A Formal Framework for the Analysis of Human-Machine Interactions

Sébastien Combéfis[1]

[1]University of Louvain (UCLouvain)

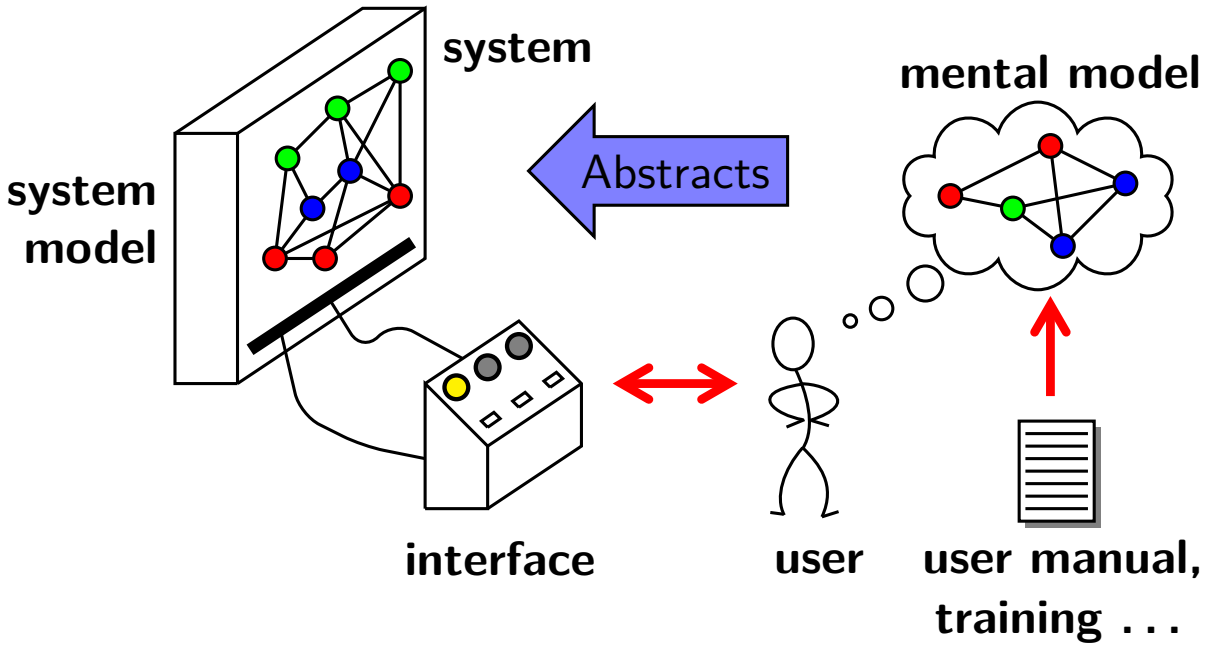
ICT, Electronics and Applied Mathematics Institute (ICTEAM)

November 17, 2011

Louvain Verification Lab

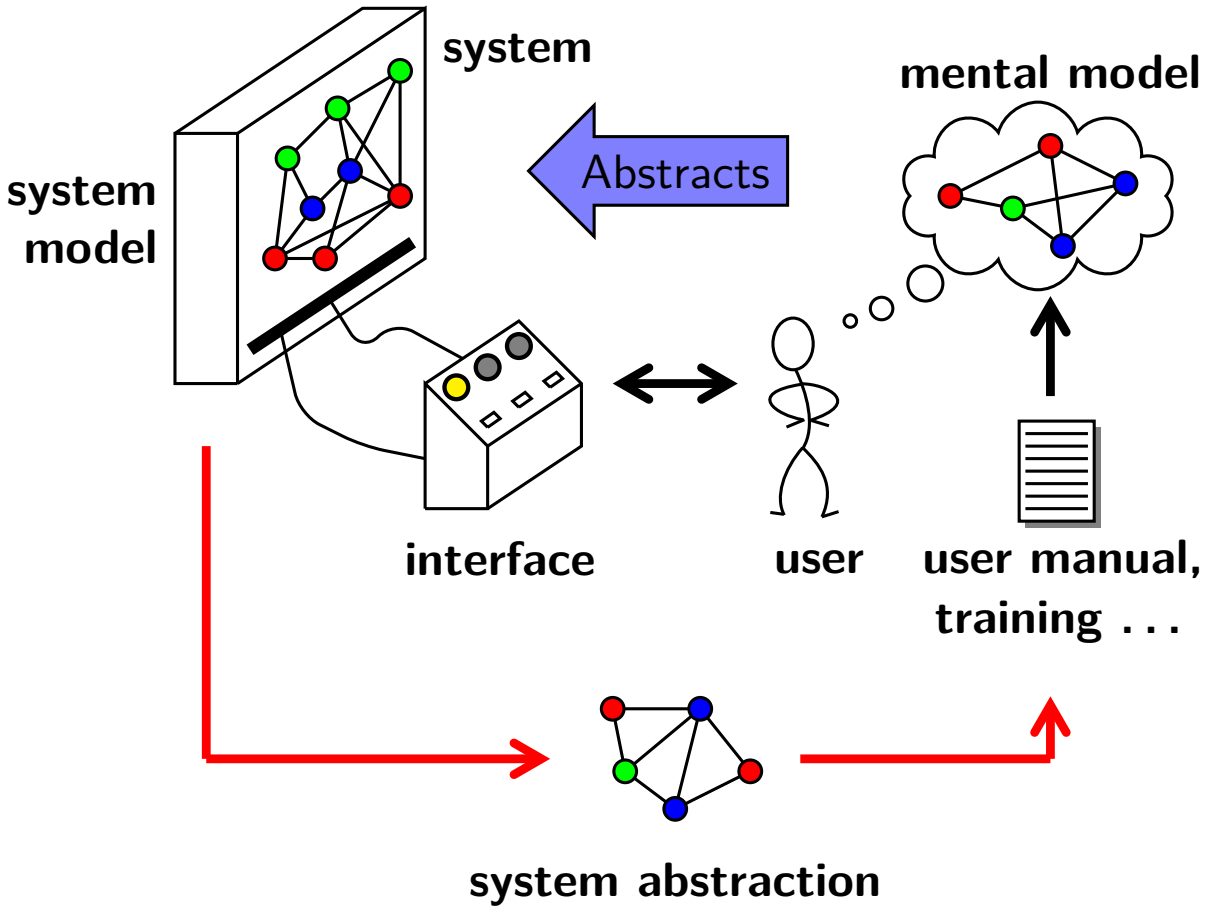
icteam

Human-Machine Interaction



- What is a good system abstraction?

Human-Machine Interaction

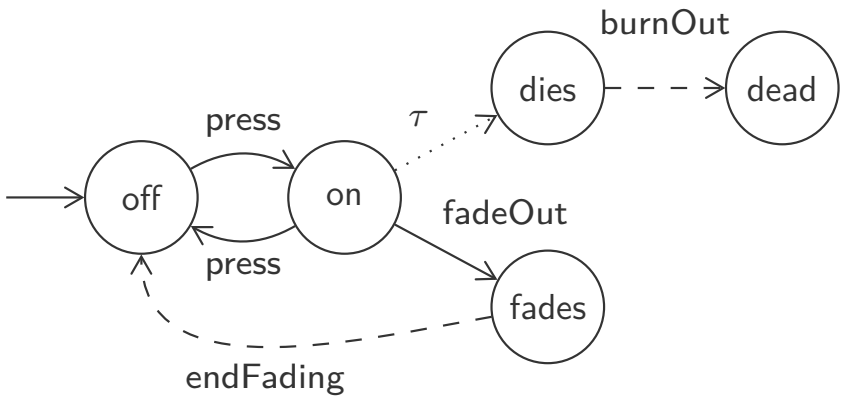


- How can such an abstraction be automatically generated?

Outline

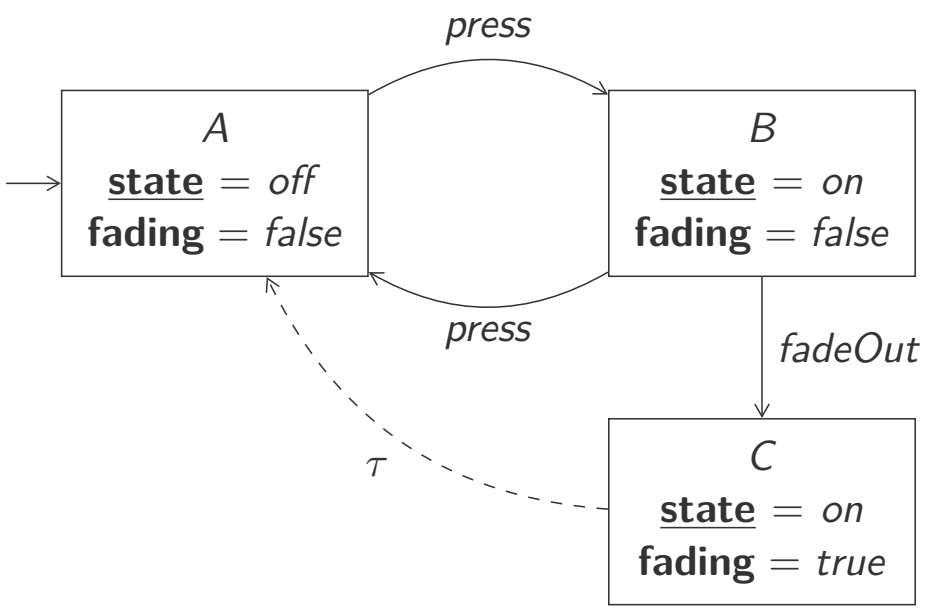
Modelling with HMI-LTS

- Multiple possibilities to model system and mental models
- In this work, modelling done with HMI-LTS
- Distinction between the actions:
  - Commands are executed by the user on the system (inputs)
  - Observations are executed by the system and observed by the user (outputs)
  - τ is the internal action, unobservable by the user



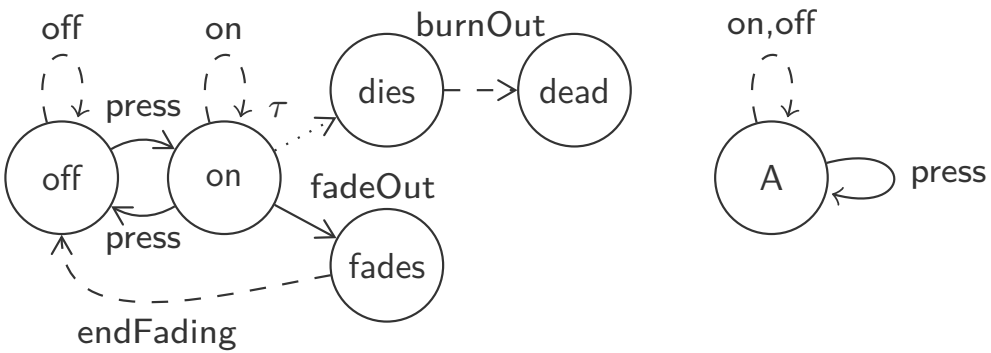
Adding State Information

- With HMI-LTS, all the behaviour of the system is represented thanks to the transitions
- Sometimes, useful to have information inside the states (with ADEPT models for example)
  - Variables are added in each state
  - Variables can be visible or non visible



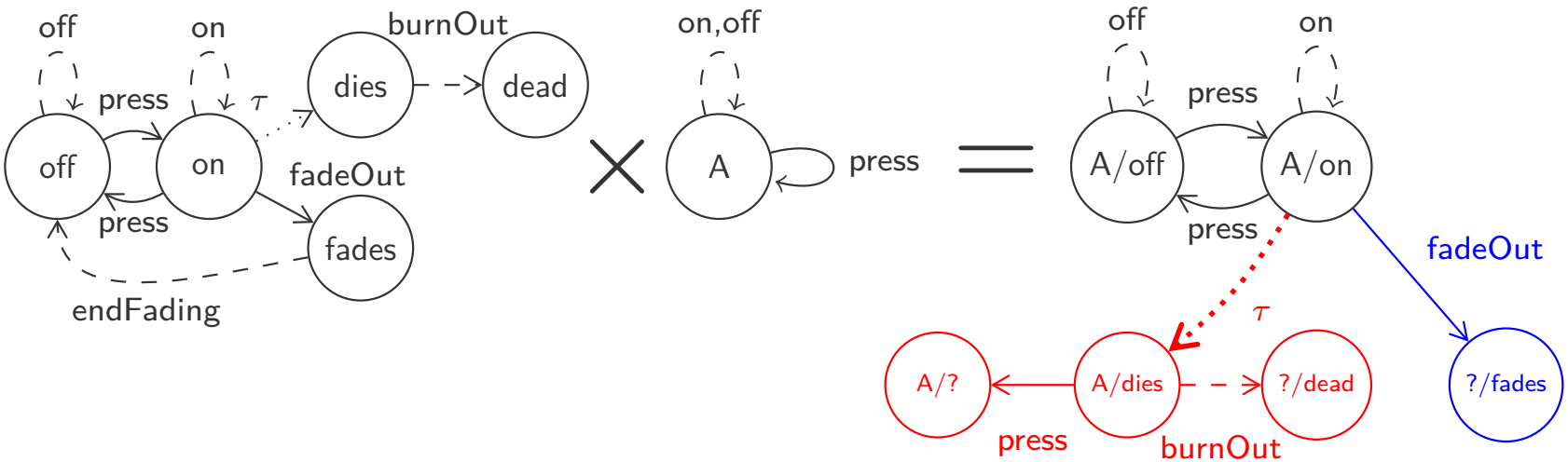
Verifying Good Abstraction

- Several ways to characterize a good system abstraction
- In this work, property based on dynamic consideration of interaction
- A good abstraction is one which allows the operator to control properly the system



Verifying Good Abstraction

- Several ways to characterize a good system abstraction
- In this work, property based on dynamic consideration of interaction
- A good abstraction is one which allows the operator to control properly the system



Full-control Property

- Full-control property captures good system abstraction.
- During the interaction between user and system:
  - The user should know exactly the available commands ...
  - ... and at least all the possible observations
- Given a system $\mathcal{M}_M = \langle S_M, s_{0_M}, \mathcal{L}^c, \mathcal{L}^o, \rightarrow_M \rangle$ and an abstraction for it $\mathcal{M}_U = \langle S_U, s_{0_U}, \mathcal{L}^c, \mathcal{L}^o, \rightarrow_U \rangle$:

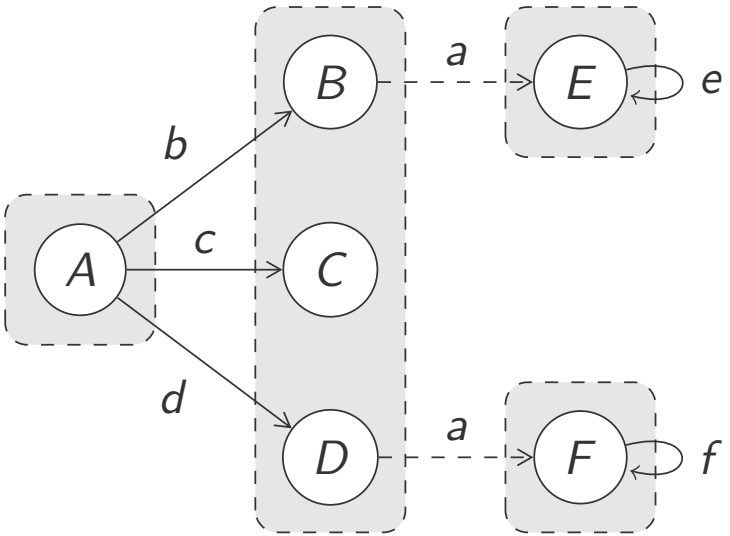
$$\mathcal{M}_U \,\mathrm{fc}\, \mathcal{M}_M \text{ iff :}$$

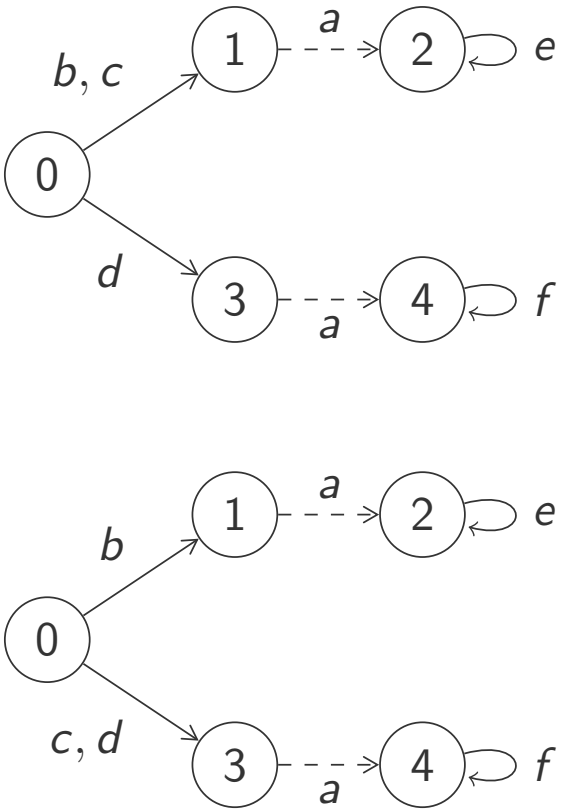
$$\forall \sigma \in \mathcal{L}^{co*} \text{ such that } s_{0_M} \stackrel{\sigma}{\Longrightarrow} s_M \text{ and } s_{0_U} \stackrel{\sigma}{\longrightarrow} s_U :$$

$$A^c(s_M) = A^c(s_U) \quad \wedge \quad A^o(s_M) \subseteq A^o(s_U)$$

Full-control Abstraction Example

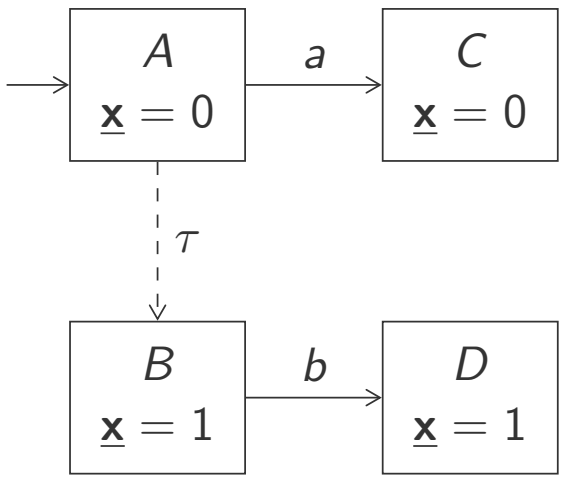
- Multiple minimal full-control abstraction are possible

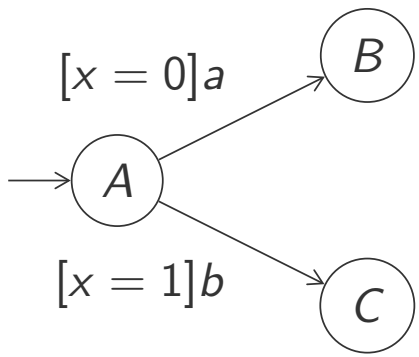




Full-control Abstraction for Richer Models

- Abstraction is enriched with guards on visible variables





Full-control Property Variants

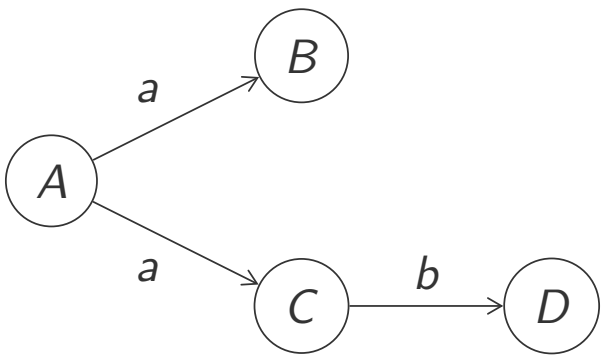
- The operator may not want to know every possible commands

$$A^c(s_M) \supseteq A^c(s_U) \quad \wedge \quad A^o(s_M) \subseteq A^o(s_U)$$

- The abstraction may be restricted to some functionalities of the system

Existence: Full-Control Determinism

- Full-control system abstraction only exist for systems which are full-control deterministic
- Any sequence $\sigma \in \mathcal{L}^{co*}$ that the user will execute on the system must always lead the system to states allowing the same set of commands



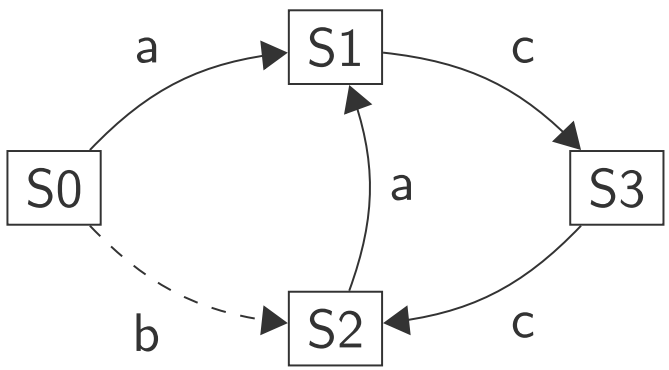
System Abstraction Generation

- **Goal**
  - Given the model of a system, automatically generate a minimal abstraction for the system, satisfying some properties
- **Motivation**
  - Extract the minimal behaviour of the system, so that it can be controlled properly (without surprise, mode awareness, . . .)
  - Help to build artifacts: manuals, procedures, trainings, . . .
  - If such abstraction does not exist, provide feedback to help redesigning the system

→ Reduction-based and learning-based algorithms for generation

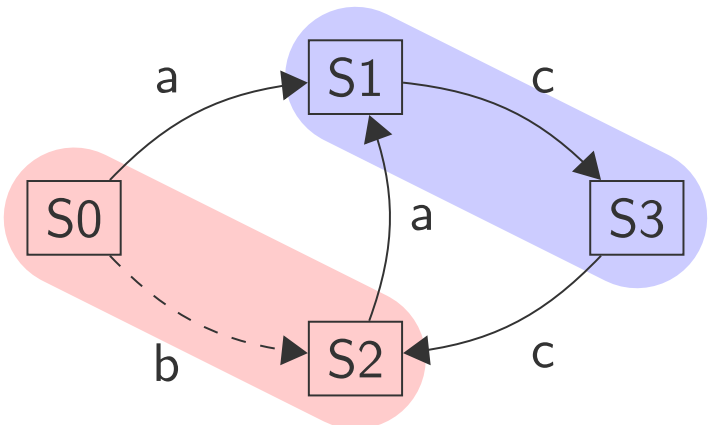
Reduction-Based Approach

- Method based on a variant of the Paige-Tarjan reduction algorithm which will partition the system by separating states which exhibit different behaviour, based on a similarity relation



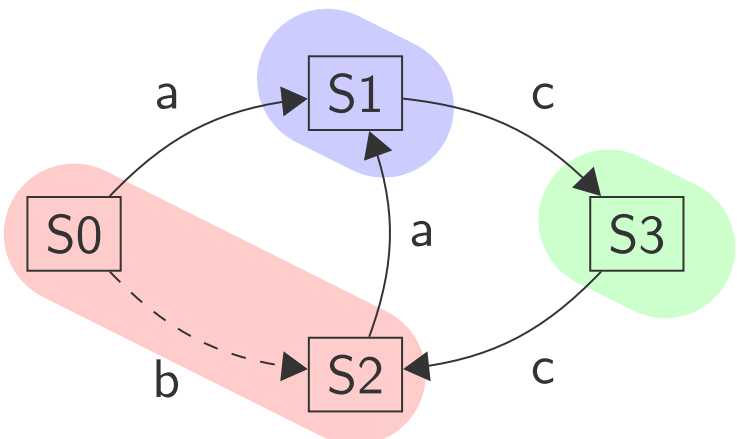
Reduction-Based Approach

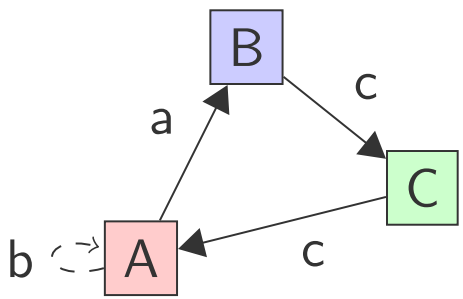
- Method based on a variant of the Paige-Tarjan reduction algorithm which will partition the system by separating states which exhibit different behaviour, based on a similarity relation



Reduction-Based Approach

- Method based on a variant of the Paige-Tarjan reduction algorithm which will partition the system by separating states which exhibit different behaviour, based on a similarity relation



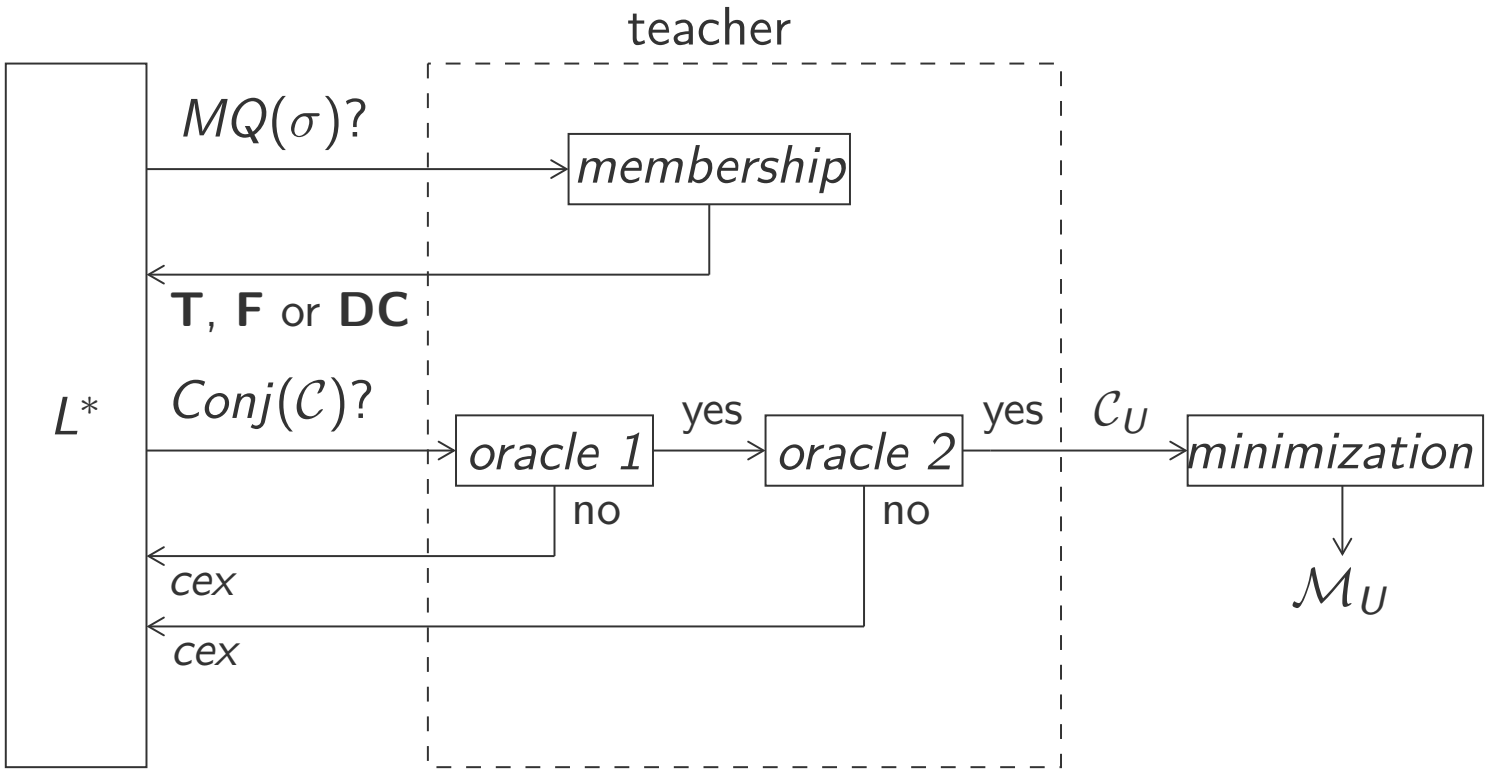


Growing an Abstraction from Tasks' Model

- Start from a tasks' model representing what the operator has to perform with the system
- Grow the model so that it satisfies full-control property

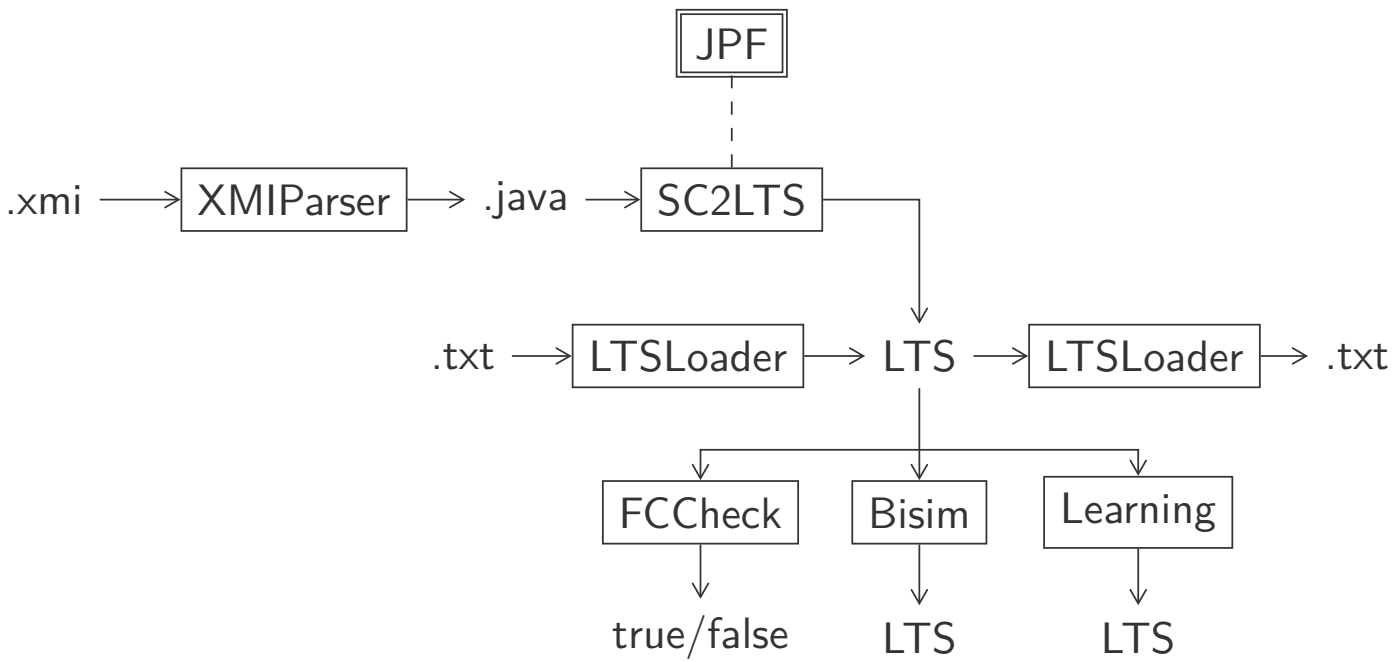
Learning-Based Approach

- Using a learning algorithm to learn a 3DFA capturing all the possible full-control system abstractions (variant of L^*)



Tool Support : `jpf-hmi` extension

- A JavaPathfinder extension `jpf-hmi`
- It supports modelling, verification and generation of system abstraction for HMI
- Models expressed with statecharts



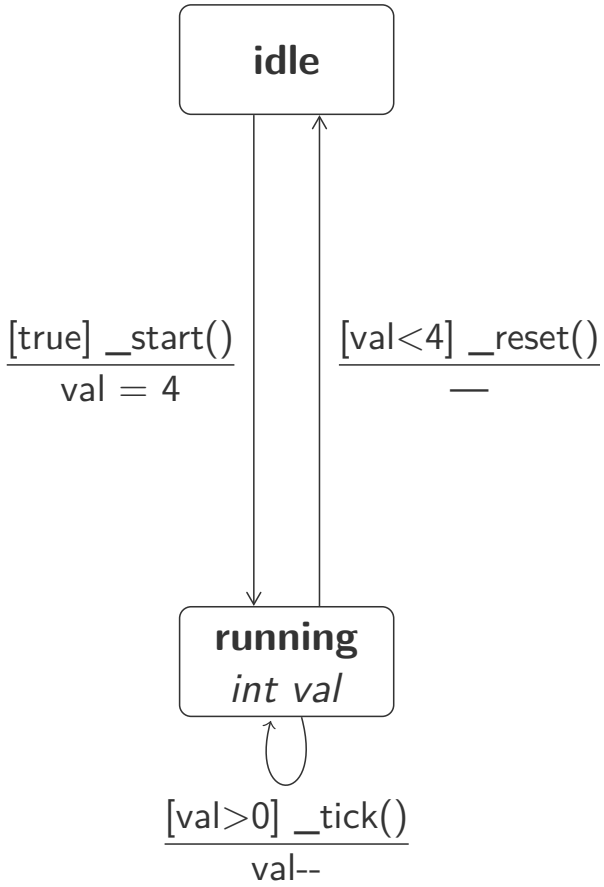
Model example

```
public class CountDown extends Model {
   @Override
   public List<Action> getActions() {
      List<Action> actions = new ArrayList<Action>();
      actions.addAll (Arrays.asList (
         new Action ("_start()", COMMAND),
         new Action ("_reset()", COMMAND),
         new Action ("_tick()", OBSERVATION)
      ));
      return actions;
   }

   public static class Behaviour extends State {
      private static final int MAX = 4;

      public class Idle extends State {
         public void _start() {
            running.val = MAX;
            setNextState (running);
         }
      } Idle idle = makeInitial (new Idle());

      public class Running extends State {
         // ...
      } Running running = new Running();
   }
}
```



Conclusion and further work

- **Conclusion**
  - Full-control property captures good abstraction
  - Two different approach to automatically generate minimal full-control mental models
  - Framework developed within Java Pathfinder and integration with ADEPT toolset
- **Further work**
  - Experiments the different variants of mental model generation
  - Develop abstraction of system models
  - Adapt theory to new semantics of system and mental models
  - Perform analysis on larger models (autopilot)
  - Consider variants of full-control property (symetric, tasks...)