Test generation using verification

Thierry Jéron
Irisa/Inria Rennes, France
http://www.irisa.fr/vertecs/

Adapted from
B. Jeannet, T. Jéron, V. Rusu, E. Zinovieva,
Symbolic Test Selection based on Approximate Analysis,
in TACAS’05, LNCS 3440, Edinburgh (Scotland), April 2005.

Vlad Rusu, Hervé Marchand, Thierry Jéron,
Automatic Verification and Conformance Testing for Validating Safety Properties of Reactive Systems,
Conformance Testing of reactive systems

Problem:
check if a black-box implementation of a reactive system conforms to a given specification by experimenting with test cases

- the source code of the implementation is unknown (or not used)
- only the interface is known
- and can be controlled/observed by the environment

Test generation

Test cases
Outline

• The ioSTS model

• Conformance Testing Theory with ioco

• Test selection using approximate analysis

• Conclusion
1. The \textit{ioSTS} model

\[ S = (V_S, \Theta_S, \Sigma, T_S) \]

with

- \( V_S \): variables \( \ni \) loc
- \( \Theta_S \): initial condition
  
  with unique solution
- \( \Sigma = \Sigma_i \cup \Sigma_? \cup \Sigma_\tau \)
  
  alphabet of actions
  
  with comm. parameters \( P \)
- \( T_S \): transition relation

\[ [a(p) : G(v_S,p); v_S := A(v_S,p)] \]

action guard assignment

Hyp: satisfiability of guards is decidable.
ioLTS semantics of ioSTS

\[ S = (V_S, \Theta_S, \Sigma, T_S) \]

\[
\begin{array}{c}
\text{Init} \quad \text{Rx} \quad \text{Ry} \quad \text{Cmp} \\
\text{x=y=0} \quad \text{p=y-x \land p \geq 2} \quad \text{p=y-x \land p < 2} \\
\end{array}
\]

\[
\text{Runs}(S): q_0 \to^{a_1(\pi_1)} q_1 \to^{a_2(\pi_2)} q_2 \ldots \in Q_0 \cdot (\Lambda.Q)^* \\
\text{Tr}(S): \text{proj}_{\Lambda_{vis}}(\text{runs}(S))
\]
Simplifying assumptions on ioSTS (for this talk)

1. **ioSTS** are supposed to be deterministic

   Determinisation of ioSTS into ioSTS is **not always possible**
   - restrictions needed e.g.
     - No internal loop
     - Finite lookahead

2. Quiescence is not treated here

   $\Delta(s)$: explicit quiescence by adding loops with $!\delta$ in all quiescent states (no fireable output/internal)

   Augment guard model with universal quantification
   - not a real problem

   $\text{STr}(S) = \text{Tr}(\Delta(S))$
2. Conformance Testing Theory with ioco [Tretmans 96]

- **Specification:** known ioLTS $S$ (semantics of an ioSTS)
- **Implementation:** unknown ioLTS $I$

- **Conformance:** $I \ \text{ioco} \ \ S$ : partial inclusion of $\text{STr}(I)$ in $\text{STr}(S)$

- **Test cases:** ioSTS $TC$ + Verdicts
  - Execution: parallel composition $\Delta(I) \ || \ TC$
  - Verdicts: $TC$ may fail $I$

- **Test generation:** $\text{gen}\_\text{test}: S \rightarrow TS= \{TC_1, TC_2, \ldots\}$
  Requested Properties of $TS$: may fail $\leftrightarrow \neg I \ \text{ioco} \ \ S$
  (soundness, limit exhaustiveness)

Simplification: an automata/language point of view
Conformance relation

\[ I \ ioco \ S \triangleq \forall \sigma \in STr(S), \]
\[ Out(\Delta(I) \text{ after } \sigma) \subseteq Out(\Delta(S) \text{ after } \sigma) \]

\[ S = (V_S, \Theta_S, \Sigma, \mathcal{T}_S) \]

Init \rightarrow Rx \rightarrow Ry \rightarrow Cmp

- x=y=0
- ?start
- Rx: x:=p
- Ry: y:=p
- ?a(p)
- p=y-x \land p \geq 2
- \!ok(p)
- p=y-x \land p < 2
- \!nok(p)

Conformant traces, e.g.
- ?start . ?a(4) . ?a(6) . \!ok(2)
- ?start . \!end
- ?start . ?start
  (unspecified input allowed)

Non-conformant traces, e.g.
- ?start . ?a(5) . ?a(7) . \!ok(3)
- ?start . ?a(6) . ?a(8) . \!nok(2)

Prop: \[ I \ ioco \ S \Leftrightarrow STr(I) \ 
\Lambda^\delta \ STr(S) \] = \emptyset
[non-conformant behaviours]
Canonical Tester $\text{Can}(S)$: observer of non-conformant behaviours

1. Add new variable $\text{Verd}$ with initial value none
   i.e. $\Theta \forall \text{Verd} = \text{none}$

2. Output $\forall a$, $\forall$ t carrying $\forall a$:

$$\text{Str}_\text{Fail} (\text{Can}(S)) = [\text{Str}(S). \forall_i \delta \setminus \text{Str}(S)]$$

$$\Rightarrow \forall i \text{oico } S \iff \text{Str}(I) \bar{\lor} \text{Str}_\text{Fail} (\text{Can}(S)) = \emptyset$$
Test cases, test execution, verdicts and properties

**Test Case:** deterministic ioSTS TC = (V_{TC}, \Theta_{TC}, \Sigma, T_{TC})

+ verdict boolean variables: Fail, Pass, Inconc, ...

plays the role of an observer delivering verdicts

\[ \text{Tr}_{\text{Fail}}(TC) \]

**Test suite:** (infinite) set of test cases TS = \{TC_1, TC_2, ... \}

**Test execution:** TC || \Delta(I) synchronization on common actions

**Possible rejection of I by TC:**

reachability of Fail in TC || \Delta(I) i.e.

TC may fail I \(\triangleq\) ST(r(I) \(\wedge\) Tr_{\text{Fail}}(TC) \neq \emptyset

TS may fail I \(\triangleq\) ST(r(I) \(\wedge\) \(\bigwedge_{TC \in TS}^{\text{Tr}_{\text{Fail}}(TC)} \neq \emptyset\)
Test suite properties

Possible rejection by a TC should correspond to non-conformance and vice-versa

\[ \neg \text{TS may fail I} \iff \text{STr(I) } \notin \bigcup_{\text{TC } \in \text{TS}} \text{Tr}_{\text{Fail}}(\text{TC}) = \emptyset \]

\[ \text{I ioco S } \iff \text{STr(I) } \notin \bigcup_{\text{TC } \in \text{TS}} \text{Tr}_{\text{Fail}}(\text{Can(S)}) = \emptyset \]

TS is sound

\[ \Delta \quad \forall \text{ I, } (\text{I ioco S } \Rightarrow \neg \text{TS may fail I}) \]

\[ \iff \quad \bigcup_{\text{TC } \in \text{TS}} \text{Tr}_{\text{Fail}}(\text{TC}) \subseteq \text{Tr}_{\text{Fail}}(\text{Can(S)}) \]

TS is exhaustive

\[ \Delta \quad \forall \text{ I, } (\neg \text{TS may fail I } \Rightarrow \text{I ioco S}) \]

\[ \iff \quad \bigcup_{\text{TC } \in \text{TS}} \text{Tr}_{\text{Fail}}(\text{TC}) \supseteq \text{Tr}_{\text{Fail}}(\text{Can(S)}) \]
3. Test selection for ioSTS

TS = \{\text{Can}(S)\} is a sound and exhaustive test suite but
- has too many (infinite) behaviours
- does not allow to control the implementation during testing

⇒ Test selection
- renounce to exhaustiveness in practice,
- focus on some targetted behaviours of \text{Can}(S)
  select a finite TS likely to discover non-conformances
- use test purposes
\[ TP = (V_S \cup V_{TP}, \Theta_{TP}, \Sigma, T_{TP}) \]

Observer of actions and variables of S

\[ [a(p) : G(v_S, v_{TP}, p); v_{TP} := A(v_S, v_{TP}, p)] \in T_{TP} \]

Hyp : complete and deterministic

\[ TP^+ : \text{reachability property} \]

\[ TP^- : \text{negation of safety property} \]
General principle

**Can(S)**
- non-conformance observer
  - **S**
  - Fail

**Can(S) × TP⁺:**
- **TP⁺**
  - Reachability Observer
  - Acc
  - Violate

**Can(S) × TP⁻:**
- **TP⁻**
  - Safety Observer
  - Fail

**TC⁺(S,TP⁺)**
- non-conf. and reach. observer

**TC⁻(S,TP⁻)**
- non-conf. and safety observer
Syntactical product \( \text{Can}(S) \times \text{TP} \)

\[
\text{Can}(S) = (V_A, \Theta_A, \Sigma, T_A)
\]

\[
G_1(v_S, p) \quad a(p) \quad v' \_S := A_1(v_S, p)
\]

\[
\text{TP} = (V_S \cup V_{TP}, \Theta_{TP}, \Sigma, T_{TP})
\]

\[
G_2(v_S, v_{TP}, p) \quad a(p) \quad v' \_TP := A_2(v_S, v_{TP}, p)
\]

\[
\text{SP} = \text{Can}(S) \times \text{TP} = (V_S \cup V_{TP}, \Theta_S \cup \Theta_{TP}, \Sigma, T_{S \times TP})
\]

\[
G_1(v_S, p) \ AE \ G_2(v_S, v_{TP}, p) \quad a(p) \quad <v' \_S; v' \_TP> := <A_1(v_S, p); A_2(v_S, v_{TP}, p)>
\]
1. Assignment of Pass verdicts

Pass: $\text{Tr}_{\text{Accept}}(SP)$  

Observer of $\text{Tr}_{\text{Accept}}(SP)$

$\text{Can}(S) \times TIP$

$G(v,p)$
$a(p)$
$v := A(v,p)$

$G(v,p) \not\in \text{Verd} = \text{none}$
$a(p)$
$v := A(v,p)$

Verd := if $A_{\text{Accept}}$ then Pass
else Verd
1. Assignment of Pass verdicts

\[ S^P = \text{Can}(S) \times T^P \]
After product we get

- \( \text{Tr}(\text{SP}) = \text{Tr}(\text{Can}(S)) \)
- \( \text{Tr}_{\text{Fail}}(\text{SP}) = \text{Tr}_{\text{Fail}}(\text{Can}(S)) \)
- \( \text{Tr}_{\text{Pass}}(\text{SP}) = \text{Tr}_{\text{Accept}}(\text{TP}) \setminus \text{Str}(S) \)

SP is both a non-conformance and reachability observer
but has too much behaviours: (Tr(Can(S))

**Goal of selection:**
- focus on \( \text{Tr}_{\text{Accept}}(\text{TP}) \setminus \text{Str}(S) \),
- detect unfeasible traces to \( \text{Accept} \)
- Amounts to compute \( \text{co-reach}(\text{Accept}) \)
- Undecidable \( \Rightarrow \) over-approximate
Syntactical Test Selection (2)

2. Selection and assignment of Inconc verdicts

coreach(Accept) not computable \(\Rightarrow\) compute over-approximation:

\[
coreach^\alpha \supseteq \text{coreach}(\text{Accept})
\]

\(\forall\) assignment \(A\), \(\text{pre}^\alpha(A) (\text{coreach}^\alpha) \supseteq \text{pre}(A) (\text{coreach}^\alpha)\)

**Idea:**

\(\text{pre}^\alpha(A) (\text{coreach}^\alpha) = \text{Nec. Cond. to go into coreach}^\alpha\)

\(\neg \text{pre}^\alpha(A) (\text{coreach}^\alpha) = \text{Suf. Cond. to go outside coreach}^\alpha\)

\(\subseteq \text{outside coreach}(\text{Accept})\)
Syntactical test selection (3): guard strengthening

Rule for inputs of $s$: keep conditions leading to $\text{coreach}^\alpha$, cut other ones (controllable):

Rule for outputs of $s$: keep all conditions (uncontrollable), those leading outside $\text{coreach}^\alpha$ produce $\text{Inconc}$:
Test selection: example
1st over-approximation : control

Abstraction on control: only the location is taken into account in coreach$^\alpha$

CTG$^1$ $(S, TP)$
Test selection: example

2\textsuperscript{nd} approximation computed by NBAC (convex polyhedra)

\[ SP = \mathcal{A}_{\text{ioco}} S \times \mathcal{T}P \]

\[ \mathcal{CTG}_2 (S, \mathcal{T}P) \]
Simplification: over-approximation reach$^\alpha$ of reach(Θ)

$\text{CTG}_2(S, \mathcal{T}P)$

Simplify guards according to reach$^\alpha$ (false $\Rightarrow$ cut)

NB: semantics is unchanged
Consequences of over-approximation on test cases

For two abstractions $\alpha_1$ and $\alpha_2$
(e.g. $\alpha_1$: control vs $\alpha_2$: polyhedra)
pre$^{\alpha_1}(A)$ (coreach$^{\alpha_1}$) $\supseteq$ pre$^{\alpha_2}(A)$ (coreach$^{\alpha_2}$)
\[ \Rightarrow \]
Tr($CTG_1$) $\supseteq$ Tr($CTG_2$)

Less precise approximation $\Rightarrow$
• More infeasible traces to Accept
• More fail verdicts (all sound)

Limit cases:
• exact analysis:
  best guiding to Accept
• no analysis:
  no guiding to Accept
Test execution

\[
CTG_2 (S, TP)
\]

Inputs: [ ? a(p) : G(v,p); v:=A(v,p) ] : v is known, choose \( \pi \) s.t. \( G(v,\pi) \), by constraint solving, send \( a(\pi) \), assign \( v:=A(v,\pi) \)

Outputs: [ ! a(p) : G(v,p); v:=A(v,p) ] : v is known, receive \( a(\pi) \), evaluate \( G(v,\pi) \) if true, assign \( v:=A(v,\pi) \) (input complete)

?start . ?a(4) . ?a(6) . !ok(2) : Pass
?start . ?a(5) . ?a(7) . !ok(3) : Fail
?start . !end : Inconc
?start . !end : Inconc
Verification and Testing

Development process

- P properties
- S specification
- I implementation

S \models P \? \gamma/n/u
I \models P ?
I \text{conf} S \? n/u
Model-checking a safety property

\[ S \]

\[ \begin{align*}
\text{End} & \rightarrow \text{!end} \\
\text{Idle} & \rightarrow \text{?start} \\
\text{Rx} & \rightarrow \text{x:=p} \\
\text{Ry} & \rightarrow \text{y:=p} \\
\text{Cmp} & \rightarrow \text{?a(p)} \\
\end{align*} \]

\[ \text{p=y-x} \land \forall p \geq 2 \rightarrow !\text{ok}(p) \]

\[ \text{p=y-x} \land \forall p < 2 \rightarrow !\text{nok}(p) \]

\[ \mathcal{T}P = \mathcal{A}_\neg P \]

\[ \begin{align*}
\text{Wait} & \rightarrow \text{!nok(p)} \\
\text{Violate} & \rightarrow \text{Violate := true} \\
\end{align*} \]

Model-checking \( S \models P \) reduces to reachability in \( S \times \mathcal{A}_\neg P \) (undecidable)

\[ \begin{align*}
\text{End} & \rightarrow \text{!end} \\
\text{Wait} & \rightarrow \text{?start} \\
\text{Rx} & \rightarrow \text{x:=p} \\
\text{Ry} & \rightarrow \text{y:=p} \\
\text{Cmp} & \rightarrow \text{?a(p)} \\
\end{align*} \]

\[ \text{p=y-x} \land \forall p \geq 2 \rightarrow !\text{ok}(p) \]

\[ \text{p=y-x} \land \forall p < 2 \rightarrow !\text{nok}(p) \]

\[ \mathcal{T}P = \mathcal{A}_\neg P \]

\[ \begin{align*}
\text{Wait} & \rightarrow \text{!nok(p)} \\
\text{Violate} & \rightarrow \text{Violate := true} \\
\end{align*} \]

\[ S \models \neg P : ?\text{start.}a(10).a(0).!\text{nok(-10)} \rightarrow (\text{Rx, Violate}) \]

With any abstraction \( S^\alpha \models \neg P \) is Yes but \( S^\alpha \models \neg P \nRightarrow S \models \neg P \)

Result of \( S \models P \) could be Unknown
Test selection from a safety observer

\( \text{Can}(S) \)

\[ \mathcal{A}_{s i o c o, S} \times \mathcal{A}_{s p} \]

\( \text{Fail} \)

\( \neg \text{I ioco } S \)

\( \text{Verd} := \text{Fail} \)

\( \neg \text{I ioco } S \)

\( \text{Fail} \)

\( \text{Verd} := \text{Fail} \)

\( \text{Fail} \)

\( \text{Verd} := \text{Fail} \)

\( \text{Violate} \)

\( \neg \text{I ioco } S \)

\( \text{I} \models \neg \exists \)

\( \neg \text{I ioco } S \)

\( \text{Verd} := \text{Fail} \)

\( \text{Violate} := \text{true} \)

\( \text{Start} \).

\( \text{La}(10) \).

\( \text{La}(0) \).

\( \text{Nok}(-10) \)
Some links between Model-checking and Conformance Testing

• Test selection using model-checking:
  - $S$ deter., controllable, $P$ reachability: $TC \simeq \text{counter-exple of } S \models \neg P$
    
    [Engels et al. 97, Gargantini et al. 99]
  - Extension to coverage using CTL [Hong et al. 02] or observers [Blom et al. 04]
  - Non-controllable case is more complex (this talk)

• Checking properties on the implementation
  - Black-box checking [Peled et al.]: learn $I$ by experiment, model-check $I \models P$
Conclusion

Simplified and general framework for Ioco-based Test selection

- For finite ioLTS and infinite ioSTS
- Unified For Reachability and Safety Observers
- Using verification: coreachability analysis, over-approximations
- Completing verification (case of safety)

More research work needed for, e.g.

- Theories and algorithms for other models of reactive systems e.g. with time, data, stack, probabilities ...and combinations
- Coverage : measures, selection
- Links with structural testing techniques
- ....