VTS: Conformance testing
Symbolic model-based test selection

Thierry Jéron

IRISA / INRIA Rennes, France
jeron@irisa.fr
http://www.irisa.fr/prive/jeron/

Brief state of the art

- Conformance testing theory for finite state models
  e.g., FSM [LeeYann. 96], ioLTS [Tretmans 96].
- On-line/on-the-fly test generation algorithms and tools
  e.g., TorX [Belinfante et al. 99], TGV [Jard-Jéron 04].

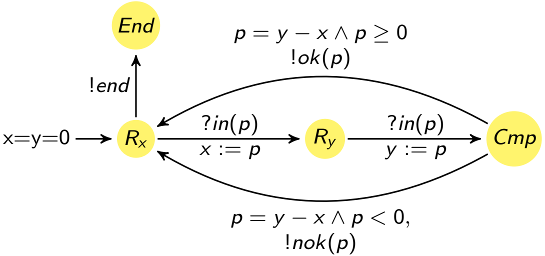
Successfully used on industrial size case studies,
but may suffer from state explosion problems.

For large/infinite state models, solutions based on

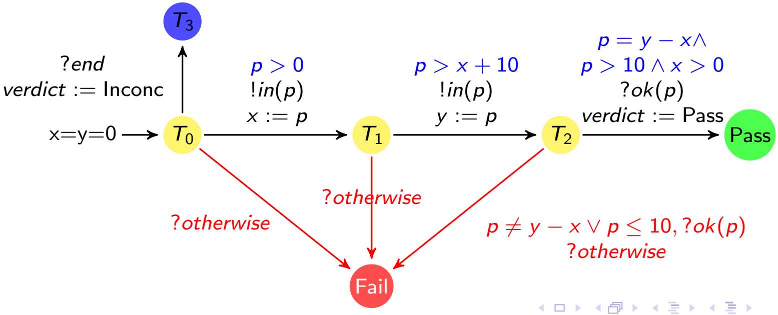
- symbolic execution and constraint solving:
  Agatha [Gaston 06], BZ-TT [Legeard 02], Gatel [Marre-Arnoud 00],
  combination with random exploration: [Godefroid 05].
- abstractions: predicate abstraction [Ball 05],
  finite state generation + concretization [Calamé et al. 05].

Generate instantiated test cases i.e. finite paths

Motivating example



Test for behaviors where $!ok(p)$ is sent with $p > 10$ while x is positive.



What we need

- a model to specify reactive systems
- a model to express testing objectives
- a theory for reasoning about testing
- an algorithm to compute test cases with:
  - backward propagation of symbolic constraints
  - fix-point computation to deal with loops
  - approximation to ensure convergence

Contribution

- Conformance testing based on the ioco testing theory.
- Adapted to infinite state models: ioSTS.
- Focus on models with data variables: guards, assignments.
- Selection of test programs based on approximate analysis.
- Implemented in the STG tool.

Outline

IOSTS syntax

Definition

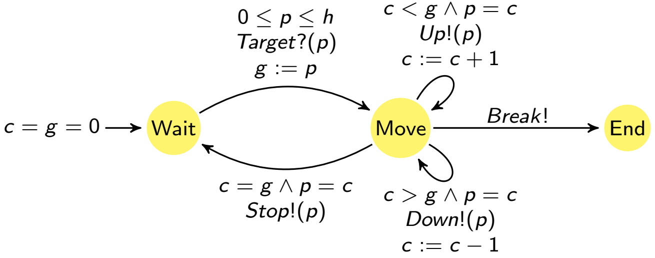
$\mathcal{M} = (V, \Theta, \Sigma, T)$ where:

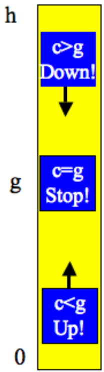
- $V = V_i \cup V_x$: partitioned set of (internal / external) variables
- $\Theta \subseteq \mathcal{D}_{V_i}$: initial condition with unique solution in $\mathcal{D}_{V_i}$.
- $\Sigma = \Sigma_? \cup \Sigma_!$: finite alphabet of actions with communication parameters of type $\mathrm{sig}(a)$.
- T: finite set of symbolic transitions $t = (a, \vec{p}, G, A)$ where
  - $a \in \Sigma$: action
  - $\vec{p}$: tuple of communication parameters local to t;
  - $G \subseteq \mathcal{D}_V \times \mathcal{D}_{\mathrm{sig}(a)}$: guard .
  - $A : \mathcal{D}_V \times \mathcal{D}_{\mathrm{sig}(a)} \rightarrow \mathcal{D}_{V_i}$: assignment.

Assumption

Guards are expressed in a theory in which satisfiability is decidable;

Running example: a simple lift-controller





Parameter: h: integer,
Variables: c, g: integer, pc: {Wait,Move,End}
Inputs: $Target?$; Outputs: $Up!$, $Down!$, $Stop!$, $Break!$;
Communication parameters: p;

IOLTS semantics of IOSTS

The semantics of an ioSTS $\mathcal{M} = (V, \Theta, \Sigma, T)$ is an ioLTS $[\![\mathcal{M}]\!] = (Q, Q_0, \Lambda, \rightarrow)$ where:

- $Q = \mathcal{D}_V$: (infinite) set of states;
- $Q^0 = \{\vec{\nu} = \langle \vec{\nu}_i, \vec{\nu}_x \rangle \mid \vec{\nu}_i \in \Theta \wedge \vec{\nu}_x \in \mathcal{D}_{V_x}\}$: set of initial states;
- $\Lambda = \{\langle a, \vec{\pi} \rangle \mid a \in \Sigma \wedge \vec{\pi} \in \mathcal{D}_{\mathrm{sig}(a)})\}$: set of valued actions partitioned into $\Lambda = \Lambda_? \cup \Lambda_!$;
- $\rightarrow$: transition relation defined by the rule:

$$\frac{(a, \vec{p}, G, A) \in T \quad \vec{\nu} = \langle \vec{\nu}_i, \vec{\nu}_x \rangle \in \mathcal{D}_V \quad \vec{\pi} \in \mathcal{D}_{\mathrm{sig}(a)} \qquad \vec{\nu}' = \langle \vec{\nu}'_i, \vec{\nu}'_x \rangle \in \mathcal{D}_V \quad G(\vec{\nu}, \vec{\pi}) \quad \vec{\nu}'_i = A(\vec{\nu}, \vec{\pi})}{\vec{\nu} \xrightarrow{\langle a, \vec{\pi} \rangle} \vec{\nu}'}$$

Runs, Traces

Run: $Runs(\mathcal{M})$

$$\langle pc = Wait, g = 0, c = 0\rangle \xrightarrow{Target?(3)} \langle Move, 3, 0\rangle \xrightarrow{Up!(0)} \cdots$$
$$\langle Move, 3, 1\rangle \xrightarrow{Up!(1)} \langle Move, 3, 2\rangle \xrightarrow{Up!(2)} \langle Move, 3, 0\rangle \xrightarrow{Stop!(3)} \langle Wait, 3, 0\rangle$$

Traces: $Traces(\mathcal{M})$: projection of runs on valued actions
$Target?(3).Up!(0).Up!(1).Up!(2).Stop!(3)$

$\rightarrow$ Accepted runs, accepted traces in $F \subseteq Q$
$Runs_F(\mathcal{M})$, $Traces_F(\mathcal{M})$.

Deterministic ioSTS

Restriction to deterministic ioSTS, where an ioSTS $\mathcal{M} = (V, \Theta, \Sigma, T)$ is deterministic if for any action $a \in \Sigma$, and any pair of transitions $t_1 = (a, \vec{p}, G_1, A_1)$ and $t_2 = (a, \vec{p}, G_2, A_2)$ carrying the same action, the conjunction of the guards $G_1 \wedge G_2$ is unsatisfiable.

Determinization of ioSTS is not always possible.
Deterministic ioSTS form a strict subclass of ioSTS.
$\longrightarrow$ Determinization heuristic terminates for a subclass of *bounded lookahead* ioSTS.

Observability for testing

The tester controls / observes:

- Inputs / Outputs
- Quiescence: state q is quiescent if no output is fireable in q.

Suspension of $\mathcal{M} = (V, \Theta, \Sigma, T)$:

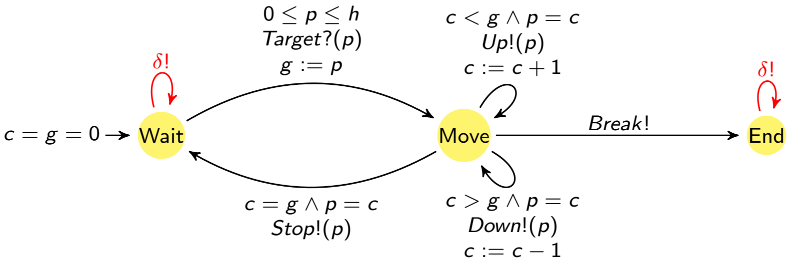
$\Delta(\mathcal{M}) = (V, \Theta, \Sigma^\delta, T_\delta)$ where:

- $\Sigma^\delta = \Sigma^\delta_! \cup \Sigma_?$ with $\Sigma^\delta_! = \Sigma_! \cup \{\delta\}$,
- $T_\delta = T \cup \{\langle \delta, G_\delta, Id_V \rangle\}$ with

$$G_\delta = \neg \left(\bigvee_{(a,\vec{p},G,A)\in T,\ a\in\Sigma_!} \exists \vec{\pi} \in \mathcal{D}_{\mathrm{sig}(a)} : G(\vec{\nu}, \vec{\pi}) \right)$$

Observable behavior for testing: $STraces(\mathcal{M}) \triangleq Traces(\Delta(\mathcal{M}))$

Suspension automaton: example



Outline

Testing framework

Specification

Deterministic ioSTS $\mathcal{S} = (V^{\mathcal{S}}, \Theta^{\mathcal{S}}, \Sigma, T^{\mathcal{S}})$, with $\Sigma = \Sigma_! \cup \Sigma_?$
and $V_x^{\mathcal{S}} = \emptyset$ (only internal variables).
$[\![\mathcal{S}]\!] = S = (Q, Q^0, \Lambda, \rightarrow)$ with $\Lambda = \Lambda_! \cup \Lambda_?$.

Implementation

unknown $\Lambda_?$-complete ioLTS $I = (Q_I, Q_I^0, \Lambda_! \cup \Lambda_?, \rightarrow_I)$.

Test case

ioSTS $\mathcal{TC} = (V^{TC}, \Theta^{TC}, \Sigma^{TC}, T^{TC})$, with $\Sigma_?^{TC} = \Sigma_!$, $\Sigma_!^{TC} = \Sigma_?$
+ variable Verdict with $\mathcal{D}_{verdict} = \{\texttt{none}, \texttt{fail}, \texttt{pass}, \texttt{inconc}\}$
deterministic, $\Sigma_?^{TC}$-complete in all states where Verdict $= \texttt{none}$.
$[\![\mathcal{TC}]\!] = TC = (Q^{TC}, q_0^{TC}, \Lambda^{TC}, \rightarrow_{TC})$
Fail = (Verdict = $\texttt{fail}$), Pass = (Verdict = $\texttt{pass}$), Inconc = (Verdict = $\texttt{inconc}$)

Conformance relation

Definition (Tretmans 96)

$$I \text{ ioco } \mathcal{S} \triangleq \forall \sigma \in Straces(S),$$
$$Out(\Delta(I) \text{ after } \sigma) \subseteq Out(\Delta(S) \text{ after } \sigma)$$

i.e., after a suspension trace of S, outputs (and quiescences) allowed by I are allowed by S.

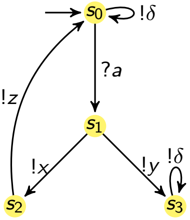
Alternative characterization

$$I \text{ ioco } \mathcal{S} \iff STraces(I) \cap [STraces(\mathcal{S}) \cdot \Lambda_!^\delta \setminus STraces(\mathcal{S})] = \emptyset$$

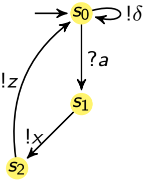
$STraces(\mathcal{S}) \cdot \Lambda_!^\delta \setminus STraces(\mathcal{S})$: minimal non-conformant traces

Examples

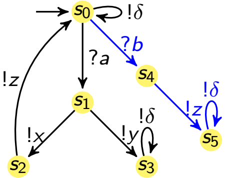
Specification S



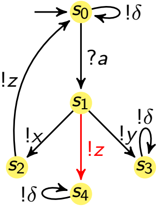
Implementation choice



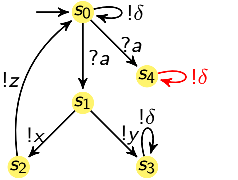
Implementation of a partial specification



Forbidden output



Forbidden quiescence



Canonical tester

Build an observer that recognizes $STraces(\mathcal{S}) \cdot \Lambda_!^\delta \setminus STraces(\mathcal{S})$

Canonical Tester of $\mathcal{S} = (V^\mathcal{S}, \Theta^\mathcal{S}, \Sigma, T^\mathcal{S})$

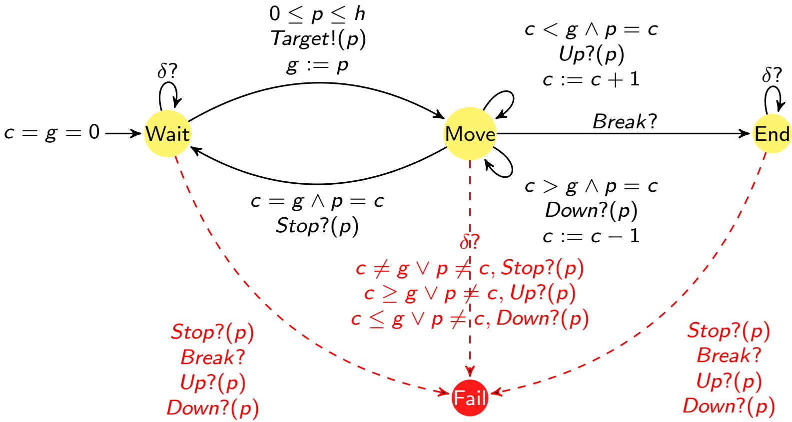
$Can(\mathcal{S}) = (V^{Can}, \Theta^{Can}, \Sigma^{Can}, T^{Can})$ such that:

- $V^{Can} = V^\mathcal{S} \cup \{\text{Verdict}\}$ where $\mathcal{D}_{\text{Verdict}} = \{\texttt{none}, \texttt{fail}\}$
- $\Theta^{Can} = \Theta^\mathcal{S} \wedge \text{Verdict} = \texttt{none}$;
- $\Sigma_?^{Can} = \Sigma_!^\delta$ and $\Sigma_!^{Can} = \Sigma_?$ (alphabet is mirrored / $\Delta(\mathcal{S})$)
- $T^{Can} = T^{\Delta(\mathcal{S})}$ + transitions defined by the rules:

$$\frac{a \in \Sigma_!^\delta = \Sigma_?^{Can} \quad G_a = \bigwedge_{(a,\vec{p},G,A)\in T^{\Delta(\mathcal{S})}} \neg G}{[\, a(\vec{p}) \; : \; G_a(\vec{v}, \vec{p}) \,?\, \text{Verdict}' := \texttt{fail} \,] \in T^{Can}}$$

$$Traces_{\text{Fail}}(Can(\mathcal{S})) = STraces(\mathcal{S}) \cdot \Lambda_!^\delta \setminus STraces(\mathcal{S})$$

Canonical tester of the lift specification



Modeling test execution

Test execution of $\mathcal{TC}$ on I

modelled by the *parallel composition* of
$\Delta(I)$ and $[\![\mathcal{TC}]\!] = TC = (Q^{TC}, q_0^{TC}, \Lambda_? \cup \Lambda_! \cup \{\delta\})$:
$\Delta(I) \| TC = (Q^I \times Q^{TC}, Q_0^I \times \{q_0^{TC}\}, \Lambda_! \cup \{\delta\} \cup \Lambda_?, \rightarrow_{\Delta(I)\|TC})$
where $\rightarrow_{\Delta(I)\|TC}$, is defined by the rule:

$$\frac{\alpha \in \Lambda_! \cup \{\delta\} \cup \Lambda_? \quad q_1 \xrightarrow{\alpha}_{\Delta(I)} q_2 \quad q'_1 \xrightarrow{\alpha}_{TC} q'_2}{(q_1, q'_1) \xrightarrow{\alpha}_{\Delta(I)\|TC} (q_2, q'_2)}$$

$Traces(\Delta(I) \| TC) = STraces(I) \cap Traces(TC) = STraces(I) \cap Traces(\mathcal{TC})$.

Test failure

For $P \in \{\text{Fail}, \text{Pass}, \text{Inconc}\}$,
$Traces_{Q^I \times P}(\Delta(I) \| TC) = STraces(I) \cap Traces_P(TC)$.

Test execution failure

$$TC \ mayfail \ I \quad \triangleq \quad Traces_{Q^I \times \text{Fail}}(\Delta(I) \| TC) \neq \emptyset$$
$$\iff \quad STraces(I) \cap Traces_{\text{Fail}}(TC) \neq \emptyset$$

Similar definitions for *maypass*, *mayinconc*.

Due to choices of the implementation, a test case may fail, pass and inconc on the same implementation

Test case properties

Soundness, Exhaustiveness, Completeness

A set of test cases TS is

- *Sound* $\triangleq \forall I : (I \text{ ioco } S \implies \forall TC \in TS : \neg(TC \textit{ mayfail } I))$,
  i.e., only non-conformant I may be rejected by a $TC \in TS$.
- *Exhaustive*
  $\triangleq \forall I : (\neg(I \text{ ioco } S) \implies \exists TC \in TS : TC \textit{ mayfail } I)$,
  i.e., any non-conformant I may be rejected by a $TC \in TS$.
- Complete = Sound and Exhaustive

Using $TC \textit{ mayfail } I \iff STraces(I) \cap Traces_{\text{Fail}}(TC) \neq \emptyset$:
$I \text{ ioco } S \iff STraces(I) \cap Traces_{\text{Fail}}(Can(\mathcal{S})) = \emptyset$

TS sound iff $\bigcup_{TC \in TS} Traces_{\text{Fail}}(TC) \subseteq Traces_{\text{Fail}}(Can(\mathcal{S}))$
TS exhaustive iff $\bigcup_{TC \in TS} Traces_{\text{Fail}}(TC) \supseteq Traces_{\text{Fail}}(Can(\mathcal{S}))$

Outline

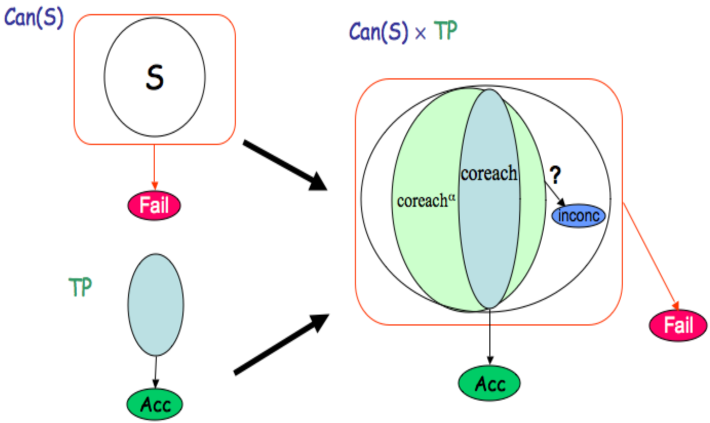
1. The ioSTS model
2. Conformance testing theory
3. Test selection using approximate analysis
4. Test execution
5. Conclusion and perspectives

Principle: overview

Guide test selection by Test Purpose: abstract description of behaviors to be tested.

- Test Purpose specified by observer of $\mathit{Can}(\mathcal{S})$: ioSTS $\mathcal{TP}$.
- Compute the behaviors of $\mathit{Can}(\mathcal{S})$ accepted by $\mathcal{TP}$.
- Problem similar to computing feasible behaviors to a goal.
- Exact computation is not possible
  $\Longrightarrow$ compute over-approximation.

Selection principle



Test purpose

Test selection is guided by a non-intrusive observer:

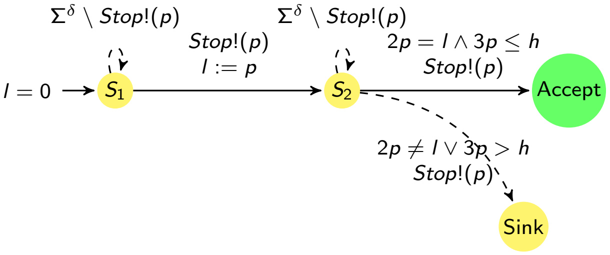
Test Purpose

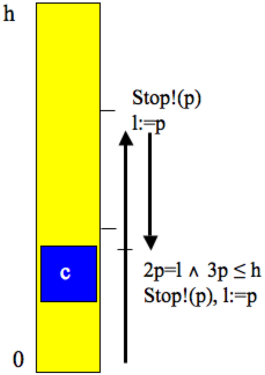
deterministic ioSTS $\mathcal{TP} = (V^{TP}, \Theta^{TP}, \Sigma^{\delta}, T^{TP})$ such that:

- $V_x^{TP} = V_i^S$: $\mathcal{TP}$ is allowed to observe the internal state of $\mathcal{S}$;
- $V_i^{TP} \cap V_i^S = \emptyset$ with $pc^{TP} \in V_i^{TP}$ and accept $\in \mathcal{D}_{pc^{TP}}$.
  Accept $\triangleq (pc^{TP} = \texttt{accept})$.
- $\mathcal{TP}$ is *complete* except in accept:
  $\forall a \in \Sigma^{\delta},\ pc^{TP} \neq \texttt{accept} \Rightarrow \bigvee_{(a,\vec{p},G,A) \in T^{TP}} G = true$.

Note: most coverage criteria can be described by a set of Test Purposes.

A Test Purpose for the lift-controller





Synchronous Product

Used to identify accepting runs.

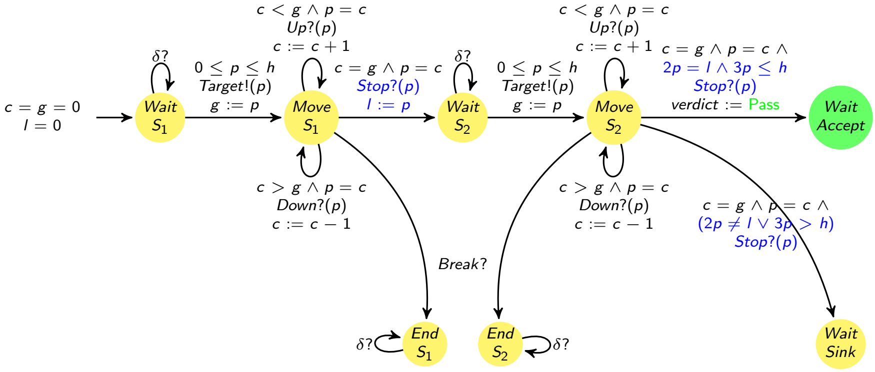
$\mathcal{P} = Can(\mathcal{S}) \times \mathcal{TP} = (V^P, \Theta^P, \Sigma^{Can}, T^P)$ where:

- $V^P = V_i^P \cup V_x^P$, with $V_i^P = V_i^{Can} \cup V_i^{TP}$ and $V_x^P = \emptyset$;
- $\Theta^P(\langle \vec{v}^{Can}, \vec{v}^{TP} \rangle) = \Theta^{Can}(\vec{v}^{Can}) \wedge \Theta^{TP}(\vec{v}^{TP})$;
- T^P is defined by the following inference rule:

$$\frac{\begin{array}{c}[\, a(\vec{p}) \;:\; G^c(\vec{v}^c, \vec{p})\,?\,(\vec{v}_i^c)' := A^c(\vec{v}^c, \vec{p})\,] \in T^{Can} \\ [\, a(\vec{p}) \;:\; G^t(\vec{v}^t, \vec{p})\,?\,(\vec{v}_i^t)' := A^t(\vec{v}^t, \vec{p})\,] \in T^{TP}\end{array}}{\begin{array}{c}[a(\vec{p}) \;:\; G^c(\vec{v}^c, \vec{p}) \wedge G^t(\vec{v}^t, \vec{p})\,? \\ (\vec{v}_i^c)' := A^c(\vec{v}^c, \vec{p}), (\vec{v}_i^t)' := A^t(\vec{v}^t, \vec{p})] \in T^P\end{array}}$$

$\mathcal{P}'$: ioSTS obtained by adding Verdict := pass to transitions with $pc' :=$ accept.

Synchronous product $Can(\mathcal{S}) \times \mathcal{TP}$ for the lift-controller



Properties of $\mathcal{P}' = Can(\mathcal{S}) \times \mathcal{TP}$

$Traces(\mathcal{P}') \subseteq Traces(Can(\mathcal{S}))$
$Traces_{\mathsf{Fail}}(\mathcal{P}') = Traces(\mathcal{P}') \cap Traces_{\mathsf{Fail}}(Can(\mathcal{S}))$.

$\mathcal{P}'$ detects every non-conformance along its traces. It is thus a sound test case.

$Traces_{\mathsf{Pass}}(\mathcal{P}') = Traces_{\mathsf{Accept}}(\mathcal{P}) \subseteq$
$STraces(\mathcal{S}) \cap Traces_{\mathsf{Accept}}(\mathcal{TP})$
(equality if $\mathcal{TP}$ does not observe variables of $\mathcal{S}$).

Over-approximation

Let $pre(A)(X)(\vec{v},\vec{p}) \;=\; \exists \vec{v'} : X(\vec{v'}) \wedge \vec{v'} = A(\vec{v},\vec{p}) \;=\; X(A(\vec{v},\vec{p}))$
i.e., precondition of X by an assignment A
and $pre^{\alpha}(A)(X)(\vec{v},\vec{p}) \supseteq pre(A)(X)(\vec{v},\vec{p})$ an over-apparoximation

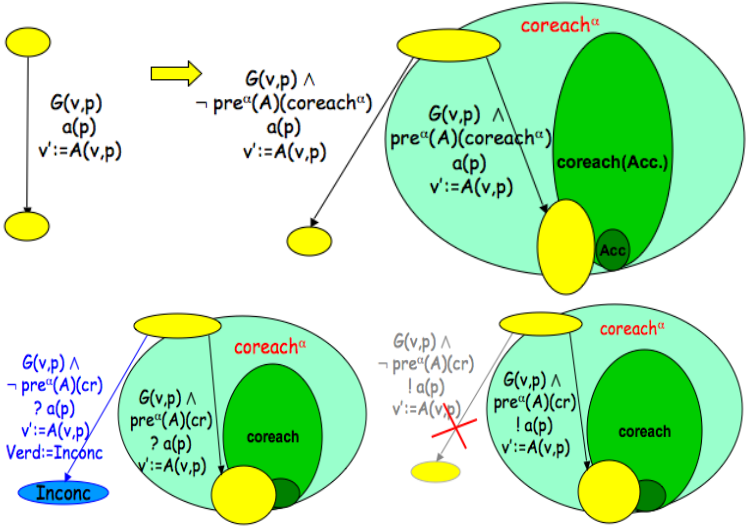
Let $coreach(\text{Pass}) = \text{lfp}(\lambda X.\text{Pass} \cup pre(X))$
where $pre(X) = \{q \mid \exists q' \in X, \exists \alpha \in \Lambda : q \xrightarrow{\alpha} q'\}$ is the set of states from which X can be reached in one transition.

If $coreach^{\alpha}$ is an over-approximation of $coreach(\text{Pass})$, then

- $pre^{\alpha}(A)(coreach^{\alpha})$ is a necessary condition to stay in $coreach(\text{Pass})$
- $\neg pre^{\alpha}(A)(coreach^{\alpha})$ is a sufficient condition to leave $coreach(\text{Pass})$.

Used to reinforce the guards and compute a test case from $\mathcal{P}'$.

Test selection using approximation



Selected test case

The test case for $\mathcal{S}$ and $\mathcal{TP}$ is $\mathcal{TC} = (V^{P'}, \Theta^{P'}, \Sigma^{Can}, T^{\mathcal{TC}})$ where $T^{\mathcal{TC}}$ is defined from $\mathcal{P}'$ by the three rules:

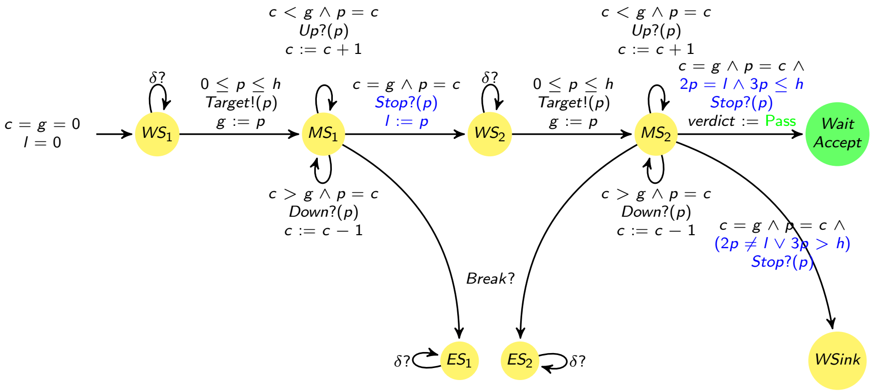
(Select output): $$\frac{(a, \vec{p}, G, A) \in T^{P'} \quad a \in \Sigma_!^{Can} \quad G' = pre^{\alpha}(A)(coreach^{\alpha})}{(a, \vec{p}, G \wedge G', A) \in T^{\mathcal{TC}}}$$

(Fail): $$\frac{(a, \vec{p}, G, A) \in T^{P'} \quad a \in \Sigma_?^{Can} \quad A_{\mathrm{Verdict}} = \mathrm{Verdict}' := \mathtt{fail}}{(a, \vec{p}, G, A) \in T^{\mathcal{TC}}}$$

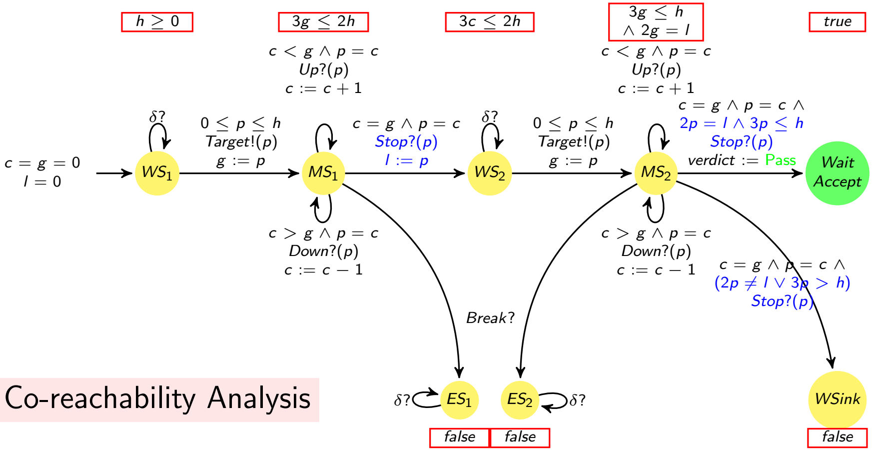
(Split): $$\frac{(a, \vec{p}, G, A) \in T^{P'} \quad a \in \Sigma_?^{Can} \quad A_{\mathrm{Verdict}} \neq \mathrm{Verdict}' := \mathtt{fail} \quad G' = pre^{\alpha}(A)(coreach^{\alpha})}{(a, \vec{p}, G \wedge G', A), (a, \vec{p}, G \wedge \neg G', A') \in T^{\mathcal{TC}}}$$

where A' is defined by $\begin{cases} A'_{\mathrm{Verdict}} = \mathrm{Verdict}' := \mathtt{inconc}, \\ A'_v = A_v \text{ for } v \neq \mathrm{Verdict}, \end{cases}$

Approximate analysis

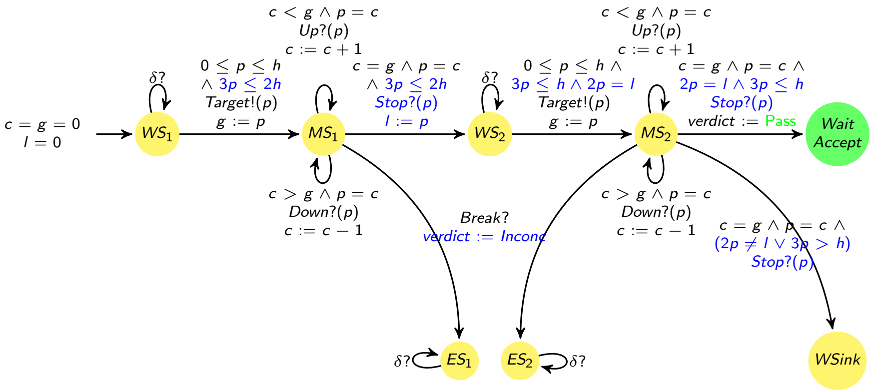


Approximate analysis

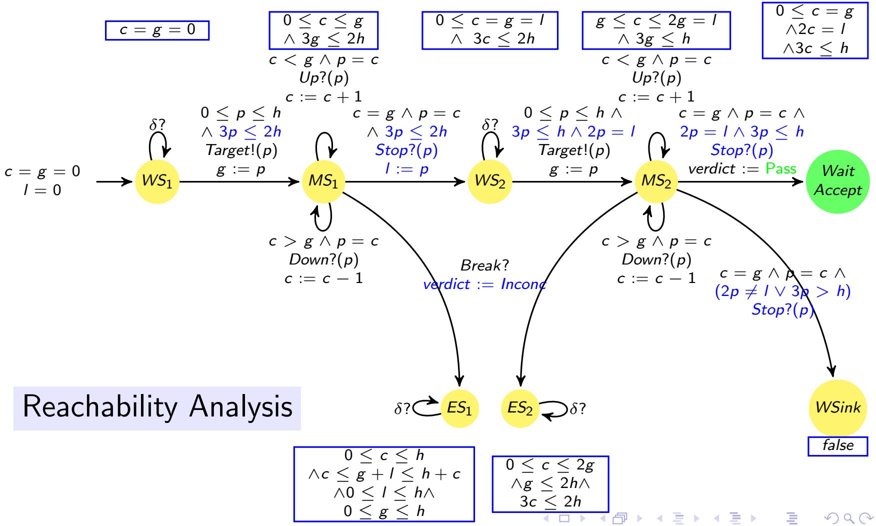


Co-reachability Analysis

Approximate analysis

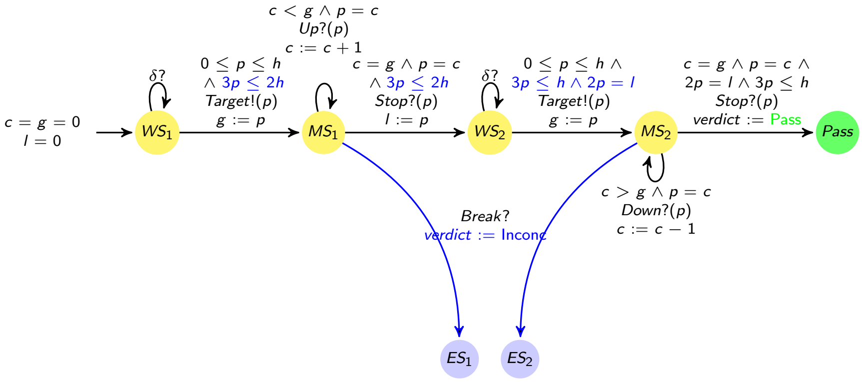


Approximate analysis



Reachability Analysis

Test case for the lift controller



Properties of selected test cases

It can be shown that the (infinite) set of test cases that can be selected is:

Sound : comes from soundness of $Can(\mathcal{S})$.
No fail verdict added by subsequent transformations.

Exhaustive : for any non-conformant implementation I,
choose a minimal non-conformant trace $\sigma.!a$,
choose $!b$ such that $\sigma.!b \in STraces(\mathcal{S})$.
Build $\mathcal{TP}$ recognizing $\sigma.!b$.
The selected $\mathcal{TC}$ fails on $\sigma.!a$.

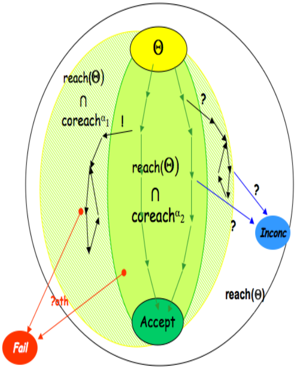
Consequences of over-approximation

For two abstractions α_1 and α_2
(e.g. α_1: control vs α_2: polyhedra)
$pre_{\alpha_1}(A)(coreach_{\alpha_1}) \supseteq pre_{\alpha_2}(A)(coreach_{\alpha_2})$
$\Longrightarrow Traces(\mathcal{TC}_1) \supseteq Traces(\mathcal{TC}_2)$
Less precise approximation $\Longrightarrow$

- More infeasible traces to Accept
- More fail verdicts (all sound)

Limit cases:

- exact analysis: best guiding to Accept
- no analysis: no guiding to Accept



Outline

Test execution

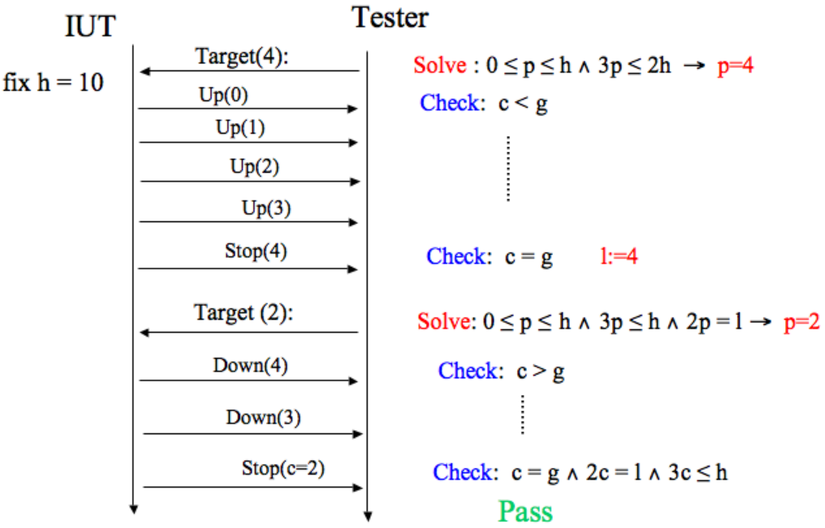
Start from the unique initial state.

In each state $\vec{v}$, repeat until a verdict is set, choose either:

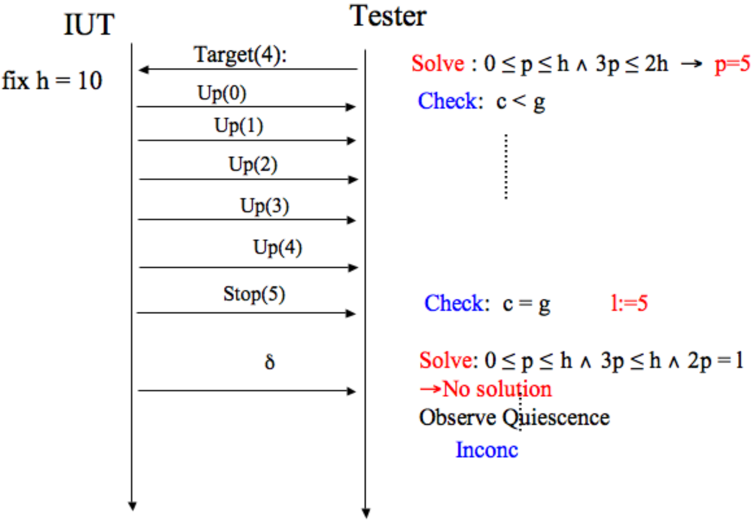
Output: Using constraint solving, choose, $\vec{\pi}$ s.t. $G(\vec{v}, \vec{\pi})$ for $(a, \vec{p}, G, A)$, $a \in \Sigma_!$.
If no solution, receive an input or observe quiescence.
Send $a(\vec{\pi})$ to I.
Move to state $\vec{v'} := A(\vec{v}, \vec{\pi})$.

Input: Receive $a(\vec{\pi})$ from I (or observe quiescence δ).
For each $(a, \vec{p}, G, A)$, $a \in \Sigma_?^\delta$, check $G(\vec{v}, \vec{\pi})$ until one of them is true ($\mathcal{TC}$ is input-complete)
Move to state $\vec{v'} := A(\vec{v}, \vec{\pi})$.

The lift-controller example



The lift-controller example



Outline

1. The ioSTS model
2. Conformance testing theory
3. Test selection using approximate analysis
4. Test execution
5. Conclusion and perspectives

Conclusion

- Test selection algorithm for infinite state (non-deterministic) models of reactive systems
- Using approximate analysis
- Test execution using constraint solving
- Implemented in STG using Nbac (AI) and Lucky (CS)
- Used for conformance testing but a similar approach can be used to eliminate infeasible paths for white box software testing [Denmat 08].

Perspectives

- Tool improvement: simplification of guards, utility of conditions in guards, improved analysis on other domains.
- Similar approach for infinite state heterogeneous models
  - Timed models + data
  - Recursive programs modeled as pushdown systems: [Constant et al. 07]
- Coverage based selection
  - AI + dynamic partitioning as a basis for coverage criteria
  - More semantic based coverage criteria.