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

- **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

- 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

⇒ Automation of test generation from formal specifications can be profit earning
Conformance testing of protocols

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

- **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

Spec

Test Purpose:
number and later connect
Expected test case

PASS

| ?return reset TAC |
| !card_out start TAC |
| !hang_up (PASS) |
| ?connected reset TAC |
| ?bell reset TAC start TAC |
| !number start TAC |
| ?accept reset TAC |
| !card_in start TAC |

FAIL

| ?timeout TAC |
| ?otherwise |

Preamble

Test Body

Postamble

Work needed

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

Work needed
Automatic Test Synthesis

Informal Specification → Formal specification (SDL, Lotos, Estelle) → Exhaustive Simulator → (Partial) State Graph → Reduction of internal actions Determinization (Minimization) → Observable behavior → Test generator → IUT → Test execution → Test Suite = { Test cases} → Verdict
Automata Theoretic Methods

- **Origin**: Hardware testing

- **Models**: Mealy machines

  - **Fault Model**: Output: $s \overset{i/o}{\rightarrow} s'$, Transfer: $s \overset{i/o}{\rightarrow} s''$

- **Hypothesis**:
  - Spec: Input complete, deterministic, minimal, strongly connected
  - IUT: Input complete, deterministic, minimal (or $< k$), strongly connected

- **Test Generation**: One test case per transition:
  - Test suite: Minimal length sequence with all elementary test cases
  - Algorithms: Traveling salesman, flow graphs, linear programming

- **Different Methods**: TT, DS, UIO, W, etc., differ on checking sequences

- **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

- **origin**: testing theory, canonical tester.
- **models**: LTS (not well adapted) or IOLTS: \[ s \xrightarrow{?i} s' \xrightarrow{!o} s'' \]
- **fault model**: conformance relation between IUT and Spec
  - difference between possible observations of IUT and Spec after same traces
- **hypothesis**: Spec: no hypothesis, IUT: input complete
- **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)
- **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)

- Joint project since 94: Verimag Grenoble - Irisa Rennes
- **Goal**: using on the fly model-checking techniques for efficient test case synthesis for conformance testing.
- **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, unbiased and (theoretical) exhaustiveness.

- **Language 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

- **Test generation:**
  - Specification
  - Test Purpose
  - TGV
  - Abstract Test Case

- **Generation of the complete test graph:**
  - Specification
  - Test Purpose
  - TGV-csg
  - Test graph (graph of all test cases for TP)
  - Abstract Test Cases

- **Verification of (manual) test cases**
  - Specification
  - Abstract Test Case
  - TGV_VTS
  - Corrected/Refined Abstract Test Case
Test Generation in TGV

**Specification S**
behavior modelled by an IOLTS

**Black box**
Implementation I

**Test purpose TP**
automata for test selection

**Test case TC**

**Test execution on the IUT**

**Verdict**

**Properties:**
- unbias: \( \text{verdict}(\text{exec}(\text{TC} \parallel I)) = \text{fail} \Rightarrow \text{not}(I \ioco S) \)
- exhaustivity: if I is fair, \( \text{not}(I \ioco S) \Rightarrow \exists \text{TP} \text{ s.t. } \text{verdict}(\text{exec}(\text{TC}(S,\text{TP}) \parallel I)) = \text{fail} \)

coherency (preorder)
Models: IOLTS

- Transition systems with three kinds of transitions:
  - input: \( s \xrightarrow{?x} s' \), output: \( s \xrightarrow{!a} s' \),
  - internal action: \( s \xrightarrow{\tau} s' \)

- Modelisation facilities:
  - non-determinism (automata):
    \[ \xrightarrow{?x} s \xrightarrow{!a} s' \]
  - observable “non-determinism”:
    \[ \xrightarrow{?x} s \xrightarrow{!a} s' \xrightarrow{!b} s'' \]
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.

**Deadlock**

**Livelock**

**Output quiescence**
Conformance relation

I ioco S
iff
∀ σ ∈ traces(Sδ), Out(Iδ after σ) ⊆ Out(Sδ after σ)
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 \iff Accept
  
 Fail \iff/ Refuse (traversal cut)

Test purpose accepting sequences with $\tau_1$ followed later by $!b$.

Refuse allows to cut sequences with a $\tau_2$ before any $\tau_1$. 
Principle of on-the-fly generation

- 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

⇒ 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.

- Necessitates special algorithms and a particular tool architecture

---

**Diagram:**

- $S \times_{\tau}$-reduction determinization
- Generation of complete test graph
- Generation + controlability
- Resolution of controllability conflicts
- On-the-fly generation

---

**Notations:**

- $S$: Specification
- $x$: Test Purpose
- $\tau$: Reduction operator
- Determinization
- SP$_{vis}$: Visible Specification
- CTG: Complete Test Graph
- TG: Test Graph
- TC: Test Case

---
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 controllability: Test Case TC
Synchronous product

S

q

q_1 q_2 q_3

a b c

q_1' q_2' q_3'

S \times TP

(q,p)

a b c

(q_1,p_1) (q_2,p_2) (q_3,p)

TP

p

A_s \{a,b\}

p_1 p_2

Accept Refuse

A_s

Accept

Refuse

A_s
$\tau^*$-reduction (local view)

SCC computation
computation of quiescence($\delta$)
synthesis of observable actions
\( \tau^* \)-reduction and determinization

\[
SP = S \times TP
\]

\[
SP_{\text{vis}} = \text{det}(\tau^*\text{-red}(SP^\delta))
\]

SCC (\( \delta \)) + subset
Selection of accepted sub-graph of $SP_{\text{vis}}$

$SP_{\text{vis}} = \text{det}(\tau^* - \text{red}(SxTP))$

$CTG = \langle L2A \cup \text{Inconc} \cup \{\text{fail}\}, \Lambda_{\text{vis}}, s_0, \rightarrow \cup \leftarrow \cup \rightarrow \rangle$

SCC + synthesis mirror image

PASS

?otherwise fail
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
  - Several possible traversals:
    - breadth first starting from Pass states
    - depth first
    - SCC (Tarjan):

Pruning modifies reachability to initial state=> reachability problem

- synthesis of information L2init in →CTG⁻¹
  - garbage collection
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

Specification Spec (SDL, LOTOS, Aldébaran, ...)

Test Case TC (Aldébaran)

Compiler

API of S

Synchronous Product

API of S x TC

τ* reduction and determinization

API of det(τ*(S x TC))

SCC (Tarjan)

+ subset construction

Depth first traversal
- verification of unbias and correction
- reduction if TC accepts non-conformant IUT

Unbiased Test case TC’ (TTCN)

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

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

Accept

!idat(red)
Complete Test Graph generated by TGV
Controlable test case produced on the fly

1. pco?idat((. red .)), (PASS)
2. pco?icon
3. pco?iconconf
4. pco !iconreq((. red .))
5. pco?idisind

0. pco !iconreq
1. pco?idisind
2. pco?icon
## TTCN test case

**Test Case Dynamic Behaviour**

<table>
<thead>
<tr>
<th>Test Case Name</th>
<th>graphes/t01_fly_obs_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nr</th>
<th>Label</th>
<th>Behaviour Description</th>
<th>Constraints Ref</th>
<th>Verdict</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L1</td>
<td>pco! iconreq, St tidisind, St ticon</td>
<td>iconreq0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>pco? icon, Cl ticon, Cl tidisind</td>
<td>icon2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>pco! iconconf, St ticonconf</td>
<td>iconconf3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>pco? iconconf, Cl ticonconf</td>
<td>iconconf4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>pco! idatreq, St tidisind, St tidat</td>
<td>idatreq5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>pco? idat, Cl tidat, Cl tidisind</td>
<td>idat6</td>
<td>(PASS)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>pco? idisind, Cl tidat, Cl tidisind</td>
<td>idisind1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>GOTO L1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>? tidat</td>
<td>idisind1</td>
<td>FAIL</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>? tidisind</td>
<td></td>
<td>FAIL</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>? ticonconf</td>
<td></td>
<td>FAIL</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>pco? idisind, Cl ticon, Cl tidisind</td>
<td>idisind1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>GOTO L1</td>
<td>idisind1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>? ticon</td>
<td></td>
<td>FAIL</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>? tidisind</td>
<td></td>
<td>FAIL</td>
<td></td>
</tr>
</tbody>
</table>
3. Experiments and Industrial Transfert

- **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.
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
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)

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

- 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
- feasibility 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

- **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, ....

- **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
- **Test synthesis for distributed object oriented software**
  - connection of TGV with our UML validation framework.

- **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

- **Controller Synthesis and Test Synthesis**

  ![Diagram](image)