On-the-fly Test Synthesis with TGV

Thierry Jéron
Irisa / Inria Rennes
http://www.irisa.fr/pampa,
e-mail: Thierry.Jeron@irisa.fr

Plan

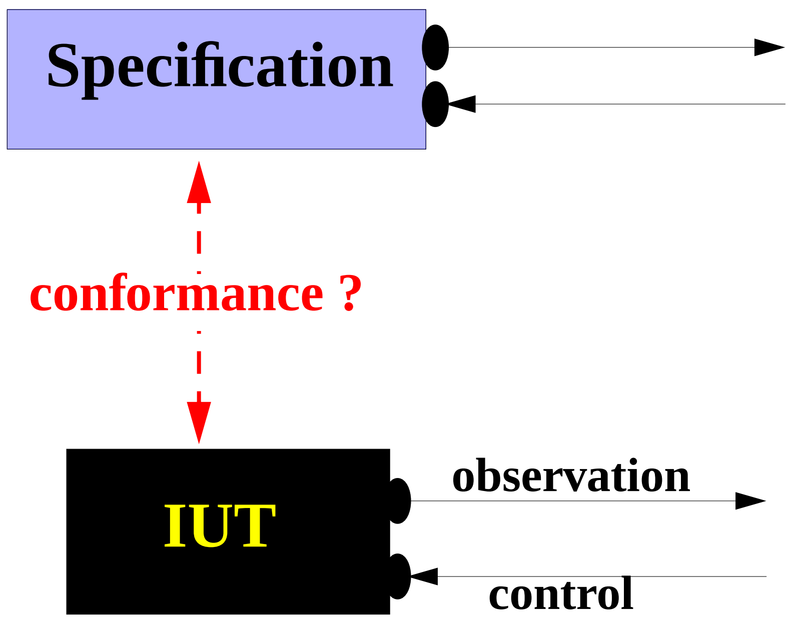
- 1. Conformance Testing
- 2. The TGV project
- 3. Experiments and Industrial Transfer
- 4. Ongoing Work in Testing

1. Conformance Testing

Testing problem: check if an implementation under test (IUT) of a reactive system conforms (or not) to its specification.

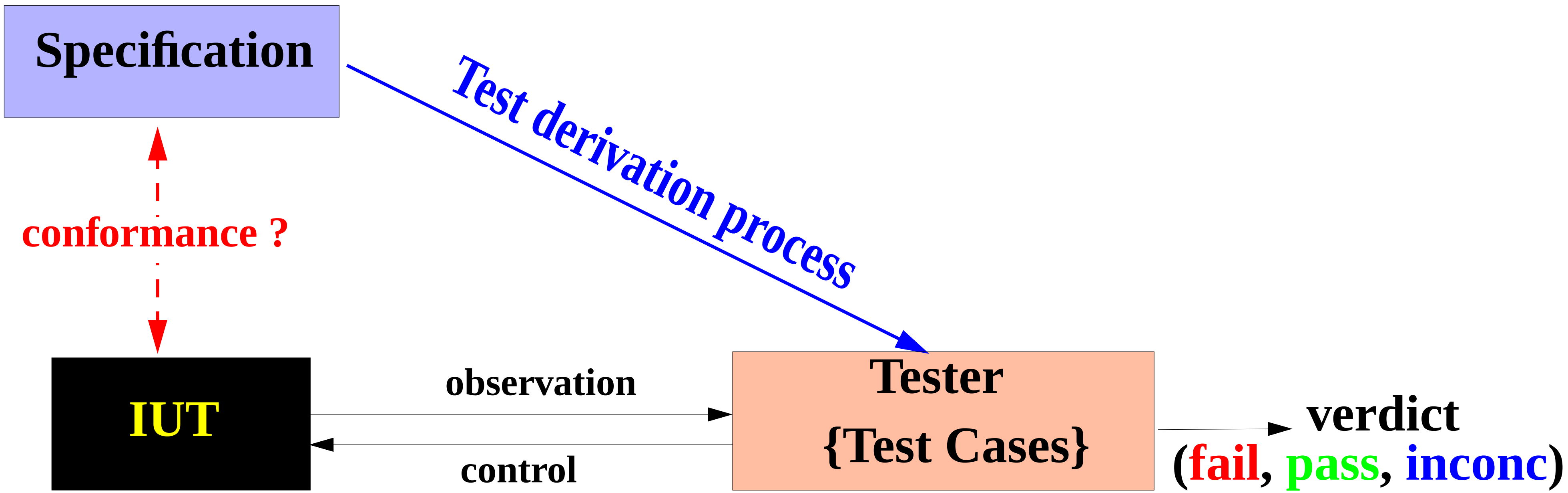
Black box testing: the source code of the IUT is unknown, only the interface can be controlled and observed by the tester.

.



Conformance testing

o **Practice:** derive a set of test cases from the specification,
implement test cases in a tester,
try to find errors (test cases serve as oracles) or gain confidence.



Industrial Practice

o **Manual conception of test suites from informal specifications**

- long and repetitive process,
  up to 30% of the cost of development process
- subject to errors : up to 20%
- no clear definition of conformance
- maintenance of test suites is difficult

$\Longrightarrow$ **Automation of test generation from formal specifications can be profit earning**

Conformance testing of protocols

o **Telecom is governed by standards**:

- **Formal description techniques:** Estelle, Lotos, SDL
- **ISO 9646:** Conformance Testing Methodology and Framework
- **-> Test description langages:** TTCN (Tree and Tabular Combined Notation), MSC (Message Sequence Charts),
- **Standardized protocols** (in SDL in general)
- **Standardized test suites**

o **Difficulties for automation**

- asynchronous communication
- non-determinism
- specificities of different levels: low level -> control
  high level -> data
- large and detailed specifications
- constraint to produce test cases similar to manual ones

Structure of test cases (TTCN)

- **Test Purpose:** goal of the test case

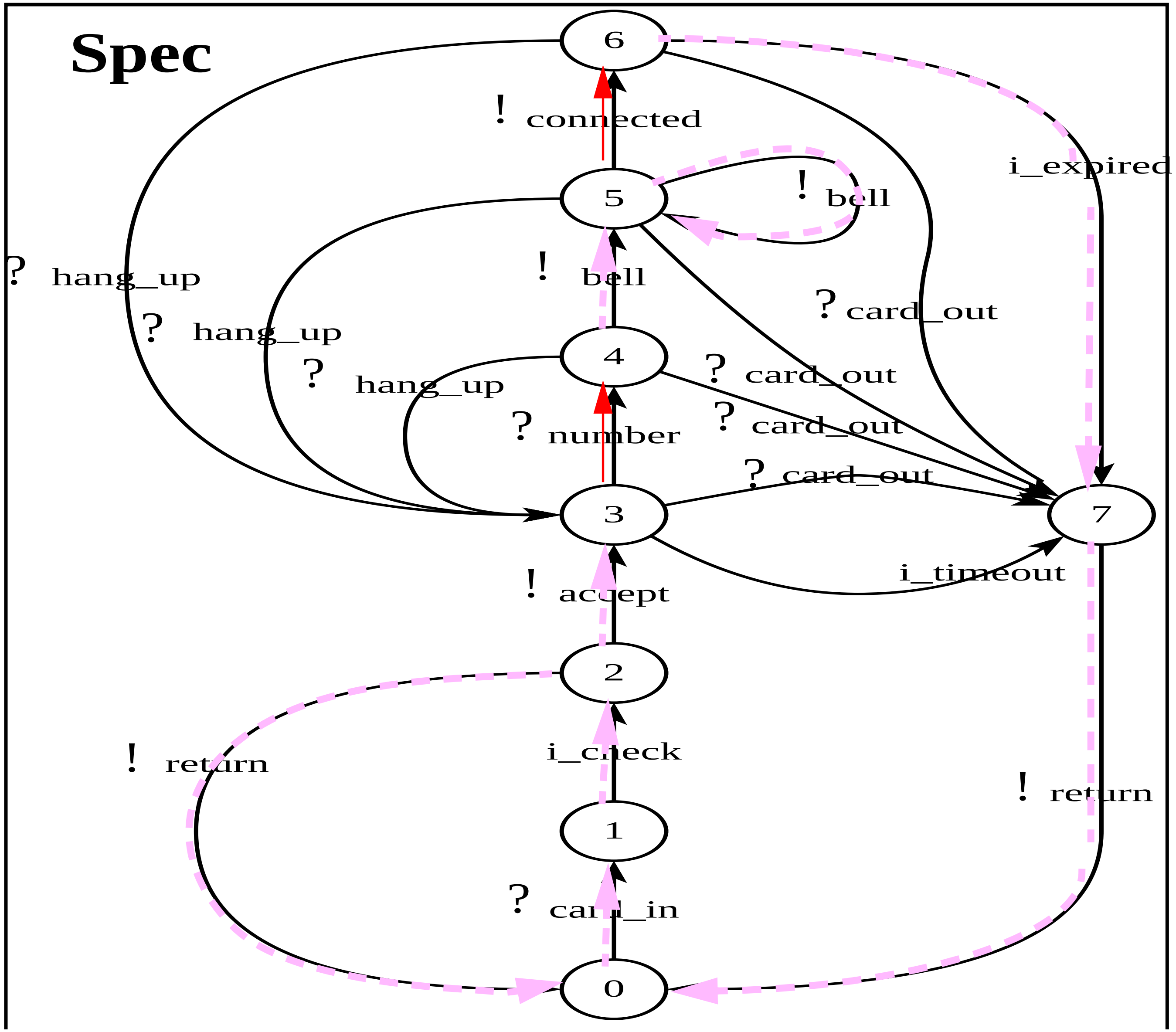
- **Declarations** (types, PCOs), **constraints** (variables and message parameters).

- **Behavior:** reactive program played by the Tester against the IUT
  - Preamble: leads to the initial state of the test purpose
  - Test body: checks the test purpose
  - Postamble: back to a stable state or initial state after a verdict.
  - Timers: observation of quiescence of the IUT.

- **Verdicts:**
  - **FAIL:** rejection (unauthorized timeout or unspecified input)
  - **(PASS):** Test Purpose reached, **PASS:** and back to a stable state
  - **INCONCLUSIVE:** specified input but Test Purpose not reachable

Test cases are re-run until a Fail or Pass verdict is reached

Example: A Simplified Phone Box

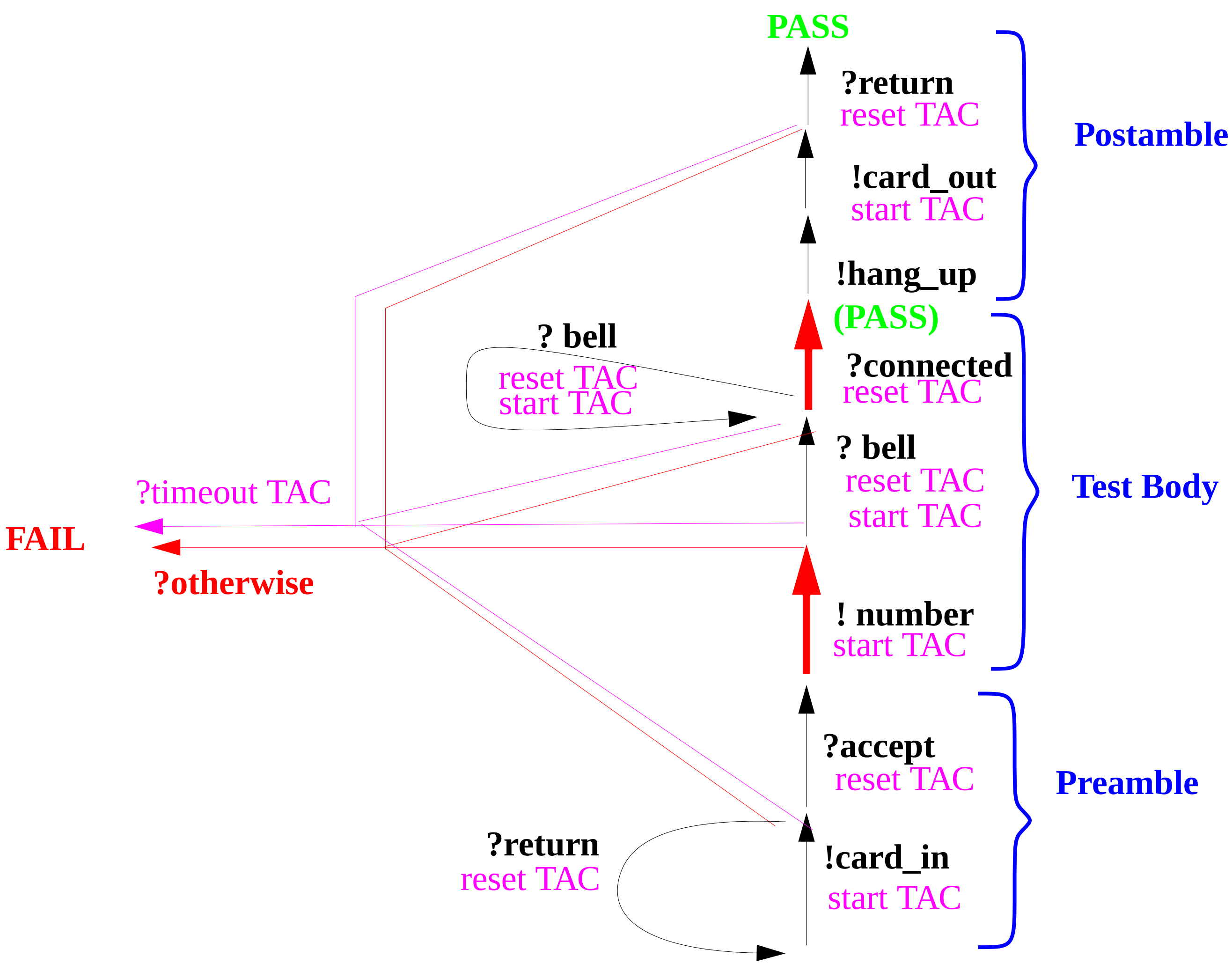


Test Purpose:

number and

later connect

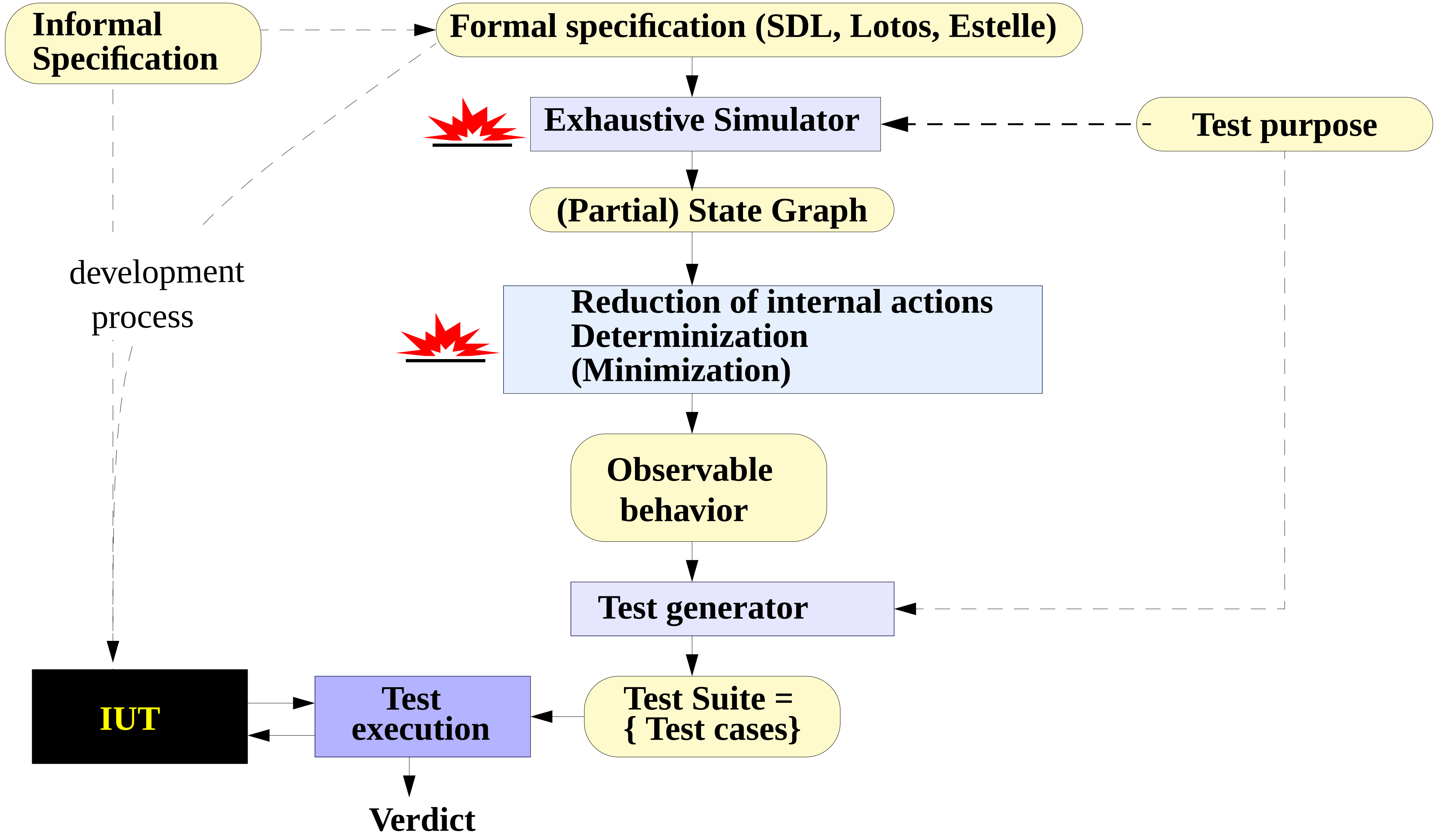
Expected test case



Work needed

- extraction of behavior
- elimination of internal actions
- output freedom of S
- mirror image
- timers management

Automatic Test Synthesis



Automata Theoretic Methods

o origin: hardware testing

o models: Mealy machines: $s \xrightarrow{i/o} s'$

o fault model: output: $s \xrightarrow{i/o'} s'$, transfer: $s \xrightarrow{i/o} s''$

o hypothesis : Spec: input complete, deterministic, minimal, strongly connected
IUT: input complete, deterministic, minimal (or < k), strongly connected

o test generation: one test case per transition: $s \xrightarrow{\text{apply } i/ \text{ check } o} \text{check } s'$
test suite: minimal length sequence with all elementary test cases
algorithms: traveling salesman, flow graphs, linear programming

o different methods: TT, DS, UIO, W, etc, differ on checking sequences

o theoretical results: correction and exhaustivity for a fault model + hypothesis.

- strong hypothesis, algorithmic complexity, treatment of non determinism

\+ completeness, checking sequences

Methods based on Labelled Transition Systems

o origin: testing theory, canonical tester.

o models: LTS (not well adapted) or IOLTS: $s \xrightarrow{?i} s' \xrightarrow{!o} s''$

o fault model <- > conformance relation between IUT and Spec
difference between possible observations of IUT and Spec after same traces

o hypothesis : Spec: no hypothesis, IUT: input complete

o test generation: graph traversal algorithms, model-checking:
- random synthesis (Twente) and on-the-fly execution
- on-the-fly synthesis guided by a test purpose (TGV)

o theoretical results:
unbias: only non conformant IUT may be rejected
exhaustiveness: all non conformant IUT may be rejected.

- no checking sequences

+ weak hypothesis, performant on-the-fly algorithms, test structure

2. The TGV project
(Test Generation with Verification technology)

- o Joint project since 94: Verimag Grenoble - Irisa Rennes
- o **Goal**: using on the fly model-checking techniques
  for efficient test case synthesis for conformance testing.
- o **Participants**:
  - **Irisa Rennes:**
    Researchers: T. Jéron, C. Jard, V. Rusu, C. Viho
    Engineers: H. Kahlouche (Montréal), S. Simon, S. Ramangalahy
    Ph. D.: P. Morel, L. Nedelka, + training students
  - **Verimag Grenoble:**
    Researchers: J.-C. Fernandez (-> LSR), A. Kerbrat (Telelogic),
    J. Sifakis,
    Ph. D. : M. Bozga, L. Ghirvu
  - **Inria Grenoble:** assistance of H. Garavel for CADP

TGV main characteristics

- **Sound testing theory**: based on IOLTS and adapted from works of Brinksma and Tretmans (Univ. Twente) and Phalippou (Cnet Lannion)
- **Algorithms:** on-the-fly model-checking
  lazzy construction of a partial state graph guided by a test purpose
- **Test quality:** comparable with manual ones, minimize Inconclusive verdicts, unbias and (theoretical) exhaustiveness
- **Langage independant:** same source code for SDL, Lotos,UML
  produces TTCN.
- **Case studies** in different application domains: protocols, hardware, embedded systems.
- **Distribution:** free version available in the CADP toolbox.
- **Industrial transfert:** TestComposer (Verilog/Telelogic)

TGV functionalities

o Test generation:

Specification

Test Purpose

TGV

Abstract Test Case

o Generation of the complete test graph:

Specification

Test Purpose

TGV -csg

Test graph (graph of all test cases for TP)

Abstract Test Cases

o Verification of (manual) test cases

Specification

Abstract Test Case

TGV_VTS

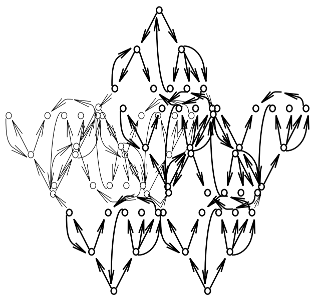
Corrected/Refined Abstract Test Case

Test Generation in TGV

coherency (preorder)

Specification S

behavior modelled by an IOLTS



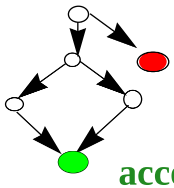
conformance
ioco

Black box
Implementation I

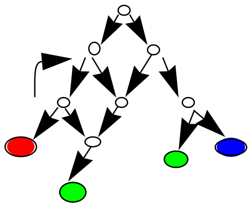
Test purpose TP

automata

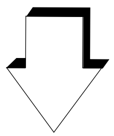
for test selection



Test case TC



Test execution on the IUT



Verdict

Properties: unbias : verdict(exec(TC || I) = fail => not (I ioco S)

exhaustivity: if I is fair, not(I ioco S) => exists TP s.t. verdict(exec(TC(S,TP) || I)) = fail

Models: IOLTS

o Transition systems with three kinds of transitions:

- input: $s \xrightarrow{?x} s'$, output: $s \xrightarrow{!a} s'$,

  internal action: $s \xrightarrow{\tau} s'$

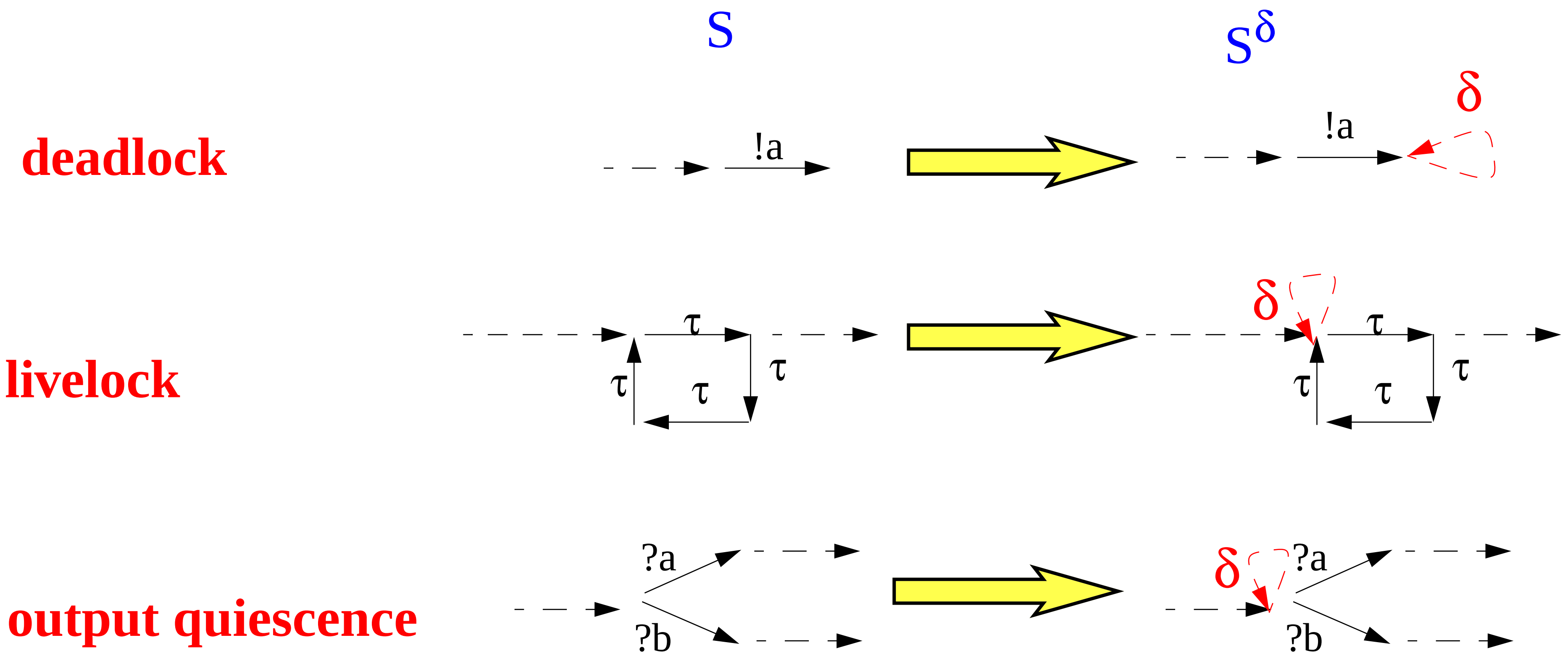
o Modelisation facilities:

- non-determinism (automata): $\xrightarrow{?x} s$, $s \xrightarrow{!a} s'$, $s \xrightarrow{!a} s''$

- observable "non-determinism": $\xrightarrow{?x} s$, $s \xrightarrow{!a} s'$, $s \xrightarrow{!b} s''$

Quiescence

- quiescence (absence of reaction) of a reactive system is observable by testing by the use of timers
- possible quiescence must be computed on the specification

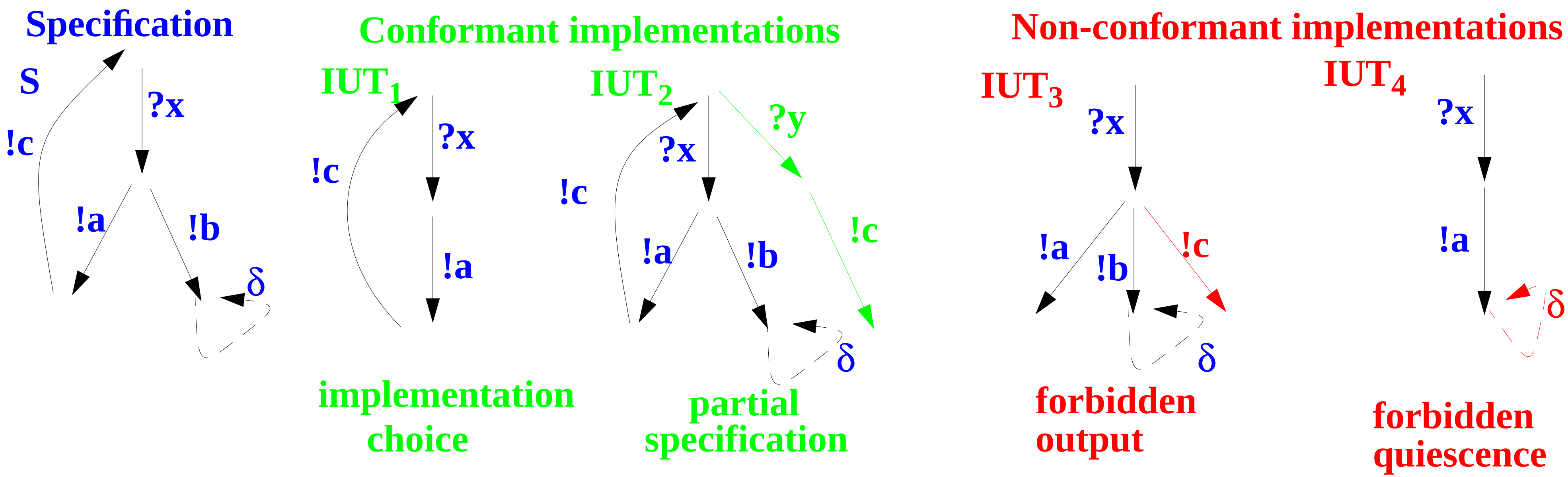


Conformance relation

I ioco S

iff

$\forall \sigma \in \textbf{traces}(\mathbf{S}^{\delta}),\ \mathrm{Out}(\mathbf{I}^{\delta}\ \textbf{after}\ \sigma) \subseteq \mathrm{Out}(\mathbf{S}^{\delta}\ \textbf{after}\ \sigma)$



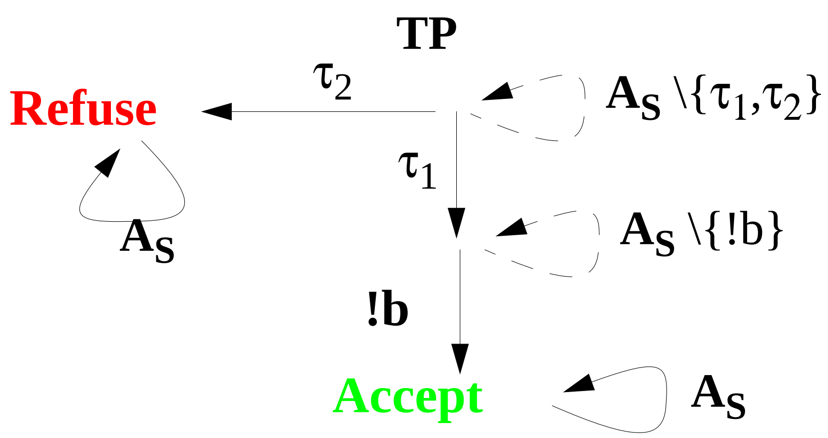
Test purposes

Automata used to select behaviors of the specification:

- Observable and internal actions (-> useful for testing in context)
- Complete (implicitly => abstraction)
- Two distinguished sets of trap states: Accept and Refuse

Pass <----> Accept

Fail <--/--> Refuse (traversal cut)



Test purpose accepting sequences with τ_1 followed later by !b

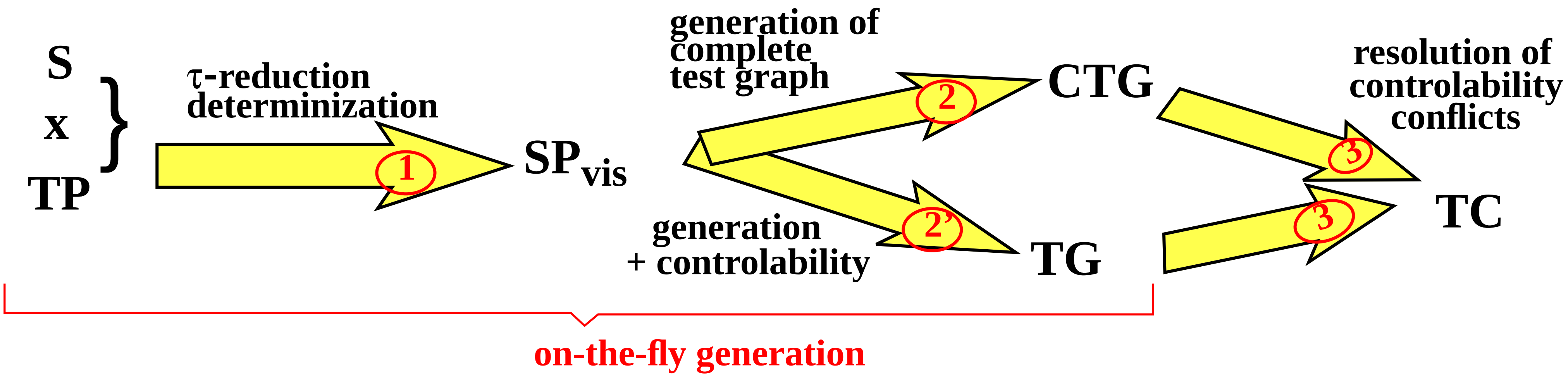
Refuse allows to cut sequences with a τ_2 before any $\tau 1$

Principle of on-the-fly generation

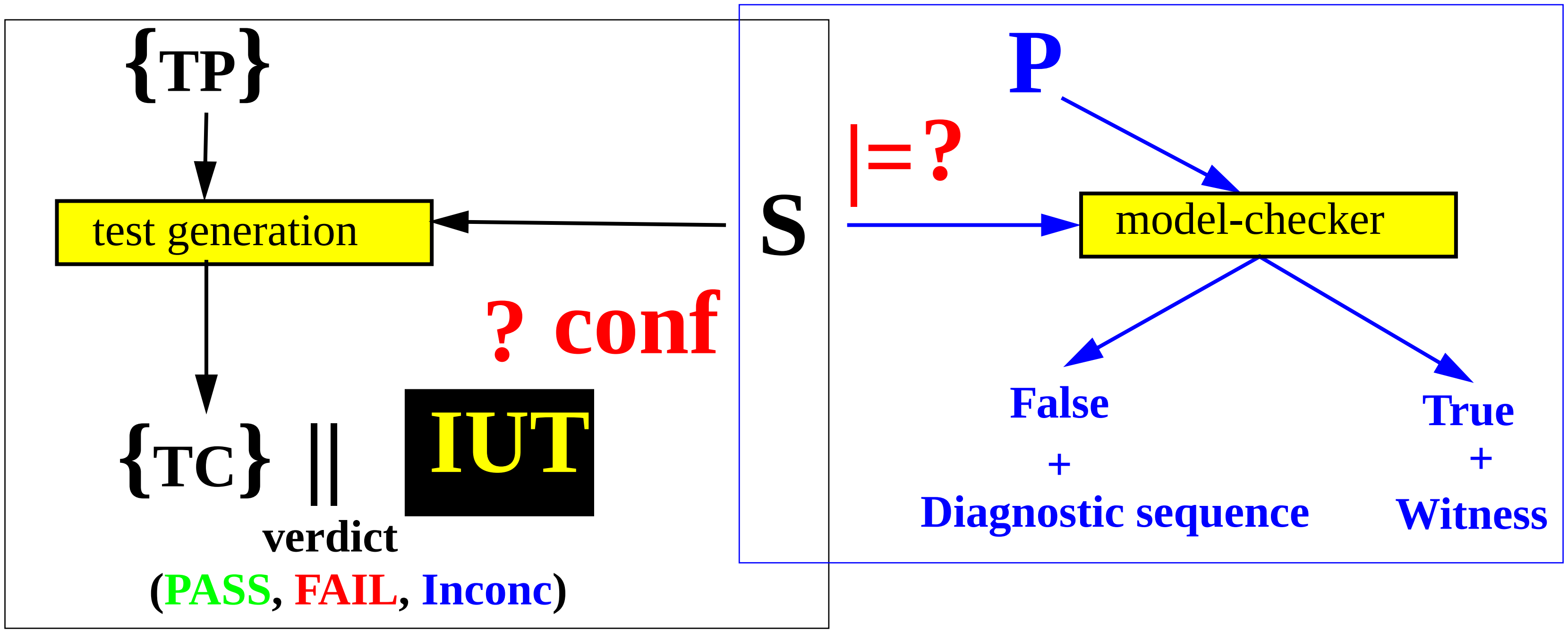
- o Test Purpose ---> Test Case = mirror image of the observable behavior of a the part of the Specification behavior selected by the Test Purpose.
  + a test case should be controlable

  $\Longrightarrow$ Lazzy construction of the behavior of the Specification S,
  its observable behavior (without internal action and deterministic)
  and Test Case selection according to Test Purpose Accept states.

- o **Necessitates special algorithms and a particular tool architecture**

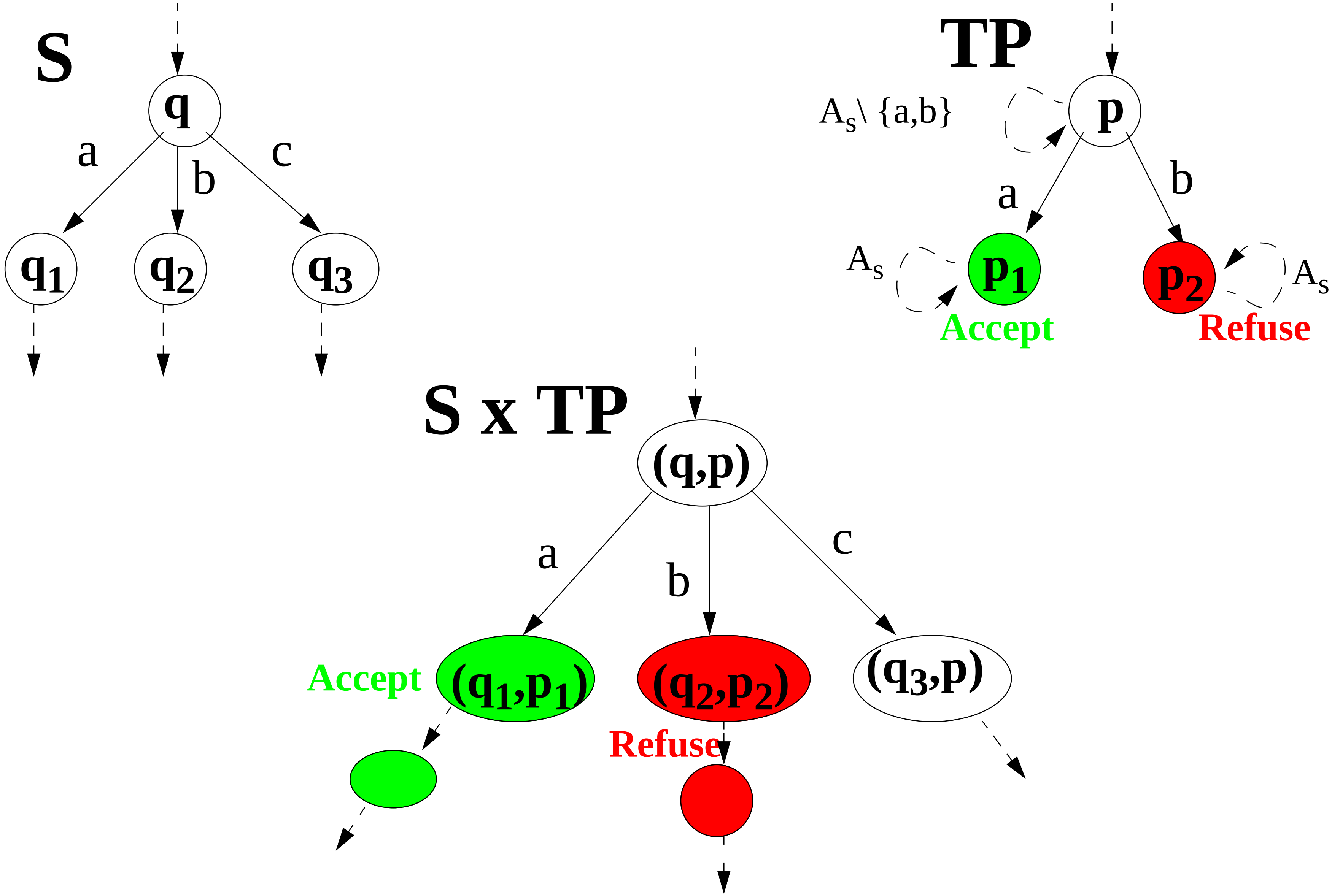


Conformance Testing / Model Checking

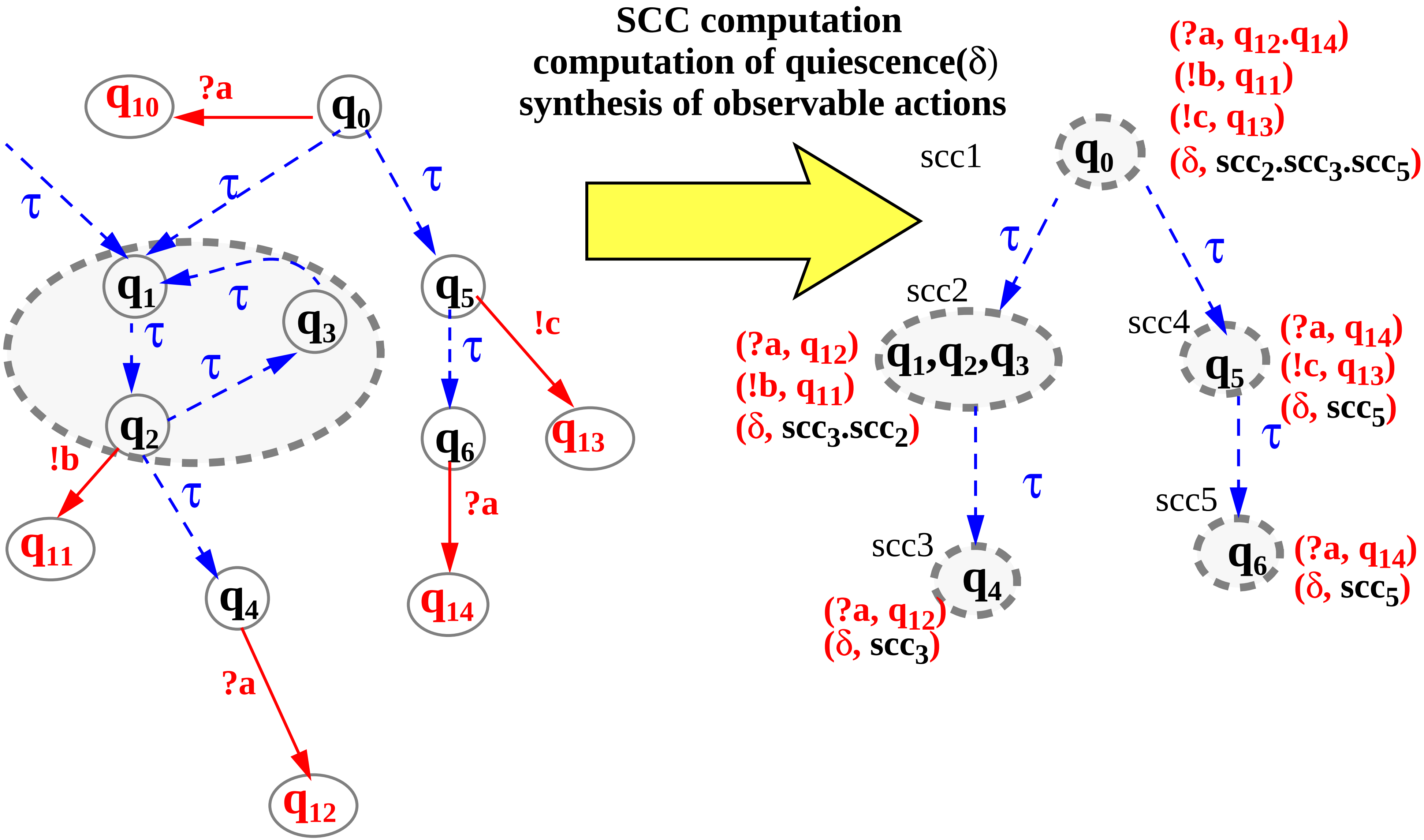


- consider each **Test Purpose TP** as a property
- model-check **S** with **TP**
- generate all witnesses + output freedom of S - internal actions
  -- > **Complete Test Graph**
- add controlability: **Test Case TC**

Synchronous product

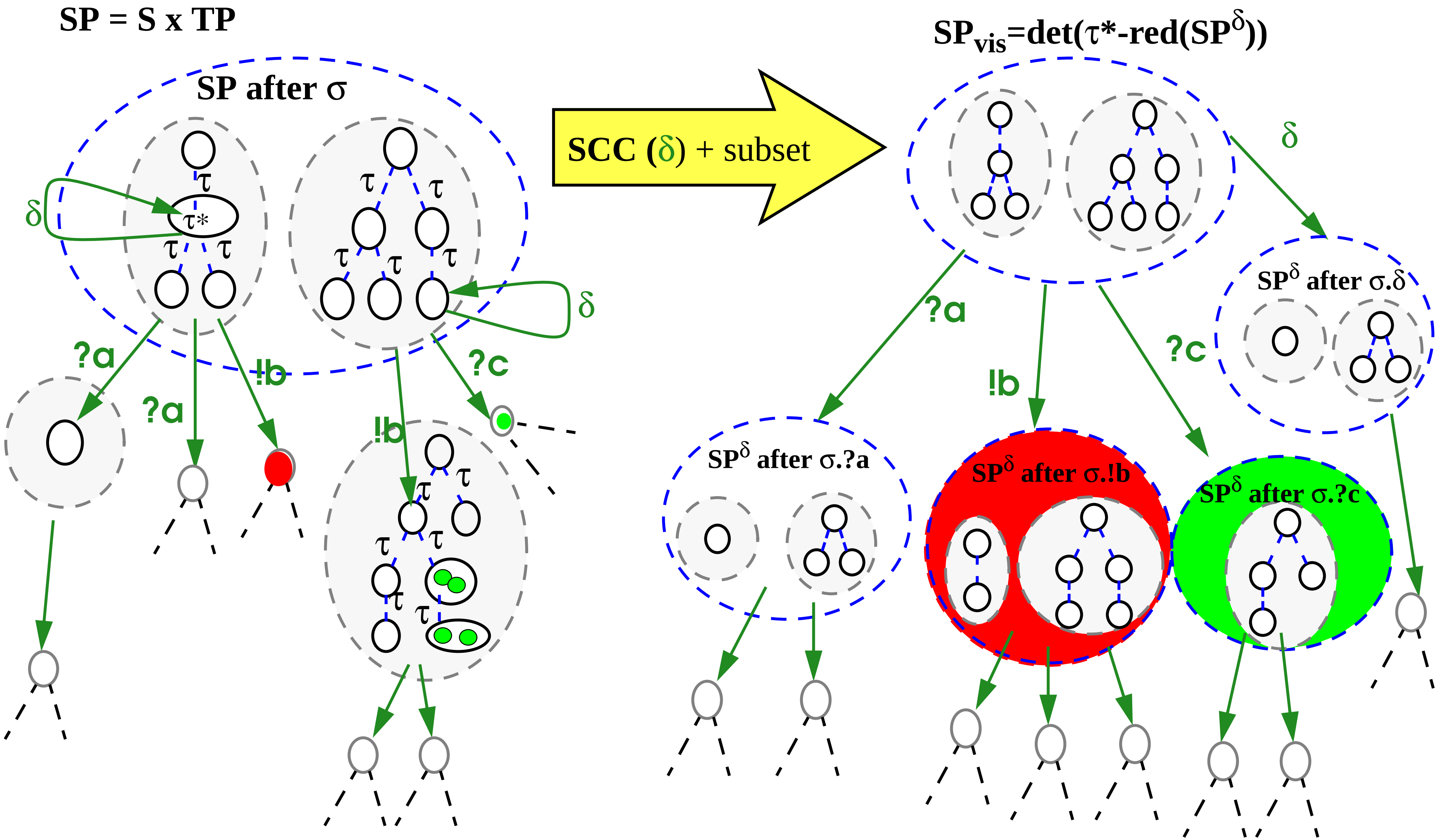


τ*-reduction (local view)



IRISA

τ*-reduction and determinization

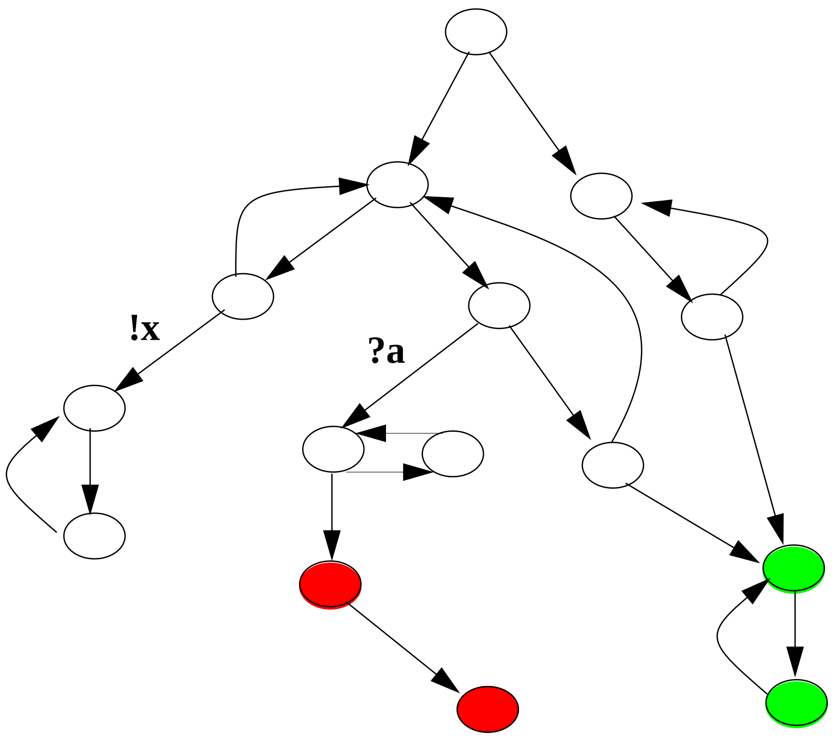


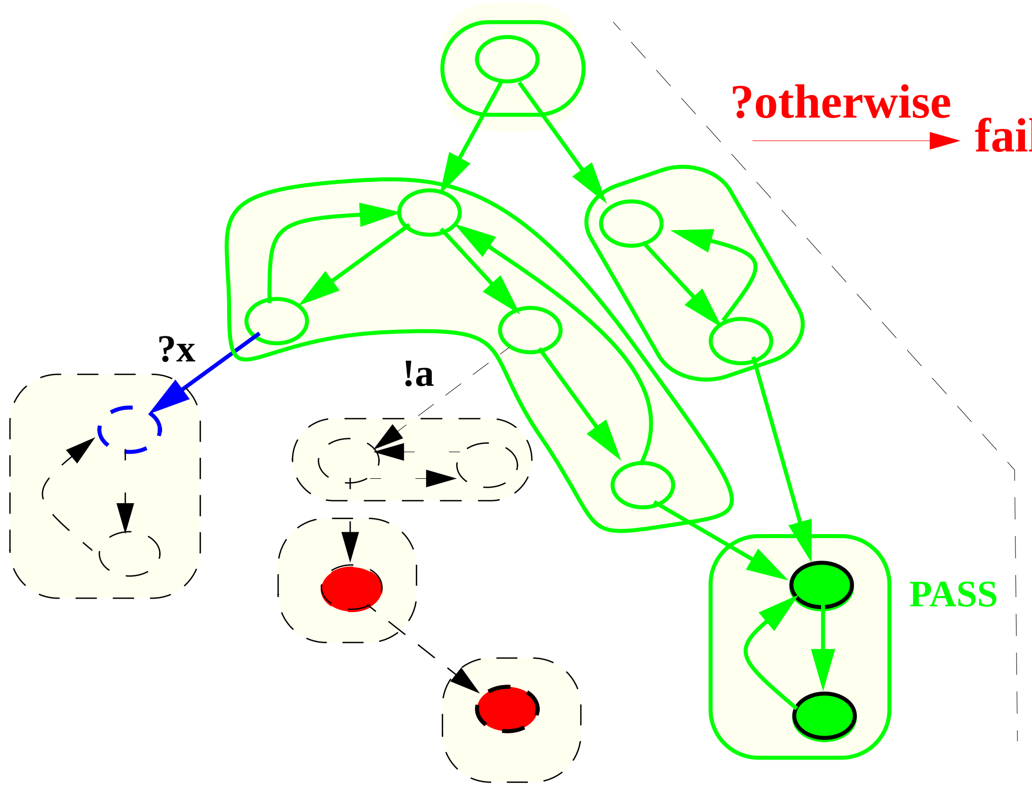
Selection of accepted sub-graph of SP_{vis}

SP_{vis}=det(τ*-red(SxTP))
+ Accept + Reject

SCC + synthesis
mirror image

CTG = (L2A ∪ Inconc ∪ {fail} ,
$\overleftarrow{A}_{vis}$, s_0, $\rightarrow \cup \rightarrow \cup \rightarrow$)

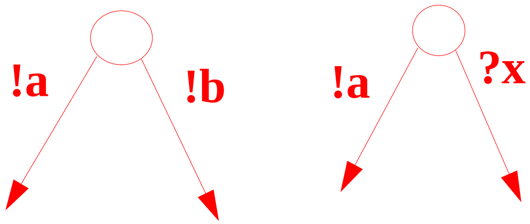




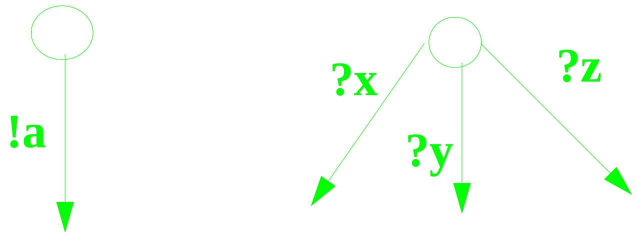
Elimination of controlability conflicts

- In general CTG is not a test case: not controlable

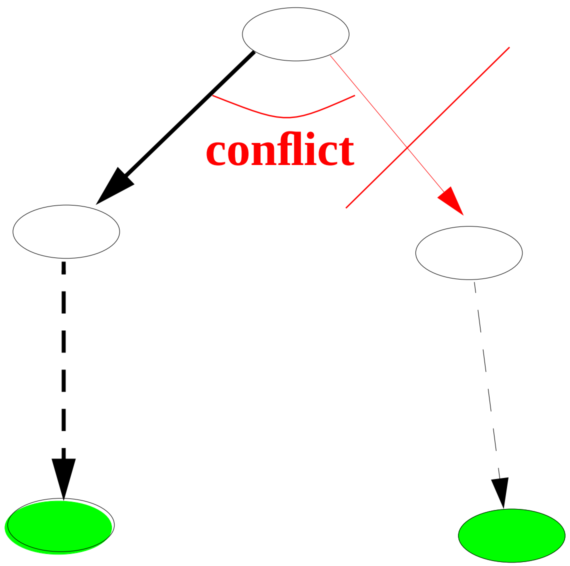
Forbidden configurations: the tester controls its outputs



Authorised configurations



- Conflicts resolution during DFS + completed by a reverse DFS of CTG



Pruning modifies reachability to initial state=> reachability problem

Several possible traversals:

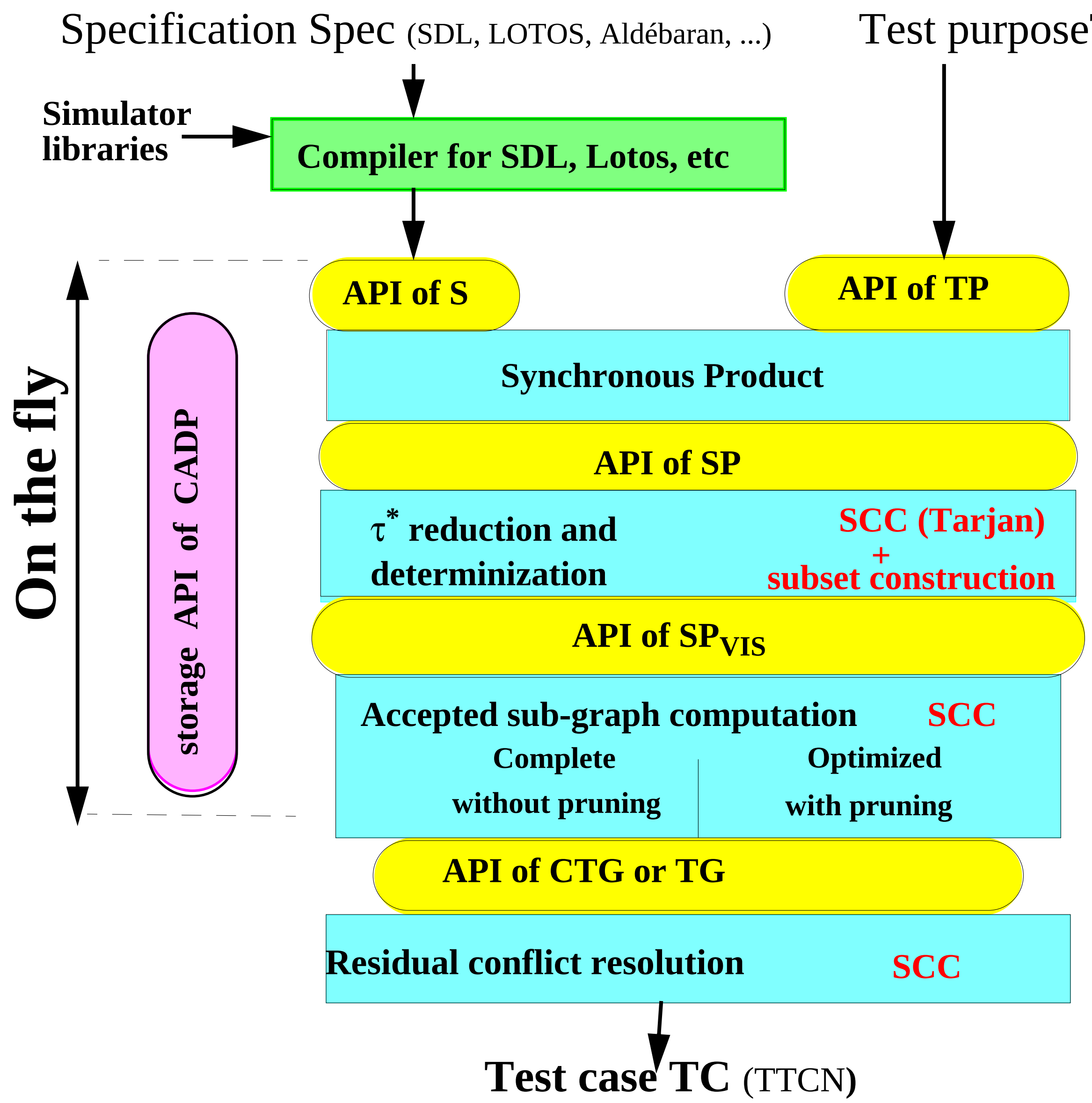
- breadth first starting from Pass states
-> shortest paths

- depth first

- SCC (Tarjan) :

synthesis of information L2init in $\rightarrow_{CTG}^{-1}$
-> garbage collection

Functional architecture of TGV

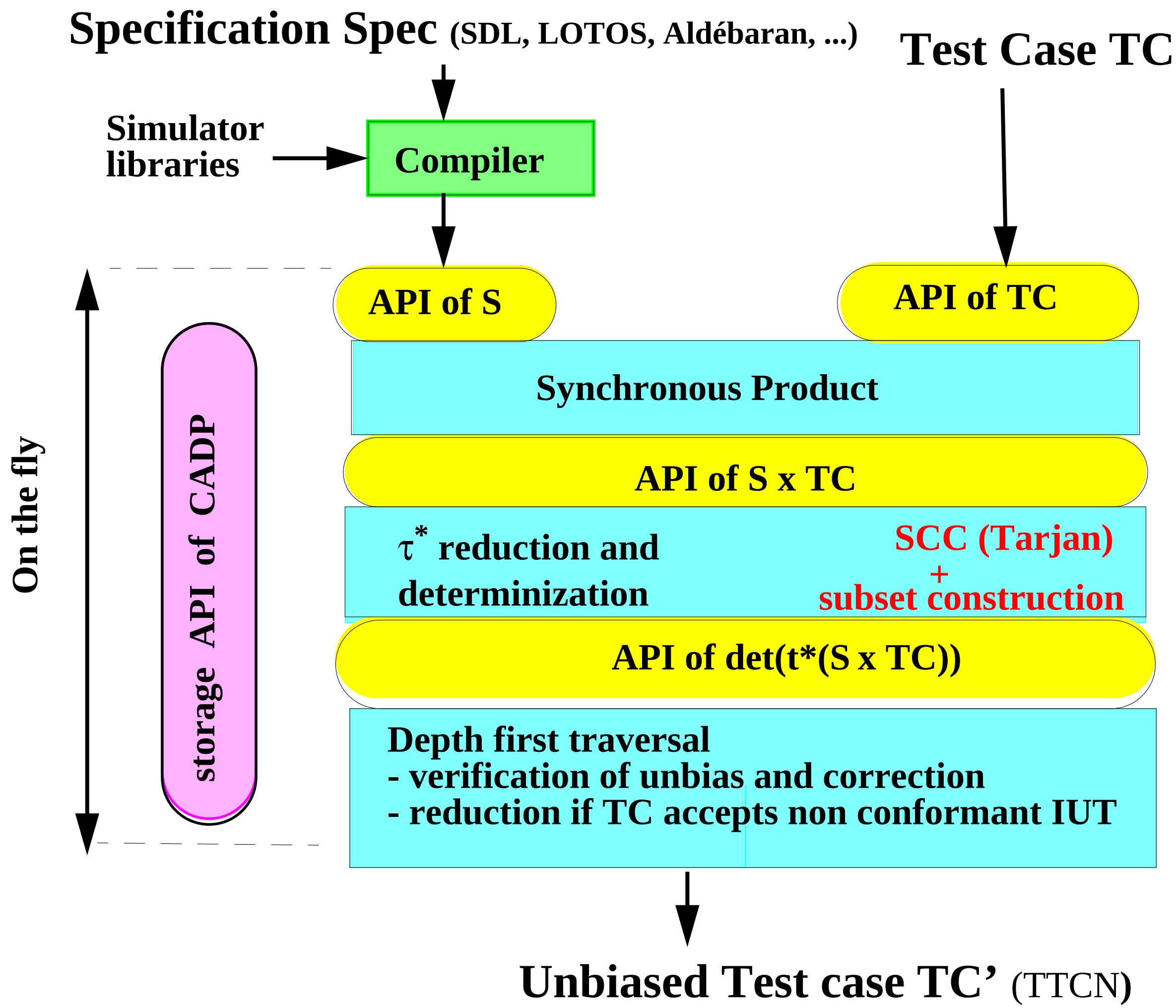


Each API provides the functions:
- init: initial state
- fireable(s): {fireable transitions in t}
- succ(s,t): state reached after t in s

The APIs of SP, SP_{vis} provides also
- Accept(s)
- Reject(s)

The API of CTG or TG provides also
- Pass(s)
- Inconc(s)
- Fail(s)

(Manual) Test case verification



Variant of TGV used for:

- Verification of unbias of (manual) test cases and correction in case of bias

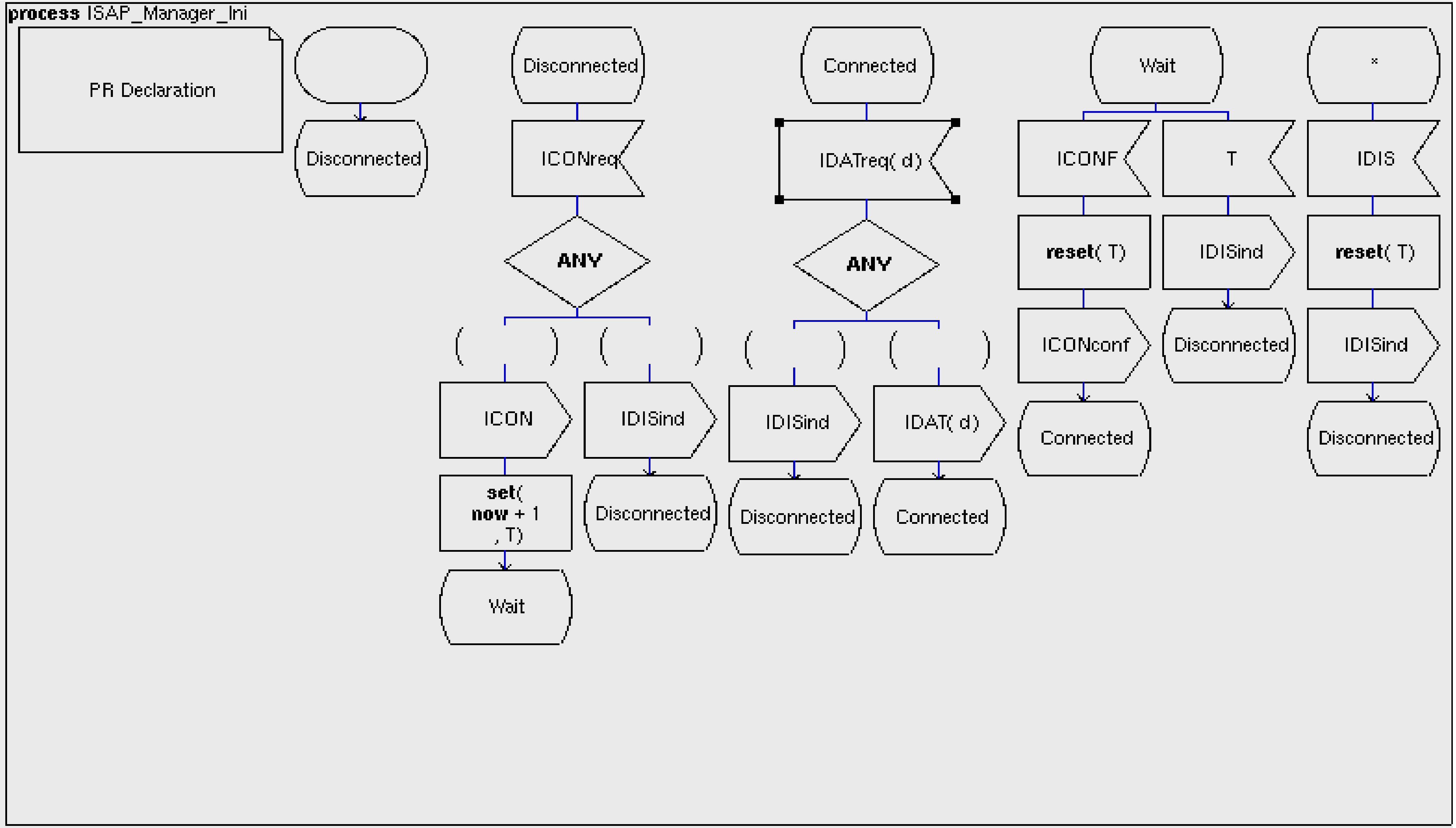
TC biased if TC rejects a conformant IUT

- Refinement if permissiveness

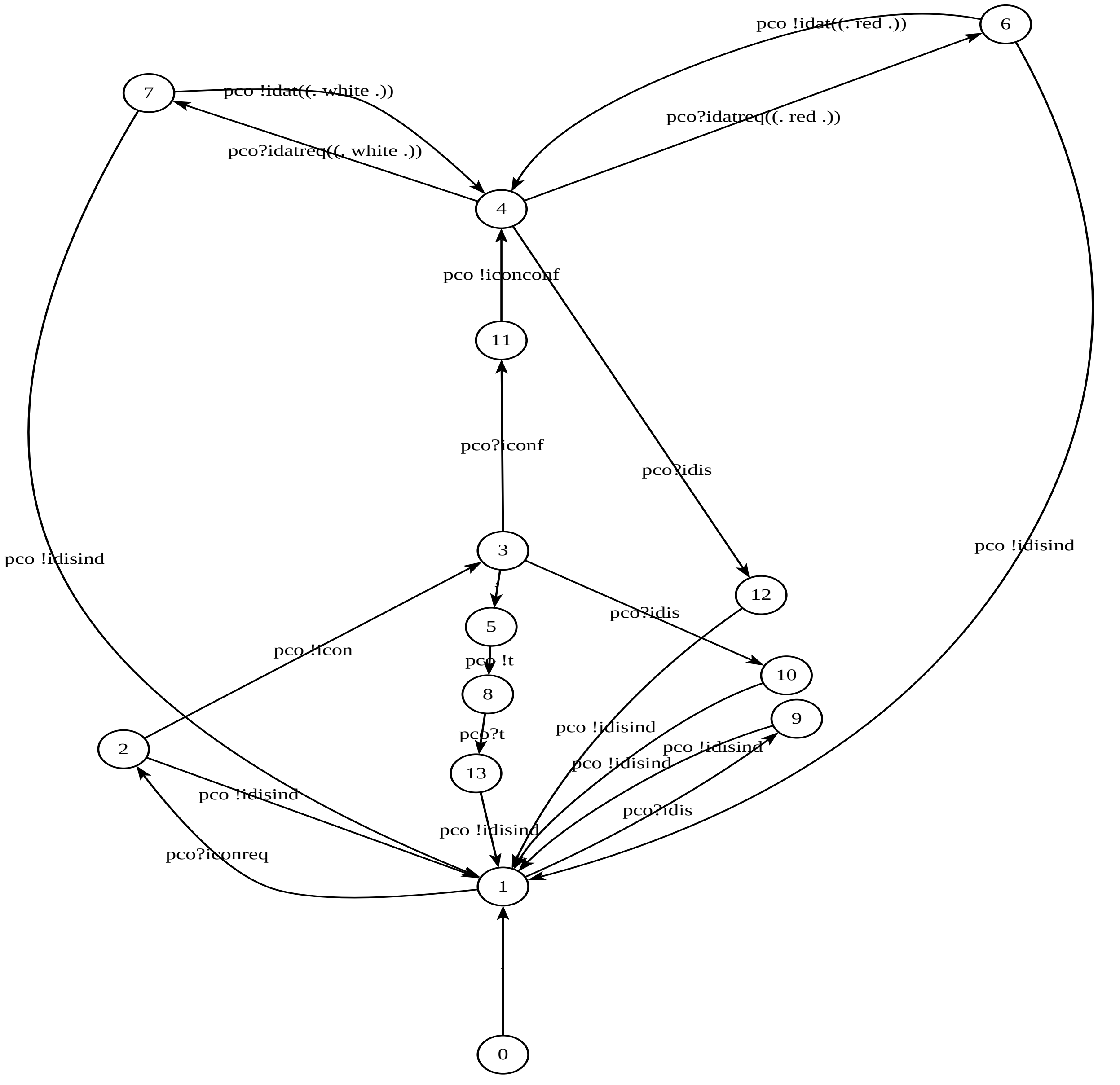
TC permissive if some of its transitions could produce a fail

o Semi-automatic generation: TC outputs proposed by user, inputs and verdicts computed according to Spec using τ^***-reduction and determinization**

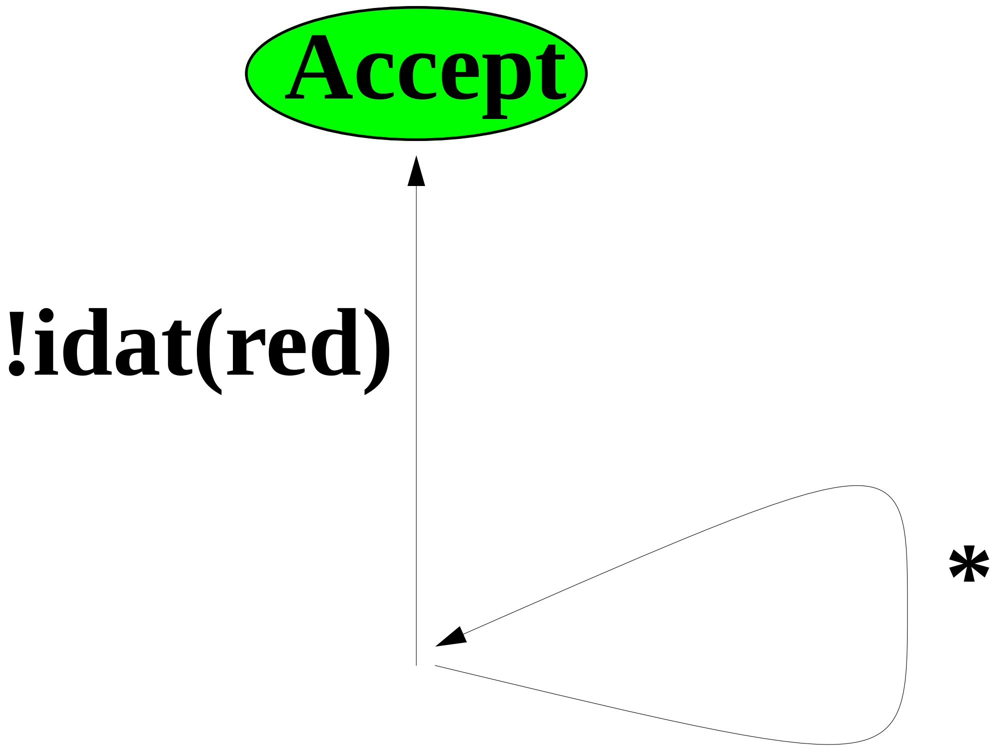
Small example: initiator process of Inres in SDL



Complete state graph (not minimized)

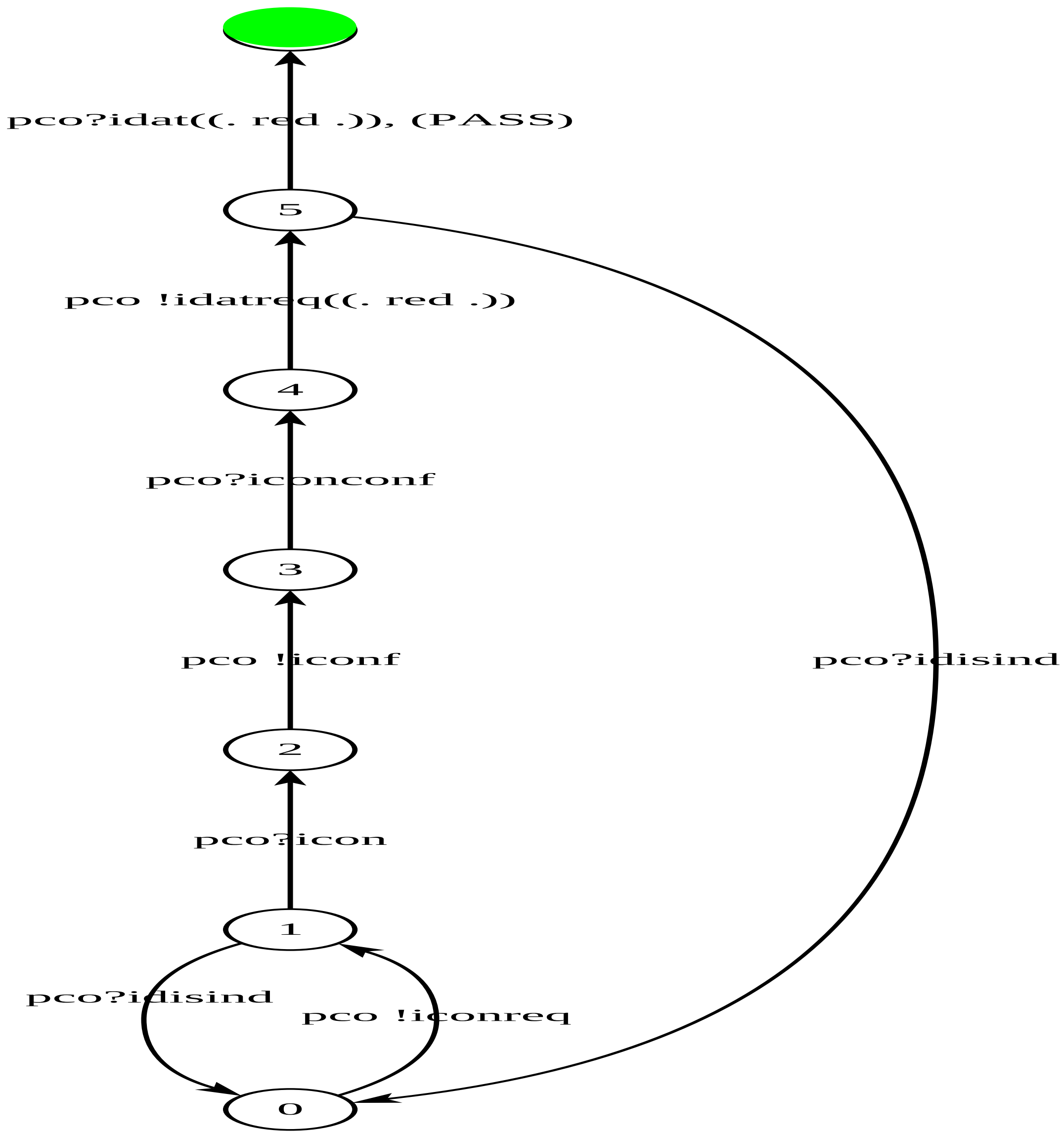


A simple test purpose



Complete Test Graph generated by TGV

Controlable test case produced on the fly



TTCN test case

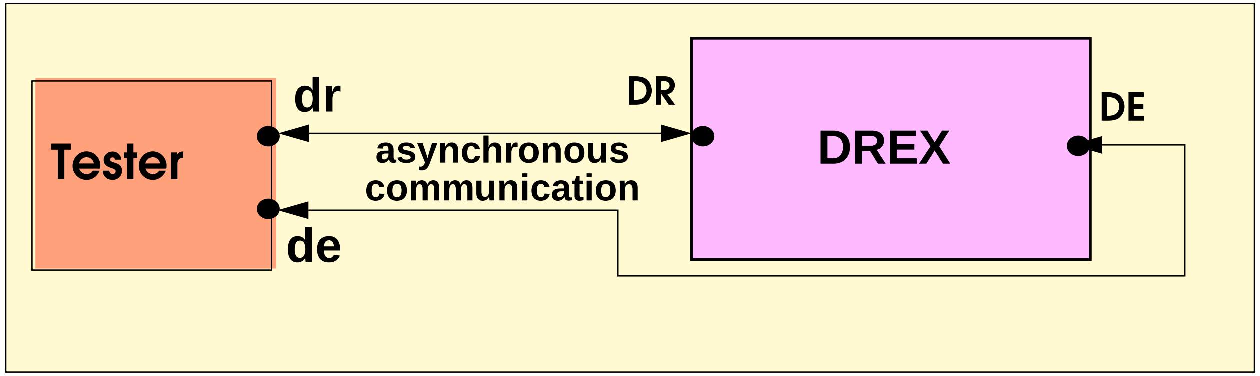
```
+-----------------------------------------------------------------------------------------------+
|                                  Test Case Dynamic Behaviour                                  |
+-----------------------------------------------------------------------------------------------+
| Test Case Name      : graphes/t01_fly_obs_2                                                   |
| Group               :                                                                         |
| Purpose             :                                                                         |
| Default             :                                                                         |
| Comments            :                                                                         |
+------+------+-------------------------------------------+-----------------+----------+----------+
| Nr   | Label| Behaviour Description                     | Constraints Ref | Verdict  | Comments |
+------+------+-------------------------------------------+-----------------+----------+----------+
|    1 | L1   | pco! iconreq, St tidisind, St ticon       | iconreq0        |          |          |
|    2 |      |   pco? icon, Cl ticon, Cl tidisind        | icon2           |          |          |
|    3 |      |     pco! iconf, St ticonconf              | iconf3          |          |          |
|    4 |      |       pco? iconconf, Cl ticonconf         | iconconf4       |          |          |
|    5 |      |         pco! idatreq, St tidisind, St tidat | idatreq5      |          |          |
|    6 |      |           pco? idat, Cl tidat, Cl tidisind | idat6          | (PASS)   |          |
|    7 |      |           pco? idisind, Cl tidat, Cl tidisind | idisind1    |          |          |
|    8 |      |             GOTO L1                       | idisind1        |          |          |
|    9 |      |           ? tidat                         |                 | FAIL     |          |
|   10 |      |           ? tidisind                      |                 | FAIL     |          |
|   11 |      |       ? ticonconf                         |                 | FAIL     |          |
|   12 |      |   pco? idisind, Cl ticon, Cl tidisind     | idisind1        |          |          |
|   13 |      |     GOTO L1                               | idisind1        |          |          |
|   14 |      |   ? ticon                                 |                 | FAIL     |          |
|   15 |      |   ? tidisind                              |                 | FAIL     |          |
+------+------+-------------------------------------------+-----------------+----------+----------+
```

3. Experiments and Industrial Transfert

o **DREX protocol in SDL** (military version of D protocol)

- contract with French Army, CNET, Verilog, Cap Sesa, Verimag, Irisa.

5 service specification (1 per service):
1 process, 35 SDL transitions, 1800 lines of SDL PR, 50 pages of SDL GR.



test case generation: first version of TGV on explicit graphs,
50 test purposes

comparison with manual test cases (some errors detected)
with test cases generated by TVeda(CNET) and TTCgeN(Verilog)

result: clearly demonstrated the interest of the approach
in terms of efficiency and test quality.

o **Cache coherency protocol of the Polykid architecture in Lotos**
Cooperation with Bull, INRIA Rhones-Alpes, Irisa

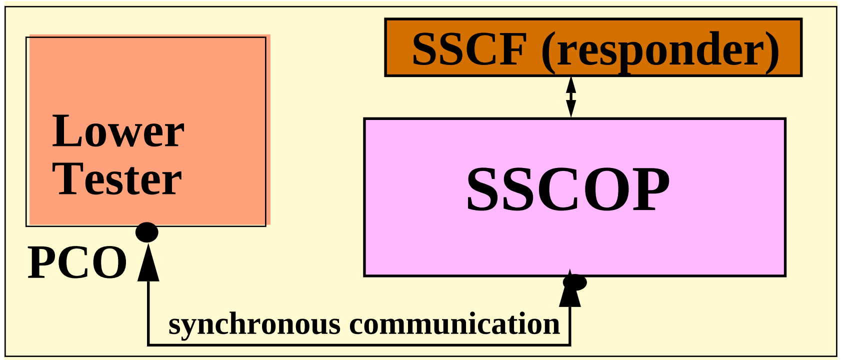
Lotos specification: 2000 lines (1800 ADT, 200 control), 1 process.

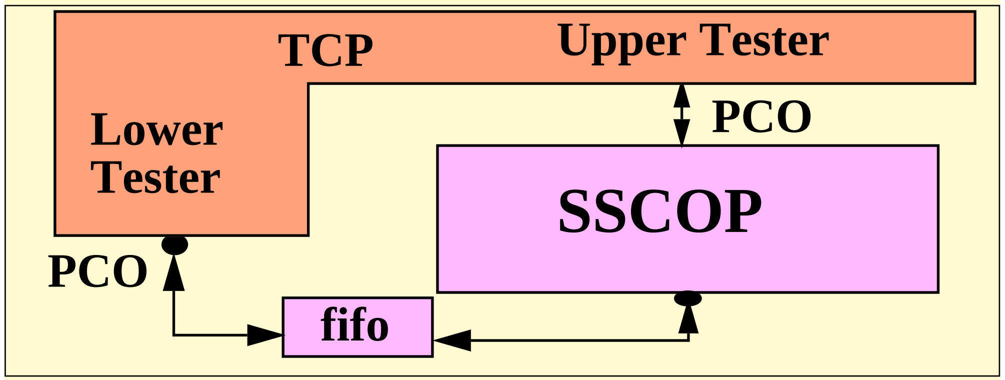
- test case generation:
  "on the fly" with a connection to the Open Caesar Lotos simulator.
- test case execution:
  on the Polykid architecture by a translation of test cases in C.
- results:
  - design and coding of a second version of TGV "on the fly"
  - use of TGV in a different application domain
  - complete chain from specification to test execution
  - work still continues on other architectures

o **SSCOP protocol of the ATM stack in SDL**

Specification: 1 process (2 in the asynchronous case),
7000 lines in SDL PR, 100 pages of SDL GR, 175 SDL transitions

test architecture: local with 1PCO and remote with 2PCOs**.**





SSCOP (continued)

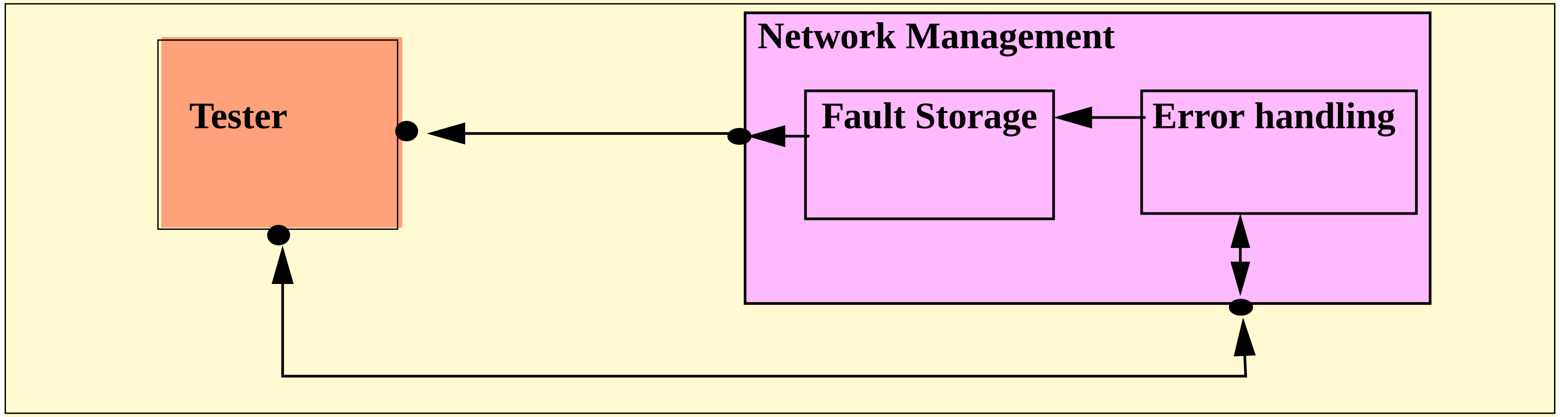
o Test generation:

- connection of the on-the-fly version of TGV to the ObjectGéode simulator of Verilog.
- test generation from 50 complex test purposes.

o Test case verification and correction (unbias and laxness)

- from TTCN test suite translated into our input format (Lex,Yacc)
- verification of test cases w.r.t SDL spec.
  using TGV_VTS connected to ObjectGeode .
- 110/250 test cases verified (valid PDU, no Invalid or Inoportune PDU)
  --> 16 erroneous test cases corrected

- **Protocol of an embedded network in the automotive area in SDL.**
  - draft of the protocol: 2 processes, 1000 lines of SDL PR, 25 pages of SDL GR

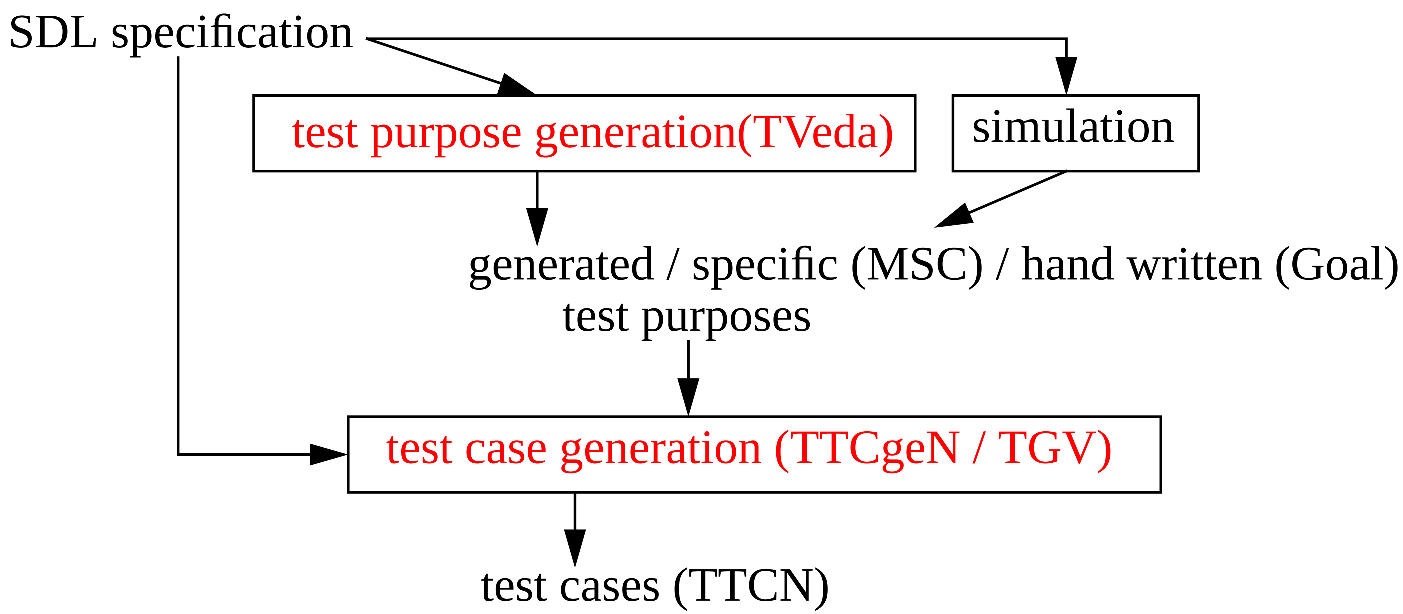


  - feasability of on-the-fly generation shown on a few test purposes
  - connection of TGV “explicit” to SDT of Telelogic by a translation of the SDT state graph format into Aldebaran format.

Industrial Transfert

- **Design of an industrial test generation tool TestComposer (Verilog) in the ObjectGéode environment based on TVéda (CNET), TTCgeN (Verilog) and TGV(Verimag/Irisa)**

GAT project 98-99: France Telecom, Verilog, Verimag, Pampa



The design, coding and validation of the algorithms of TGV was partially done at Irisa.

4. Ongoing Work in Testing

o **Symbolic test generation for a better treatment of data in specifications**

- combination of TGV techniques with constraint solving, static analysis and proof (PVS).
- application domains: protocols, smart cards, ...

o **Distributed testing and asynchronous communication**

- synthesis of distributed test cases from sequential ones,
- direct synthesis of distributed test cases (true concurrency models)
- results on respective powers of local synchronous testing and remote asynchronous testing using stamps

o **Test synthesis for distributed object oriented software**

- connection of TGV with our UML validation framework.

o **Use of game theory for test generation**

- testing = game between the system and the tester
- winning strategies = test cases with best possible control of the IUT.
- to be implemented in TGV

o **Controler Synthesis and Test Synthesis**