Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ is the subset of states $q \in M_\Gamma|_Q$ such that there exists $f_\Gamma$ compatible with $M_\Gamma$ forcing, from $q$, to reach $Q_2$ through $Q_1$ or to stay in $Q_1$ forever.*

*Proof.* Let $\tau(Z) = Q_2 \cup (Q_1 \cap Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(Z, M_\Gamma))$. Let us prove, by induction on $i$, that, for all $i \geq 0$, $\tau^i(Q) \cap M_\Gamma|_Q$ is the subset of states $q \in M_\Gamma|_Q$ such that there exists $f_\Gamma$ compatible with $M_\Gamma$ forcing to reach $Q_2$ through $Q_1$ in at most $i-1$ steps, or to stay in $Q_1$ for at least $i$ steps, from $q$.

**Base case** $\tau^0(Q) \cap M_\Gamma|_Q = M_\Gamma|_Q$ and all strategies $f_\Gamma$ compatible with $M_\Gamma$ trivially force to stay in $Q_1$ for at least 0 steps. Thus the base case is proved.

**Inductive case** Let us suppose that $\tau^i(Q) \cap M_\Gamma|_Q$ is the subset of states $q \in M_\Gamma|_Q$ such that there exists $f_\Gamma$ compatible with $M_\Gamma$ forcing to reach $Q_2$ through $Q_1$ in at most $i-1$ steps, or to stay in $Q_1$ for at least $i$ steps, from $q$.

Let us show that $\tau^{i+1}(Q) \cap M_\Gamma|_Q$ is the subset of states $q \in M_\Gamma|_Q$ such that there exists $f_\Gamma$ compatible with $M_\Gamma$ forcing to reach $Q_2$ through $Q_1$ in at most $i$ steps, or to stay in $Q_1$ for at least $i+1$ steps, from $q$.

$$\begin{aligned}&\tau^{i+1}(Q) \cap M_\Gamma|_Q = (Q_2 \cup (Q_1 \cap Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(\tau^i(Q), M_\Gamma))) \cap M_\Gamma|_Q\\ &= (Q_2 \cap M_\Gamma|_Q) \cup (Q_1 \cap M_\Gamma|_Q \cap Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(\tau^i(Q), M_\Gamma))\\ &= (Q_2 \cap M_\Gamma|_Q) \cup (Q_1 \cap M_\Gamma|_Q \cap Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(\tau^i(Q) \cap M_\Gamma|_Q, M_\Gamma)).\end{aligned}$$

The last step is correct because $M_\Gamma$ is closed, so only the states of $M_\Gamma|_Q$ influence the result of $Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(\tau^i(Q), M_\Gamma)$.

Let us suppose that $q \in \tau^{i+1}(Q) \cap M_\Gamma|_Q$. Then $q \in M_\Gamma|_Q$, and either $q \in Q_2 \cap M_\Gamma|_Q$ or $q \in Q_1 \cap M_\Gamma|_Q \cap Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(\tau^i(Q) \cap M_\Gamma|_Q, M_\Gamma)$. In the former case, any strategy compatible with $f_\Gamma$ forces to reach $Q_2$ through $Q_1$ in 0 steps.

In the latter case, $q \in Q_1 \cap \in M_\Gamma|_Q \cap Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(\tau^i(Q) \cap M_\Gamma|_Q, M_\Gamma)$. Thus $q \in Q_1$, $q \in M_\Gamma|_Q$ and there exists $f_\Gamma$ compatible with $M_\Gamma$ that forces to reach $\tau^i(Q) \cap M_\Gamma|_Q$ in one step from $q$, by Lemma A.1. So, $q \in Q_1$, $q \in M_\Gamma|_Q$ and, by the inductive hypothesis, there exists $f_\Gamma$ compatible with $M_\Gamma$ that forces to reach states $q' \in M_\Gamma|_Q$ such that there exists $f'_\Gamma$ compatible with $M_\Gamma$ forcing to reach $Q_2$ through $Q_1$ in at most $i-1$ steps, or to stay in $Q_1$ for at least $i$ steps, from $q'$. This means that $q \in M_\Gamma|_Q$ and there exists $f''_\Gamma$ compatible with $M_\Gamma$ forcing to reach $Q_2$ through $Q_1$ in at most $i$ steps, or to stay in $Q_1$ for at least $i+1$ steps, from $q$: $f''_\Gamma$ follows the choice of $f_\Gamma$ in $q$, then $f'_\Gamma$. As $q \in Q_1$, all paths enforced by $f''_\Gamma$ reach $Q_2$ through $Q_1$ in at most $i$ steps, or stay in $Q_1$ for at least $i+1$ steps, and this direction is proved.

For the other direction, let us suppose that $q \in M_\Gamma|_Q$ is such that there exists $f_\Gamma$ compatible with $M_\Gamma$ forcing to reach $Q_2$ through $Q_1$ in at most $i$ steps, or to stay in $Q_1$ for at least $i+1$ steps, from $q$. Thus either $q \in Q_2$, or $q \notin Q_2$, $q \in Q_1$, and there exists $f_\Gamma$ compatible with $M_\Gamma$ forcing to reach $Q_2$ through $Q_1$ in at least one and at most $i$ steps, or to stay in $Q_1$ for at least $i+1$ steps, from $q$.

In the former case, $q \in Q_2 \cap M_\Gamma|_Q$, thus $q \in \tau^{i+1}(Q) \cap M_\Gamma|_Q$. In the latter case, $q \in Q_1 \cap M_\Gamma|_Q$ and $f_\Gamma$ reaches in one step, from $q$, states $q' \in M_\Gamma|_Q$ (because $M_\Gamma$ is closed) such that there exists $f'_\Gamma$ compatible with $M_\Gamma$—$f'_\Gamma$ follows $f_\Gamma$—forcing to reach $Q_2$ through $Q_1$ in at most $i-1$ steps, or to stay in $Q_1$ for at least $i$ steps, from $q'$. So, $q$ belongs to $Q_1 \cap M_\Gamma|_Q$ and, by induction hypothesis, there exists $f_\Gamma$ compatible with $M_\Gamma$ forcing to reach states of $\tau^i(Q) \cap M_\Gamma|_Q$ in one step from $q$. Thus $q \in Q_1 \cap M_\Gamma|_Q \cap Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(\tau^i(Q) \cap M_\Gamma|_Q, M_\Gamma)$ by Lemma A.1, and this direction is proved, too.

Finally, $Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q = \bigcup_i \tau^i(Q) \cap M_\Gamma|_Q$. This implies that $Stay_{\langle\langle\Gamma\rangle\rangle}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ is the subset of states $q \in M_\Gamma|_Q$ such that there exists $f_\Gamma$ compatible with $M_\Gamma$ forcing to reach $Q_2$ through $Q_1$ or to stay in $Q_1$ forever, from $q$. □

From $Pre_{\langle\langle\Gamma\rangle\rangle}$ and $Stay_{\langle\langle\Gamma\rangle\rangle}$, the $NFair_{\langle\langle\Gamma\rangle\rangle}$ function is defined as

$$NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma) = \mu Q'. \bigcup_{fc \in FC} Pre_{\langle\langle\Gamma\rangle\rangle}\left(Stay_{\langle\langle\Gamma\rangle\rangle}(Q' \cup \overline{fc}, \varnothing, M_\Gamma), M_\Gamma\right).$$

It returns the set of states $q$ such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ that forces unfair paths from $q$. To prove its correctness, we need the following lemma. It says that in every state $q'$ reached by a strategy $f_\Gamma$ forcing unfair paths from some state $q$, $f_\Gamma$ forces unfair paths from $q'$, too.

**Lemma A.3.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a subset of agents $\Gamma \subseteq Ag$, a closed set of $\Gamma$-moves $M_\Gamma$, a state $q \in M_\Gamma|_Q$, and a strategy $f_\Gamma$ compatible with $M_\Gamma$ that forces unfair paths from $q$, then*

$$\forall \pi \in out(f_\Gamma, q), \forall i \geq 0, \forall \pi' \in out(\pi(i), f_\Gamma), \pi' \textit{ is unfair.}$$

*Proof.* Let us suppose that $f_\Gamma$ forces unfair paths from $q$, and that

$$\exists \pi \in out(f_\Gamma, q), \exists i \geq 0, \exists \pi' \in out(\pi(i), f_\Gamma), \text{ s.t. } \pi' \text{ is fair.}$$

Then the outcomes of $f_\Gamma$ contain a path $\pi''$ that is fair: $\pi''$ follows $\pi$ up to $\pi(i)$, then follows $\pi'$ that is fair. Thus $f_\Gamma$ does not force unfair paths, and we reach a contradiction. □

The correctness of $NFair$ is then captured by the following lemma.

**Lemma A.4.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a subset of agents $\Gamma \subseteq Ag$, and a closed set of $\Gamma$-moves $M_\Gamma$, the result of $NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma) \cap M_\Gamma|_Q$ is the subset of states $q \in M_\Gamma|_Q$ such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ that forces unfair paths from $q$.*

*Proof.* To prove this lemma, let

$$\tau(Z) = \bigcup_{fc \in FC} Pre_{\langle\langle\Gamma\rangle\rangle}\left(Stay_{\langle\langle\Gamma\rangle\rangle}(Z \cup \overline{fc}, \varnothing, M_\Gamma), M_\Gamma\right).$$

Let also $NF$ be the subset of states $q \in M_\Gamma|_Q$ such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ that forces unfair paths from $q$.

Let us prove that

1. $NF$ is a fixpoint of $\tau$.
2. $NF$ is contained in any fixpoint of $\tau$.

This leads to the fact that $NF$ is the least fixpoint of $\tau$, and the proof is done.

$NF$ **is a fixpoint of** $\tau$ Let us suppose that $q \in NF$. That is, there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ that forces unfair paths from $q$. By Lemma A.3, all states reached by $f_\Gamma$ from $q$ are in $NF$. Thus $f_\Gamma$ forces to stay in $NF$. So, there exists $fc \in FC$ such that $f_\Gamma$ forces to stay in $NF \cup \overline{fc}$. So $q \in Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(NF \cup \overline{fc}, \varnothing, M_\Gamma)$. Furthermore, by definition of $Stay_{\langle\!\langle\Gamma\rangle\!\rangle}$, $q \in Pre_{\langle\!\langle\Gamma\rangle\!\rangle}\big(Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(NF \cup \overline{fc}, \varnothing, M_\Gamma), M_\Gamma\big)$. So $q \in \bigcup_{fc \in FC} Pre_{\langle\!\langle\Gamma\rangle\!\rangle}\big(Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(NF \cup \overline{fc}, \varnothing, M_\Gamma), M_\Gamma\big)$.

For the other direction, let us suppose that $q \in \tau(NF)$. That is, $q \in \bigcup_{fc \in FC} Pre_{\langle\!\langle\Gamma\rangle\!\rangle}\big(Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(NF \cup \overline{fc}, \varnothing, M_\Gamma), M_\Gamma\big)$. Thus there exists $fc \in FC$ such that $q \in Pre_{\langle\!\langle\Gamma\rangle\!\rangle}\big(Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(NF \cup \overline{fc}, \varnothing, M_\Gamma), M_\Gamma\big)$, that is, there exists $fc \in FC$ such that there exists $f_\Gamma$ compatible with $M_\Gamma$ that forces to stay in $\overline{fc} \cup NF$ forever. Furthermore, in every state $q'$ reached by $f_\Gamma$ that belongs to $NF$, there exists a strategy $f'_\Gamma$ that forces unfair paths from $q'$. Thus, the strategy $f''_\Gamma$ that follows $f_\Gamma$ up to a state of $NF$, then follows $f'_\Gamma$ afterwards forces paths that (1) either stay in $\overline{fc}$ if $NF$ is never reached, or (2) are unfair as they end with a unfair path forced by $f'_\Gamma$ from $q'$. Thus there exists a strategy $f''_\Gamma$ that forces unfair paths from $q$, and $q \in NF$.

$NF$ **is contained in any fixpoint of** $\tau$**.** Let $Z$ be a fixpoint of $\tau$, that is, $Z = \tau(Z)$. Let us show that $Z \supseteq NF$ by showing that $\overline{Z} \subseteq \overline{NF}$. If $q$ is not in $Z$, then $q \notin \tau(Z)$, that is,

$$q \notin \bigcup_{fc \in FC} Pre_{\langle\!\langle\Gamma\rangle\!\rangle}\big(Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(Z \cup \overline{fc}, \varnothing, M_\Gamma), M_\Gamma\big).$$

Thus $q \in \bigcap_{fc \in FC} Pre_{[\![\Gamma]\!]}\big(Reach_{[\![\Gamma]\!]}(Q, \overline{Z} \cap fc, M_\Gamma), M_\Gamma\big)$, by duality.

Thus, every strategy $f_\Gamma$ compatible with $M_\Gamma$ cannot avoid a path, from $q$, that reaches $q' \in fc \cap \overline{Z}$, for any $fc \in FC$. As $q' \in \overline{Z}$, every strategy $f'_\Gamma$ compatible with $M_\Gamma$—including $f_\Gamma$ above—cannot avoid a path, from $q'$, that reaches $q'' \in fc' \cap \overline{Z}$, for any other $fc' \in FC$. This argument can be repeated indefinitely, thus every $f_\Gamma$ compatible with $M_\Gamma$ cannot avoid a path that go through all fairness constraints infinitely often, that is, a fair path. So there is no strategy $f_\Gamma$ that can forces unfair paths from $q$, so $q \notin NF$. Thus $\overline{Z} \subseteq \overline{NF}$, and $NF \subseteq Z$.

Finally, as $Z$ is any fixpoint of $\tau$, its least fixpoint is contained in $NF$, so $NF$ is the least fixpoint of $\tau$ and the proof is done. □

The $filter_{\langle\!\langle\Gamma\rangle\!\rangle}$ algorithms are defined as

$$filter_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q', M_\Gamma) = Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(Q' \cup NFair_{\langle\!\langle\Gamma\rangle\!\rangle}(M_\Gamma), M_\Gamma),$$

$$\begin{aligned}
&filter_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, M_\Gamma) = \\
&\mu Q'.\ Q_{1,2,N} \cap (Q_2 \cup \\
&\bigcup_{fc \in FC} Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(Q_{1,2,N} \cap (Q' \cup \overline{fc}), Q_2 \cap (Q' \cup \overline{fc}), M_\Gamma), M_\Gamma)), \\
&filter_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}(Q_1, Q_2, M_\Gamma) = Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(Q_{1,2,N}, Q_2, M_\Gamma),
\end{aligned}$$

where

$$Q_{1,2,N} = Q_1 \cup Q_2 \cup NFair_{\langle\!\langle\Gamma\rangle\!\rangle}(M_\Gamma).$$

They return the set of states $q$ such that there exists a strategy $f_\Gamma$ such that all fair paths forced from $q$ have the second state in $Q'$, reach $Q_2$ through $Q_1$, or either reach $Q_2$ through $Q_1$ or stay in $Q_1$ forever, respectively. This is captured by the three following theorems.

**Theorem A.5.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a subset of agents $\Gamma \subseteq Ag$, a subset of states $Q' \subseteq Q$, and a closed set of $\Gamma$-moves $M_\Gamma$, $filter_{\langle\!\langle\Gamma\rangle\!\rangle\boldsymbol{X}}(Q', M_\Gamma) \cap M_\Gamma|_Q$ is the subset of states $q \in M_\Gamma|_Q$ such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ such that all fair paths enforced by $f_\Gamma$ from $q$ have their second state in $Q'$, that is, $\forall \pi \in out(f_\Gamma, q), \pi$ is fair $\implies \pi(1) \in Q'$.*

*Proof.* The proof is trivial by definition of $Pre_{\langle\!\langle\Gamma\rangle\!\rangle}$ and Lemma A.4. □

To prove the correctness of the $filter_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}$ algorithm, we need an intermediate result, given by the following lemma.

**Lemma A.6.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a subset of agents $\Gamma \subseteq Ag$, two subsets of states $Q_1, Q_2 \subseteq Q$, and a closed set of $\Gamma$-moves $M_\Gamma$, if, for some strategy $f_\Gamma$ compatible with $M_\Gamma$ and state $q \in M_\Gamma|_Q$, for all paths $\pi \in out(f_\Gamma, q)$, $\pi$ is unfair or $\pi$ reaches a state of $Q_2$ through states of $Q_1$, then for all paths $\pi \in out(f_\Gamma, q)$, $\pi$ reaches $Q_2$ through $Q_1 \cup NFair_{\langle\!\langle\Gamma\rangle\!\rangle}$ or stays in $Q_1 \cup NFair_{\langle\!\langle\Gamma\rangle\!\rangle}$.*

*Proof.* Let $q \in M_\Gamma|_Q$ and $f_\Gamma$ a strategy compatible with $M_\Gamma$. Let us suppose that there exists a path $\pi \in out(f_\Gamma, q)$ and an index $j \geq 0$ such that there is no index $j' \leq j$ such that $\pi(j') \in Q_2$, and $\pi(j) \notin Q_1 \cup NFair_{\langle\!\langle\Gamma\rangle\!\rangle}$. That is, $\pi(j) \in \overline{Q_1} \cap Fair_{[\![\Gamma]\!]}$. So $\pi$ does not reach $Q_2$ through $Q_1$ or stays in $Q_1$ forever. Furthermore, as $\pi(j) \in Fair_{[\![\Gamma]\!]}$, $f_\Gamma$ cannot avoid a fair path. Thus $f_\Gamma$ cannot avoid a fair path that does not reach $Q_2$ through $Q_1$ or stays in $Q_1$, and the proof is done. □

Then, the correctness of $filter_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}$ is given by the following theorem.

**Theorem A.7.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$*, a subset of agents* $\Gamma \subseteq Ag$*, two subsets of states* $Q_1, Q_2 \subseteq Q$*, and a closed set of* $\Gamma$*-moves* $M_\Gamma$*,* $filter_{\langle\!\langle\Gamma\rangle\!\rangle U}(Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ *is the subset of states* $q \in M_\Gamma|_Q$ *such that there exists a strategy* $f_\Gamma$ *compatible with* $M_\Gamma$ *such that all fair paths forced from* $q$ *by* $f_\Gamma$ *reach a state of* $Q_2$ *through states of* $Q_1$*, that is,* $\forall \pi \in out(f_\Gamma, q), \pi$ *is fair* $\implies \exists i \geq 0$ *s.t.* $\pi(i) \in Q_2$ *and* $\forall j, 0 \leq j < i, \pi(j) \in Q_1$.

*Proof.* To prove this lemma, let

$$\tau(Z) = Q_{1,2,N} \cap (Q_2 \cup \bigcup_{fc \in FC} Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(Q_{1,2,N} \cap (Z \cup \overline{fc}), Q_2 \cap (Z \cup \overline{fc}), M_\Gamma), M_\Gamma)),$$

where

$$Q_{1,2,N} = Q_1 \cup Q_2 \cup NFair_{\langle\!\langle\Gamma\rangle\!\rangle}(M_\Gamma).$$

Let also $NFU$ be the subset of states $q \in M_\Gamma|_Q$ such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ such that all fair paths forced from $q$ by $f_\Gamma$ reach a state of $Q_2$ through states of $Q_1$.

Let us prove that

1. $NFU$ is a fixpoint of $\tau$.

2. $NFU$ is contained in any fixpoint of $\tau$.

This leads to the fact that $NFU$ is the least fixpoint of $\tau$, and the proof is done.

**$NFU$ is a fixpoint of $\tau$** First, let us suppose that $q \in NFU$. So there exists $f_\Gamma$ compatible with $M_\Gamma$ s.t. $\forall \pi \in out(f_\Gamma, q)$, if $\pi$ is fair, then $\pi$ reaches $Q_2$ through $Q_1$.

If $q \in Q_2$, then $q \in \tau(NFU)$. Also, if $\forall \pi \in out(f_\Gamma, q)$ $\pi$ is not fair, then $q \in NFair_{\langle\!\langle\Gamma\rangle\!\rangle}(M_\Gamma)$. In this case, $f_\Gamma$ forces to reach in one step states in which $f_\Gamma$ forces paths that stay in $NFair_{\langle\!\langle\Gamma\rangle\!\rangle}(M_\Gamma)$ by definition of $NFair_{\langle\!\langle\Gamma\rangle\!\rangle}(M_\Gamma)$, that is,

$$q \in Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(NFair_{\langle\!\langle\Gamma\rangle\!\rangle}(M_\Gamma), \varnothing, M_\Gamma), M_\Gamma).$$

Thus $q$ belongs to

$$\bigcup_{fc \in FC} Pre_{\langle\!\langle\Gamma\rangle\!\rangle}\left(Stay_{\langle\!\langle\Gamma\rangle\!\rangle}\left(\begin{array}{c} Q_{1,2,N} \cap (NFU \cup \overline{fc}), \\ Q_2 \cap (NFU \cup \overline{fc}), M_\Gamma \end{array}\right), M_\Gamma\right),$$

by monotonicity. We can conclude that $q \in \tau(NFU)$.

If $q$ misses the two cases above, then $q \in Q_1$ and $\forall \pi \in out(f_\Gamma, q)$, if $\pi$ is fair then it reaches $Q_2$ through $Q_1$. By Lemma A.6, $q$ belongs to $Stay_{\langle\langle\Gamma\rangle\rangle}(Q_1 \cup NFair_{\langle\langle\Gamma\rangle\rangle}, Q_2, M_\Gamma)$, thus $q$ belongs to

$$\bigcup_{fc \in FC} Pre_{\langle\langle\Gamma\rangle\rangle}\left(Stay_{\langle\langle\Gamma\rangle\rangle}\left(\begin{array}{c} Q_{1,2,N} \cap (NFU \cup \overline{fc}), \\ Q_2 \cap (NFU \cup \overline{fc}), M_\Gamma \end{array}\right), M_\Gamma\right),$$

by definition of $Stay$ and monotonicity. So $q \in \tau(NFU)$, and this direction is proved.

For the other direction, let us suppose that $q \in \tau(NFU)$. If $q \in Q_2$, then $q \in NFU$. Otherwise, if $q$ belongs to

$$NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma) \cap$$
$$\bigcup_{fc \in FC} Pre_{\langle\langle\Gamma\rangle\rangle}\left(Stay_{\langle\langle\Gamma\rangle\rangle}\left(\begin{array}{c} Q_{1,2,N} \cap (NFU \cup \overline{fc}), \\ Q_2 \cap (NFU \cup \overline{fc}), M_\Gamma \end{array}\right), M_\Gamma\right),$$

then it belongs to $NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma)$. Thus it belongs to

$$Pre_{\langle\langle\Gamma\rangle\rangle}(Stay_{\langle\langle\Gamma\rangle\rangle}(NFair_{\langle\langle\Gamma\rangle\rangle}, \varnothing, M_\Gamma), M_\Gamma)$$

by definition of $NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma)$, and so it belongs to

$$\bigcup_{fc \in FC} Pre_{\langle\langle\Gamma\rangle\rangle}\left(Stay_{\langle\langle\Gamma\rangle\rangle}\left(\begin{array}{c} Q_{1,2,N} \cap (NFU \cup \overline{fc}), \\ Q_2 \cap (NFU \cup \overline{fc}), M_\Gamma \end{array}\right), M_\Gamma\right),$$

by monotonicity. Thus $q \in NFU$.

Finally, if $q \notin Q_2$ and $q \notin NFair_{\langle\langle\Gamma\rangle\rangle}$, then $q$ belongs to

$$Q_1 \cap \bigcup_{fc \in FC} Pre_{\langle\langle\Gamma\rangle\rangle}\left(Stay_{\langle\langle\Gamma\rangle\rangle}\left(\begin{array}{c} Q_{1,2,N} \cap (NFU \cup \overline{fc}), \\ Q_2 \cap (NFU \cup \overline{fc}), M_\Gamma \end{array}\right), M_\Gamma\right).$$

So there exist $fc \in FC$ and $f_\Gamma$ compatible with $M_\Gamma$ such that all paths $\pi$ forced by $f_\Gamma$ from $q$ reach $Q_2 \cap (NFU \cup \overline{fc})$ through $Q_{1,2,N} \cap (NFU \cup \overline{fc})$, or stay in $Q_{1,2,N} \cap (NFU \cup \overline{fc})$ forever. That is, $\pi$ reaches $Q_2 \cap (NFU \cup \overline{fc})$ through $(Q_1 \cup NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma)) \cap (NFU \cup \overline{fc})$, or stays in $(Q_1 \cup NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma)) \cap (NFU \cup \overline{fc})$ forever.

Let $\pi$ be such a path forced by $f_\Gamma$ from $q$, $j \geq 0$ be the smallest index such that $\pi(j) \in Q_2 \cup NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma) \cup NFU$. $\forall j' < j, \pi(j') \in Q_1 \cap \overline{fc}$. As $\pi(j) \in Q_2 \cup NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma) \cup NFU$, $\pi(j) \in NFU$, so there exists a strategy $f'_\Gamma$ compatible with $M_\Gamma$ that forces, from $\pi(j)$, paths that are either unfair or that reach $Q_2$ through $Q_1$. Furthermore, if there is no such index $j$, $\pi$ stays in $Q_1 \cap \overline{fc}$, so $\pi$ is unfair. Thus the strategy $f''_\Gamma$ that follows $f_\Gamma$ until reaching some state of $Q_2 \cup NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma) \cup NFU$, in which it follows $f'_\Gamma$, results in paths that are either unfair or that reach $Q_2$ through $Q_1$. So $q \in NFU$ and this other direction is proved.

$NFU$ **is contained in any fixpoint of $\tau$.** Let $Z$ be a fixpoint of $\tau$, that is, $Z = \tau(Z)$. Let us show that $Z \supseteq NFU$ by showing that $\overline{Z} \subseteq \overline{NFU}$. If $q \notin Z$, then $q \notin \tau(Z)$, that is,

$$q \notin Q_{1,2,N} \cap (Q_2 \cup \bigcup_{fc \in FC} Pre_{\langle\langle\Gamma\rangle\rangle}\left(Stay_{\langle\langle\Gamma\rangle\rangle}(Q_{1,2,N} \cap (Z \cup \overline{fc}), Q_2 \cap (Z \cup \overline{fc}), M_\Gamma\right), M_\Gamma)).$$

Thus

$$q \in Q_{\overline{1},\overline{2},F} \cup (\overline{Q_2} \cap \bigcap_{fc \in FC} Pre_{[\![\Gamma]\!]}(Reach_{[\![\Gamma]\!]}(\overline{Q_2}, Q_{\overline{1},\overline{2},F} \cup (\overline{Z} \cap fc), M_\Gamma), M_\Gamma)),$$

where

$$Q_{\overline{1},\overline{2},F} = (\overline{Q_1} \cap \overline{Q_2} \cap Fair_{[\![\Gamma]\!]}(M_\Gamma)),$$

by duality.

If $q \in Q_{\overline{1},\overline{2},F}$, then $q \in \overline{Q_1} \cap \overline{Q_2}$ and $q \in Fair_{[\![\Gamma]\!]}(M_\Gamma)$, so for all strategies $f_\Gamma$ compatible with $M_\Gamma$, $\exists \pi \in out(f_\Gamma, q)$ s.t. $\pi$ is fair and $\pi$ reaches $\overline{Q_1} \cap \overline{Q_2}$ through $\overline{Q_2}$ as the first state of $\pi$ is already in $\overline{Q_1} \cap \overline{Q_2}$. So $q \notin NFU$.

Otherwise, $q \in \overline{Q_2}$ and

$$q \in \bigcap_{fc \in FC} Pre_{[\![\Gamma]\!]}\left(Reach_{[\![\Gamma]\!]}\left(\overline{Q_2}, Q_{\overline{1},\overline{2},F} \cup (\overline{Z} \cap fc), M_\Gamma\right), M_\Gamma\right).$$

So, for all $f_\Gamma$ compatible with $M_\Gamma$, $\exists \pi \in out(f_\Gamma, q)$ s.t. $\pi$ reaches $Q_{\overline{1},\overline{2},F} \cup (\overline{Z} \cap fc)$ through $\overline{Q_2}$, for any $fc \in FC$, in at least one step. If $\pi$ reaches $Q_{\overline{1},\overline{2},F}$ through $\overline{Q_2}$, the arguments just above also apply, so $q \notin NFU$. In the other case, $\pi$ reaches $q' \in fc \cap \overline{Z}$ for some $fc \in FC$, so $q' \in \overline{Z}$. We can thus repeat this argument indefinitely, up to reaching $\overline{Q_1} \cap \overline{Q_2}$ through $\overline{Q_2}$, or reaching every $fc \in FC$ infinitely often through $\overline{Q_2}$. So for every strategy $f_\Gamma$ compatible with $M_\Gamma$, there exists $\pi \in out(f_\Gamma, q)$ such that $\pi$ is fair and reaches $\overline{Q_1} \cap \overline{Q_2}$ through $\overline{Q_2}$, or stays in $\overline{Q_2}$ forever, so $q \notin NFU$.

Thus $\overline{Z} \subseteq \overline{NFU}$, and $NFU \subseteq Z$.

Finally, as $Z$ is any fixpoint of $\tau$, its least fixpoint is contained in $NFU$, so $NFU$ is the least fixpoint of $\tau$ and the proof is done. □

The correctness of $filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}$ is given by the following theorem.

**Theorem A.8.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a subset of agents $\Gamma \subseteq Ag$, two subsets of states $Q_1, Q_2 \subseteq Q$, and a closed set of $\Gamma$-moves $M_\Gamma$, $filter_{\langle\langle\Gamma\rangle\rangle W}(\Gamma, Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$ is the subset of* *states $q \in M_\Gamma|_Q$ such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ such that all fair path forced by $f_\Gamma$ from $q$ reach a state of $Q_2$ through states of $Q_1$, or stay in $Q_1$ forever, that is, $\forall \pi \in out(f_\Gamma, q), \pi$ is fair $\implies \exists i \geq 0$ s.t. $\pi(i) \in Q_2$ and $\forall j, 0 \leq j < i, \pi(j) \in Q_1$, or $\forall i \geq 0, \pi(i) \in Q_1$.*

*Proof.* This theorem is easily proved thanks to Lemmas A.2 and A.4. Indeed, let us suppose that $q \in filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(\Gamma, Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$, that is, $q$ belongs to $Stay_{\langle\langle\Gamma\rangle\rangle}(Q_1 \cup Q_2 \cup NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma), Q_2, M_\Gamma) \cap M_\Gamma|_Q$. By Lemma A.2, $q$ is such that there exists a strategy $f_\Gamma$ compatible with $M_\Gamma$ such that, $\forall \pi \in out(f_\Gamma, q)$, $\pi$ reaches $Q_2$ through $Q_{1,2,N}$, or stays in $Q_{1,2,N}$ forever.

First, let us suppose that $\pi$ stays in $Q_{1,2,N}$ forever. Let $j \geq 0$ be the first index such that $\pi(j) \in Q_2 \cup NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma)$. Because $\pi$ stays in $Q_{1,2,N}$ forever, this means that, $\forall j' < j, \pi(j') \in Q_1$. Furthermore, either $\pi(j) \in Q_2$, so $\pi$ reaches $Q_2$ through $Q_1$, or $\pi(j) \in NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma)$ and, in this case, there exists $f'_\Gamma$ compatible with $M_\Gamma$ to forces only unfair paths from $\pi(j)$. Also, if there exists no such index $j$, $\pi$ stays in $Q_1$ forever.

Second, let us suppose that $\pi$ reaches $Q_2$ through $Q_{1,2,N}$. Let $j \geq 0$ be the first index such that $\pi(j) \in Q_2 \cup NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma)$. The arguments above also apply. We can thus conclude that if $q$ belongs to $filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(\Gamma, Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$, then there exists a strategy $f''_\Gamma$ compatible with $M_\Gamma$ that forces, from $q$, that every fair path reaches $Q_2$ through $Q_1$ or stays in $Q_1$ forever. $f''_\Gamma$ follows the strategy to stay in $Q_{1,2,N}$ or to reach $Q_2$ through $Q_{1,2,N}$, and switches to the strategy that forces unfair paths whenever it reaches a state of $NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma)$.

For the other direction, let us suppose that $q$ does not belong to $filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(\Gamma, Q_1, Q_2, M_\Gamma) \cap M_\Gamma|_Q$, that is, $q$ does not belong to the result of $Stay_{\langle\langle\Gamma\rangle\rangle}(Q_1 \cup Q_2 \cup NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma), Q_2, M_\Gamma) \cap M_\Gamma|_Q$. This means that, for all $f_\Gamma$ compatible with $M_\Gamma$, $\exists \pi \in out(f_\Gamma, q)$ s.t. $\pi$ reaches $\overline{Q_{1,2,N}} \cap \overline{Q_2}$ through $\overline{Q_2}$.

Let $\pi$ such a path. Let $j$ the first index such that $\pi(j) \in \overline{Q_{1,2,N}} \cap \overline{Q_2}$. That is, $\pi(j) \in \overline{Q_1} \cap \overline{Q_2} \cap \overline{NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma)}$. Because $\pi(j)$ belongs to $\overline{NFair_{\langle\langle\Gamma\rangle\rangle}(M_\Gamma)}$, it can be extended with a fair path forced by $f_\Gamma$, by Lemma A.4. Furthermore, as $j$ is the smallest index filling the conditions above, all the previous $j' < j$ are such that $\pi(j') \in \overline{Q_2}$. This means that there exists a fair path $\pi' \in out(f_\Gamma, q)$ that reaches $\overline{Q_1} \cap \overline{Q_2}$ through $\overline{Q_2}$. In other words, it is false that there exists $f_\Gamma$ compatible with $M_\Gamma$ that forces, from $q$, paths that are either unfair, reach $Q_2$ through $Q_1$, or stay in $Q_1$ forever. □

In the sequel, we sometimes abbreviate the three $filter_{\langle\langle\rangle\rangle}$ algorithms with the notation $filter_{op}(Q_1, Q_2, M_\Gamma)$, that depends on the operator

$op \in \{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}, \langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}, \langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}\}$.

## A.2 Enumerating all strategies

This section proves the correctness of the $eval_{ATLK_{irF}}$ algorithm defined in Section 5.2. Before proving its correctness, we need to prove that the *Split* algorithms are correct. These proofs are given by the two following theorems.

**Theorem A.9.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$*, a subset of agents* $\Gamma \subseteq Ag$*, an agent* $ag \in \Gamma$*, and a set of* $\Gamma$*-moves* $M_\Gamma$*,* $SplitAgent(ag, \Gamma, M_\Gamma)$ *is the set of largest subsets of non-ag-conflicting moves of* $M_\Gamma$*.*

*Proof.* We can prove the correctness of *SplitAgent* by induction over the number of *ag*-conflicting equivalence classes of $M_\Gamma$. Indeed, $conflicting$ contains the *ag*-conflicting moves of $M_\Gamma$. If $conflicting$ is empty, $M_\Gamma$ does not contain any conflicting equivalence classes, and $M_\Gamma$ is its own single largest subset in which no conflicts appear for *ag*.

Otherwise, *equivalent* is a set of conflicting moves of $M_\Gamma$ corresponding to a set of states indistinguishable for *ag*. Furthermore, *actions* are the possible actions for *ag* proposed by moves of *equivalent*.

Let us assume that $SplitAgent(ag, \Gamma, M_\Gamma \backslash equivalent)$ returns the set of all the largest non-*ag*-conflicting subsets of $M_\Gamma \backslash equivalent$. Then, the result of $SplitAgent(ag, \Gamma, M_\Gamma)$ is the cartesian product between all the largest non-*ag*-conflicting subsets of *equivalent*—that is, the *equivsubset* subsets—and all the largest non-*ag*-conflicting subsets of $M_\Gamma \backslash equivalent$—the *ncsubset* subsets. Because these cannot be conflicting as they belong to different equivalence classes of *ag* observations, we can conclude that *SplitAgent* returns the set of the largest non-*ag*-conflicting subsets of $M_\Gamma$. □

From the proof of correctness of the *SplitAgent* algorithm, we can prove the correctness of the *Split* one, given by the following theorem.

**Theorem A.10.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$*, a subset of agents* $\Gamma \subseteq Ag$*, and a set of* $\Gamma$*-moves* $M_\Gamma$*,* $Split(\Gamma, M_\Gamma)$ *returns the set of largest subsets of non-*$\Gamma$*-conflicting moves of* $M_\Gamma$*.*

*Proof.* We can prove the correctness of the *Split* algorithm by showing that the invariant of the **for** loop is that *subsets* is the set of largest subsets of non-$\Gamma'$-conflicting moves of $M_\Gamma$, where $\Gamma'$ is the subset from which *ag* has already taken the value.

First, the invariant is true before entering the loop the first time. Indeed, in this case $subsets = \{M_\Gamma\}$, $\Gamma' = \varnothing$ as the algorithm has not passed in the loop yet. $\{M_\Gamma\}$ is effectively the set of largest subsets of non-$\Gamma'$-conflicting as $M_\Gamma$ is trivially non-$\varnothing$-conflicting.

Second, suppose that the invariant is true before executing the loop body, and that there exists an agent $ag \in \Gamma$ that has not been chosen in the loop. Let $\Gamma' \subset \Gamma$ be the set of agents $ag'$ for which $ag$ has already taken the value. Since the invariant is true before executing the loop body, *subsets* is the set of largest subsets of non-$\Gamma'$-conflicting moves of $M_\Gamma$.

Then, the body of the loop splits in $subsets'$ each subset of *subsets* thanks to *SplitAgent*. After executing the loop body, $subsets'$—and thus *subsets*—is the set of all the largest subsets of non-$(\Gamma' \cup \{ag\})$-conflicting moves of $M_\Gamma$. Indeed, they are all non-conflicting subsets of $M_\Gamma$: each $subset \in subsets$ is non-$\Gamma'$-conflicting by hypothesis, thus each subset of $SplitAgent(ag, \Gamma, subset)$ is non-$\Gamma'$-conflicting as it is a subset of *subset*. Furthermore, each subset of $SplitAgent(ag, \Gamma, subset)$ is non-$ag$-conflicting, thus each $subset' \in subsets'$ is non-$(\Gamma' \cup \{ag\})$-conflicting.

Furthermore, each $subset' \in subsets'$ is a largest subset of non-$(\Gamma' \cup \{ag\})$-conflicting moves of $M_\Gamma$. Indeed, let suppose that some $subset'$ in $subsets'$ is not a largest subset of non-$(\Gamma' \cup \{ag\})$-conflicting moves of $M_\Gamma$. So there exists a move $\langle q, a_\Gamma \rangle \in M_\Gamma$ s.t. $subset' \cup \{\langle q, a_\Gamma \rangle\}$ is non-$(\Gamma' \cup \{ag\})$-conflicting. $subset'$ is a subset of some *subset* in *subsets* since $SplitAgent(ag, \Gamma, subset)$ returns subsets of *subset*. As $SplitAgent(ag, \Gamma, subset)$ returns all the largest non-$ag$-conflicting subsets of *subset*, $\langle q, a_\Gamma \rangle$ should be outside *subset*, otherwise *SplitAgent* would not miss it. But if $\langle q, a_\Gamma \rangle$ is outside *subset*, this means that *subset* is not a largest non-$\Gamma'$-conflicting subset of $M_\Gamma$, and this leads to a contradiction with the invariant hypothesis.

Also, $subsets'$ does not miss any largest non-$\Gamma'$-conflicting subset of $M_\Gamma$. Indeed, suppose there exists a subset $subset' \notin subsets'$ that is a largest subset of non-$(\Gamma' \cup \{ag\})$-conflicting moves of $M_\Gamma$. By construction, it cannot be a subset of some $subset \in subsets$ otherwise $SplitAgent(ag, \Gamma, subset)$ would have returned it. Since it is not a subset of some $subset \in subsets$ and it is non-$\Gamma'$-conflicting, it is a largest subset of non-$\Gamma'$-conflicting moves of $M_\Gamma$, and it should be in *subsets*, leading to a contradiction with the invariant hypothesis. So, after executing the loop body, $subsets'$—and thus *subsets*—contains the set of all the largest subsets of non-$(\Gamma' \cup \{ag\})$-conflicting moves of $M_\Gamma$, and the invariant is preserved.

Finally, when the loop is done, the invariant is still true and $\Gamma' = \Gamma$, thus *subsets* is the set of largest subsets of non-$\Gamma$-conflicting moves of

$M_\Gamma$, and the proof is done. □

Finally, the correctness of the $eval_{ATLK_{irF}}$ algorithm is given by the following theorem.

**Theorem A.11.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$*, and an* $ATLK_{irF}$ *strategic formula* $\langle\langle\Gamma\rangle\rangle\ \psi$*,* $eval_{ATLK_{irF}}(S, \langle\langle\Gamma\rangle\rangle\ \psi)$ *returns the states of* $S$ *satisfying* $\langle\langle\Gamma\rangle\rangle\ \psi$*.*

*Proof.* We can prove this theorem by induction over the syntactic structure of the checked formula. Given the strategic formula $\langle\langle\Gamma\rangle\rangle\ \psi$, we suppose that for all sub-formulas $\phi'$ of $\psi$, $eval_{ATLK_{irF}}(S, \phi')$ returns the states of $S$ satisfying $\phi'$, and we prove that $eval_{ATLK_{irF}}(S, \langle\langle\Gamma\rangle\rangle\ \psi)$ returns the states satisfying $\langle\langle\Gamma\rangle\rangle\ \psi$.

First, $Split(\Gamma, E_\Gamma)$ returns the set of uniform strategies for $\Gamma$. This is a corollary of Theorem A.10.

Second, let $F'_\Gamma$ be the set of strategies of which $f_\Gamma$ has already taken the value. Let us show that the invariant of the **for** loop is that *sat* is the set of states $q$ such that there exists a strategy $f_\Gamma$ in $F'_\Gamma$ such that,

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \models \psi.$$

The loop invariant is trivially satisfied before entering the loop for the first time as $F'_\Gamma$ and *sat* are empty. Furthermore, we can show that the loop body preserves the invariant by considering the three cases for $\psi$.

$\psi = \mathbf{X}\ \phi'$ The loop body computes $\Phi' = eval_{ATLK_{irF}}(S, \phi')$. By induction hypothesis, this is the set of states satisfying $\phi'$. Then it computes the set of states *winning* such that all fair paths enforced by the strategy $f_\Gamma$ satisfy $\mathbf{X}\ \phi'$, by Theorem A.5. The last statement of the **for** loop then adds in *sat* the set of states $q$ such that

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \models \mathbf{X}\ \phi',$$

and the invariant is still satisfied after the loop body.

$\psi = \phi_1\ \mathbf{U}\ \phi_2$ The loop body computes $\Phi_i = eval_{ATLK_{irF}}(S, \phi_i)$, for $i = 1$ and $i = 2$. By induction hypothesis, these are the sets of states satisfying $\phi_1$ (resp. $\phi_2$). Then it computes the set of states *winning* such that all fair paths enforced by $f_\Gamma$ satisfy $\phi_1\ \mathbf{U}\ \phi_2$, by Theorem A.7. The last statement of the **for** loop then adds in *sat* the set of states $q$ such that

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \models \phi_1\ \mathbf{U}\ \phi_2,$$

and the invariant is still satisfied after the loop body.

$\psi = \phi_1$ **W** $\phi_2$ This case is similar to the two others. The loop body computes $\Phi_i = eval_{ATLK_{irF}}(S, \phi_i)$, for $i = 1$ and $i = 2$. By induction hypothesis, these are the sets of states satisfying $\phi_1$ (resp. $\phi_2$). Then it computes the set of states *winning* such that all fair paths enforced by $f_\Gamma$ satisfy $\phi_1$ **W** $\phi_2$, by Theorem A.8. The last statement of the **for** loop then adds in *sat* the set of states $q$ such that

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \vDash \phi_1 \ \mathbf{W} \ \phi_2,$$

and the invariant is still satisfied after the loop body.

Finally, after the execution of the **for** loop, the invariant is satisfied and all uniform strategies for $\Gamma$ have been treated. Thus,

$$sat = \{q \in Q \mid \exists f_\Gamma \text{ s.t. } \forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \vDash \psi\},$$

and that is exactly the set of states satisfying $\langle\langle\Gamma\rangle\rangle \ \psi$. □

## A.3 Partial strategies

This section proves the correctness of the $eval^{Partial}_{ATLK_{irF}}$ algorithm described in Section 5.3. To prove its correctness, we need to prove that the algorithms it depends on are correct. The correctness of the *ReachSplit* algorithm is captured by the following theorem.

**Theorem A.12.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a group of agents $\Gamma \subseteq Ag$, and a set $M_\Gamma \subseteq E_\Gamma$ of non-$\Gamma$-conflicting $\Gamma$-moves, $ReachSplit(\Gamma, M_\Gamma)$ return the set of all the smallest uniform partial strategies extending $M_\Gamma$.*

*Proof.* If *new* is empty, then $M_\Gamma$ is already closed and thus is its own smallest uniform extension, as it is non-$\Gamma$-conflicting.

If *new* is not empty, $M_\Gamma \cup M'_\Gamma$ extends $M_\Gamma$ and is part of a closed extension since it contains moves reachable from $M_\Gamma$. Also, $M_\Gamma \cup M'_\Gamma$ is non-$\Gamma$-conflicting as $M_\Gamma$ is non-$\Gamma$-conflicting, $M'_\Gamma$ is composed of moves compatible with $M_\Gamma$, and contains no conflicting moves because it is the result of *Split*. So, since $M_\Gamma \cup M'_\Gamma$ extends $M_\Gamma$, $ReachSplit(\Gamma, M_\Gamma \cup M'_\Gamma)$ contains some smallest uniform partial strategies extending $M_\Gamma \cup M'_\Gamma$.

Finally, $ReachSplit(\Gamma, M_\Gamma)$ contains all the smallest uniform strategies extending $M_\Gamma$. Otherwise, a missing strategy $f_\Gamma$ would make a choice not taken into account by *Split*, and this is impossible since *Split* returns all the possible largest non-$\Gamma$-conflicting subsets of *compatible*, thus all combinations of uniform choices of *compatible*. □

Second, the correctness of the *PartialStrats* function is captured by the following theorem.

**Theorem A.13.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$*, a set of agents* $\Gamma$*, and a set of states* $Q' \subseteq Q$*,* $\mathit{PartialStrats}(\Gamma, Q')$ *is the set of smallest uniform partial strategies for* $\Gamma$ *adequate for* $Q'$*.*

*Proof.* Let us consider a smallest uniform partial strategy adequate for $Q'$ that is not in $\mathit{PartialStrats}(\Gamma, Q')$. Either this strategy proposes a choice for a state of $Q'$ that is not considered in any strategies of $\mathit{PartialStrats}(\Gamma, Q')$, thus *Split* should be incorrect, leading to a contradiction.

Or this strategy proposes a choice for a state outside $Q'$ that is not considered in any strategies $\mathit{PartialStrats}(\Gamma, Q')$, thus, for $M_\Gamma$ being the moves proposed by this strategy for states of $Q'$, the strategy is a smallest uniform partial strategy extending $M_\Gamma$, and thus *ReachSplit* should be incorrect since we found a smallest uniform partial strategy extending $f_\Gamma$ that is not in *ReachSplit*, leading to a contradiction. □

Finally, we can prove the correctness of $eval^{Partial}_{ATLK_{irF}}$ with the following theorem.

**Theorem A.14.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$*, a subset of states* $Q' \subseteq Q$*, and an* $ATLK_{irF}$ *strategic formula* $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$*,* $eval^{Partial}_{ATLK_{irF}}(S, Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi)$ *returns the subset of states of* $Q'$ *satisfying* $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$*.*

*Proof.* First, Theorems 5.1 and A.13 show that Line 3 of Algorithm 5.5 enumerates all uniform partial strategies $f_\Gamma$ that have to be checked to know which states of $Q'$ satisfy $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$.

Second, let $F'_\Gamma$ be the set of partial strategies of which $f_\Gamma$ has already taken the value during the **for** loop. Let us show that the invariant of this loop is that *sat* is the set of states $q$ such that there exists a uniform partial strategy $f_\Gamma$ in $F'_\Gamma$ such that,

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \models \psi.$$

This loop invariant is trivially satisfied before entering the loop for the first time as *sat* and $F'_\Gamma$ are empty. Furthermore, we can show that the loop body preserves this invariant. Indeed, Lines 4 to 14 compute the set of states of $Q'$ for which $f_\Gamma$ is winning. Let us consider each case separately.

$\psi = \mathbf{X}\ \phi'$ Line 6 computes the set of states for which the fair paths enforced by $f_\Gamma$ have their second state is a successor of some state in $[Q']^E_\Gamma$ that satisfies $\phi'$. Restricting the states satisfying $\phi'$ to the successors of $[Q']^E_\Gamma$ is sufficient because no state that is not a successor of $[Q']^E_\Gamma$ can make states of $[Q']^E_\Gamma$ satisfy $\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}\ \phi'$. So, *winning* is the states such that all fair paths enforced by $f_\Gamma$ satisfy $\mathbf{X}\ \phi'$, that is, the states $q$ such that

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \vDash \mathbf{X}\ \phi'.$$

$\psi = \phi_1\ \mathbf{U}\ \phi_2$ Line 10 computes the set of states for which the fair paths enforced by $f_\Gamma$ reach a state satisfying $\phi_2$ through states satisfying $\phi_1$. Again, restricting the states satisfying $\phi_i$ (for $i = 1$ and $i = 2$) to the ones in the domain of $f_\Gamma$ is sufficient. So, *winning* is the states such that all fair paths enforced by $f_\Gamma$ satisfy $\phi_1\ \mathbf{U}\ \phi_2$, that is, the states $q$ such that

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \vDash \phi_1\ \mathbf{U}\ \phi_2.$$

$\psi = \phi_1\ \mathbf{W}\ \phi_2$ Line 14 computes the set of states for which the fair paths enforced by $f_\Gamma$ reach a state satisfying $\phi_2$ through states satisfying $\phi_1$, or stay in states satisfying $\phi_1$ forever. Restricting the states satisfying $\phi_i$ (for $i = 1$ and $i = 2$) to the ones in the domain of $f_\Gamma$ is sufficient. So, *winning* is the states such that all fair paths enforced by $f_\Gamma$ satisfy $\phi_1\ \mathbf{W}\ \phi_2$, that is, the states $q$ such that

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \vDash \phi_1\ \mathbf{W}\ \phi_2.$$

We showed that Lines 4 to 14 compute the set of states *winning* for which the strategy is winning, so *sat* accumulates states of $Q'$ for which all states indistinguishable by some agent of $\Gamma$ is in *winning*, that is, the states of $Q'$ satisfying $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$. So, the last statement of the **for** loop adds in *sat* the set of states $q$ such that

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, q' \in winning,$$

and the invariant is still satisfied after the loop body.

Finally, after the execution of the **for** loop, the invariant is satisfied and all uniform partial strategies adequate for $[Q']^E_\Gamma$ have been treated. Thus, *sat* is exactly the set of states of $Q'$ for which there exists a uniform partial strategy that is winning for all indistinguishable states, and that is exactly the set of states of $Q'$ satisfying $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$. $\square$

## A.4 Pre-filtering

This section presents the proofs of correctness of the naive and partial approaches with pre-filtering described in Section 5.4.

### A.4.1 Computing the winning moves

The $\mathit{filter}^M$ algorithms do not ensure that the result is a closed set of $\Gamma$-moves anymore. Indeed, let us take the example of the card game with a cheating player, and let us consider the formula

$$\langle\!\langle \mathit{player} \rangle\!\rangle[\neg \mathit{end}\ \mathbf{U}\ (\mathit{player\ wins} \wedge \mathit{player} = Q)]$$

saying that the player can win the current game with the $Q$. Pre-filtering the $\Gamma$-moves of the structure with the $\mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}$ algorithm gives the moves illustrated in Figure A.1. Only three moves are kept because in all other states, there is no general strategy that wins the objective. Indeed, the only way for the player to win the current game with $Q$ is that the dealer has $A$. In this case, the player can swap his card with $Q$ if he has $K$ and keep it otherwise. Nevertheless, this subset of pre-filtered moves is not closed; indeed, the (only) successor of the bottom state is not in the resulting set of $\Gamma$-moves.

Nevertheless, we can still show that the proposed model-checking algorithms are correct. First, we need to prove that, given a closed set of moves $M_\Gamma$, reducing it to the pre-filtered moves does not change the result of the verification. In other words, if a strategy $f_\Gamma$ is compatible with $M_\Gamma$ and is winning for a state $q$, $f_\Gamma$ reduced to the pre-filtered moves will still be winning for $q$ (according to the corresponding $\mathit{filter}$ algorithm). This is captured by the following theorem.

**Theorem A.15.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a group of agents $\Gamma \subseteq Ag$, three sets of states $Q', Q_1, Q_2 \subseteq Q$, and a closed set $M_\Gamma \subseteq E_\Gamma$ of $\Gamma$-moves,*

$$\begin{aligned}
&\mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{X}}(Q', M_\Gamma) = \mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{X}}(Q', M_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{X}}(Q', E_\Gamma)),\\
&\mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{U}}(Q_1, Q_2, M_\Gamma) =\\
&\mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{U}}(Q_1, Q_2, M_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{U}}(Q_1, Q_2, E_\Gamma)),\\
&\mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{W}}(Q_1, Q_2, M_\Gamma) =\\
&\mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{W}}(Q_1, Q_2, M_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{W}}(Q_1, Q_2, E_\Gamma)).
\end{aligned}$$

In terms of the abbreviations for the $\mathit{filter}$ and $\mathit{filter}^M$ algorithms, this theorem says that,

$$\mathit{filter}_{op}(Q_1, Q_2, M_\Gamma) = \mathit{filter}_{op}(Q_1, Q_2, M_\Gamma \cap \mathit{filter}^M_{op}(Q_1, Q_2, E_\Gamma)).$$

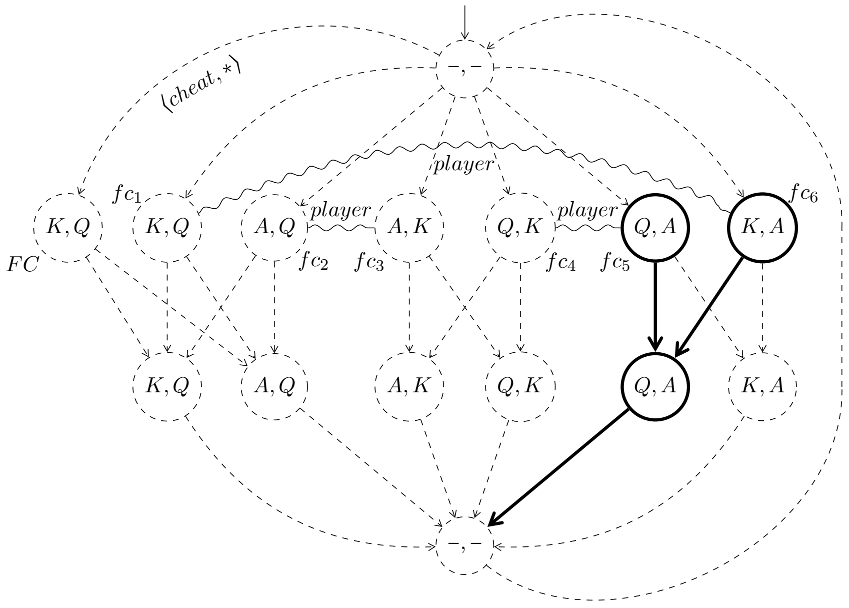


Figure A.1: The graph of the card game with a cheating player. The dashed states and transitions do not belong to a strategy of the player to win the current game with a $Q$, the bold ones are the remaining ones.

To prove that Theorem A.15 is correct, we can show the following properties:

1. Let $M_\Gamma$ be a closed set of $\Gamma$-moves, the set of *important moves* of $M_\Gamma$ for a given objective $\psi$ is defined as

   $\{\langle q, a_\Gamma\rangle \in M_\Gamma \mid \exists f_\Gamma$ compatible with $M_\Gamma$ s.t. $\forall \pi \in out(f_\Gamma, q), \pi \models \psi\}$.

   That is, the important moves are the decisions of $M_\Gamma$ made by strategies in states in which they are winning.

2. The $filter_{\langle\!\langle\rangle\!\rangle}$ algorithms do not need $M_\Gamma$ to compute their result, but only its important moves for the corresponding objective. In other words, let $M_\Gamma^\psi$ be the important moves of $M_\Gamma$ for the objective $\psi$, and let *op* be the operator of the objective $\psi$,

   $$filter_{op}(Q_1, Q_2, M_\Gamma) = filter_{op}(Q_1, Q_2, M_\Gamma^\psi).$$

3. The $filter^M_{\langle\!\langle\rangle\!\rangle}$ algorithms return all the important moves of $M_\Gamma$ for the corresponding objective. That is,

   $$filter^M_{op}(Q_1, Q_2, M_\Gamma) \supseteq M_\Gamma^\psi.$$

It remains to formally prove that these properties are correct.

### A.4.2 The naive approach with pre-filtering

Theorem A.15 showed that we can reduce any strategy $f_\Gamma$ to its pre-filtered moves, we do not need to check them all. Indeed, let $f_\Gamma$ and $f'_\Gamma$ be two strategies for $\Gamma$, if they share the same pre-filtered moves, then the result of the $filter$ algorithms on these pre-filtered moves will be the same. We can then check only one of the two strategies. Furthermore, by splitting the pre-filtered moves into non-conflicting moves, we get subsets of moves that are sufficient to determine the states in which there exists a winning uniform startegy. This is captured by the following lemma.

**Lemma A.16.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a group of agents $\Gamma \subseteq Ag$, two sets of states $Q_1, Q_2 \subseteq Q$, and a strategic operator $op \in \{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{X}, \langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{U}, \langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{W}\}$, for all uniform strategies $f_\Gamma$ for $\Gamma$, there exists a set $M_\Gamma$ of non-$\Gamma$-conflicting $\Gamma$-moves in*

$$Split(\Gamma, filter^M_{op}(Q_1, Q_2, E_\Gamma))$$

*such that*

$$filter_{op}(Q_1, Q_2, f_\Gamma) \subseteq filter_{op}(Q_1, Q_2, M_\Gamma).$$

*Proof.* Let $f_\Gamma$ be a uniform strategy for $\Gamma$. First, the set

$$f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)$$

is a non-$\Gamma$-conflicting subset of $filter^M_{op}(Q_1, Q_2, E_\Gamma)$, as $f_\Gamma$ is a uniform strategy. Thus, the set $f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)$ is a subset of $filter^M_{op}(Q_1, Q_2, E_\Gamma)$ composed of non-$\Gamma$-conflicting moves. So there exists a set $M_\Gamma$ in

$$Split(\Gamma, filter^M_{op}(Q_1, Q_2, E_\Gamma))$$

such that

$$f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma) \subseteq M_\Gamma.$$

Furthermore, the $filter_{op}$ algorithms are monotone since the $Pre_{\langle\!\langle\Gamma\rangle\!\rangle}$ function is monotone. Thus,

$$filter_{op}(Q_1, Q_2, f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)) \subseteq filter_{op}(Q_1, Q_2, M_\Gamma),$$

and by Theorem A.15,

$$\begin{aligned} filter_{op}(Q_1, Q_2, f_\Gamma) &= filter_{op}(Q_1, Q_2, f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)) \\ &\subseteq filter_{op}(Q_1, Q_2, M_\Gamma). \end{aligned}$$

□

Thanks to this lemma, we know that, if there exists a winning strategy for a state $q$, then there exists a set of moves in the split pre-filtered moves for which $q$ is winning. To prove the correctness of the $eval^{PF}_{ATLK_{irF}}$ algorithm, we still need to show that, for any set of split pre-filtered moves, there effectively exists a winning strategy in the states the set of moves is winning for. This is captured by the following lemma.

**Lemma A.17.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a group of agents $\Gamma \subseteq Ag$, two sets of states $Q_1, Q_2 \subseteq Q$, and a strategic operator $op \in \{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{X}, \langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{U}, \langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{W}\}$, for all sets of moves $M_\Gamma$ in $Split(\Gamma, filter^M_{op}(Q_1, Q_2, E_\Gamma))$, there exists a uniform strategy $f_\Gamma$ such that*

$$filter_{op}(Q_1, Q_2, M_\Gamma) = filter_{op}(Q_1, Q_2, f_\Gamma).$$

*Proof.* Let $M_\Gamma$ be a set in $Split(\Gamma, filter^M_{op}(Q_1, Q_2, E_\Gamma))$. By definition of $Split$, $M_\Gamma$ is a subset of $filter^M_{op}(Q_1, Q_2, E_\Gamma)$, thus

$$M_\Gamma = M_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma).$$

So, for all $M'_\Gamma \subseteq E_\Gamma$ such that

$$M'_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma) = M_\Gamma,$$

we have, by Theorem A.15,

$$\begin{aligned} filter_{op}(Q_1, Q_2, M'_\Gamma) &= filter_{op}(Q_1, Q_2, M'_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)), \\ &= filter_{op}(Q_1, Q_2, M_\Gamma). \end{aligned}$$

Furthermore, there exists a uniform strategy $f_\Gamma$ for $\Gamma$ such that

$$f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma) = M_\Gamma.$$

Indeed, such a strategy makes the same choices as $M_\Gamma$ in states of $M_\Gamma|_Q$—this is possible since $M_\Gamma$ is a non-$\Gamma$-conflicting set—, and choices conflicting with $filter^M_{op}(Q_1, Q_2, E_\Gamma)$ elsewhere. It is always possible to make choices conflicting with $filter^M_{op}(Q_1, Q_2, E_\Gamma)$ in some state in $Q \backslash M_\Gamma|_Q$ because, otherwise, such a choice would be in $M_\Gamma$ since $M_\Gamma$ is a largest subset of non-conflicting moves. Thus, there exists a uniform strategy $f_\Gamma$ such that

$$\begin{aligned} filter_{op}(Q_1, Q_2, f_\Gamma) &= filter_{op}(Q_1, Q_2, f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)), \\ &= filter_{op}(Q_1, Q_2, f_\Gamma), \end{aligned}$$

and the proof is done. □

Finally, we can show that the $eval^{PF}_{ATLK_{irF}}$ algorithm is correct. This is captured by the following theorem.

**Theorem A.18.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, *and an* $ATLK_{irF}$ *strategic formula* $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$, $eval^{PF}_{ATLK_{irF}}(S, \langle\!\langle\Gamma\rangle\!\rangle\ \psi)$ *returns the set of states of* $S$ *satisfying* $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$.

*Proof.* We can prove this theorem by induction over the structure of the formula. Let *op* be the strategic operator of $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$. First, $eval^{PF}_{ATLK_{irF}}(S, \langle\!\langle\Gamma\rangle\!\rangle\ \psi)$ computes the *filtered* set as

$$filtered = filter^{M}_{op}(Q_1, Q_2, E_\Gamma),$$

where $Q_1$ (resp. $Q_2$) is the set of states of $S$ satisfying $\phi_1$ (resp. $\phi_2$), by the induction hypothesis. Then, if $filtered = \varnothing$, there exists no state $q$ satisfying $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$. Indeed, for all uniform strategies $f_\Gamma$ for $\Gamma$,

$$\begin{aligned} filter_{op}(Q_1, Q_2, f_\Gamma) &= filter_{op}(Q_1, Q_2, f_\Gamma \cap filtered),\\ &= filter_{op}(Q_1, Q_2, \varnothing),\\ &= \varnothing. \end{aligned}$$

Thus, there exists no strategy $f_\Gamma$ such that all fair paths enforced by $f_\Gamma$ from $q$ satisfy $\psi$, and there is no uniform strategy that is winning for all states indistinguishable from $q$.

Otherwise, the algorithm iterates over the elements $M_\Gamma$ of

$$Split(\Gamma, filtered) = Split(\Gamma, filter^{M}_{op}(Q_1, Q_2, E_\Gamma)).$$

Let $\mathcal{M}_\Gamma$ be the set of elements of $Split(\Gamma, filtered)$ of which $M_\Gamma$ has already taken the value. Let us show that the invariant of the **for** loop is that *sat* is the set of states $q \in Q$ such that there exists $M_\Gamma \in \mathcal{M}_\Gamma$, and a uniform strategy $f_\Gamma$ such that

$$filter_{op}(Q_1, Q_2, M_\Gamma) = filter_{op}(Q_1, Q_2, f_\Gamma),$$

and

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \models \psi.$$

This invariant is trivially verified before entering the loop for the first time as *sat* and $\mathcal{M}_\Gamma$ are both empty. Then, let us suppose that the invariant is true before any iteration of the loop. By Lemma A.17, *winning* is the set of states $q$ for which there exists a uniform strategy $f_\Gamma$ such that

$$filter_{op}(Q_1, Q_2, M_\Gamma) = filter_{op}(Q_1, Q_2, f_\Gamma),$$

and

$$\forall \pi \in out(f_\Gamma, q), \pi \models \psi.$$

*sat* is thus augmented with the set of states $q$ such that there exists a uniform strategy $f_\Gamma$ such that

$$filter_{op}(Q_1, Q_2, M_\Gamma) = filter_{op}(Q_1, Q_2, f_\Gamma),$$

and

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \models \psi.$$

The invariant is thus verified after the iteration.

Finally, let us suppose that $\mathcal{M}_\Gamma = Split(\Gamma, filter^M_{op}(Q_1, Q_2, E_\Gamma))$ and that the loop is done. The invariant says that for all states $q \in sat$, $q \models \langle\!\langle\Gamma\rangle\!\rangle\ \psi$. Furthermore, Lemma A.16 ensures that we forgot no winning state, and the proof is done. □

### A.4.3 The partial approach with pre-filtering

Proving the correctness of the $eval^{Partial,PF}_{ATLK_{irF}}$ algorithm is more complex than for the naive approach with pre-filtering. We have to show that $ReachSplit^{PF}(\Gamma, M_\Gamma, filtered)$ returns the set of largest non-$\Gamma$-conflicting extensions of $M_\Gamma$ with moves of *filtered* reachable from $M_\Gamma$. In other words, given a set of agents $\Gamma$, a set of non-$\Gamma$-conflicting $\Gamma$-moves $M_\Gamma$, and a set of $\Gamma$-moves *filtered*, $ReachSplit^{PF}(\Gamma, M_\Gamma, filtered)$ returns the set of largest subsets $M'_\Gamma$ of $\Gamma$-moves such that

- $M_\Gamma \subseteq M'_\Gamma$,
- $M'_\Gamma \subseteq M_\Gamma \cup filtered$,
- $M'_\Gamma$ is non-$\Gamma$-conflicting,
- for all $\langle q', a'_\Gamma\rangle \in M'_\Gamma$, $\langle q', a'_\Gamma\rangle$ is reachable from some $m \in M_\Gamma$, that is,

$$\forall \langle q', a'_\Gamma\rangle \in M'_\Gamma, \exists \langle q, a_\Gamma\rangle \in M_\Gamma, \exists q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n \text{ s.t.}$$
$$q_0 = q \wedge q_n = q' \wedge \forall 1 \le i \le n, \exists \langle q_{i_1}, a_{i_\Gamma}\rangle \in M'_\Gamma \text{ s.t. } a_{i_\Gamma} \sqsubseteq a_i.$$

This is captured by the following theorem.

**Theorem A.19.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$*, a group of agents* $\Gamma \subseteq Ag$*, a set of non-*$\Gamma$*-conflicting* $\Gamma$*-moves* $M_\Gamma \subseteq E_\Gamma$*, and another set of* $\Gamma$*-moves* $filtered \subseteq E_\Gamma$*, ReachSplit*$^{PF}(\Gamma, M_\Gamma, filtered)$ *returns the set of largest subsets* $M'_\Gamma$ *of* $\Gamma$*-moves such that*

$$M_\Gamma \subseteq M'_\Gamma, \tag{A.1}$$

$$M'_\Gamma \subseteq M_\Gamma \cup filtered, \tag{A.2}$$

$$M'_\Gamma \text{ is non-}\Gamma\text{-conflicting}, \tag{A.3}$$

$$\begin{cases} \forall \langle q', a'_\Gamma\rangle \in M'_\Gamma, \exists \langle q, a_\Gamma\rangle \in M_\Gamma, \exists q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n \text{ s.t.} \\ q_0 = q \wedge q_n = q' \wedge \forall 1 \leq i \leq n, \exists \langle q_{i_1}, a_{i_\Gamma}\rangle \in M'_\Gamma \text{ s.t. } a_{i_\Gamma} \sqsubseteq a_i. \end{cases} \tag{A.4}$$

*Proof.* By definition of the *Post* function, *new_states* is the set of states reachable in one step from a move of $M_\Gamma$ for which $M_\Gamma$ does not define a move. *new_moves* is the set of moves of $filtered$ defined for states of *new_states*. *compatible* is the set of moves of *new_moves* that are not $\Gamma$-conflicting with some move of $M_\Gamma$. *compatible* is thus the set of moves of $filtered$ for states reachable in one step from moves of $M_\Gamma$, that are not in $M_\Gamma$ but that are compatible with it.

If this set of compatible moves reachable in one step is empty, then $M_\Gamma$ is its own single largest extension with reachable moves of $filtered$. Indeed, $M_\Gamma \subseteq M_\Gamma$ and $M_\Gamma \subseteq M_\Gamma \cup filtered$. Furthermore, by pre-condition of *ReachSplit*$^{PF}$, $M_\Gamma$ is a non-$\Gamma$-conflicting set of moves. Equation A.4 is also satisfied by $M_\Gamma$ since, for each move $m'$ of $M_\Gamma$, there exists a path of length 0 from a move of $M_\Gamma$ reaching $m'$. Finally, $M_\Gamma$ is a largest subset. Indeed, if there was a larger subset, there would by a move outside $M_\Gamma$ reachable in one step through moves of $filtered$ from a move of $M_\Gamma$, that would be non-$\Gamma$-conflicting with $M_\Gamma$, and *compatible* would be not empty. Furthermore, there are no other largest subset, otherwise *compatible* would be not empty, too.

If *compatible* is not empty, this means that there exist some moves of $filtered$ compatible with $M_\Gamma$ and reachable from $M_\Gamma$. *compatible* contains all such moves that are compatible. Splitting them into largest non-conflicting subsets and extending $M_\Gamma$ with each such subset leads to non-$\Gamma$-conflicting subsets $M'_\Gamma$ such that $M_\Gamma \subseteq M'_\Gamma$, $M'_\Gamma \subseteq M_\Gamma \cup filtered$, and for which all new moves are moves of $filtered$ reachable (in one step) from some moves of $M_\Gamma$. Such $M'_\Gamma$ satisfy the pre-conditions of *ReachSplit*$^{PF}$, thus it can be recursively called.

The resulting subsets are the sets of largest non-$\Gamma$-conflicting extensions of $M'_\Gamma$ with moves of $filtered$ reachable from $M'_\Gamma$, thus are also non-$\Gamma$-conflicting extensions of $M_\Gamma$ with moves of $filtered$ reachable from $M_\Gamma$. Furthermore, these are largest subsets, otherwise *Compatible*$^M$ or *Post* should be incorrect.

Finally, there are no other largest subsets, otherwise such a subset would either contain an incompatible move, or a move out of $\mathit{filtered}$, or a move that is not reachable from some move in $M_\Gamma$. Thus, the proof is done. □

Given a set of agents $\Gamma \subseteq Ag$, a set of states $Q' \subseteq Q$ and a set of $\Gamma$-moves $M_\Gamma \subseteq E_\Gamma$, $\mathit{PartialStrats}^{PF}(\Gamma, Q', M_\Gamma)$ returns the set of largest non-$\Gamma$-conflicting subsets of $M_\Gamma$ reachable from states of $Q'$. These subsets are smaller than the largest subsets of non-$\Gamma$-conflicting subsets of $M_\Gamma$, as used by $\mathit{eval}^{\mathit{Partial},PF}_{ATLK_{irF}}$, because they are restricted to the moves reachable from states of $Q'$. Nevertheless, restricting the search of winning strategies to these ones is sufficient, as captured by the following lemma.

**Lemma A.20.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC \rangle$, a group of agents $\Gamma \subseteq Ag$, three sets of states $Q', Q_1, Q_2 \subseteq Q$, and a strategic operator $op \in \{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{X}, \langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{U}, \langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{W}\}$, for all uniform partial strategies $f_\Gamma$ adequate for $Q'$, there exists a set $M_\Gamma$ in*

$$\mathit{PartialStrats}^{PF}(\Gamma, Q', \mathit{filter}^{M}_{op}(Q_1, Q_2, E_\Gamma))$$

*such that*

$$\mathit{filter}_{op}(Q_1, Q_2, f_\Gamma) \cap Q' \subseteq \mathit{filter}_{op}(Q_1, Q_2, M_\Gamma) \cap Q'.$$

*Proof.* Let $f_\Gamma$ be a uniform partial strategy adequate for $Q'$. First, the set

$$f_\Gamma \cap \mathit{filter}^{M}_{op}(Q_1, Q_2, E_\Gamma)$$

is a non-$\Gamma$-conflicting subset of $\mathit{filter}^{M}_{op}(Q_1, Q_2, E_\Gamma)$ as $f_\Gamma$ is a uniform strategy. Thus, the set $f_\Gamma \cap \mathit{filter}^{M}_{op}(Q_1, Q_2, E_\Gamma)$ is a subset of $\mathit{filter}^{M}_{op}(Q_1, Q_2, E_\Gamma)$ composed of non-$\Gamma$-conflicting moves. So there exists a set $M_\Gamma$ in

$$\mathit{PartialStrats}^{PF}(\Gamma, Q', \mathit{filter}^{M}_{op}(Q_1, Q_2, E_\Gamma))$$

such that $M_\Gamma$ can be augmented with non-conflicting moves of the set $\mathit{filter}^{M}_{op}(Q_1, Q_2, E_\Gamma)$ that are not reachable from $M_\Gamma$ to get a set $M'_\Gamma$ such that

$$f_\Gamma \cap \mathit{filter}^{M}_{op}(Q_1, Q_2, E_\Gamma) \subseteq M'_\Gamma.$$

Finding such a $M'_\Gamma$ is possible as $M_\Gamma$ is a non-$\Gamma$-conflicting set of moves, and because $M'_\Gamma \in \mathit{Split}(\Gamma, \mathit{filter}^{M}_{op}(Q_1, Q_2, E_\Gamma))$. Furthermore, the moves of $M'_\Gamma \backslash M_\Gamma$ are not reachable from moves of $M_\Gamma$, otherwise $PartialStrats^{PF}$ would be incorrect.

The $filter_{op}$ algorithms are monotone since the $Pre_{\langle\langle\Gamma\rangle\rangle}$ function is monotone. Thus,

$$filter_{op}(Q_1, Q_2, f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)) \subseteq filter_{op}(Q_1, Q_2, M'_\Gamma),$$

and by Theorem A.15,

$$\begin{aligned} filter_{op}(Q_1, Q_2, f_\Gamma) &= filter_{op}(Q_1, Q_2, f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)) \\ &\subseteq filter_{op}(Q_1, Q_2, M'_\Gamma). \end{aligned}$$

Finally, the following is true:

$$filter_{op}(Q_1, Q_2, M_\Gamma) \cap Q' = filter_{op}(Q_1, Q_2, M'_\Gamma) \cap Q'.$$

Indeed, as $M'_\Gamma$ is $M_\Gamma$ augmented with moves that are *unreachable* from $M_\Gamma$, and because $Q' \subseteq M_\Gamma|_Q$, the result of the $filter$ algorithm, for the states of $Q'$, cannot be influenced by moves that they cannot reach.

Thus,

$$filter_{op}(Q_1, Q_2, f_\Gamma) \cap Q' \subseteq filter_{op}(Q_1, Q_2, M_\Gamma) \cap Q'.$$

□

Thanks to this lemma we know that, if there exists a winning partial strategy for a state $q \in Q'$, then there exists a set of moves in $PartialStrats^{PF}$ for which $q$ is winning. To prove the correctness of the $eval^{Partial,PF}_{ATLK_{irF}}$ algorithm, we still need to show that, for any set of $PartialStrats^{PF}$, there effectively exists a winning strategy in the states the set of moves is winning for. This is captured by the following lemma.

**Lemma A.21.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a group of agents $\Gamma \subseteq Ag$, three sets of states $Q', Q_1, Q_2 \subseteq Q$, and a strategic operator $op \in \{\langle\langle\Gamma\rangle\rangle \boldsymbol{X}, \langle\langle\Gamma\rangle\rangle \boldsymbol{U}, \langle\langle\Gamma\rangle\rangle \boldsymbol{W}\}$, for all sets of moves $M_\Gamma$ in $PartialStrats^{PF}(\Gamma, Q', filter^M_{op}(Q_1, Q_2, E_\Gamma))$, there exists a uniform partial strategy $f_\Gamma$ adequate for $Q'$ such that*

$$filter_{op}(Q_1, Q_2, M_\Gamma) \cap Q' = filter_{op}(Q_1, Q_2, f_\Gamma) \cap Q'.$$

*Proof.* Let $M_\Gamma$ be a set in $PartialStrats^{PF}(\Gamma, Q', filter^M_{op}(Q_1, Q_2, E_\Gamma))$. By definition of $PartialStrats^{PF}$, $M_\Gamma \subseteq filter^M_{op}(Q_1, Q_2, E_\Gamma)$, thus

$$M_\Gamma = M_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma).$$

So, for all $M'_\Gamma \subseteq E_\Gamma$ such that

$$M'_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma) = M_\Gamma,$$

we have, by Theorem A.15,

$$\begin{aligned} filter_{op}(Q_1, Q_2, M'_\Gamma) &= filter_{op}(Q_1, Q_2, M'_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)), \\ &= filter_{op}(Q_1, Q_2, M_\Gamma). \end{aligned}$$

Furthermore, there exists a uniform partial strategy $f_\Gamma$ adequate for $Q'$ such that $M_\Gamma$ can be augmented with moves of unreachable non-$\Gamma$-conflicting moves of $filter^M_{op}(Q_1, Q_2, E_\Gamma)$ to get a set $M'_\Gamma$ such that

$$f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma) = M'_\Gamma.$$

Indeed, such a strategy makes the same choices as $M'_\Gamma$ in states of $M'_\Gamma|_Q$—this is possible since $M_\Gamma$ is a non-$\Gamma$-conflicting set—, and choices conflicting with $filter^M_{op}(Q_1, Q_2, E_\Gamma)$ elsewhere. It is always possible to make choices conflicting with $filter^M_{op}(Q_1, Q_2, E_\Gamma)$ in some state in $Q \backslash M'_\Gamma|_Q$ because, otherwise, such a choice would be in $M'_\Gamma$ since $M'_\Gamma$ extends $M_\Gamma$ with moves of $filter^M_{op}(Q_1, Q_2, E_\Gamma)$.

Furthermore, because moves added to $M_\Gamma$ to get $M'_\Gamma$ are unreachable from moves of $M_\Gamma$, and because $M_\Gamma$ contains moves for $Q'$, the following is true:

$$filter_{op}(Q_1, Q_2, M_\Gamma) \cap Q' = filter_{op}(Q_1, Q_2, M'_\Gamma) \cap Q'.$$

Thus, there exists a uniform partial strategy $f_\Gamma$ adequate for $Q'$ such that

$$\begin{aligned} filter_{op}(Q_1, Q_2, f_\Gamma) &= filter_{op}(Q_1, Q_2, f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)), \\ &= filter_{op}(Q_1, Q_2, M'_\Gamma). \end{aligned}$$

Thus,

$$\begin{aligned} filter_{op}(Q_1, Q_2, f_\Gamma) \cap Q' &= filter_{op}(Q_1, Q_2, M'_\Gamma) \cap Q' \\ &= filter_{op}(Q_1, Q_2, M_\Gamma) \cap Q'. \end{aligned}$$

□

We also need a last lemma to ensure that we can stop searching for winning strategies if no state of $Q'$ appears in $filtered|_Q$. This is captured by the following lemma.

**Lemma A.22.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, an $ATLK_{irF}$ strategic formula $\phi = \langle\!\langle \Gamma \rangle\!\rangle\ \psi$ with operator op, the two sets $Q_1, Q_2 \subseteq Q$ of states satisfying the sub-formulas $\phi_1, \phi_2$ of $\phi$ (resp.), and a set of states $Q' \subseteq Q$, if $filter^M_{op}(Q_1, Q_2, E_\Gamma)|_Q \cap Q' = \varnothing$, then there exists no state of $Q'$ satisfying $\langle\!\langle \Gamma \rangle\!\rangle\ \psi$.*

*Proof.* Suppose that $filter^M_{op}(Q_1, Q_2, E_\Gamma)|_Q \cap Q' = \varnothing$ but there exists a state $q \in Q'$ that satisfies $\langle\!\langle \Gamma \rangle\!\rangle\ \psi$. Then, there exists a strategy $f_\Gamma$ such that $q \in filter_{op}(Q_1, Q_2, f_\Gamma)$. Furthermore, by Theorem A.15,

$$filter_{op}(Q_1, Q_2, f_\Gamma) = filter_{op}(Q_1, Q_2, f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)),$$

and

$$q \in filter_{op}(Q_1, Q_2, f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)).$$

Let us show that it is impossible, by definition of the $filter$ and $filter^M$ algorithms.

$\psi = \mathbf{X}\ \phi_1$ If $q \in filter_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{X}}(Q_1, f_\Gamma \cap filter^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{X}}(Q_1, E_\Gamma))$, then there exists a move $\langle q, a_\Gamma\rangle \in f_\Gamma \cap filter^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{X}}(Q_1, E_\Gamma)$ that enforces to reach $Q_1 \cup NFair_{\langle\!\langle \Gamma \rangle\!\rangle}(f_\Gamma \cap filter^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{X}}(Q_1, E_\Gamma))$. But there cannot be such a move as $q \notin filter^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{X}}(Q_1, E_\Gamma)|_Q$ by hypothesis. This leads to a contradiction, and prove the lemma for this case.

$\psi = \phi_1\ \mathbf{U}\ \phi_2$ If $q \in filter_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}}(Q_1, Q_2, f_\Gamma \cap filter^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}}(Q_1, Q_2, E_\Gamma))$, then either

$$q \in Q_{1,2,N} \cap Q_2 = Q_2,$$

or

$$q \in Q_{1,2,N} \cap Pre_{\langle\!\langle \Gamma \rangle\!\rangle}\left(Stay_{\langle\!\langle \Gamma \rangle\!\rangle}\left(\begin{array}{l} Q_{1,2,N} \cap (f_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}} \cup \overline{fc}), \\ Q_2 \cap (f_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}} \cup \overline{fc}), f^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}} \end{array}\right), f^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}}\right),$$

for some $fc \in FC$, where

$$\begin{aligned} f_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}} &= filter_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}}(Q_1, Q_2, f_\Gamma \cap filter^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}}(Q_1, Q_2, E_\Gamma)), \\ f^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}} &= f_\Gamma \cap filter^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}}(Q_1, Q_2, E_\Gamma), \\ Q_{1,2,N} &= Q_1 \cup Q_2 \cup Nfair_{\langle\!\langle \Gamma \rangle\!\rangle}(f_\Gamma \cap filter^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}}(Q_1, Q_2, E_\Gamma)). \end{aligned}$$

In the former case, there would be a move $\langle q, a_\Gamma\rangle$ for $q$ in the set $f^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}}$ as $Moves_\Gamma(Q_2) \in filter^M_{\langle\!\langle \Gamma \rangle\!\rangle \mathbf{U}}(Q_1, Q_2, E_\Gamma)$, leading to a contradiction and proving the lemma. In the latter case, there would be a move $\langle q, a_\Gamma\rangle$ in $filter^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, E_\Gamma)$ enforcing to reach the set

$$Stay_{\langle\!\langle\Gamma\rangle\!\rangle}(Q_{1,2,N} \cap (f_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}} \cup \overline{fc}), Q_2 \cap (f_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}} \cup \overline{fc}), f^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}})$$

in one step, thus a move for $q$ in $filter^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, E_\Gamma)$, leading to a contradiction and proving the lemma.

$\psi = \phi_1 \ \mathbf{W} \ \phi_2$ This case is similar to the previous one. If $q$ belongs to $filter_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}(Q_1, Q_2, f_\Gamma \cap filter^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma))$, then either

$$q \in Q_2,$$

or

$$q \in Q_{1,2,N} \cap Pre_{\langle\!\langle\Gamma\rangle\!\rangle}(Stay(Q_{1,2,N}, Q_2, f^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}), f^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}),$$

where

$$\begin{aligned} f^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}} &= f_\Gamma \cap filter^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma), \\ Q_{1,2,N} &= Q_1 \cup Q_2 \cup Nfair_{\langle\!\langle\Gamma\rangle\!\rangle}(f_\Gamma \cap filter^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)). \end{aligned}$$

In the former case, there would be a move $\langle q, a_\Gamma\rangle$ for $q$ in the set $filter^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)$ as $Moves_\Gamma(Q_2) \in filter^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)$, leading to a contradiction and proving the lemma. In the latter case, there would be a move $\langle q, a_\Gamma\rangle \in filter^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)$ enforcing to reach the set

$$Stay(Q_{1,2,N}, Q_2, f^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}})$$

in one step, thus a move for $q$ in $filter^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)$, leading to a contradiction and proving the lemma. □

Finally, thanks to these theorem and lemmas, we can prove the correctness of the $eval^{Partial,PF}_{ATLK_{irF}}$ algorithm.

**Theorem A.23.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, an $ATLK_{irF}$ strategic formula $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$, and a set of states $Q' \subseteq Q$, $eval^{Partial,PF}_{ATLK_{irF}}(S, Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi)$ returns the set of states of $Q'$ satisfying $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$.*

*Proof.* We can prove this theorem by induction over the structure of the formula. Let *op* be the strategic operator of $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$.

First, $eval^{Partial,PF}_{ATLK_{irF}}(S, \langle\!\langle\Gamma\rangle\!\rangle\ \psi)$ computes the *filtered* set as

$$filtered = filter^{M}_{op}(Q_1, Q_2, E_\Gamma),$$

where $Q_1$ and $Q_2$ are the sets of states satisfying $\phi_1$ (resp. $\phi_2$), by the induction hypothesis. Then, if $filtered|_Q \cap Q' = \varnothing$, there exists no state $q \in Q'$ satisfying $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$, by Lemma A.22. Otherwise, the algorithm iterates over the elements $M_\Gamma$ of

$$\begin{aligned}&PartialStrats^{PF}(\Gamma, [Q']^E_\Gamma, filtered) = \\ &PartialStrats^{PF}(\Gamma, [Q']^E_\Gamma, filter^{M}_{op}(Q_1, Q_2, E_\Gamma)).\end{aligned}$$

Let $\mathcal{M}_\Gamma$ be the set of elements of $PartialStrats^{PF}(\Gamma, [Q']^E_\Gamma, filtered)$ of which $M_\Gamma$ has already taken the value. Let us show that the invariant of the **for** loop is that *sat* is the set of states $q' \in Q'$ such that there exists $M_\Gamma \in \mathcal{M}_\Gamma$, and a uniform strategy $f_\Gamma$ such that

$$filter_{op}(Q_1, Q_2, M_\Gamma) \cap [Q']^E_\Gamma = filter_{op}(Q_1, Q_2, f_\Gamma) \cap [Q']^E_\Gamma,$$

and

$$\forall ag \in \Gamma, \forall q \sim_{ag} q', \forall \pi \in out(f_\Gamma, q), \pi \models \psi.$$

This invariant is trivially verified before entering the loop for the first time as *sat* and $\mathcal{M}_\Gamma$ are both empty. Then, suppose that the invariant is true before any iteration of the loop. By Lemma A.21, *winning* is the set of states $q \in [Q']^E_\Gamma$ for which there exists a uniform strategy $f_\Gamma$ such that

$$filter_{op}(Q_1, Q_2, M_\Gamma) \cap [Q']^E_\Gamma = filter_{op}(Q_1, Q_2, f_\Gamma) \cap [Q']^E_\Gamma,$$

and

$$\forall \pi \in out(f_\Gamma, q), \pi \models \psi.$$

*sat* is thus augmented with the set of states $q \in Q'$ such that there exists a uniform strategy $f_\Gamma$ such that

$$filter_{op}(Q_1, Q_2, M_\Gamma) \cap [Q']^E_\Gamma = filter_{op}(Q_1, Q_2, f_\Gamma) \cap [Q']^E_\Gamma,$$

and

$$\forall ag \in \Gamma, \forall q' \sim_{ag} q, \forall \pi \in out(f_\Gamma, q'), \pi \models \psi.$$

The invariant is thus verified after the iteration.

Finally, let us suppose that

$$\mathcal{M}_\Gamma = PartialStrats^{PF}(\Gamma, [Q']^E_\Gamma, filter^{M}_{op}(Q_1, Q_2, E_\Gamma))$$

and that the loop is done. The invariant says that for all states $q \in sat$, $q \models \langle\!\langle\Gamma\rangle\!\rangle\ \psi$. Furthermore, Lemma A.20 ensures that we forgot no winning state, and the proof is done. □

## A.5 Backward generation of strategies

This section proves the correctness of the backward approach described in Section 5.5, through the following lemma and theorems.

**Lemma A.24.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$ such that $FC = \{Q\}$, two subsets of states $Q_1, Q_2 \subseteq Q$, and a state $q \in Q$, there exists a uniform strategy $f_\Gamma$ such that all fair outcomes from all states indistinguishable from $q$ by $\Gamma$ reach a state of $Q_2$ through states of $Q_1$ if and only if there exists a set of non-$\Gamma$-conflicting $\Gamma$-moves $M'_\Gamma$ that* enforces to reach $Q_2$ through $Q_1$ *such that $[q]^E_\Gamma \subseteq M'_\Gamma|_Q$ and $f_\Gamma \supseteq M'_\Gamma$.*

*Proof.* First, let us assume the existence of a uniform strategy $f_\Gamma$ such that all fair outcomes from all states indistinguishable from $q$ by $\Gamma$ reach a state of $Q_2$ trough states of $Q_1$. As $S$ has only one fairness constraint $Q$, all paths of the structure are fair, thus all outcomes from $[q]^E_\Gamma$ reach $Q_2$ through $Q_1$.

From $f_\Gamma$, we can build a non-$\Gamma$-conflicting set of $\Gamma$-moves $M'_\Gamma$ that *enforces to reach $Q_2$ through $Q_1$* such that $[q]^E_\Gamma \subseteq M'_\Gamma|_Q$ and $M'_\Gamma \subseteq f_\Gamma$. Indeed, let $M'_\Gamma$ be the moves defined by the prefixes of the paths enforced by $f_\Gamma$ from states of $[q]^E_\Gamma$ and ending in the first encountered state of $Q_2$, that is, the set

$$M'_\Gamma = \left\{ \langle q', f_\Gamma(q')\rangle \;\middle|\; \begin{array}{l} \exists q_0 \in [q]^E_\Gamma, \exists \pi \in out(f_\Gamma, q_0), \exists n \geq 0 \text{ s.t.} \\ \pi(n) \in Q_2 \wedge \forall j, 0 \leq j \leq n-1, \pi(j) \in Q_1 \backslash Q_2 \wedge \\ \exists i \text{ s.t. } 0 \leq i \leq n \wedge q_i = q' \end{array} \right\}.$$

$M'_\Gamma$ is a non-$\Gamma$-conflicting set of $\Gamma$-moves as they are defined by the uniform strategy $f_\Gamma$. Furthermore, $M'_\Gamma$ *enforces to reach $Q_2$ through $Q_1$* because all outcomes of $M'_\Gamma$ are finite paths reaching a state of $Q_2$ through states of $Q_1 \backslash Q_2$, by definition. The proof is thus done for this direction.

For the other direction, let us assume the existence of a non-$\Gamma$-conflicting set of $\Gamma$-moves $M'_\Gamma$ that *enforces to reach $Q_2$ through $Q_1$* such that $[q]^E_\Gamma \subseteq M'_\Gamma|_Q$. From $M'_\Gamma$, we can build a uniform strategy $f_\Gamma$ such that all fair outcomes from all states of $[q]^E_\Gamma$ reach a state of $Q_2$ through states of $Q_1$.

Indeed, we can add to $M'_\Gamma$ $\Gamma$-moves for states outside $M'_\Gamma|_Q$ with any non-conflicting action; there exist such moves as $M'_\Gamma$ is non-$\Gamma$-conflicting. Adding these moves extends the finite paths from states of $[q]^E_\Gamma$ to infinite paths. Nevertheless, these paths still reach a state of $Q_2$ through states of $Q_1$, by definition of $M'_\Gamma$. As $S$ has only one fairness constraint $Q$, all these paths are fair, and the proof is done. □

**Theorem A.25.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$ with $FC = \{Q\}$, a set of states $Q' \subseteq Q$ such that $Q' = [Q']^E_\Gamma$, a non-$\Gamma$-conflicting set of $\Gamma$-moves $M_\Gamma$, and two subsets of states $Q_1, Q_2 \subseteq Q$ such that $M_\Gamma$* enforces to reach $Q_2$ through $Q_1$, *$eval^{Backward}_{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{U}}(Q', M_\Gamma, Q_1, Q_2)$ returns the set of states $q \in Q'$ such that there exists a strategy $f'_\Gamma \supseteq M_\Gamma$ such that all outcomes of $f'_\Gamma$ from all states indistinguishable from $q$ reach a state of $Q_2$ through states of $Q_1$.*

*Proof.* The algorithm starts by computing the *lose* and *win* sets of states. First *notlose* is the set of states such that there exists a general strategy to reach $M_\Gamma|_Q$ through $Q_1$. Thus, *lose* is the set of states $q$ of $Q'$ such that there does not exist a strategy to reach $M_\Gamma|_Q$ through $Q_1$ for some state indistinguishable from $q$. There exists no uniform strategy $f'_\Gamma \supseteq M_\Gamma$ that is winning for the objective in all indistinguishable states of *lose* as if there were such a strategy, then it would first reach $M_\Gamma$ through $Q_1$, and then act as $M_\Gamma$.

Second, *win* is the set of states for which all indistinguishable states are in $M_\Gamma|_Q$. *win* are thus the states $q$ of $Q'$ such that $[q]^E_\Gamma \subseteq M_\Gamma|_Q$ and $M_\Gamma$ *enforces to reach* $Q_2$ *through* $Q_1$. By Lemma A.24, there exists a strategy $f'_\Gamma \supseteq M_\Gamma$ such that all fair outcomes of $f'_\Gamma$ from all states indistinguishable from $q$ reach a state of $Q_2$ through states of $Q_1$.

If $Q' \backslash (lose \cup win) = \varnothing$, then all states of $Q'$ are covered and the states for which there exists a winning strategy are effectively the states of *win*. Otherwise, there are states of interest that are not in $lose \cup win$. In this case, the algorithm removes from $Q'$ the states of $lose \cup win$ for which we already can conclude. As *lose* and *win* contain entire classes of indistinguishable states by definition, $Q'$ is still such that $Q' = [Q']^E_\Gamma$ after execution of Line 6.

Then the algorithm computes *new_moves*, the set of moves that enforce to reach states of $M_\Gamma$ in one step from states of $Q_1$, and not already in $M_\Gamma$. *compatible* is thus the moves from states of $Q_1$ that enforce to reach $M_\Gamma$ in one step but are not already in $M_\Gamma$.

If *compatible* is empty, then $M_\Gamma$ cannot be extended with moves that will allow the remaining states of $Q'$ to reach $Q_2$ through $Q_1$ with moves of $M_\Gamma$. Thus there exists no $M'_\Gamma \supseteq M_\Gamma$ such that $[q]^E_\Gamma \subseteq M'_\Gamma|_Q$ for any state $q \in Q'$ and $M'_\Gamma$ *enforces to reach* $Q_2$ *through* $Q_1$, and there exists no strategy $f''_\Gamma \supseteq M'_\Gamma \supseteq M_\Gamma$ such that all fair outcomes of $f''_\Gamma$ from all states indistinguishable from states of $Q'$ reach a state of $Q_2$ through states of $Q_1$. The states $q \in Q'$ such that there exists a uniform strategy $M'_\Gamma \supseteq M_\Gamma$ such that all outcomes of $M'_\Gamma$ from all states indistinguishable from $q$ reach a state of $Q_2$ through states of $Q_1$ are thus the states of *win*.

If *compatible* is not empty, the algorithm enumerates the greatest non-conflicting subsets $M'_\Gamma$ of *compatible* and accumulates in *win* the states such that there exists a uniform strategy $f''_\Gamma \supseteq (M'_\Gamma \cup M_\Gamma) \supseteq M_\Gamma$ such that all fair outcomes of $f''_\Gamma$ from all states indistinguishable from some state $q$ in $Q'$ reach a state of $Q_2$ through states of $Q_1$. This loop is terminated when there are no such $M'_\Gamma$ anymore, or when all states of interest are covered by *win*.

In both cases, *win* is returned, and these are effectively the states $q$ of $Q'$ such that there exists a uniform strategy $f''_\Gamma \supseteq M_\Gamma$ such that all fair paths enforced by $f''_\Gamma$ from all states indistinguishable from $q$ reach a state of $Q_2$ through states of $Q_1$. Indeed, no state $q$ is missing from the returned set because, otherwise, either $[q]^E_\Gamma \subseteq M_\Gamma|_Q$ and $q$ would be in *win* at Line 3, or the corresponding strategy would choose an action enforcing to reach $M_\Gamma$ in one step in some state of $Q_1$, and this choice would be in some $M'_\Gamma \in Split(\Gamma, compatible)$. □

**Theorem A.26.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$ such that $FC = \{Q\}$, a set of states $Q' \subseteq Q$, and an $ATLK_{irF}$ strategic formula $\phi = \langle\langle\Gamma\rangle\rangle\ \psi$ with top-level operator $\langle\langle\Gamma\rangle\rangle \boldsymbol{X}$ or $\langle\langle\Gamma\rangle\rangle \boldsymbol{U}$, the result of $eval^{Backward}_{ATLK_{irF}}(S, Q', \phi)$ is the set of states of $Q'$ satisfying $\langle\langle\Gamma\rangle\rangle\ \psi$.*

*Proof.* This theorem can be proved by induction over the structure of the formula $\phi = \langle\langle\Gamma\rangle\rangle\ \psi$.

$\psi = \mathbf{X}\ \phi'$ First, the algorithm computes the set $Q'''$ of successors of $Q''$ satisfying $\phi'$, by induction hypothesis. Then the **for** loop accumulates in *sat* the set of states $q \in Q''$ for which there exists $M_\Gamma$ in $Split(\Gamma, Pre^M_{\langle\langle\Gamma\rangle\rangle}(Moves_\Gamma(Q'''), E_\Gamma))$ such that all states indistinguishable from $q$ are in $M_\Gamma$.

First, any state $q \in sat$ effectively satisfies $\phi$. Indeed, if there exists $M_\Gamma \in Split(\Gamma, Pre^M_{\langle\langle\Gamma\rangle\rangle}(Moves_\Gamma(Q'''), E_\Gamma))$ such that all states indistinguishable from $q$ are in $M_\Gamma$, then there exists an action that will surely lead to states of $Q'''$ in one step, for all states indistinguishable from $q$. Thus, all (fair) outcomes of any uniform strategy that makes the same choice will have their second state in $Q'''$, that is, their second state will satisfy $\phi'$.

Second, for any state $q$ of $Q''$ satisfying $\phi$, there exists an element $M_\Gamma \in Split(\Gamma, Pre^M_{\langle\langle\Gamma\rangle\rangle}(Moves_\Gamma(Q'''), E_\Gamma))$ such that all states indistinguishable from $q$ are in $M_\Gamma$. Indeed, if $q \in Q''$ satisfies $\phi$, there exists a strategy such that all outcomes from all states indistinguishable from $q$ have their second state satisfying $\phi'$, that is, in $Q'''$. Thus, there exists an action in the states that are indistinguishable from $q$ that enforces to reach states of $Q'''$ in one step. The corresponding moves must belong to $Pre^{M}_{\langle\!\langle\Gamma\rangle\!\rangle}(Moves_\Gamma(Q'''), E_\Gamma)$, as they enforce to reach $Q'''$ in one step. Furthermore, they are non-$\Gamma$-conflicting as they give the same action for indistinguishable states. Thus there exists $M_\Gamma \in \mathit{Split}(\Gamma, Pre^{M}_{\langle\!\langle\Gamma\rangle\!\rangle}(Moves_\Gamma(Q'''), E_\Gamma))$ such that all states indistinguishable from $q$ are in $M_\Gamma$, and the proof is done.

$\psi = \phi_1 \mathbf{~U~} \phi_2$ $Q_i$ is the set of states of $S$ satisfying $\phi_i$ (for $i \in \{1, 2\}$), by induction hypothesis. Thus, after Line 14, *sat* contains the set of states $q$ of $Q''$ such that all states indistinguishable from $q$ satisfy $\phi_2$. These states effectively satisfy $\phi$ as any strategy would be winning in these states as all paths enforced from these states would have already reached $Q_2$.

Then the **for** loop accumulates in *sat* the states $q \in Q''$ such that there exists a uniform strategy $f'_\Gamma \supseteq M_\Gamma$ such that all fair outcomes of $f'_\Gamma$ from all states indistinguishable from $q$ reach a state of $Q_2$ through states of $Q_1$, for some $M_\Gamma \in \mathit{Split}(\Gamma, Moves_\Gamma(Q_2))$.

First, these states $q \in \mathit{sat}$ satisfy $\phi$, as there exists a strategy $f'_\Gamma$ such that all fair outcomes of $f'_\Gamma$ from all states indistinguishable from $q$ reach a state of $Q_2$ through states of $Q_1$.

Second, for any state $q$ of $Q''$ satisfying $\phi$, there exists some set of $\Gamma$-moves $M_\Gamma \in \mathit{Split}(\Gamma, Moves_\Gamma(Q_2))$ such that there exists a uniform strategy $f'_\Gamma \supseteq M_\Gamma$ such that all fair outcomes of $f'_\Gamma$ from all states indistinguishable from $q$ reach a state of $Q_2$ through states of $Q_1$.

Indeed, $\mathit{Split}(\Gamma, Moves_\Gamma(Q_2))$ covers all possible choices of actions in all states of $Q_2$. Thus, if $q \in Q''$ satisfies $\phi$, there exists a strategy $f'_\Gamma$ such that all fair outcomes of $f'_\Gamma$ from all states indistinguishable from $q$ reach a state satisfying $\phi_2$ through states satisfying $\phi_1$. Furthermore, as $\mathit{Split}(\Gamma, Moves_\Gamma(Q_2))$ covers all possible choices, there exists $M_\Gamma$ in $\mathit{Split}(\Gamma, Moves_\Gamma(Q_2))$ such that $f'_\Gamma \supseteq M_\Gamma$, and $q$ is in *sat*. □

## A.6 Interleaving strategy generation and verification

This section proves the correctness of the early approaches with and without pre-filtering, described in Section 6.1.

### A.6.1 The early model-checking algorithm

First, the correctness of the *Complete* algorithm is captured by the following theorem.

**Theorem A.27.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a subset of agents $\Gamma \subseteq Ag$, and a set of non-$\Gamma$-conflicting $\Gamma$-moves $M_\Gamma$, $Complete(M_\Gamma)$ returns the set of $\Gamma$-moves reachable from some move of $M_\Gamma$ and compatible with $M_\Gamma$.*

*Proof.* To prove this theorem, we can show that an invariant of the **while** loop is the following: $\exists i \geq 0$ s.t. $M'_\Gamma$ is the set of $\Gamma$-moves reachable from some move of $M_\Gamma$ in at most $i$ steps, and compatible with all the moves of $M_\Gamma$, and *new_moves* is the set of moves compatible with $M_\Gamma$ in the states reachable in one step from some move of $M'_\Gamma$, for which no move is given in $M'_\Gamma$.

This invariant is satisfied before the first iteration of the loop because $M'_\Gamma = M_\Gamma$, thus $M'_\Gamma$ is effectively the set of moves reachable from $M_\Gamma$ in at most 0 step (and compatible with $M_\Gamma$). Furthermore, *new_moves* is the set of moves compatible with $M_\Gamma$ defined for the states reachable in one step from some move of $M'_\Gamma$, and compatible with $M_\Gamma$, by the definition of *Post* and *Compatible*.

Suppose now that the invariant is satisfied before any iteration and that the condition of the loop is satisfied, that is, *new_moves* $\neq \emptyset$. We can show, that, in this case, the invariant is still satisfied after the execution of the body of the loop.

$M'_\Gamma$ already contains all $\Gamma$-moves reachable from some move of $M_\Gamma$ in at most $i$ steps, and compatible with $M_\Gamma$. *new_moves* contains all the moves compatible with $M_\Gamma$ that are reachable from some move of $M_\Gamma$ in $i+1$ steps, but not in $i$ steps. Thus, $M'_\Gamma \cup$ *new_moves* contains all moves reachable from some move of $M_\Gamma$ in at most $i+1$ steps, and compatible with $M_\Gamma$.

Furthermore, after the execution of the body of the loop, *new_moves* is the set of moves compatible with $M_\Gamma$ defined for the states reachable in one step from some move of $M'_\Gamma$, and compatible with $M_\Gamma$, by the definition of *Post* and *Compatible*, and for which no move is given in $M'_\Gamma$, as *new_states* is restricted to states for which no move is given in $M'_\Gamma$. Thus the invariant is effectively satisfied after the execution of the body of the loop.

Finally, at the end of the algorithm, the invariant and the condition of the loop are satisfied. In this case, $M'_\Gamma$ is the set of $\Gamma$-moves reachable (in any number of steps) from some move of $M_\Gamma$ and compatible with $M_\Gamma$. Indeed, because *new_moves* $= \emptyset$, $M'_\Gamma$ is closed. So there cannot be missing moves reachable from $M_\Gamma$ and compatible with it, thus $M'_\Gamma$ is the set of $\Gamma$-moves reachable from some move of $M_\Gamma$ and compatible with $M_\Gamma$, and the proof is done. □

The correctness of the $eval^{alt}_{ATLK_{irF}}$ and $eval^{Early}_{ATLK_{irF}}$ algorithms is captured by the two following theorems.

**Theorem A.28.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a subset of states $Q' \subseteq Q$ such that $[Q']^E_\Gamma = Q'$, an $ATLK_{irF}$ strategic formula $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$, and an incomplete partial strategy $f_\Gamma$ such that $Q' \subseteq f_\Gamma|_Q$, $eval^{alt}_{ATLK_{irF}}(Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi, f_\Gamma)$ computes the set of states $q \in Q'$ such that there exists an extension of $f_\Gamma$ that is winning for $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$ in all states indistinguishable from $q$.*

*Proof.* First, $cf_\Gamma$ is the set of moves reachable from some move of $f_\Gamma$ and compatible with it. In particular, $f_\Gamma \subseteq cf_\Gamma$ and $cf_\Gamma$ is closed.

Then, *notlose* is the subset of states $q \in Q'$ such that there exists a general strategy in $cf_\Gamma$ to win the objective $\psi$ from $q$. Thus, *lose* is the set of states of $Q'$ such that there exists at least one equivalent state in $Q'$—and thus, in $[Q']^E_\Gamma$, by pre-condition of the algorithm—for which there exists no general strategy in $cf_\Gamma$ to win $\psi$. In other words, *lose* is the set of states such that there cannot be a winning strategy in $cf_\Gamma$ for all equivalent states.

Furthermore, *win* is, before execution of Line 18 of Algorithm 6.3, the set of states $q \in Q'$ such that all extensions of $f_\Gamma$ satisfy the objective $\psi$ from $q$. Thus, after execution of Line 18, *win* is the set of states of $Q'$ such that all extensions of $f_\Gamma$ are winning for all indistinguishable states.

If *lose* and *win* cover all states—that is, if $Q' \backslash (lose \cup win) = \varnothing$—, then the computation is done as there exists an extension of $f_\Gamma$ winning for $\psi$ in states of *win*, and there cannot be an extension of $f_\Gamma$ winning for $\psi$ in states of *lose*. Thus *win* is effectively the set of states for which there exists an extension of $f_\Gamma$ winning for $\psi$, and the proof is done.

If *lose* and *win* do not cover all states, then $f_\Gamma$ is not closed. Indeed, if $f_\Gamma$ is closed, then $f_\Gamma = cf_\Gamma$, $cf_\Gamma$ is one adequate partial strategy, and, for states $q$ of $cf_\Gamma|_Q$, either $cf_\Gamma$ is winning for all indistinguishable states—and $q \in win$—, or $cf_\Gamma$ is losing for some indistinguishable state—and $q \in lose$. As $f_\Gamma$ is not closed, it can be extended with compatible moves to build a larger incomplete partial strategy.

*newstrats* is the set of sets of compatible moves that can extend $f_\Gamma$ in one step. Thus, the elements $f_\Gamma \cup f'_\Gamma$, for $f'_\Gamma \in newstrats$, are all the one-step extensions of $f_\Gamma$. Let $F'_\Gamma$ be the set of sets of moves $f'_\Gamma$ has already taken the value of in the iterations of the loop. Then we can show that an invariant of the **for** loop is: *win* is the set of states $q$ such that all extensions of $f_\Gamma$ are winning in all states indistinguishable from $q$, or such that there exists an $f'_\Gamma \in F'_\Gamma$ such that there exists an extension of $f_\Gamma \cup f'_\Gamma$ that is winning for $\psi$ in all states indistinguishable from $q$.

This invariant is easily proved to be maintained by any iteration of the loop: suppose that the invariant is satisfied before the execution of the body. Then after its execution, the invariant is still satisfied as $F'_\Gamma$ gained one new $f'_\Gamma$, and *win* has been extended with states $q$ such that there exists an extension of $f_\Gamma \cup f'_\Gamma$ that is winning for $\psi$ in all states indistinguishable from $q$.

Finally, at the end of the loop, $F'_\Gamma$ contains all $f'_\Gamma \in newstrats$, thus all sets of compatible moves that can extend $f_\Gamma$ in one step have been considered, and *win* is the set of states $q$ such that there exists an extension of $f_\Gamma$ that is winning for $\psi$ in all states indistinguishable from $q$. □

**Theorem A.29.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a subset of states $Q' \subseteq Q$, and an $ATLK_{irF}$ strategic formula $\langle\langle\Gamma\rangle\rangle\ \psi$, $eval^{Early}_{ATLK_{irF}}(S, Q', \langle\langle\Gamma\rangle\rangle\ \psi)$ is the set of states of $Q'$ satisfying $\langle\langle\Gamma\rangle\rangle\ \psi$.*

*Proof.* $Split(\Gamma, Moves([Q']^E_\Gamma))$ is the set of one-step incomplete partial strategies $f_\Gamma$ such that $[Q']^E_\Gamma \in f_\Gamma|_Q$. Thus, all elements of this set satisfy the pre-conditions of the $eval^{alt}_{ATLK_{irF}}$ algorithm.

Then, let $F'_\Gamma$ be the set of partial strategies that $f_\Gamma$ has already taken the value of, in the iterations of the loop. We can show that one invariant of the **for** loop is: *sat* is the set of states $q \in Q'$ such that there exists some strategy $f_\Gamma \in F'_\Gamma$ for which there exists an extension that is winning for $\psi$ in all states indistinguishable from $q$.

This invariant is easily proved to be maintained by the body of the loop. Suppose that the invariant is satisfied before one execution of the body. Then, after its execution, $F'_\Gamma$ has been updated with $f_\Gamma$, and *sat* has been updated with states $q$ such that there exists an extension of $f_\Gamma$ that is winning for $\psi$ in all states indistinguishable from $q$.

Finally, when the loop is done, the invariant is still satisfied, and $F'_\Gamma$ contains all $f_\Gamma \in Split(\Gamma, Moves([Q']^E_\Gamma))$. Thus, *sat* contains all states $q \in Q'$ such that there exists an extension of some one-step partial strategy that is winning for $\psi$ in all states indistinguishable from $q$. Thus, *sat* contains only states of $Q'$ satisfying $\langle\langle\Gamma\rangle\rangle\ \psi$. Furthermore, no state of $Q'$ satisfying $\langle\langle\Gamma\rangle\rangle\ \psi$ is not in *sat*. Otherwise there would be a strategy $f_\Gamma$ that would be winning in all states indistinguishable from $q \in Q'$ and, either $f_\Gamma$ would not be an extension of some $f'_\Gamma$ of $Split(\Gamma, Moves_\Gamma(Q''))$ and *Split* would be incorrect, or $f_\Gamma$ would be an extension of some $f'_\Gamma \in Split(\Gamma, Moves_\Gamma(Q''))$ and $eval^{alt}_{ATLK_{irF}}$ would be incorrect. □

### A.6.2 Pre-filtering

This section proves the correctness of the early approach with pre-filtering described in Section 6.1.3. The correctness of $eval^{alt,PF}_{ATLK_{irF}}$ and $eval^{Early,PF}_{ATLK_{irF}}$ are captured by the two following theorems.

**Theorem A.30.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, a subset of states $Q' \subseteq Q$ such that $[Q']^E_\Gamma = Q'$, an $ATLK_{irF}$ strategic formula $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$, a set of $\Gamma$-moves $filtered$ such that*

$$filtered = filter^M_{op}(Q_1, Q_2, E_\Gamma),$$

*with $Q_i$ be the states satisfying the ith sub-formula of $\psi$, and an incomplete partial strategy $f_\Gamma$ such that $Q' \subseteq f_\Gamma|_Q$ and $f_\Gamma \subseteq filtered$, the result of $eval^{alt,PF}_{ATLK_{irF}}(Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi, f_\Gamma, filtered)$ is the set of states $q \in Q'$ such that there exists an extension of $f_\Gamma$ that is winning for $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$ in all states indistinguishable from $q$.*

*Proof.* This proof is similar to the one of Theorem A.28 for $eval^{alt}_{ATLK_{irF}}$. First, $cf_\Gamma$ is the set of moves reachable from some move of $f_\Gamma$ and compatible with it. In particular, $f_\Gamma \subseteq cf_\Gamma$ and $cf_\Gamma$ is closed.

Then, *lose* is the set of states such that there cannot be a winning strategy in $cf_\Gamma$ for all equivalent states, and *win* is the set of states of $Q'$ such that all extensions of $f_\Gamma$ are winning for all indistinguishable states. If *lose* and *win* cover all states, *win* is the set of states $q \in Q'$ such that there exists an extension of $f_\Gamma$ winning for $\psi$ in all states indistinguishable from $q$, and the proof is done.

If *lose* and *win* do not cover all states of $Q'$, then $eval^{alt,PF}_{ATLK_{irF}}$ computes the set of moves of $filtered$, compatible with $f_\Gamma$ and reachable in one step from some move of $f_\Gamma$. If there are no such move, then $Q' \backslash lose$ is the set of states $q \in Q'$ such that there exists an extension of $f_\Gamma$ winning for $\psi$ in all states indistinguishable from $q$. Indeed, if there are no such move, then for any extension $f'_\Gamma$ of $f_\Gamma$,

$$\begin{aligned}
&filter_{op}(Q_1, Q_2, f_\Gamma) \cap Q' \\
&= filter_{op}(Q_1, Q_2, f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)) \cap Q' \\
&= filter_{op}(Q_1, Q_2, f'_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)) \cap Q' \\
&= filter_{op}(Q_1, Q_2, f'_\Gamma) \cap Q',
\end{aligned}$$

by the fact that $f'_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)$ reaches no more moves from $f_\Gamma$ than $f_\Gamma$—otherwise there would be some moves in *compatible*—and by Theorem A.15. Furthermore,

$$\begin{aligned}
&filter_{op}(Q_1, Q_2, f'_\Gamma) \cap Q' \\
&= filter_{op}(Q_1, Q_2, f'_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)) \cap Q' \\
&= filter_{op}(Q_1, Q_2, cf_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)) \cap Q' \\
&= filter_{op}(Q_1, Q_2, cf_\Gamma) \cap Q',
\end{aligned}$$

by the fact that $cf_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)$ and $f'_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)$ reach the same moves as $f_\Gamma \cap filter^M_{op}(Q_1, Q_2, E_\Gamma)$ from the moves of $f_\Gamma$, and by Theorem A.15. Thus, $Q' \backslash lose$ contains all the states $q \in Q'$ such that all extensions are winning in all states indistinguishable from $q$.

If the set *compatible* is not empty, *newstrats* is the set of sets of compatible moves that can extend $f_\Gamma$ in one step with moves of $filtered$. Thus, the elements $f_\Gamma \cup f'_\Gamma$, for $f'_\Gamma \in newstrats$, are all the one-step extensions of $f_\Gamma$ with moves of $filtered$. Let $F'_\Gamma$ be the set of sets of moves $f'_\Gamma$ has already taken the value of in the iterations of the loop. We can show that an invariant of the **for** loop is: *win* is the set of states $q \in Q'$ such that all extensions of $f_\Gamma$ are winning in all states indistinguishable from $q$, or such that there exists an $f'_\Gamma \in F'_\Gamma$ such that there exists an extension of $f_\Gamma \cup f'_\Gamma$ that is winning for $\psi$ in all states indistinguishable from $q$.

This invariant is easily proved to be maintained by any iteration of the loop. Suppose that the invariant is satisfied before the execution of the body. Then after its execution, the invariant is still satisfied as $F'_\Gamma$ gained one new $f'_\Gamma$, and *win* has been extended with the states $q \in Q'$ such that there exists an extension of $f_\Gamma \cup f'_\Gamma$ that is winning for $\psi$ in all states indistinguishable from $q$.

Finally, at the end of the loop, $F'_\Gamma$ contains all $f'_\Gamma \in newstrats$, thus all sets of compatible moves that can extend $f_\Gamma$ in one step with moves of $filtered$ have been considered. In this case, *win* is the set of states $q \in Q'$ such that there exists an extension of $f_\Gamma$ that is winning for $\psi$ in all states indistinguishable from $q$.

Indeed, for all states $q$ of *win*, there exists an extension of $f_\Gamma$ that is winning for $\psi$ in all states indistinguishable from $q$, as either all extensions of $f_\Gamma$ are winning, or there exists an element $f'_\Gamma$ of *newstrats* such that there is an extension of $f_\Gamma \cup f'_\Gamma$ that is winning for $\psi$ in all states indistinguishable from $q$.

Furthermore, any state $q$ from $Q'$ such that there exists an extension of $f_\Gamma$ that is winning for $\psi$ in all states indistinguishable from $q$ is in *win*. Let us suppose that it is not true, and let $q$ be a state outside *win* such that there exists an extension of $f_\Gamma$ that is winning for $\psi$ in all states indistinguishable from $q$. If all extensions of $f_\Gamma$ are winning for $\psi$ in $q$, then $q$ must be in the *win* set computed at Line 18, otherwise the $filter_{\mathbf{A}}$ algorithms would be incorrect.

Thus, there are some extensions of $f_\Gamma$ that are not winning in $q$, but there is at least one extension of $f_\Gamma$ that is winning in $q$. Let $f^{ext}_\Gamma$ be such an extension. Then $f^{ext}_\Gamma$ must extend some partial strategy corresponding to $f_\Gamma \cup f'_\Gamma$, where $f'_\Gamma$ is an element of *newstrats*, and $q$ should be in *win*, leading to a contradiction. Indeed, let $Q^{ext}$ be the states for which $f^{ext}_\Gamma$ is winning, and $Q^{cf}$ the states for which $cf_\Gamma$ is winning. Because $q$ belongs to $Q^{ext}$ but not to $Q^{cf}$, we have

$$Q^{cf} \subsetneq Q^{ext},$$

that is,

$$filter_{op}(Q_1, Q_2, cf_\Gamma) \subsetneq filter_{op}(Q_1, Q_2, f_\Gamma^{ext}),$$

where $op$ is the top-level operator of $\psi$, and $Q_i$ is the set of states satisfying the $i$th sub-formula of $\psi$. By Theorem A.15, we also have

$$filter_{op}(Q_1, Q_2, cf_\Gamma \cap filtered) \subsetneq filter_{op}(Q_1, Q_2, f_\Gamma^{ext} \cap filtered),$$

where $filtered = filter^M_{op}(Q_1, Q_2, E_\Gamma)$. As the $filter^M$ algorithms are monotone, this means that

$$cf_\Gamma \cap filtered \subsetneq f_\Gamma^{ext} \cap filtered,$$

that is, $f_\Gamma^{ext}$ has more moves of $filtered$ than $cf_\Gamma$. Furthermore, for these additional moves to be significant in the fact that $q$ belongs to $Q^{ext}$ but not to $Q^{cf}$, they must be reachable from $q$, and thus reachable from $f_\Gamma$. Thus $f_\Gamma^{ext}$ extends $f_\Gamma$ with some moves of $filtered$, and these moves must belong to some $f'_\Gamma$ of $newstrats$. Thus, $q$ must belong to $win$, and this ends the proof. □

**Theorem A.31.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$*, a subset of states* $Q' \subseteq Q$*, and an* $ATLK_{irF}$ *strategic formula* $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$*,* $eval^{Early,PF}_{ATLK_{irF}}(S, Q', \langle\!\langle\Gamma\rangle\!\rangle\ \psi)$ *is the set of states of* $Q'$ *satisfying* $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$*.*

*Proof.* The proof is exactly the same as for Theorem A.29 that proves the correctness of the $eval^{Early}_{ATLK_{irF}}$ algorithm. The only differences are: (1) the computation of $filtered$, (2) Lines 15 to 17, that stop the process if $filtered$ contains no move for states of $Q''$, and (3) the limitation of the split moves to the ones of $filtered$.

The computation of $filtered$ trivially respects the pre-conditions of the $eval^{alt,PF}_{ATLK_{irF}}$ algorithm. Furthermore, stopping the process if $filtered$ does not cover any state of $Q''$ is correct: if $filtered|_Q \cap Q'' = \varnothing$, there exists no state $q \in Q''$ satisfying $\langle\!\langle\Gamma\rangle\!\rangle\ \psi$ by Lemma A.22 and the proof is done in this case.

Finally, any partial strategy $f_\Gamma$ adequate for $Q''$ that chooses a move of $Moves_\Gamma(Q'') \backslash filtered$ for some state $q$ cannot be winning for $q$. This can be shown for the three possible top-level operators of $\psi$.

$\psi = \mathbf{X}\ \phi_1$ If $f_\Gamma$ is winning for $q$, then $q \in \mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q_1, f_\Gamma)$, where $Q_1$ is the set of states satisfying $\phi_1$. By Theorem A.15,

$$\mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q_1, f_\Gamma) = \mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q_1, f_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q_1, E_\Gamma)).$$

If $q \in \mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q_1, f_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q_1, E_\Gamma))$, there exists a move $\langle q, a_\Gamma\rangle \in f_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q_1, E_\Gamma)$ that enforces to reach

$$Q_1 \cup \mathit{NFair}_{\langle\!\langle\Gamma\rangle\!\rangle}(f_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q_1, E_\Gamma)).$$

But there cannot be such a move as the move $\langle q, a_\Gamma\rangle \in f_\Gamma$ does not belong to $\mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q_1, E_\Gamma)$ by hypothesis. This leads to a contradiction, and proves that $q \notin \mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{X}}(Q_1, f_\Gamma)$, and $f_\Gamma$ is not winning for $q$.

$\psi = \phi_1\ \mathbf{U}\ \phi_2$ If $f_\Gamma$ is winning for $q$, $q \in \mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, f_\Gamma)$, where $Q_i$ is the set of states satisfying $\phi_i$. By Theorem A.15,

$$\begin{aligned}&\mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, f_\Gamma) =\\ &\mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, f_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, E_\Gamma)).\end{aligned}$$

If $q \in \mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, f_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, E_\Gamma))$, then either

$$q \in Q_{1,2,N} \cap Q_2 = Q_2,$$

or

$$q \in Q_{1,2,N} \cap \mathit{Pre}_{\langle\!\langle\Gamma\rangle\!\rangle}\left(\mathit{Stay}_{\langle\!\langle\Gamma\rangle\!\rangle}\left(\begin{array}{l} Q_{1,2,N} \cap (f_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}} \cup \overline{fc}),\\ Q_2 \cap (f_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}} \cup \overline{fc}), f^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}\end{array}\right), f^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}\right),$$

for some $fc \in FC$, where

$$\begin{aligned}f_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}} &= \mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, f_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, E_\Gamma)),\\ f^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}} &= f_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, E_\Gamma),\\ Q_{1,2,N} &= Q_1 \cup Q_2 \cup \mathit{Nfair}_{\langle\!\langle\Gamma\rangle\!\rangle}(f_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, E_\Gamma)).\end{aligned}$$

In the former case, there would be a move $\langle q, a_\Gamma\rangle$ for $q$ in $f^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}$, leading to a contradiction as, by hypothesis, there is no move for $q$ in $f_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, E_\Gamma)$. In the latter case, there would be a move $\langle q, a_\Gamma\rangle \in f_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, E_\Gamma)$ enforcing to reach the set

$$\mathit{Stay}_{\langle\!\langle\Gamma\rangle\!\rangle}(Q_{1,2,N} \cap (f_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}} \cup \overline{fc}), Q_2 \cap (f_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}} \cup \overline{fc}), f^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}})$$

in one step, thus a move for $q$ in $f_\Gamma \cap \mathit{filter}^M_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, E_\Gamma)$, leading to a contradiction. This proves that $q \notin \mathit{filter}_{\langle\!\langle\Gamma\rangle\!\rangle\mathbf{U}}(Q_1, Q_2, f_\Gamma)$, and $f_\Gamma$ is not winning for $q$.

$\psi = \phi_1 \ \mathbf{W} \ \phi_2$ This case is similar to the previous one. If $f_\Gamma$ is winning for $q$, then $q \in filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, f_\Gamma)$, where $Q_i$ is the set of states satisfying $\phi_i$. By Theorem A.15,

$$\begin{aligned}&filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, f_\Gamma) = \\ &filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, f_\Gamma \cap filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)).\end{aligned}$$

If $q \in filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, f_\Gamma \cap filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma))$, then either

$$q \in Q_2,$$

or

$$q \in Q_{1,2,N} \cap Pre_{\langle\langle\Gamma\rangle\rangle}(Stay(Q_{1,2,N}, Q_2, f^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}), f^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}),$$

where

$$\begin{aligned}f^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}} &= f_\Gamma \cap filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma), \\ Q_{1,2,N} &= Q_1 \cup Q_2 \cup Nfair_{\langle\langle\Gamma\rangle\rangle}(f_\Gamma \cap filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)).\end{aligned}$$

In the former case, there would be a move $\langle q, a_\Gamma\rangle$ for $q$ in the set $f_\Gamma \cap filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)$, leading to a contradiction as the move for $q$ in $f_\Gamma$ is not in $filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)$ by hypothesis. In the latter case, there would be a move $\langle q, a_\Gamma\rangle \in f_\Gamma \cap filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)$ enforcing to reach the set

$$Stay(Q_{1,2,N}, Q_2, f^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}})$$

in one step, thus a move for $q$ in $f_\Gamma \cap filter^M_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, E_\Gamma)$, leading to a contradiction. This proves that $q \notin filter_{\langle\langle\Gamma\rangle\rangle\mathbf{W}}(Q_1, Q_2, f_\Gamma)$, and $f_\Gamma$ is not winning for $q$. □

## A.7 The fully symbolic approach

This section proves the correctness of the symbolic approaches with and without pre-filtering described in Section 6.2. As for the other *filter* algorithms, we sometimes abbreviate the $filter^{ES}$ algorithms with the notation $filter^{ES}_{op}(ESS, Q^{ES}_1, Q^{ES}_2)$ that depends on the operator $op \in \{\langle\langle\Gamma\rangle\rangle\mathbf{X}, \langle\langle\Gamma\rangle\rangle\mathbf{U}, \langle\langle\Gamma\rangle\rangle\mathbf{W}\}$.

To prove the correctness of the $eval^{Symbolic}_{ATLK_{irF}}$ algorithm, we need to prove an intermediate lemma. This lemma tells that, given two sets of derived states $Q^{ES}_1$ and $Q^{ES}_2$ and a strategic operator *op*, the result of $filter_{op}^{ES}(ESS, Q_1^{ES}, Q_2^{ES})$ is the set of derived states $q^{ES}$ such that all fair paths enforced by the strategy for $\Gamma$ stored in $q^{ES}$ satisfy the objective defined by *op*.

**Lemma A.32.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$*, a group of agents* $\Gamma \subseteq Ag$*, a strategic operator* $op \in \{\langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{X}, \langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{U}, \langle\!\langle\Gamma\rangle\!\rangle \boldsymbol{W}\}$*, and two subsets of states of* $EncStrats(S)$*,* $Q_1^{ES}, Q_2^{ES} \subseteq Q^{ES}$*, such that, for* $i \in \{1,2\}$*,*

$$Q_i^{ES} = Q_i \times \prod_{ag \in Ag} F_{ag}^u,$$

*for some* $Q_1, Q_2 \subseteq Q$*, then*

$$\begin{aligned}&filter_{op}^{ES}(ESS, Q_1^{ES}, Q_2^{ES}) = \\ &\left\{q^{ES} \in Q^{ES} \middle| state(q^{ES}) \in filter_{op}(Q_1, Q_2, strategy(\Gamma, q^{ES}))\right\}.\end{aligned}$$

*Proof.* We can prove this lemma by the fact that the fixpoint computations defining both $filter$ and $filter^{ES}$ algorithms are the same, and by the fact that, by definition of $Pre_{str}$, $q^{ES}$ belongs to $Pre_{str}(Q')$ if and only if the states enforced in one step by the strategy stored in $q^{ES}$, and sharing the same strategy, are in $Q'$. Thus, the states in $filter_{op}^{ES}(Q_1^{ES}, Q_2^{ES})$ are the states for which the stored strategy is winning for the objective, and this is exactly the states that are in the set

$$\left\{q^{ES} \in Q^{ES} \middle| state(q^{ES}) \in filter_{op}(Q_1, Q_2, strategy(\Gamma, q^{ES}))\right\}.$$

□

The correctness of the $eval_{ATLK_{irF}}^{Symbolic}$ algorithm is given by the following theorem.

**Theorem A.33.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$*, and an* $ATLK_{irF}$ *formula* $\phi = \langle\!\langle\Gamma\rangle\!\rangle\ \psi$*,* $eval_{ATLK_{irF}}^{Symbolic}(EncStrats(S), \phi)$ *is the set of states* $Q_1^{ES} = Q_1 \times \prod_{ag \in Ag} F_{ag}^u$ *of* $EncStrats(S)$*, where* $Q_1$ *is the set of states of* $S$ *satisfying* $\phi$*.*

*Proof.* We can prove this theorem by induction on the structure of the formula $\phi$. Let us assume that $eval_{ATLK_{irF}}^{Symbolic}(EncStrats(S), \phi_i)$ is the set of states $Q_i^{ES} = Q_i \times \prod_{ag \in Ag} F_{ag}^u$ of $EncStrats(S)$, where $Q_i$ is the set of states of $S$ satisfying $\phi_i$, for all sub-formulas $\phi_i$ of $\phi$.

Thus, the hypotheses of Lemma A.32 are satisfied by $Q_1^{ES}$ and $Q_2^{ES}$ and, before executing Line 13 of Algorithm 6.13, *winning* contains the set of states $filter_{op}^{ES}(ESS, Q_1^{ES}, Q_2^{ES})$, where *op* is the top-level operator of $\phi$.

By Lemma A.32, *winning* contains the states $q^{ES}$ such that all fair paths enforced by the strategy for $\Gamma$ stored in $q^{ES}$ satisfy $\psi$, as $state(q^{ES})$ belongs to $filter_{op}(Q_1, Q_2, strategy(\Gamma, q^{ES}))$, where $Q_1$ and $Q_2$ are the states of $S$ satisfying $\phi_1$ (resp. $\phi_2$), by induction hypothesis.

Thus, the states in $EqQ(EquivQEqStr(\Gamma, winning))$ are the states for which there exists a strategy for $\Gamma$ (by definition of $EqQ$) such that all equivalent states storing this strategy (by definition of $EquivQEqStr$) are in *winning*, that is, the states for which there exists a strategy $f_\Gamma$ such that all fair paths enforced by $f_\Gamma$ satisfy $\psi$. These are the states of $S$ satisfying $\phi$. Furthermore, by definition of $EqQ$, if, for some $q \in Q$, there exists $q^{ES} \in eval_{ATLK_{irF}}^{Symbolic}(EncStrats(S), \phi)$ such that $state(q^{ES}) = q$, then

$$\{q\} \times \prod_{ag \in Ag} F_{ag}^{u} \subseteq eval_{ATLK_{irF}}^{Symbolic}(EncStrats(S), \phi),$$

and the proof is done. □

### A.7.1 Pre-filtering

This section proves the correctness of the symbolic approach with pre-filtering described in Section 6.2.1. To prove the correctness of the $eval_{ATLK_{irF}}^{Symbolic,PF}$ algorithm, we need to prove an intermediate lemma similar to Lemma A.32, but related to $EncStrats^{PF}$ instead of $EncStrats$.

**Lemma A.34.** *Given an iCGSf* $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$, *a group of agents* $\Gamma \subseteq Ag$, *an* $ATLK_{irF}$ *strategic formula* $\langle\langle\Gamma\rangle\rangle\ \psi$ *with top-level operator* $op \in \{\langle\langle\Gamma\rangle\rangle \boldsymbol{X}, \langle\langle\Gamma\rangle\rangle\, \boldsymbol{U}, \langle\langle\Gamma\rangle\rangle\, \boldsymbol{W}\}$, *and two subsets of states of* $ESS = EncStrats^{PF}(S, filtered)$, $Q_1^{ES}, Q_2^{ES} \subseteq Q^{ES}$, *such that, for* $i \in \{1, 2\}$,

$$Q_i^{ES} = Q_i \times Split(\Gamma, filtered),$$

*for some* $Q_1, Q_2 \subseteq Q$, *and filtered being defined as*

$$filtered = filter_{op}^{M}(Q_1, Q_2, E_\Gamma),$$

$filter_{op}^{ES}(ESS, Q_1^{ES}, Q_2^{ES})$ *is the set of states* $q^{ES}$ *of* $Q^{ES}$ *such that there exists a winning strategy for* $\psi$ *in* $state(q^{ES})$.

*Proof.* First, the definition of $Pre_{str}$ implies that, if $strategy(\Gamma, q^{ES})$ is not defined for some original state—as it is the case with $EncStrats^{PF}$ since only the remaining moves are encoded—, then $q^{ES}$ belongs to

$Pre_{str}(Q_1^{ES})$ iff either $strategy(\Gamma, q^{ES})$ is defined for $state(q^{ES})$ and the states enforced in one step by this strategy, and sharing the same strategy, are in $Q_1^{ES}$, or this strategy is not defined for $state(q^{ES})$ and all successors of $q^{ES}$ sharing the same strategy are in $Q_1^{ES}$.

Thus, given a state $q^{ES}$, let $M_\Gamma$ be defined as

$$strategy(\Gamma, q^{ES}) \cup (E_\Gamma \backslash (strategy(\Gamma, q^{ES})|_Q \times Act^\Gamma)),$$

that is, the strategy $strategy(\Gamma, q^{ES})$ augmented with all enabled moves in states in which $strategy(\Gamma, q^{ES})$ is not defined. $M_\Gamma$ is closed by definition, as it is defined for all states. $q^{ES}$ is in $filter_{op}^{ES}(ESS, Q_1^{ES}, Q_2^{ES})$ if there exists a strategy in $M_\Gamma$ that is winning for the objective in $state(q^{ES})$, by definition of $Pre_{str}$ and the fact that the fixpoint computations are the same as the $filter$ algorithms.

Conversely, if there exists a winning strategy $f_\Gamma$ for $\psi$ in some state $q \in Q$, then there exists $q^{ES} \in filter_{op}^{ES}(ESS, Q_1^{ES}, Q_2^{ES})$ such that $state(q^{ES}) = q$. Indeed, $f_\Gamma \cap filtered$ belongs to $Split(\Gamma, filtered)$, so $M_\Gamma$ defined as above is such that

$$\begin{aligned} filter_{op}(Q_1, Q_2, M_\Gamma) &= filter_{op}(Q_1, Q_2, M_\Gamma \cap filtered),\\ &\supseteq filter_{op}(Q_1, Q_2, f_\Gamma \cap filtered),\\ &= filter_{op}(Q_1, Q_2, f_\Gamma),\end{aligned}$$

and $\{q\} \times \{f_\Gamma\}$ belongs to $filter_{op}^{ES}(ESS, Q_1^{ES}, Q_2^{ES})$. □

The correctness of the $eval_{ATLK_{irF}}^{Symbolic,PF}$ algorithm is given by the following theorem.

**Theorem A.35.** *Given an iCGSf $S = \langle Ag, Q, Q_0, Act, e, \delta, \sim, V, FC\rangle$ and an $ATLK_{irF}$ strategic formula $\langle\langle\Gamma\rangle\rangle\ \psi$ such that all sub-formulas are atomic propositions, $eval_{ATLK_{irF}}^{Symbolic,PF}(S, \langle\langle\Gamma\rangle\rangle\ \psi)$ is the set of states $Q_1^{ES}$ such that $States(Q_1^{ES})$ is the set of states of $S$ satisfying $\phi$.*

*Proof.* This proof is similar to the one of Theorem A.33. We can prove this theorem by induction on the structure of the formula $\phi$. Let us assume that the result of $eval_{ATLK_{irF}}^{Symbolic,PF}(S, \phi_i)$ is the set $Q_i^{ES}$ such that $States(Q_i^{ES})$ is the set of states of $S$ satisfying $\phi_i$, for all sub-formulas $\phi_i$ of $\phi$. First, Algorithm 6.14 computes $filtered$. If $filtered = \varnothing$, there exists no state satisfying $\langle\langle\Gamma\rangle\rangle\ \psi$.

Otherwise, $States(Q_1^{ES})$ and $States(Q_2^{ES})$ satisfy the hypotheses of Lemma A.34 and, before executing the last line of Algorithm 6.14, $winning$ contains the set of states $filter_{op}^{ES}(ESS, Q_1^{ES}, Q_2^{ES})$, where $op$ is the top-level operator of $\phi$.

By Lemma A.34, $winning$ contains the states $q^{ES} \in Q^{ES}$ such that $state(q^{ES})$ belongs to $filter_{op}(Q_1, Q_2, strategy(\Gamma, q^{ES}))$, where $Q_1$ and $Q_2$ are the states of $S$ satisfying $\phi_1$ (resp. $\phi_2$), by induction hypothesis. By Lemmas A.16 and A.17, these are the states from which all fair paths enforced by the strategies for $\Gamma$ stored in $q^{ES}$ satisfy $\psi$.

Thus, the states in $EqQ(EquivQEqStr(\Gamma, winning))$ are the states for which there exists a strategy for $\Gamma$ (by definition of $EqQ$) such that all equivalent states storing this strategy (by definition of $EquivQEqStr$) are in $winning$, that is, the states for which there exists a strategy $f_\Gamma$ such that all fair paths enforced by $f_\Gamma$ satisfy $\psi$. These are the states of $S$ satisfying $\phi$. $\square$

# *Appendix B*

# *μ-calculus rich explanations: proofs of theorems*

This appendix proves the properties and theorems linked to the μ-calculus framework described in Chapter 10. The first property says that adequate explanations are necessary and sufficient proofs for the satisfaction of μ-calculus formulas.

**Property B.1.** *Given a Kripke structure $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$, a state $q \in Q$, a $\mu$-calculus formula $\phi$ and an environment $e$, $q \in [\![\phi]\!]^S e$ if and only if there exists an adequate explanation $E$ for $q \in [\![\phi]\!]^S e$.*

*Proof.* To prove that if $q \in [\![\phi]\!]^S e$ then there exists an adequate explanation $E$ for $q \in [\![\phi]\!]^S e$, we can simply rely on the proof of correctness of Algorithm 10.1 presented in Theorem B.2. This algorithm generates an adequate explanation for $q \in [\![\phi]\!]^S e$, assuming that $q \in [\![\phi]\!]^S e$. The requirements for Algorithm 10.1 are met, thus the algorithm can generate an adequate explanation, and the point is proved.

Suppose now that there exists an adequate explanation $E = \langle O, T\rangle$ for $q \in [\![\phi]\!]^S e$. Let us consider the well-founded relation defined as the standard sub-formula relation extended with the fact that, if $\phi = \mu v.\ \psi$, then $\psi^k(false)$ is a sub-formula of $\phi$, for any integer $k \geq 0$. This relation is still well-founded because $\phi$ does not appear as a sub-formula of $\psi^k(false)$. Let us thus show by induction over this well-founded relation that $q \in [\![\phi]\!]^S e$.

$\phi = true$ $\quad q \in [\![true]\!]^S e$ is always true. The proof is thus trivially done.

$\phi = false$ $\quad$ The hypothesis that there exists an adequate explanation $E$ for $q \in [\![\phi]\!]^S e$ cannot be met in this case as $E$ should be consistent, thus

$\langle q, false, e\rangle$ cannot belong to $O$, and $E$ cannot be adequate. This case is thus trivially proved.

$\phi = p$ **or** $\phi = \neg p$ If $\phi = p$, then $\langle q, p, e\rangle \in O$ as $E$ is adequate. Furthermore, as $E$ matches $S$, $p \in V(q)$, thus $q \in [\![p]\!]^S e$, and the proof is done. The other case is similar.

$\phi = v$ **or** $\phi = \neg v$ If $\phi = v$, then $\langle q, v, e\rangle \in O$ as $E$ is adequate. Furthermore, as $E$ is consistent, $q \in e(v)$. Thus $q \in [\![v]\!]^S e$, and the proof is done. The other case is similar.

$\phi = \phi_1 \wedge \phi_2$ $succ(\langle q, \phi, e\rangle) = \{o_1, o_2\}$, where $o_1 = \langle q, \phi_1, e\rangle \in O$ and $o_2 = \langle q, \phi_2, e\rangle \in O$, as $E$ is consistent. As $E$ is consistent and matches $S$ and $o_1, o_2 \in O$, we have $q \in [\![\phi_j]\!]^S e$ for both $j \in \{1, 2\}$ by induction hypothesis. Thus $q \in [\![\phi]\!]^S e$ and the proof is done.

$\phi = \phi_1 \vee \phi_2$ $succ(\langle q, \phi, e\rangle) = \{o_j\}$, where $o_j = \langle q, \phi_j, e\rangle \in O$ for some $j \in \{1, 2\}$, as $E$ is consistent. As $E$ is consistent and matches $S$, and $o_j \in O$, we have $q \in [\![\phi_j]\!]^S e$ by induction hypothesis. Thus $q \in [\![\phi]\!]^S e$ and the proof is done.

$\phi = \Diamond_i\ \phi'$ $succ(\langle q, \phi, e\rangle) = \{o'\}$, where $o' = \langle q', \phi', e\rangle \in O$ for some $q' \in Q$, as $E$ is consistent and matches $S$. Also, $\langle q, q'\rangle \in R_i$ as $E$ matches $S$. By induction hypothesis, $q' \in [\![\phi']\!]^S e$ as $E$ is consistent, matches $S$, and $o' \in O$. Thus there exists a successor $q'$ of $q$ through $R_i$ such that $q'$ belongs to the interpretation of $\phi'$ under $e$ in $S$, thus $q \in [\![\phi]\!]^S e$ and the proof is done.

$\phi = \Box_i\ \phi'$ For all $o' \in succ(\langle q, \phi, e\rangle)$, $o' = \langle q', \phi', e\rangle \in O$ for some $q' \in Q$, as $E$ is consistent and matches $S$. For all $o' = \langle q', \phi', e\rangle \in succ(\langle q, \phi, e\rangle)$, as $E$ is consistent and matches $S$ and $o' \in O$, we have that $q' \in [\![\phi']\!]^S e$ by induction hypothesis. Furthermore, as $E$ matches $S$, all states appearing in these $o'$ are the successors of $q$ through $R_i$. Thus all successors $q'$ of $q$ through $R_i$ belong to the interpretation of $\phi'$ under $e$ in $S$, thus $q \in [\![\phi]\!]^S e$, and the proof is done.

$\phi = \mu v.\ \psi$ $succ(\langle q, \phi, e\rangle) = \{o'\}$ where $o' = \langle q, \psi^k(false), e\rangle \in O$ for some $k \geq 0$, as $E$ is consistent. By induction hypothesis, as $E$ is consistent and matches $S$, and $o' \in O$, $q \in [\![\psi^k(false)]\!]^S e$. Furthermore, by definition of the semantics of the µ-calculus, $q \in [\![\phi]\!]^S e$ if and only if there exists $k \geq 0$ such that $q \in [\![\psi^k(false)]\!]^S e$. Thus $q \in [\![\phi]\!]^S e$ and the proof is done.

$\phi = \nu v.\ \psi$ $\quad succ(o) = \{o'\}$ where $o' = \langle q, \psi(\phi), e\rangle \in O$, as $E$ is consistent. Nevertheless, we cannot rely on the induction hypothesis to conclude that $q \in [\![\psi(\phi)]\!]^S e$ as $\psi(\phi)$ is not a strict sub-formula of $\phi$ by the relation defined at the beginning of this proof.

Let us first consider that $o$ is not a descendant of $o'$ through $T$, that is, there is no path from $o'$ to $o$ through $T$. In this case, let us assume that for all descendants $o'' = \langle q'', \phi, e\rangle$ of $o'$, $q'' \in [\![\phi]\!]^S e$. With the same arguments as for the cases above, we can show that $q \in [\![\psi(\phi)]\!]^S e$. Thus $q \in [\![\phi]\!]^S e$, and the proof is done.

Let us now consider that $o$ is a descendant of $o'$ through $T$. In this case, using the same arguments as for the cases above, we can show that $q \in [\![\psi^k(true)]\!]^S e$, for any $k \geq 0$. Indeed, $q \in [\![true]\!]^S e$ by definition, and from $o'$, we can reach $o$, then $o'$ as it is the only successor of $o$, $k$ times. As $q \in [\![\psi^k(true)]\!]^S e$ for any $k \geq 0$, $q \in [\![\phi]\!]^S e$, and the proof is done. □

The following theorem proves the correctness of the algorithm generating µ-calculus explanations.

**Theorem B.2.** *Given a Kripke structure $S = \langle Q, \{R_i \mid i \in \Sigma\}, V\rangle$, a state $q \in Q$, a $\mu$-calculus formula $\phi$ and an environment $e$ such that $q \in [\![\phi]\!]^S e$, $explain(S, q, \phi, e)$ is an adequate explanation $E$ for $q \in [\![\phi]\!]^S e$.*

*Proof.* We can prove this theorem by showing that an invariant of the main **while** loop of the algorithm is

1. $\langle q, \phi, e\rangle \in O \cup pending$, and
2. for all $o \in O$, $o$ is locally consistent in $\langle O \cup pending, T\rangle$, and
3. for all $o \in pending$, $succ(o) = \varnothing$, and
4. for all $\langle q', \phi', e'\rangle \in O \cup pending$, $q' \in Q$, and
5. for all $\langle q', \phi', e'\rangle \in O \cup pending$, $q' \in [\![\phi']\!]^S e'$, and
6. for all $\langle\langle q', \phi', e'\rangle, \langle q'', \phi'', e''\rangle\rangle \in T$, either $q' = q''$, or $\phi'$ belongs to $\{\Diamond_i \phi'', \Box_i \phi''\}$ and $\langle q', q''\rangle \in R_i$, and
7. for all $o' = \langle q', \Box_i \phi', e'\rangle \in O$,
$$\begin{aligned} &\exists o'' \in succ(o') \text{ s.t. } o'' = \langle q'', \phi'', e''\rangle \text{ for some } \phi'', e'' \\ &\iff \langle q', q''\rangle \in R_i, \end{aligned}$$
and
8. $O \cap pending = \varnothing$.

First, the invariant is satisfied before entering the loop for the first time: $O = \varnothing$, $T = \varnothing$, and $pending = \{\langle q, \phi, e\rangle\}$. Thus the different points of the invariant are satisfied:

1. $\langle q, \phi, e\rangle \in O \cup pending$ as $o \in pending$,

2. all obligations of $O$ are locally consistent in $\langle O \cup pending, T\rangle$ as there are no obligations in $O$,

3. for all $o \in pending$, $succ(o) = \varnothing$ as $T = \varnothing$,

4. for all $\langle q', \phi', e'\rangle \in O \cup pending$, $q' \in Q$ as $O \cup pending = \{\langle q, \phi, e\rangle\}$ and $q \in Q$ by pre-condition of the algorithm,

5. for all $\langle q', \phi', e'\rangle \in O \cup pending$, we have that $q' \in [\![\phi']\!]^S e'$ as $O \cup pending = \{\langle q, \phi, e\rangle\}$ and $q \in [\![\phi]\!]^S e$ by pre-condition of the algorithm,

6. $T = \varnothing$, so this point is trivially satisfied, and

7.–8. $O = \varnothing$, so these points are trivially satisfied.

Second, suppose that the invariant is satisfied before executing the body of the loop, and suppose also that $pending \neq \varnothing$. We can show that the invariant is still satisfied after executing the body. First, it picks one element $o' = \langle q', \phi', e'\rangle$ from *pending*, removes it from *pending* and adds it to $O$. Then, depending on the top-level operator of $\phi'$, new obligations are computed in $O'$, the newly discovered ones are added to *pending*, and new transitions are added to $T$ between $o'$ and the elements of $O'$. Thus, the first point of the invariant is still satisfied by $O \cup pending$ as no obligation is removed from this set.

Furthermore, the last point of the invariant is still satisfied after executing the body of the loop because, by the invariant, $o'$ is not in $O$, and it is transferred from *pending* to $O$. Also, the body adds to *pending* the elements of $O'$ that are not in $O$. Thus this point stays satisfied.

Let us now show that Points 2 to 7 of the invariant are still satisfied for all possible top-level operators of $\phi'$.

$\phi' \in \{true, p, \neg p, v, \neg v\}$ The obligation $o' = \langle q', \phi', e'\rangle$ is simply removed from *pending* and added to $O$. The invariant is still satisfied after execution of the body:

2. All obligations of $O$ before executing the body stay locally consistent as they are not modified, nor their successors. Furthermore, $o'$ is locally consistent in $\langle O \cup pending, T\rangle$ as $o'$ has no successor by

Point 3 of the invariant. Also, if $\phi' = v$ (or $\phi' = \neg v$), $q' \in e'(v)$ ($q' \notin e'(v)$, resp.) by Point 5 of the invariant.

3. No obligation has been added to *pending* and no edge to $T$, so this point is still satisfied.

4.–5. No obligation has been added to $O \cup pending$, so Points 4 and 5 are still satisfied.

6. No edge has been added to $T$, so this point is still satisfied.

7. No new obligation with a $\Box_i$ operator has been added to $O$—as $\phi' \in \{true, p, \neg p, v, \neg v\}$—so this point is still satisfied.

$\phi' = \phi_1 \wedge \phi_2$ The obligation $o' = \langle q', \phi', e' \rangle$ is removed from *pending* and added to $O$, and two new obligations $o_1 = \langle q', \phi_1, e' \rangle$ and $o_2 = \langle q', \phi_2, e' \rangle$ are created. $\langle o', o_1 \rangle$ and $\langle o', o_2 \rangle$ are added to $T$, and the obligations that were not already in $O$ are added to *pending*. The invariant is still satisfied after execution of the body:

2. All obligations of $O$ before executing the body stay locally consistent as they are not modified, nor their successors. $o'$ is locally consistent in $\langle O \cup pending, T \rangle$ as, after execution, $o_1$ and $o_2$ are in $O \cup pending$, and are the only successors of $o'$ as $o'$ had no successor through $T$ before executing the body (by Point 3 of the invariant and the fact that $o'$ was in *pending*). Also, the three obligations share the same environment $e'$.

3. Either $o_1$ was already in $O \cup pending$, and the point remains satisfied, or it is a new obligation, thus has no successor through $T$, and the point remains satisfied, too. The proof for $o_2$ is the same.

4. As the states of $o_1$ and $o_2$ are the same as $o'$ by construction, this point is still satisfied.

5. As $q' \in [\![\phi_1 \wedge \phi_2]\!]^S e'$ by the Point 5, $q' \in [\![\phi_1]\!]^S e'$. Thus this point is satisfied by $o_1$, too, and all other obligations are not modified. The proof for $o_2$ is the same.

6. Two new edges are added to $T$. They satisfy Point 6 of the invariant as $\phi' \notin \{\Diamond_i \phi'', \Box_i \phi''\}$ and the states of $o_1$ and $o_2$ are the same as the one of $o'$.

7. This point is still satisfied as no obligation with $\Box_i \phi''$ has been added to $O$.

$\phi' = \phi_1 \vee \phi_2$ The obligation $o' = \langle q', \phi', e' \rangle$ is removed from *pending* and added to $O$, and a new obligation $o_1 = \langle q', \phi_1, e' \rangle$ or $o_2 = \langle q', \phi_2, e' \rangle$ is created, depending on whether $q' \in [\![\phi_1]\!]^S e'$ or $q' \in [\![\phi_2]\!]^S e'$. Let us suppose that $q' \in [\![\phi_1]\!]^S e'$. Thus $o_1$ is created. $\langle o', o_1 \rangle$ is added to $T$, and if $o_1$ was not already in $O$, it is added to *pending*. The invariant is still satisfied after execution of the body:

2. All obligations of $O$ before executing the body stay locally consistent as they are not modified, nor their successors. $o'$ is locally consistent in $\langle O \cup pending, T \rangle$ as, after execution, $o_1$ is in $O \cup pending$, and is the only successor of $o'$ as $o'$ had no successor through $T$ before executing the body (by Point 3 of the invariant and the fact that $o'$ was in *pending*). Also, the two obligations share the same environment $e'$.

3. Either $o_1$ was already in $O \cup pending$, and the point remains satisfied, or it is a new obligation, thus has no successor through $T$, and the point remains satisfied, too.

4. As the state of $o_1$ is the same as $o'$ by construction, this point is still satisfied.

5. $q' \in [\![\phi_1]\!]^S e'$ by the **if** condition. Thus this point is satisfied by $o_1$, too, and all other obligations are not modified. For $o_2$, as $q' \in [\![\phi']\!]^S e'$ and $q' \notin [\![\phi_1]\!]^S e'$ as the **if** condition is not satisfied, we have that $q' \in [\![\phi_2]\!]^S e'$.

6. One new edge is added to $T$. It satisfies Point 6 of the invariant as $\phi' \notin \{\Diamond_i \phi'', \Box_i \phi''\}$ and the state of $o_1$ is the same as the one of $o'$.

7. This point is still satisfied as no obligation with $\Box_i \phi''$ has been added to $O$.

The case for $q' \in [\![\phi_2]\!]^S e$ is the same.

$\phi' = \Diamond_i \phi''$ The obligation $o' = \langle q', \phi', e' \rangle$ is removed from *pending* and added to $O$, and a new obligation $o'' = \langle q'', \phi'', e' \rangle$ is created, where $q'' \in [\![\phi'']\!]^S e'$ and $q''$ is a successor of $q'$ through $R_i$. Then $\langle o', o'' \rangle$ is added to $T$, and if $o''$ was not already in $O$, it is added to *pending*. The invariant is still satisfied after execution of the body:

2. All obligations of $O$ before executing the body stay locally consistent as they are not modified, nor their successors. $o'$ is locally consistent in $\langle O \cup pending, T \rangle$ as, after execution $o''$ is in $O \cup pending$, and is the only successor of $o'$ as $o'$ had no successor through $T$ before

executing the body (by Point 3 of the invariant and the fact that $o'$ was in *pending*). Also, the two obligations share the same environment $e'$.

3. Either $o''$ was already in $O \cup \mathit{pending}$, and the point remains satisfied, or it is a new obligation, thus has no successor through $T$, and the point remains satisfied, too.

4. The state of $o''$ is in $Q$ by the **pick** statement.

5. $q'' \in [\![\phi'']\!]^S e'$ by the **pick** statement. This statement is sure to succeed as $q' \in [\![\phi']\!]^S e'$. Thus this point is satisfied by $o''$, and all other obligations are not modified.

6. One new edge is added to $T$. It satisfies Point 6 of the invariant as $\phi' = \Diamond_i\ \phi''$ and the state of $o''$ is a successor of the one of $o'$ through $R_i$ by the **pick** statement.

7. This point is still satisfied as no obligation with $\Box_i\ \phi''$ has been added to $O$.

$\phi' = \Box_i\ \phi''$ The obligation $o' = \langle q', \phi', e'\rangle$ is removed from *pending* and added to $O$, and a new obligation $o'' = \langle q'', \phi'', e'\rangle$ is created for each successor $q''$ of $q'$ through $R_i$. Then $\langle o', o''\rangle$ are added to $T$, and all $o''$ that were not already in $O$ are added to *pending*. The invariant is still satisfied after execution of the body:

2. All obligations of $O$ before executing the body stay locally consistent as they are not modified, nor their successors. $o'$ is locally consistent in $\langle O \cup \mathit{pending}, T\rangle$ as $o'$ had no successor through $T$ before executing the body (by Point 3 of the invariant and the fact that $o'$ was in *pending*), and all $o''$ follow the conditions for the local consistency of $o'$. Also, all obligations share the same environment $e'$ by construction.

3. For each $o''$, either it was already in $O \cup \mathit{pending}$, and the point remains satisfied, or it is a new obligation, thus has no successor through $T$, and the point remains satisfied, too.

4. The states of all $o''$ are in $Q$ by construction.

5. Because $q' \in [\![\Box_i\ \phi'']\!]^S e'$, the states $q''$ of all $o''$ are such that $q'' \in [\![\phi'']\!]^S e'$. Thus this point is satisfied by all $o''$, and all other obligations are not modified.

6. One new edge is added to $T$ for each $o''$. They satisfy Point 6 of the invariant as $\phi' = \Box_i\ \phi''$ and the state of each $o''$ is a successor of $q'$ through $R_i$ by construction.

7. This point is still satisfied as the states of the $o''$ obligations are all the successors of $q'$ through $R_i$ by construction. Furthermore, all other obligations are not modified.

$\phi' = \mu v.\ \psi$ The obligation $o' = \langle q', \phi', e'\rangle$ is removed from *pending* and added to $O$, and a new obligation $o'' = \langle q', \psi^k(false), e'\rangle$ for some $k \geq 0$ is created, where $q' \in [\![\psi^k(false)]\!]^S e'$. Note that such an obligation always exists as $q' \in [\![\mu v.\ \psi]\!]^S e$. Then $\langle o', o''\rangle$ is added to $T$, and if $o''$ was not already in $O$, it is added to *pending*. The invariant is still satisfied after execution of the body:

2. All obligations of $O$ before executing the body stay locally consistent as they are not modified, nor their successors. $o'$ is locally consistent in $\langle O \cup pending, T\rangle$ as, after execution, $o''$ is in $O \cup pending$, and is the only successor of $o'$ as $o'$ had no successor through $T$ before executing the body (by Point 3 of the invariant and the fact that $o'$ was in *pending*). Also, the two obligations share the same environment $e'$.

3. Either $o''$ was already in $O \cup pending$, and the point remains satisfied, or it is a new obligation, thus has no successor through $T$, and the point remains satisfied, too.

4. As the state of $o''$ is the same as $o'$ by construction, this point is still satisfied.

5. $q' \in [\![\psi^k(false)]\!]^S e'$ as stated above. Thus this point is satisfied by $o''$, and all other obligations are not modified.

6. One new edge is added to $T$. It satisfies Point 6 of the invariant as $\phi' = \mu v.\ \psi$ and the state of $o''$ is the same as the one of $o'$ by construction.

7. This point is still satisfied as no obligation with $\Box_i\ \phi''$ has been added to $O$.

$\phi' = \nu v.\ \psi$ The obligation $o' = \langle q', \phi', e'\rangle$ is removed from *pending* and added to $O$, and a new obligation $o'' = \langle q', \psi(\phi'), e'\rangle$ is created. Then $\langle o', o''\rangle$ is added to $T$, and if $o''$ was not already in $O$, it is added to *pending*. The invariant is still satisfied after execution of the body:

2. All obligations of $O$ before executing the body stay locally consistent as they are not modified, nor their successors. $o'$ is locally consistent in $\langle O \cup pending, T\rangle$ as, after execution, $o''$ is in $O \cup pending$, and is the only successor of $o'$ as $o'$ had no successor through $T$ before executing the body (by Point 3 of the invariant and the fact that $o'$ was in *pending*). Also, the two obligations share the same environment $e'$.

3. Either $o''$ was already in $O \cup pending$, and the point remains satisfied, or it is a new obligation, thus has no successor through $T$, and the point remains satisfied, too.

4. As the state of $o''$ is the same as $o'$ by construction, this point is still satisfied.

5. $q' \in [\![\psi(\phi')]\!]^S e'$ as $q' \in [\![\phi']\!]^S e$. Thus this point is satisfied by $o''$, and all other obligations are not modified.

6. One new edge is added to $T$. It satisfies Point 6 of the invariant as $\phi' = \nu v.\ \psi$ and the state of $o''$ is the same as the one of $o'$ by construction.

7. This point is still satisfied as no obligation with $\Box_i\ \phi''$ has been added to $O$.

Finally, after exiting the loop, $E = \langle O, T\rangle$ is an adequate explanation for $q \in [\![\phi]\!]^S e$ as (1) $\langle q, \phi, e\rangle \in O$ by the invariant, (2) $E$ is consistent by the invariant and the facts that $pending = \varnothing$ and all $o \in O$ are locally consistent in $E$, and (3) $E$ matches $S$ as Points 4 to 7 of the invariant imply that $E$ matches $S$ when $pending = \varnothing$.

The proof above showed that whenever the algorithm terminates, the result is correct. Let us show now that it always terminates. It is easy to show that the algorithm builds a finite number of different obligations: (1) the states of these obligations all belong to the finite set of states $Q$ by Point 4 of the invariant, (2) the formulas of these obligations are sub-formulas of the original $\phi$, augmented with $\psi^k(false)$ for each least fixpoint sub-formula and some finite $k \geq 0$, and $\psi(\phi')$ for each greatest fixpoint formula $\phi'$, and (3) the environments are all the same as the explanation is consistent. There are thus at most $|Q| \times |Sub|$ obligations, where $Sub$ is the set of sub-formulas discussed above.

Furthermore, a variant of the **while** loop is $(|Q| \times |Sub|) - |O|$: this number decreases through every execution of the body of the loop. Indeed, the body of the loop removes one element $o'$ of *pending* and add it to $O$. As $o'$ is not in $O$ before the execution of the loop by Point 8 of the invariant, the size of $O$ increases by 1 after every execution. As the size of $O$ is upper bounded by $|Q| \times |Sub|$, the variant above cannot be smaller than 0 and decreases by 1 through every execution of the loop body, thus there can be only a finite number of iterations and the algorithm always terminates. This ends to proof of correctness of the algorithm. $\square$