Software Testing, Quality Assurance and Maintenance Winter 2010

Lecture 25 — March 8, 2010

Patrick Lam *version 1*

Input Space Grammars

We've talked about generating inputs using grammars briefly last week (L22). Let's return to this topic in a bit more detail.

Refer to the book for an XML example. We'll see the book's debit/deposit example.

Grammar.

```
roll    =  action*
action  =  dep | deb
dep     =  "deposit" account amount
deb     =  "debit" account amount
account =  digit { 3 }
amount  =  "$" digit+ "." digit { 2 }
digit   =  ["0" - "9"]
```

Examples of valid strings.

Note: creating a grammar for a system that doesn't have one, but should, is a useful QA exercise. Using this grammar at runtime to validate inputs can improve software reliability, although it makes tests generated from the grammar less useful.

Testing Invalid Inputs

Reasons to test invalid inputs:

- Often implementers overlook invalid inputs;
- Undefined behaviour no longer acceptable (e.g. buffer overruns).

One way to get invalid inputs: mutate grammars and generate test strings from the mutated grammars.

Some Grammar Mutation Operators.

- Nonterminal Replacement; e.g.
  dep = "deposit" account amount $\Longrightarrow$ dep = "deposit" amount amount

  (Use your judgement to replace nonterminals with similar nonterminals.)

- Terminal Replacement; e.g.
  amount = "$" digit$^+$ "." digit { 2 } $\Longrightarrow$ amount = "$" digit$^+$ "$" digit { 2 }

- Terminal and Nonterminal Deletion; e.g.
  dep = "deposit" account amount $\Longrightarrow$ dep = "deposit" amount

- Terminal and Nonterminal Duplication; e.g.
  dep = "deposit" account amount $\Longrightarrow$ dep = "deposit" account account amount

Using grammar mutation operators.

1. mutate grammar, generate (invalid) inputs; or,
2. use correct grammar, but mis-derive a rule once—gives "closer" inputs (since you only miss once.)

Some notes:

- Book claims we don't have much experience using grammar-based operators.
- Can generate strings still in the grammar even after mutation.
- Recall that we aren't talking about semantic checks.
- Some programs accept only a subset of a specified larger language, e.g. Blogger HTML comments. Then testing intersection is useful.

Summary of Syntax-Based Testing.

	Program-based	Input Space
Grammar	Programming language	Input languages / XML
Summary	Mutates programs / tests integration	Input space testing
Use Ground String?	Yes (compare outputs)	No
Use Valid Strings Only?	Yes (mutants must compile)	Invalid only
Tests	Mutants are not tests	Mutants are tests
Killing	Generate tests by killing	Not applicable

Notes:

- Program-based testing has notion of strong and weak mutants; applied exhaustively, program-based testing subsumes many other techniques.
- Sometimes we mutate the grammar, not strings, and get tests from the mutated grammar.