Runtime monitoring of contract regulated web services

Maciej Szreter²

a joint work with Alessio Lomuscio³, Wojciech Penczek¹,² and Monika Solanki³

¹University of Podlasie, Siedlce, Poland
²Polish Academy of Sciences, Warsaw, Poland
³Department of Computing, Imperial College London, UK

Toledo, September 2009
Outline

Contracts and monitoring
  Related work
  The general idea

Contracts
  Repair Car contract
  Introduction to Time Automata
  Timed Automata for specifying contracts

Bounded Model Checking as monitoring engine
  Introduction to Bounded Model Checking
  BMC-based monitoring

Experimental results
Outline

Contracts and monitoring
- Related work
  - The general idea

Contracts
- Repair Car contract
- Introduction to Time Automata
- Timed Automata for specifying contracts

Bounded Model Checking as monitoring engine
- Introduction to Bounded Model Checking
- BMC-based monitoring

Experimental results
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)*
- [NEW] Symbolic model checking approach
  - no need to construct the product automaton
  - model checkers designed for model checking
  - easy to extend
Runtime monitoring of contract regulated web services

Outline

Contracts and monitoring
- Related work
- The general idea

Contracts
- Repair Car contract
- Introduction to Time Automata
- Timed Automata for specifying contracts

Bounded Model Checking as monitoring engine
- Introduction to Bounded Model Checking
- BMC-based monitoring

Experimental results
The general architecture
Monitoring

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

$WS$ possibly distributed, we do not know implementations and specifications
Outline

Contracts and monitoring
Related work
The general idea

Contracts
Repair Car contract
Introduction to Time Automata
Timed Automata for specifying contracts

Bounded Model Checking as monitoring engine
Introduction to Bounded Model Checking
BMC-based monitoring

Experimental results
Repair Car

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

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

Some contract regulated actions for RC

* - (take offline action)
## Explanation

<table>
<thead>
<tr>
<th>Step</th>
<th>state</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>step 1</td>
<td>source</td>
<td>RC waits to receives the request for repairing cars.</td>
</tr>
<tr>
<td></td>
<td>target</td>
<td>RC receives the request for repairing $x$ cars. We show here an example of the clock and variable valuations for three cars.</td>
</tr>
<tr>
<td>step 3</td>
<td>source</td>
<td>RC accepts the request for repairing $x$ cars.</td>
</tr>
<tr>
<td></td>
<td>target</td>
<td>RC sends repair proposals for repairing $x$ cars.</td>
</tr>
</tbody>
</table>

Explanation of trace contents for steps 1 and 3
Set of behaviours for a service
Outline

Contracts and monitoring
  Related work
  The general idea

Contracts
  Repair Car contract

Introduction to Time Automata
  Timed Automata for specifying contracts

Bounded Model Checking as monitoring engine
  Introduction to Bounded Model Checking
  BMC-based monitoring

Experimental results
Networks of automata

Fischer’s mutual exclusion

- $n$ components; $A_i = (L_i, l^i, T_i, \Sigma_i)$
- product automaton (model): $A = A_1 \parallel \ldots \parallel A_n$
- initial state $l^\nu = (l^\nu_1, \ldots, l^\nu_n)$
- set of labels $\Sigma$

Simplified mutual exclusion protocol [Fischer]:
Networks of automata

Fischer’s mutual exclusion

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

Simplified mutual exclusion protocol [Fischer]:

![Diagram of simplified mutual exclusion protocol]

Process 1: 0 (idle1) → 1 (try1) → 2 (enter1) → 3 (exit1) → 0

Process 3: 0 (idle3) → 1 (try3) → 2 (enter3) → 3 (exit3) → 0

Shared Variable: 0 (enter1) → 1 (exit1)
 Networks of automata

Fischer’s mutual exclusion

- $n$ components; $A_i = (L_i, l_i^\ell, T_i, \Sigma_i)$
- product automaton (model): $A = A_1 \parallel \ldots \parallel A_n$
- initial state $I^\ell = (I_1^\ell, \ldots, I_n^\ell)$
- set of labels $\Sigma$

Simplified mutual exclusion protocol [Fischer]:

![Diagram of simplified mutual exclusion protocol]
Networks of automata

Fischer’s mutual exclusion

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

Simplified mutual exclusion protocol [Fischer]:

```
Shared Variable
exit1
enter1
3
enter3 exit3
0 1
.
.
.
10
3 2
idle3
try3
enter3
CRIT3
Process3
kropki
exit3
10
Process1
enter1
3 2
exit1
idle1
try1
CRIT1
```
Networks of automata

Fischer’s mutual exclusion

- $n$ components: $A_i = (L_i, l_i^\nu, T_i, \Sigma_i)$
- product automaton (model): $A = A_1 \parallel \ldots \parallel A_n$
- initial state $l^\nu = (l_1^\nu, \ldots, l_n^\nu)$
- set of labels $\Sigma$
- local transitions

Simplified mutual exclusion protocol [Fischer]:

![Diagram showing the simplified mutual exclusion protocol]
Networks of automata

Fischer’s mutual exclusion

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

Simplified mutual exclusion protocol [Fischer]:

- Process 1
  - States: 0 (idle), 1 (CRIT1), 2 (idle), 3 (CRIT1)
  - Transitions: try1, enter1, exit1

- Process 3
  - States: 0 (idle), 1 (CRIT3), 2 (idle), 3 (CRIT3)
  - Transitions: try3, enter3, exit3

- Shared Variable
  - States: 0 (enter1), 1 (exit1)
  - Transitions: enter1, exit1

- Notation:
  - $l_i$: state of component $i$
  - $T_i$: transition relation of component $i$
  - $\Sigma_i$: set of labels for component $i$
Networks of automata

Fischer’s mutual exclusion

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

- properties expressed in CTL
  mutual exclusion (3 processes):
  $$\varphi = EF((\text{CRIT}_1 \land \text{CRIT}_2) \lor (\text{CRIT}_1 \land \text{CRIT}_3) \lor (\text{CRIT}_2 \land \text{CRIT}_3))$$

Simplified mutual exclusion protocol [Fischer]:

![Diagram of mutual exclusion protocol]
Timed automata enable modeling time flow

- clocks, invariants, guards

Process 1

- try1 \( x_1 \leq \Delta \)
- retry1 \( \{x_1\} \)
- set1 \( x_1 \leq \Delta \) \( \{x_1\} \)
- exit1 \( \{x_1\} \)

Process 2

- try2 \( x_2 \leq \Delta \)
- retry2 \( x_2 \leq \Delta \) \( \{x_2\} \)

Shared Variable

- try1 try2
- retry1 retry2
- exit1 exit2
- set1 set2

models

- concrete model
- detailed regions graph
- abstract graph
Timed automata

enable modeling time flow

- clocks, invariants, guards

Process 1

Process 2

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

![Timed automata diagrams]

- time zones

- models
  - concrete model
  - detailed regions graph
  - abstract graph
models discretized

clocks\(\mathcal{X}\) and variables\(V\),

clock constraints\(C(\mathcal{X}, V)\),

defined by the grammar:

\[
cc ::= \text{true} \mid x_i \sim c \mid x_i \otimes x_j \sim c \mid x_i \otimes x_j \sim v \mid x_i \otimes v \sim c \mid v \otimes w \sim x_i \mid cc \land cc,
\]

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

constraints
\( \mathcal{X} \) - clocks
\( \mathcal{V} \) - integer variables
\( \mathcal{C}(\mathcal{X}, \mathcal{V}) \) - clock constraints over \( \mathcal{X} \) and \( \mathcal{V} \),
defined by the grammar:
\[
\text{cc} ::= \text{true} \mid x_i \sim c \mid x_i \otimes x_j \sim c \mid x_i \otimes x_j \sim v \mid x_i \otimes v \sim c \mid v \otimes w \sim x_i \mid \text{cc} \land \text{cc}, \text{ where}
\]
x\(_i\), x\(_j\) \( \in \mathcal{X} \), v, w \( \in \mathcal{V} \), 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 $A = (\Sigma, L, l^0, V, X, E, I)$, where

- $\Sigma$ 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,
- $X$ is the finite set of clocks,
- $E \subseteq L \times \Sigma \times \text{Bool}(V) \times C(X, V) \times \Sigma(V) \times \text{Asg}(X) \times L$ is a transition relation, and
- $I : L \rightarrow C(X, \emptyset)$ is an invariant function.
The semantics of automata

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

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

Contracts and monitoring
  Related work
  The general idea

Contracts
  Repair Car contract
  Introduction to Time Automata
  Timed Automata for specifying contracts

Bounded Model Checking as monitoring engine
  Introduction to Bounded Model Checking
  BMC-based monitoring

Experimental results
Repair Car contract as TA

<table>
<thead>
<tr>
<th>clause</th>
<th>Contract regulated actions</th>
<th>Deadline</th>
<th>Violation</th>
<th>Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Assess damage to the vehicle</td>
<td>3 days</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

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 \(S(A)\) 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 \(A\) 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 \(S(A)\) 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 \(A\) 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 \(a \in \Sigma\).
Runtime monitoring of contract regulated web services

- Bounded Model Checking as monitoring engine
- Introduction to Bounded Model Checking

Outline

Contracts and monitoring
  Related work
  The general idea

Contracts
  Repair Car contract
  Introduction to Time Automata
  Timed Automata for specifying contracts

Bounded Model Checking as monitoring engine
  Introduction to Bounded Model Checking
  BMC-based monitoring

Experimental results
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 $\varphi$ holds in $M_k$, then $\varphi$ holds in $M$.
- The problem $M_k \models \varphi$ is translated to checking satisfiability of the propositional formula $[M_k] \land [\varphi]$ 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

\[ \varphi = (a \lor b \lor \neg c) \land (\neg c) \land (a \lor \neg b) \]
Symbolic methods
Boolean encoding of the system

Local states

every location \( l_i \in L_i \) is represented by the vector \( w_i = (w_i[1], \ldots, w_i[l_i]) \)

\( l_{l_0}(w) = \neg w[1] \land \neg w[2] \)

\[
T(w, a, v) \equiv l_{l_3}(w) \land l_{l_0}(v)
\]
Symbolic methods
Boolean encoding of the system

Local states

\[ \text{every location } l_i \in L_i \text{ is represented by the vector} \]
\[ w_i = (w_i[1], \ldots, w_i[l_i]) \]
\[ l_{l_0}(w) = \neg w[1] \land \neg w[2] \]

transition relation

\[ T(w, a, v) \equiv l_3(w) \land l_{l_0}(v) \]

Local transition relation: \[ T(w_i, v_i) = \bigvee_{a \in \Sigma_i} T(w_i, a, v_i) \]
Symbolic methods

Boolean encoding of the system

Local states

\[
\begin{align*}
&\text{enc0} & s0 & s1 & \text{enc1} \\
&\text{enc3} & s3 & s2 & \text{enc2}
\end{align*}
\]

every location \( l_i \in L_i \) is represented by the vector \( w_i = (w_i[1], \ldots, w_i[l_i]) \)

\[
l_{l_0}(w) = \neg w[1] \land \neg w[2]
\]

transition relation

\[
\begin{align*}
&\text{enc0} & \text{stf} & s1 & \text{enc1} \\
&\text{enc3} & s3 & s2 & \text{enc2}
\end{align*}
\]

\[
T(w, a, v) \equiv l_{l_3}(w) \land l_{l_0}(v)
\]

Local transition relation: \( T(w_i, v_i) = \bigvee_{a \in \Sigma_i} T(w_i, a, v_i) \)
BMC - symbolic encoding

▶ $k$-path: $w^0, \ldots, w^k$

▶ $k$-path is encoded by a propositional formula:

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

$$\varphi_k(w^0, \ldots, w^k) = path_k \land [\varphi](w^k)$$

BMC

▶ $k = 0$

▶ if $\varphi_k$ is satisfiable - the property is true

▶ if not, increase $k$
BMC - symbolic encoding

- **k-path**: \( w^0, \ldots, w^k \)
- **k-path** is encoded by a propositional formula:

\[
\varphi_k(w^0, \ldots, w^k) = path_k \land [\varphi](w^k)
\]

BMC

- \( k = 0 \)
- if \( \varphi_k \) is satisfiable - the property is true
- if not, increase \( k \)
Runtime monitoring of contract regulated web services

Bounded Model Checking as monitoring engine

Introduction to Bounded Model Checking

BMC - symbolic encoding

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

$$\text{path}_k(w^0, \ldots, w^k) = I_{\mu}(w^0) \land \bigwedge_{i=1}^{k} T(w^{i-1}, w^i)$$

$$\varphi_k(w^0, \ldots, w^k) = \text{path}_k \land [\varphi](w^k)$$

BMC

- $k = 0$
- if $\varphi_k$ is satisfiable - the property is true
- if not, increase $k$
BMC - symbolic encoding

- **k-path**: $w^0, \ldots, w^k$

- $k$-path is encoded by a propositional formula:

  $$\text{path}_k(w^0, \ldots, w^k) = l_0(w^0) \wedge \bigwedge_{i=1}^{k} T(w^{i-1}, w^i)$$

  $$\varphi_k(w^0, \ldots, w^k) = \text{path}_k \wedge [\varphi](w^k)$$

BMC

- $k = 0$

- if $\varphi_k$ is satisfiable - the property is true

- if not, increase $k$
BMC - symbolic encoding

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

$$path_k(w^0, \ldots, w^k) = \mu(w^0) \land \bigwedge_{i=1}^{k} T(w^{i-1}, w^i)$$

$$\varphi_k(w^0, \ldots, w^k) = path_k \land [\varphi](w^k)$$

BMC

- $k = 0$
- if $\varphi_k$ is satisfiable - the property is true
- if not, increase $k$
BMC - effectiveness example

Simplified Fischer’s mutual exclusion:

BMC is effective

the reachable property

\[ \psi_1 = crit_1, \quad k = 2 \]

| n  | \( |\varphi_k| \) | 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

\[ \psi_2 = \bigvee_{i,j \in \{1, \ldots, n\}, i \neq j} crit_i \land crit_j \]

| k  | \( |\varphi_k| \) | time  |
|-----|-----------------|-------|
| 3   | 1429            | 0.011 |
| 6   | 2689            | 0.34  |
| 9   | 4369            | 18.70 |
| 12  | 5209            | 129   |
| 15  | 6460 \( \geq 1000 \) |       |
Outline

Contracts and monitoring
  Related work
  The general idea

Contracts
  Repair Car contract
  Introduction to Time Automata
  Timed Automata for specifying contracts

Bounded Model Checking as monitoring engine
  Introduction to Bounded Model Checking
  BMC-based monitoring

Experimental results
Specifying sets of states

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

- states can be specified not completely:
  - full state specification → a concrete state
  - empty specification → all states
  - missing parts of specifications → 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 $S(\mathcal{A})$, 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 $Q_1 \xrightarrow{} Q_2$.

- Encode $T(w, v)$. 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 $(Q_1 \xrightarrow{} R \cap Q_2)$: the input $(Q_1, R \cap Q_2)$ is encoded as $\varphi_1$. If $\varphi_1$ is satisfiable, then “non compliance” is reported.
  - If $\varphi_1$ is not satisfiable, then the input $(Q_1, Q_2)$ is encoded as $\varphi_2$. Depending on its satisfiability, either “compliance” or “invalid transition” is reported.
From monitoring to model checking

- For a given TADD $A$ and a pair $(Q_1, Q_2)$ of sets of global states of $S(A)$, 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 $T(w, v)$. 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 $\varphi_1$. If $\varphi_1$ is satisfiable, then “non compliance” is reported.

  - If $\varphi_1$ is not satisfiable, then the input $(Q_1, Q_2)$ is encoded as $\varphi_2$. Depending on its satisfiability, 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
Runtime monitoring of contract regulated web services

Experimental results

Monitoring against a clause

<table>
<thead>
<tr>
<th>clause</th>
<th>Contract regulated actions</th>
<th>Deadline</th>
<th>Violate</th>
<th>Recover</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Sends a repair proposal to C</td>
<td>7 days</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Timeline in Days | Status               | Snapshot logger - RSA | Clock (x) | Snapshot RSA – Monitoring engine | Step | Results |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>start 0</td>
<td>Received Request</td>
<td>receivedRequest=true, maxRepairAcceptTime=5, acceptRequest=false</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Received Request</td>
<td>receivedRequest=true, maxRepairAcceptTime=5, acceptRequest=true</td>
<td>2</td>
<td>reset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Accepted Request</td>
<td>acceptRequest=true, maxSendProposalTime=3, sendProposal=false</td>
<td>3</td>
<td>receiveRequest=true, maxRepairAcceptTime=5, acceptRequest=true, x=3</td>
<td>source</td>
<td>GREEN, reset</td>
</tr>
<tr>
<td>4</td>
<td>Accepted Request</td>
<td>acceptRequest=true, maxSendProposalTime=3, sendProposal=true</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Proposal Sent</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>step</th>
<th>cars</th>
<th>int vars</th>
<th>clocks</th>
<th>( \frac{N_c}{N_{vars}} )</th>
<th>time [s]</th>
<th>answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>( 6779/16528 )</td>
<td>&lt;1</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>( 17738/43455 )</td>
<td>&lt;1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>( 265741/652852 )</td>
<td>4.3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>( 6743/16431 )</td>
<td>&lt;1</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>( 26781/65822 )</td>
<td>&lt;1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>( 265811/653052 )</td>
<td>5.4</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** The experimental results. Size of encoding: \( \frac{N_c}{N_{vars}} \) 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)
- ...

Runtime monitoring of contract regulated web services

Experimental results
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)
▶ ...

Runtime monitoring of contract regulated web services

Experimental results