# Extensible Symbolic Analysis of Systems

José Meseguer

University of Illinois at Urbana-Champaign

# Motivation

Many distributed systems such as:

- distributed cyber-physical systems; and
- secure distributed systems

are open, interacting with an external, possibly hostile, environment; and are often safety-critical.

At present, analyzing such systems with methods that are scalable and amenable to automation is difficult because:

1. they are highly concurrent, and often infinite-state;
2. the external enviroment they interact with is highly non-deterministic.

# Some Partial Answers

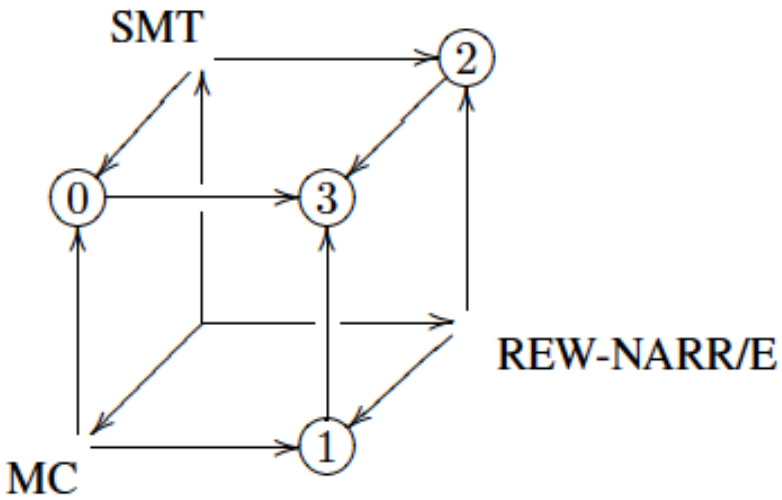
Several symbolic, formal analysis techniques provide certain ways to automatically analyze open systems, with varying degrees of scalability:

1. **Automata-Based Model Checking**, where possibly infinite sets of behaviors and/or states are symbolically represented by various kinds of automata;
2. **SMT Solving**, where, for domains having decidable theories, possibly infinite sets of states are symbolically represented as constraints; and
3. **Rewriting and Narrowing Modulo Theories**, where, modulo an equational theory $E$, $E$-equivalence classes of states, or even patterns defining infinite sets of such equivalence classes modulo $E$ are represented as terms, and their transitions as rewrite rules.

# The Extensibility Problem

What we lack are extensibility techniques that can combine the power of SMT solving, rewriting- and narrowing-based analysis, and automata-based model checking; and new formal tools that can apply these symbolic techniques together.

The needed combinations are summarized in the “cube” below as vector additions.




0 SMT-MC

1 REW-NARR/E – MC

2 REW-NARR/SMT+E

3 REW-NARR/SMT+E – MC

# Plan of This Talk

In the rest of this talk I will present some recent work towards completing the above cube within the context of Maude and rewriting logic, including:

- Explaining techniques we are developing for rewriting and narrowing modulo $E$ for a rich variety of equational theories, and model checking methods based on that.
- Explaining techniques we are developing for rewriting modulo $E + SMT$, and model checking methods based on that.

**Acknowledgements**. The work presented is joint work with Andrew Cholewa, Kyungmin Bae, Steven Eker, Santiago Escobar, Vijay Ganesh, Catherine Meadows, César Muñoz, Paliath Narendran, Camilo Rocha, Ralf Sasse, Carolyn Talcott and Fan Yang.

# Outline

# Rewriting Logic in a Nutshell

Rewriting logic is a flexible logical framework to specify concurrent systems.

- A concurrent system specified as rewrite theory $\mathcal{R} = (\Sigma, E, R)$ where:
  - $\Sigma$ is signature defining the syntax of the system and of its states
  - $E$ is a set of equations defining system's states as an algebraic data type
  - $R$ is a set of rewrite rules of the form $t \rightarrow t'$, specifying system's local concurrent transitions.
- Rewriting logic deduction consists of applying rewriting rules $R$ concurrently, modulo the equations $E$.

Maude, CafeOBJ and Elan are rewrite engines capable of executing rewrite theories. Maude provides several model checkers and theorem proving tools.

# Rewriting Modulo $E$ and Model Checking

This is the area where we have the longest experience.

A concurrent system is represented as a rewrite theory $\mathcal{R} = (\Sigma, G \cup B, R)$, where $E = G \cup B$, the equational theory we rewrite modulo, can be any convergent and coherent theory modulo $B$, with $B$ any combination of $A$, $C$, and $U$ axioms.

Thanks to the requirement that the rewrite rules $R$ describing the system's transitions are coherent with $G$ modulo $B$, rewriting with $R$ modulo $E$ can be achieved by rewriting with $R$ modulo $B$.

Model checking temporal logic properties of the system axiomatized by $\mathcal{R} = (\Sigma, G \cup B, R)$ can be done not only for state-based *LTL* properties, but, as developed in joint work with Kyungmin Bae, for state/event properties in the linear temporal logic of rewriting *LTLR*, and under parameterized fairness conditions.

# Outline

# Modeling Open Systems with Rewrite Theories

**Q**: How is the fact that a distributed system is open, i.e., interacts with a non-deterministic environment, modeled in the rewrite theory $\mathcal{R} = (\Sigma, G \cup B, R)$?

**A**: It is modeled by the fact that the (possibly conditional) rewrite rules in $R$, axiomatizing the system transitions may have extra variables $\vec{y}$ in their righthand side. I.e., rules have the form:

$$t(\vec{x}) \rightarrow t'(\vec{x}, \vec{y}) \ \textbf{if} \ C$$

Such rules appear naturally in:

- rules with non-equational rewrite conditions;
- rules describing interactions with an environment, e.g., time (Real-Time Maude "tick" rules), and values of sensors;
- probabilistic rewrite rules, where the $\vec{y}$ are chosen with a probability distribution modeling the environment.

# Outline

# Symbolic Reachabilty Analysis by Narrowing Modulo

Given a rewrite theory $\mathcal{R} = (\Sigma, G \cup B, R)$, the narrowing modulo $G \cup B$ relation

$$t \rightsquigarrow_{R,(G \cup B)} t'$$

is defined if there is:

- a non-variable position $p \in Pos(t)$;
- a rule $l \rightarrow r$ in $R$; and
- a $(G \cup B)$-unifier $\sigma$ such that $\sigma(t|_p) =_{(G \cup B)} \sigma(l)$, and $t' = \sigma(t[r]_p)$.

As shown by Meseguer and Thati, if $\mathcal{R}$ is a topmost rewrite theory, narrowing modulo $G \cup B$ is a complete reachability analysis method to solve queries of the form:

$$(\exists \vec{x})\ t \longrightarrow^* t'$$

Note that narrowing can be performed with non-deterministic rules $t(\vec{x}) \rightarrow t'(\vec{x}, \vec{y})$ in $R$.

# Technical Challenges of Narrowing Modulo $G \cup B$

In order to make narrowing modulo $G \cup B$ a practical symbolic method to analyze a system specified by $\mathcal{R} = (\Sigma, G \cup B, R)$, two important questions had to be answered:

1. How can unification modulo $G \cup B$ be efficiently supported?
2. How can not just symbolic reachability analysis but narrowing-based *LTL* model checking be supported?

Question (1) has been answered by Escobar, Meseguer and Sasse: $(G \cup B)$-unifiers can be optimally computed by folding variant narrowing for any convergent and coherent $G$ modulo $B$.

Question (2) has been answered by Bae, Escobar and Meseguer by developing narrowing-based *LTL* and *LTLR* model checking algorithms and model checkers.

# Outline

# Folding Variant Narrowing in a Nutshell

If $G$ is coherent and convergent modulo $B$, Jouannaud, K. Kirchner, and H. Kirchner proved that narrowing modulo $B$ provides a complete $(G \cup B)$-unification procedure.

Narrowing can be very inefficient. When the set $B$ of axioms is empty, Hullot's basic narrowing has been used as an efficient, complete strategy.

Even for $AC$ basic narrowing is incomplete. No efficient strategy was known in the terra incognita of narrowing modulo $B$.

Folding variant narrowing is an optimally terminating, complete strategy for narrowing with $G$ modulo $B$:

- if any complete strategy terminates on an input problem, folding variant narrowing will also;
- even for $B = \emptyset$, folding variant narrowing terminates strictly more often than basic narrowing.

# Folding Variant Narrowing in a Nutshell (II)

Folding variant narrowing has been developed by Escobar, Meseguer, and Sasse, using the Comon-Delaune notion of variant for $G$ convergent and coherent equations modulo $B$.

A $G/B$-variant of a term $t$ is a pair $(u, \theta)$ with $\theta$ a substitution and $u$ a $G/B$-canonical form of $t\theta$. Variants are naturally (pre-)ordered by a $B$-subsumption relation. $G \cup B$ has the finite variant property (FVP) if there is a finite set of most general variants.

Folding variant narrowing only keeps those narrowing steps $t \overset{\theta}{\leadsto}{}^{*}_{G,B} u$ where: (i) $(u, \theta)$ is a variant; (ii) $\theta$ is $G/B$-normalized, and (iii) no earlier computed variant subsumes $(u, \theta)$. This provides an optimally terminating narrowing-based $G \cup B$-unification procedure, which is finitary iff $G \cup B$ is FVP.

# Folding Variant Narrowing in a Nutshell (III)

Maude 2.7 (soon to be released) supports variant-based unification for any equations $G$ that are convergent modulo $B$, with $B$ a combination of $A$, $C$ and $U$ axioms, except $A$ without $C$.

The number of generated unifiers is always finite for $G \cup B$ FVP.

**Theorem.** $(\Sigma, G \cup B)$ is FVP iff for each $f \in \Sigma$, $f(x_1, \ldots, x_n)$ has a finite number of variants, denoted $|variants(f(x_1, \ldots, x_n))|$ $\Box$

**Definition.** Let $(\Sigma, G \cup B)$ be convergent modulo $B$. Its variant complexity is the number

$$\Sigma_{f \in \Sigma} |variants(f(x_1, \ldots, x_n))|$$

or $\infty$ if $|variants(f(x_1, \ldots, x_n))|$ infinite for some $f \in \Sigma$.

# Outline

# The Maude-NPA Crypto Protocol Analyzer

Narrowing modulo $G \cup B$ with the rules $R$ of a rewrite theory $\mathcal{R} = (\Sigma, G \cup B, R)$ has been implemented in Maude for $G \cup B$ FVP. This is the basis of the Maude-NPA tool of Escobar, Meadows and Meseguer, where a crypto protocol modeled as $\mathcal{P} = (\Sigma, G \cup B, R)$ is analyzed modulo $G \cup B$.

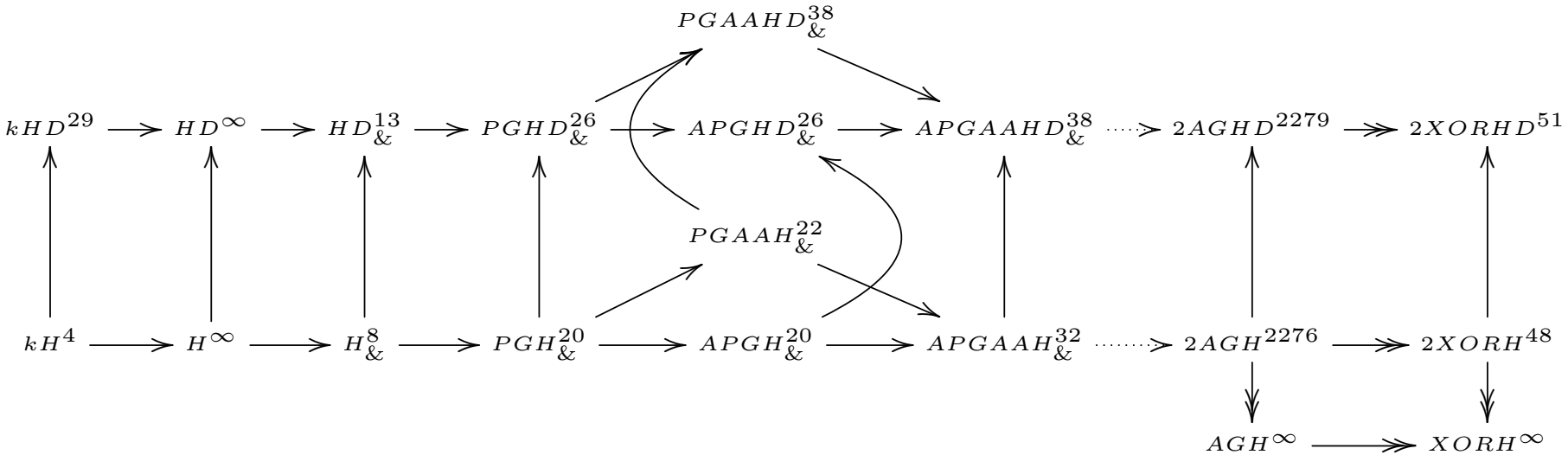
Many protocols have been or are being analyzed modulo non-trivial theories such as: (i) encryption-decryption; (ii) exclusive or; (iii) Diffie-Hellman exponentiation; (iv) abelian groups, (v) homomorphic encryptions, and combinations of such theories.

Although Maude-NPA deals with unbounded sessions for which reachability is undecidable, its use of very effective symbolic state space reduction techiques often makes the state space finite, allowing full verification.

# The Maude-NPA Crypto Protocol Analyzer (II)

Homomorphic encryption is challenging: the theories *H* and *AGH* are not FVP, and combining their unification algorithms with those of other theories is computationally expensive.

In recent work of Yang et al. (PPDP14), a host of FVP theories of homomorphic encryption have been identified and used with protocols in Maude-NPA with a clear tradeoff between accuracy and variant complexity.



# Outline

# Maude's Narrowing-Based LTL(R) Model Checker

Many concurrent systems are infinite-state.

The Maude Logical Bounded Model Checker tool developed by K. Bae with S. Escobar and J. Meseguer is an infinite-state model checker for LTL and LTLR properties supporting:

- Symbolic representation of states and transitions through narrowing.
- Acceleration using folding
- Abstraction using equational abstractions
- bounded model checking, which can detect a finite symbolic space to provide full verification.

# Lamport's Bakery Example

In Lamport's Bakery protocol for mutual exclusion each state with $n$ processes:

$$i \; ; \; j \; ; \; [k_1, m_1] \ldots [k_n, m_n]$$

- $i$: the current number in the bakery's number dispenser,
- $j$: the number currently served,
- $[k_l, m_l]$: a process $k_l$ in a mode $m_l$, either `idle`, `wait(t)`, or `crit(t)`.

Behaviors:

```
rl [wake]: N ; M ; [K, idle] PS => s N ; M ; [K, wait(N)] PS .
rl [crit]: N ; M ; [K, wait(M)] PS => N ; M ; [K, crit(M)] PS .
rl [exit]: N ; M ; [K, crit(M)] PS => N ; s M ; [K, idle] PS .
```

Mutual exclusion: $\Box ex?$ where:

```
eq N ; M ; [K1, crit(M1)] [K2, crit(M2)] PS |= ex? = false .
```

# Lamport's Bakery Example (II)

The commands below show the results of the bounded model checking with depth 10, and of full model checking using an equational abstraction, for an arbitrary number of processes.

```
Maude> (lmc [10] N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex? .)

logical model check in BAKERY-SAFETY-SATISFACTION :
  N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex?
result:
  no counterexample found within bound 10

Maude> (lfmc N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex? .)

logical folding model check in BAKERY-SAFETY-SATISFACTION-ABS :
  N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex?
result:
  true
```

The tool is available at

http://formal.cs.illinois.edu/kbae/lmc/

# Unification-Based Predicate Abstraction

Predicate abstraction $\mathcal{S}/AP$ of system $\mathcal{S}$ with state predicates $AP$ and state-labeling function $\mathcal{L}$ has:

(i) set of states $s \in AP$, and (ii) transitions $s \to s' \in \mathcal{S}/AP$ iff $\exists$ concrete transition $t \to t' \in \mathcal{S}$ with:

$$\mathcal{L}(t) = s \quad \wedge \quad \mathcal{L}(t') = s'$$

Then for $\varphi$ any LTL formula, $\mathcal{S}/AP \models \varphi \;\Rightarrow\; \mathcal{S} \models \varphi$.

The problem is finding a witness for the existential quantifier.

In RTA 2014, K. Bae and J. Meseguer present a unification-based method to automatically build $\mathcal{S}/AP$ or an approximation of it for rewrite theories of the form: $\mathcal{R} = (\Sigma, E \cup B, R)$ with $E$ convergent modulo $B$ and $R$ possibly conditional rules.

# Outline

# Rewriting Modulo SMT

Camilo Rocha's thesis and subsequent work by Rocha, Meseguer and Muñoz consider rewrite theories $\mathcal{R} = (\Sigma, E, R)$ such that:

- The signature $\Sigma$ is ordered-sorted, there is a downward closed sub-poset of sorts $(S_0, \leq)$ and a subsignature $\Sigma_0$ on $(S_0, \leq)$ such that for each $f : w \to s \in \Sigma - \Sigma_0$, $s \in S - S_0$, and there are no subsort-overloaded operators in common between $\Sigma_0$ and $\Sigma - \Sigma_0$.
- The equations $E$ are a disjoint union $E = B \uplus E_0$, with the $E_0$ $\Sigma_0$-equations and the $B$ regular and collapse-free $\Sigma - \Sigma_0$-axioms with a matching algorithm. Furthermore, the theory of the initial algebra $\mathcal{T}_{\Sigma_0/E_0}$ is decidable.
- The rules in $R$ are topmost, with top sort in $S - S_0$, and are non-deterministic and conditional, having the form

$$t(\vec{x}) \to t'(\vec{x}, \vec{y}) \ \textbf{if} \ \phi$$

with the sorts of variables $\vec{y}$ in $S_0$, and $\phi$ a q.f. $\Sigma_0$-formula.

# Rewriting Modulo SMT (II)

Intuitively, such a theory specifies an open system whose non-deterministic environment consists only of inputs and outputs in the decidable theory of $\mathcal{T}_{\Sigma_0/E_0}$.

Because of its non-determinism, the theory is not executable in the usual sense. However, we can easily transform each rewrite rule $t(\vec{x}) \rightarrow t'(\vec{x},\vec{y})$ **if** $\phi$ in $R$, into an equivalent rule where $t$ and $t'$ have no non-variable $\Sigma_0$-subterms, and all variables in $S_0$ are linear.

This can be achieved by decomposing $t$ as $t = C[u_1,\ldots,u_n]$, and $t'$ as $t' = D[v_1,\ldots,v_n,\vec{y}]$, where the $u_i$, $v_j$, are maximal $\Sigma_0$-subterms and transforming the above rule into the rule:

$$t = C[\vec{z}] \rightarrow D[\vec{z'},\vec{y}] \ \textbf{if} \ \vec{z} = \vec{u} \wedge \vec{z'} = \vec{v} \wedge \phi.$$

where the $\vec{z}$ and $\vec{z'}$ are abstraction variables with sorts in $S_0$.

# Rewriting Modulo SMT (III)

Using such a transformation, we can define the, non-executable but decidable, $\mathcal{R}$-rewrite relation of ground terms $u \to_{\mathcal{R}} v$ iff there is a rule $t \to t'$ **if** $\phi$ in $R$ and a ground substitution $\theta$ of all its variables such that: (i) $u =_E t\theta$ and $v =_E t'\theta$, and (ii) $\mathcal{T}_{\Sigma_0/E_0} \models \phi\theta$.

Using an SMT solver, we can make rewriting in $\mathcal{R}$ executable in a symbolic way by rewriting constrained terms of the form $u \mid \psi$, with $\psi$ a quantifier-free $\Sigma_0$-formula; where the variables of $t \mid \psi$ all have sorts in $S_0$, and are always disjoint from those of $R$. We say that the SMT-rewriting relation

$$u \mid \psi \rightsquigarrow_{\mathcal{R}} v \mid \delta$$

holds iff there is a rule $t \to t'$ **if** $\phi$ in $R$ and a substitution $\theta$ such that: (i) $u =_B t\theta$ and $v =_B t'\theta$, with $\theta$ a renaming with fresh new variables for all variables in $t \to t'$ **if** $\phi$ but not in $t$; and (ii) $\mathcal{T}_{\Sigma_0/E_0} \models \delta \Leftrightarrow (\psi \wedge \phi\theta)$. A Lifting Lemma holds, relating $\to_{\mathcal{R}}$ and $\rightsquigarrow_{\mathcal{R}}$.

# Symbolic Semantics of NASA's PLEXIL Language

NASA's PLEXIL Language for distributed programmig of robot tasks has been used in:

- Mars Drill
  - executive for the Drilling Automation for Mars Exploration drilling application
  - used at the Haughton Crater on Devon Island, perhaps, in the first fully automated drill rig
- International Space Station
  - demonstrate automation for ISS operations
- Habitat Demonstration Unit
  - automated control of several subsystems

# Symbolic Semantics of NASA's PLEXIL Language (II)

Although PLEXIL is a deterministic language, its interaction with a physical environment through sensors and actuators make it non-deterministic.

Muñoz and Rocha developed a formal executable semantics of PLEXIL in Maude that is used regularly within NASA to analyze PLEXIL programs and to design new versions of PLEXIL. However, the non-determinism of the environment had to be dealt with by assuming specific environment inputs.

Using rewriting modulo SMT and Rocha's implementation of it in Maude using CVC-3 as an oracle, full symbolic verification of safety properties for PLEXIL programs such as absence of race conditions can be performed for arbitrary environment inputs.

# Analyzing the CASH Scheduling Algorithm

The CASH scheduling algorithm, of Caccamo, Buttazzo, and Sha attempts to maximize system performance by maintaining a queue of unused execution budgets that can be reused by other jobs to maximize processor utilization. Unbounded data types such as queues cannot be modeled in timed-automata.

Using Real-Time Maude, Ölveczky and Caccamo found a subtle deadline miss in an optimized version of CASH by explicit state model chacking.

However, the CASH algorithm is parameterized by: (i) the number $N$ of servers in the system, and (ii) the values of a maximum budget $b_i$ and period $p_i$, for each server $1 \leq i \leq N$. Even if we fix $N$, there are infinitely many initial states for $N$ servers, since the maximum budgets $b_i$ and periods $p_i$ range over the natural numbers. Therefore, explicit state model checking cannot perform a full analysis.

# Analyzing the CASH Scheduling Algorithm (II)

Rewriting modulo SMT is useful for symbolically analyzing infinite-state systems like CASH. Infinite sets of states are symbolically described by terms which may involve user-definable data structures such as queues, but whose only variables range over decidable types for which an SMT solving procedure is available.

The symbolic transitions of CASH are specified by 14 conditional rewrite rules whose conditions specify constraints solvable by the SMT procedure. For example, the rule below models the detection of a deadline miss for a server with nonzero maximum budget.

```
crl [deadlineMiss] : [iB, < G : global | dead-miss |-> B, AtSG > <
O : server | state |-> St, usedOfBudget |-> iT, timeToDeadline |->
iT', maxBudget |-> iNZT, AtS > Cnf] => [iB & iT >= c(0) & iNZT > c(0)
& iT' > c(0) & iNZT > iT + iT', <G:global |dead-miss|->true,AtSG>
<O : server | state |-> St, usedOfBudget |-> iT, timeToDeadline |->
iT', maxBudget |-> iNZT, AtS> Cnf] if St =/= idle / check-sat(iB &
iT >= c(0) & iNZT > c(0) & iT' > c(0) & iNZT > iT + iT') .
```

# Analyzing the CASH Scheduling Algorithm (III)

We want to verify symbolically the existence of missed deadlines of the CASH algorithm for the infinite set of initial configurations containing two server objects $s_0$ and $s_1$ with maximum budgets $b_0$ and $b_1$ and periods $p_0$ and $p_1$ as unspecified natural numbers, and such that each server's maximum budget is strictly smaller than its period (i.e., $0 \leq b_0 < p_0 \wedge 0 \leq b_1 < p_1$).

This infinite set of initial states is specified symbolically by the equational definition (not shown) of term `symbinit`. Maude's `search` command can then be used to symbolically check if there is a reachable state for any ground instance of `symbinit` that misses the deadline.

# Analyzing the CASH Scheduling Algorithm (IV)

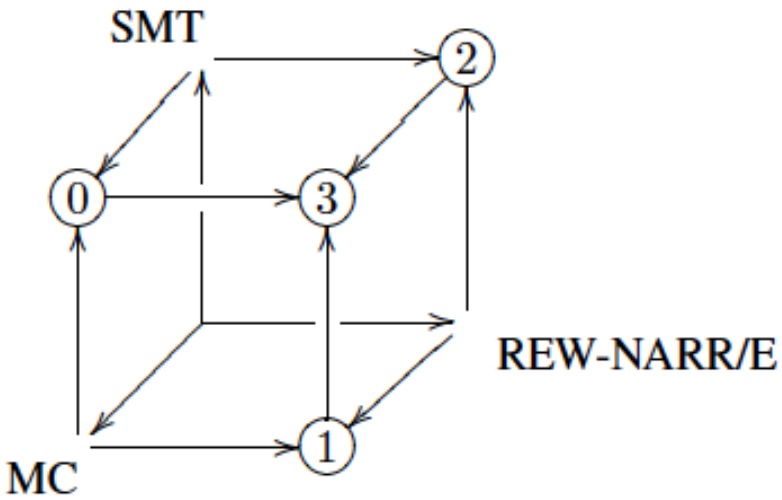
```
search in SYMBOLIC-FAILURE : symbinit =>*
  {iB:iBool,  Cnf:Configuration
              < g : global | AtS:AttributeSet, deadline-miss |-> true >} .
Solution 1 (state 233)
states: 234  rewrites: 60517 in 2865ms cpu (2865ms real) (21118 rewrites/second)
iB:iBool --> ((i(0) <= c(0) ^ i(1) <= c(0)) v i(0) <= c(0) + i(1) ^
              ... ^ ((i(1) -- c(1)) -- c(1)) > c(0)
Cnf:Configuration -->
< s1 : server | maxBudget |-> i(0), period |-> i(1), state |-> waiting,
           usedOfBudget |-> c(0), timeToDeadline |-> ((i(1) -- c(1)) -- c(1)), timeExecuted |-> c(0) >
< s2 : server | maxBudget |-> i(2), period |-> i(3), state |-> executing,
           usedOfBudget |-> c(2), timeToDeadline |-> ((i(3) -- c(1)) -- c(1)), timeExecuted |-> c(2) >
AtS:AttributeSet --> time |-> c(2), cq |-> emptyQueue, available |-> false
```

A counterexample is found at (modeling) time two, after exploring 233 symbolic states in less than 3 seconds. By using a satisfiability witness of the constraint `iB` computed by the search command, a concrete counterexample is found by exploring only 54 ground states. This result compares favorably, in both time and computational resources, with the ground counterexample found by explicit-state model checking, where more that 52,000 concrete states where explored before finding a counterexample.

# Outline

# Conclusion

Extensible, symbolic methods that combine the power of automata-based model checking, SMT solving, and rewriting/narrowing modulo theories can make the analysis of open distributed system much more scalable and mostly automatic. I have given a summary of recent work completing the “cube” to combine such power.




0 SMT-MC

1 REW-NARR/E – MC

2 REW-NARR/SMT+E

3 REW-NARR/SMT+E – MC

But, as usual, much more work remains ahead.