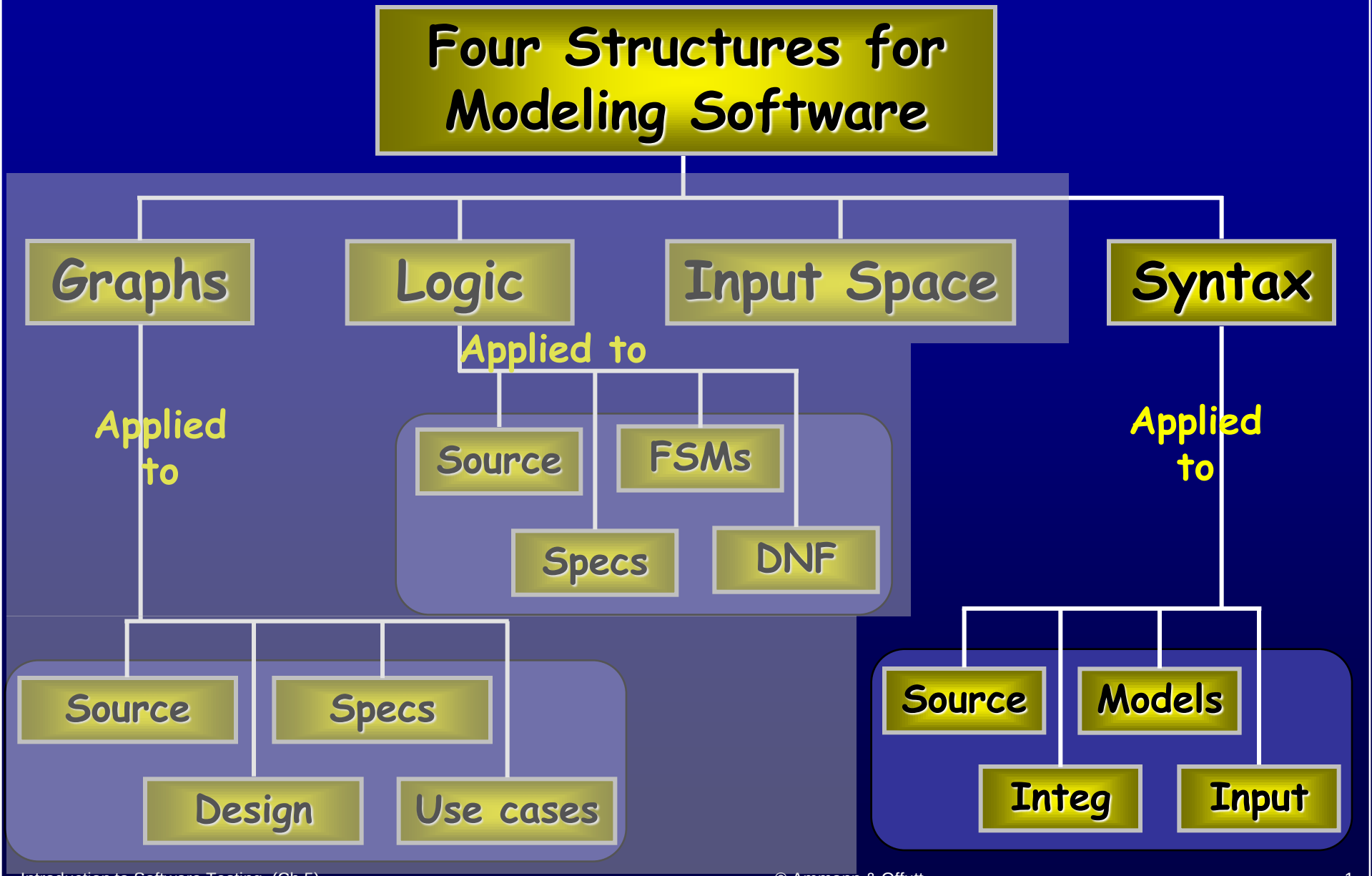
Ch. 5 : Syntax Coverage



Using the Syntax to Generate Tests (5.1)

- Lots of software artifacts follow strict syntax rules
- The syntax is often expressed as some sort of grammar such as BNF
- Syntactic descriptions can come from many sources
  - Programs
  - Integration elements
  - Design documents
  - Input descriptions
- Tests are created with two general goals
  - Cover the syntax in some way
  - Violate the syntax (invalid tests)

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”

Start symbol

Non-terminals

Production rule

Terminals

Test Cases from Grammar

- A string that satisfies the derivation rules is said to be "*in the grammar*"
- A test case is a sequence of strings that satisfy the regular expression
- Suppose 's', 't' and 'n' are numbers

G 20 08.01.90

B 16 06.27.94

G 15 11.21.94

B 07 01.09.03

Could be one test with four parts, four separate tests, . . .

Exercise 1

- Consider the grammar providedfor Canadian postal codes (taken from the lecture materials of Neal R. Wagner)
- Which of the following strings are invalid? Why?
  - K1N 6N5
  - B3D 1Z7
  - M5W 2E4
  - X0A 1A1
  - A4A CE3

Postalcode ::= ForwardSortationArea Space LocalDeliveryUnit

ForwardSortationArea ::= Letter Digit Letter

LocalDeliveryUnit ::= Digit Letter Digit

Space ::= " "

Letter ::= "A" | "B" | "C" | "E" | "G" | "H" | "J" | "K" | "L" | "M" | "N" | "P"
| "R" | "S" | "T" | "V" | "W" | "X" | "Y" | "Z"

Digit ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Exercise 1

- Consider the grammar providedfor Canadian postal codes (taken from the lecture materials of Neal R. Wagner)
- Which of the following strings are invalid? Why?
  - K1N 6N5: OK
  - B3D 1Z7: "D" does not exist in the list of terminals
  - M5W 2E4: OK
  - X0A 1A1: OK
  - A4A CE3: Second part cannot start with a letter

Postalcode ::= ForwardSortationArea Space LocalDeliveryUnit

ForwardSortationArea ::= Letter Digit Letter

LocalDeliveryUnit ::= Digit Letter Digit

Space ::= " "

Letter ::= "A" | "B" | "C" | "E" | "G" | "H" | "J" | "K" | "L" | "M" | "N" | "P"
| "R" | "S" | "T" | "V" | "W" | "X" | "Y" | "Z"

Digit ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Syntax-based Coverage Criteria

- The most common and straightforward 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

Exercise 2

- **Provide a set of test inputs based on the grammar below such that**
  - **TSC is satisfied**
  - **PDC is satisfied**

```
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"
```

Exercise 2

- **Provide a set of test inputs based on the grammar below such that**
  - **TSC is satisfied (There are 13 Terminal symbols: G, B, ., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, all of which should take place in the test cases)**
  - **PDC is satisfied (There are 18 productions, all of which must be exercised)**
    - **Note that the '|' symbol adds productions, so "action" has two productions and "digit" has 10.**

```
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"
```

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"

Mutation Testing

- **Ground string: A string in the grammar**
  - **The term "ground" is used as a reference to algebraic ground terms**
- **Mutation Operator : A rule that specifies syntactic variations of strings generated from a grammar**
- **Mutant : The result of one application of a mutation operator**
  - **A mutant is a string**

Mutants and Ground Strings

- The key to mutation testing is the **design** of the mutation operators
  - Well designed **operators** lead to powerful testing
- Sometimes **mutant strings** 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

Valid Mutants

Ground Strings	Mutants
G 20 08.01.90	*B 20 08.01.90*
B 16 06.27.94	*B 45 06.27.94*

Invalid Mutants

7 20 08.01.90

B 134 06.27.1

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

Grammar

```
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"
```

Ground String

G 20 08.01.90

B 16 06.27.94

Mutation Operators

- *Exchange actG and actB*
- *Replace digits with other digits*

Mutants using MOC

B 20 08.01.90

B 19 06.27.94

Mutants using MPC

B 20 08.01.90	*G 16 06.27.94*
G 20 08.01.90	*B 11 06.27.94*
G 30 08.01.90	*B 13 06.27.94*
G 40 08.01.90	*B 14 06.27.94*
G 50 08.01.90	*B 15 06.27.94*
…	…

Mutation Testing

- The number of test requirements for mutation 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 – considered the "gold standard" of testing
- Mutation testing is often used to evaluate other criteria