Software Testing V

Prof. Dr. Holger Schlingloff
Humboldt-Universität zu Berlin
and
Fraunhofer Institute of Computer Architecture
and Software Technology FIRST
Outline of this Lecture Series

- **2006/ 11/ 24:** Introduction, Definitions, Examples
- **2006/ 11/ 25-1:** Functional testing
- **2006/ 11/ 25-2:** Structural testing
- **2006/ 11/ 26-1:** Model-based test generation
- **2006/ 11/ 26-2:** Specification-based test generation

- Next week: Your turn!
Outline of This Lecture

• Test generation from algebraic specifications
  ▪ LOTOS specification language
    - Abstract data types, terms, denotations
    - Process algebra, composition of processes
  ▪ Test generation from LOTOS ADTs
    - Testing hypotheses
    - Test suite refinement
Test Generation

• **State of the art:** Test generation from
  - executable code (coverage tools)
  - scripting languages, e.g. TTCN-3
  - graphical models, e.g. StateCharts

• **Research problems:** Test generation from
  - UML interaction diagrams
  - algebraic and logic specification languages
  - natural language
Specification Based Testing

• **(formal) Specification** = (formal) description of the intended behaviour of the SUT
  - *formal* = syntax & semantics & algorithms
  - in a certain sense, FORTRAN and C are formal specification languages
  - specification need not be executable

• **Specification transformation**
  - refinement and abstraction are relations between different specifications
  - conformance is a relation between a test suite and a specification or implementation
  - relation between test suites?
Example Language: Lotos

- **Algebraic specification language**
  - *Language of Temporal Ordering Specification*
  - defined and used in telecommunications
  - standardised (ISO 8807, 1989)
  - much theory, some practical examples
  - supports object-oriented design
  - extension / variants (e.g. CSP-CASL)

- **Syntax**
  - abstract data type
  - process algebraic behavior description

- **Semantics**
  - term algebra, equalities impose equivalence partitioning, initial semantics
  - traces, failures, divergence semantics for process part

- **Algorithms**
  - correctness proofs, model checking
  - transformational development
  - test case generation

LOTOS Syntax

- Abstract data type
  - data type identifier(s)
  - functions / operations with type
  - defining equations

- Process algebraic behaviour description
  - recursive process definitions
  - parallelism and synchronisation / communication
Example: ADT Stack

```adam
Example: ADT Stack

**type** Stack is Boolean
  **formalsorts** Element
**sorts** Stack
  **opns**
  empty : Stack - > Bool
  emptyStack : - > Stack
  push : Element, Stack - > Stack
  peek : Stack - > Element
  pop : Stack - > Stack
  **eqns**
  forall e: Element, s: Stack
  ofsort Bool
    empty(emptyStack) = true;
    empty(push(e,s)) = false;
  ofsort Element
    peek(push(e,s)) = e;
  ofsort Stack
    pop(push(e,s)) = s;
**endtype** (* Stack *)
```

Exercise:

-opn search
Semantics of Terms

- Term algebra: all well-typed expressions
- Free algebra („Herbrand-Universum“): no equations, each term is its own denotation
- Equations induce an equivalence partitioning
- Several possibilities for the semantics
  - initial semantics: smallest equivalence partitioning of the free algebra (everything is unequal unless you can show that it’s equal)
  - loose semantics: some partitioning which respects the equations
Instantiation

- Stack corresponds to an abstract class
- Concrete class:

  type NatStack is
  GenericStack actualizedby NaturalNumber
  using sortnames
      Nat for Element
      NatStack for Stack
  endtype (* NatStack *)
Further LOTOS Possibilities

- Conditional equations
- Parametrised Types (abstracte classes)
- Overloading of functions (polymorphism)
  - e.g. equality
  - `ofsort` for marking the type
- Renaming and subtyping
  - `type B is A renamedby sortnames ... for ...`
Process Part

- Extension of ADT’s by behavioral descriptions
- Base component: action
  - internal action: invisible to the outside
  - observable action: value appears at the connection point (gate)
    - $g!e$: sending of expression $e$ via gate $g$
    - $g?x:s[c]$: receiving a new value of type $s$ for variable $x$ at gate $g$ if condition $c$ holds
  - intuitively: the connection points transmit values of the corresponding abstract data type
Process Definitions

• Processes are used to denote behaviour
  - **process** $P [...] = ...$

• Three main possibilities for composition of processes
  - sequentialisation: $(P >> Q)$ or $(a; P)$ (a is an action)
  - alternatives: $([c_1]-> P \ [ ] \ [c_2]-> Q)$
  - parallelism: $(P / [g_1, ..., g_n]/ Q)$
    $(P // Q)$ and $(P /// Q)$ are abbreviations for synchronisation on all or no gate

• Recursive process definitions
  - **stop** as regular end (no action executable)
  - **exit** as return from a process definition
  - other syntactical sugaring
Communication, Synchronisation, Coordination

- $(P \parallel [g] \parallel Q)$ may
  - either perform an action from P or Q which does not concern g, or
  - perform a common action on gate g, if it is executable both for P and for Q

  - **Communication:** $g!e$ and $g?x:s[c]
    Transmitting the value $e$ to $x$, if $c$ holds
  - **Synchronisation:** $g!e_1$ and $g!e_2
    If $e_1 = e_2 = e$, then $e$ appears at $g$
  - **Coordination:** $g?x_1:s[c_1]$ and $g?x_2:s[c_2]
    At $g$ some value $e$, appears which satisfies $c_1$ and $c_2$
Semantics of Processes

- If two parallel processes cannot synchronize, deadlock results
  - $g!5 \parallel [g] \parallel g?x:\text{Nat} [x>7]$
  - $g_1!5 \parallel [g_1, g_2] \parallel g_2?x:\text{Nat}$

- Event $(g,e)$: executing an action $g!e$ or $g?x$ where $x=e$

- Traces$(P)$: set of all sequences of observable events of a process

- Trace-, failure-, divergence-semantics
Examples for Processes

process Boss [in] : noexit = 
choice item : Nat_Sort [] in!item >> Boss [in] endproc

process ToDo [in, out] (liste: Stack) : noexit =
(in?item; ToDo[in,out](push(item,liste))
  []
  [not empty(liste)] -> out!peek(liste); ToDo[in,out](pop(liste))
)
  endproc

process Slave [out] : noexit =
out? x; i >> Slave [out]
  endproc

System specification: Boss [[in]] ToDo [[out]] Slave
Further Language Constructs

- In the process part, you can use
  - parameterised processes
    \[
    \text{process } P[g_1,g_2](p_1:s_1, p_2:s_2) : \text{exit} = \ldots \text{ endproc}
    \]
  - local variable definitions
    \[
    \text{let name : sorte = expr in } \ldots
    \]
  - generalised sequences, alternatives, parallelism
    \[
    \text{expr}_1 >> \text{accept pardef in expr}_2
    \]
    \[
    \text{choice } g \text{ in } [a_1, a_2, a_3] [ ] B [g]
    \]
    \[
    \text{par } g \text{ in } [a_1, a_2, a_3] \parallel B [g]
    \]
  - disabling, hiding, locale processes, ...
    \[
    P[> Q, \text{hide } g \text{ in } P, \text{where process} P = \ldots
    \]
  - a module concept
    \[
    \text{library importierte Datentypen endlib}
    \]
LOTOS System Specification

**specification** S [a, b, c, d] : noexit

**library** predefined Data types **endlib**

type ExampleType is
  sorts ExampleSorts
  opns ExampleOperations: ExampleSorts - > ExampleSorts
  endtype

**behaviour**
  (P [a, b, c] ||[b]|| Q [b, d])

**where**
  process P[a, b, c] ... **endproc**
  process Q[b, d] ... **endproc**

**endspec**
A Larger Example (part 1)

specification Example1 : exit

library Boolean, OctetString, NaturalNumber

type Message is
  Octet, NaturalNumber, Boolean

sorts
  Message

opns
  ε :  -> Message
  . . . : Octet, Message-> Message
  Pack : Message, Message -> Message
  Size : Message -> Nat

eqns
  forall m1, m2: Message, o1: Octet
    ofsort Message
    Pack(ε, m1) = m1;
    Pack(b.m1, m2) = b.Pack(m1, m2);
  ofsort Nat
    Size(ε) = 0;
    Size(o1.m1) = Succ(Size(m1));

datatype
A Larger Example (part 2)

Assume we are given a program which claims to implement this specification. How can we test it?

Source:
Test Generation from ADTs

• Given ADT $Spec = (\Sigma, Eq)$
  - implementation $Imp$ is correct wrt. $Spec$ if all axioms are satisfied for all terms
  - term-generated models
  - test case for universally quantified formula is one particular instance

• Test case: ground instance of axiom
  - e.g. $\text{pop}(\text{push}(\text{"a"}, \text{emptyStack})) = \text{emptyStack}$
  - Problem: how to choose terms?

• Test verdict: evaluation of instance
  - may be arbitrarily hard, even undecidable
  - problems: non-primitive data types, partial functions

Procedure for Test Generation

- **Exhaustive test suite:** if all tests in the suite pass, then the implementation is correct
- Testing hypotheses
  - regularity
  - uniformity
  - observational context
- **Complete** test suite wrt. test hypothesis
- Test suite **refinement**
  - stronger hypotheses
  - more errors detected
Exhaustive Test Suites

- **Test suite** $T$: set of ground formulas
  - Assumption: each object is term generated
  - Example: $s(s(s(z))) = p(s(z), s(s(z)))$

- **Test oracle** $O \subseteq T$ (bzw. $O: T \rightarrow \{\text{true, false}\}$)
  - determines for each test case whether it passes or fails (follows from the axioms or not)
  - e.g. $s(s(s(z))) = p(s(z), s(s(z))) \rightarrow \text{true}$
  - in general this problem is undecidable!

- **Exhaustive test suite**: a set of test cases, such that the following holds: if all test cases pass, then the implementation is correct
  - in general infinite
  - how to find an approximation?
Testing Context

- **Testing context TC**: \((T, O, H)\)
  - test suite \(T\) (set of ground terms)
  - test oracle \(O \subseteq T\) (or \(O : T \rightarrow \{\text{true}, \text{false}\}\))
  - testing hypothesis \(H\) for the implementation

\[ H \land O = T \rightarrow \text{Correct(Imp,Spec)} \]

- **Minimal testing hypothesis**: "empty assumption"
  - the set of all derivable ground formulas, or the set of all ground instances of equalities is a complete test suite

- **Maximal testing hypothesis**: "Imp is correct"
  - empty set is a complete test suite
Testing Hypotheses

- Regularity hypothesis
  - “The SUT contains no irregularities”

- Uniformity hypothesis
  - “the SUT acts uniform on its data”

- Observability hypothesis
  - “the SUT data can be identified by finite observations”
Regularity Hypothesis

• Assume a complexity measure for formulas

• **Regularity hypothesis:** If some statement $A$ holds for all formulas up to a certain size $\delta$, then $A$ holds for all formulas

• Allows to restrict attention to those test cases smaller than $\delta$
  - e.g. $p(x,y)=p(y,x)$ for $|x|<3$, $|y|<3$
Uniformity Hypothesis

- Assume a property of expressions
- **Uniformity hypothesis**: If any statement holds for all formulas containing expressions with this property, then it holds for all formulas

- Generalisation of the regularity hypothesis
- Allows to restrict test cases to certain variable patterns

- Application: partitioning of domains
- extrem case: collapsing a domain to a single representative
  - cf. abstraction of variables in the previous lecture
Observability Hypothesis (1)

• Equality of primitive data types (boolean, integer,...) is observable
• How to observe equality of compound (non-primitive) data types?
  ▪ special equality function in Imp?
    ➔ transfers the problem
  ▪ component-wise comparison: replace \( x = y \) by \( C_1(x) = C_1(y) \), \( C_2(x) = C_2(y) \), ...

• **Observable context:** Mapping of compound to primitive data type
• **Leibnitz‘s extensionality principle:** two object are identical if they behave equally in each observable context
Observability Hypothesis (2)

- Leibnitz’ principle can be used for testing

- Problem: “very many” possible contexts
- Fix a set of contexts for each compound data type
- **Observability hypothesis**: If any statement holds for all observable contexts, than it holds for all formulas

- special case of uniformity hypothesis
- Allows reduction to primitive comparisons

- Example: top and second stack element, hash or similar
Test Suite Refinement

• TC₂ refines TC₁ (TC₂<TC₁), if
  ▪ TC₂ has stronger hypotheses than TC₁
    \[ H_2 \rightarrow H_1 \]
  ▪ TC₂ can discover at least as many faults as TC₁
    \[ \text{failed}(T_1,\text{Imp}) \rightarrow \text{failed}(T_2,\text{Imp}) \]
  ▪ TC₂ has more passed tests than TC₁
    \[ \text{pass}(T_2,\text{Imp}) \rightarrow \text{pass}(T_1,\text{Imp}) \]

• The set of all ground terms with empty testing hypothesis is the largest test suite in this partial order

• Test case development = Adding hypotheses to this largest test suite
Method of Refinement

- Starting with the exhaustive test suite, add
  - Regularity hypotheses for defined types
  - Uniformity hypotheses for imported types
  - Observation functions and -hypotheses for compound types
- ... until the test suite is complete and the oracle is decidable and well-defined
- Tool support possible
Example

\[
\text{exhaust}_{ax1} = \{ \text{Pack}(\varepsilon, m) = m \mid m \in T_{\text{Message}} \} \\
\text{exhaust}_{ax2} = \{ \text{Pack}(o_1.m_1, m) = o_1.\text{Pack}(m_1, m) \mid o_1 \in T_{\text{Octet}}, m_1, m \in T_{\text{Message}} \} \\
\text{exhaust}_{ax3} = \{ \text{Size}(\varepsilon) = 0 \} \\
\text{exhaust}_{ax4} = \{ \text{Size}(o_1.m) = \text{Succ}(\text{Size}(m)) \mid o_1 \in T_{\text{Octet}}, m \in T_{\text{Message}} \}
\]

- Uniformity hypothesis "all variables equal" yields 4 test cases
- "Unfolding" of \textit{Pack} yields new test cases

\[
\begin{align*}
\text{Pack}(o_1.\varepsilon, m) &= o_1.m \\
\text{Pack}(o_1.o'_1.m'_1, m) &= o_1.o'_1.\text{Pack}(m'_1, m)
\end{align*}
\]
Testing the process part

- For ADT
  - correctness of Imp wrt. Spec = all equations / formulas of Spec are satisfied by Imp

- For PA
  - correctness defined by observable behaviour
  - simulation or containment of behaviour
  - ioco
Test generation for Full Lotos

- One test suite for the data part and one for the process part
- Use of the data type properties in testing of the processes
  - Example: Splitting $\leq$ into $<$ and $=$
  - Example: $\text{Size}(\varepsilon)=0$, $\text{Size}(\text{o.m})=\ldots$ yields four test cases for the process part $[\text{Size}(x)+\text{Size}(y)<\text{Max}] \rightarrow \ldots$
  - Calculation of limit values from the equations
Some test cases for the example

Find tests for *Compact*(7):

- control!4; ...  
- control!0; ...  
- inGate!H.E.L.L.O.ε; inGate!W.O.R.L.D.ε;  
  outGate!H.E.L.L.O.ε!W.O.R.L.D.ε; ...  
- inGate!H.E.L.ε; inGate!W.O.ε; outGate!H.E.L.W.O.ε; ...  

Uniformity hypotheses and representative values for Max, newMax,  
and Size(x)+Size(y)>Max  
Research: systematic derivation by an algorithm