Symbolic Model Checking of Domain Models for Autonomous Spacecrafts

Charles Pecheur (RIACS / NASA Ames)
**Introduction**

**Past:**
Time-stamped control sequences

**Future:**
On-board intelligence

+ Can respond to unanticipated scenarios!

– How do we verify all those scenarios?

*Remote Agent*
In flight on DS-1, May 99
Outline

• Model-Based Autonomy and Livingstone

• Symbolic Model Checking and SMV

• Verification of Livingstone Models
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
Remote Agent's model-based diagnosis sub-system

Livingstone

Plan Execution System

High level operational plan

Goals

Model

Mode updates

Livingstone

Reconfig Command

Command

MI

MR

current state

Discretized Observations

Courtesy Autonomous Systems Group, NASA Ames
Outline

• Model-Based Autonomy and Livingstone

• Symbolic Model Checking and SMV

• Verification of Livingstone Models
Model ...
Model Checking

Modeling Abstraction

Verification

“Valve is closed when Tank is empty”

AG (tank=empty => valve=closed)
Symbolic Model Checking

Instead of considering each individual state, Symbolic model checking...
Symbolic Model Checking

Instead of considering each individual state, Symbolic model checking...

- Manipulates sets of states,
Symbolic Model Checking

Instead of considering each individual state, Symbolic model checking...

- Manipulates sets of states,
- Represented as boolean formulas,
Symbolic Model Checking

Instead of considering each individual state, Symbolic model checking...

- Manipulates sets of states,
- Represented as boolean formulas,
- Encoded as binary decision diagrams.
Symbolic Model Checking

Instead of considering each individual state, Symbolic model checking...

• Manipulates sets of states,
  – Can handle very large state spaces ($10^{50}+$)
• Represented as boolean formulas,
  – Suited for boolean/abstract models

• Encoded as binary decision diagrams.
  – The limit is BDD size (hard to control)
Boolean Functions

• Represent a state as boolean variables
  \[ s = b_1, \ldots, b_n \]
  Non-boolean variables => use boolean encoding

• A set of states as a boolean function
  \[ s \text{ in } S \iff f(b_1, \ldots, b_n) = 1 \]

• A transition relation as a boolean function over two states
  \[ s \rightarrow s' \iff f(b_1, \ldots, b_n, b'_1, \ldots, b'_n) = 1 \]
Binary Decision Trees

• Encoding for boolean functions

• Notational convention:
  \[ = \text{if } c \text{ then } e \text{ else } e' \]
  \[ = (c ? e : e') \]

• Always exists but not unique

\[ (a \mid b) \Rightarrow c \]
From Trees to Diagrams

- Fixed variable ordering
  "layered" tree

\[(a | b) \Rightarrow c\]
From Trees to Diagrams

- Fixed variable ordering
  "layered" tree
- Merge equal subtrees

\[(a \mid b) \Rightarrow c\]
From Trees to Diagrams

- Fixed variable ordering
  "layered" tree
- Merge equal subtrees
- Remove nodes with equal subtrees

\[(a \mid b) \Rightarrow c\]

\(\Rightarrow\) Ordered Binary Decision Diagram
[Ordered] Binary Decision Diagrams

- [O]BDDS for short
- Unique normal form
  - for a given ordering and
  - up to isomorphism
  $\Rightarrow$ compare in constant time
  (using hash table)

\[(a \mid b) \Rightarrow c\]
Computations with BDDs

• All needed operations can be efficiently computed using BDDS

• Example: boolean combinator \( f \& g \):
  \[(b \oplus f' : f'') \& (b \oplus g' : g'') = (b \oplus f' \& g' : f'' \& g'')\]
cache results → O(|f|.|g|) time

• Other operations:
  – Negation \(!f\)
  – Instantiation \(f[b=1], f[b=0]\)
  – Quantifiers exists \(b . f\), forall \(b . f\)
Transition Systems with BDDs

Given boolean state variables \( v = b_1, \ldots, b_n \)

a set of states as a BDD \( p(v) \)

a transition relation as a BDD \( T(v, v') \)

we can compute the predecessors and successors of \( p \) as BDDs:

\[
(p_{\text{pred}} p)(v) = \exists v'. T(v, v') \land p(v')
\]

\[
(p_{\text{succ}} p)(v) = \exists v'. p(v') \land T(v', v)
\]
Computation Tree Logic (branching):
Consider the tree of possible executions

Always ...

$\neg\neg p$

$\Delta$

$AX p$

Sometimes ...

$\neg\neg p$

$\nabla$

$EX p$

... Next $p$

... Globally $p$

$AG p$

In all states

$duals$

... Finally $p$

$AF p$

In some state

$EF p$

$E[p U q]$

... $p$ Until $q$

$A[p U q]$
Evaluating CTL with BDDS

Example: compute $\text{EF } p$ from $p$ with BDDs:

$$\text{EF } p = \text{lfp } U . (p | \text{EX } U)$$

$= \text{least solution of } U = p | \text{EX } U$

$U_0 = 0$

$U_1 = p | \text{EX } U_0 = p$

$\ldots$

$U_{n+1} = p | \text{EX } U_n = p | \text{EX } p | \ldots | \text{EX}^n p$

until $U_n = U_{n+1} = \text{EF } p$

– Convergence assured because finite domain

– Backward search from $p$ to $\text{EF } p$
Variable Ordering

• Must be the same for all BDDs
• Size of BDDs depends critically on ordering
• Worst case: exponential w.r.t. #variables
  – sometimes exponential for any ordering
    e.g. middle output bit of n-bit multiplier
• Finding optimum is hard (NP-complete)
  => optimization uses heuristics
SMV

• **SMV** = *Symbolic Model Verifier.*
• Modeling language based on parallel assignments.
• Specifications in temporal logic **CTL**.
• **BDD-based symbolic model checking.**
• Several versions:
  - (CMU) SMV: original work by McMillan (Carnegie Mellon)
  - NuSMV: clean re-writing, faster (ITC-IRST and CMU)
  - Cadence SMV: following McMillan (Cadence Berkeley Labs)
What SMV Does

Model

```plaintext
MODULE user(...) ...
MODULE main
VAR turn: {1, 2};
user1: user(...);
...
```

Specification

```plaintext
SPEC AG !(user1.state = c) & (user2.state = c)
```

Counter-example

```plaintext
-- specification AG ... is false
-- as demonstrated by ...
state 1.1:
  turn = 1
  user1.state = n
  user2.state = n
state 1.2:
...
resources used: ...
```
MODULE user(turn, id, other)
VAR state: {n, t, c};
DEFINE my_turn :=
  (other=n) | ((other=t) & (turn=id));
ASSIGN
init(state) := n;
next(state) := case
  (state = n) : {n, t};
  (state = t) & my_turn: c;
  (state = c) : n;
  1 : state;
esac;

SPEC AG((state = t) -> AF (state = c))
MODULE main
VAR turn: {1, 2};
    user1: user(turn, 1, user2.state);
    user2: user(turn, 2, user1.state);
ASSIGN
init(turn) := 1;
next(turn) := case
    (user1.state=n) & (user2.state=t): 2;
    (user2.state=n) & (user1.state=t): 1;
    1: turn;
esac;

SPEC AG !(user1.state=c) & (user2.state=c)
SPEC AG !(user1.state=c)    -- false!
Diagnostic Trace Example

-- specification AG (state = t -> AF state = c) (in module user1) is true
-- specification AG (state = t -> AF state = c) (in module user2) is true
-- specification AG (!(user1.state = c & user2.state = c))... is true
-- specification AG (!user1.state = c) is false
-- as demonstrated by the following execution sequence
state 1.1:
  turn = 1
  user1.state = n
  user2.state = n

state 1.2:
  user1.state = t

state 1.3:
  user1.state = c
The SMV program defines:
- a finite transition model $M$ (Kripke structure),
- a set of possible initial states $I$ (may be several),
- specifications $P_1 \ldots P_m$ (CTL formulas).

SMV checks that each specification $P$ is satisfied in all initial states $s_o$ of model $M$.

$$\forall s_o \in I . \ M, s_o \models P$$
Outline

• Model-Based Autonomy and Livingstone

• Symbolic Model Checking and SMV

• Verification of Livingstone Models
Livingstone Models

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

Essentially ≈ SMV model

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

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

Courtesy Autonomous Systems Group, NASA Ames
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)
  - Spacecraft simulator ($10^{55}$ states)
  - 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

Livingstone
Livingstone Model
Livingstone Requirement
Livingstone Trace

Verification

SMV Model
SMV Requirement
SMV Trace
SMV

MPL2SMV

MLP
2
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)
```

### SMV output

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

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

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; \quad (p = 0.01)
proba = heater.proba + valve.proba + sensor.proba;
SPEC AG (broken & proba < 3 \rightarrow \text{EF working})
Functional Dependency

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

\[
\text{VAR } x_0, y_0 : \{a, b, c\}; \\
\text{TRANS } \text{next}(x_0) = x_0 \\
\text{TRANS } \text{next}(y_0) = y_0 \\
\text{SPEC } (\text{EF } x=x_0 \& y=y_0) \rightarrow (\text{AG } x=x_0 \rightarrow y=y_0)
\]

- Limitation: counter-example needs two traces, SMV gives only one
  \( \Rightarrow \) instantiate second half by hand, re-run SMV
Temporal Queries

• Temporal Query = CTL formula with a hole:
  \[ AG (\ ? \rightarrow EF \text{ working}) \]

• 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 $x=f(x_1,...,x_n)$,
  - substitute $f(x_1,...,x_n)$ 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)
  $= \text{is } F \text{ unambiguously observable?}$
  $\forall \text{obs0} . (EF F \& \text{obs}=\text{obs0}) \rightarrow (AG F \rightarrow \text{obs}=\text{obs0})$

• Similar to functional dependency

• $\text{obs} = \text{observable variables (many of them)}$

• Static variant (ignore transitions):
  SAT on two states $S, S'$ such that
  $F \& \neg F' \& \text{obs} = \text{obs'}$
• 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 instead 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.
• Symbolic model checking is very appropriate
• Verify well-formedness + validity w.r.t. hardware
• Verify suitability for diagnosis: to be explored
Thank You
Symbolic Model Checking

References


*The seminal paper on Binary Decision Diagrams.*


*Survey paper on the principles of symbolic model checking.*


*Paper on SAT-based bounded model checking.*
Symbolic Model Checking

References (cont'd)


*Symbolic model checking of CTL with fairness.*


*Verifying LTL using symbolic model checking.*
SMV

References


*Based on Ken McMillan's PhD thesis on SMV.*


http://www.cs.cmu.edu/~modelcheck/smv/smvmanual.r2.2.ps

*The (old) user manual provided with the SMV program.*


*Survey paper on NuSMV.*