Runtime monitoring of contract regulated web services

Maciej Szreter[2]

a joint work with Alessio Lomuscio[3], Wojciech Penczek[1,2] and Monika Solanki[3]

[1]University of Podlasie, Siedlce, Poland

[2]Polish Academy of Sciences, Warsaw, Poland

[3]Department of Computing, Imperial College London, UK

Toledo, September 2009

Outline

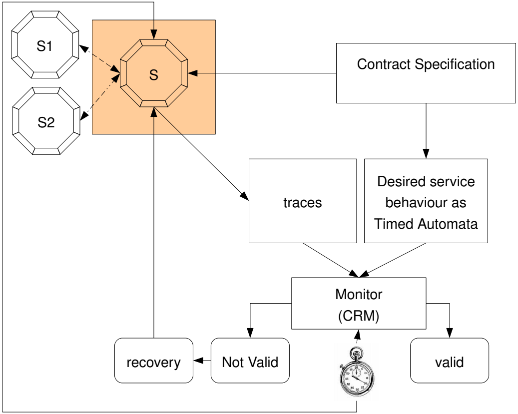
Outline

Related work

- Several papers on monitoring of web services (WS)
- Monitoring of WS based on model checking
  - [8] Krichen, Tripakis. Black-box conformance testing for real-time systems. *In 11th International SPIN Workshop on Model Checking of Software (SPIN04)*
  - [15] Raimondi, Skene, Chen, Emmerich. Efficient monitoring of web service SLAs. *Technical report, UCL, London, 2007.*
- [NEW] Symbolic model checking approach
  - no need to construct the product automaton
  - model checkers designed for model checking
  - easy to extend

Outline

The general architecture



The general architecture and methodology

Monitoring

- contract C → modeled by Time Automata
- web services WS → we consider only executions (snapshots)
- check if WS execute according to C

WS possibly distributed, we do not know implementations and specifications

Outline

Repair Car

- Contract between Repair Car (RC) and Customer (C)

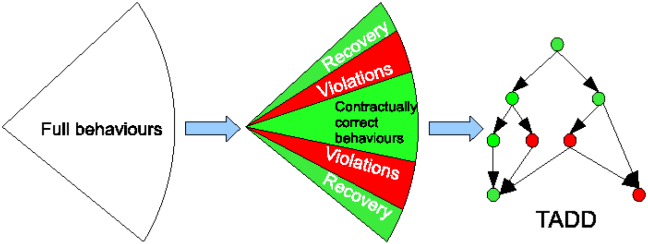
clause	Contract regulated actions	Deadline	Violate	Recover
1	Receives a repair request by *C*	5 days	-	-
2	Sends a repair proposal to *C*	7 days	-	-
3	Assess damage to the vehicle	3 days	yes	yes
4	Execute repair	30 days	yes	yes
5	Send repair report to *C*	5 days	yes	yes
6	For any violation take recovery action	3 days	yes	no (*)

Some contract regulated actions for *RC*
* - (take offline action)

Explanation

Step	state	Explanation
step 1	source	RC waits to receives the request for repairing cars.
	target	RC receives the request for repairing x cars. We show here an example of the clock and variable valuations for three cars.
step 3	source	RC accepts the request for repairing x cars.
	target	RC sends repair proposals for repairing x cars.

Explanation of trace contents for steps 1 and 3



Set of behaviours for a service

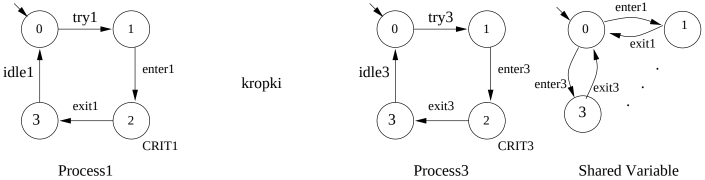
Outline

Networks of automata

Fischer's mutual exclusion

- n components; $A_i = (L_i, l_i^\iota, T_i, \Sigma_i)$
- product automaton (model): $A = A_1 || \ldots || A_n$
- initial state $l^\iota = (l_1^\iota, \ldots, l_n^\iota)$
- set of labels Σ

Simplified mutual exclusion protocol [Fischer]:

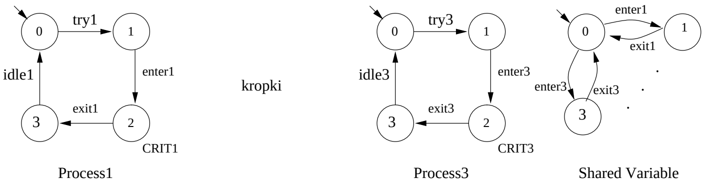


Networks of automata

Fischer's mutual exclusion

- n components; $A_i = (L_i, l_i^\iota, T_i, \Sigma_i)$
- product automaton (model): $A = A_1 || \ldots || A_n$
- initial state $l^\iota = (l_1^\iota, \ldots, l_n^\iota)$
- set of labels Σ

Simplified mutual exclusion protocol [Fischer]:

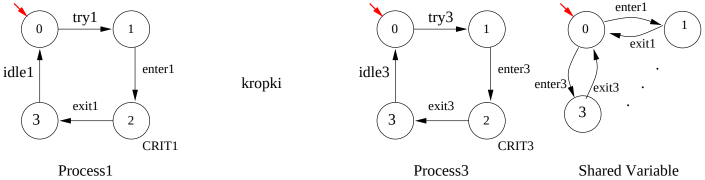


Networks of automata

Fischer's mutual exclusion

- n components; $A_i = (L_i, l_i^\iota, T_i, \Sigma_i)$
- product automaton (model): $A = A_1 || \ldots || A_n$
- initial state $l^\iota = (l_1^\iota, \ldots, l_n^\iota)$
- set of labels Σ

Simplified mutual exclusion protocol [Fischer]:

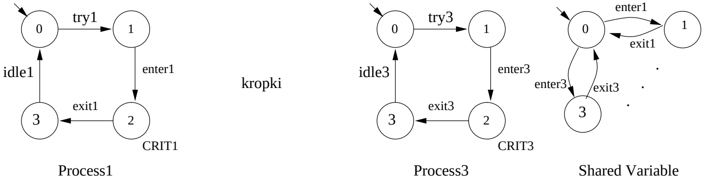


Networks of automata

Fischer's mutual exclusion

- n components; $A_i = (L_i, l_i^\iota, T_i, \Sigma_i)$
- product automaton (model): $A = A_1 || \ldots || A_n$
- initial state $l^\iota = (l_1^\iota, \ldots, l_n^\iota)$
- set of labels Σ

Simplified mutual exclusion protocol [Fischer]:

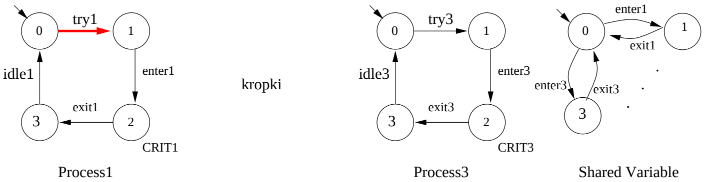


Networks of automata

Fischer's mutual exclusion

- n components; $A_i = (L_i, l_i^\iota, T_i, \Sigma_i)$
- product automaton (model): $A = A_1 || \dots || A_n$
- initial state $l^\iota = (l_1^\iota, \dots, l_n^\iota)$
- set of labels Σ
- local transitions

Simplified mutual exclusion protocol [Fischer]:

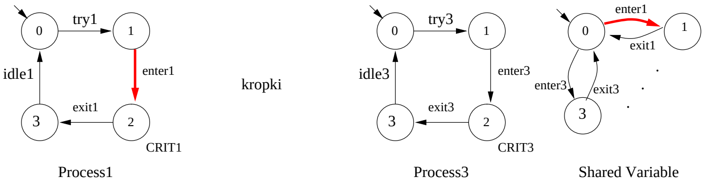


Networks of automata

Fischer's mutual exclusion

- n components; $A_i = (L_i, l_i^\iota, T_i, \Sigma_i)$
- product automaton (model): $A = A_1 || \ldots || A_n$
- initial state $l^\iota = (l_1^\iota, \ldots, l_n^\iota)$
- set of labels Σ
- synchronized transitions

Simplified mutual exclusion protocol [Fischer]:

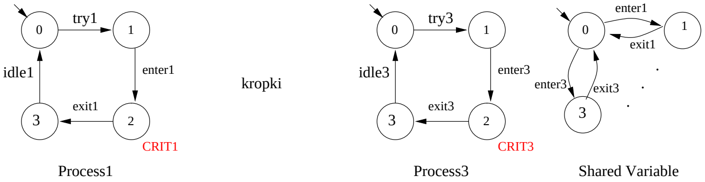


Networks of automata

Fischer's mutual exclusion

- n components; $A_i = (L_i, l_i^\iota, T_i, \Sigma_i)$
- product automaton (model): $A = A_1 || \ldots || A_n$
- initial state $l^\iota = (l_1^\iota, \ldots, l_n^\iota)$
- set of labels Σ
- properties expressed in CTL
  mutual exclusion (3 processes):
  $\varphi = EF((CRIT_1 \wedge CRIT_2) \vee (CRIT_1 \wedge CRIT_3) \vee (CRIT_2 \wedge CRIT_3))$

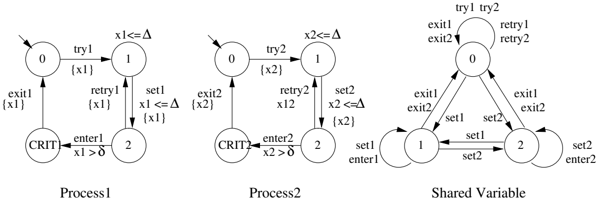
Simplified mutual exclusion protocol [Fischer]:



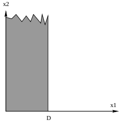
Timed automata

enable modeling time flow

- clocks, invariants, guards



time zones



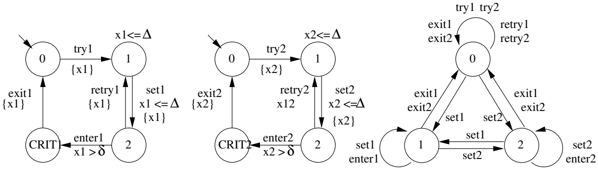
models

- concrete model
- detailed regions graph
- abstract graph

Timed automata

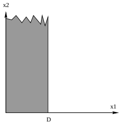
enable modeling time flow

- clocks, invariants, guards



Process1 Process2 Shared Variable

time zones



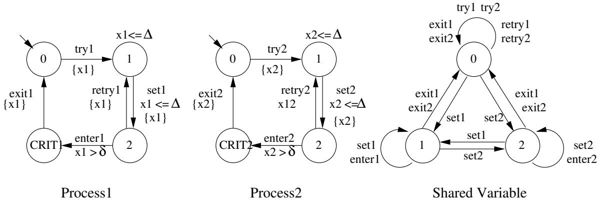
models

- concrete model
- detailed regions graph
- abstract graph

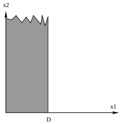
Timed automata

enable modeling time flow

- clocks, invariants, guards



time zones



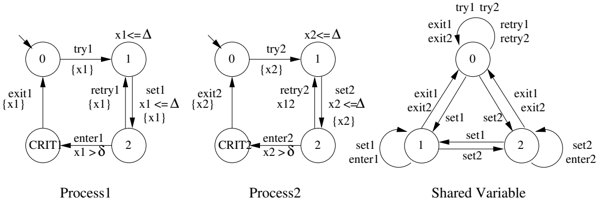
models

- concrete model
- detailed regions graph
- abstract graph

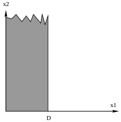
Timed automata

enable modeling time flow

- clocks, invariants, guards



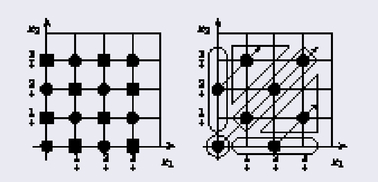
time zones



models

- concrete model
- detailed regions graph
- abstract graph

models discretized



discretized clock valuations

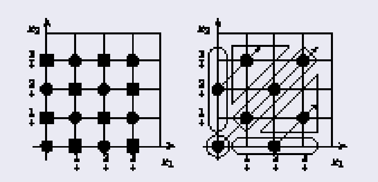
constraints

$\mathcal{X}$ - clocks

V - integer variables

$\mathcal{C}(\mathcal{X}, V)$ - *clock constraints* over $\mathcal{X}$ and V,
defined by the grammar:
$cc ::= \mathbf{true} \mid x_i \sim c \mid x_i \otimes x_j \sim c \mid x_i \otimes x_j \sim v \mid x_i \otimes v \sim c \mid v \otimes w \sim x_i \mid cc \wedge cc$, where
$x_i, x_j \in \mathcal{X}$, $v, w \in V$, $c \in \mathbb{N}$,
$\otimes \in \{+, -\}$, and
$\sim \in \{\leq, <, =, >, \geq\}$.

models discretized



discretized clock valuations

constraints

$\mathcal{X}$ - clocks

V - integer variables

$\mathcal{C}(\mathcal{X}, V)$ - *clock constraints* over $\mathcal{X}$ and V, defined by the grammar:
$\mathfrak{cc} ::= \textbf{true} \mid x_i \sim c \mid x_i \otimes x_j \sim c \mid x_i \otimes x_j \sim v \mid x_i \otimes v \sim c \mid v \otimes w \sim x_i \mid \mathfrak{cc} \wedge \mathfrak{cc}$, where $x_i, x_j \in \mathcal{X}$, $v, w \in V$, $c \in \mathbb{N}$, $\otimes \in \{+, -\}$, and $\sim \in \{\leq, <, =, >, \geq\}$.

Timed automata

- standard formalism used in model checking
- several extensions - timed, parametric...

Definition

A *timed automaton with discrete data* (TADD) is a tuple $\mathcal{A} = (\Sigma, L, l^0, V, \mathcal{X}, \mathcal{E}, \mathcal{I})$, where

- Σ is a finite set of *labels (actions)*,
- L is a finite set of *locations*,
- $l^0 \in L$ is the *initial location*,
- V is the finite set of integer variables,
- $\mathcal{X}$ is the finite set of clocks,
- $\mathcal{E} \subseteq L \times \Sigma \times Bool(V) \times \mathcal{C}(\mathcal{X}, V) \times \Sigma(V) \times Asg(\mathcal{X}) \times L$ is a *transition relation*, and
- $\mathcal{I} : L \longrightarrow \mathcal{C}(\mathcal{X}, \emptyset)$ is an *invariant function*.

The semantics of automata

The *semantics* of $\mathcal{A} = (\Sigma, L, l^0, V, \mathcal{X}, \mathcal{E}, \mathcal{I})$ for an initial valuation $\mathbf{v}^0 : V \longrightarrow \mathbb{Z}$ is a labelled transition system $\mathcal{S}(\mathcal{A}) = (Q, q^0, \Sigma_{\mathcal{S}}, \longrightarrow)$:

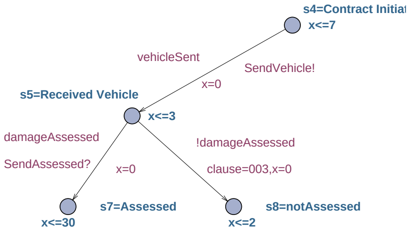
- $Q = \{(l, \mathbf{v}, \mathbf{c}) \mid l \in L \wedge \mathbf{v} \in \mathbb{Z}^{|V|} \wedge \mathbf{c} \in R_+^{|\mathcal{X}|} \wedge \mathbf{c} \models \mathcal{I}(l)\}$ is the set of states,
- $q^0 = (l^0, \mathbf{v}^0, \mathbf{c}^0)$ is the initial state,
- $\Sigma_{\mathcal{S}} = \Sigma \cup R_+$ is the set of labels,
- $\longrightarrow \subseteq Q \times \Sigma_{\mathcal{S}} \times Q$ is the smallest transition relation:
  - for $a \in \Sigma$,
    $(l, \mathbf{v}, \mathbf{c}) \xrightarrow{a} (l', \mathbf{v}', \mathbf{c}')$ iff there exists a transition $t = (l, a, \beta, \mathfrak{cc}, \alpha, A, l') \in \mathcal{E}$ such that $\mathbf{v} \models \beta$, $(\mathbf{c}, \mathbf{v}) \models \mathfrak{cc}$, $\mathbf{v}' = \mathbf{v}(\alpha)$, $\mathbf{c} \models \mathcal{I}(l)$, and $\mathbf{c}' = \mathbf{c}(A) \models \mathcal{I}(l')$ *(action transition)*,
  - for $\delta \in R_+$,
    $(l, \mathbf{v}, \mathbf{c}) \xrightarrow{\delta} (l, \mathbf{v}, \mathbf{c} + \delta)$ iff $\mathbf{c} \models \mathcal{I}(l)$ and $\mathbf{c} + \delta \models \mathcal{I}(l)$ *(time transition)*.

Outline

Repair Car contract as TA

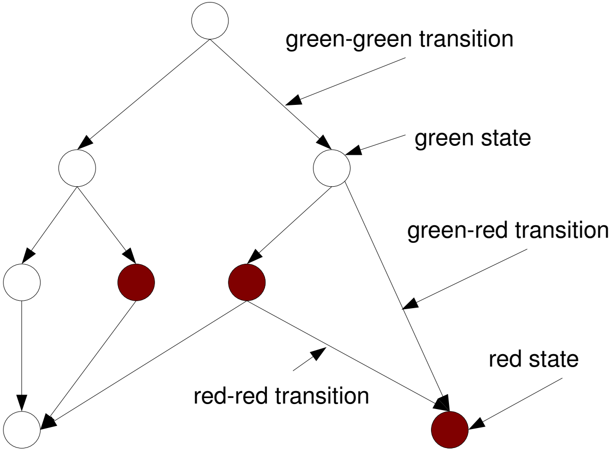
clause	Contract regulated actions	Deadline	Violation	Recovery
3	Assess damage to the vehicle	3 days	yes	yes

Some contract regulated actions for *RC*



TA specification of clause (3)

TADD semantics for RMCS



Partitioning of states and transitions in TADD

Partitioning of transitions

Based on the above partitioning each action transition (q, a, q') of $\mathcal{S}(\mathcal{A})$ can be one of the following four types of transitions:

- **Contract compliant**: between green and green states, i.e., $q, q' \in G$, (compliance with the prescribed behaviour).
- **Contract violating**: between green and red states, i.e., $q \in G$ and $q' \in R$ (violates the prescribed behaviour of the contract)
- **Recovery**: between red and green states, i.e., $q \in R$ and $q' \in G$. (a recovery action is taken by the service after a violation is recorded)
- **Continuous contract violating**: between red and red states, i.e., $q, q' \in R$ (no recovery results from a previous violation)

We say that there is a *step* from state q_1 to q_2 in $\mathcal{A}$ if $q_1 \xrightarrow{\delta_1} q'_1 \xrightarrow{a} q'_2 \xrightarrow{\delta_2} q_2$, for some states $q'_1, q'_2 \in Q$, $\delta_1, \delta_2 \in R_+$, and $a \in \Sigma$.

Partitioning of transitions

Based on the above partitioning each action transition (q, a, q') of $\mathcal{S}(\mathcal{A})$ can be one of the following four types of transitions:

- **Contract compliant**: between green and green states, i.e., $q, q' \in G$, (compliance with the prescribed behaviour).
- **Contract violating**: between green and red states, i.e., $q \in G$ and $q' \in R$ (violates the prescribed behaviour of the contract)
- **Recovery**: between red and green states, i.e., $q \in R$ and $q' \in G$. (a recovery action is taken by the service after a violation is recorded)
- **Continuous contract violating**: between red and red states, i.e., $q, q' \in R$ (no recovery results from a previous violation)

We say that there is a *step* from state q_1 to q_2 in $\mathcal{A}$ if $q_1 \xrightarrow{\delta_1} q'_1 \xrightarrow{a} q'_2 \xrightarrow{\delta_2} q_2$, for some states $q'_1, q'_2 \in Q$, $\delta_1, \delta_2 \in R_+$, and $a \in \Sigma$.

Outline

Bounded Model Checking

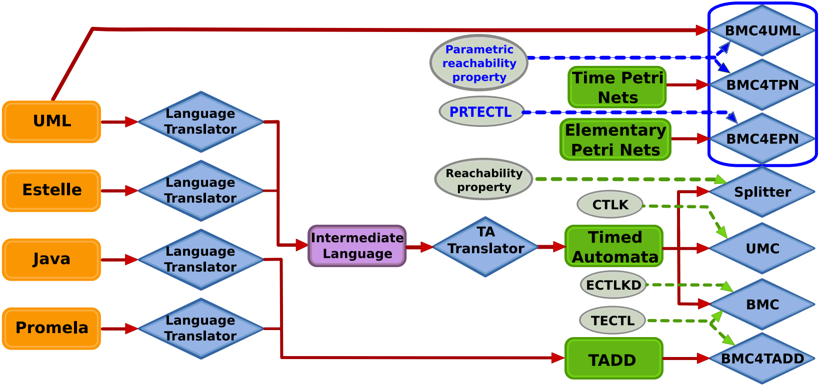
- consider all the executions of the system to a depth k
- encode them in propositional logic
- check the resulting formula using a SAT solver

advantage: no need to construct the model in advance
disadvantage: not complete in general

Experience of our team with BMC and SAT

- adding branching-time *CTL* logic to BMC
- BMC for timed systems
- BMC for epistemic logics
- BMC for cryptographic protocols
- BMC-based verification of Java programs
- BMC-based verification of UML state machines
- Unbounded Model Checking via SAT

VerICS: architecture

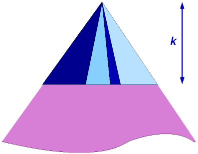


BMC for reachability

k–models

Idea – to unwind the computation tree of a model M up to depth k.

- M – a model, $k \in \mathbb{N}$,
- $Path_k$ – the set of all sequences $(q_0, \ldots, q_k)$, where $q_i \rightarrow q_{i+1}$.
- $M_k = (Path_k, \mathcal{L})$ is called the *k-model*.
- If a propositional formula φ holds in M_k, then φ holds in M.
- The problem $M_k \models \varphi$ is translated to checking satisfiability of the propositional formula $[M_k] \wedge [\varphi]$ using a SAT-solver.



SAT solvers

SAT

- Problem: is a propositional formula satisfiable?
- Theoretical complexity: NP-complete (Cook, 1971)
- Practical and efficient realizations of SAT solvers: only in the last decade
- A general idea: search efficiently for a satisfying assignment

Details

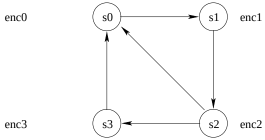
- Efficient data representation
- Heuristics for deducing and learning information
- Frequently efficient in practice
- CNF: conjunctive normal form, conjunction of disjunctions of literals

$$\varphi = (a \vee b \vee \neg c) \wedge (\neg c) \wedge (a \vee \neg b)$$

Symbolic methods

Boolean encoding of the system

Local states



every location $l_i \in L_i$ is represented by the vector $\mathbf{w}_i = (\mathbf{w}_i[1], \ldots, \mathbf{w}_i[l_i])$

$$I_{l_0}(\mathbf{w}) = \neg \mathbf{w}[1] \wedge \neg \mathbf{w}[2]$$

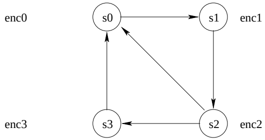
transition relation

$$T(\mathbf{w}, a, \mathbf{v}) \equiv I_{l_3}(\mathbf{w}) \wedge I_{l_0}(\mathbf{v})$$

Symbolic methods

Boolean encoding of the system

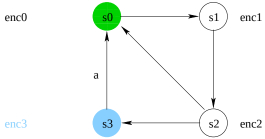
Local states



every location $l_i \in L_i$ is represented by the vector $\mathbf{w}_i = (\mathbf{w}_i[1], \ldots, \mathbf{w}_i[l_i])$

$$I_{l_0}(\mathbf{w}) = \neg\mathbf{w}[1] \wedge \neg\mathbf{w}[2]$$

transition relation



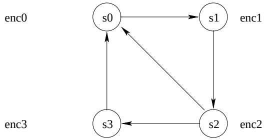
$$T(\mathbf{w}, a, \mathbf{v}) \equiv I_{l_3}(\mathbf{w}) \wedge I_{l_0}(\mathbf{v})$$

Local transition relation: $T(\mathbf{w}_i, \mathbf{v}_i) = \bigvee_{a \in \Sigma_i} T(\mathbf{w}_i, a, \mathbf{v}_i)$

Symbolic methods

Boolean encoding of the system

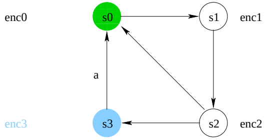
Local states



every location $l_i \in L_i$ is represented by the vector $\mathbf{w}_i = (\mathbf{w}_i[1], \ldots, \mathbf{w}_i[l_i])$

$$I_{l_0}(\mathbf{w}) = \neg \mathbf{w}[1] \wedge \neg \mathbf{w}[2]$$

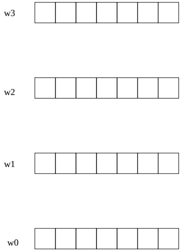
transition relation



$$T(\mathbf{w}, a, \mathbf{v}) \equiv I_{l_3}(\mathbf{w}) \wedge I_{l_0}(\mathbf{v})$$

Local transition relation: $T(\mathbf{w}_i, \mathbf{v}_i) = \bigvee_{a \in \Sigma_i} T(\mathbf{w}_i, a, \mathbf{v}_i)$

BMC - symbolic encoding



- *k-path*: $\mathbf{w}^0, \dots, \mathbf{w}^k$
- k-path is encoded by a propositional formula:

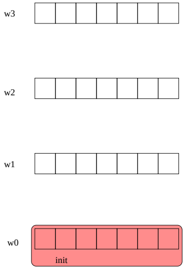
$$path_k(\mathbf{w}^0, \dots, \mathbf{w}^k) = I_\iota(\mathbf{w}^0) \wedge \bigwedge_{i=1}^{k} T(\mathbf{w}^{i-1}, \mathbf{w}^i)$$

$$\varphi_k(\mathbf{w}^0, \dots, \mathbf{w}^k) = path_k \wedge [\varphi](\mathbf{w}^k)$$

BMC

- $k = 0$
- if φ_k is satisfiable - the property is true
- if not, increase k

BMC - symbolic encoding



- *k-path*: $\mathbf{w}^0, \ldots, \mathbf{w}^k$
- k-path is encoded by a propositional formula:

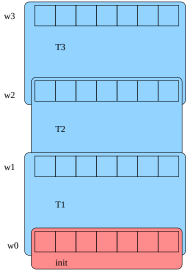
$$path_k(\mathbf{w}^0, \ldots, \mathbf{w}^k) = I_{I^\iota}(\mathbf{w}^0) \wedge \bigwedge_{i=1}^{k} T(\mathbf{w}^{i-1}, \mathbf{w}^i)$$

$$\varphi_k(\mathbf{w}^0, \ldots, \mathbf{w}^k) = path_k \wedge [\varphi](\mathbf{w}^k)$$

BMC

- $k = 0$
- if φ_k is satisfiable - the property is true
- if not, increase k

BMC - symbolic encoding



- *k-path*: $\mathbf{w}^0, \ldots, \mathbf{w}^k$
- k-path is encoded by a propositional formula:

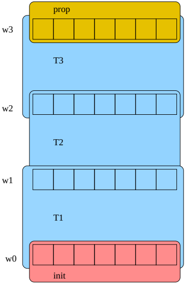
$$path_k(\mathbf{w}^0, \ldots, \mathbf{w}^k) = I_{I^\iota}(\mathbf{w}^0) \wedge \bigwedge_{i=1}^{k} T(\mathbf{w}^{i-1}, \mathbf{w}^i)$$

$$\varphi_k(\mathbf{w}^0, \ldots, \mathbf{w}^k) = path_k \wedge [\varphi](\mathbf{w}^k)$$

BMC

- $k = 0$
- if φ_k is satisfiable - the property is true
- if not, increase k

BMC - symbolic encoding



- *k-path*: $\mathbf{w}^0, \ldots, \mathbf{w}^k$
- k-path is encoded by a propositional formula:

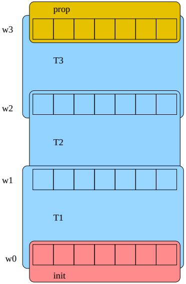
$$path_k(\mathbf{w}^0, \ldots, \mathbf{w}^k) = I_{\iota}(\mathbf{w}^0) \wedge \bigwedge_{i=1}^{k} T(\mathbf{w}^{i-1}, \mathbf{w}^i)$$

$$\varphi_k(\mathbf{w}^0, \ldots, \mathbf{w}^k) = path_k \wedge [\varphi](\mathbf{w}^k)$$

BMC

- $k = 0$
- if φ_k is satisfiable - the property is true
- if not, increase k

BMC - symbolic encoding



- *k-path*: $\mathbf{w}^0, \ldots, \mathbf{w}^k$
- k-path is encoded by a propositional formula:

$$path_k(\mathbf{w}^0, \ldots, \mathbf{w}^k) = I_{\iota}(\mathbf{w}^0) \wedge \bigwedge_{i=1}^{k} T(\mathbf{w}^{i-1}, \mathbf{w}^i)$$

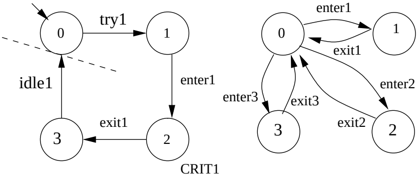
$$\varphi_k(\mathbf{w}^0, \ldots, \mathbf{w}^k) = path_k \wedge [\varphi](\mathbf{w}^k)$$

BMC

- $k = 0$
- if φ_k is satisfiable - the property is true
- if not, increase k

BMC - effectiveness example

Simplified Fischer's mutual exclusion:



Process1 Shared Variable

BMC is effective

the reachable property

$\psi_1 = crit_1,\ k = 2$

n	$\lvert\varphi_k\rvert$	time
3	985	0.002
10	2581	0.006
20	5746	0.02
50	18655	0.39
100	52252	3.62

BMC is not effective, $n = 4$

unreachable mutex property

$\psi_2 = \bigvee_{i,j \in \{1,\dots,n\}, i \neq j} crit_i \wedge crit_j$

k	$\lvert\varphi_k\rvert$	time
3	1429	0.011
6	2689	0.34
9	4369	18.70
12	5209	129
15	6469	> 1000

Outline

Specifying sets of states

- input: pairs of observations (corresponding to a step)
- tool is stateless

- states can be specified not completely:
  - full state specification $\longrightarrow$ a concrete state
  - empty specification $\longrightarrow$ all states
  - missing parts of specifications $\longrightarrow$ a set $Q' \subseteq Q$ of states

Monitoring results: The engine checks at runtime whether the stream of execution steps received as inputs from the RSA, conforms with its symbolic representation of all possible behaviours. For each execution step, the answer returned by the monitoring engine is one of the following facts:

- **GREEN** - the step is conforming with the specification, i.e., there is a contract compliant transition between the source and target states.
- **RED** - a red state is reached as a target of the transition given, i.e., a contract has been violated as a result of the transition.

  Also can signify that the inputs do not comply with the extended format of the TADD for the service.
- **NONE** - the step is not conforming with the specification, i.e., there is no such transition, neither contract compliant or otherwise.
- **ERROR** - the specification given does not mirror the observed transition so it amounts to an error.

From monitoring to model checking

- For a given TADD $\mathcal{A}$ and a pair (Q_1, Q_2) of sets of global states of $\mathcal{S}(\mathcal{A})$, we check whether there are two states $q_1 \in Q_1$ and $q_2 \in Q_2$ such that there is a step from q_1 to q_2.
- If so, we denote the step as $Q_1 \rightsquigarrow Q_2$.

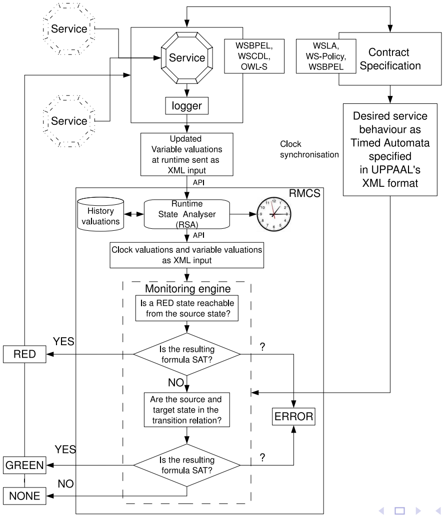
- Encode $T(\mathbf{w}, \mathbf{v})$. Then for each step, this formula is conjuncted with the encodings of a pair of sets of states (Q_1, Q_2) given as an input:
- First make a query about a step from Q_1 to the set of the red states $(Q_1 \rightsquigarrow R \cap Q_2)$: the input $(Q_1, R \cap Q_2)$ is encoded as φ_1. If φ_1 is satisfiable, then "non compliance" is reported.
- If φ_1 is not satisfiable, then the input (Q_1, Q_2) is encoded as φ_2. Depending on its safisfiability, either "compliance" or "invalid transition" is reported

From monitoring to model checking

- For a given TADD $\mathcal{A}$ and a pair (Q_1, Q_2) of sets of global states of $\mathcal{S}(\mathcal{A})$, we check whether there are two states $q_1 \in Q_1$ and $q_2 \in Q_2$ such that there is a step from q_1 to q_2.
- If so, we denote the step as $Q_1 \rightsquigarrow Q_2$.

- Encode $T(\mathbf{w}, \mathbf{v})$. Then for each step, this formula is conjuncted with the encodings of a pair of sets of states (Q_1, Q_2) given as an input:
- First make a query about a step from Q_1 to the set of the red states $(Q_1 \rightsquigarrow R \cap Q_2)$: the input $(Q_1, R \cap Q_2)$ is encoded as φ_1. If φ_1 is satisfiable, then "non compliance" is reported.
- If φ_1 is not satisfiable, then the input (Q_1, Q_2) is encoded as φ_2. Depending on its safisfiability, either "compliance" or "invalid transition" is reported

The scheme of the tool - in more detail

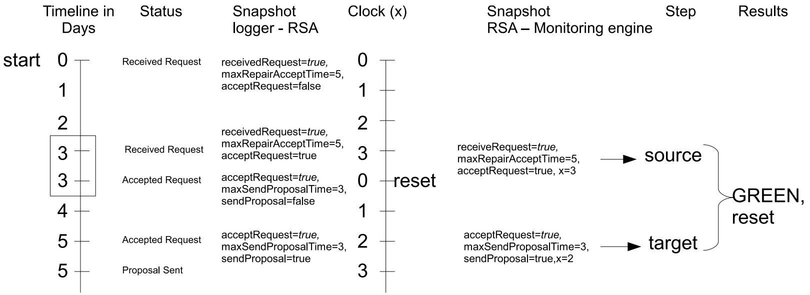


Experimental results

- the implementation is based on Verics BMC
- tool extended with additional constraints
- MiniSAT solver used for testing SAT

Monitoring against a clause

clause	Contract regulated actions	Deadline	Violate	Recover
2	Sends a repair proposal to C	7 days	-	-



Runtime valuations for clause (2)

Results

step	|cars|	|int vars|	|clocks|	Nc/Nvars	time [s]	answer
1	10	10	10	6779/16528	<1	YES
	20	20	20	17738/43455	<1	
	300	300	300	265741/652852	4.3	
3	10	10	10	6743/16431	<1	NO
	30	30	30	26781/65822	<1	
	300	300	300	265811/653052	5.4	

Table: The experimental results. Size of encoding: *Nc*/*Nvars* is the number of clauses/Boolean variables in the result CNF formula; time refers to checking this formula using the tool Minisat.

Future work

short-term

- (distributed) contracts expressed by networks of automata
- more than one step - exploit the power of SAT solvers
- real-world examples
- attaching contracts to running web services

long-term view

- temporal logics
- translating of web services (test all executions)
- ...

Future work

short-term

- (distributed) contracts expressed by networks of automata
- more than one step - exploit the power of SAT solvers
- real-world examples
- attaching contracts to running web services

long-term view

- temporal logics
- translating of web services (test all executions)
- ...