TGV: theory, principles and algorithms

David Reisenberger

May 31, 2012

The paper

- Claude Jard, Thierry Jeron
- Published online 2004, Springer-Verlag

TGV

- Test generation with verification technology
- automatic synthesis of conformance test cases from a formal specification of a (non-deterministic) reactive system
- ”on-the-fly” synthesis

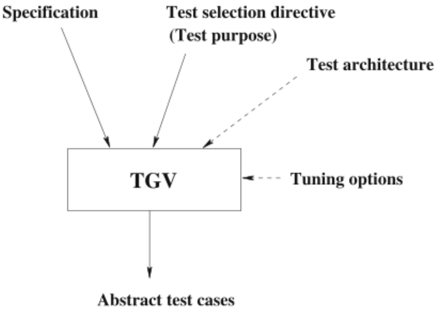
Outline

1. Basics
   - TGV overview
   - IOLTS
   - Formal test purposes
2. Principles and algorithms
   - Synchronous product
   - Extracting visible behaviour
   - Test selection
   - Controllability
   - On-The-Fly synthesis
3. The Tool
4. Conclusion

Definitions

- **Test case**: testing particular functionality
- **Test suite**: a set of tests
- **Test**: outputs are stimuli for IUT, inputs are observations of IUT's outputs
- **Fail verdict**: IUT is rejected
- **Pass verdict**: IUT is accepted
- **Inconclusive verdict**: correct behaviour is observed but test purpose can't be reached
- **Soundness**: test cases only reject non-conformant IUT's
- **Exhaustiveness**: all non-conformant IUT's are rejected

Functional view



Specification

Test selection directive
(Test purpose)

Test architecture

TGV

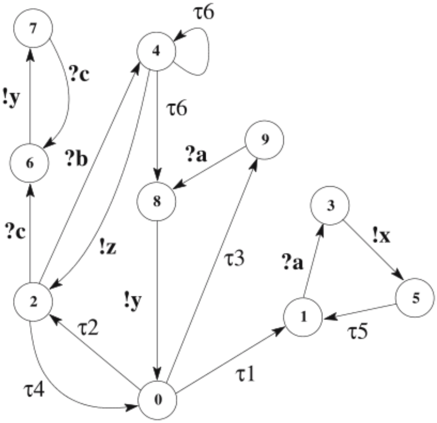
Tuning options

Abstract test cases

IOLTS

- $M = (Q^M, A^M, \rightarrow_M, q_0^M)$
- $A^M = A_I^M \cup A_O^M \cup I^M$
- $a_{(i)} \in A^M \setminus I^M$
- $\tau_{(i)} \in I^M$
- **Fireable actions**: $\Gamma(q)$
- **Transitions**: $\xrightarrow{a}$
- **Visible behaviour**: $\Rightarrow$
- **Input**: $?a$
- **Output**: $!x$
- **det(M)**: M without internal actions

IOLTS cont.

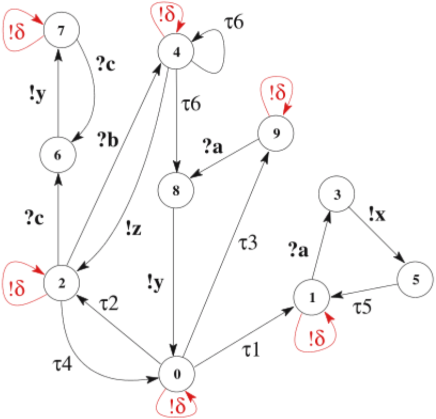


Quiescence

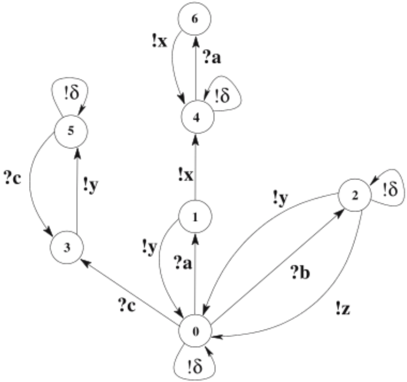
- Deadlock: $\Gamma(q) \; = \emptyset$
- Output quiescence: $\Gamma(q) \; \subseteq A_I^M$
- Livelock
- $deadlock(M) \subseteq outputlock(M)$
- $quiescent(M) = livelock(M) \cup outputlock(M)$

Suspension automaton $\Delta(S)$

▸ TP ▸ SP



$det(\Delta(S)) = S^{VIS}$

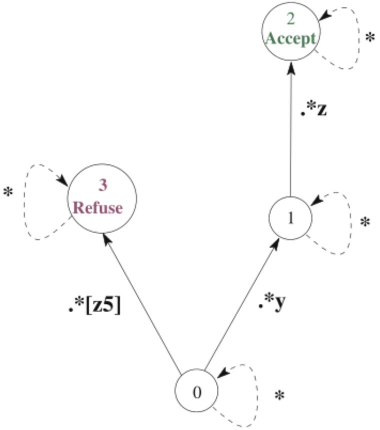


Test purpose

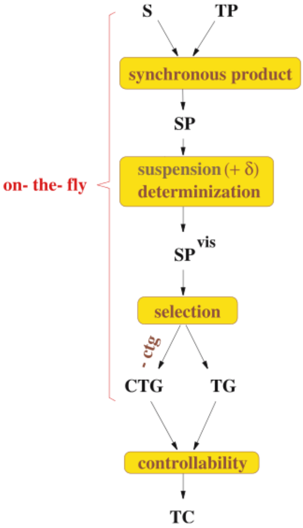
- $TP = (Q^{TP}, A^{TP}, \rightarrow_{TP}, q_0^{TP})$
- $Accept^{TP}$: select target behaviour
- $Refuse^{TP}$: cut down specification
- Allows efficient test selection on-the-fly
- Smaller than specification but complete (...?)

Test purpose cont.

▸ S ▸ SP



Principles and algorithms

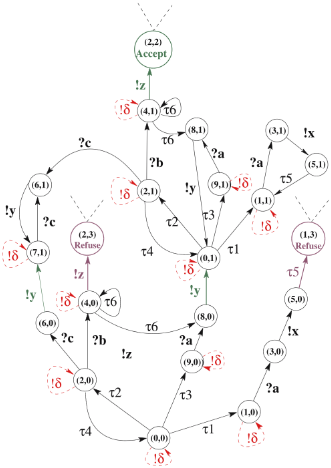


Synchronous product

- Mark behaviours of S by *Accept*, *Refuse*
- Accepted behaviour of SP are accepted behaviours of S by TP

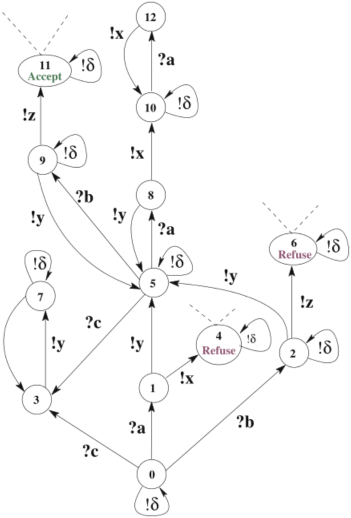
$S \times TP = SP$

▸ S ▸ TP



$SP^{VIS} = det(\Delta(SP))$

▸ CTG

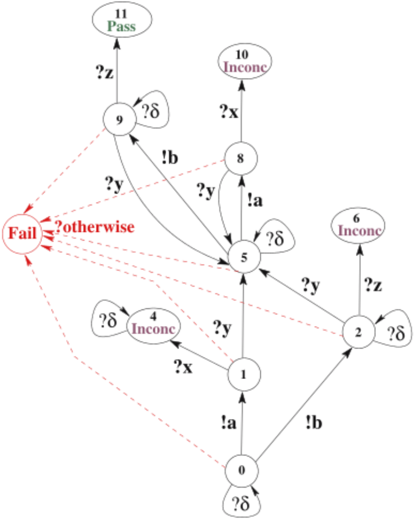


Test selection

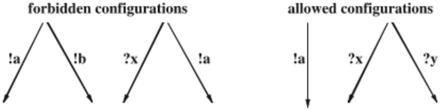
- Extracting test case by selection of accepted behaviours
- $A_O^{CTG} \subseteq A_I^{VIS}$
- $A_I^{CTG} = A_O^{VIS}$
- **L2A**: all states that lead to accept
- **Pass**: $Accept^{VIS}$
- **Inconc**: direct successors of states in L2A by output in SP^{VIS}
- **Fail**: else
- **Result**: CTG (Complete Test Graph)
- **TGVLoop**, based on Tarjan's algo $(O(n), S(n))$

Test selection

▸ SP^{VIS} ▸ TC



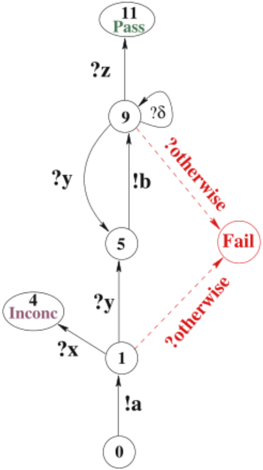
Controllability



- Extract a controllable subgraph of CTG
- Get rid of choices between I/O (pruning)
- This can happen during TGVLoop (partially)

A test case

▸ CTG



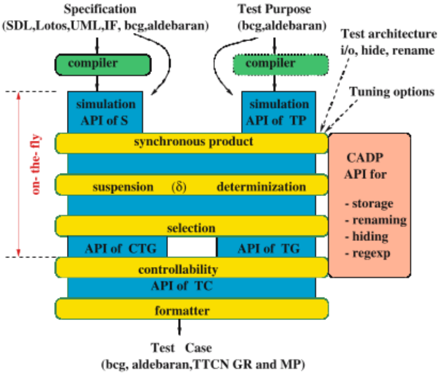
On-The-Fly synthesis

- perform lazy construction of subgraphs of S, SP, SP^{VIS}
- needed functions
  - **init**
  - **fireable**
  - **succ**
  - Comparison function
  - Function to compute membership of **Accept/Refuse**
- goal: reduce size of graphs

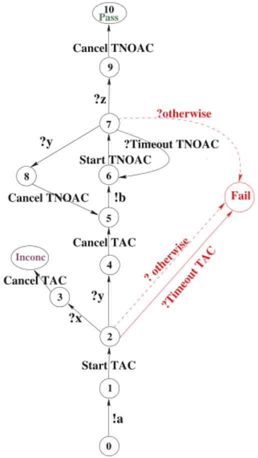
The Tool

- Several software layers
- Communicate through APIs
- Each one is simulation of IOLTS (allows graph traversal)
- Each level implements one of the algorithms
- Output: test cases in TTCN or graph formats (.aut, .bcg)
- Can be used to verify manually created test cases
- SunOS 5, Windows XP, Linux

Architecture



Timers


10
Pass
Cancel TNOAC
9
?z
?otherwise
7
?y
?Timeout TNOAC
Start TNOAC
8
6
Fail
Cancel TNOAC
!b
5
Cancel TAC
Inconc
4
? otherwise
Cancel TAC
3
?y
?Timeout TAC
?x
2
Start TAC
1
!a
0

Conclusion

- TGV can synthesize tests from industrial size specs
- Drawback: manual creation of test purpose
- Still better than manual test case creation
- Future: distributed tests, improvements of algorithms