# 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

# Outline

1. Introduction to conformance testing
2. Testing theory for LTS
   - The IOLTS model
   - Non-determinism
   - Quiescence
   - Conformance relation
   - Canonical tester
   - Test execution and verdicts
   - Test suite properties
3. Test selection
   - Non-deterministic selection
   - Test selection guided by test purpose
4. Conclusion

# Introduction to conformance testing

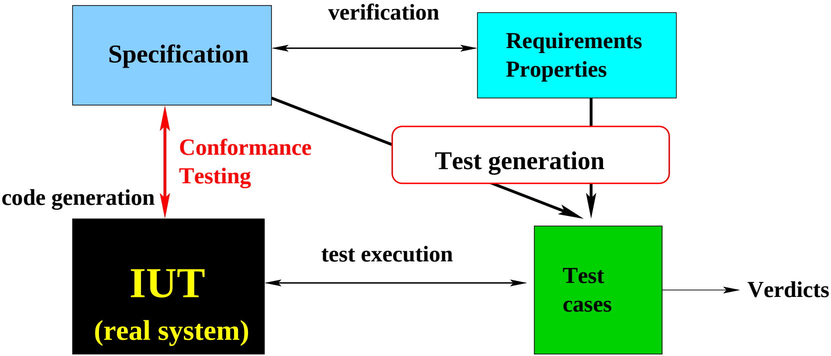
Testing that a black-box implementation (IUT) of a system behaves correctly wrt its functional specifition Spec.

IUT: *implementation under test*
real system (hardware or software)

Main differences with structural testing (white box):

- **black box** IUT: unknown code, but known interface
- Spec is the **reference**.
  $\Longrightarrow$ Oracle defined by admissible behaviors of Spec.
  $\rightarrow$ formal unambiguous specifications should be used.

# Conformance testing: general scheme



# Motivation

Industrial practice: manual design of test suites from informal specifications

- more than 30% of the development cost ($\geq$ for critical systems)
- ad-hoc, long, repetitive, error prone,
- maintenance is difficult in case of modifications of Spec,
- no clear definition of conformance and of the testing process.

$\Rightarrow$ automatization of test synthesis from formal specifications can be profit earning (effort $\Rightarrow$ cost)

$\longrightarrow$ *model-based testing/test generation*

# Modeling

Description of a system in a language (syntax) which semantics can be expressed in a mathematical model.

$\Longrightarrow$ allows to describe and analyze its properties : verification.

Depends on the application domain (here embedded system)

Models: algebraic spec., logics and sets, automata/ transition systems (extended), timed automata, hybrid automata, etc.

Gain:

- non-ambiguity, abstraction, masking of implementation choices,
- required properties of software / implementation choice
- required for critical applications
- useful for verification, code generation,
- reference for test verdicts (oracle)

# 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

# Conformance testing of reactive systems

**Reactive systems**

System which reacts to its environment through its interfaces.

- Environment: human, software, hardware
- Interfaces: commands, captors, communication channels, operations, methods, duration.
- Difference btw control/observation: input/output + duration, call /return, etc

# Outline

1. Introduction to conformance testing
2. Testing theory for LTS
   - The IOLTS model
   - Non-determinism
   - Quiescence
   - Conformance relation
   - Canonical tester
   - Test execution and verdicts
   - Test suite properties
3. Test selection
   - Non-deterministic selection
   - Test selection guided by test purpose
4. Conclusion

# Testing theory for transition systems

**Origins:**

- Models : labeled transition systems (LTS)
- Testing pre-orders (Henessy, 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

# Overview

## Models

- IOLTS, semantics
- Quiescence, non-determinism

## Testing theory

- Specifications, implementations, conformance relation
- Tests and their properties

## Test generation for finite state IOLTSs

- Random generation: algorithm and properties
- Generation guided by a test purpose: algorithm and properties

## Test generation for infinite state systems

- The IOSTS model
- Generation by approximate analysis

# Bibliography

**J. Tretmans.** Test Generation with Inputs, Outputs and Repetitive Quiescence, Software-Concepts and Tools, 17(3), pp 103-120, 1996

**C. Jard, T. Jéron** TGV: theory, principles and algorithms, A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems, Software Tools for Technology Transfer (STTT), 6, October 2004. http://www.irisa.fr/vertecs/Publis/Ps/2004-STTT.pdf

**M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, A. Pretschner (eds.)**, Model-Based Testing of Reactive Systems. Lecture Notes in Computer Science No 3472, Springer Verlag, 2005. http://www.springerlink.com/content/br3e64927j30/

# 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

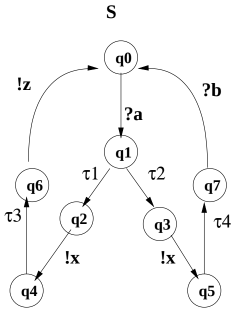
Testing infinite state systems:

**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

# The IOLTS model: input/output labeled transition systems

$M = (Q, \Lambda_?, \Lambda_!, \mathcal{T}, \rightarrow, Q_0)$ with

- $Q$: enumerable set of states,
- $\Lambda = \Lambda_? \cup \Lambda_! \cup \mathcal{T}$ : action alphabet , where
  $\Lambda_?$: inputs (?a), $\Lambda_!$: outputs (!x),
  $\mathcal{T}$: internal actions ($\tau_i$),
  $\Lambda_{\text{VIS}} = \Lambda \setminus \mathcal{T} = \Lambda_? \cup \Lambda_!$: visible actions
- $\rightarrow \subseteq Q \times \Lambda \times Q$: transition **relation**,
- $Q_0 \subseteq Q$: set of initial states.




**Assumption:** no $\tau$-divergence.
i.e. no infinite sequence of $\tau$
through infinitely many states

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

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

# IOLTS seen as automata

An IOLTS $M = (Q, \Lambda_?, \Lambda_!, \mathcal{T}, \rightarrow, Q_0)$ with set of marked states $X \subseteq Q$ can be interpreted as an automaton:

Run accepted in $X$ :

$\rho = q_0 \xrightarrow{\lambda_0} q_1 \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_{n-1}} q_n \in Runs(M)$ s.t. $q_n \in X$.

Accepted sequence = projection on $\Lambda^*$ of an accepted run.

Accepted trace = projection on $\Lambda_{\mathrm{VIS}}^*$ of an accepted run.

$Runs_X(M) \subseteq Runs(M) = Runs_Q(M)$,
$L_X(M) = proj_\Lambda(Runs_X(M)) \subseteq L(M) = L_Q(M)$,
$Traces_X(M) = proj_{\Lambda_{\mathrm{VIS}}}(Runs_X(M)) \subseteq Traces(M) = Traces_Q(M)$.

# $\Rightarrow$ and after relations

Let $M = (Q, \Lambda_?, \Lambda_!, \mathcal{T}, \rightarrow, Q_0)$ be an IOLTS.
The trace semantics induces a relation $\Rightarrow \subseteq Q \times \Lambda_{\mathrm{VIS}}^* \times Q$ defined by :

- $q \overset{\varepsilon}{\Rightarrow} q' \triangleq q = q'$ or $\exists \tau_1, \tau_2 \ldots \tau_n \in \mathcal{T} \; : \; q \xrightarrow{\tau_1 \cdot \tau_2 \cdots \tau_n} q'$
- $a \in \Lambda_{\mathrm{VIS}}$, $q \overset{a}{\Rightarrow} q' \triangleq \exists q_1, q_2 \; : q \overset{\varepsilon}{\Rightarrow} q_1 \xrightarrow{a} q_2 \overset{\varepsilon}{\Rightarrow} q'$,
- for $\sigma = a_1 \cdots a_n \in \Lambda_{\mathrm{VIS}}^*$,

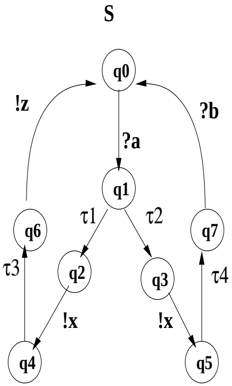
$q \overset{\sigma}{\Rightarrow} q' \triangleq \exists q_0, \ldots q_n \; : q = q_0 \overset{a_1}{\Rightarrow} \cdots \overset{a_n}{\Rightarrow} q_n = q'$

The after notation:

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

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

# Illustration




$q_1 \overset{\varepsilon}{\Rightarrow} q_2$ et $q_1 \overset{\varepsilon}{\Rightarrow} q_3$
$q_1 \overset{!x}{\Rightarrow} q_4$, $q_1 \overset{!x}{\Rightarrow} q_6$, $q_1 \overset{!x}{\Rightarrow} q_5$, $q_1 \overset{!x}{\Rightarrow} q_7$,
$q_1 \overset{!x.!z}{\Rightarrow} q_0$

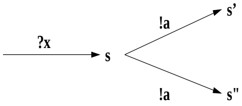
$q_0$ after $?a.!x = \{q_4, q_5, q_6, q_7\}$,
$q_0$ after $?a.!z = \emptyset$

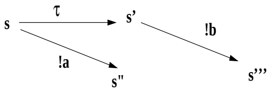
$\{q_2, q_3\}$ after $!x = \{q_4, q_5, q_6, q_7\}$

$Traces(S) = \{\varepsilon, ?a, ?a.!x, ?a.!x.!z, ?a.!x.?b, \ldots\}$

# Non-determinism and choice

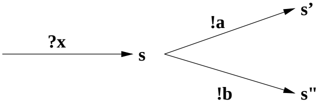
Non-determinism in the usual automata meaning: existence of internal actions $\tau$ and $\rightarrow$ is a relation




$M$ is deterministic if it has no internal action,
$Card(Q_0) = 1$ and $\forall q \in Q, \forall a \in \Lambda_{\mathsf{VIS}},\ Card(\{q' \mid q \xrightarrow{a} q'\}) \leq 1$.

Uncontrolled choice: sometimes called *observable non-determinism*



# 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 $det(M)$ with same traces as $M$.

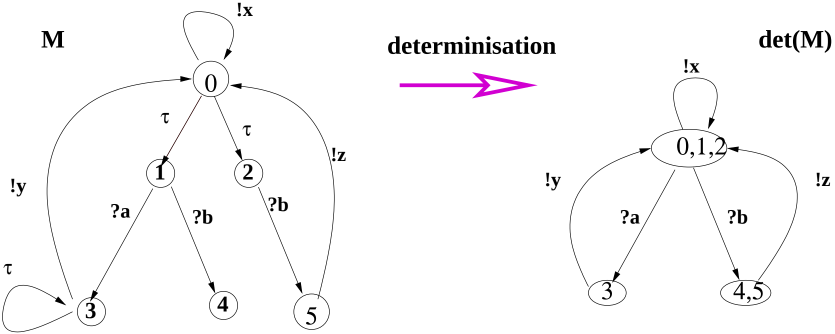
**Determinized automaton**

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

$$Traces(M) = Traces(det(M))$$

# Determinization: example



# 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:
$\forall \lambda \in A, q \overset{\lambda}{\Rightarrow}$
$M$ is (str/wk) $A$-complete if every state in $Q$ is $A$-complete.

**Proposition**

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

# Testing Models

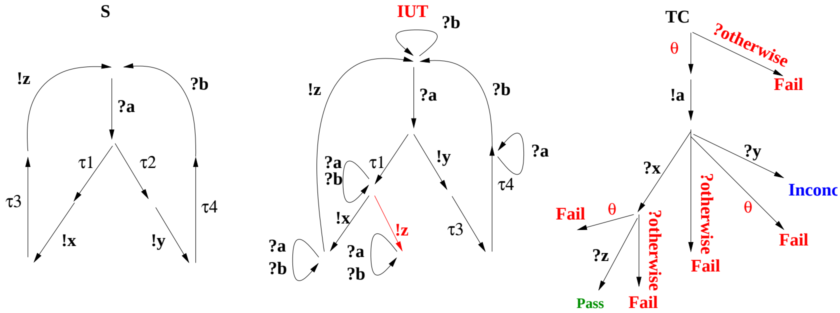
Specification : IOLTS $S = (Q^S, \Lambda_?, \Lambda_!, \mathcal{T}^S, \rightarrow_S, q_0^S)$

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

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

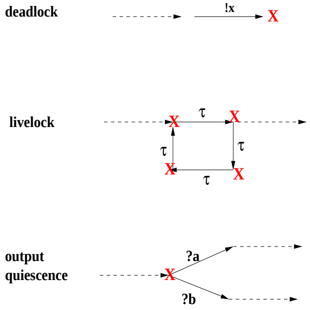
**Hyp.:** $I$ is weakly $\Lambda_?$-complete :
$\forall q \in Q^I, \forall a \in \Lambda_?, q \stackrel{a}{\Rightarrow}$.
In every state, $I$ accepts any input, possibly after internal actions (e.g. replies by an error).
NB: assumption required to avoid deadlocks in the interaction tester/IUT.

# Specification and implementation: example



# Observation of quiescence

In testing practice, one can observe traces of the $IUT$, but also its quiescences with timers.
Only $IUT$'s quisecences unspecified in $S$ should be rejected.




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

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

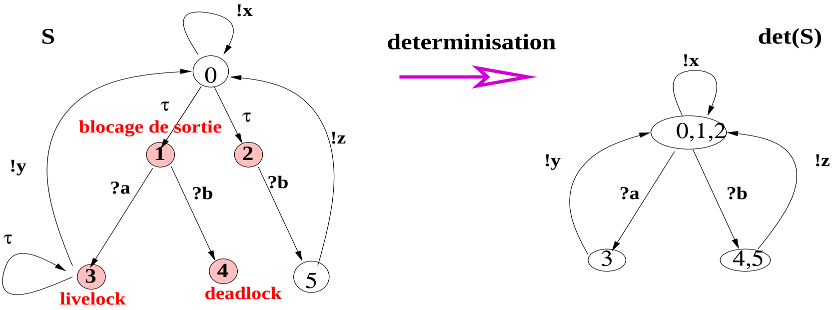
$q \in livelock(M) \triangleq \exists \tau_1, \ldots, \tau_n, q \xrightarrow{\tau_1 \cdots \tau_n} q$

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

$quiescent(M) = deadlock(M) \cup livelock(M) \cup outputlock(M)$

# Lost of quiescence by determinization

Determinization preserves trace equivalence but not quiescence.
A more refined equivalence is needed
$\Rightarrow$ quiescence must be explicited before determinization.



# Explicitation of quiescence

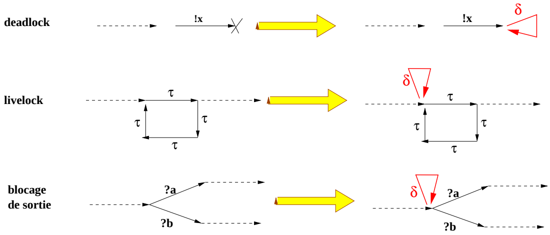
## Suspension

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 $\rightarrow_\delta = \rightarrow \cup \{(q, \delta, q) \mid q \in quiescent(M)\}$

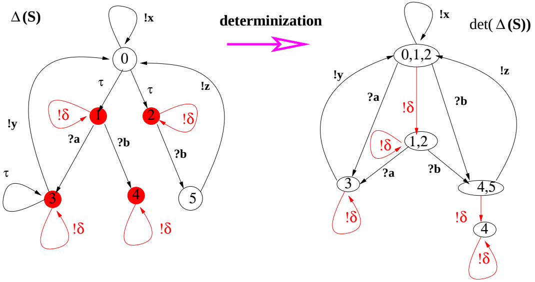
We note $\Lambda_!^\delta = \Lambda_! \cup \{\delta\}$

$\delta$ 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 $S = (Q^{\mathsf{S}}, \Lambda_?, \Lambda_!, \mathcal{T}^{\mathsf{S}}, \rightarrow_{\mathsf{S}}, q_0^{\mathsf{S}})$ be a specification and
$I = (Q^{\mathsf{I}}, \Lambda_?, \Lambda_!, \mathcal{T}^{\mathsf{I}}, \rightarrow_{\mathsf{I}}, q_0^{\mathsf{I}})$ an implementation with same interface

## Conformance

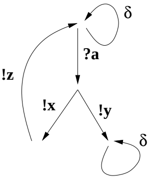
$$I \ \mathbf{ioco}\ S$$
$$\triangleq$$
$$\forall \sigma \in STraces(S), Out(\Delta(IUT) \ \text{ after } \ \sigma) \subseteq Out(\Delta(S) \ \text{ after } \ \sigma)$$

with $Out(P) \triangleq \Gamma(P) \cap \Lambda_!^{\delta}$ 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 $S$.

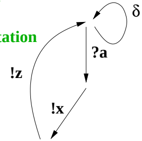
# ioco: example

**Specification**



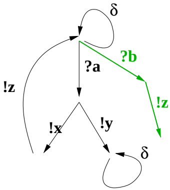

**IUT1**

**choix d'implementation**



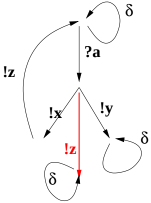

**IUT2**

**specification partielle**



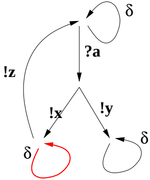

**IUT3**

**sortie interdite**




**IUT4**

**blocage interdit**



# Characterization of **ioco** in terms of $STraces$

**Proposition**

$$
\begin{aligned}
I \ \mathbf{ioco}\ S \quad & \iff \quad STraces(I) \cap [STraces(S).\Lambda_!^\delta] \subseteq STraces(S) && (1) \\
& \iff \quad STraces(I) \cap [STraces(S).\Lambda_!^\delta \setminus STraces(S)] = \emptyset && (2)
\end{aligned}
$$

$STraces(I)$= visible behaviors of $I$
$STraces(S)$ = visible behaviors of $S$
$STraces(S).\Lambda_!^\delta$ = visible behaviors of $S$ prolongated by output or $\delta$.

(1): $STraces$ of $I$ prolongating $STraces$ of $S$ by outputs or quiescences should remain $STraces$ of $S$.

(2): $I$ has no STrace which is an STrace of $S$ prolongated by an output or quiscence without being an STrace of $S$.

# Canonical tester of $S$

$Can(S) = (Q^c, \Lambda^c_? = \Lambda^\delta_!, \Lambda^c_! = \Lambda_?, \rightarrow_c, q^c_0, )$ equiped with $Fail \in Q^c$ built from $det(\Delta(S)) = (Q^d, \Lambda_?, \Lambda^\delta_!, \rightarrow^d, q^d_0)$ as follows :

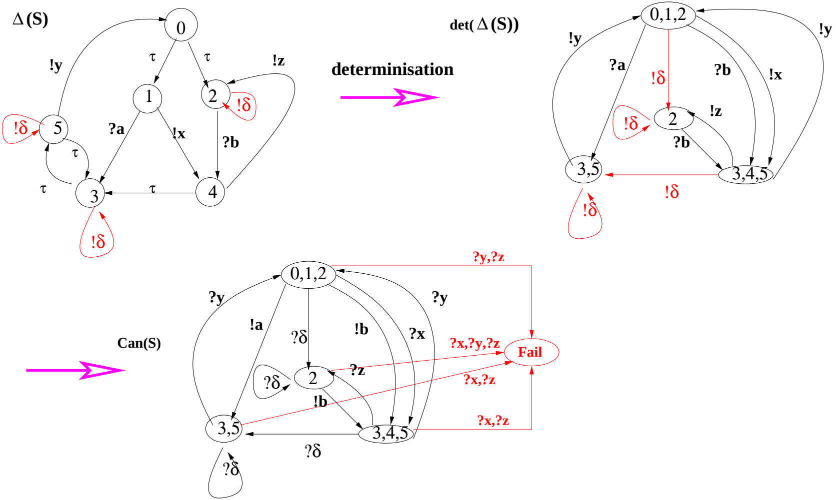
- $Q^c \triangleq Q^d \cup \{Fail\}$, $Fail \notin Q^d$ (new state $Fail$)
- $q^c_0 \triangleq q^d_0$
- $\rightarrow_c \triangleq \rightarrow_d \cup \{q \xrightarrow{a}_c Fail \mid q \in Q_d, a \in \Lambda^\delta_! = \Lambda^c_? \wedge \neg(q \xrightarrow{a}_d)\}$
  i.e. output completion to $Fail$ (notation $q \overset{?othw}{\longrightarrow}_c Fail$).

Language recognized by $Can(S)$:
$Traces_{Fail}(Can(S)) = STraces(S).\Lambda^\delta_! \setminus STraces(S)$

$I$ **ioco** $S \iff STraces(I) \cap Traces_{Fail}(Can(S)) = \emptyset$
i.e. $Can(S)$ is a non-conformance observer

**Rem.:** For a given $S$, $I$ **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 $TC = (Q^{\mathsf{TC}}, \Lambda_?^{\mathsf{TC}}, \Lambda_!^{\mathsf{TC}}, \rightarrow_{\mathsf{TC}}, q_0^{\mathsf{TC}})$ s.t. :

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

$$Traces_{\mathsf{Verdict}}(TC) =$$
$$Traces_{\mathsf{Fail}}(TC) \cup Traces_{\mathsf{Pass}}(TC) \cup Traces_{\mathsf{Inconc}}(TC)$$

# Test execution

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

$TC \| \Delta(I) = (Q^{\mathsf{TC}} \times Q^{\mathsf{I}}, \Lambda^{\mathsf{I}}, \rightarrow_{TC\|\Delta(I)}, (q_0^{\mathsf{TC}}, q_0^{\mathsf{I}}))$
where $\rightarrow_{TC\|\Delta(I)}$ is defined by:

$$\frac{tc \xrightarrow{a}_{\mathsf{TC}} tc' \quad q \xrightarrow{a}_{\Delta(I)} q' \quad a \in \Lambda^{\delta}_{\mathsf{VIS}}}{(tc, q) \xrightarrow{a}_{TC\|\Delta(I)} (tc', q')} \qquad \frac{q \xrightarrow{\tau}_{\Delta(I)} q' \quad \tau \in \mathcal{T}^{\mathsf{I}}}{(tc, q) \xrightarrow{\tau}_{TC\|\Delta(I)} (tc, q')}$$

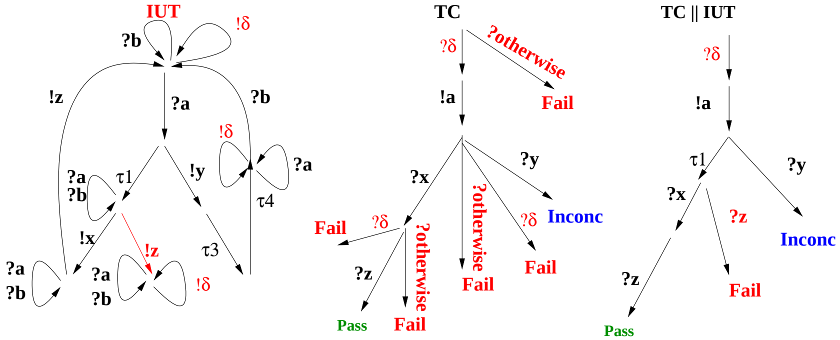
**Prop**

$Traces(TC \| \Delta(I)) = Traces(TC) \cap STraces(I)$

**Prop**

$I$ weakly $\Lambda_{?}$-complete and $TC$ $\Lambda^{\delta}_{!}$-complete (except in **Verdicts**)
$\Rightarrow$ $TC \| \Delta(I)$ is never blocked except in **Verdicts** states of $TC$

# Example


IUT
?b
!δ
!z
?a
?b
!δ
?a
?b
τ1
!y
?a
τ4
!x
!z
τ3
?a
?b
?a
?b
!δ
TC
?δ
?otherwise
!a
Fail
?x
?y
?otherwise
?δ
Inconc
Fail
?δ
?otherwise
Fail
Fail
?z
Fail
Pass
Fail
TC || IUT
?δ
!a
τ1
?y
?x
?z
Inconc
?z
Fail
Pass

# Verdicts

We say that a test case $TC$ rejects (*fails*) $I$ iff an execution of $TC \| \Delta(I)$ reaches **Fail**
This expresses a *possibility* for rejection.

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

! due to non-controlable choices of $I$, a single test case applied on a single $IUT$ can produce all different verdicts !
See the preceding example where
$STraces(I) \cap Traces_{\mathbf{Fail}}(TC) \neq \emptyset$ and
$STraces(I) \cap Traces_{\mathbf{Pass}}(TC) \neq \emptyset$ and
$STraces(I) \cap Traces_{\mathbf{Inconc}}(TC) \neq \emptyset$

# 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:

$$TC \text{ sound}/S \quad \triangleq \quad \forall I, [TC \text{ fails } I \Rightarrow \neg(I \textbf{ ioco } S)]$$

$$( \iff \quad \forall I, [I \textbf{ ioco } S \Rightarrow \neg(TC \text{ fails } I)] \,)$$

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

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

# Necessary and sufficient condition for soundness

Remember that

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

This implies the necessary and sufficient condition :

**Proposition**

$$
\begin{aligned}
TC \text{ sound}/S &\iff Traces_{\text{Fail}}(TC) \subseteq Traces_{\text{Fail}}(Can(S)) \\
TS \text{ sound}/S &\iff \bigcup_{TC \in TS} Traces_{\text{Fail}}(TC) \subseteq Traces_{\text{Fail}}(Can(S))
\end{aligned}
$$

# 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:

$$TS \text{ exhaustive} \triangleq \forall I, [\neg(I \textbf{ ioco } S) \Rightarrow [\exists TC \in TS, TC \text{ fails } I]]$$

$$(\iff \forall I, [[\forall TC \in TS, \neg(TC \text{ fails } I)] \Rightarrow I \textbf{ ioco } S])$$

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

## Proposition

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

# Canonical tester / soundness and exhaustiveness

**Proposition**

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

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

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

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

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

# Outline

# Test case selection

Objective : Find an algorithm that, for **ioco** and for a given $S$, produces a test suite $TS$ which is both

- sound (easy to obtain from $Can(S)$)
- limit-exhaustive i.e. by considering the infinite suite of test cases that can be produced

Two techniques :

1. Non-deterministic selection (à la TorX)
2. Selection guided by a test purpose (à la TGV)

# Non-deterministic selection: simplified view

## Algorithm

After any trace $\sigma$ in $Can(S)$

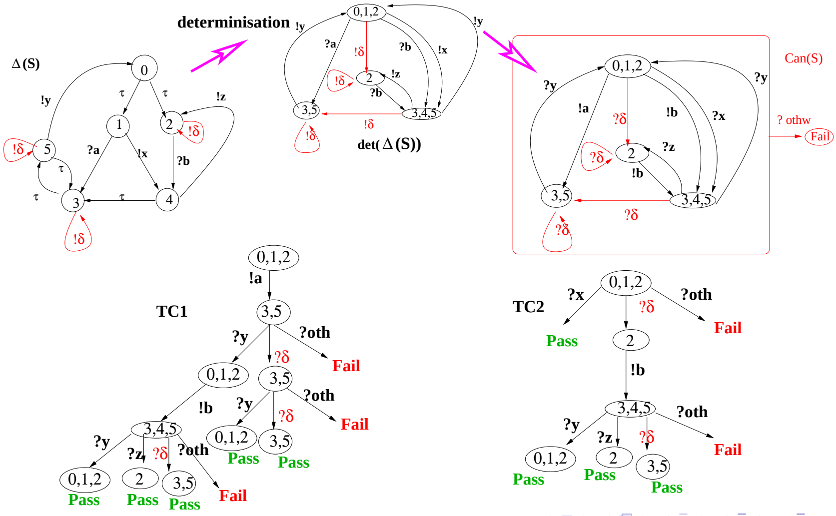
- emit a Fail verdict if $Can(S)$ after $\sigma \subseteq$ **Fail**
- otherwise make a choice between
  - stop and produce a Pass verdict,
  - observe an output or quiescence of $I$ by an input of $Can(S)$ after $\sigma$ and continue.
  - choose one output among those of $Can(S)$ after $\sigma$, emit it to $I$ and continue.

## Properties

Test suite $TS$= all finite and controlable unfoldings of $Can(S)$.
$TS$ is sound and exhaustive:

$$\bigcup_{TC \in TS} Traces_{\mathrm{Fail}}(TC) = Traces_{\mathrm{Fail}}(Can(S))$$

# Example



# Test selection guided by a test purpose

Main difference with preceding algorithm:

- test selection by test purposes.
- off-line selection, *a posteriori* execution

Test purpose: describes a set of behaviors to be tested, targetted by a test case.
$\Rightarrow$ modelling with reachability observers (accepted language)
$\Rightarrow$ "model-checking"-like selection algorithms.

# Test purpose

## Test purpose

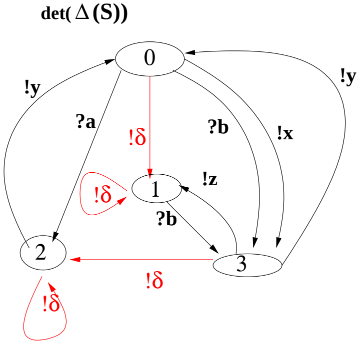
IOLTS $TP = (Q^{\mathrm{TP}}, \Lambda^{\delta}_{VIS}, \rightarrow_{\mathrm{TP}}, q_0^{\mathrm{TP}})$,
deterministic and complete for $\Lambda^{\delta}_{VIS}$
(complete: $\forall q, \forall a \in \Lambda^{\delta}_{VIS}, q \xrightarrow{a}_{\mathrm{TP}}$),
equipped with a set $Accept^{\mathrm{TP}}$ of trap states
(trap: $\forall a \in \Lambda^{\delta}_{VIS}, Accept \xrightarrow{a}_{\mathrm{TP}} Accept$).

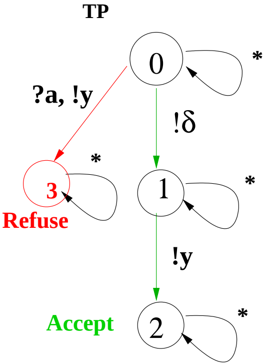
$Traces_{\mathrm{Accept}}(TP) = L_{\mathrm{Accept}}(TP)$ is "extension-closed" and
$Traces(TP) = L(TP) = (\Lambda^{\delta}_{\mathrm{VIS}})^*$

$TP$ can be seen as an observer of a reachability property.

**Rem:** if Accept is interpreted as Bad, $TP$ models the negation of a safety property $\Rightarrow$ different interpretation of verdicts.

# Test purpose example



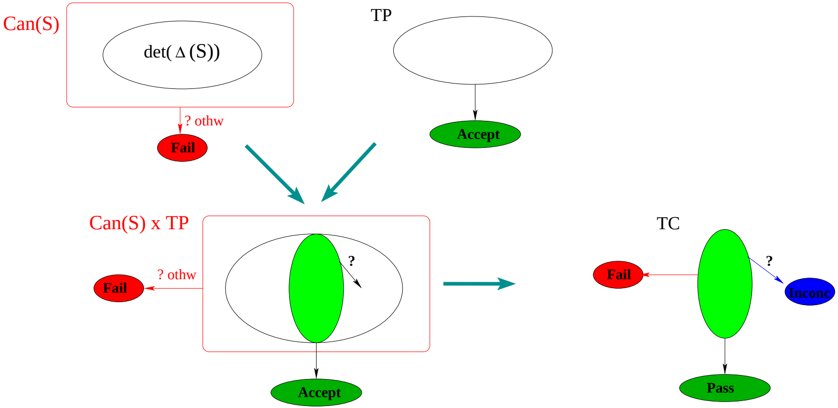
# Selection principle

Generate test cases which are both

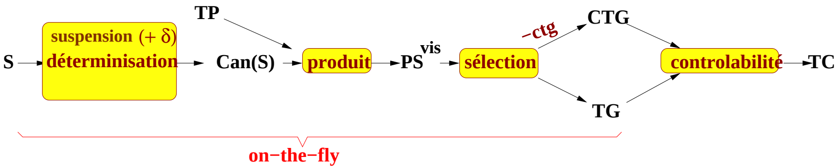
- non-conformance observers
  $Traces_{\text{Fail}}(TC) \subseteq Traces_{\text{Fail}}(Can(S))$
  (implies soundness)
- reachability observers
  $Traces_{\text{Pass}}(TC) \subseteq Traces_{\text{Accept}}(TP)$

while focusing on accepted traces $STraces(S) \cap Traces_{\text{Accept}}(TP)$
(or $Traces(Can(S)) \cap Traces_{\text{Bad}}(TP)$)

# Selection scheme



# Operations for selection



# Synchronous product

Let $M_1 = (Q^1, A, \rightarrow_1, q_0^1)$ equipped with $F_1$
and $M2 = (Q^2, A, \rightarrow_2, q_0^2)$ equipped with $F_2$
be two (IO)LTS with same alphabet $A$.
The synchronous product of $M_1$ and $M_2$ is the (IO)LTS
$M_1 \times M_2 = (Q^1 \times Q^2, A, \rightarrow, (q_0^1, q_0^2))$ equipped with $F_1 \times F_2$
where $\rightarrow$ is defined by the rule:

$$\frac{q_1 \xrightarrow{a}_1 q_1' \quad q_2 \xrightarrow{a}_2 q_2'}{(q_1, q_2) \xrightarrow{a} (q_1', q_2')}$$

We get:

$L(M_1 \times M_2) = L(M_1) \cap L(M_2)$ and
$L_{F_1 \times F_2}(M_1 \times M_2) = L_{F_1}(M_1) \cap L_{F_2}(M_2)$.

# Synchronous product $Can(S) \times TP$

$Can(S) = (Q^c, \Lambda^{\delta}_{\mathrm{VIS}}, \rightarrow_c, q^c_0, )$ equipped with $Fail \subseteq Q^c$ and
$TP = (Q^{\mathrm{TP}}, \Lambda^{\delta}_{\mathrm{VIS}}, \rightarrow_{\mathrm{TP}}, q^{\mathrm{TP}}_0)$ equipped with $Accept_{\mathrm{TP}} \subseteq Q^{\mathrm{TP}}$
are two IOLTS with same alphabet $\Lambda^{\delta}_{\mathrm{VIS}}$.

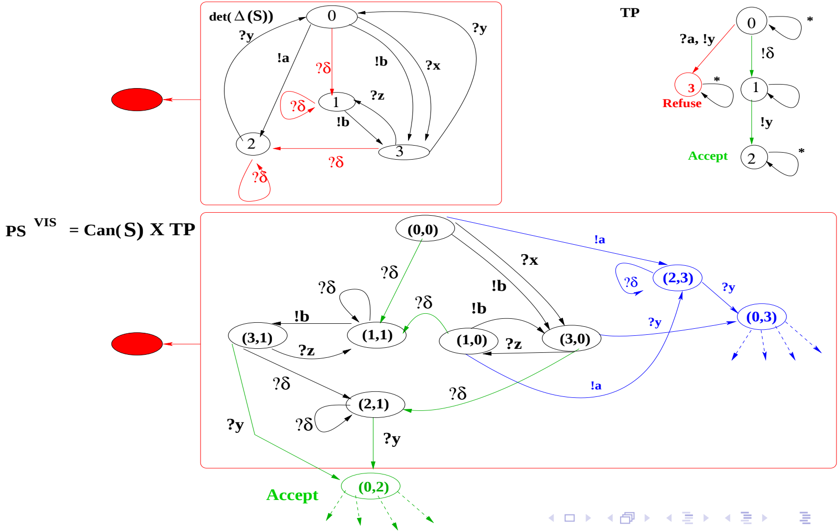
Let $PS^{\mathrm{VIS}} = Can(S) \times TP$ equipped with the state sets

- $Accept_{\mathrm{VIS}} = Q_c \setminus \{Fail\} \times Accept_{\mathrm{TP}}$ and
- $Fail_{\mathrm{VIS}} = \{Fail\} \times Q^{\mathrm{TP}}$

$TP$ being complete, we have $Traces(TP) = (\Lambda^{\delta}_{\mathrm{VIS}})^*$, thus

$$Traces(PS^{\mathrm{VIS}}) = Traces(Can(S))$$
$$Traces_{\mathrm{Accept}}(PS^{\mathrm{VIS}}) = STraces(S) \cap Traces_{\mathrm{Accept}}(TP)$$
$$Traces_{\mathrm{Fail}}(PS^{\mathrm{VIS}}) = Traces_{\mathrm{Fail}}(Can(S))$$

# Example



det( Δ (S))
0
1
2
3
?y
!a
?δ
!b
?x
?z
TP
?a, !y
!δ
Refuse
Accept
*
PS VIS = Can(S) X TP
(0,0)
(1,1)
(3,1)
(2,1)
(1,0)
(3,0)
(2,3)
(0,3)
(0,2)

# Selection

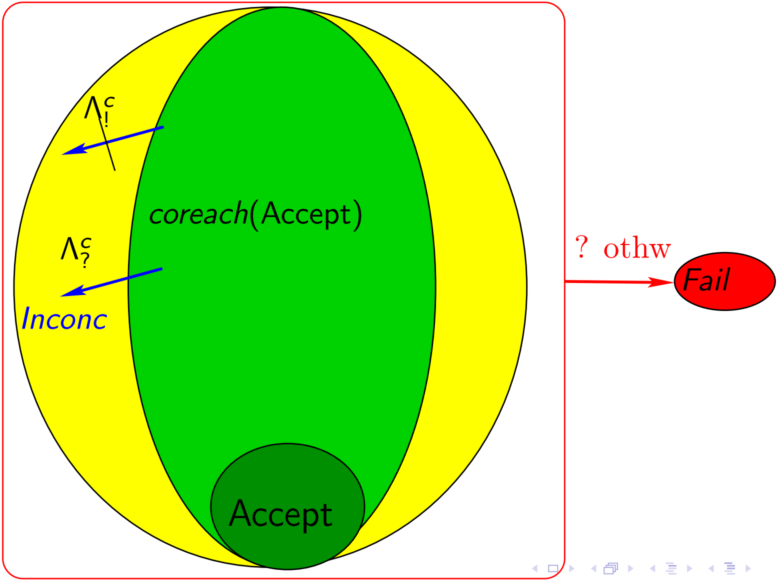
Objective: Extract from $PS^{\mathsf{VIS}}$ a test case with adequate verdicts:

- **Fail** : detect $STraces(S).\Lambda_!^\delta \setminus STraces(S)$
  We have $Traces_{\mathsf{Fail}}(PS^{\mathsf{VIS}}) = Traces_{\mathsf{Fail}}(Can(S))$
- Pass : detect $STraces(S) \cap Traces_{\mathsf{Accept}}(TP)$
  We have $Traces_{\mathsf{Accept}}(PS^{\mathsf{VIS}}) = STraces(S) \cap Traces_{\mathsf{Accept}}(TP)$
  Thus Pass $= Accept_{\mathsf{VIS}}$
- **Inconc** : detect
  $Rtraces(PS^{\mathsf{VIS}}) \triangleq STraces(S) \setminus pref_{\leq}(Traces_{\mathsf{Accept}}(PS^{\mathsf{VIS}}))$
  i.e. suspension traces of $S$ that are not prefix of accepted traces.

Observations :

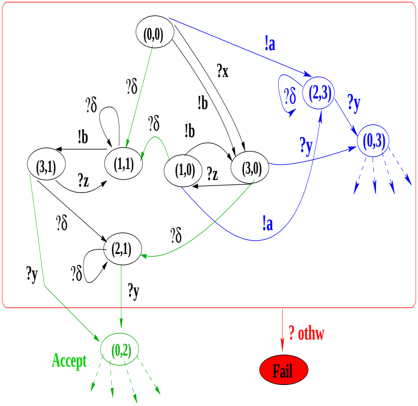
- Controlable inputs can be cut.
- $pref_{\leq}(Traces_{\mathsf{Accept}}(PS^{\mathsf{VIS}})) = Traces_{coreach(Accept_{\mathsf{VIS}})}(PS^{\mathsf{VIS}})$
  $\Rightarrow$ analysis of co-reachability to $Accept_{\mathsf{VIS}}$

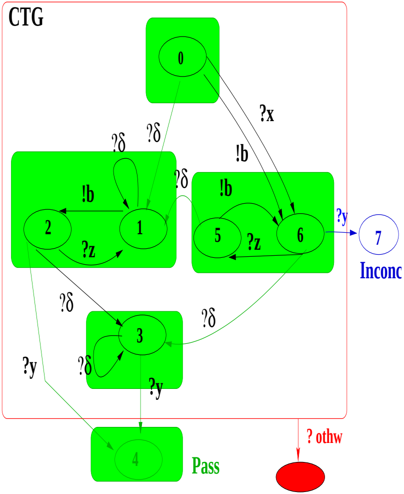
# Illustration



# Extraction of the complete test graph: illustration

$PS^{VIS} = Can(S) \times TP$



# Reachability and co-reachability

$post_B(P) = \{q' \in Q^{\mathrm{M}} \mid \exists b \in B, \exists q \in P,\ q \xrightarrow{b}_{\mathrm{M}} q'\}$
i.e. immediate successors of $P$ by actions in $B$

$reach_B(P) = \mu X.P \cup post_B(X) = \bigcup_{i \geq 0} post^i_B(P)$
i.e. reachable from $P$ by actions in $B$

$pre_B(P) = \{q' \in Q^{\mathrm{M}} \mid \exists b \in B, \exists q \in P,\ q' \xrightarrow{b}_{\mathrm{M}} q\}$
i.e. immediate predecessors of $P$ by actions in $B$,

$coreach_B(P) = \mu X.P \cup pre_B(X) = \bigcup_{i \geq 0} pre^i_B(P)$
i.e. co-reachable from $P$ by actions in $B$.

# Complete test graph (CTG): definition

Let $PS^{\mathsf{VIS}} = Can(S) \times TP = (Q^{\mathsf{VIS}}, \Lambda^{\delta}_{\mathsf{VIS}}, \rightarrow_{\mathsf{VIS}}, q_0^{\mathsf{VIS}})$, equipped with $Accept_{\mathsf{VIS}}$ and $Fail_{\mathsf{VIS}}$.

The complete test graph is the IOLTS $CTG = (Q^{\mathsf{VIS}}, \Lambda^{\delta}_{\mathsf{VIS}}, \rightarrow_{\mathsf{CTG}}, q_0^{\mathsf{VIS}})$ equipped with **Pass** $\triangleq Accept_{\mathsf{VIS}}$, **Inconc** $\subseteq Q^{\mathsf{VIS}}$ and **Fail** $= \mathbf{Fail}_{\mathsf{VIS}}$,

where **Inconc** and $\rightarrow_{\mathsf{CTG}}$ are defined by the rules:

## Keep

$$\frac{q \in coreach(Accept_{\mathsf{VIS}}) \quad q' \in coreach(Accept_{\mathsf{VIS}}) \cup \{\mathbf{Fail}_{\mathsf{VIS}}\} \qquad q \xrightarrow{\alpha}_{\mathsf{VIS}} q' \qquad \alpha \in \Lambda^{\delta}_{\mathsf{VIS}}}{q \xrightarrow{\alpha}_{\mathsf{CTG}} q'}$$

## Inconc

$$\frac{q \in coreach(Accept_{\mathsf{VIS}}) \quad q' \notin (coreach(Accept_{\mathsf{VIS}}) \cup \{\mathbf{Fail}_{\mathsf{VIS}}\}) \qquad q \xrightarrow{\alpha}_{\mathsf{VIS}} q' \qquad \alpha \in \Lambda^{\delta}_{!}}{q \xrightarrow{\alpha}_{\mathsf{CTG}} q' \qquad q' \in \mathbf{Inconc}}$$

# Trace properties of CTG

$$Traces_{\text{Pass}}(CTG) = STraces(S) \cap Traces_{\text{Accept}}(TP)$$

*pass* is produced on any suspension trace of $S$ accepted by $TP$.

$$Traces_{\text{Inconc}}(CTG) = [STraces(S) \cap pref_{\leq}(Traces_{\text{Accept}}(TP))].\Lambda_!^\delta \\ \cap STraces(S) \setminus pref_{\leq}(Traces_{\text{Accept}}(TP))$$

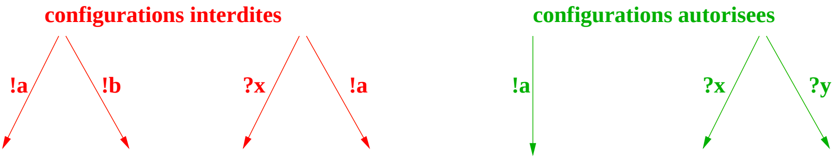
*Inconc* is produced on any suspension trace of $S$ which last action is an output or $\delta$ that cannot lead to the satisfaction of $TP$.

$$Traces_{\text{Fail}}(CTG) = \\ [STraces(S) \cap pref_{\leq}(Traces_{\text{Accept}}(TP))].\Lambda_!^\delta \setminus STraces(S)$$

$\Rightarrow Traces_{Fail}(CTG) \subseteq Traces_{Fail}(Can(S))$ (NSC for soundness)
*Fail* is produced on any suspension trace of $S$ which is a prefix of a trace leading to *Accept*, prolongated with an unspecified output of $S$.

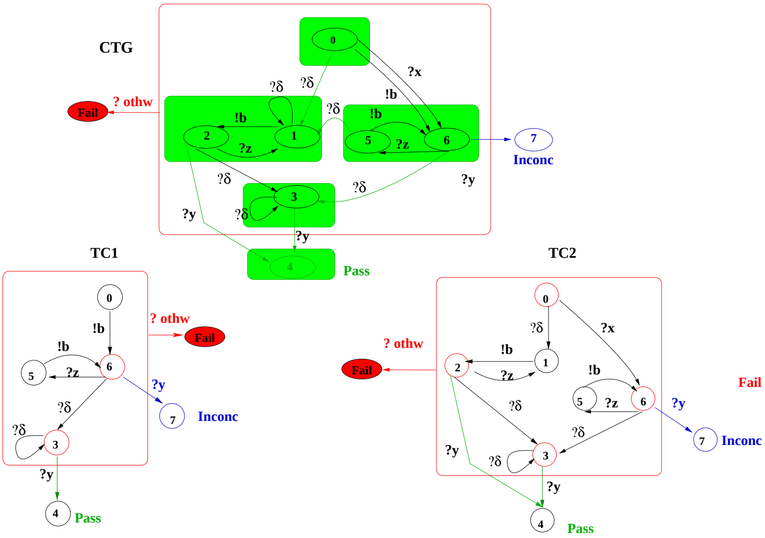
# Controlability conflits: test case computation

In practice, a test case should be controlable :




**Algorithm:** backward traversal of $CTG$ from Pass states to initial states and conflicts pruning.
Adapted from $coreach(Pass)$.
$TC$ is a sub-IOLTS of $CTG$ thus
$Traces_{\mathsf{Fail}}(TC) \subseteq Traces_{\mathsf{Fail}}(CTG) \subseteq Traces(Can(S))$
$\Rightarrow$ soundness is preserved

# Pruning: example



# 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:** $Traces_{\mathrm{Fail}}(TC) \subseteq Traces_{Fail}(Can(S))$
$\Rightarrow TC$ is sound.
**Exhaustiveness:** if $\neg(I$ **ioco** $S)$ then $\exists \sigma \in STraces(S), \exists x \in \Lambda_!^\delta, x \in Out(\Delta(I)$ after $\sigma) \wedge x \notin Out(\Delta(S)$ after $\sigma)$.
We have $\exists y \in Out(\Delta(S)$ after $\sigma)$, (as $Out(\Delta(S)$ after $\sigma) \neq \emptyset$).
Let $\sigma' = \sigma.y(\in STraces(S))$ and $TP = \sigma'$ finished by $Accept$.
By definition, $CTG$ after $\sigma.x \in$ **Fail**. Prune $CTG$ in $TC$ s.t. $\sigma.x \in Traces(TC) \implies TC$ after $\sigma.x \in$ **Fail**. Thus $TC$ fails $IUT$.

# Outline

# Conclusion

- Testing theory for IOLTS
- Non-deterministic selection: unfolding of $Can(S)$
- Selection by test purpose: for finite IOLTSs based on co-reachability analysis.

Problems:

- (partial) enumeration of the set of reachable states.
- Infinite state models: enumeration is impossible.

Example: models with data (IOSTS).
Analysis is undecidable $\Rightarrow$ approximate analysis.