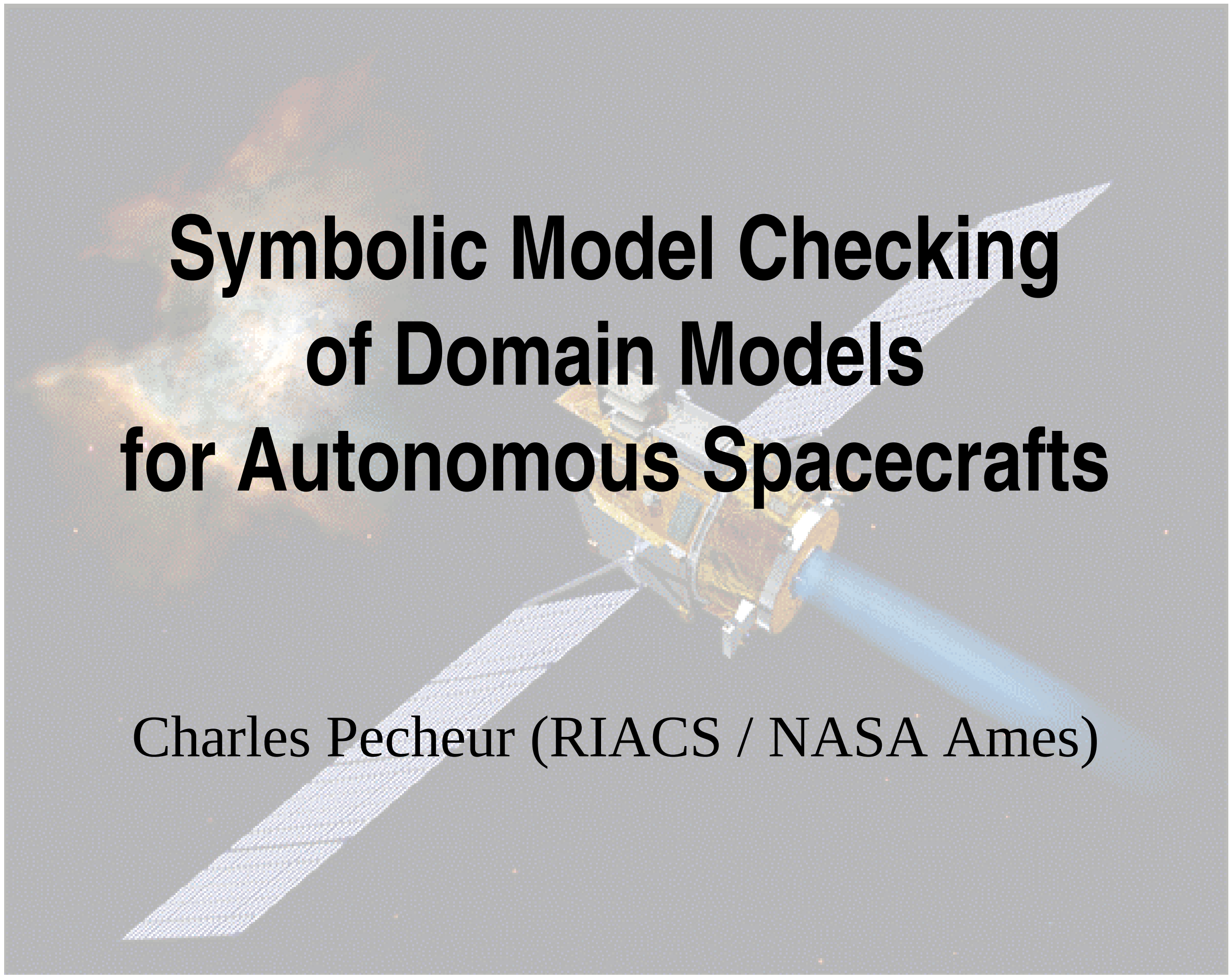


Symbolic Model Checking of Domain Models for Autonomous Spacecrafts

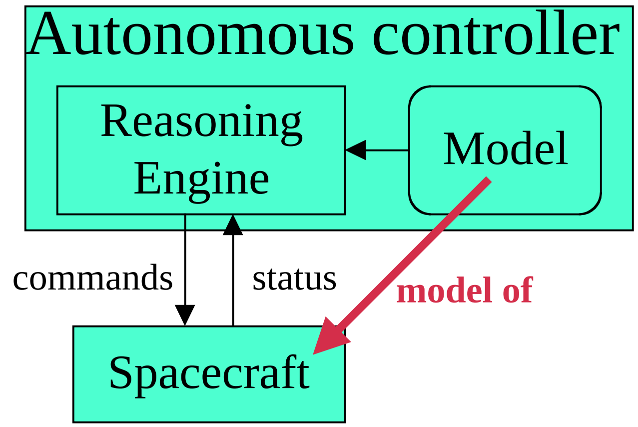
Charles Pecheur (RIACS / NASA Ames)

Model-Based Autonomy

Goal: "intelligent" autonomous spacecrafts

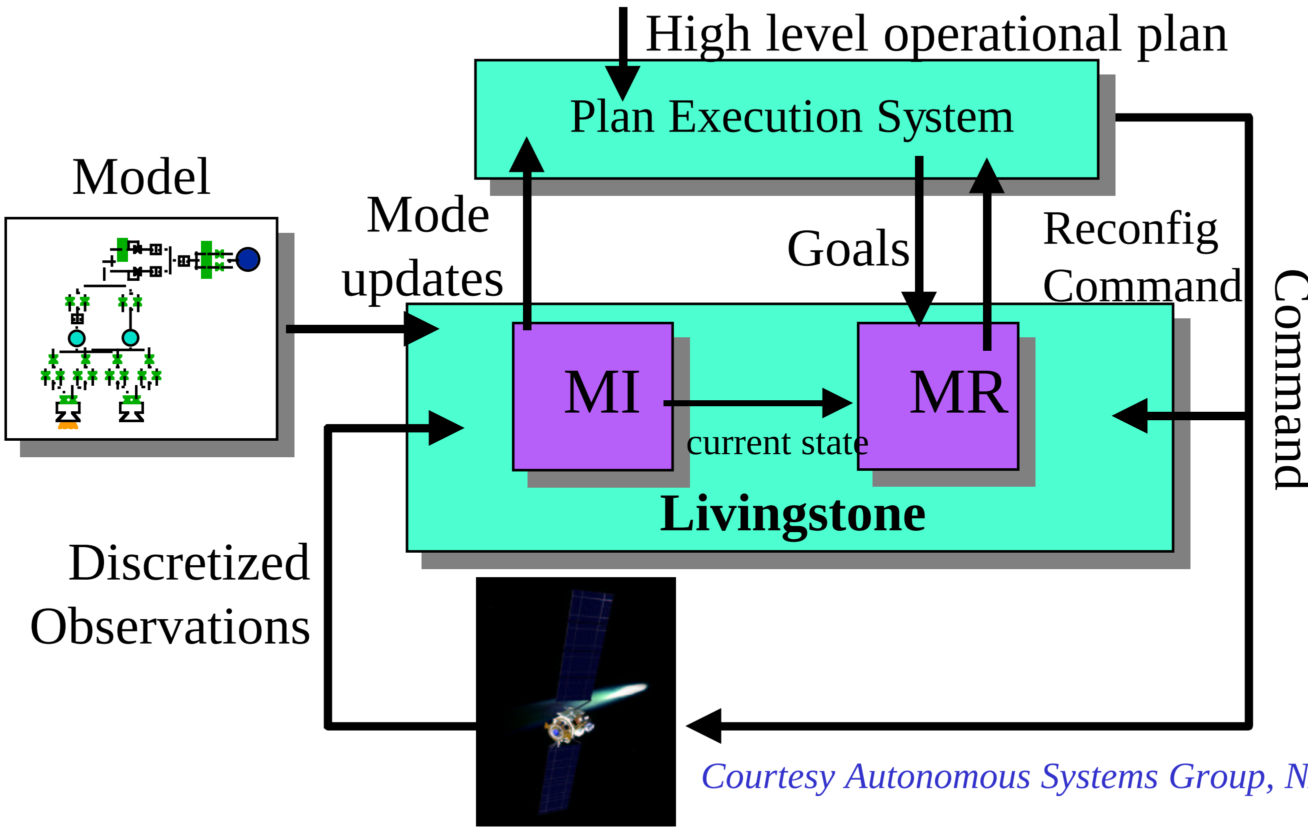
- cheaper (smaller ground control)
- more capable (delays, blackouts)

- General reasoning engine + application-specific model
- Use model to respond to unanticipated situations
- For planning, diagnosis
- Huge state space, reliability is critical



Livingstone

Remote Agent's model-based diagnosis sub-system



Courtesy Autonomous Systems Group, NASA Ames

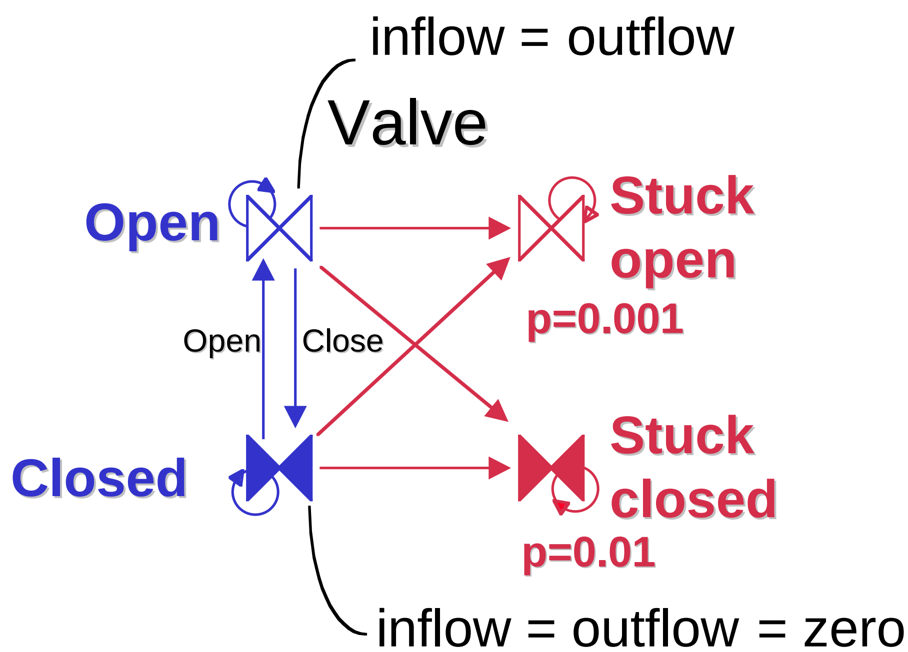
Livingstone Models

inflow, outflow : {zero,low,high}

- concurrent transition systems (components)
- synchronous product
- enumerated types
  => finite state

Essentially ≈ SMV model

\+ nominal/fault modes,
commands/monitors (I/O),
probabilities on faults, ...



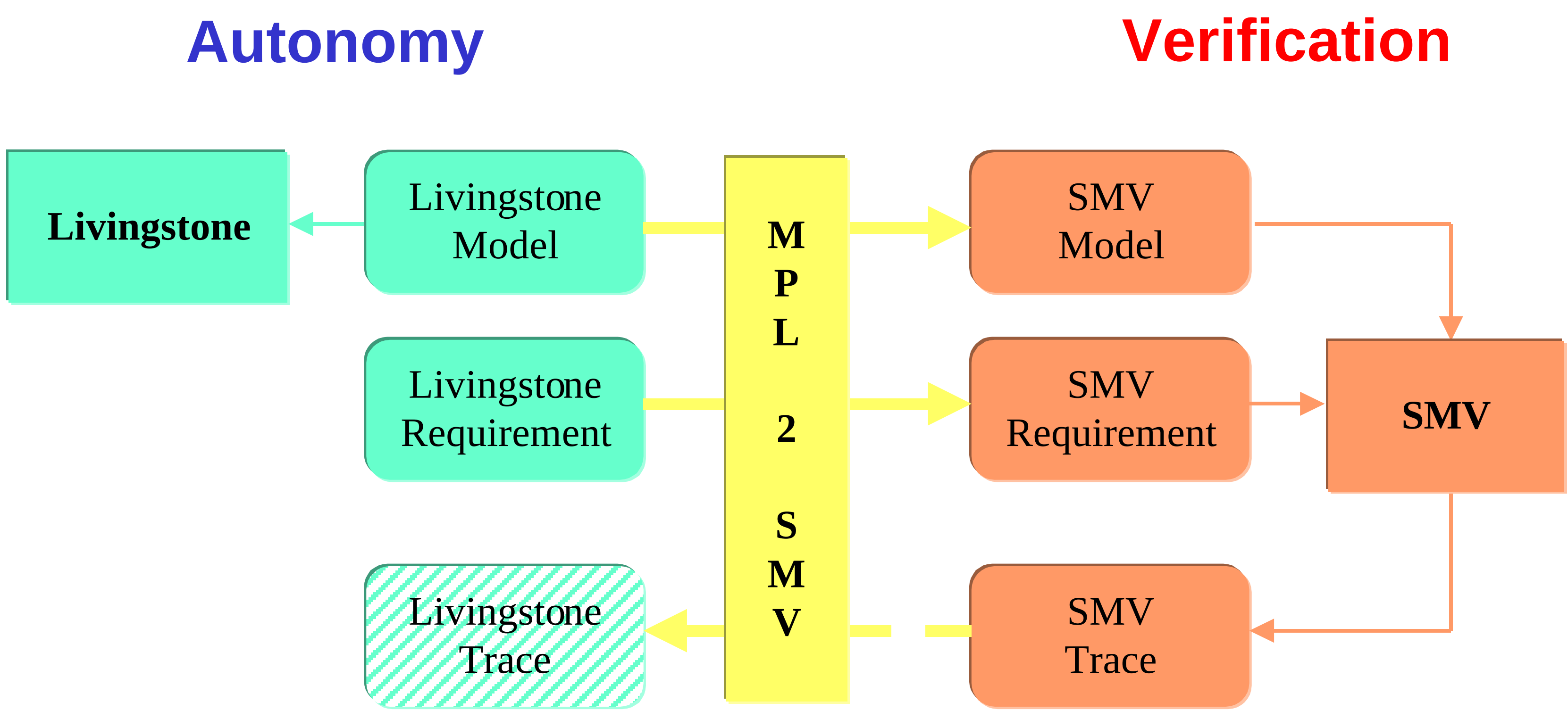
Courtesy Autonomous Systems Group, NASA Ames

Diagnosis = find the most likely assumptions (modes) that are consistent with the observations (commands/monitors)

Large State Space?

- Example: model of ISPP = $7.16 \cdot 10^{55}$ states
- This is only the Livingstone model – a complete verification model could be
  - Exec driver (10-100 states)
  - x Spacecraft simulator (10^{55} states)
  - x Livingstone system (keeps history – $10^{n \cdot 55}$ states)
- Verify a system that analyzes a large state space!
- Approach: the model is the program
  - – Verify it (using symbolic model checking)
  - – Assume Livingstone correct (and complete)

MPL2SMV


Autonomy
Verification
Livingstone
Livingstone Model
Livingstone Requirement
Livingstone Trace
M P L 2 S M V
SMV Model
SMV Requirement
SMV Trace
SMV

Translator from Livingstone to SMV

- Co-developed with CMU (Reid Simmons)
- Similar semantics => translation is easy
- Properties in temporal logic + pre-defined patterns
- Initially for Livingstone 1 (Lisp),
  upgraded to Livingstone 2 (C++/Java)

Principle of Operations

Lisp shell

```
(load "mpl2smv.lisp")
;; load the translator
;; Livingstone not needed!

(translate "ispp.lisp" "ispp.smv")
;; do the translation

(smv "ispp.smv")
;; call SMV
;; (as a sub-process)
```

ispp.lisp

```
(defcomponent  heater …)
(defmodule valve-mod …)
…
(defverify
  :structure (ispp)
  :specification (all (globally …)))
```

ispp.smv

```
MODULE Mheater …
MODULE Mvalve-mod …
…
MODULE main
VAR Xispp:Mispp
SPEC AG …
```

SMV output

```
Specification AG … is false as shown …
State 1.1: …
State 1.2: …
```

Simple Properties

- Supported by the translator:
  - syntax sugar
  - iterate over model elements (e.g. all component modes)
- Examples
  - Reachability (no dead code)

    EF heater.mode = on
  - Path Reachability (scenario)

    AG (s1 –> EF (s2 & EF (s3 & EF s4)))

Probabilistic Properties

- Use probabilities associated to failure transitions
- Use order of magnitude: -log(p), rounded to a small integer
- Combine additively, OK for BDD computations
- Approximate – but so are the proba. values

```
heater.mode = overheat -> heater.proba = 2;      (p = 0.01)
proba = heater.proba + valve.proba + sensor.proba;
SPEC AG (broken  &  proba < 3  –>  EF working)
```

Functional Dependency

- Check that y=f(x) for some unknown f
- Use universally quantified variables in CTL
  = undetermined constants in SMV

```
VAR x0,y0 : {a,b,c};
TRANS next(x0) = x0
TRANS next(y0) = y0
SPEC (EF x=x0 & y=y0) –> (AG x=x0 –> y=y0)
```

(VAR and TRANS lines ≈ ∀ x0, y0)

- Limitation: counter-example needs two traces, SMV gives only one
  => instantiate second half by hand, re-run SMV

Temporal Queries

- Temporal Query = CTL formula with a hole:

  AG (**?** –> EF working)
- Search (canonical) condition for ? that satisfies the formula (computable for useful classes of queries)
- Recent research, interrupted (William Chan, †1999)
- Problem: visualize solutions (CNF, projections, ...)
- Core algorithm implemented in NuSMV (Wolfgang Heinle)
- Deceptive initial results, to probe further

SMV with Macro Expansion

- Custom version of SMV (Bwolen Yang, CAV 99)
- Eliminates variables by Macro Expansion:
  - analyzes static constraints of the model (invariants),
  - find dependent variables x=f(x1,...,xn),
  - substitute f(x1,...,xn) for x everywhere,
  - eliminate x from the set of BDD variables.
- For models with lots of invariants
  => useful for Livingstone models
- Full ISPP model in < 1 min, vs. SMV runs out of memory.

ISPP Model Statistics

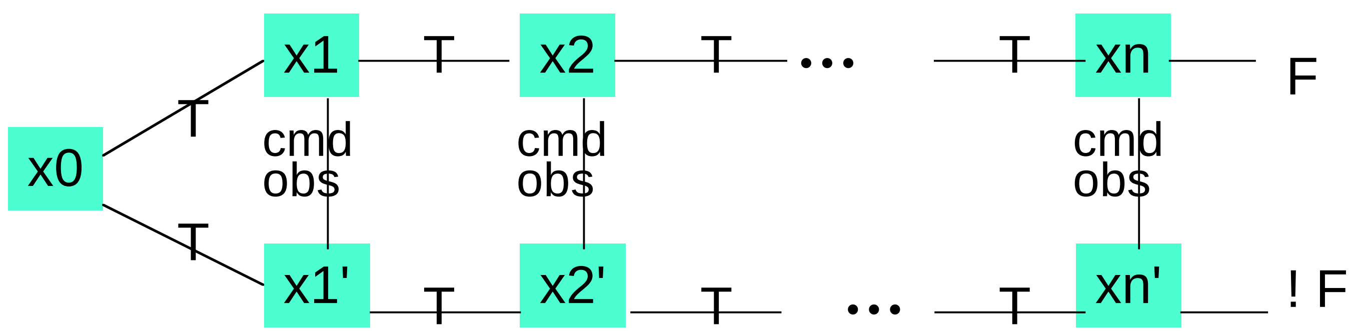
- In Situ Propellant Production (ISPP)
  = turn Mars atmosphere into rocket fuel (NASA KSC)
- Original model state = 530 bits (trans. = 1060 bits)
- Total BDD vars 588 bits
  Macro expanded -209 bits
  Reduced BDD vars 379 bits
- Reachable state space $7.16 \cdot 10^{55} = 2^{185.5}$
  Total state space $1.06 \cdot 10^{81} = 2^{269.16}$
- Reachability of all modes (163):
  29.14" CPU time in 63.6 Mb RAM

Diagnosis Properties

- Can fault F always be diagnosed?
  (assuming perfect diagnosis and accurate model)
  = is F unambiguously observable?
  ∀ obs0 . (EF F & obs=obs0) –> (AG F –> obs=obs0)
- Similar to functional dependency
- obs = observable variables (many of them)
- Static variant (ignore transitions):
  SAT on two states S, S' such that
  F & ! F' & obs=obs'

Diagnosis Properties Revisited

- Very recent (yesterday), with Alessandro Cimatti
- Can fault F be diagnosed knowing the last n steps?
- Apply SAT to:



- Variants are possible (e.g. fork at n-1 intead of 0)

Diagnosis Properties (cont'd)

- Does it work?
  - Computational cost of extra variables
- Has it been done?
  - Similar work in hardware testability?
- Is it useful?
  - It is unrealistic to expect all faults to be immediately observable (e.g. valve closed vs. stuck-closed)
  - What weaker properties? Are they verifiable?
- To be explored

Summary

- Verification of model-based diagnosis:
  - Space flight => safety critical.
  - Huge state space (w.r.t. fixed command sequence).
- Focus on models (the model is the program)
- Quite different from executable programs
  - Loose coupling, no threads of control, passive.
  - Huge but shallow state spaces.
- <u>Symbolic</u> model checking is very appropriate
- Verify well-formedness + validity w.r.t. hardware
- Verify suitability for diagnosis: to be explored

Thank You