CS 5154

Syntax-based Testing

Owolabi Legunsen

The following are modified versions of the publicly-available slides for Chapter 9 in the Ammann and Offutt Book, “Introduction to Software Testing” (http://www.cs.gmu.edu/~offutt/softwaretest)
Syntax-based Testing

Four Structures for Modeling Software

- Input Space
- Graphs
- Logic
- Syntax

Applied to

- Source
- FSMs
- Specs
- DNF

Applied to

- Source
- Models
- Integ
- Input

Design
Use cases

Specs

Introduction to Software Testing, Edition 2 (Ch 07) © Ammann & Offutt
Using Syntax to Generate Tests

• Lots of software artifacts follow strict syntax rules
  – Syntax is often expressed as a grammar in a language, e.g., BNF

• Syntactic descriptions can come from many sources
  – Programs, integration elements, design docs, input descriptions

• Syntax-based tests are created with two general goals
  – Cover the syntax in some way
  – Violate the syntax (invalid tests)
Grammar Coverage Criteria

- Software engineers use automata theory in several ways
  - Programming languages defined in BNF
  - Program behavior described as finite state machines
  - Allowable inputs defined by grammars

- A simple regular expression:
  \[(G s n | B t n)^*\]

- Any sequence of “G s n” and “B t n”
- ‘G’ and ‘B’ could represent commands, methods, or events
- ‘s’, ‘t’, and ‘n’ can represent arguments, parameters, or values
- ‘s’, ‘t’, and ‘n’ could represent literals or a set of values
Test Cases from the Regex

- Strings satisfying the derivation rules are “in the grammar”

- Test: a sequence of strings that satisfy the regex

- Suppose ‘s’, ‘t’ and ‘n’ are numbers

<table>
<thead>
<tr>
<th>G 26 08.01.90</th>
</tr>
</thead>
<tbody>
<tr>
<td>B 22 06.27.94</td>
</tr>
<tr>
<td>G 22 11.21.94</td>
</tr>
<tr>
<td>B 13 01.09.03</td>
</tr>
</tbody>
</table>

Could be one test with four parts or four separate tests, etc.
BNF Grammars

Stream ::= action*

action ::= actG | actB

actG ::= “G” s n

actB ::= “B” t n

s ::= digit<1-3>

t ::= digit<1-3>

n ::= digit<2> “.” digit<2> “.” digit<2>

digit ::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”
Using Grammars

Stream ::= action action *
  ::= actG action*
  ::= G s n action*
  ::= G digit^1-3 digit^2 . digit^2 . digit^2 action*
  ::= G digitdigit digitdigit.digitdigit.digitdigit action*
  ::= G 25 08.01.90 action*
  ...

- **Recognizer**: Is a string (or test) in the grammar?
  - This is called parsing
  - Tools exist to support parsing
  - Programs can use them for input validation

- **Generator**: Derive strings that are in a given grammar
Grammar-based Coverage Criteria

• The most common and straightforward criteria use every terminal and every production at least once

**Terminal Symbol Coverage (TSC):** TR contains each terminal symbol $t$ in the grammar $G$.

**Production Coverage (PDC):** TR contains each production $p$ in the grammar $G$.

• PDC subsumes TSC
• Grammars and graphs are interchangeable
  – PDC is equivalent to EC, TSC is equivalent to NC
• Other graph-based coverage criteria could be defined on grammar
  – But have not
Grammar-based Coverage Criteria (2)

• A related criterion involves deriving all possible strings from the grammar

**Derivation Coverage (DC):** TR contains every possible string that can be derived from the grammar \( G \).

• DC often requires an impractical number of tests…
Number of tests produced by Grammar-based Criteria

- Number of TSC tests is bound by the number of terminal symbols
  - 13 in the stream grammar

- The number of PDC tests is bound by the number of productions
  - 18 in the stream grammar

- The number of DC tests depends on the details of the grammar
  - 2,000,000,000 in the stream grammar!

- All TSC, PDC and DC tests are in the grammar ... how about tests that are NOT in the grammar?
Mutation as Grammar-Based Testing

Grammar-based Testing

UnMutated Derivations (valid strings)

Mutated Derivations (invalid strings)

Grammar Mutation (invalid strings)

Ground String Mutation

Invalid Strings

Valid Strings
Mutation Testing

- Grammars describe both valid and invalid strings

- Both types can be produced as mutants

- A mutant is a variation of a valid string
  - Mutants may be valid or invalid strings

- Mutation is based on “mutation operators” and “ground strings”
What is Mutation?

General View

We are performing mutation analysis whenever we
• use well defined rules
• defined on syntactic descriptions
• to make systematic changes
• to the syntax or to objects developed from the syntax

mutation operators
grammars
Applied universally or according to empirically verified distributions

grammar
ground strings (tests or programs)
Mutation Testing

- **Ground string**: A string in the grammar
  - “ground” is used as an analogy to algebraic ground terms

- **Mutation Operator**: A rule that specifies syntactic variations of strings generated from a grammar

- **Mutant**: Result of one application of a mutation operator
  - a string in the grammar or close to being in the grammar
Mutants and Ground Strings

- The key to mutation testing: **design** of mutation operators
  - Well-designed **operators** lead to powerful testing
  - Well-designed or not?: change all predicates to true and false
- Sometimes **mutants** are based on ground strings
- Sometimes they are derived directly from the grammar
  - **Ground** strings are used for **valid** tests
  - **Invalid** tests do not need ground strings

<table>
<thead>
<tr>
<th>Valid Mutants</th>
<th>Invalid Mutants</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ground Strings</strong></td>
<td><strong>Mutants</strong></td>
</tr>
<tr>
<td>G 26 08.01.90</td>
<td>B 26 08.01.90</td>
</tr>
<tr>
<td>B 22 06.27.94</td>
<td>B 45 06.27.94</td>
</tr>
<tr>
<td>7 26 08.01.90</td>
<td>B 22 06.27.1</td>
</tr>
</tbody>
</table>
Two Questions About Mutation

• Apply **more than one operator** at the same time?
  – Should mutated strings contain multiple mutated elements?
  – Usually not: multiple mutations may interfere with each other
  – Experience with program-based mutation indicates not
  – Recent research is finding exceptions

• Consider **all possible applications** of a mutation operator?
  – Necessary with program-based mutation (subsumption)
Mutation Operators are often language-based

- Mutation operators have been defined for many languages
  - Programming languages (*Fortran, Lisp, Ada, C, C++, Java*)
  - Specification languages (*SMV, Z, Object-Z, algebraic specs*)
  - Modeling languages (*Statecharts, activity diagrams*)
  - Input grammars (*XML, SQL, HTML*)
Testing Goal: Killing Mutants

- **Hope:** Mutants created as valid strings from ground strings should exhibit **different behavior** from the ground string.

- Normally used when grammars are **prog. languages**, strings are **programs**, and ground strings are **pre-existing programs**.

- **Killing Mutants**: Given a mutant $m \in M$ for a derivation $D$ and a test $t$, $t$ is said to kill $m$ if and only if the output of $t$ on $D$ is different from the output of $t$ on $m$.

- $D$ may be shown as list of productions or as the final string.
Syntax-based Coverage Criteria

• Coverage is defined in terms of killing mutants

**Mutation Coverage (MC):** For each $m \in M$, TR contains exactly one requirement, to kill $m$.

• Coverage in mutation equates to killing mutants

• **Mutation score:** ratio of mutants killed over all mutants
Syntax-based Coverage Criteria

• When creating invalid strings, we just apply the operators
• This results in two simple criteria
• It makes sense to either use every operator once or every production once

**Mutation Operator Coverage (MOC):** For each mutation operator, TR contains exactly one requirement, to create a mutated string $m$ that is derived using the mutation operator.

**Mutation Production Coverage (MPC):** For each mutation operator, TR contains several requirements, to create one mutated string $m$ that includes every production that can be mutated by that operator.
Example

Stream ::= action*
action ::= actG | actB
actG ::= “G” s n
actB ::= “B” t n
s ::= digit¹⁻³
t ::= digit¹⁻³
n ::= digit² “.” digit² “.” digit²
digit ::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

Grammar

Ground String
G 25 08.01.90
B 21 06.27.94

Mutation Operators
• Exchange actG and actB
• Replace digits with all other digits

Mutants using MPC
B 25 08.01.90  G 21 06.27.94
G 15 08.01.90  B 22 06.27.94
G 35 08.01.90  B 23 06.27.94
G 45 08.01.90  B 24 06.27.94
... ...
Mutation Testing

• **Number of test requirements** depends on two things
  – The *syntax* of the artifact being mutated
  – The mutation *operators*

• Mutation testing is very difficult to apply by hand

• Mutation testing is very effective – sometimes considered the “*gold standard*” of testing

• Mutation testing is often used to *evaluate* other criteria
  – How good is your test suite?
Instantiating Grammar-Based Testing

Grammar-Based Testing

Program-based
- String mutation
  - Compiler testing
    - Program mutation
    - Valid strings
    - Mutants are not tests
    - Tests must kill mutants
  - Valid and invalid strings

Integration
- String mutation
  - Test how classes interact
    - Valid strings
    - Mutants are not tests
    - Must kill mutants
    - Includes OO

Model-Based
- String mutation
  - FSMs
  - Model checking
  - Valid strings
  - Traces are tests

Input-Based
- String mutation
  - Input validation testing
  - XML and others
  - Invalid strings
  - No ground strings
  - Mutants are tests
Next

• Mutation testing (including a demo 😊)

• We should release scores on HW1 and HW2 by end of week
  – Drop deadline?

• Course Project…
Stats about your preferences

- **Groups:**
  - 20 expressed no preference for teammates
  - 26 expressed preferences for teammates in a consistent way
  - 18 expressed preferences in an inconsistent way

- **Reasons for preferences**
  - Asia time zone
  - Previous working relationship
  - Personal reasons and friendships

- **Options:**
  - ~50 chose option 1
  - ~10 chose option 2
  - ~4 chose options 3 and 4
My decision

• Groups:
  – 58 will form groups of 3 (one group will have 4)
  – 6 will form two groups of 3 based on Asia Time Zone

• Reasons for preferences
  – Asia time zone
  – Previous working relationship
  – Personal reasons and friendships

• Options:
  – ~62 decided on option 1
  – ~2 are still on option 2 but have no one to work with
Next Steps on Course Project

• Project requirements and groups will be released soon

• Spend time meeting your group mates
  – We may dedicate some time in the next class for you to meet
  – We may also have you do HW3 with your project team to facilitate bonding

• Keep working on your course project through the rest of the semester
  – 35% of your course grade
  – Contributions will be clear(er) from the CI server

• We may hold project office hours so you can ask questions