

VTS : Verification and Testing of Embedded Systems

Conformance Testing of reactive systems

Thierry Jéron

IRISA / INRIA Rennes, France

jeron@irisa.fr

<http://www.irisa.fr/prive/jeron/>

VerTeCS project team

<http://www.irisa.fr/vertecs>

Main ingredients of a testing theory

Specification, implementation and conformance

Specification: model of requested behaviors,

Implementations: model of *observable* real behavior

Conformance relation: formalizes “IUT conforms to Spec”

Tests cases and their executions

Test cases, test suites: model of tests (control/observation)

Test execution: interaction test \leftrightarrow IUT, produced **observations**, associated **verdicts** (e.g. pass, fail)

Test suite properties: “IUT passes TS” \leftrightarrow “IUT conf S”

Test generation

Algorithms : tests = testgen(Spec) + TS properties & proofs

Testing theory for transition systems

Origins:

- Models : labeled transition systems (LTS)
- Testing pre-orders (Hennessy, De Nicola), refusal testing (Philips), canonical tester (Brinksma)

Not directly useful for test generation.

Problem : no distinction btw observation and control.

Models : LTS with distinction btw input and output (IOSM, IOTS, IOLTS).

Extensions: TIOA (input output timed automata), IOSTS (input output transition systems + data), hybrid automata

Conformance relations : ioco, tioco, etc

Tools : TVeda, TGV, TorX, TestComposer, etc

Bibliography(2)

Testing from FSM:

D. Lee and M. Yannakakis Principles and methods of testing finite state machines - a survey," Proc. of the IEEE, vol. 84, pp. 1090–1123, Aug 1996.

<http://citeseer.ist.psu.edu/lee96principles.html>

Timed automata testing:

M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In SPIN'04 Workshop on Model Checking Software.

<http://www-verimag.imag.fr/~tripakis/papers/timetest.pdf>

Testing infinite state systems:

B. Jeannet, T. Jéron, V. Rusu, E. Zinovieva, Symbolic Test Selection based on Approximate Analysis, in TACAS'05, Volume 3440 of LNCS, p349-364, 2005.

<http://www.irisa.fr/vertecs/Publis/Ps/tacas05.pdf>

IOLTS semantics

Let $M = (Q, \Lambda?, \Lambda!, \mathcal{T}, \rightarrow, Q_0)$ an IOLTS,

Run : $\rho = q_0 \xrightarrow{\lambda_0} q_1 \xrightarrow{\lambda_1} \dots \xrightarrow{\lambda_{n-1}} q_n$ s.t. $q_0 \in Q_0$ is a *run*.

$Runs(M) \subseteq Q_0.(\Lambda.Q)^*$: set of runs of M .

Sequence: projection of a run ρ on Λ :

$\mu = proj_{\Lambda}(\rho) = \lambda_0.\lambda_1 \dots \lambda_{n-1}$.

Language generated by M : $L(M) = proj_{\Lambda}(Runs(M)) \subseteq \Lambda^*$.

Traces: projection $\sigma = proj_{\Lambda_{VIS}}(\rho) = a_1.a_2 \dots a_k$ of a run ρ on Λ_{VIS}

$Traces(M) = proj_{\Lambda_{VIS}}(Runs(M)) \subseteq \Lambda_{VIS}^*$: set of traces of M .

\Rightarrow and after relations

Let $M = (Q, \Lambda?, \Lambda!, \mathcal{T}, \rightarrow, Q_0)$ be an IOLTS.

The trace semantics induces a relation $\Rightarrow \subseteq Q \times \Lambda_{\text{VIS}}^* \times Q$ defined by :

- $q \xrightarrow{\varepsilon} q' \triangleq q = q'$ or $\exists \tau_1, \tau_2 \dots \tau_n \in \mathcal{T} : q \xrightarrow{\tau_1 \cdot \tau_2 \dots \tau_n} q'$
- $a \in \Lambda_{\text{VIS}}, q \xrightarrow{a} q' \triangleq \exists q_1, q_2 : q \xrightarrow{\varepsilon} q_1 \xrightarrow{a} q_2 \xrightarrow{\varepsilon} q'$,
- for $\sigma = a_1 \dots a_n \in \Lambda_{\text{VIS}}^*$,
 $q \xrightarrow{\sigma} q' \triangleq \exists q_0, \dots, q_n : q = q_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} q_n = q'$

The **after** notation:

- $q \text{ after } \sigma \triangleq \{q' \in Q \mid q \xrightarrow{\sigma} q'\}$
- for $P \subseteq Q, P \text{ after } \sigma \triangleq \bigcup_{q \in P} q \text{ after } \sigma$

$M \text{ after } \sigma \triangleq Q_0 \text{ after } \sigma$ denotes the set of states where M can stay after observing σ from an initial state.

Determinization of IOLTSs

For a finite IOLTS $M = (Q, \Lambda_?, \Lambda_!, \mathcal{T}, \rightarrow, Q_0)$ (Q is finite), one can build a deterministic IOLTS $\text{det}(M)$ with same traces as M .

Determinized automaton

$\text{det}(M) = (2^Q, \Lambda_?, \Lambda_!, \rightarrow_{\text{det}}, Q_0 \text{ after } \varepsilon)$ where

- $2^Q = \mathcal{P}(Q)$ is the powerset of Q
- for $P, P' \in 2^Q$ and $a \in \Lambda^{\text{vis}} = \Lambda_? \cup \Lambda_!$,
 $P \xrightarrow{a}_{\text{det}} P'$ iff $P' = P$ after a

$$\text{Traces}(M) = \text{Traces}(\text{det}(M))$$

Complete IOLTS

Let $M = (Q, \Lambda?, \Lambda!, \mathcal{T}, \rightarrow, Q_0)$ be an IOLTS, $q \in Q$ a state of M and $A \subseteq \Lambda_{\text{VIS}}$ a sub-alphabet of visible actions.

q is **strongly A -complete** if any action in A is fireable in q :

$$\forall \lambda \in A, q \xrightarrow{\lambda}$$

q is **weakly A -complete** if any action in A is fireable after some internal actions:

$$\forall \lambda \in A, q \xRightarrow{\lambda}$$

M is (str/wk) A -complete if every state in Q is A -complete.

Proposition

M deterministic \Rightarrow strongly = weakly

M deterministic and Λ_{VIS} -complete $\Rightarrow \text{Traces}(M) = \Lambda_{\text{VIS}}^*$

Testing Models

Specification : IOLTS $S = (Q^S, \Lambda_?, \Lambda_I, \mathcal{T}^S, \rightarrow_S, q_0^S)$

Implementation : IOLTS $I = (Q^I, \Lambda_?, \Lambda_I, \mathcal{T}^I, \rightarrow_I, q_0^I)$

The implementation is unknown, except for its interface, identical to S 's

Hyp.: I is weakly $\Lambda_?$ -complete :

$\forall q \in Q^I, \forall a \in \Lambda_?, q \xrightarrow{a}$.

In every state, I accepts any input, possibly after internal actions (e.g. replies by an error).

NB: assumption required to avoid deadlocks in the interaction tester/IUT.

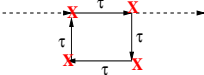
Observation of quiescence

In testing practice, one can observe traces of the *IUT*, but also its **quiescences** with **timers**.

Only *IUT*'s quiescences unspecified in *S* should be rejected.


deadlock 

Notation: $\Gamma(q) \triangleq \{a \in \Lambda \mid q \xrightarrow{a}\}$

livelock 

$q \in \text{deadlock}(M) \triangleq \Gamma(q) = \emptyset$

$q \in \text{livelock}(M) \triangleq \exists \tau_1, \dots, \tau_n, q \xrightarrow{\tau_1 \dots \tau_n} q$

output quiescence 

$q \in \text{outputlock}(M) \triangleq \Gamma(q) \subseteq \Lambda_?$

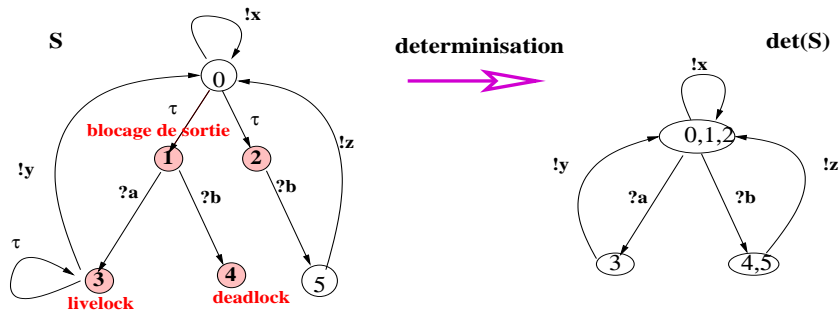
$\text{quiescent}(M) = \text{deadlock}(M) \cup \text{livelock}(M) \cup \text{outputlock}(M)$

Lost of quiescence by determinization

Determinization preserves trace equivalence but not quiescence.

A more refined equivalence is needed

⇒ quiescence must be explicitly before determinization.



Explicitation of quiescence

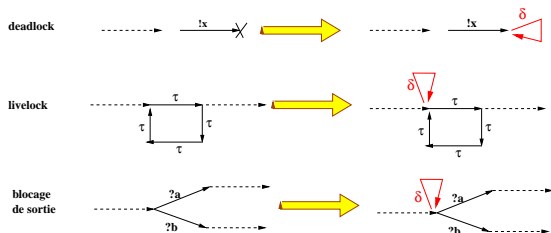
Suspension

The **suspension IOLTS** of $M = (Q, \Lambda_?, \Lambda!, \mathcal{T}, \rightarrow, q_0)$ is an IOLTS $\Delta(M) = (Q, \Lambda_?, \Lambda! \cup \{\delta\}, \mathcal{T}, \rightarrow_\delta, q_0)$ where $\rightarrow_\delta = \rightarrow \cup \{(q, \delta, q) \mid q \in \text{quiescent}(M)\}$

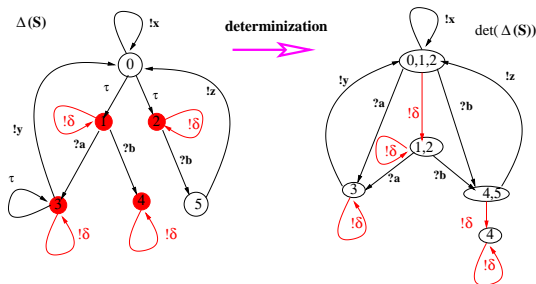
We note $\Lambda!^\delta = \Lambda! \cup \{\delta\}$

δ is considered as an output (it is observable)

and $\Lambda^\delta = \Lambda \cup \{\delta\} = \Lambda_? \cup \Lambda! \cup \{\delta\} \cup \mathcal{T}$



Suspension traces



Suspension traces

$$STraces(M) \triangleq Traces(\Delta(M)) = Traces(\det(\Delta(M)))$$

$STraces(S)$ and $STraces(I)$ represent visible behaviors of S and I for testing \Rightarrow a base for the definition of conformance.

Conformance relation

The **conformance relation** defines the set of IUT I that are conformant to S .

Let $S = (Q^S, \Lambda_?, \Lambda!, \mathcal{T}^S, \rightarrow_S, q_0^S)$ be a specification and $I = (Q^I, \Lambda_?, \Lambda!, \mathcal{T}^I, \rightarrow_I, q_0^I)$ an implementation with same interface

Conformance

$$I \text{ ioco } S$$

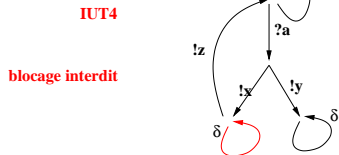
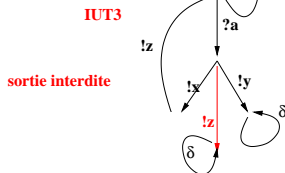
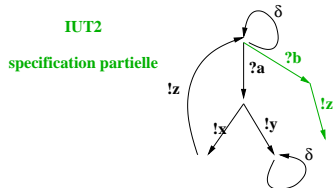
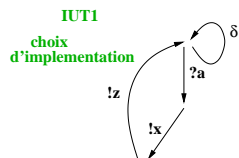
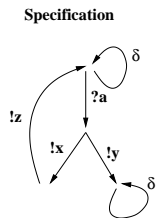
$$\triangleq$$

$$\forall \sigma \in S\text{Traces}(S), \text{Out}(\Delta(IUT) \text{ after } \sigma) \subseteq \text{Out}(\Delta(S) \text{ after } \sigma)$$

with $\text{Out}(P) \triangleq \Gamma(P) \cap \Lambda!^\delta$ the set of output/quiescences in P

Intuition : I conforms to S if and only if, after any suspension trace of S , all outputs and quiescences of I are specified by S .

ioco: example



Characterization of **ioco** in terms of *S*Traces

Proposition

$$I \text{ ioco } S \iff STraces(I) \cap [STraces(S).\Lambda_!^\delta] \subseteq STraces(S) \quad (1)$$

$$\iff STraces(I) \cap [STraces(S).\Lambda_!^\delta \setminus STraces(S)] = \emptyset \quad (2)$$

STraces(I) = visible behaviors of *I*

STraces(S) = visible behaviors of *S*

STraces(S). $\Lambda_!^\delta$ = visible behaviors of *S* prolonged by output or δ .

(1): *S*Traces of *I* prolongating *S*Traces of *S* by outputs or quiescences should remain *S*Traces of *S*.

(2): *I* has no *S*Trace which is an *S*Trace of *S* prolonged by an output or quiescence without being an *S*Trace of *S*.

Canonical tester of S

$Can(S) = (Q^c, \Lambda_?^c = \Lambda_!^\delta, \Lambda_!^c = \Lambda_?, \rightarrow_c, q_0^c,)$ equipped with $Fail \in Q^c$ built from $det(\Delta(S)) = (Q^d, \Lambda_?, \Lambda_!^\delta, \rightarrow^d, q_0^d)$ as follows :

- $Q^c \triangleq Q^d \cup \{Fail\}$, $Fail \notin Q^d$ (new state $Fail$)
- $q_0^c \triangleq q_0^d$
- $\rightarrow_c \triangleq \rightarrow_d \cup \{q \xrightarrow{a}_c Fail \mid q \in Q_d, a \in \Lambda_!^\delta = \Lambda_?^c \wedge \neg(q \xrightarrow{a}_d)\}$
i.e. output completion to $Fail$ (notation $q \xrightarrow{?othw}_c Fail$).

Language recognized by $Can(S)$:

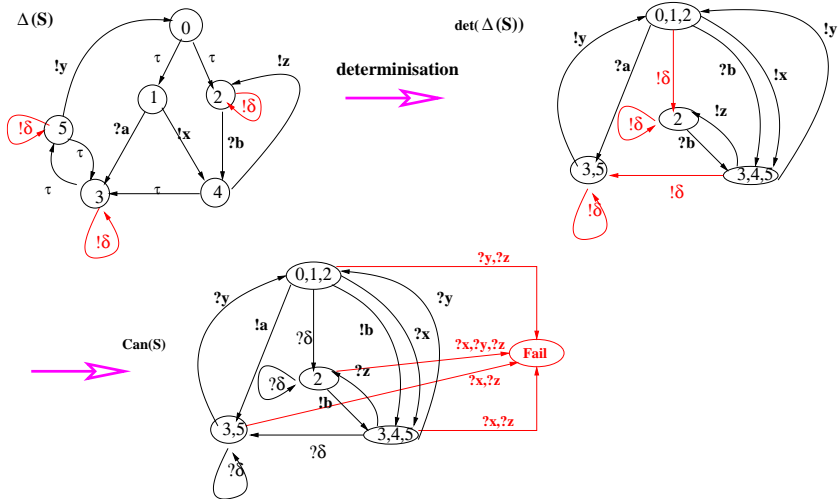
$$Traces_{Fail}(Can(S)) = STraces(S). \Lambda_!^\delta \setminus STraces(S)$$

$$I \text{ ioco } S \iff STraces(I) \cap Traces_{Fail}(Can(S)) = \emptyset$$

i.e. $Can(S)$ is a non-conformance observer

Rem.: For a given S , $I \text{ ioco } S$ is a safety property on $STraces(I)$.
 $Can(S)$ is an observer of the negation of this property.

Canonical tester: example



Test Case

A **test case** for S is an IOLTS $TC = (Q^{TC}, \Lambda_?^{TC}, \Lambda_!^{TC}, \rightarrow_{TC}, q_0^{TC})$ s.t. :

- TC is **deterministic**
- $\Lambda_!^{TC} = \Lambda_?$ and $\Lambda_?^{TC} = \Lambda_!^\delta = \Lambda_! \cup \{\delta\}$ (**input/output inversion**)
- TC is equipped with a set of trap states (q s.t. $\forall a \in \Lambda^{TC}, \neg(q \xrightarrow{a})$) representing **verdicts** :
In general **Verdicts** = **Pass** \cup **Fail** \cup **Inconc** $\subseteq Q^{TC}$
- states of TC , except **Verdicts**, are $\Lambda_?^{TC}$ -**complete**
i.e. TC is ready to receive any input in $\Lambda_?^{TC}$ = output in $\Lambda_!^\delta$.
?othw denotes the complement of a set of input.

$$\begin{aligned} \text{Traces}_{\text{Verdict}}(TC) = \\ \text{Traces}_{\text{Fail}}(TC) \cup \text{Traces}_{\text{Pass}}(TC) \cup \text{Traces}_{\text{Inconc}}(TC) \end{aligned}$$

Test execution

Modelled by the parallel composition $TC \parallel \Delta(I)$ with synchronization on actions of the common interface $\Lambda_{\text{VIS}}^\delta$:

$$TC \parallel \Delta(I) = (Q^{\text{TC}} \times Q^I, \Lambda^I, \rightarrow_{TC \parallel \Delta(I)}, (q_0^{\text{TC}}, q_0^I))$$

where $\rightarrow_{TC \parallel \Delta(I)}$ is defined by:

$$\frac{tc \xrightarrow{a}_{\text{TC}} tc' \quad q \xrightarrow{a}_{\Delta(I)} q' \quad a \in \Lambda_{\text{VIS}}^\delta}{(tc, q) \xrightarrow{a}_{TC \parallel \Delta(I)} (tc', q')} \quad \frac{q \xrightarrow{\tau}_{\Delta(I)} q' \quad \tau \in \mathcal{T}^I}{(tc, q) \xrightarrow{\tau}_{TC \parallel \Delta(I)} (tc, q')}$$

Prop

$$\text{Traces}(TC \parallel \Delta(I)) = \text{Traces}(TC) \cap \text{STraces}(I)$$

Prop

I weakly $\Lambda_{\text{VIS}}^\delta$ -complete and TC $\Lambda_{\text{VIS}}^\delta$ -complete (except in **Verdicts**)
 $\Rightarrow TC \parallel \Delta(I)$ is never blocked except in **Verdicts** states of TC

Verdicts

We say that a test case TC rejects (*fails*) I iff an execution of $TC \parallel \Delta(I)$ reaches **Fail**

This expresses a *possibility* for rejection.

$$TC \text{ fails } I \triangleq STraces(I) \cap Traces_{\text{Fail}}(TC) \neq \emptyset$$

! due to non-controllable choices of I , a single test case applied on a single *IUT* can produce all different verdicts !

See the preceding example where

$$STraces(I) \cap Traces_{\text{Fail}}(TC) \neq \emptyset \text{ and}$$

$$STraces(I) \cap Traces_{\text{Pass}}(TC) \neq \emptyset \text{ and}$$

$$STraces(I) \cap Traces_{\text{Inconc}}(TC) \neq \emptyset$$

Test suites properties

The verdicts obtained by the execution of a test suite on an implementation should be related to conformance:

- rejection should imply non-conformance (soundness)
- conversely, it would be fine if non-conformance imply rejection (exhaustiveness)

Soundness

Soundness of a test case/ a test suite

A test case TC is **sound** for S wrt **ioco** if it may reject non-conformant IUTs only:

$$\begin{aligned}
 TC \text{ sound}/S &\triangleq \forall I, [TC \text{ fails } I \Rightarrow \neg(I \text{ ioco } S)] \\
 &(\iff \forall I, [I \text{ ioco } S \Rightarrow \neg(TC \text{ fails } I)])
 \end{aligned}$$

A test suite TS is **sound** if all its test cases are sound.

$$TS \text{ sound}/S \triangleq \forall TC \in TS, TC \text{ sound}/\text{ioco}, S$$

Necessary and sufficient condition for soundness

Remember that

- $I \text{ ioco } S \iff S\text{Traces}(I) \cap \text{Traces}_{\text{Fail}}(\text{Can}(S)) = \emptyset$
- $TC \text{ fails } I \iff S\text{Traces}(I) \cap \text{Traces}_{\text{Fail}}(TC) \neq \emptyset$
- $TC \text{ sound}/S \iff \forall I, [I \text{ ioco } S \Rightarrow \neg(TC \text{ fails } I)]$

This implies the necessary and sufficient condition :

Proposition

$$TC \text{ sound}/S \iff \text{Traces}_{\text{Fail}}(TC) \subseteq \text{Traces}_{\text{Fail}}(\text{Can}(S))$$

$$TS \text{ sound}/S \iff \bigcup_{TC \in TS} \text{Traces}_{\text{Fail}}(TC) \subseteq \text{Traces}_{\text{Fail}}(\text{Can}(S))$$

Exhaustiveness

Exhaustiveness of a test suite

A test suite TS is **exhaustive** for S wrt **ioco** if for any non-conformant IUT, there exists a test case in TS that **may reject** it:

$$\begin{aligned}
 TS \text{ exhaustive} &\triangleq \forall I, [\neg(I \text{ ioco } S) \Rightarrow [\exists TC \in TS, TC \text{ fails } I]] \\
 &(\iff \forall I, [[\forall TC \in TS, \neg(TC \text{ fails } I)] \Rightarrow I \text{ ioco } S])
 \end{aligned}$$

i.e. for any I , if no TC can reject I , then I conforms to S

Proposition

$$TS \text{ exhaustive}/S \iff \text{Traces}_{\text{Fail}}(\text{Can}(S)) \subseteq \bigcup_{TC \in TS} \text{Traces}_{\text{Fail}}(TC)$$

Canonical tester / soundness and exhaustiveness

Proposition

TS is complete (sound and exhaustive)/**ioco**, $S \iff$
 $\bigcup_{TC \in TS} Traces_{Fail}(TC) = Traces_{Fail}(Can(S))$

In particular $TS = \{Can(S)\}$ is sound and exhaustive.

$$\begin{aligned}
 I \text{ ioco } S &\iff STraces(I) \cap Traces_{Fail}(Can(S)) = \emptyset \text{ (def of ioco)} \\
 &\iff \neg(Can(S) \text{ fails } I) \text{ (def. of fails)}
 \end{aligned}$$

$Can(S)$ is a **canonical tester** for S wrt **ioco** i.e. the most general test case.

Pb: $Can(S)$ has too much behaviors \Rightarrow selection.

Test case properties (I)

Theorem

The test suite composed of the (infinite) set of test cases that the algorithm can produce is **sound** and **exhaustive**.

Soundness: $Traces_{Fail}(TC) \subseteq Traces_{Fail}(Can(S))$

$\Rightarrow TC$ is sound.

Exhaustiveness: if $\neg(I \text{ ioco } S)$ then $\exists \sigma \in STraces(S), \exists x \in \Lambda_1^\delta, x \in Out(\Delta(I) \text{ after } \sigma) \wedge x \notin Out(\Delta(S) \text{ after } \sigma)$.

We have $\exists y \in Out(\Delta(S) \text{ after } \sigma)$, (as $Out(\Delta(S) \text{ after } \sigma) \neq \emptyset$).

Let $\sigma' = \sigma.y (\in STraces(S))$ and $TP = \sigma'$ finished by *Accept*.

By definition, CTG after $\sigma.x \in \mathbf{Fail}$. Prune CTG in TC s.t.

$\sigma.x \in Traces(TC) \implies TC \text{ after } \sigma.x \in \mathbf{Fail}$. Thus TC fails *IUT*.

