

Verification of Embedded Software from Mars to Actions

Charles Pecheur, UC Louvain
(formerly RIACS / NASA Ames)



From Mars ...



... to Actions

MODULE agent

IVAR move : boolean;

VAR count : 0..10;

ASSIGN

init(count) := 0 ;

next(count) := case

move & count < 10: count + 1;

1 : count;

esac;

DEFINE win := (count=10);

MODULE main

VAR alice : agent;

VAR bob : agent;

SPEC !EAX (bob.move) bob.count = 0

SPEC AAX (bob.move & alice.move) (bob.count > 0 & alice.count > 0)

SPEC AAF (bob.move) bob.win

Outline

Model-Based Autonomy and Diagnosis

Verification of Model-Based Controllers

Verification of Diagnosability

Symbolic Verification with Knowledge

Symbolic Verification with Actions

Conclusions

Autonomy (at NASA)

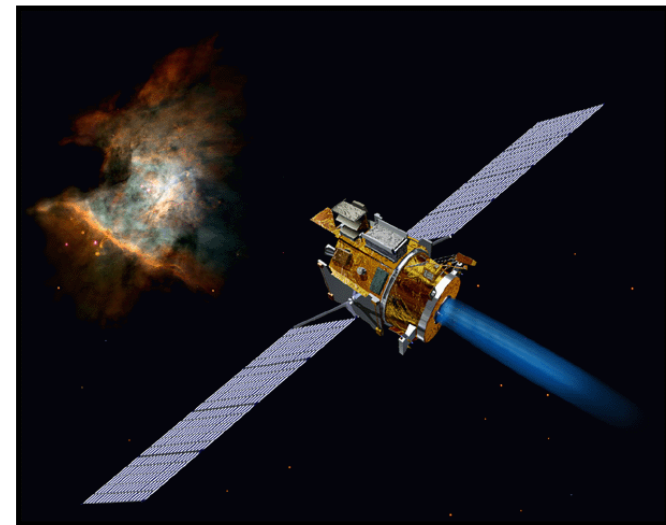
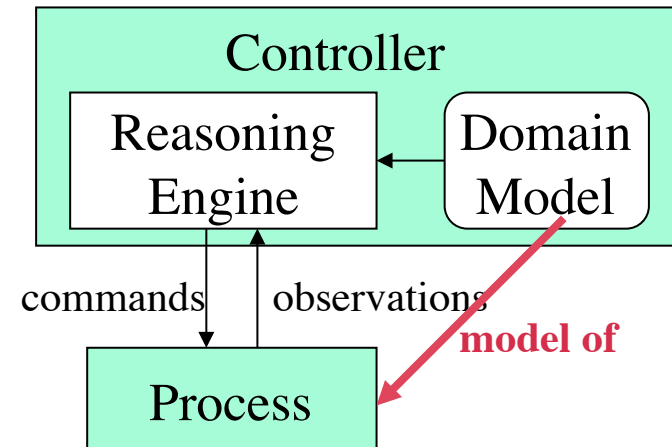
Autonomous spacecraft = on-board intelligence (= AI)

- **Goal:** Unattended operation in an unpredictable environment
- **Approach:** **model-based reasoning**
- **Pros:** smaller mission control crews, no communication delays/blackouts
- **Cons:** **Verification and Validation ???**
Much more complex, huge state space
- **Better verification is critical for adoption**



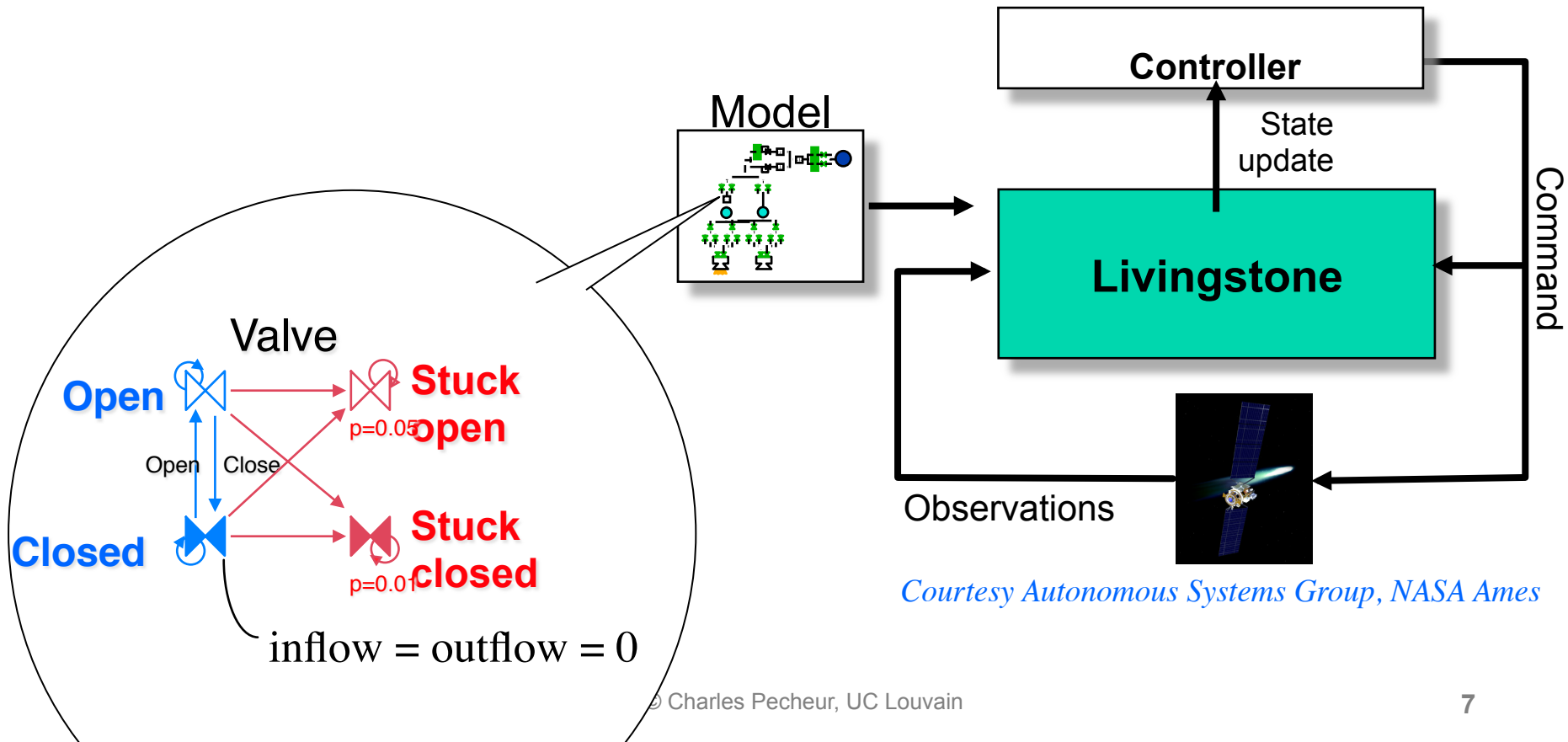
Model-Based Autonomy

- Based on AI technology
- Generic **reasoning engine** + application-specific **model**
- Model describes (normal and faulty) behaviour of the process
- Engine selects control actions "on-the-fly" based on the model
 - ... rather than pre-coded decision rules
 - better able to respond to unanticipated situations

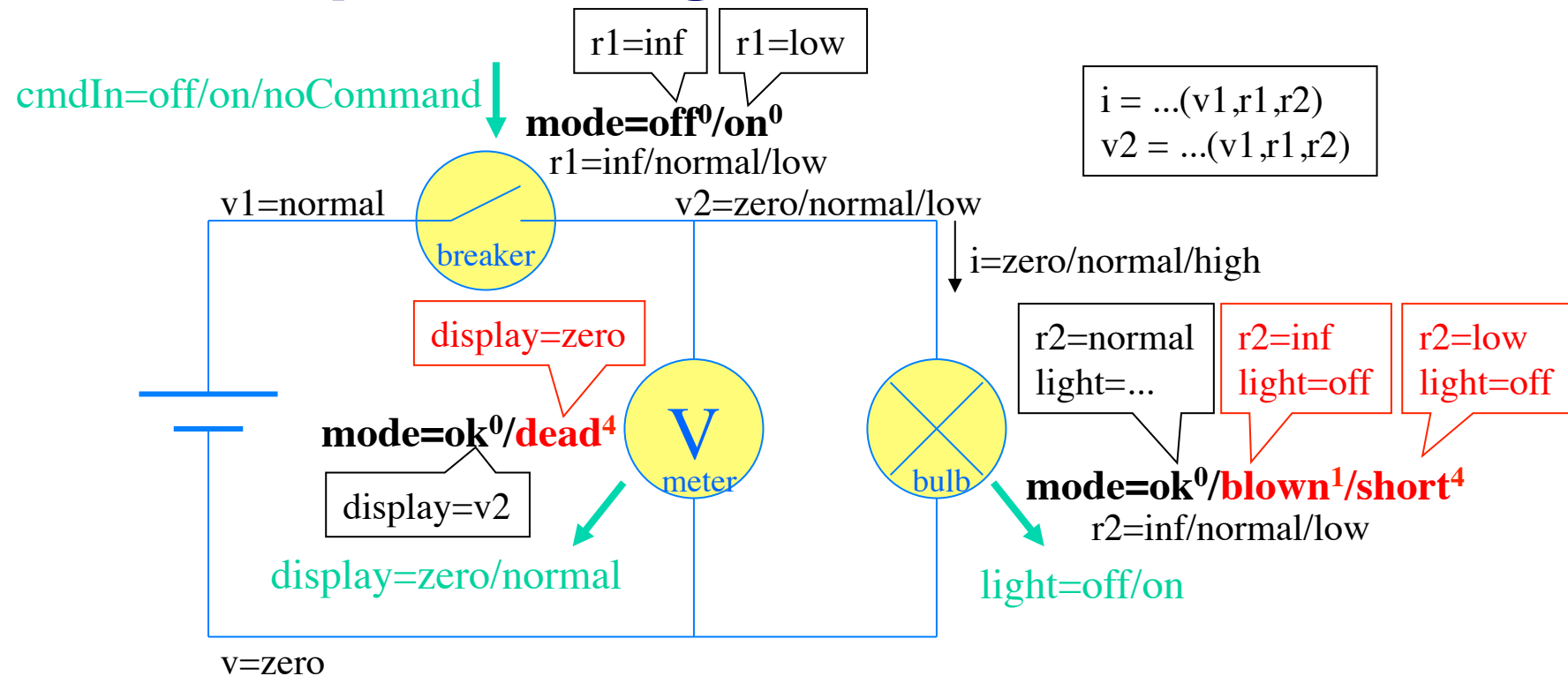


Livingstone

- **Model-based diagnosis system** from NASA Ames
 - i.e. an advanced state estimator
- Uses a discrete, qualitative model to reason about faults
 - => naturally amenable to formal analysis



A Simple Livingstone Model



Goal: determine **modes** from **observations**
Generates and tracks *candidates*

breaker	bulb	meter	rank
off ⁰	ok ⁰	ok ⁰	0
off ⁰	ok ⁰	blown ¹	1
on ⁰	dead ⁴	short ⁴	8

Outline

Model-Based Autonomy and Diagnosis

Verification of Model-Based Controllers

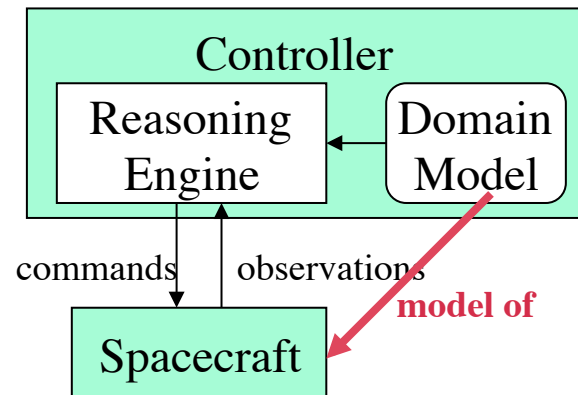
Verification of Diagnosability

Symbolic Verification with Knowledge

Symbolic Verification with Actions

Conclusions

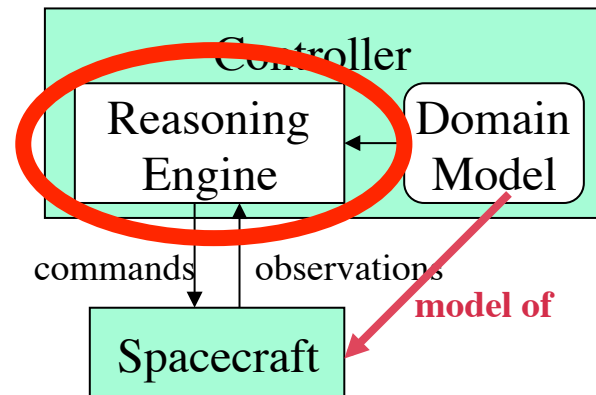
Verify Model-Based Control?



Of course, but what exactly?

- The model?
- The engine?
- The whole controller?
- **All of the above!**

Verification of the Engine

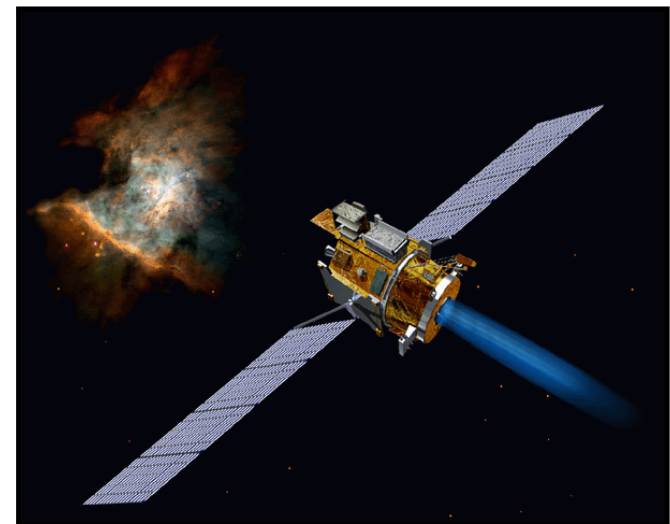


- A (technically complex) computer program
 - Use traditional software verification approaches
 - Maybe full-blown proof on core algorithms
- Generic, re-used across applications
 - More likely to be stable and trustable
 - Like compilers, interpreters, virtual machines, etc

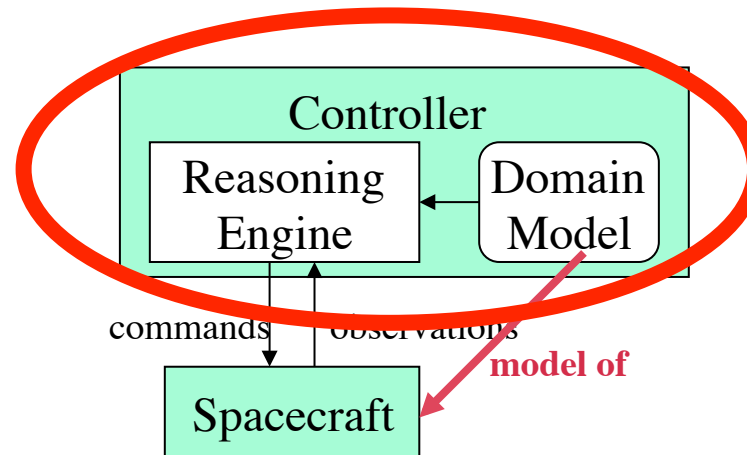
The RAX Bug

Remote Agent Experiment (1999)

- **cause** : missing critical section in concurrent program
- **effect** : race condition and deadlock in flight
 - in supervised experiment, no mission damage
- **solution** : model checking
 - a similar bug was found before flight using SPIN on another part of the code
 - See [Havelund et al. 2000]



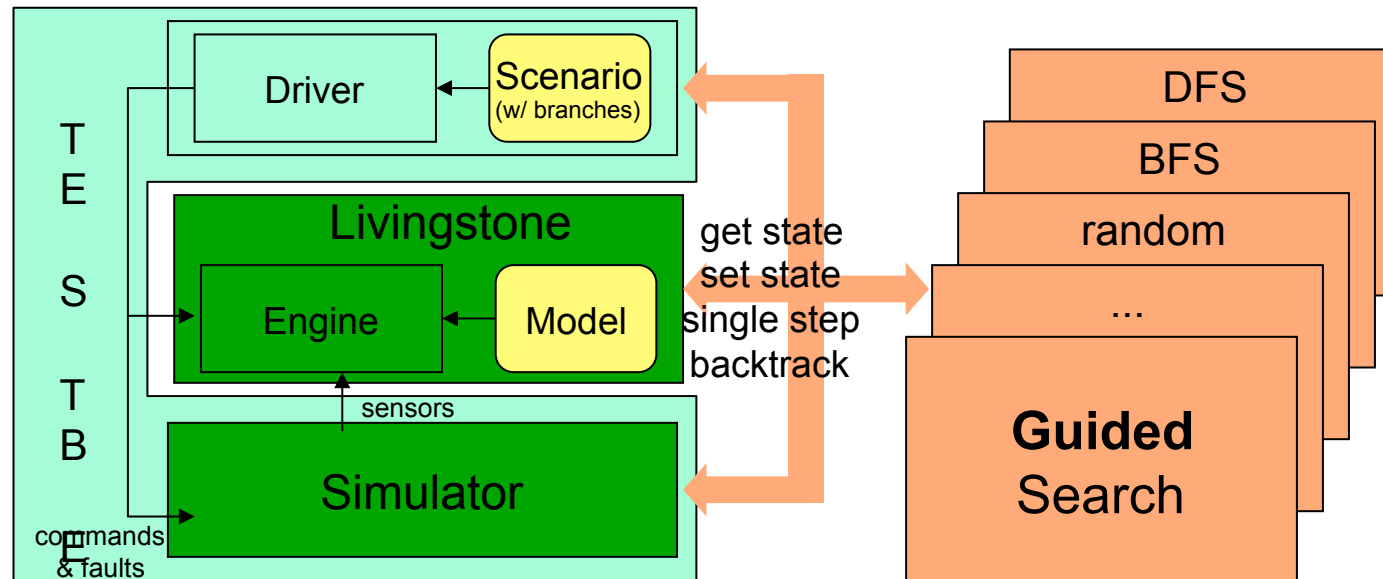
Verification of the Controller



- good model + good engine \nRightarrow good controller
 - Heuristics in engine, simplifications in model
- System-level verification
 - Controller as black (or grey) box
 - Need a model of the environment (test harness)
 - Applicable to others than model-based

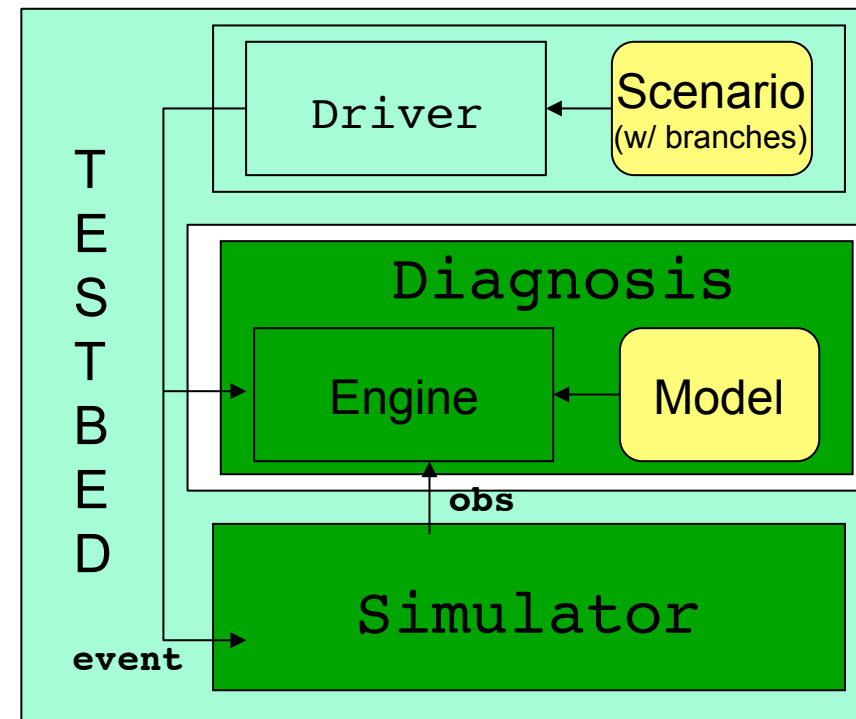
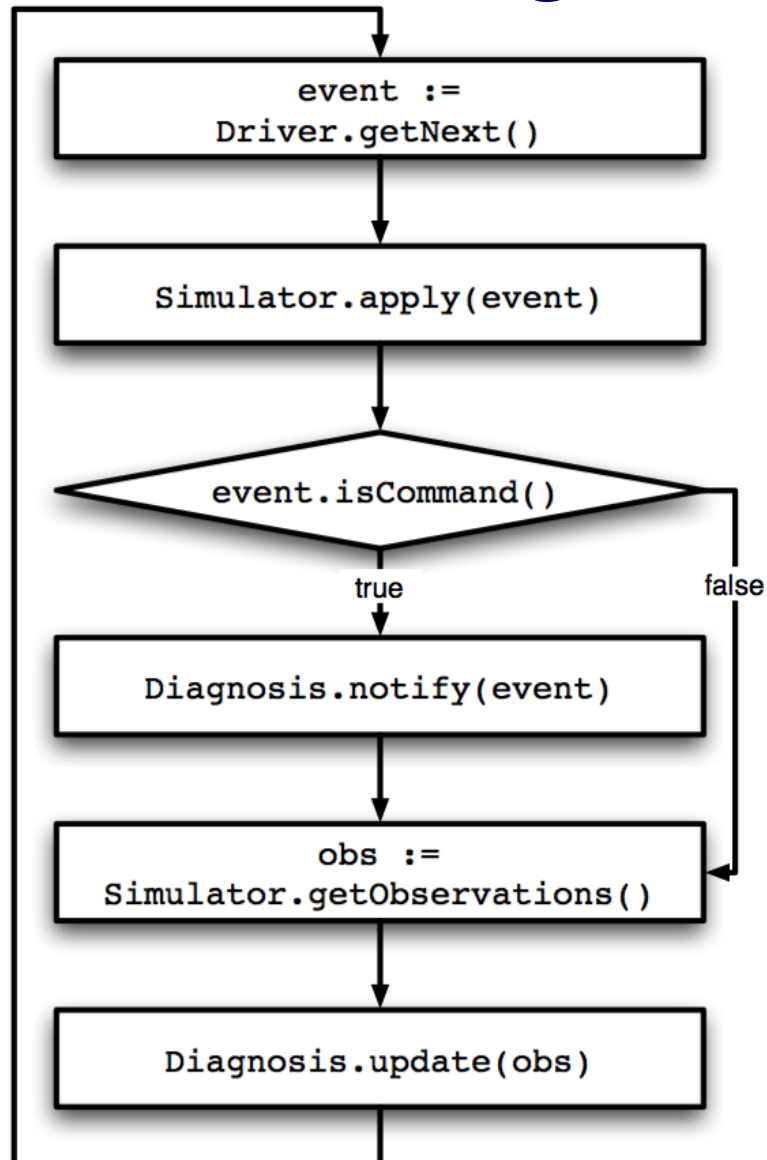
Livingstone PathFinder

with Tony Lindsey (QSS / NASA Ames)



- An advanced testing/simulation framework for Livingstone applications
 - Executes the **Real Livingstone Program** in a simulated environment (testbed)
 - **Instrument** the code to be able to **backtrack** between alternate paths
- **Modular** architecture with generic APIs (in Java)
 - allows different diagnosers, simulators, search algorithms and strategies, error conditions, ...
- See TACAS'04 paper

One Diagnosis Step

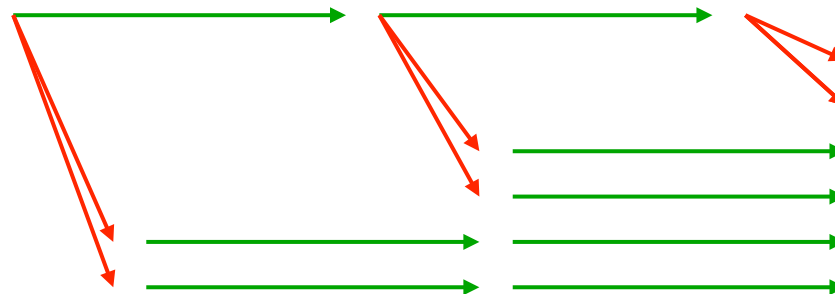


LPF Scenario Example

```

mix {
  "command test.sv02.valveCmdIn=close";
  "command test.sv02.valveCmdIn=open";
  ...
} and {
  choose
    "fault test.forwardLO2.mode=unknownFault"; or
    "fault test.mpre101p.mode=faulty"; or
    ...
}

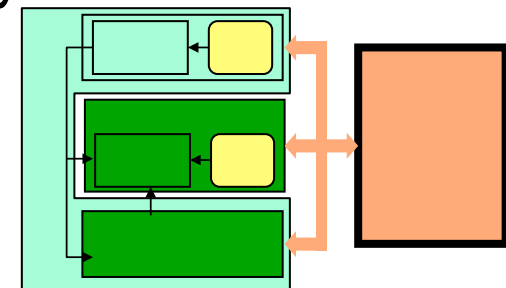
```



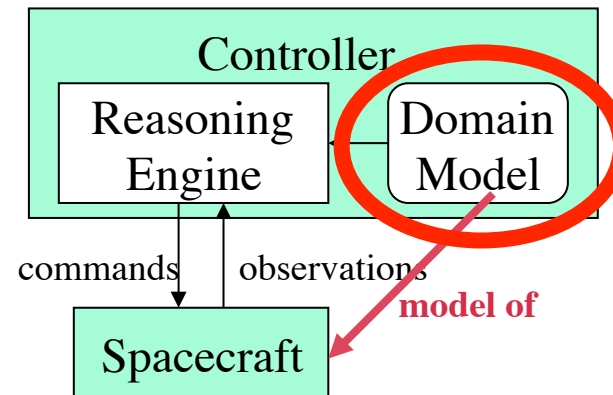
- Sequence of commands || choice of faults
- "default" scenario, can be generated automatically

LPF Search

- The whole testbed is seen as a transition system
- API to enumerate transitions, backtrack, get/set state
 - Shared with Java PathFinder (v.2)^[Visser et al. 00]
 - Principle inspired from OPEN/CAESAR^[Garavel 98]
- Search engine fixes exploration strategy
 - Depth-First
 - Breadth-First
 - **Heuristic**
 - Others are possible (random, pattern-based, interactive)
- + Halting conditions (for any strategy)
 - Find first / all / shortest error trace(s)



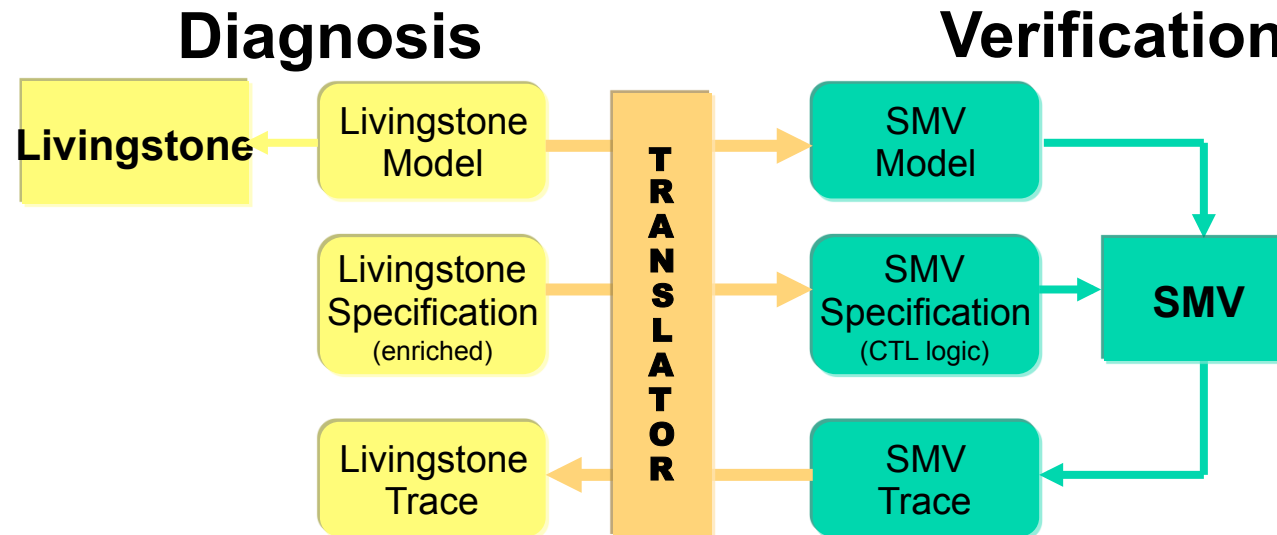
Verification of the Model



- This is the "application code"
 - where the development effort (and bugs) are
- Abstract, concise, amenable to formal analysis
 - this is another benefit of model-based approaches
 - ... or model-based design in general
- Use **symbolic model checking**

Livingstone-to-SMV Translator

Joint work with Reid Simmons (Carnegie Mellon)

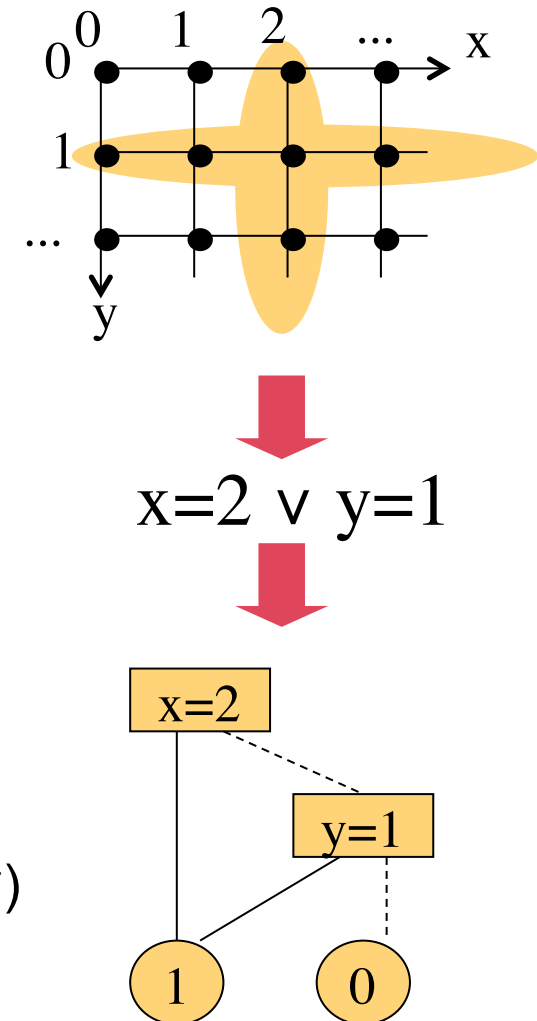


- A translator that converts Livingstone models, specs, traces to/from SMV (in Java)
 - SMV: symbolic model checker (both BDD and SAT-based)
allows exhaustive analysis of very large state spaces (10^{50+})
- Hides away SMV, offers a **model checker for Livingstone**
- Enriched specification syntax (vs. SMV's core temporal logic)
- Graphical interface, integration in Livingstone development tools

SMV / NuSMV

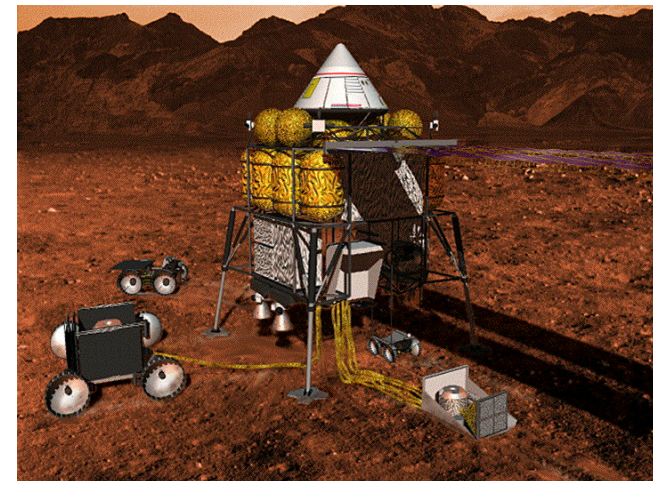
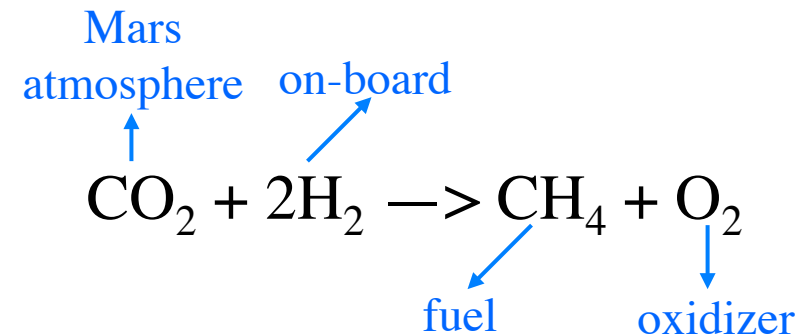
Mainstream **symbolic** model checker

- Original SMV from Carnegie Mellon, currently NuSMV from IRST (and Cadence SMV)
- Rich modeling **language**
- Many **features and options**
- Uses **symbolic** computation over **boolean** encoding
 - using BDDs or SAT (bounded)
 - finite models
 - Can handle **very large state spaces** (10^{50+})



In-Situ Propellant Production

- Use atmosphere from Mars to make fuel for return flight.
- Livingstone controller developed at NASA KSC.
- Components are tanks, reactors, valves, sensors...
- Exposed improper flow modeling.
- Latest model is 10^{50} states.



Verification of Diagnosis Models

- Coding Errors
 - e.g. Consistency, well-defined transitions, ...
 - Generic
 - Compare to Lint for C
- Model Correctness
 - Expected properties of modeled system
 - e.g. flow conservation, operational scenarios, ...
 - Application-specific
- **Diagnosability**
 - Are faults detectable/diagnosable?
 - Given available sensors
 - In all/specific operational situations (dynamic)

Outline

Model-Based Autonomy and Diagnosis

Verification of Model-Based Controllers

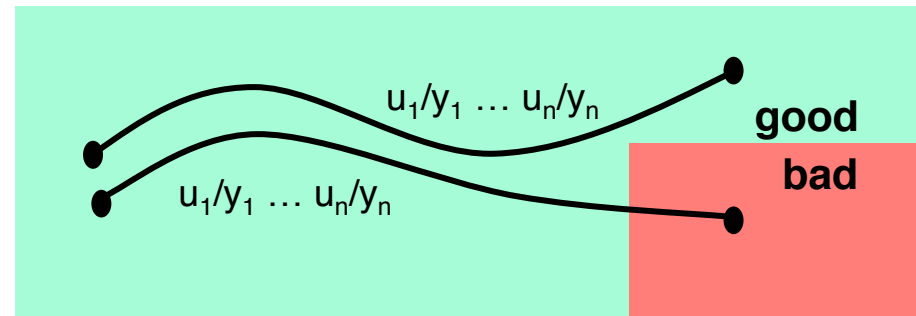
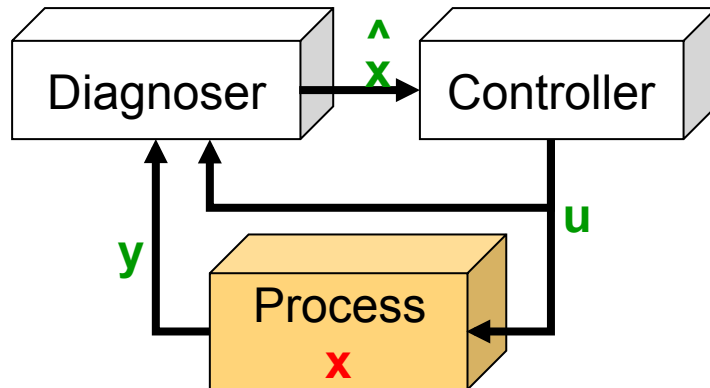
Verification of Diagnosability

Symbolic Verification with Knowledge

Symbolic Verification with Actions

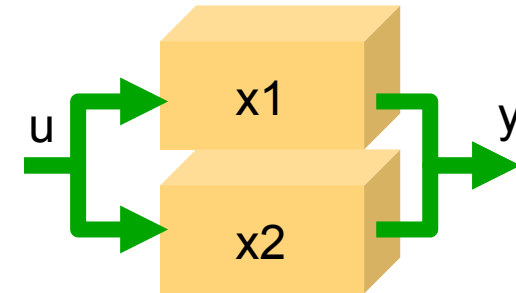
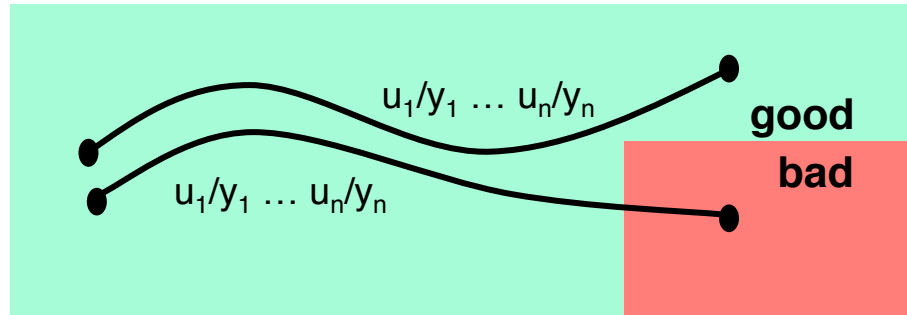
Conclusions

Diagnosability



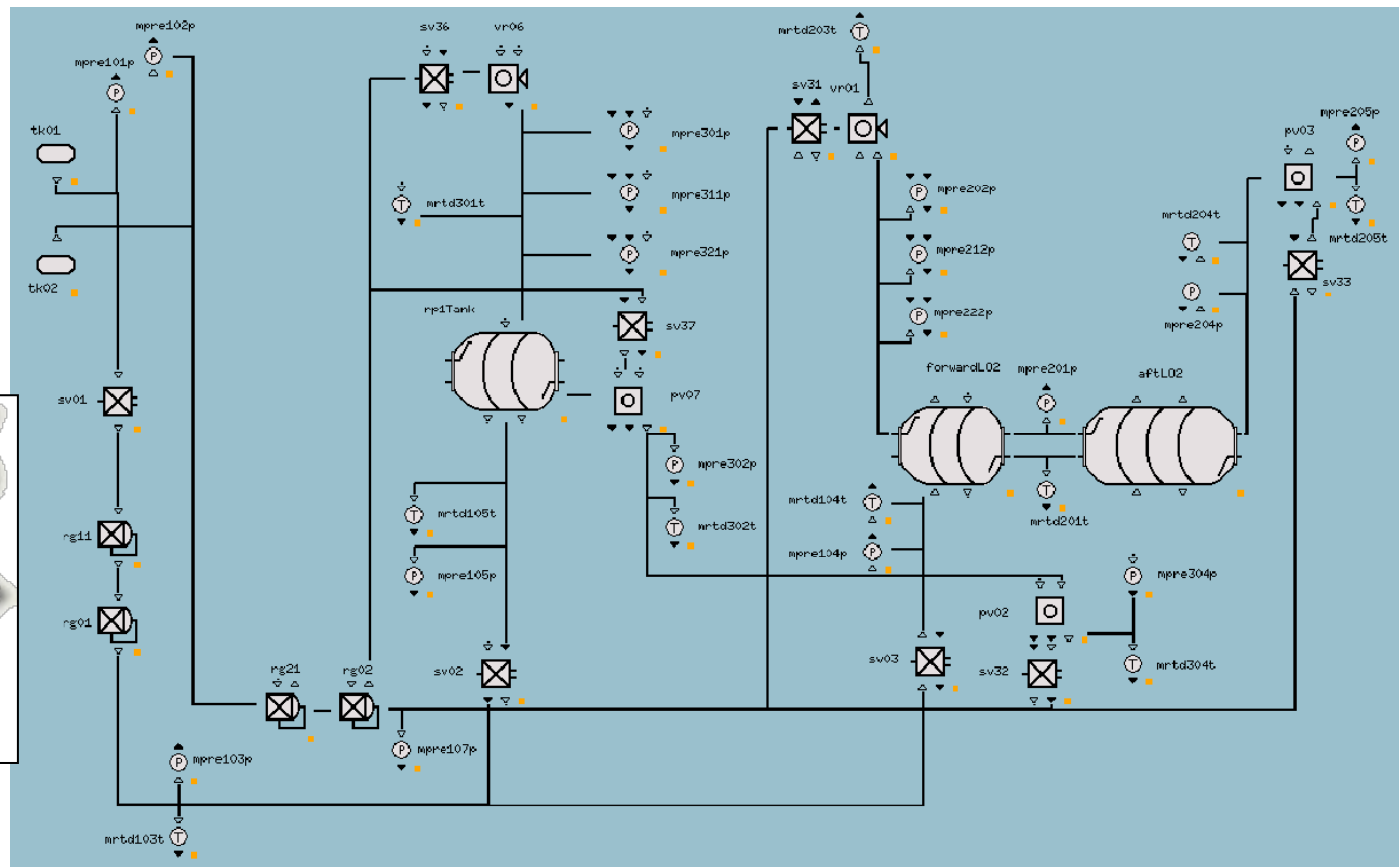
- **Diagnosis:** estimate the hidden state **x** (incl. failures) given observable commands **u** and sensors **y**.
- **Diagnosability:** Can (a smart enough) *Diagnoser* always tell when *Process* comes to a **bad** state?
- **Property of the Process** (not the Diagnoser)
 - even for non-model-based diagnosers
 - but analysis needs a (process) model

Verification of Diagnosability



- **Intuition:** **bad** is diagnosable if and only if there is no pair of trajectories, one reaching a **bad** state, the other reaching a **good** state, with identical observations.
 - or some generalization of that: (context, two different faults, ...)
- **Principle:**
 - consider two concurrent copies $x1$, $x2$ of the process, with coupled inputs u and outputs y
 - check for reachability of (**good**($x1$) && **bad**($x2$))
- Back to a classical (symbolic) model checking problem !
- Supported by Livingstone-to-SMV translator

- Propulsion IVHM Technology Experiment (ARC, GRC)
- Livingstone applied to propulsion feed system of space vehicle
- Livingstone model is $4 \cdot 10^{33}$ states



PITEX Diagnosability Error

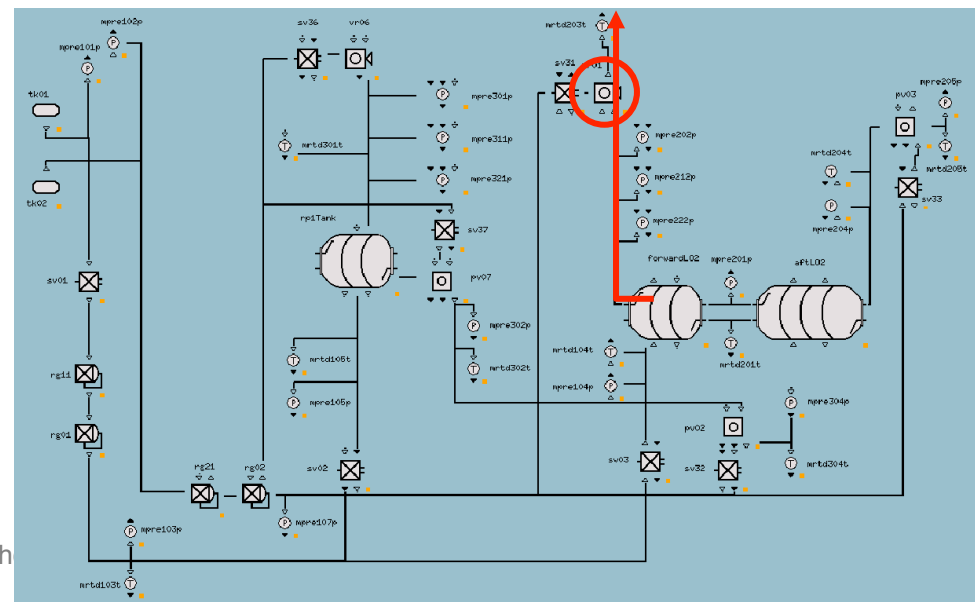
with Roberto Cavada (IRST, NuSMV developer)

- "Diagnosis can decide whether the venting valve VR01 is closed or stuck open (assuming no other failures)"

INVAR !test.multibroken() & twin(!test.broken())

VERIFY INVARIANT !(test.vr01.mode=stuckOpen & twin(test.vr01.valvePosition=closed))

- Results show a pair of traces with same observations, one leading to **VR01 stuck open**, the other to **VR01 closed**. Application specialists fixed their model.



Outline

Model-Based Autonomy and Diagnosis

Verification of Model-Based Controllers

Verification of Diagnosability

Symbolic Verification with Knowledge

Symbolic Verification with Actions

Conclusions

Epistemic Logic

- Reasoning about knowledge
 - $K_a \varphi$ = agent a **knows** φ
- Interpreted over an **Interpreted System (IS)**
 - **Transition system** T +
 - **Observation functions** $\text{obs}_a(\sigma)$ over runs σ of T
 - $K_a \varphi$ holds after σ iff φ holds after all σ' such that $\text{obs}_a(\sigma) = \text{obs}_a(\sigma')$
- **CTLK** = **temporal + epistemic** logic

Observation Function

- **In general** : agents reason about “everything they have seen so far” (total recall)
 - $\text{obs}_a(\sigma)$ over **runs** σ
 - memory built into the logic
 - model checking hard to undecidable
- **Observational view** : agents reason about the current state only
 - $\text{obs}_a(s)$ over **states** s
 - memory explicit in the model
 - **symbolic model checking** can be generalized from CTL to CTLK

Diagnosability and CTLK

joint work with Franco Raimondi (UC London)

Considering the diagnoser as an agent D observing the system,

Fault F is diagnosable

iff

$$\text{AG } (K_D F \vee K_D \sim F)$$

- **Diagnosability** can be framed as a **temporal epistemic** model-checking problem
- Caveat : general diagnosability requires total recall
 - or explicit (bounded) memory of observations

From CMAS to SMV

- CMAS : symbolic model checker for CTLK
 - developed by Franco Raimondi
 - BDD-based
 - Good performance but very crude modelling language
- Could we do CTLK in NuSMV?
 - Leverage SMV's rich modelling language
 - Re-use models generated from Livingstone
- Need a reduction from CTLK to (enhanced?) CTL

Outline

Model-Based Autonomy and Diagnosis

Verification of Model-Based Controllers

Verification of Diagnosability

Symbolic Verification with Knowledge

Symbolic Verification with Actions

Conclusions

From Knowledge to Actions

- The observation function $\text{obs}_a(s)$ induces an **accessibility** (equivalence) **relation** \sim_a over reachable states s

$$s \sim_a s' \text{ iff } \text{obs}_a(s) = \text{obs}_a(s')$$
- An **interpreted system** is a Kripke structure with several transition relations $\rightarrow, \sim_{a1}, \dots, \sim_{an}$
- Or equivalently, a **labelled transition system** (LTS) over an action alphabet $\{t, a1, \dots, an\}$
- **Corresponding reduction of CTLK?**

Action-Based Logics

- Large body of published work in **action-based temporal logics** (applicable to LTS)
 - ACTL [deNicola-Vaandrager], ACTL*, Hennessy-Milner, etc.
 - Do not quite fit our purpose
 - No (well-known?) symbolic model-checker

Action-Restricted CTL (ARCTL)

- Variant of ACTL
- Action conditions α on path quantifiers
 - e.g. $\mathbf{A}_\alpha \mathbf{F} \varphi$ = on all α -paths, sooner or later φ
 - vs. on temporal quantifiers in ACTL
 - e.g. $\mathbf{A} \mathbf{F}_\alpha \varphi$ = on all paths, there is an α -prefix to φ
- α -restricted formula on full model =
unrestricted formula on α -restricted model
- (IS sat CTLK) can be reduced to (LTS sat ARCTL)
 - needs reachability = reverse temporal transitions

Symbolic Model-Checking for Action-Based Logics

- Classical symbolic model-checking for CTL generalizes naturally to ARCTL or ACTL
 - some subtleties due to finite α -paths and fairness

$$eax(A, S) = \{s \mid \exists a, s' \cdot s \xrightarrow{a} s' \wedge a \in A \wedge s' \in S\}$$

$$eau(A, S, S') = \mu Z \cdot S' \cup (S \cap eax(A, Z))$$

$$eag(A, S) = \nu Z \cdot S \cap eax(A, Z)$$

- NuSMV already has “actions” in models
 - called input variables (IVARs)
 - but not allowed in CTL

Action-Based Logics in NuSMV

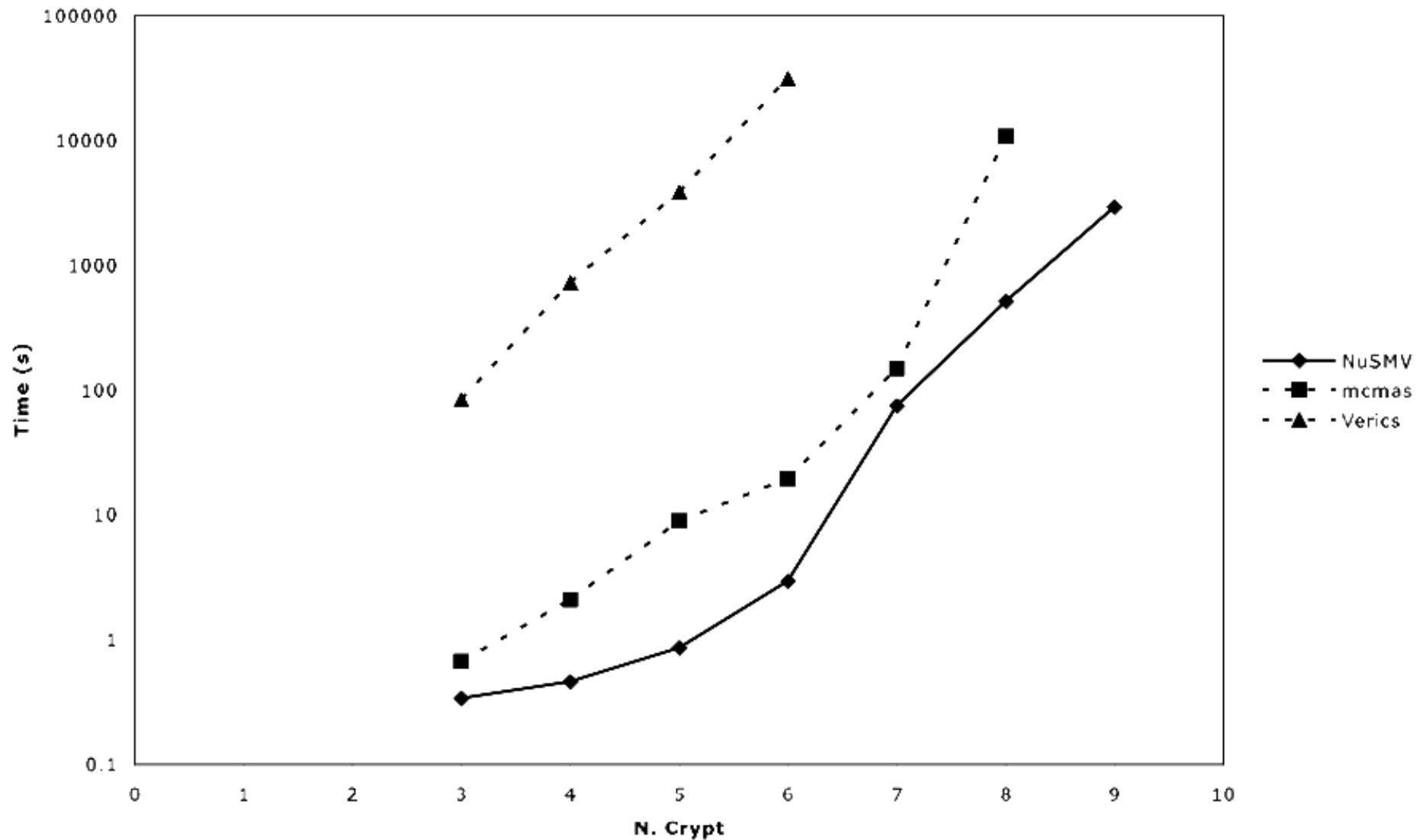
We added ARCTL support to NuSMV

- V1: reduction to KS + CTL,
projecting actions into post-states
e.g. $\mathbf{A}_\alpha \mathbf{X} \varphi$ reduces to $\mathbf{AX} (\alpha \Rightarrow \varphi) \wedge \mathbf{EX} \alpha$
- V2: native ARCTL support, using IVARs
- see [Pecheur-Raimondi 2006]

CTLK in NuSMV

- CTLK and agents (observed variables) handled by a macro package (m4)
- Good performance wrt. dedicated model checkers (CMAS, Verics), see next slide
- see [Raimondi-Pecher-Lomuscio 2005]

CTLK on Dining Cryptographers



Outline

Model-Based Autonomy and Diagnosis

Verification of Model-Based Controllers

Verification of Diagnosability

Symbolic Verification with Knowledge

Symbolic Verification with Actions

Conclusions

Summary: From Mars to Actions

Deep-space missions (incl. **Mars**)

=> Model-based autonomy (incl. diagnosis)

=> **Model-based verification**

=> **Diagnosability**

=> **Epistemic Logics**

=> **Logics with **Actions****

Lessons Learned

- Verification of **model-based controllers**
 - **Needs** advanced verification (because of large state space)
 - **Facilitates** advanced verification (thanks to model)
- Verification of **control software**
 - Control loop, observability/commandability
 - In particular, failure diagnosability and recoverability
 - Leads to epistemic, action logics
- **Model checking**
 - Applicable to these problems
 - symbolic model checking saves the day
- Verification of **software**
 - All other principles still apply: process, testing, ...

Perspectives

- Key ideas:
 - **model-based analysis (model checking)**
 - **partial observability**
- Extensions
 - from discrete to continuous, real-time, **hybrid models**
 - from fault diagnosis to **planning**
 - e.g. test-case generation for planners
see [Raimondi-Pecheur-Brat 2007]
- Connections
 - with classical **risk analysis** (fault trees, FMEA)
 - with **man-machine interface** issues (observability!)
 - with **game theory** (the Controller vs. the Environment)

Thank you!

Publications available at
<http://www.info.ucl.ac.be/~pecheur/publi/>