Killing Bugs in a Black Box with Model-based Mutation Testing

Bernhard K. Aichernig

Institute of Software Technology
Graz University of Technology, Austria

MT CPS Workshop
Vienna, 11 Apr 2016

Acknowledgements

Joint work with

J. Auer · H. Brandl · W. Herzner · E. Jöbstl · W. Krenn · R. Korosec ·
F. Lorber · D. Nickovic · A. Rosenmann · R. Schlick · B.V. Schmidt ·
M. Tappler · S. Tiran

Strong Collaboration:
Since 2008 with AIT
Since 2011 with AVL

Projects

Past:

- **CREDO:** FP6, MBT of distributed systems
- **MOGENTES:** FP7, MBT of embedded systems, mutation testing, qualitative reasoning for testing hybrid systems
- **TRUFAL:** national, scalability of test-case generators via symbolic analysis
- **MBAT:** FP7, integration of methods and tools, MBT + consistency checking

Ongoing:

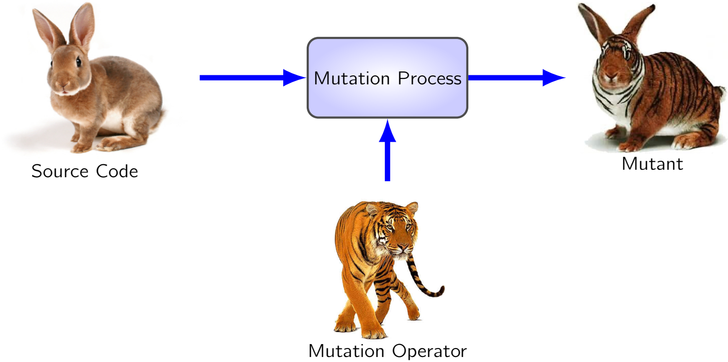
- **CRYSTAL:** FP7, integration of tools, MBT + requirements engineering
- **TRUCONF:** national, MBT + non-functional requirements + systems of systems

Agenda

- Model-based Mutation Testing
- Real-Time Systems
- Hybrid Systems
- Discrete Systems

Mutation Testing I

Step 1: Create mutants



Mutation Testing II

Step 2: Try to kill mutants

A test case kills a mutant if its run shows different behaviour.

Quality of tests:
How many mutants survived? [Lipton71, Hamlet77, DeMillo et al.78]

Objective

Don’t write test cases,

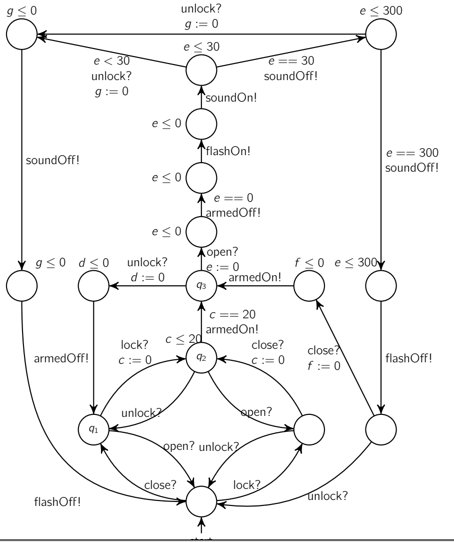
generate them!

Objective

Don’t write test cases,

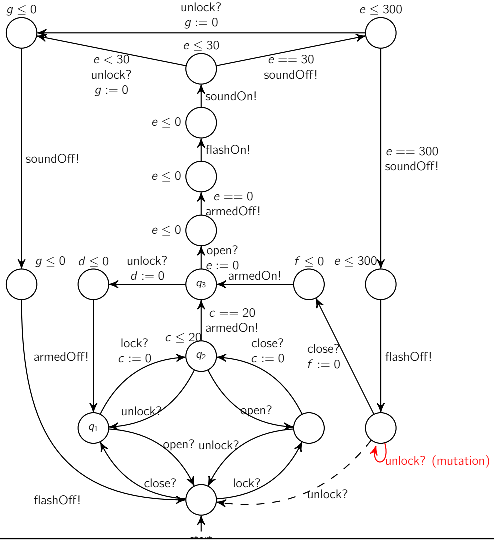
generate them!

Timed Automata Model of a Car Alarm System



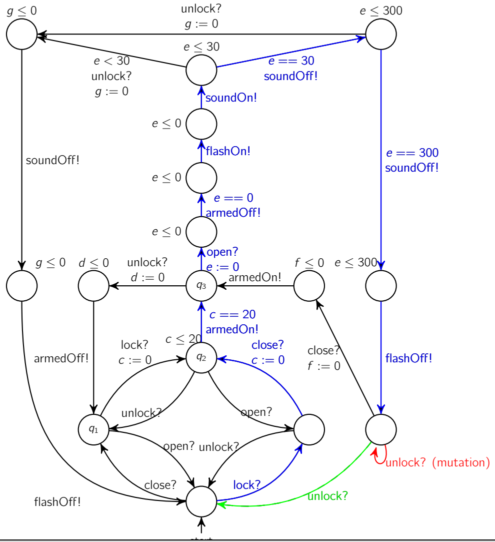
- Car alarm system model
- and a mutation representing a fault
- leading to non-conformance representing an observable failure
- resulting in a test case triggering this fault
- and propagating it to a visible failure

Timed Automata Model of a Car Alarm System



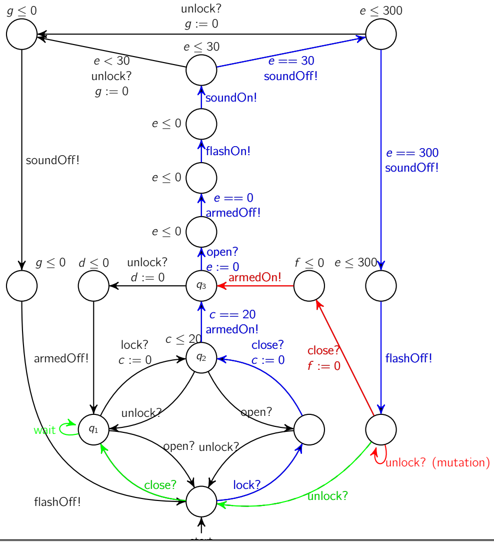
- Car alarm system model
- and a mutation representing a fault
- leading to non-conformance representing an observable failure
- resulting in a test case triggering this fault
- and propagating it to a visible failure

Timed Automata Model of a Car Alarm System



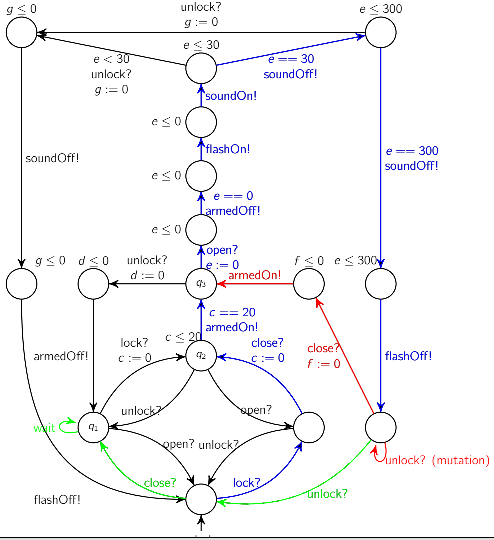
- Car alarm system model
- and a mutation representing a fault
- leading to non-conformance representing an observable failure
- resulting in a test case triggering this fault
- and propagating it to a visible failure

Timed Automata Model of a Car Alarm System



- Car alarm system model
- and a mutation representing a fault
- leading to non-conformance representing an observable failure
- resulting in a test case triggering this fault
- and propagating it to a visible failure

Timed Automata Model of a Car Alarm System

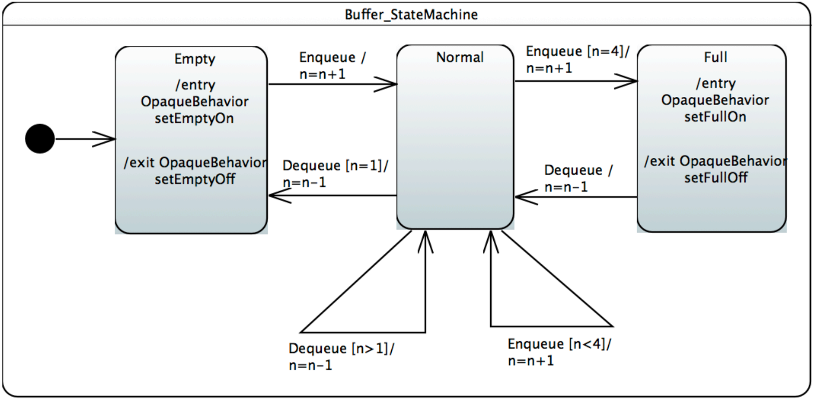


- Car alarm system model
- and a mutation representing a fault
- leading to non-conformance representing an observable failure
- resulting in a test case triggering this fault
- and propagating it to a visible failure

What is a failure?

Fault-Propagation in Models

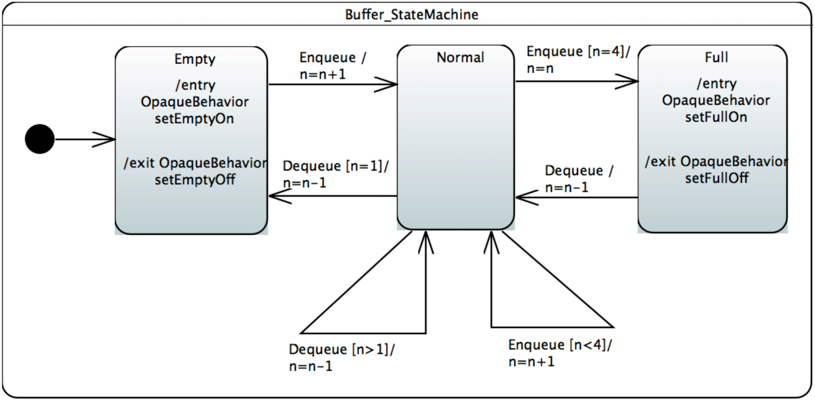
Abstract 5-place buffer model:



Counter variable n is internal!

Fault-Propagation in Models

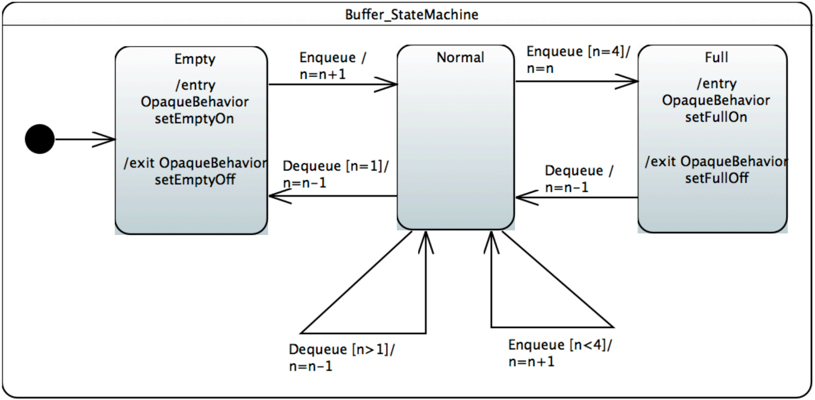
Let's inject a fault:



How does this fault propagate?

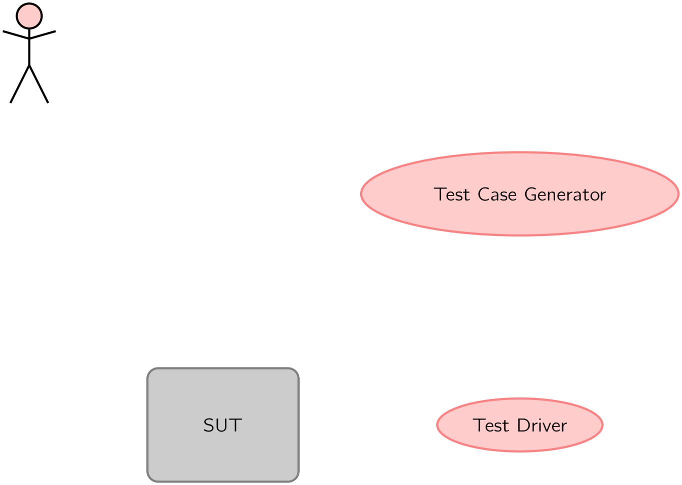
A Good Test Case

... triggers this fault and propagates it to a (visible) failure:

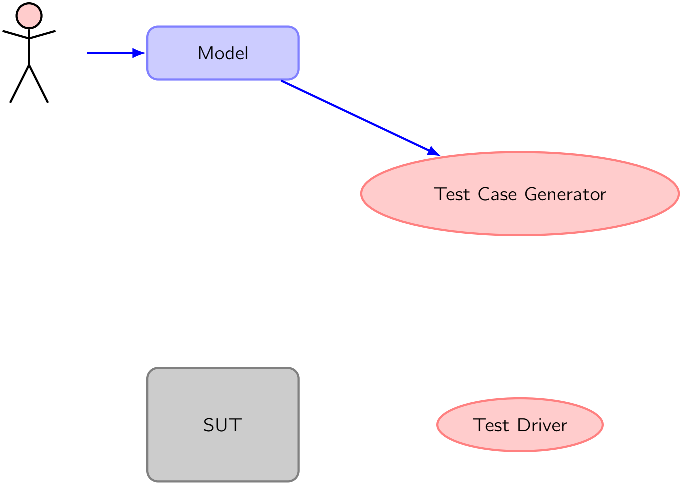


⟨!setEmptyOn, ?Enqueue, !setEmptyOff, ?Enqueue, ?Enqueue, ?Enqueue, ?Enqueue, !setFullOn, ?Dequeue, !setFullOff, ?Enqueue, !setFullOn⟩

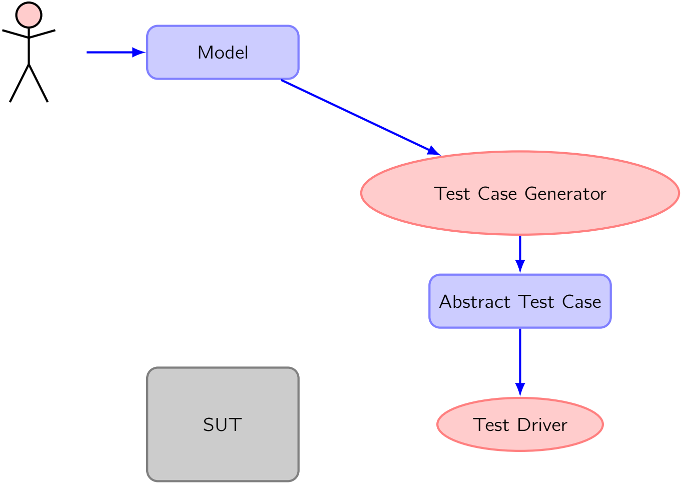
Model-Based Testing



Model-Based Testing



Model-Based Testing



Model

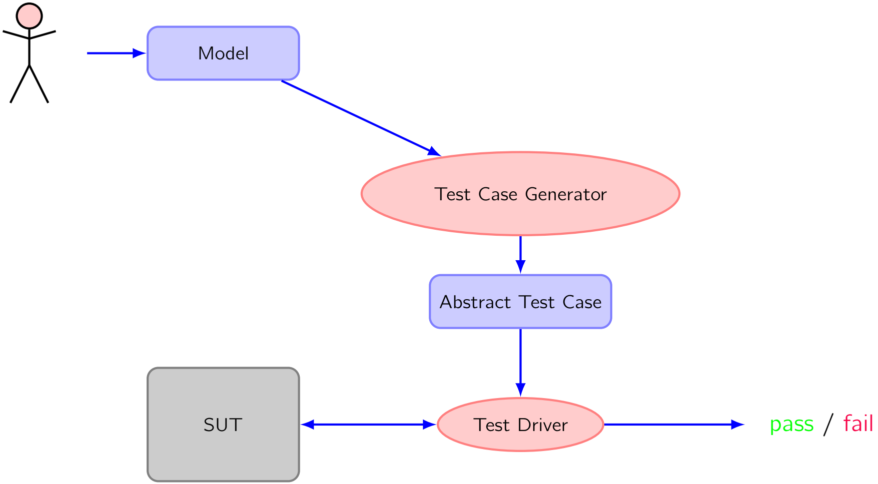
Test Case Generator

Abstract Test Case

SUT

Test Driver

Model-Based Testing



Model

Test Case Generator

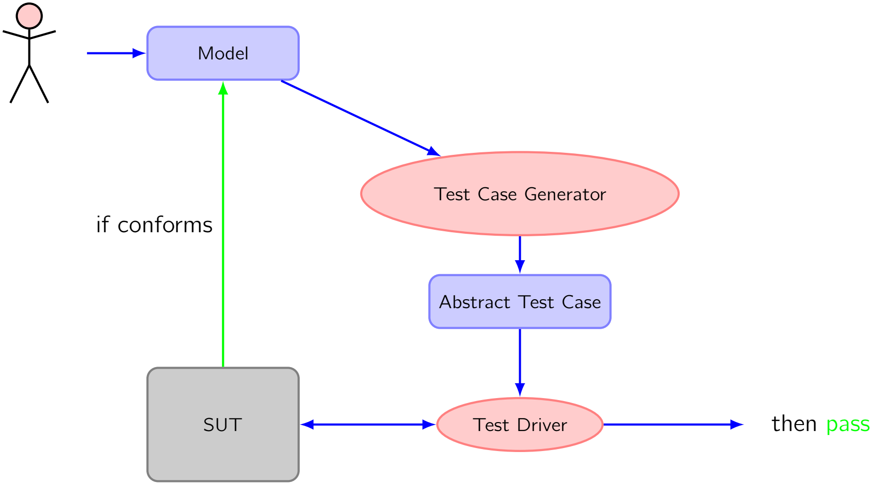
Abstract Test Case

SUT

Test Driver

pass / fail

Model-Based Testing



Model

Test Case Generator

Abstract Test Case

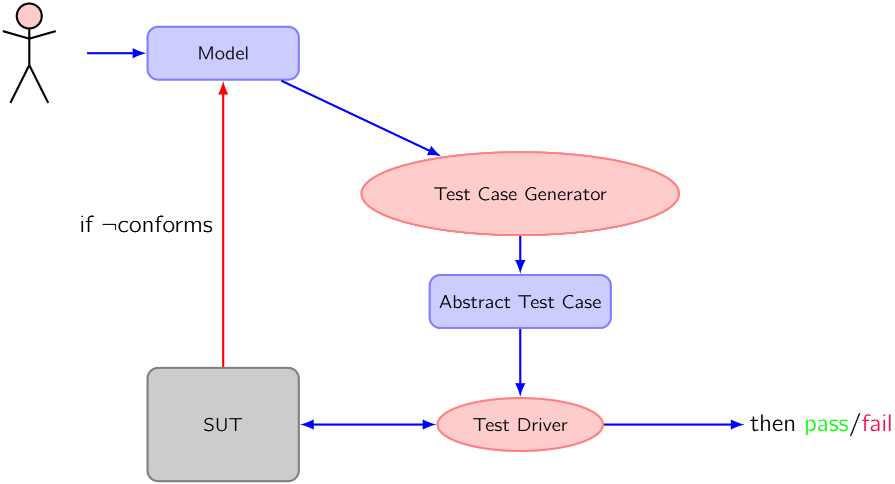
if conforms

SUT

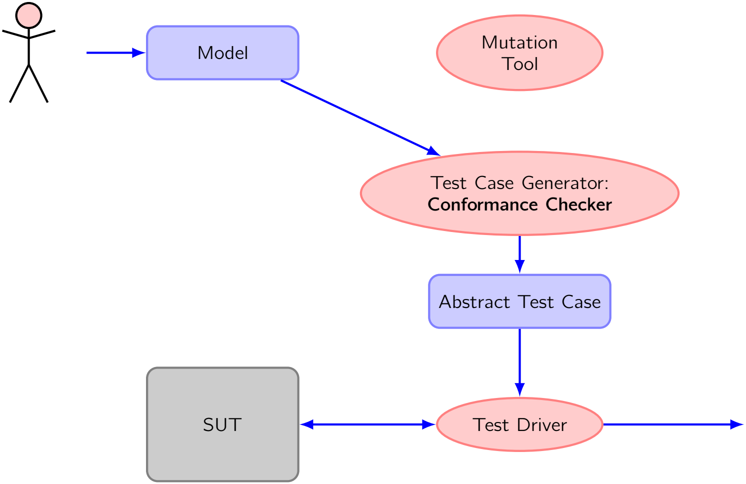
Test Driver

then pass

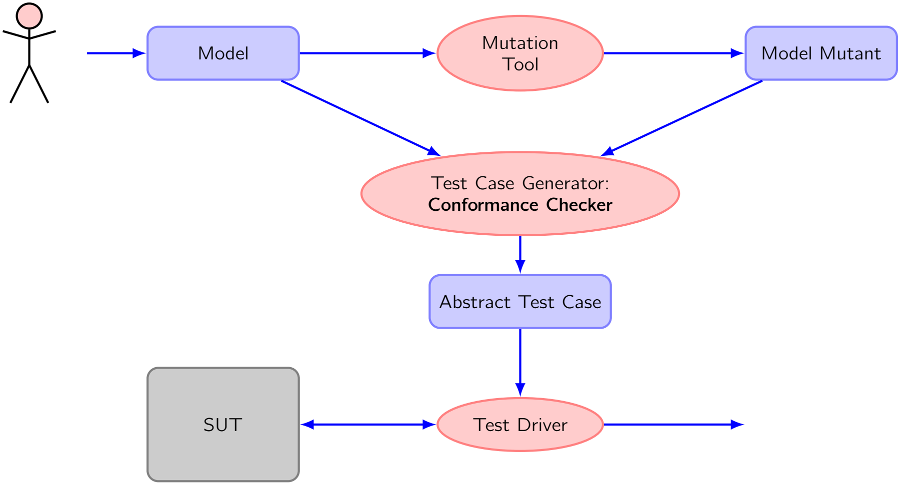
Model-Based Testing



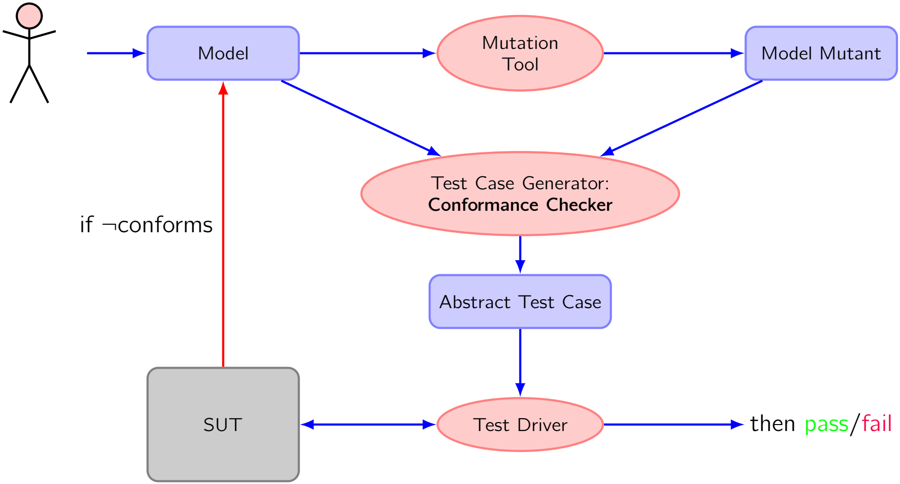
Model-Based Mutation Testing



Model-Based Mutation Testing



Model-Based Mutation Testing



Model

Mutation Tool

Model Mutant

Test Case Generator:
Conformance Checker

Abstract Test Case

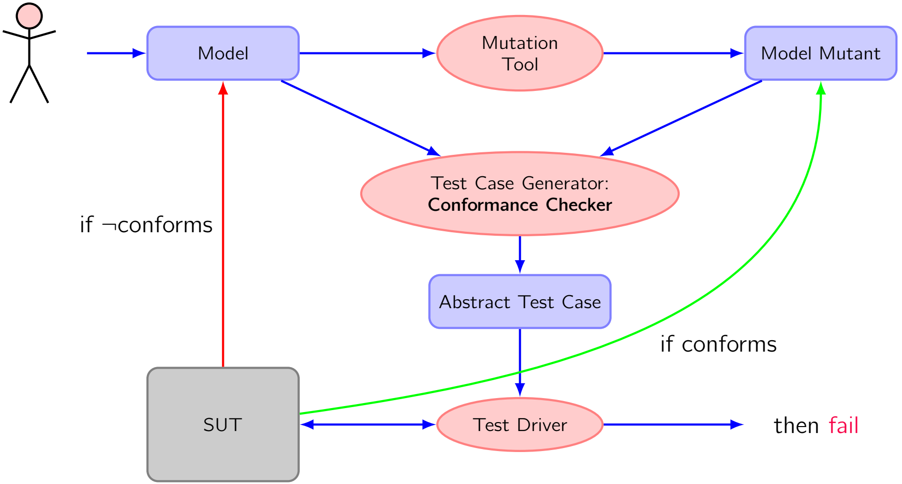
if ¬conforms

SUT

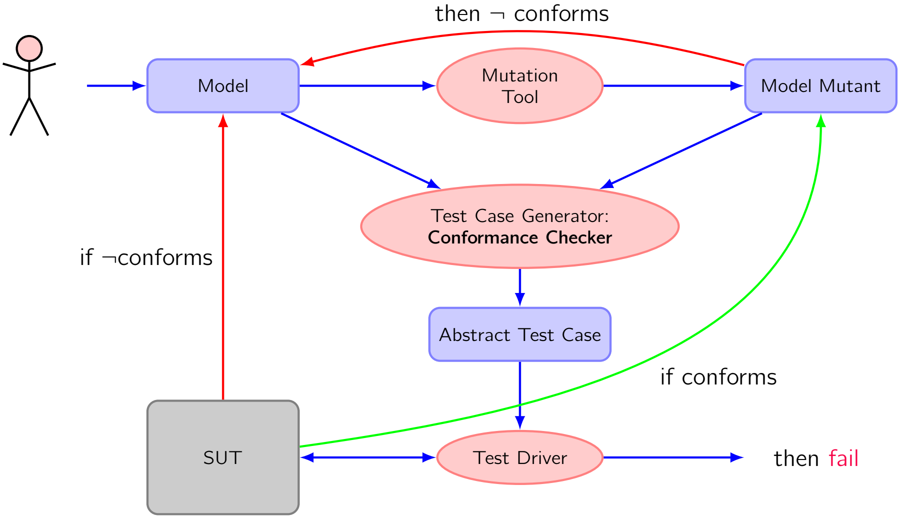
Test Driver

then pass/fail

Model-Based Mutation Testing



Model-Based Mutation Testing



MoMuT Tools

MoMuT

- is a family of tools implementing Model-based Mutation Testing.
- is jointly developed and maintained by AIT and TU Graz
- supports different modelling styles:
  - MoMuT::UML (UML state machines)
  - MoMuT::OOAS (OO Action Systems)
  - MoMuT::QAS (Qualitative Action Systems)
  - MoMuT::TA (Timed Automata)
  - MoMuT::TAS (Timed Action Systems)
  - MoMuT::REQs (Synchronous Requirement Interfaces)

www.momut.org

Agenda

- Model-based Mutation Testing
- Real-Time Systems
- Hybrid Systems
- Discrete Systems

Conformance Relation of Timed Systems

… defines in a testing theory what constitutes a failure.

Definition (Timed input-output conformance – tioco [Krichen&Tripakis09])

Given a timed automaton Model and a Mutant with inputs and outputs

$$\mathit{Mutant}\ \mathrm{tioco}\ \mathit{Model}\ \text{iff}$$

$$\forall \sigma \in L(\mathit{Model}) : \mathrm{out}(\mathit{Mutant}\ \mathrm{after}\ \sigma) \subseteq \mathrm{out}(\mathit{Model}\ \mathrm{after}\ \sigma)$$

S	…	set of all states
s_0	…	initial state
σ	…	timed trace of labels
Σ_O	…	output labels

$$\begin{aligned}
A\ \mathrm{after}\ \sigma &= \{s \in S \mid s_0 \xrightarrow{\sigma} s\} \\
\mathrm{elapse(s)} &= \{t > 0 \mid s \xrightarrow{t}\} \\
\mathrm{out}(s) &= \{a \in \Sigma_O \mid s \xrightarrow{a}\} \cup \mathrm{elapse(s)} \\
\mathrm{out}(S) &= \textstyle\bigcup_{s \in S} \mathrm{out}(s)
\end{aligned}$$

tioco and Language Inclusion

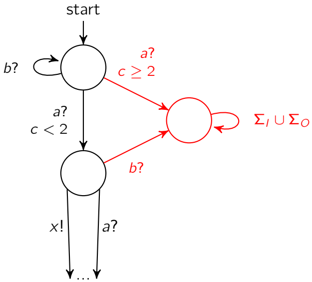
Theorem ([Krichen&Tripakis09])

$L(Mutant) \subseteq L(Model) \Rightarrow\ Mutant\ \mathit{tioco}\ Model$

Theorem ([Krichen&Tripakis09])

If Model is **input-enabled**, *then*

$Mutant\ \mathit{tioco}\ Model\ \ \Rightarrow\ \ L(Mutant) \subseteq L(Model)$



Demonic completion for deterministic TA

For deterministic TA,
reduce tioco check to language inclusion check (PSPACE-complete).

k-Bounded Language Inclusion

- Construct a formula $\varphi^k_{A_I,A_S}$ that is satisfiable if $L(A_I) \not\subseteq L(A_S)$
  - providing a timed trace as witness

$$
\begin{array}{lll}
\varphi^k_{A_I,A_S} & \equiv & \\
& \bigwedge_{i=1}^{k}(d^i \geq 0 \wedge 1 \leq \alpha^i \leq |\Sigma|) \wedge i \geq 1 \wedge i \leq k & \wedge \quad \text{(delays and actions)} \\
& 1 \leq i \leq k & \wedge \quad \text{(in i steps)} \\
& \mathrm{init}_{A_I}(X_I, C_I) \wedge \mathrm{path}^{1,i-1}_{A_I}(\mathcal{A}, D, X_I, C_I) & \wedge \quad \text{(reach in mutant)} \\
& \mathrm{init}_{A_S}(X_S, C_S) \wedge \mathrm{path}^{1,i-1}_{A_S}(\mathcal{A}, D, X_S, C_S) & \wedge \quad \text{(reach in model)} \\
& \mathrm{path}^{i,i}_{A_I}(\mathcal{A}, D, X_I, C_I) \wedge \neg\mathrm{path}^{i,i}_{A_S}(\mathcal{A}, D, X_S, C_S) & \quad\;\; \text{(failure)}
\end{array}
$$

Variable sets:
$x^i \in X$ … location at step i
$\alpha^i \in \mathcal{A}$ … i^{th} discrete action
$d^i \in D$ … i^{th} time delay
$\{c^i, c^{*,i}\} \subseteq C$ … clock valuation after i^{th} time and discrete step

Experimental Results I

- Bounded language inclusion check for deterministic Uppaal TA
- Implemented in Scala calling SMT solver Z3
- Car alarm system characteristics: deterministic,
    - 5 clock variables, 16 locations, 25 transitions.
- 8 mutation operators → 1,320 mutants
- Overall runtime: 30 minutes ($k = 12$)

Depth	Bounded Model Checking				Symbolic Execution			
	Mean	Median	Max	Min	Mean	Median	Max	Min
12	$1.4s$	$1.1s$	$33s$	$0.07s$				

Runtime details

Experimental Results I

- Bounded language inclusion check for deterministic Uppaal TA
- Implemented in Scala calling SMT solver Z3
- Car alarm system characteristics: deterministic,
    - 5 clock variables, 16 locations, 25 transitions.
- 8 mutation operators → 1,320 mutants
- Overall runtime: 30 minutes ($k = 12$)

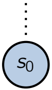
Depth	Bounded Model Checking				Symbolic Execution			
	Mean	Median	Max	Min	Mean	Median	Max	Min
12	$1.4s$	$1.1s$	$33s$	$0.07s$				

Runtime details

Timed Action Systems

```
types{
  State = [ ... | Flash | FlashSound | Silent | SwitchOffAlarm | ... ]; }
state{
  loc : State; }
clocks [Real]{ c;d;e;f;g }
init {
  loc := OpenAndUnlocked;}
invariant {
  if loc == Flash      then e <= 0;
  if loc == FlashSound then e <= 30;
  if loc == Silent     then e <= 300;
 ... }
actions{
  !soundOn#1() if loc == Flash && e == 0 then { loc := FlashSound; };

  !soundOff#1() if loc == FlashSound && e == 30 then { loc := Silent ; };

  ?unlock#6() resets g if loc == FlashSound && e < 30 then { loc := SwitchOffAlarm; };
 ... }
```

Symbolic Execution of Timed Action Systems



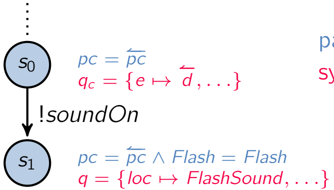
$pc = \overleftarrow{pc}$

$q_c = \{e \mapsto \overleftarrow{d}, \dots\}$

path conditions . . . blue

symbolic (clock) states . . . red

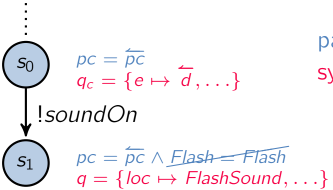
Symbolic Execution of Timed Action Systems



path conditions . . . blue

symbolic (clock) states . . . red

Symbolic Execution of Timed Action Systems



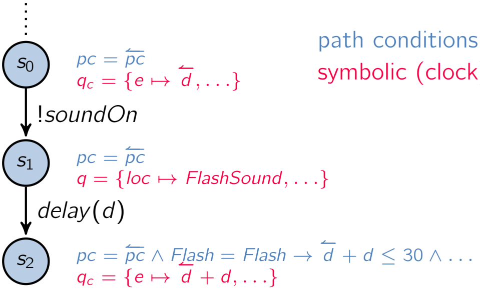
$pc = \overleftarrow{pc}$
$q_c = \{e \mapsto \overleftarrow{d}, \ldots\}$

$pc = \overleftarrow{pc} \wedge \cancel{Flash = Flash}$
$q = \{loc \mapsto FlashSound, \ldots\}$

path conditions . . . blue

symbolic (clock) states . . . red

Symbolic Execution of Timed Action Systems



$pc = \overleftarrow{pc}$
$q_c = \{e \mapsto \overleftarrow{d}, \ldots\}$

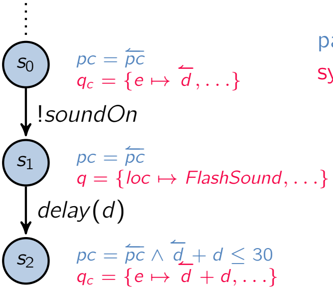
$pc = \overleftarrow{pc}$
$q = \{loc \mapsto FlashSound, \ldots\}$

$pc = \overleftarrow{pc} \wedge Flash = Flash \rightarrow \overleftarrow{d} + d \leq 30 \wedge \ldots$
$q_c = \{e \mapsto \overleftarrow{d} + d, \ldots\}$

path conditions . . . blue

symbolic (clock) states . . . red

Symbolic Execution of Timed Action Systems



$pc = \overleftarrow{pc}$
$q_c = \{e \mapsto \overleftarrow{d}, \ldots\}$

$pc = \overleftarrow{pc}$
$q = \{loc \mapsto FlashSound, \ldots\}$

$pc = \overleftarrow{pc} \wedge \overleftarrow{d} + d \leq 30$
$q_c = \{e \mapsto \overleftarrow{d} + d, \ldots\}$

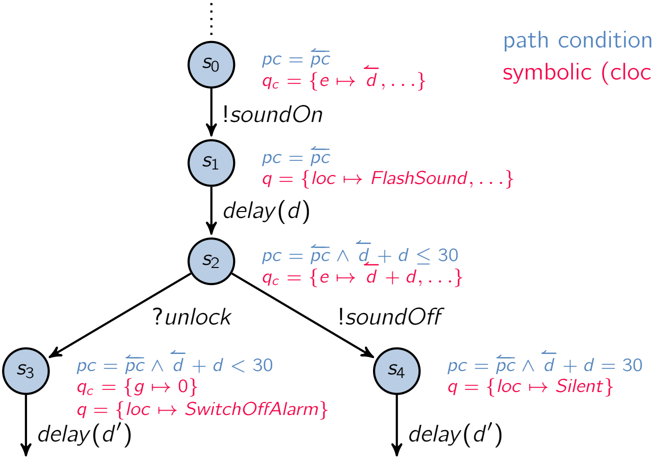
path conditions . . . blue

symbolic (clock) states . . . red

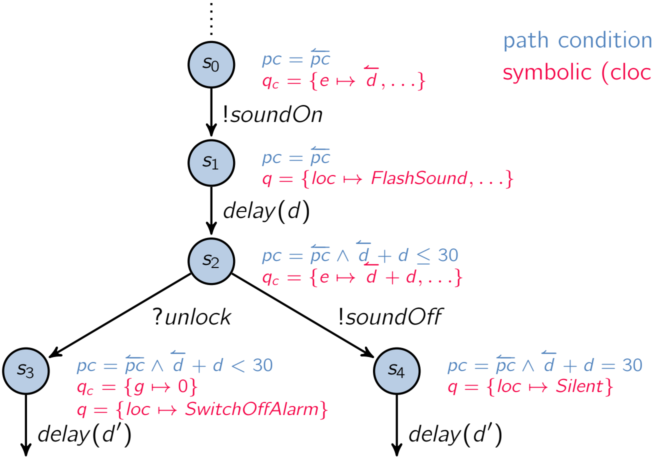
Symbolic Execution of Timed Action Systems

path conditions . . . blue

symbolic (clock) states . . . red



Symbolic Execution of Timed Action Systems



path conditions . . . blue
symbolic (clock) states . . . red

s_0
$pc = \overleftarrow{pc}$
$q_c = \{e \mapsto \overleftarrow{d}, \ldots\}$

$!soundOn$

s_1
$pc = \overleftarrow{pc}$
$q = \{loc \mapsto FlashSound, \ldots\}$

$delay(d)$

s_2
$pc = \overleftarrow{pc} \wedge \overleftarrow{d} + d \leq 30$
$q_c = \{e \mapsto \overleftarrow{d} + d, \ldots\}$

$?unlock$

$!soundOff$

s_3
$pc = \overleftarrow{pc} \wedge \overleftarrow{d} + d < 30$
$q_c = \{g \mapsto 0\}$
$q = \{loc \mapsto SwitchOffAlarm\}$

$delay(d')$

s_4
$pc = \overleftarrow{pc} \wedge \overleftarrow{d} + d = 30$
$q = \{loc \mapsto Silent\}$

$delay(d')$

Provides all symbolic timed traces through model!

Conformance Checking via Symbolic Execution

- Bounded implicit product graph exploration
- Simultaneous symbolic execution of all model traces
- Non-conformance checks (stioco) of the form:

pc_q . . . path condition of symbolic state q

Conformance Checking via Symbolic Execution

- Bounded implicit product graph exploration
- Simultaneous symbolic execution of all model traces
- Non-conformance checks (stioco) of the form:

$$\exists q_{fail} \in \underbrace{ModelStates}_{\text{all symbolic states after current trace}}$$

$$\underbrace{pc_{q_{fail}}}_{\text{state reachable (model)}}$$

pc_q . . . path condition of symbolic state q

Conformance Checking via Symbolic Execution

- Bounded implicit product graph exploration
- Simultaneous symbolic execution of all model traces
- Non-conformance checks (stioco) of the form:

$$\exists q_{fail} \in \underbrace{ModelStates}_{\text{all symbolic states after current trace}}, \exists \lambda \in Observations :$$

$$\underbrace{pc_{q_{fail}}}_{\text{state reachable (model)}} \wedge \underbrace{\left(\bigvee_{s \in MutantStates} pc_s \wedge guards_\lambda[state_s] \right)}_{\text{observation possible (mutant)}} \wedge$$

$$\underbrace{\neg \left(\bigvee_{q \in ModelStates} pc_q \wedge guards_\lambda[state_q] \right)}_{\text{observation not possible (model)}}$$

pc_q . . . path condition of symbolic state q

Experimental Results II

- Symbolic execution tioco check for deterministic Timed Action Systems
- Implemented in Scala calling SMT solver Z3
- Car alarm system characteristics: deterministic,
  - 5 clock variables, 16 locations, 25 transitions.
- 8 mutation operators → 986 mutants
- Overall runtime: 27.5 minutes ($k = 12$)

Depth	Bounded Model Checking				Symbolic Execution			
	Mean	Median	Max	Min	Mean	Median	Max	Min
12	$1.4s$	$1.1s$	$33s$	$0.07s$	$1.7s$	$0.02s$	$38.83s$	$\sim 0s$

Runtime details

Experimental Results II

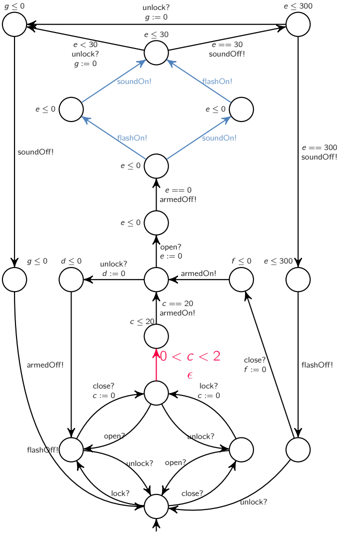
- Symbolic execution tioco check for deterministic Timed Action Systems
- Implemented in Scala calling SMT solver Z3
- Car alarm system characteristics: deterministic,
  - 5 clock variables, 16 locations, 25 transitions.
- 8 mutation operators → 986 mutants
- Overall runtime: 27.5 minutes ($k = 12$)

Depth	Bounded Model Checking				Symbolic Execution			
	Mean	Median	Max	Min	Mean	Median	Max	Min
12	$1.4s$	$1.1s$	$33s$	$0.07s$	$1.7s$	$0.02s$	$38.83s$	$\sim 0s$

Runtime details

Experimental Results III

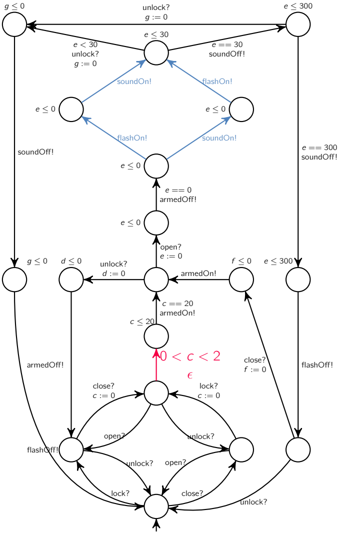
- Symbolic tioco checker also for non-deterministic models
- Car Alarm System: silent transition with non-deterministic delay
- Plus underspecification in switching on alarm
- 3 equivalent mutants timed out after 10min



Experimental Results III

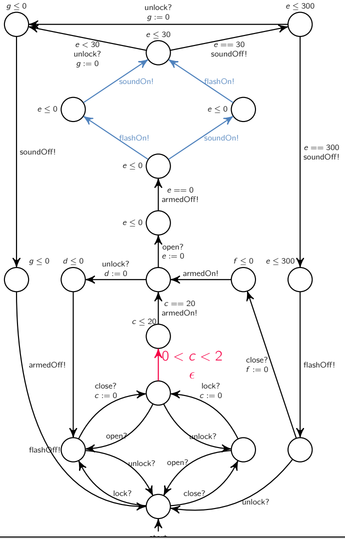
- Symbolic tioco checker also for non-deterministic models
- Car Alarm System: silent transition with non-deterministic delay
- Plus underspecification in switching on alarm
- 3 equivalent mutants timed out after 10min

Depth	Symbolic Execution			
	Mean	Median	Max	Min
12	$0.79s$	$0.06s$	$360.84s$	$\sim 0s$



Experimental Results III

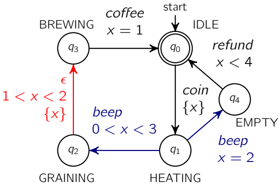
- Symbolic tioco checker also for non-deterministic models
- Car Alarm System: silent transition with non-deterministic delay
- Plus underspecification in switching on alarm
- 3 equivalent mutants timed out after 10min



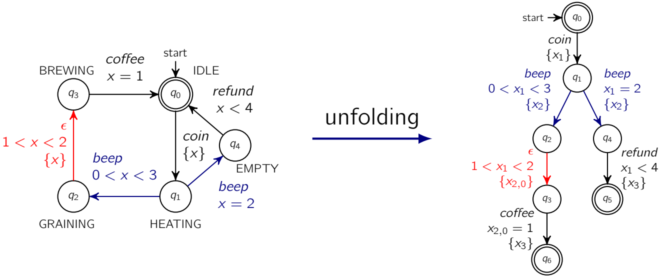
Depth	Symbolic Execution			
	Mean	Median	Max	Min
12	$0.79s$	$0.06s$	$360.84s$	$\sim 0s$

… and the bounded model checking?

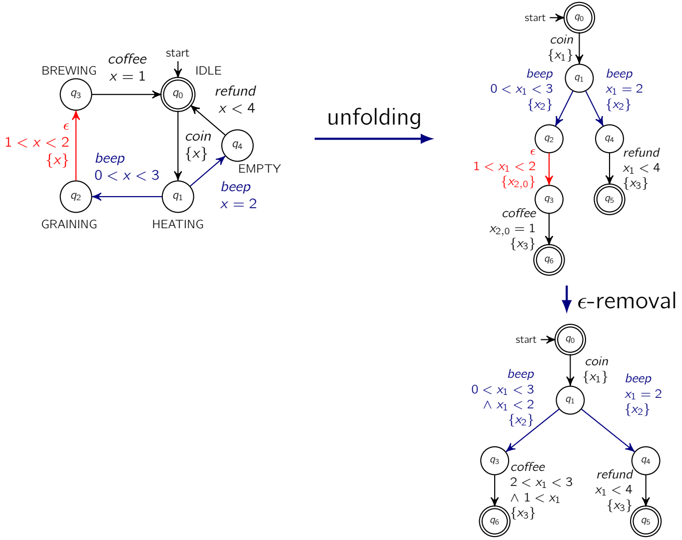
Bounded Determinisation of Timed Automata



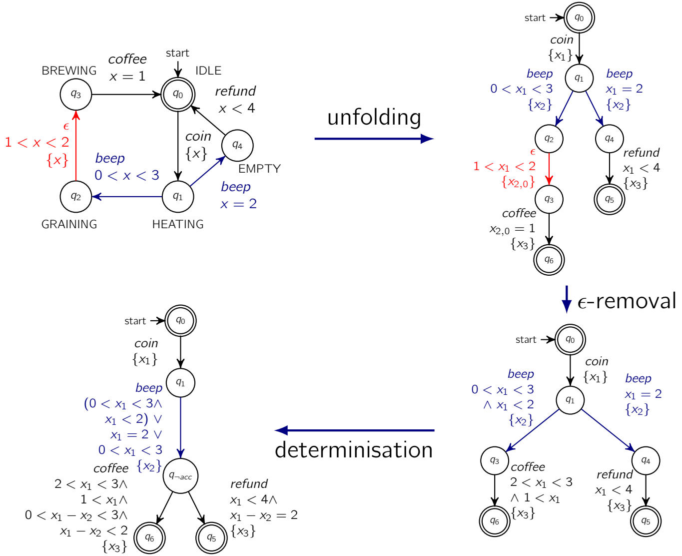
Bounded Determinisation of Timed Automata



Bounded Determinisation of Timed Automata

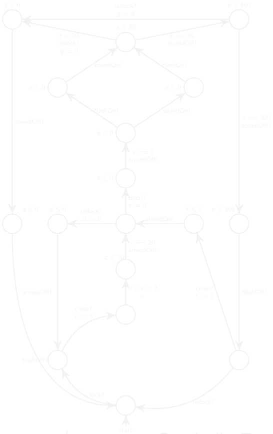


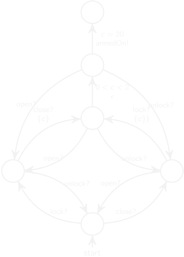
Bounded Determinisation of Timed Automata



Experimental Results IV

- Bounded determinization
- → 13,545 locations (depth 12)
- → bounded model check fails
- **Partial models!**





Model	D.	Bounded Model Checking				Symbolic Execution			
		Mean	Median	Max	Min	Mean	Median	Max	Min
Partial 1	8	9.7s	8.0s	85.1s	0.3s	0.28s	0.04s	16.78s	$\sim 0s$
Partial 2	12	1.6s	1.63s	37.3s	0.08s	0.08s	0.03s	2.28s	$\sim 0s$
Complete	12	x	x	x	x	0.79s	0.06s	360.84s	$\sim 0s$

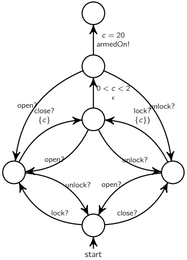
Experimental Results IV

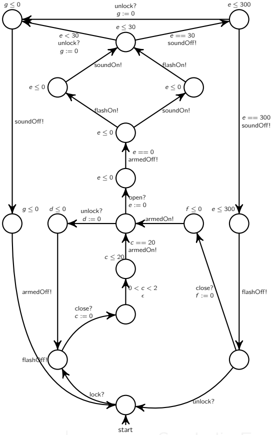
- Bounded determinization

→ 13,545 locations (depth 12)

→ bounded model check fails

- **Partial models!**





Model	D.	Bounded Model Checking				Symbolic Execution			
		Mean	Median	Max	Min	Mean	Median	Max	Min
Partial 1	8	9.7s	8.0s	85.1s	0.3s	0.28s	0.04s	16.78s	$\sim$ 0s
Partial 2	12	1.6s	1.63s	37.3s	0.08s	0.08s	0.03s	2.28s	$\sim$ 0s
Complete	12	×	×	×	×	0.79s	0.06s	360.84s	$\sim$ 0s

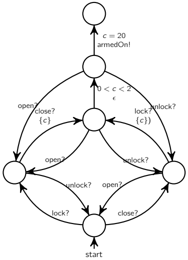
Experimental Results IV

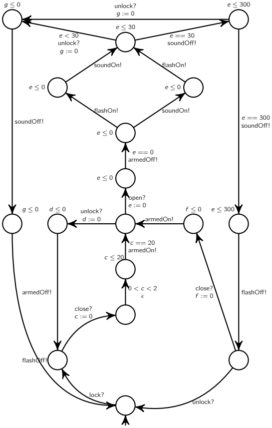
- Bounded determinization

→ 13,545 locations (depth 12)

→ bounded model check fails

- **Partial models!**





Model	D.	Bounded Model Checking				Symbolic Execution			
		Mean	Median	Max	Min	Mean	Median	Max	Min
Partial 1	8	$9.7s$	$8.0s$	$85.1s$	$0.3s$	$0.28s$	$0.04s$	$16.78s$	$\sim 0s$
Partial 2	12	$1.6s$	$1.63s$	$37.3s$	$0.08s$	$0.08s$	$0.03s$	$2.28s$	$\sim 0s$
Complete	12	x	x	x	x	$0.79s$	$0.06s$	$360.84s$	$\sim 0s$

Experimental Results V

- Adding data variable and parameters to
  - deterministic Car Alarm System with one clock
  - 3-digit PIN code for unlocking
- No negative effects, even with higher digit PIN codes
- Symbolic execution faster with 1 clock (0.24s) than with 5 clocks (1.7s)

Depth	Bounded Model Checking				Symbolic Execution			
	Mean	Median	Max	Min	Mean	Median	Max	Min
8	1.46s	0.28s	59.41s	0.12s	0.07s	0.05s	0.82s	$\sim 0s$
12	4.12s	0.35s	35.41s	0.13s	0.24s	0.05s	3.67s	$\sim 0s$

Experimental Results V

- Adding data variable and parameters to
  - deterministic Car Alarm System with one clock
  - 3-digit PIN code for unlocking
- No negative effects, even with higher digit PIN codes
- Symbolic execution faster with 1 clock (0.24s) than with 5 clocks (1.7s)

Depth	Bounded Model Checking				Symbolic Execution			
	Mean	Median	Max	Min	Mean	Median	Max	Min
8	1.46s	0.28s	59.41s	0.12s	0.07s	0.05s	0.82s	$\sim 0s$
12	4.12s	0.35s	35.41s	0.13s	0.24s	0.05s	3.67s	$\sim 0s$

Real-Time Systems Summary

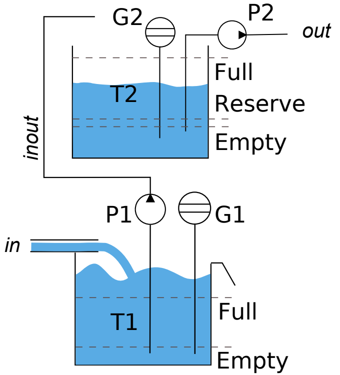
Symbolic execution (SE) seems to perform better, but no clear winner!

- Number of clocks:
    - BMC: small impact (was faster in deterministic case)
    - SE: high impact
- Non-determinism: is an obstacle for conformance checking
    - BMC: state-space explosion → partial models
    - SE: lowered performance (40s vs. 6min) → 3 mutants timed out
- Statistical outliers: due to equivalent mutants
    - BMC: runtime almost equal
    - SE: extreme differences due to optimisations

Agenda

- Model-based Mutation Testing
- Real-Time Systems
- Hybrid Systems
- Discrete Systems

A Hybrid System: Two Tank System



P1, P2 . . . water pumps

G1, G2 . . . water-level sensors

Requirements:

- P1 starts pumping, if T2 below Reserve and T1 is full
- until T1 is empty or T2 is full
- P2 is controlled by button *WaterRequest*
- runs if there is water in T2.
- Note: T1 may overflow

Related Work

- Hybrid Systems
  - Hybrid Automata (Alur,Courcoubetis,Henzinger,Ho 93)
  - Action Systems [Back,Kurki-Suonio 83]
  - Hybrid Action Systems [Rönkkö,Ravn,Sere 03]
  - Qualitative Reasoning [Kuipers 94]
- Testing
  - Mutation Testing [Hamlet 77, De Millo et al. 78]
  - Input-Output Conformance [Brinksma,Tretmans 92]

Abstraction 1: Action Systems

Modeling the Controller

Controller:

$$
\begin{array}{lll}
|[& \textbf{var} & \mathit{P1\_running}, \mathit{P2\_running} : \mathit{Bool}, \\
 & & \mathit{out*}, \mathit{inout*} : \mathit{Real} \\
 & & \bullet \\
 & & \mathit{P1\_running} := \mathit{false}; \\
 & & \mathit{P2\_running} := \mathit{false}; \\
 & & \mathit{out} := 0; \mathit{inout} := 0; \\
 & \textbf{do} & \\
 & & g_1 \rightarrow \mathit{P1\_running} := \mathit{true}; \mathit{inout} := (0, \mathit{Max}] \\
 & & \square \\
 & & g_2 \rightarrow \mathit{P1\_running} := \mathit{false}; \mathit{inout} := 0 \\
 & & \square \\
 & & g_3 \rightarrow \mathit{P2\_running} := \mathit{true}; \mathit{out} := (0, \mathit{Max}] \\
 & & \square \\
 & & g_4 \rightarrow \mathit{P2\_running} := \mathit{false}; \mathit{out} := 0 \\
 & \textbf{od} & \\
]| & : & \mathit{WaterRequest}, x_1, x_2
\end{array}
$$

Guards:

- $g_1 =_{df} \quad x_2 \leq \mathit{Reserve} \wedge$
  $x_1 = \mathit{Full} \wedge$
  $\neg \mathit{P1\_running}$
- $g_2 =_{df} \quad \mathit{P1\_running} \wedge$
  $(x_1 \leq \mathit{Empty} \vee x_2 = \mathit{Full})$
- $g_3 =_{df} \quad \mathit{WaterRequest} \wedge$
  $\neg \mathit{P2\_running} \wedge$
  $x_2 > \mathit{Reserve}$
- $g_4 =_{df} \quad \mathit{P2\_running} \wedge$
  $(\neg \mathit{WaterRequest} \vee$
  $x_2 = \mathit{Empty})$

Abstraction 1: Action Systems

Modeling the Controller

Controller:

```
|[  var  $P1\_running, P2\_running : Bool,$
         $out*, inout* : Real$
         $\bullet$
         $P1\_running := false;$
         $P2\_running := false;$
         $out := 0; inout := 0;$
    do
         $g_1 \rightarrow P1\_running := true; inout := (0, Max]$
         $\Box$
         $g_2 \rightarrow P1\_running := false; inout := 0$
         $\Box$
         $g_3 \rightarrow P2\_running := true; out := (0, Max]$
         $\Box$
         $g_4 \rightarrow P2\_running := false; out := 0$
    od
]|  :    $WaterRequest, x_1, x_2$
```

Guards:

- $g_1 =_{df}$ $x_2 \leq Reserve \wedge$ $x_1 = Full \wedge$ $\neg P1\_running$
- $g_2 =_{df}$ $P1\_running \wedge$ $(x_1 \leq Empty \vee x_2 = Full)$
- $g_3 =_{df}$ $WaterRequest \wedge$ $\neg P2\_running \wedge$ $x_2 > Reserve$
- $g_4 =_{df}$ $P2\_running \wedge$ $(\neg WaterRequest \vee$ $x_2 = Empty)$

Why Action Systems?

- Well-suited for embedded systems modeling
- Action view maps naturally to LTS testing theories
- Solid foundation:
  - precise semantics
  - refinement
- Compositional modeling
- Many extensions available:
  - object-orientation
  - hybrid systems

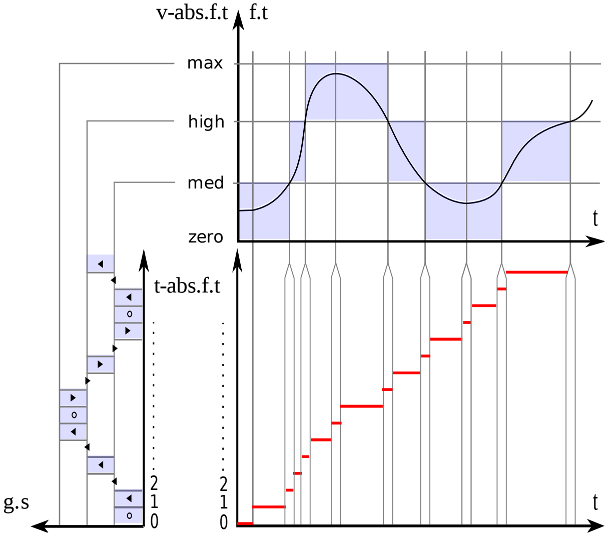
Hybrid Action Systems

Environment:

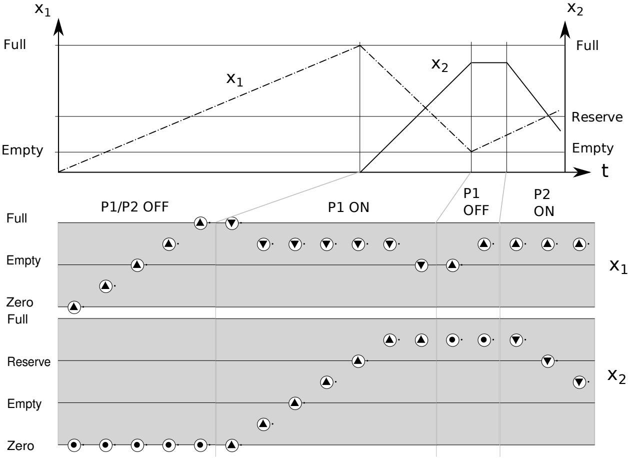
$$
\begin{array}{lll}
|[& \textbf{var} & x_1{*}, x_2{*} : \mathit{Real} \\
 & & \bullet \\
 & & x_1 := 0; x_2 := 0 \\
 & \textbf{alt} & \\
 & & g_1 \rightarrow \ldots \\
 & & \square \\
 & & \ldots \\
 & \textbf{with} & \\
 & & \neg(g_1 \vee \ldots) :\rightarrow \dot{x}_1 = (\mathit{in} - \mathit{inout})/A_1 \wedge \dot{x}_2 = (\mathit{inout} - \mathit{out})/A_2 \\
 & & \\
]| & : & \mathit{inout}, \mathit{out}
\end{array}
$$

- Rönkkö, M., Ravn, A.P., Sere, K.: *Hybrid action systems.* Theoretical Computer Science 290 (2003) 937–973.

Abstraction 2: Qualitative Flows



Example Qualitative Flow of Water Tanks



Qualitative Reasoning (QR)

- QR originates from Artificial Intelligence
- Common sense reasoning about physical systems with possibly incomplete knowledge.
- Ordinary Differential Equations (ODE)
  → Qualitative Differential Equations (QDE):

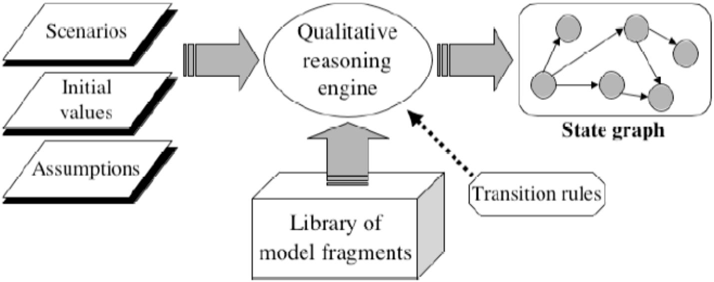
  $$\dot{x}_1 = (in - inout)/A_1 \quad \rightarrow \quad d/dt(x_1, diff_1) \wedge add(diff_1, inout, in)$$
- Arithmetic is reduced to sign algebra:

  $$\begin{array}{lcl} 5 - 1 = 4 & \rightarrow & [+] + [-] = [+] \mid [-] \\ -3 * 2 = -6 & \rightarrow & [-] * [+] = [-] \end{array}$$

Qualitative Action Systems

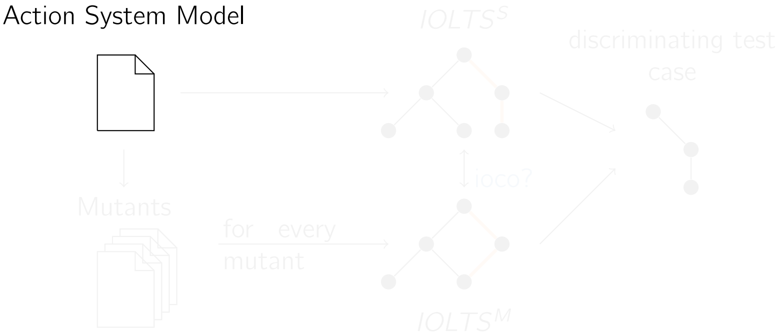
```
|[  var    $x_1*, x_2* : Real$
           $\bullet$
           $x_1 := 0; x_2 := 0$
    alt
           $g_1 \rightarrow \ldots$
           $\Box$
           $\ldots$
    with
           $\neg(g_1 \vee \ldots) :\rightarrow$
           $d/dt(x_1, diff_1) \wedge d/dt(x_2, diff_2) \wedge$
           $add(diff_2, out, inout) \wedge add(diff_1, inout, in)$

]|  :      $inout, out$
```

Qualitative Simulation



- Implementations:
  - QSIM (Lisp)
  - Garp3 (SWI-Prolog)
  - ASIM (GNU-Prolog)

Model-based Mutation Testing



Action System Model

$IOLTS^S$

discriminating test case

ioco?

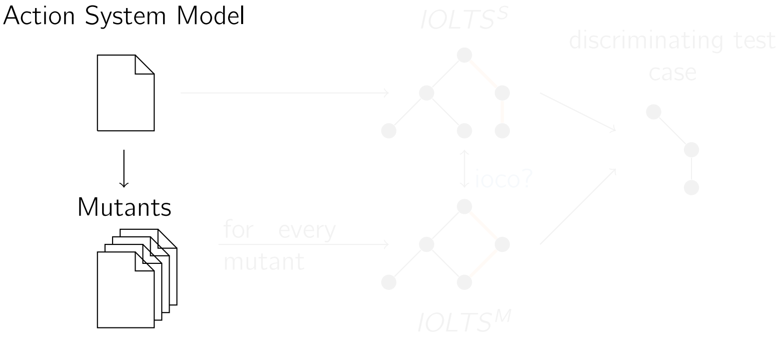
Mutants

for every mutant

$IOLTS^M$

ioco ... input-output conformance

Model-based Mutation Testing

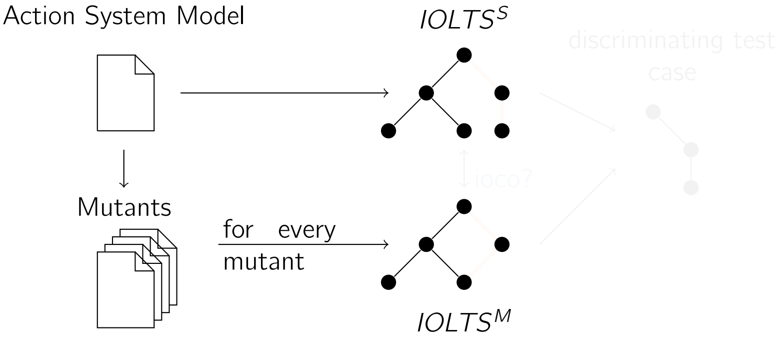


Action System Model

Mutants

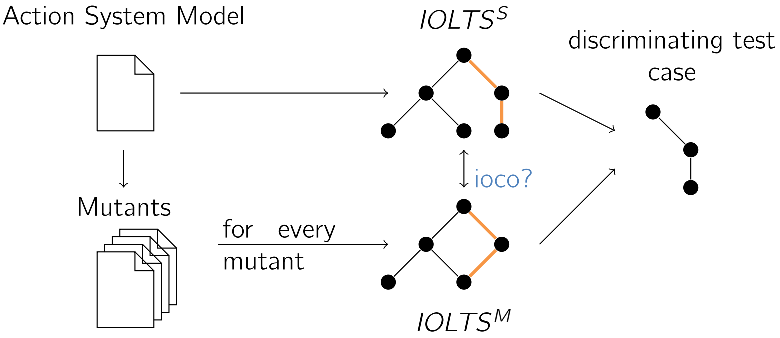
ioco . . . input-output conformance

Model-based Mutation Testing



ioco . . . input-output conformance

Model-based Mutation Testing



Action System Model

$IOLTS^S$

discriminating test case

ioco?

Mutants

for every mutant

$IOLTS^M$

ioco . . . input-output conformance

Conformance Checking

- Event-view: labeled actions
- Input and Output Labels

Def. IOCO [Tretmans 96]

$$\forall \sigma \in \mathrm{Straces}(\mathit{Model}) : \mathrm{out}(\mathit{Mutant}\ \mathrm{after}\ \sigma) \subseteq \mathrm{out}(\mathit{Model}\ \mathrm{after}\ \sigma)$$

out … outputs labels + quiescence

after … reachable states after trace

- ioco supports: partial, non-deterministic models
- ioco-checker Ulysses
  - implemented in GNU Prolog
  - explores discrete actions + qualitative flows
  - builds synchronous product modulo ioco
  - highly non-deterministic → on-the-fly determinization

Conformance Checking

- Event-view: labeled actions
- Input and Output Labels

Def. IOCO [Tretmans 96]

$$\forall \sigma \in \mathrm{Straces}(\mathit{Model}) : \mathrm{out}(\mathit{Mutant}\ \mathrm{after}\ \sigma) \subseteq \mathrm{out}(\mathit{Model}\ \mathrm{after}\ \sigma)$$

out … outputs labels + quiescence
after … reachable states after trace

- ioco supports: partial, non-deterministic models
- ioco-checker Ulysses
  - implemented in GNU Prolog
  - explores discrete actions + qualitative flows
  - builds synchronous product modulo ioco
  - highly non-deterministic → on-the-fly determinization

Generating a Testcase: Original Model

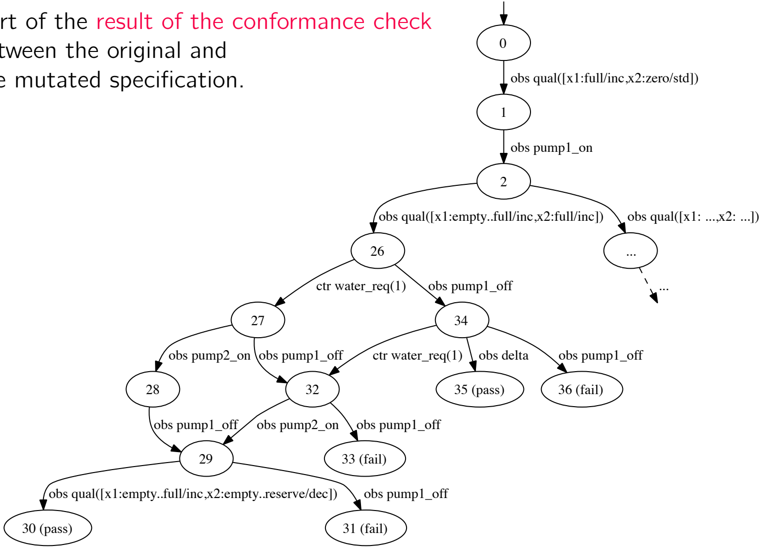
$$
\begin{array}{llll}
\multicolumn{4}{l}{System =} \\
|[& \textbf{var} & & x_1 : T1, x_2 : T2, out, inout : FR, \\
 & & & diff_1, diff_2 : NZP, \\
 & & & p1\_running, p2\_running, wr : Bool \\
 & & \bullet & x_1 := (0,0); x_2 := (0,0); \\
 & & & out := (0,0); inout := (0,0); wr := false \\
 & & & p1\_running := false; p2\_running := false \\
 & \textbf{alt} & & obs\ pump1\_on : g_1 \rightarrow p1\_running := true; \\
 & & & inout := (0..Max, 0) \\
 & & \Box & obs\ pump1\_off : g_2 \rightarrow p1\_running := false; \\
 & & & inout := (0,0) \\
 & & \Box & obs\ pump2\_on : g_3 \rightarrow p2\_running := true; \\
 & & & out := (0..Max, 0) \\
 & & \Box & obs\ pump2\_off : g_4 \rightarrow p2\_running := false; \\
 & & & out := (0,0) \\
 & & \Box & ctr\ water\_req(X) : g_5 \rightarrow wr := X \\
 & \textbf{with} & & \neg(g_1 \vee g_2 \vee g_3 \vee g_4 \vee g_5) :\neg \\
 & & & add(diff_2, out, inout) \wedge add(diff_1, inout, in) \wedge \\
 & & & d/dt(x_1, diff_1) \wedge d/dt(x_2, diff_2) \\
\multicolumn{4}{l}{]| : in}
\end{array}
$$

Generating a Testcase II: Mutated Model

$$
\begin{array}{llll}
\multicolumn{4}{l}{\mathit{System} =} \\
|[& \textbf{var} & & x_1 : T1, x_2 : T2, \mathit{out}, \mathit{inout} : \mathit{FR}, \\
 & & & \mathit{diff}_1, \mathit{diff}_2 : \mathit{NZP}, \\
 & & & \mathit{p1\_running}, \mathit{p2\_running}, \mathit{wr} : \mathit{Bool} \\
 & & \bullet & x_1 := (0,0); x_2 := (0,0); \\
 & & & \mathit{out} := (0,0); \mathit{inout} := (0,0); \mathit{wr} := \mathit{false} \\
 & & & \mathit{p1\_running} := \mathit{false}; \mathit{p2\_running} := \mathit{false} \\
 & \textbf{alt} & & \mathit{obs\ pump1\_on} : g_1 \rightarrow \mathit{p1\_running} := \mathit{true}; \\
 & & & \mathit{inout} := (0..\mathit{Max}, 0) \\
 & & \Box & \mathit{obs\ pump1\_off} : g_2 \rightarrow \mathit{p1\_running} := \mathit{true}; \\
 & & & \mathit{inout} := (0,0) \\
 & & \Box & \mathit{obs\ pump2\_on} : g_3 \rightarrow \mathit{p2\_running} := \mathit{true}; \\
 & & & \mathit{out} := (0..\mathit{Max}, 0) \\
 & & \Box & \mathit{obs\ pump2\_off} : g_4 \rightarrow \mathit{p2\_running} := \mathit{false}; \\
 & & & \mathit{out} := (0,0) \\
 & & \Box & \mathit{ctr\ water\_req(X)} : g_5 \rightarrow \mathit{wr} := X \\
 & \textbf{with} & & \neg(g_1 \vee g_2 \vee g_3 \vee g_4 \vee g_5) :\rightharpoondown \\
 & & & \mathit{add}(\mathit{diff}_2, \mathit{out}, \mathit{inout}) \wedge \mathit{add}(\mathit{diff}_1, \mathit{inout}, \mathit{in}) \wedge \\
 & & & d/dt(x_1, \mathit{diff}_1) \wedge d/dt(x_2, \mathit{diff}_2) \\
\multicolumn{4}{l}{]| : \mathit{in}}
\end{array}
$$

Generating a Testcase III: Product Graph

Part of the result of the conformance check between the original and the mutated specification.



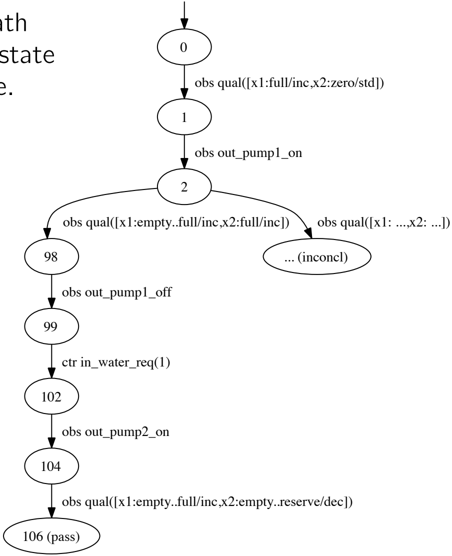
Results

Mut. Op.	No. Mutants	Avg. Time [s]	Average No. States	Average No. Trans.	$\neq$	$=$ No.	$=$ Perc.
ASO	10	13.9	64	117	7	3	30%
ENO	6	7.6	68	120	5	1	17%
ERO	20	12.9	62	110	20	0	0%
LRO	13	12.8	93	168	9	4	31%
MCO	16	12.8	70	126	10	6	38%
RRO	12	12.0	40	73	10	2	17%
Total	77	12.0	66	119	61	16	21%

ASO … Association Shift Operator
ENO … Expression Negation Operator
ERO … Event Replacement Operator
LRO … Logical Operator Replacement
MCO … Missing Condition Operator
RRO … Relational Replacement Operator

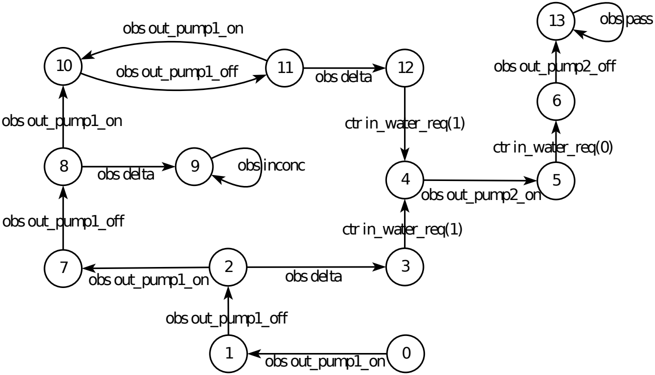
Generating a Testcase IV: Linear TC

Selecting one path for each unsafe state leading to failure.



Generating a Testcase V: Adaptive TC

A test graph including all paths to a given unsafe state leading to failure.



Qualitative events are internal (not visible).

Hybrid Systems Summary

- AI meets FM: qualitative reasoning
- Requirements → incomplete qualitative models
- Model exploration: controller (discrete) + environment (qualitative)
- TCG based on mutation testing and ioco conformance checking
- Different strategies for selecting test case

Agenda

- Model-based Mutation Testing
- Real-Time Systems
- Hybrid Systems
- Discrete Systems

Discrete Systems: MoMuT::UML

Applications:

- Car Alarm System (Ford)
- Railway Interlocking System (Thales)
- Automotive Meassurement Device: Particle Counter (AVL)

SUT: AVL489 Particle Counter

- One of AVL's automotive measurement devices
- Measures particle number concentrations in exhaust gas
- Focus: testing of the control logic
- AVL uses virtual test-beds with simulated devices for integration and regression testing.
- We tested a simulation of the particle counter:
    - Matlab Simulink model compiled to real-time executable
    - Same interface as real device!

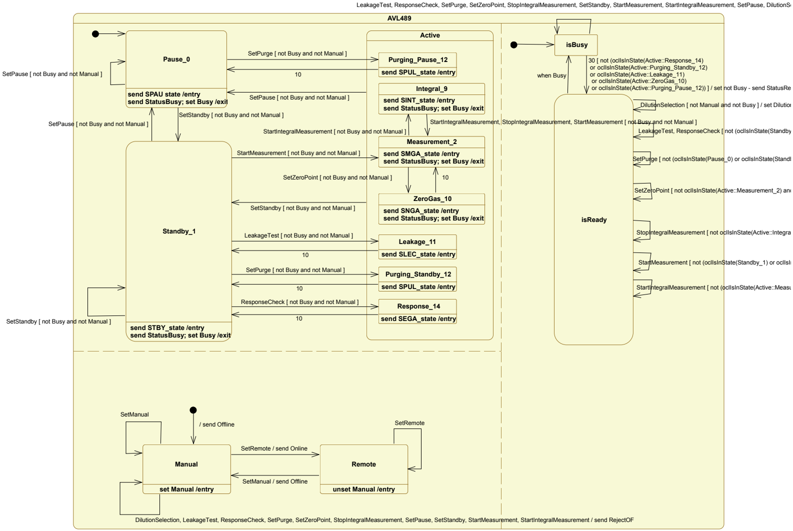
SUT: AVL489 Particle Counter

- One of AVL's automotive measurement devices
- Measures particle number concentrations in exhaust gas
- Focus: testing of the control logic
- AVL uses virtual test-beds with simulated devices for integration and regression testing.
- We tested a simulation of the particle counter:
  - Matlab Simulink model compiled to real-time executable
  - Same interface as real device!

SUT: AVL489 Particle Counter

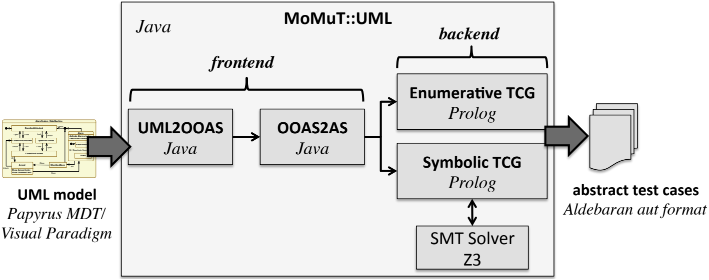
- One of AVL's automotive measurement devices
- Measures particle number concentrations in exhaust gas
- Focus: testing of the control logic
- AVL uses virtual test-beds with simulated devices for integration and regression testing.
- We tested a simulation of the particle counter:
  - Matlab Simulink model compiled to real-time executable
  - Same interface as real device!

UML Test Model of AVL489



MoMuT::UML

- Test-case generator of AIT and TU Graz
- Implementing model-based mutation testing for UML state machines

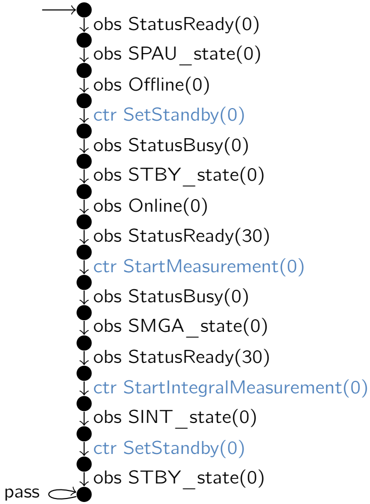


Architecture of the *MoMuT::UML* tool chain

AS … Action Systems [Back83]

OOAS … Object-Oriented Action Systems

Abstract Test Case of AVL489



obs StatusReady(0)
obs SPAU_state(0)
obs Offline(0)
ctr SetStandby(0)
obs StatusBusy(0)
obs STBY_state(0)
obs Online(0)
obs StatusReady(30)
ctr StartMeasurement(0)
obs StatusBusy(0)
obs SMGA_state(0)
obs StatusReady(30)
ctr StartIntegralMeasurement(0)
obs SINT_state(0)
ctr SetStandby(0)
obs STBY_state(0)
pass

Abstract test cases → concrete C# NUnit test cases.

ctr … controllable event (input)
obs … observable event (output)

Test Execution on Particle Counter

We found several bugs in the SUT:

- Forbidden changes of operating state while busy
  - Pause → Standby
  - Normal Measurement → Integral Measurement
- Ignoring high-frequent input without error-messages
- Loss of error messages in client for remote control of the device

Refinement + ioco Conformance Checking

Refinement:

- state-based
- predicative semantics

Def. Refinement [Hoare & He 98]

$$\forall s, s' : Mutant(s, s') \Rightarrow Model(s, s')$$

s … state before
s' … state after execution

Input-Output Conformance:

- event-based
- io labelled transition systems

Def. IOCO [Tretmans 96]

$$\forall \sigma \in \text{traces}(Model) :$$
$$\text{out}(Mutant \text{ after } \sigma) \subseteq \text{out}(Model \text{ after } \sigma)$$

out … outputs labels + quiescence
after … reachable states after trace

New combined conformance checking:

- Refinement checker searches for faulty state (fast)
- Ioco checker looks if faulty state propagates to different observations

Refinement + ioco Conformance Checking

Refinement:

- state-based
- predicative semantics

Input-Output Conformance:

- event-based
- io labelled transition systems

Def. Refinement [Hoare & He 98]

$$\forall s, s' : \mathit{Mutant}(s, s') \Rightarrow \mathit{Model}(s, s')$$

s … state before
s' … state after execution

Def. IOCO [Tretmans 96]

$$\forall \sigma \in \text{traces}(\mathit{Model}) : \text{out}(\mathit{Mutant} \text{ after } \sigma) \subseteq \text{out}(\mathit{Model} \text{ after } \sigma)$$

out … outputs labels + quiescence
after … reachable states after trace

New combined conformance checking:

- Refinement checker searches for faulty state (fast)
- Ioco checker looks if faulty state propagates to different observations

Refinement + ioco Conformance Checking

Refinement:

- state-based
- predicative semantics

Input-Output Conformance:

- event-based
- io labelled transition systems

Def. Refinement [Hoare & He 98]

$$\forall s, s' : \mathit{Mutant}(s, s') \Rightarrow \mathit{Model}(s, s')$$

s … state before
s' … state after execution

Def. IOCO [Tretmans 96]

$$\forall \sigma \in \text{traces}(\mathit{Model}) :$$
$$\text{out}(\mathit{Mutant}\ \text{after}\ \sigma) \subseteq \text{out}(\mathit{Model}\ \text{after}\ \sigma)$$

out … outputs labels + quiescence
after … reachable states after trace

New combined conformance checking:

- Refinement checker searches for faulty state (fast)
- Ioco checker looks if faulty state propagates to different observations

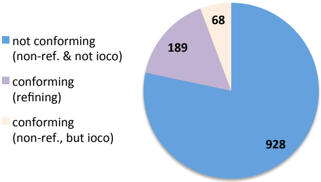
Symbolic Refinement Checking

Is non-refinement reachable?

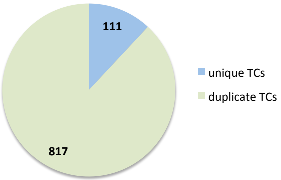
$$\exists\ s, s', tr, tr' : reachable(s, tr) \wedge Mutant(s, s', tr, tr') \wedge \neg Model(s, s', tr, tr')$$

s ... state before
s' ... states after execution
tr ... trace of labels before
tr' ... trace of labels after execution

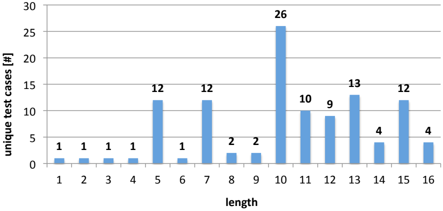
TCG Particle Counter



(a) Breakup into conforming and not conforming model mutants.



(b) Breakup into unique and duplicate test cases.



(c) Lengths of the unique test cases.

Fault Propagation

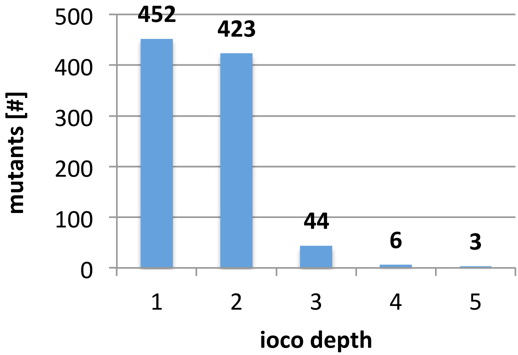


Figure: Number of steps from fault to failure (ioco depths)

Run-times

... for combined conformance checking (in min., max. depth 15+5) :

		conforming (refining)	conforming (non-ref., but ioco)	not conforming (non-ref. & not ioco)	total
mutants [#]		189	68	928	1185
ref. check	Σ	6.1 h	7.7	7.1 h	13.3 h
	ϕ	1.9	6.8 sec	27 sec	40 sec
	max	4.3	1.8	3.9	4.3
ioco check	Σ	-	0.7 h	1.7 h	2.4 h
	ϕ	-	38 sec	7 sec	7.4 sec
	max	-	2	27 sec	2
tc constr.	Σ	-	-	22.9	22.9
	ϕ	-	-	1.5 sec	1.2 sec
	max	-	-	3.7 sec	3.7 sec
total without logging	Σ	6.1 h	0.9 h	9.2 h	**16.2 h**
	ϕ	1.9	0.8	0.6	0.8
	max	4.3	2.2	4.1	4.3

Run-times

... comparison to stand-alone ioco check (in min., max. depth 10):

		not ioco	ioco	total
mutants [#]		719	466	1185
time – ioco check	Σ	9.8 h	22.8 h	32.6 h
	ϕ	0.8	2.9	1.7
	max	3.9	5.2	5.2
time – tc constr.	Σ	19	-	19
	ϕ	1.6 sec	-	1 sec
	max	5.8 sec	-	5.8 sec
total without logging	Σ	10.1 h	22.8 h	**32.9 h**
	ϕ	0.8	2.9	1.7
	max	3.9	5.2	5.2

appr. 16h vs. 33h

Discrete Systems Summary

- Fault propagation important for test-case design
- Faster test-case generator
    - find fault fast (refinement check)
    - analyze if fault propagates to failure (ioco check)
- Optimized refinement check
    - incremental SMT solving, state caching
    - exploiting the location of mutation
    - checking if existing test cases cover next fault
- Applied at AVL: many bugs found [TAP 2014]

Discrete Systems Summary

- Fault propagation important for test-case design
- Faster test-case generator
  - find fault fast (refinement check)
  - analyze if fault propagates to failure (ioco check)
- Optimized refinement check
  - incremental SMT solving, state caching
  - exploiting the location of mutation
  - checking if existing test cases cover next fault
- Applied at AVL: many bugs found [TAP 2014]

Discrete Systems Summary

- Fault propagation important for test-case design
- Faster test-case generator
  - find fault fast (refinement check)
  - analyze if fault propagates to failure (ioco check)
- Optimized refinement check
  - incremental SMT solving, state caching
  - exploiting the location of mutation
  - checking if existing test cases cover next fault
- Applied at AVL: many bugs found [TAP 2014]

Synchronous Systems – MoMuT::REQs

Contract-based Requirement Interfaces:

- Synchronous assume-guarantee pairs
- Combined via conjunction
- Efficient SMT solving

Application: Airbag Chip (Infineon)

```
Inputs  coin,  teabutton,  coffeebutton;
Outputs coffee,  tea;
 Internals  paid;

{I}  not paid and not  coffee  and not  tea
{R1} assume coin'
     guarantee  paid'
{R2} assume paid and teabutton'  and not  coffeebutton'
     guarantee  tea'  and not  paid'
{R3} assume paid and  coffeebutton'  and not  teabutton'
     guarantee  coffee'  and not  paid'
{R4} assume teabutton'  and  coffeebutton'
     guarantee  skip
```

Bernhard K. Aichernig, Klaus Hörmaier, Florian Lorber, Dejan Nickovic, Stefan Tiran. *Require, Test and Trace IT*, FMICS 2015

Bernhard K. Aichernig and Dejan Nickovic and Stefan Tiran. *Scalable Incremental Test-case Generation from Large Behavior Models*, TAP 2015.

Bernhard K. Aichernig, Klaus Hörmaier, Florian Lorber, Dejan Nickovic, Rupert Schlick, Didier Simoneau, Stefan Tiran. *Integration of Requirements Engineering and Test-Case Generation via OSLC*, QSIC 2014

Synchronous Systems – MoMuT::REQs

Contract-based Requirement Interfaces:

- Synchronous assume-guarantee pairs
- Combined via conjunction
- Efficient SMT solving

Application: Airbag Chip (Infineon)

```
Inputs  coin,  teabutton,  coffeebutton;
Outputs coffee,  tea;
 Internals  paid;

{I}  not paid and not coffee and not tea
{R1} assume coin'
     guarantee paid'
{R2} assume paid and teabutton' and not coffeebutton'
     guarantee tea' and not paid'
{R3} assume paid and coffeebutton' and not teabutton'
     guarantee coffee' and not paid'
{R4} assume teabutton' and coffeebutton'
     guarantee skip
```

Bernhard K. Aichernig, Klaus Hörmaier, Florian Lorber, Dejan Nickovic, Stefan Tiran. *Require, Test and Trace IT*, FMICS 2015

Bernhard K. Aichernig and Dejan Nickovic and Stefan Tiran. *Scalable Incremental Test-case Generation from Large Behavior Models*, TAP 2015.

Bernhard K. Aichernig, Klaus Hörmaier, Florian Lorber, Dejan Nickovic, Rupert Schlick, Didier Simoneau, Stefan Tiran. *Integration of Requirements Engineering and Test-Case Generation via OSLC*, QSIC 2014

Summary

- **Model-based Mutation Testing**
  - Automatically test against anticipated faults
  - TCG via conformance checks
- Real-Time Systems: Timed Automata
- Hybrid Systems: Action Systems + Qualitative Reasoning
- Discrete Systems: UML
- Synchronous Systems: Assume-Guarantee Contracts
- Ongoing projects:
  - DSL for easier modelling, performance testing (AVL)
  - Event-B refinement checker including sets, maps (Thales)
  - Dependable Internet of Things: test-based model learning

Summary

- **Model-based Mutation Testing**
    - Automatically test against anticipated faults
    - TCG via conformance checks
- Real-Time Systems: Timed Automata
- Hybrid Systems: Action Systems + Qualitative Reasoning
- Discrete Systems: UML
- Synchronous Systems: Assume-Guarantee Contracts
- Ongoing projects:
    - DSL for easier modelling, performance testing (AVL)
    - Event-B refinement checker including sets, maps (Thales)
    - Dependable Internet of Things: test-based model learning

Summary

- **Model-based Mutation Testing**
  - Automatically test against anticipated faults
  - TCG via conformance checks
- Real-Time Systems: Timed Automata
- Hybrid Systems: Action Systems + Qualitative Reasoning
- Discrete Systems: UML
- Synchronous Systems: Assume-Guarantee Contracts
- Ongoing projects:
  - DSL for easier modelling, performance testing (AVL)
  - Event-B refinement checker including sets, maps (Thales)
  - Dependable Internet of Things: test-based model learning

References

Real-Time Systems

- B.K. Aichernig, F. Lorber, M. Tappler: Conformance Checking of Real-Time Models - Symbolic Execution vs. Bounded Model Checking. Theory and Practice of Formal Methods 2016: 15-32
- F. Lorber, A. Rosenmann, D. Nickovic, B.K. Aichernig: Bounded Determinization of Timed Automata with Silent Transitions. FORMATS 2015: 288-304
- B.K. Aichernig, F. Lorber, D. Nickovic: Time for Mutants - Model-Based Mutation Testing with Timed Automata. TAP 2013: 20-38

Hybrid Systems

- B.K. Aichernig, H. Brandl, E. Jöbstl, W. Krenn: Model-Based Mutation Testing of Hybrid Systems. FMCO 2009: 228-249
- B. K. Aichernig, H. Brandl, W. Krenn: Qualitative Action Systems. ICFEM 2009: 206-225

Discrete Systems

- B.K. Aichernig, J. Auer, E. Jöbstl, R. Korosec, W. Krenn, R. Schlick, B.V. Schmidt: Model-Based Mutation Testing of an Industrial Measurement Device. TAP 2014: 1-19
- Willibald Krenn, Rupert Schlick, Stefan Tiran, Bernhard K. Aichernig, Elisabeth Jöbstl, Harald Brandl: MoMut: : UML Model-Based Mutation Testing for UML. ICST 2015: 1-8