Symbolic Verification of System-Level Specifications for Aerospace Applications

roveri@fbk.eu
Fondazione Bruno Kessler
Istituto per la Ricerca Scientifica e Tecnologica
Motivations
Symbolic encoding of Slim models
Satisfiability Modulo Theory
Bounded Model Checking via SMT
Counterexample Guided Abstraction Refinement
The NuSMV tool
The COMPASS toolset
Conclusions
Motivations

◆ COMPASS analyses for Slim models
  – Functional correctness
  – Safety analysis
  – Diagnosability
  – Performability

◆ These analyses rely on model checking techniques
  – Semantics of Slim given as Network of Event-Data Automata (NEDA).
  – Model checkers operates on labeled transition systems
  – Slim specifications can be large
    » State space explosion
    » Real and continuous variables
  – Need of techniques to tackled with these problems
    » Symbolic encoding of labeled transition system
    » Advanced symbolic verification techniques for dealing with real and continuous variables
Outline

- Motivations
- Symbolic encoding of Slim models
- Satisfiability Modulo Theory
- Bounded Model Checking via SMT
- Counterexample Guided Abstraction Refinement
- The NuSMV tool
- The COMPASS toolset
- Conclusions
Symbolic encoding of Slim models

- Slim semantics is given as a NEDA: \( \mathcal{N} = \langle U^i, \alpha, EC, DC \rangle \) \( i \in [n] \)
  - \( U^i = \langle M^i, m_0^i, X^i, v_0^i, i^i, E^i, \rightarrow^i \rangle \)
    - \( M^i \) = finite set of modes
    - \( m_0^i \) = initial mode
    - \( X^i \) = set of input/output/local variables
    - \( v_0^i \) = initial valuation for variables
    - \( i^i \) = mode invariants
    - \( E^i \) = input/output events
    - \( \rightarrow^i \) = transition relation

- Model checkers operates on Labeled Transition Systems
  - \( L = \langle V, \mathfrak{T}, I, R \rangle \)
    - \( V \) = finite set of variables
    - \( \mathfrak{T} \) = finite set of transition labels
    - \( I \) = initial condition
    - \( R \) = transition relation
Symbolic Representation of EDA

- \( V = X \cup loc \cup \partial \)
  - \( loc \) is a variable representing the modes (domain of \( loc \) is \( M \))
  - \( \partial \) is a real variable representing time elapse
- \( \mathcal{I} = E \cup \{\tau\} \)
- \( V' = X' \cup loc' \cup \partial' \)
- \( \text{inv}(m) = \{\Sigma_i c_i x \propto d\} \)
  - \((\Sigma_i c_i x \propto d) \in i(m)\) and \(\in \{=,<,\geq,\leq,\neq\}\)
- \( \text{flow}(m) = \{\Sigma_i c_i (x' - x) \propto d\partial\} \)
  - \((\Sigma_i c_i x' \propto d\partial) \in i(m)\) and \(\in \{=,<,\geq,\leq,\neq\}\)
- \( I = (loc = m_0 \land \text{inv}(m_0) \land v=v_0) \)
- \( R = \bigvee_j R_{e_j} \lor \bigvee_j R_{\tau_j} \)
  - \( R_{e_j} = (loc = m_s \land loc' = m_d \land e \land \text{inv}(m_d) \land v' = \rho(v)) \)
    » For \(< m_s, e, v' = \rho(v), m_d > \in \rightarrow \)
  - \( R_{\tau_j} = (loc = m_s \land loc' = m_s \land \partial > 0 \land \text{inv}(m_d) \land \text{flow}(m_s) \land v' = \rho(v)) \)
    » For \(< m_s, \tau, v' = \rho(v), m_s > \in \rightarrow \)
device Battery
  features
  empty: out event port;
  voltage: out data port real;
end Battery;

device implementation Battery.Imp
  subcomponents
  energy: data continuous initially 100.0;
  modes
  charged: initial mode
    while energy' = -0.02 and
    energy >= 20.0;
  depleted: mode
    while energy' = -0.03;
  transitions
  charged -[ then voltage := energy/50.0 + 4.0 ]-> charged;
  charged -[empty when energy<20 ]-> depleted;
  depleted -[ then voltage := energy/50.0 + 4.0 ]-> depleted;
end Battery.Imp;
Symbolic encoding: example

\[ V = \]

\[ S = \{empty, \tau\} \]

\[ inv(m) = \]

\[ flow(m) = \]

**device** Battery

**features**

empty: out event port;

voltage: out data port real;

**end** Battery;

**device** implementation Battery.Imp

**subcomponents**

energy: data continuous initially 100.0;

**modes**

charged: initial mode

while energy' = -0.02 and 

energy >= 20.0;

depleted: mode

while energy' = -0.03;

<table>
<thead>
<tr>
<th>energy</th>
<th>real</th>
</tr>
</thead>
<tbody>
<tr>
<td>voltage</td>
<td>real</td>
</tr>
<tr>
<td>loc</td>
<td>{charged, depleted}</td>
</tr>
<tr>
<td>(\partial)</td>
<td>real</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>charged</th>
<th>energy (\geq 20.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>depleted</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>charged</th>
<th>energy' - energy = -0.02(\partial)</th>
</tr>
</thead>
<tbody>
<tr>
<td>depleted</td>
<td>energy' - energy = -0.03(\partial)</td>
</tr>
</tbody>
</table>
Symbolic encoding: initial condition

- **subcomponents**
  - energy: data continuous initially 100.0;

...  

- **modes**
  - charged: initial mode
    - while energy' = -0.02 and energy >= 20.0;

  \[ \text{loc} = \text{charged} \]
  - energy $\geq 20.0 \land$
    - energy = 100.0
Symbolic encoding: transitions

charged: initial mode
while energy' = -0.02 and energy >= 20.0;

....
charged -[then voltage := energy/50.0 + 4.0] -> charged

\[ loc = \text{charged} \land loc' = \text{charged} \land \]
\[ \partial > 0 \land \]
\[ \text{energy}' \geq 20.0 \land \]
\[ \text{energy}' - \text{energy} = -0.02\partial \land \]
\[ \text{voltage}' = \text{energy}/50.0 + 4.0 \]
Symbolic encoding: transitions (II)

charged - [empty when energy < 20] -> depleted;

\[ \text{loc} = \text{charged} \land \text{loc}' = \text{depleted} \land \]
\[ \text{energy} < 20 \land \text{empty} \land \text{true} \land \]
\[ \text{energy}' = \text{energy} \land \text{voltage}' = \text{voltage} \]
Symbolic encoding: transitions (III)

depleted: **mode**

    **while** energy' = -0.03;

    ...

    depleted -> [ **then** voltage := energy/50.0 + 4.0 ] -> depleted

    \[ \downarrow \]

    \[ \begin{align*}
        \text{loc} &= \text{depleted} \land \text{loc}' = \text{depleted} \land \\
        \partial &> 0 \land \\
        \text{true} \land \\
        \text{energy}' - \text{energy} &= -0.03\partial \land \\
        \text{voltage}' &= \text{energy}/50.0 + 4.0
    \end{align*} \]
Symbolic encoding of NEDA

- Symbolic encoding for EDA generalizes to NEDA
  - $V = \bigcup_i X^i \cup \bigcup_i \text{loc}^i \cup \varnothing \cup \bigcup_i \text{active}^i$
  - $\text{active}^i$ being a Boolean variable true if component $i$ is active
  - $\mathcal{S} = \bigcup_i E^i \cup \{\tau\}$
  - $\alpha$ encoded symbolically with a formula $A(\text{loc,active})$
  - EC encoded symbolically with $EC(\text{loc,E})$
  - DC encoded symbolically with $DC(\text{loc,X})$
  - Initialization determined by active EDAs
  - Transition relation determined by active EDAs
    - Perform local transitions
      - Timed local transitions in all EDAs
      - Internal transition in EDA
      - Multiway event communications from EDA to connected EDAs
    - Initialize (re-)activated components
    - Establish consistency w.r.t. DC
Motivations
Symbolic encoding of Slim models
Satisfiability Modulo Theory
Bounded Model Checking via SMT
Counterexample Guided Abstraction Refinement
The NuSMV tool
The COMPASS toolset
Conclusions
Beyond the Boolean case

- Verification engines used in Model Checking are very powerful
  - Symbolic model checking techniques
    » Binary Decision Diagrams
    » Propositional SAT solvers

- They work at the Boolean level

- Reasoning at the Boolean level is a limitation
  - Boolean representation not expressive enough
    » encoding may not exist (e.g. reals), or can "blow up“ (bitvectors)
  - Boolean reasoning not the “right” level of abstraction
    » important information may be lost during encoding
Examples

◆ **RTL circuits**
  – word $w[n]$ reduced to $w_1 \ldots w_n$ Boolean variables
  – booleanization destroys data path structure!

◆ **Pipelines**
  – function symbols used to abstract blocks

◆ **Timed automata**
  – real-valued variables for timing
  – difference constraints to express time elapse

◆ **Hybrid automata (e.g. Slim models)**
  – real-valued variables for physical dynamics
  – mathematical constraints to express continuous evolution

◆ **Software verification**
  – integer-valued variables for proof obligations
Satisfiability Modulo Theory

- **Trade off between expressiveness and reasoning**
  - SAT solvers
    » Boolean reasoning, completely automatic, very efficient
  - Theorem provers
    » General FOL, limited automation

- **SMT aims at**
  - Retain efficiency of Boolean reasoning
  - Increase expressiveness
  - Use decidable fragments of FOL

- **Expected impact in formal verification**
  - Increase capacity by working above the Boolean level
Satisfiability Modulo Theory

- Is an extension of Boolean SAT

- Some atoms have non Boolean (theory) content
  - $A_1 = x - y \leq 3$
  - $A_2 = y - z = 10$
  - $A_3 = x - z \geq 15$

- Constants, individual variables, functions and predicates are interpreted over a theory
  - If $x = 0$, $y = 20$, $z = 10$
  - Then $A_1 = T$, $A_2 = T$, $A_3 = F$

- Interpretation of atoms are constrained
  - $A_1$, $A_2$, $A_3$ cannot be all true at the same time
FOL Theories of Practical Interest

- **Equality Uninterpreted Functions (EUF)**
  - \( x = f(y), \ h(x) = g(y) \)

- **Difference constraints (DL)**
  - \( x - y \leq 3 \)

- **Linear Arithmetic**
  - \( 3x - 5y + 7z \leq 1 \)
  - reals (LRA), integers (LIA)

- **Arrays (Ar)**
  - \( \text{read(write}(A, i, v), j) \)

- **Bit Vectors (BV)**
  - \( A[4:8] \& 0b4_1001 \)

- **Their combination**
The SMT problem

- **Given one theory** $T$ (e.g. LRA, ...)
  - Terms $t$, $t_1$, $t_2$, ...
    - Constants $c$, $c_1$, $c_2$, ...
    - Variables $x$, $x_1$, $x_2$, ..., $x_n$, $y$, ...
    - Function application $f(c, x_1)$, $g(f(x, y))$, ...
  - Theory atoms
    - Predicate applications $P(t_1, t_2)$, $Q(t_1)$, ...

- **Atoms are either**
  - Boolean atoms $A$, $A_1$, $A_2$, ..., or
  - Theory atoms

- **Formulas are Boolean combination of atoms**
  - $\neg \phi_1$, $\phi_1 \lor \phi_2$, $\phi_1 \land \phi_2$, $\phi_1 \rightarrow \phi_2$, $\phi_1 \leftrightarrow \phi_2$

- **Is the theory formula** $\phi$ **satisfiable?**
The search combines Boolean reasoning (DPLL) and theory reasoning

Find Boolean model
- Theory atoms treated as Boolean atoms
- Truth values to Boolean and theory atoms
- Model propositionally satisfies the formula

Check consistency w.r.t. theory
- Set of constraints induced by truth values to theory atoms
- Existence of values to theory variables
Boolean DPLL search space

- The DPLL procedure
- Incremental construction of satisfying assignment
- Backtrack/backjump on conflict
- Learn reason for conflict
- Splitting heuristics
SMT DPLL search space

\[ x, y, z : \text{reals} \]
\[ Q: \text{Booleans} \]

<table>
<thead>
<tr>
<th>P</th>
<th>x – y ≤ 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>x – z ≤ 4</td>
</tr>
<tr>
<td>S</td>
<td>y – z ≥ 2</td>
</tr>
</tbody>
</table>

Many Boolean models are not theory consistent!
Optimizations

- **Early pruning**
  - Check theory consistency of partial assignments

- **Learning theory conflicts**
  - The theory solver can detect a reason for inconsistency
  - I.e. a subset of the literals that are mutually unsatisfiable
    - E.g. $x = y$, $y = z$, $x \neq z$
  - Learn a conflict clause
    - $x \neq y$ or $y \neq z$ or $x = z$
  - By BCP the Boolean enumeration will never make same mistake again

- **Theory deduction**
  - The theory solver can detect that certain atoms have forced values
    - E.g. from $x = y$ and $x = z$ infer that $y = z$ should be true
  - Force deterministic assignments
  - Theory version of BCP
  - Furthermore, the solver can learn the deduction:
    - $x = y \& x = z \Rightarrow y = z$
Optimizations

- **Incrementality/backtrackability**
  - Add constraints to the theory solver without restarting from scratch
  - Remove constraints without paying too much

- **Limiting cost of early pruning**
  - Filtering, incomplete calls

- **Static learning**
  - Pre-compile obvious theory reasoning to Boolean
State variables of various types
  - in addition to discrete
  - reals, integers, bitvectors, arrays, …

Representation
  - higher level
  - structural information is retained
Outline

- Motivations
- Symbolic encoding of Slim models
- Satisfiability Modulo Theory
- Bounded Model Checking via SMT
- Counterexample Guided Abstraction Refinement
- The NuSMV tool
- The COMPASS toolset
- Conclusions
SMT based model checking

- From SAT based to SMT based algorithms
- Simply replace the SAT solver with an SMT solver
  - Bounded model checking
  - K-induction
BMC and Induction

◆ Look for bugs of increasing length
  – $I(X^0) \land R(X^0, X^1) \land \ldots \land R(X^{k-1}, X^k) \land B(X^k)$
  – bug if satisfiable
  – increase $k$ until …

◆ Prove absence of bugs by induction
  – $I(X^0) \land \neg B(X^0)$
  – $I(X^0) \land R(X^0, X^1) \land \neg B(X^1)$
  – …
  – $\neg B(X^0) \land R(X^0, X^1) \land \ldots \land \neg B(X^{k-1}) \land R(X^{k-1}, X^k) \land B(X^k)$
  – proved correct if unsatisfiable (and no bugs until $k$)

◆ Important features of (SMT) solver
  – incremental interface
  – theory lemmas should be retained and can be shifted over time
    » from $\Phi(X^0, X^1)$ to $\Phi(X^i, X^i)$
  – Unsatisfiable core and generation of interpolants
Outline

- Motivations
- Symbolic encoding of Slim models
- Satisfiability Modulo Theory
- Bounded Model Checking via SMT
- Counterexample Guided Abstraction Refinement
- The NuSMV tool
- The COMPASS toolset
- Conclusions
Counterexample Guided Abstraction Refinement

◆ Model checking validates and debugs systems by exploration of their state spaces

◆ PROBLEM: state-space explosion
  – Hardware and protocols: very large number of states
  – Software: typically infinite-state

◆ SOLUTION: analyze a finite-state abstraction of the system

◆ ABSTRACTION:
  – INPUT: a concrete LTS C (initial states + transition relation) and a an abstraction function
  – OUTPUT: finite-state conservative abstraction A
    » If a property holds in A, a concrete version holds in C
    » If a property does not holds in A the counterexample need to be analyzed
    » If the counterexample is not spurious, than the property does not hold in C
    » If the counterexample is spurious the abstraction function has to be refined
CounterExample Guided Abstraction Refinement

Abstraction

Model Checking

Refinement

CounterExample Analysis

G (voltage \geq 10)

Conc. Model

Abstr. Model

\[ G(p) \]

\[ \text{CounterExample Analysis} \]

\[ p, q \]
Predicate abstraction

◆ **Given a concrete LTS over variables** \( X \)

◆ **Given a set of predicates** \( \Psi_i(X) \) **associated to abstract variable** \( P_i \)

\[
P_i \leftrightarrow \Psi_i(X)
\]

◆ **Obtain the corresponding abstract program**

» **AI(P)** is defined by

\[
\exists X. ( \text{CI}(X) \land \bigwedge_i P_i \leftrightarrow \Psi_i(X) )
\]

» **AR(P, P')** is defined by

\[
\exists X X'. ( \text{CR}(X, X') \land \bigwedge_i P_i \leftrightarrow \Psi_i(X) \land \bigwedge_i P_i' \leftrightarrow \Psi_i(X') )
\]

– **Basic computation: existential quantification**
Existential Quantification

- Let $\Phi(X, V)$ be a formula where
  - $V$ are Boolean variables (important variables)
  - $X$ are the other variables

- Compute a Boolean formula equivalent to $\exists X. \Phi(X, V)$

- Example (Boolean case):
  - $\exists B. (A \land (B \lor C))$
  - $V = \{A, C\}$

- Example:
  - $\exists x y. ( (P \leftrightarrow x + y = 2) \land (Q \leftrightarrow x - y < 10) \land x + y > 12 )$
  - $V = \{P, Q\}$
[LNO'06] use SMT solver on $\Phi(x, V)$

Compute all satisfiable assignments to $V$

```plaintext
SMTAbstract(Phi, V) {
  res = false;
  loop {
    mu = SMT(Phi);
    if mu == UNSAT then return res;
    else
      vmu = restrict(V, mu);
      res = res or vmu;
      Phi = Phi and \neg vmu;
  }
}
∃ B. (A ∧ (B ∨ C))

V = { A, C }

First iteration:
mu: A, ¬C, B
vmu: A, ¬C
res: A, ¬C
blocking clause: ¬A or C

Second iteration:
mu: A, C, ¬B
vmu: A, C
res: (A,C) or res = A
blocking clause: ¬A ∨ ¬C

Third iteration: unsatisfiable

Result: A

In fact,

∃ B. (A and (B or C)) reduces to

(A and (true or C)) or
(A and (false or C))

that is, A
AllSMT at work (Theory case)

- $\exists x\ y. (P \leftrightarrow (x + y = 2)) \land (Q \leftrightarrow (x - y < 10)) \land (x + y > 12)$
- $V = \{P, Q\}$

**First iteration:**
- $\mu$: $\neg P, \neg (x + y = 2), \neg Q, \neg (x - y < 10), (x + y > 12)$
- $\nu\mu$: $\neg P, \neg Q$
- $\text{res}$: $\neg P, \neg Q$
- $\text{blocking clause}$: $P \lor Q$

**Second iteration:**
- $\mu$: $\neg P, \neg (x + y = 2), Q, (x - y < 10), (x + y > 12)$
- $\nu\mu$: $\neg P, Q$
- $\text{res}$: $(\neg P, Q)$ or $\text{res} = \neg P$
- $\text{blocking clause}$: $P \lor \neg Q$

**Third iteration:** unsatisfiable

**Result:** $\neg P$
Hybrid abstraction: BDD + SMT [FMCAD’07]

◆ **AllSMT: a closer look**
  - The approach constructs the DNF of the result
  - Enumerating all the disjuncts
  - Can blow up in number of disjuncts

◆ **Binary Decision Diagrams (BDDs)**
  - Canonical representation for Boolean functions
  - Can blow up in space
    » Order of variables can make a difference
  - Core of traditional EDA tools
    » Often replaced by SAT techniques
    » Capacity, automation, …
  - But …
    » In practice, can be extremely efficient
    » They provide QBF functionalities
      » $\exists x.\Phi(x, V) \equiv \Phi(\text{false}, V) \lor \Phi(\text{true}, V)$
      » Fundamental operation in model checking

◆ **The idea**
  - extend BDD-based quantification
  - to deal with theory constraints
Hybrid abstraction: BDD + SMT [FMCAD’07]

◆ Intuitive reduction

- \( \exists x. \Phi(x, V) \)
- \( \exists x. \Phi(C_1(x), \ldots, C_n(x), V) \)
- \( \exists x A_1, \ldots, A_n. (\Phi(A_1, \ldots, A_n, V) \land \bigwedge_i (A_i \leftrightarrow C_i(x))) \)
- \( \exists A_1, \ldots, A_n. \Phi(A_1, \ldots, A_n, V) \)
  » this is BDD existential quantification, but…
  » "modulo theory", i.e. interpreting each \( A_i \) as \( C_i(x) \)

◆ Result

- A BDD whose paths are all theory consistent
Hybrid abstraction: BDD + SMT [FMCAD’07]

- An SMT solver without selection heuristic
- NOT a theory solver!

- Contains stack and implication graph
- Carries out BCP
Hybrid abstraction: a closer look

- BDD is monolithic
- No reuse of theory lemmas and
- No learning theory conflicts

New version where

- BDD is no longer monolithic
- Reuse of theory lemmas
- Learning of theory conflicts
- Tighter integration and collaboration of BDDs and SMT solver
Counterexample analysis

◆ The abstract counterexample:
  – $AS_0(P), \ AS_1(P), \ldots, AS_{n-1}(P), \ AS_n(P)$

◆ It has a corresponding concrete counterpart if

\[
\wedge_{j \in [0,n]} \left( \wedge_i P_i \leftrightarrow \Psi_i(X^i) \land AS_i(P^i) \right) \land CI(X^0) \land \wedge_{j \in [0,n-1]} CR(X^j, X^{j+1})
\]

◆ Solved as a call to the SMT solver
  – If satisfiable then it is a counterexample for the concrete model
  – If unsatisfiable then the counterexample is spurious
Refinement

◆ Analyze simulation of abstract trace in the concrete

◆ Discover new predicates to refine the abstraction via removal of the spurious abstract transition:
  – Weakest precondition
  – Extraction of the unsatisfiable core
  – Use of Craig interpolants
Outline

- Motivations
- Symbolic encoding of Slim models
- Satisfiability Modulo Theory
- Bounded Model Checking via SMT
- Counterexample Guided Abstraction Refinement
- The NuSMV tool
- The COMPASS toolset
- Conclusions
The NuSMV model checker

- Provides advanced symbolic model checking algorithms
  - BDD based algorithms
  - SAT based algorithms

- Extended to deal with infinite state domains (Integers, Reals)

- Tightly integrated with the MathSAT SMT solver
  - Bitvectors, IDL, RDL, LIA, LRA, EUF

- Bounded Model checking with SMT and SAT

- Implement full CEGAR loop
  - Predicate abstraction via AllSMT, Hybrid-BDD-SMT, Partitioned- Hybrid-BDD-SMT
  - State of the art Boolean model checking
  - Check for spuriousness via SMT
  - Refinement via SMT unsat core extraction, interpolants, weak preconditions
Outline

◆ Motivations
◆ Symbolic encoding of Slim models
◆ Satisfiability Modulo Theory
◆ Bounded Model Checking via SMT
◆ Counterexample Guided Abstraction Refinement
◆ The NuSMV tool
◆ The COMPASS toolset
◆ Conclusions
The COMPASS tool suite

- The Requirements Analysis Tool (RAT)
  - http://rat.fbk.eu

- The NuSMV MC extended with MathSAT and FSAP
  - http://nusmv.fbk.eu
  - http://mathsat.fbk.eu
  - http://fsap.fbk.eu

- The Markov Reward Model Checker
  - http://www.mrmc-tool.org

- The Symbolic Bisimulation Tool Sigref
  - http://sigref.gforge.avacs.org/
The COMPASS tool suite
The COMPASS tool suite
The COMPASS tool suite

![COMPASS Prototype Tool interface](image)

**Properties**
- **Name**
  - Observed output
  - Always output is

**Model Checking**
- **Model Checking Options**
  - Use BDD (CTL and LTL)
  - Use SAT (LTL only)
  - SAT Bound: 10
  - Use SBMC
  - Try to Complete

**The property is false**

The LTL property: $G !output$

has been found **false**. A counter-example is shown below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Step1</th>
<th>Step2</th>
<th>Step3</th>
<th>Step4</th>
<th>Step5</th>
<th>Step6</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>init</td>
<td>gone_rnd2</td>
<td>gone_rnd12</td>
<td>gone_bit2</td>
<td>gone_bit12</td>
<td>goto</td>
</tr>
<tr>
<td>run</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rnd1.output</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rnd2.output</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The COMPASS tool suite

You can generate a FMEA table

<table>
<thead>
<tr>
<th>Num</th>
<th>Failure Model</th>
<th>Failure Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>adder_errorSubcomponent._errorState = _stuck at one</td>
<td>output</td>
</tr>
<tr>
<td>2</td>
<td>adder_errorSubcomponent._errorState = _stuck at zero</td>
<td>output</td>
</tr>
</tbody>
</table>
The COMPASS tool suite
Conclusions

- We have presented a symbolic encoding for Slim models
- We have described advanced model checking techniques based on the use of SMT
- The verification techniques have been integrated in an extended version of the NuSMV symbolic model checker
- The symbolic encoding, and NuSMV are the enabling technologies for the verification functionalities of the COMPASS tool suite
- We have developed a first prototype of the COMPASS tool suite providing
  - Requirements validation via RAT
  - Correctness checks of CTL/LTL properties
  - Model simulation
  - (Probabilistic) Safety Analysis
  - (Probabilistic) FDIR
Thanks!!

A demo of the current COMPASS tool suite prototype is available on request