Fu(t|z)zing with Grammars

Moeketsi Raselimo
raselimm@hu-berlin.de
(joint work with Jan Taljaard and Bernd Fischer)

Grammar-Based Testing

Test suite construction:

prog → **module** *prio* id **=** *block* .
prio → [num]
block → **begin** (*decl* **;**)* (*stmt* **;**)* **end**
decl → **var** id : *type*
type → **bool** | **int**
stmt → **if** *expr* **then** *stmt* (**else** *stmt*)? |
  while *expr* **do** *stmt* | id **=** *expr* | *block*
expr → *expr* **=** *expr* | *expr* **+** *expr* | **(** *expr* **)** | id | num

grammar *G*

sentence generation

test suite *TS* ⊆ L(*G*)

```
module[1] x = begin begin end; end.
module[2] y = begin end.
module[3] z = begin x = (y); end.
module[1] z = begin x = x + y; end.
module[2] x = begin y = z; end.
module[3] z = begin x = z = y; end.
module[1] y = begin y = 1; end.
module[2] y = begin if x then begin end; end.
module[3] y = begin var x : bool; end.
module[2] z = begin var z : int; end.
module[1] x = begin while x do begin end; end.
...
```

execution

... on unit under test U

Testing:

- some test fails ⇒ L(*G*) ⊈ L(*U*)
  - since *TS* ⊆ L(*G*)
- (all) tests pass ⇒ L(*G*) = L(*U*)?
- what if L(*G*) ⊊ L(*U*)?
  - *U* can never fail on *TS*!

Systematic construction of positive test suites

(all) tests pass $\Rightarrow$ L(G) = L(U)?

Grammar-Based Testing Assumptions

Key assumption #1: Bigger is Better

> Better input space coverage gives better system coverage.

Corollary #1: Longer is Better

> Longer derivations give better input space coverage.

Corollary #2: Harder is Better

> More complex derivations give better input space coverage.

Problem: Size Matters…

> We need to balance test suite size and system coverage.

Grammar-Based Test Suite Adequacy

When is good enough good enough?

Define different **test data adequacy criteria** in terms of grammar **elements** and **derivations**.

Compare to traditional program coverage criteria:

- **statement** coverage
  (each statement is executed)
- **branch** coverage
  (each branch is taken)
- **MCDC** coverage
  (each sub-condition is independently evaluated to true and false)
- ???

symbol coverage
(each symbol is used in a derivation)

rule coverage
(each rule is used in a derivation)

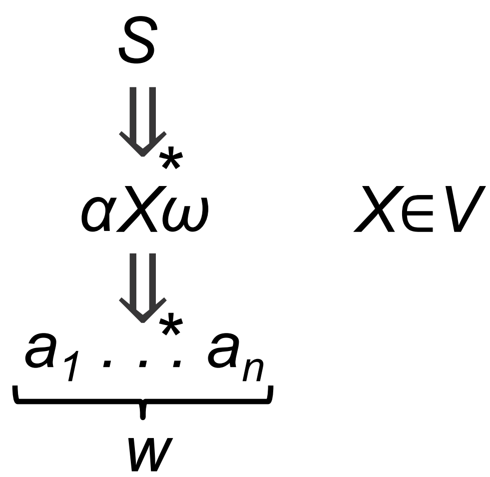
CDRC coverage
(each rule's rhs is used at each occurrence of its lhs in the rhs of other rules)

k-step coverage
(each derivation $X \Rightarrow^l \alpha Y \omega$ ($l \leq k$) is used to produce a word)

A Family of Grammar-Based Test Suite Adequacy Criteria

Assume: **grammar** $G=(N,T,P,S)$, $V=N \cup T$, **test suite** $TS \subseteq L(G)$.

Symbol: A word w covers a symbol $X \in V$ iff $S \Rightarrow^* \alpha X \omega \Rightarrow^* w$.
TS satisfies **symbol coverage** iff each X is covered by a word $w \in TS$.



A Family of Grammar-Based Test Suite Adequacy Criteria

Assume: **grammar** $G=(N,T,P,S)$, $V=N \cup T$, **test suite** $TS \subseteq L(G)$.

Symbol: A word w covers a symbol $X \in V$ iff $S \Rightarrow^* \alpha X \omega \Rightarrow^* w$.
TS satisfies **symbol coverage** iff each X is covered by a word $w \in TS$.

$$S \overset{*}{\Downarrow} \alpha X \omega \overset{*}{\Downarrow} \underbrace{a_1 \ldots a_n}_{w} \qquad X \in V$$

Rule: A word w covers a rule $p = A \to \gamma \in P$ iff $S \Rightarrow^* \alpha A \omega \Rightarrow \alpha \gamma \omega \Rightarrow^* w$.
TS satisfies **rule coverage** iff each p is covered by a word $w \in TS$.

$$S \overset{*}{\Downarrow} \alpha A \omega \Downarrow \alpha \gamma \omega \overset{*}{\Downarrow} \underbrace{a_1 \ldots a_n}_{w} \qquad A \to \gamma \in P$$

A Family of Grammar-Based Test Suite Adequacy Criteria

Assume: **grammar** $G=(N,T,P,S)$, $V=N \cup T$, **test suite** $TS \subseteq L(G)$.

CDRC: **context-dependent rule coverage** requires that each non-terminal B on the right-hand side of a rule $A \rightarrow \beta B \delta \in P$ is expanded with each rule $B \rightarrow \gamma \in P$.

$$S \overset{*}{\Downarrow} \alpha A \omega \qquad A \rightarrow \beta B \delta \in P$$

$$\alpha A \omega \Downarrow \alpha \beta B \delta \omega \qquad B \rightarrow \gamma \in P$$

$$\alpha \beta B \delta \omega \Downarrow \alpha \beta \gamma \delta \omega \overset{*}{\Downarrow} \underbrace{a_1 \ldots a_n}_{w}$$

A Family of Grammar-Based Test Suite Adequacy Criteria

Assume: **grammar** $G=(N,T,P,S)$, $V=N\cup T$, **test suite** $TS\subseteq L(G)$.

CDRC: context-dependent rule coverage requires that each non-terminal B on the right-hand side of a rule $A \rightarrow \beta B \delta \in P$ is expanded with each rule $B \rightarrow \gamma \in P$.

$$S \Rightarrow^* \alpha A \omega \Rightarrow \alpha\beta B\delta\omega \Rightarrow \alpha\beta\beta' C\delta'\delta\omega \Rightarrow \alpha\beta\beta'\gamma\delta'\delta\omega \Rightarrow^* \underbrace{a_1 \ldots a_n}_{w}$$

$$A \rightarrow \beta B \delta \in P \qquad B \rightarrow \beta' C \delta' \in P \qquad C \rightarrow \gamma \in P$$

3-step

k-step: each derivation $X \Rightarrow^l \alpha Y \omega$ ($l \leq k$) used to produce a word

$$S \Rightarrow^* \alpha X \omega \Rightarrow^* \underbrace{a_1 \ldots a_n}_{w} \qquad X \in V$$

0-step

$$S \Rightarrow^* \alpha A \omega \Rightarrow \alpha\gamma\omega \Rightarrow^* \underbrace{a_1 \ldots a_n}_{w} \qquad A \rightarrow \gamma \in P$$

1-step

$$S \Rightarrow^* \alpha A \omega \Rightarrow \alpha\beta B\delta\omega \Rightarrow \alpha\beta\gamma\delta\omega \Rightarrow^* \underbrace{a_1 \ldots a_n}_{w} \qquad A \rightarrow \beta B \delta \in P, \quad B \rightarrow \gamma \in P$$

2-step

A Family of Grammar-Based Test Suite Adequacy Criteria

Assume: **grammar** $G=(N,T,P,S)$, $V=N\cup T$, **test suite** $TS\subseteq L(G)$.

CDRC: context-dependent rule coverage requires that each non-terminal B on the right-hand side of a rule $A \rightarrow \beta B\delta \in P$ is expanded with each rule $B \rightarrow \gamma \in P$.

k-step: each derivation $X \Rightarrow^{l} \alpha Y\omega$ ($l \leq k$) used to produce a word

$$S \Downarrow_* \alpha X\omega \Downarrow_* \underbrace{a_1 \ldots a_n}_{w} \qquad X\in V$$

symbol

$$S \Downarrow_* \alpha A\omega \Downarrow \alpha\gamma\omega \Downarrow_* \underbrace{a_1 \ldots a_n}_{w} \qquad A\rightarrow\gamma \in P$$

rule

$$S \Downarrow_* \alpha A\omega \Downarrow \alpha\beta B\delta\omega \Downarrow \alpha\beta\gamma\delta\omega \Downarrow_* \underbrace{a_1 \ldots a_n}_{w} \qquad A \rightarrow \beta B\delta \in P,\ B \rightarrow \gamma \in P$$

CDRC

$$S \Downarrow_* \alpha A\omega \Downarrow \alpha\beta B\delta\omega \Downarrow \alpha\beta\beta' C\delta'\delta\omega \Downarrow \alpha\beta\beta'\gamma\delta'\delta\omega \Downarrow_* \underbrace{a_1 \ldots a_n}_{w}$$

$$A \rightarrow \beta B\delta \in P \qquad B \rightarrow \beta' C\delta' \in P \qquad C \rightarrow \gamma \in P$$

3-step

Generic Cover Algorithm

```
cover(Crit,A,W):-
  symbol(A),                  % iterate over symbols
  derive(s,α,A,ω),            % find (minimal) embedding
  call(Crit,A,β),             % expand via Crit, can iterate
  append([α,β,ω],γ),
  yield(γ,W).                 % find (minimal) yield

% coverage criteria for positive test suites
sym(A,[A]).
rule(A,γ):- prod(A,γ).
cdrc(A,γ):- prod(A,α),
            append([γ,[B],δ],α), prod(B,β),
            append([γ,β,δ],γ).
```

Generic Cover Algorithm

Algorithm 1: Generic cover algorithm

input : A CFG $G = (N, T, P, S)$
input : A coverage criterion C
input : A minimal derivation relation $\Rightarrow^*_\leq$
output: A test suite TS over G

$TS \leftarrow \varnothing$
for $X \in V$ **do**
  compute $S \Rightarrow^*_\leq \alpha X \omega$
  for $\theta \in C(X)$ **do**
    compute $\alpha\theta\omega \Rightarrow^*_\leq w$
    TS.add(w)
  end
end
return TS
// coverage criteria
rule$(X) \mathrel{\hat{=}} \{\alpha \mid X \to \alpha \in P\}$
cdrc$(X) \mathrel{\hat{=}} \{\alpha\gamma\omega \mid X \to \alpha Y \omega \in P, Y \to \gamma \in P\}$
step$_k(X) \mathrel{\hat{=}} \{\alpha Y \omega \mid X \Rightarrow^k_\leq \alpha Y \omega, Y \in V\}$
bfs$_k(X) \mathrel{\hat{=}} \{\alpha Y \omega \mid X \Rrightarrow^k \alpha Y \omega, Y \in V\}$
deriv$(X) \mathrel{\hat{=}} \{\alpha Y \omega \mid X \Rightarrow^*_\leq \alpha Y \omega, Y \in V\}$
pll$(X) \mathrel{\hat{=}} \{a\omega \mid X \Rightarrow^*_\leq a\omega, X \in N, a \in \text{first}(X)\}$

A Family of Grammar-Based Test Suite Adequacy Criteria…and some odd cousins

CDRC²: A rule $A \rightarrow \alpha$ is multiplied out if **all** non-terminals B_i on its right-hand side are simultaneously replaced by γ_i (for a rule $B_i \rightarrow \gamma_i \in P$). **Full context-dependent rule coverage** requires that rules are multiplied out using all rule combinations.

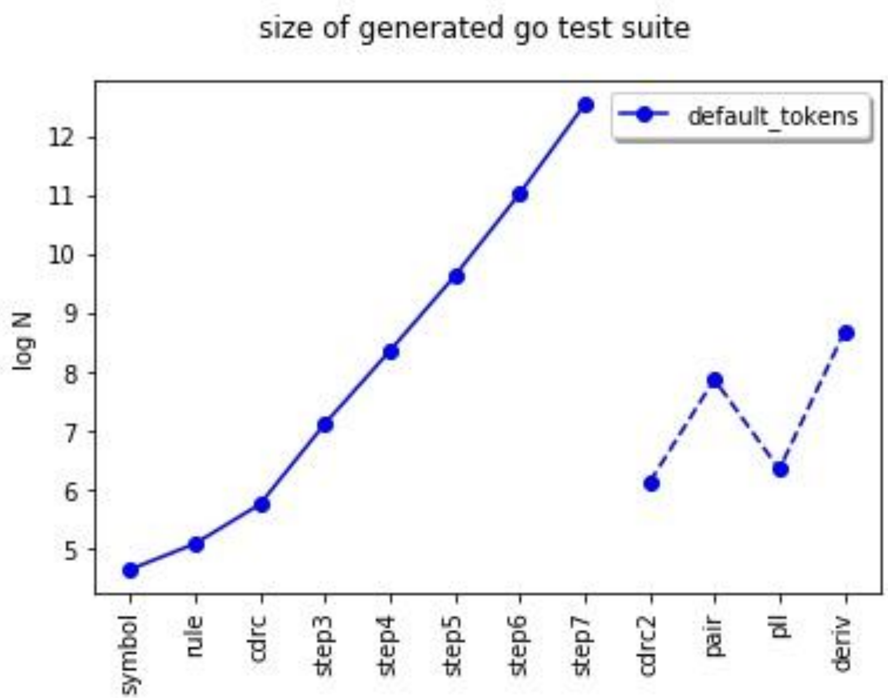
Deriv: A word w covers a derivable pair $(X,Y) \in V \times V$ iff $S \Rightarrow^* \alpha X \omega \Rightarrow^* \alpha \beta Y \psi \omega \Rightarrow^* w$. TS satisfies **derivable pair coverage** iff each pair (X,Y) with $X \prec Y$ is covered by a word $w \in TS$ and **PLL coverage** iff each pair (A,a) with $a \in \text{first}(A)$ is covered by a word $w \in TS$.

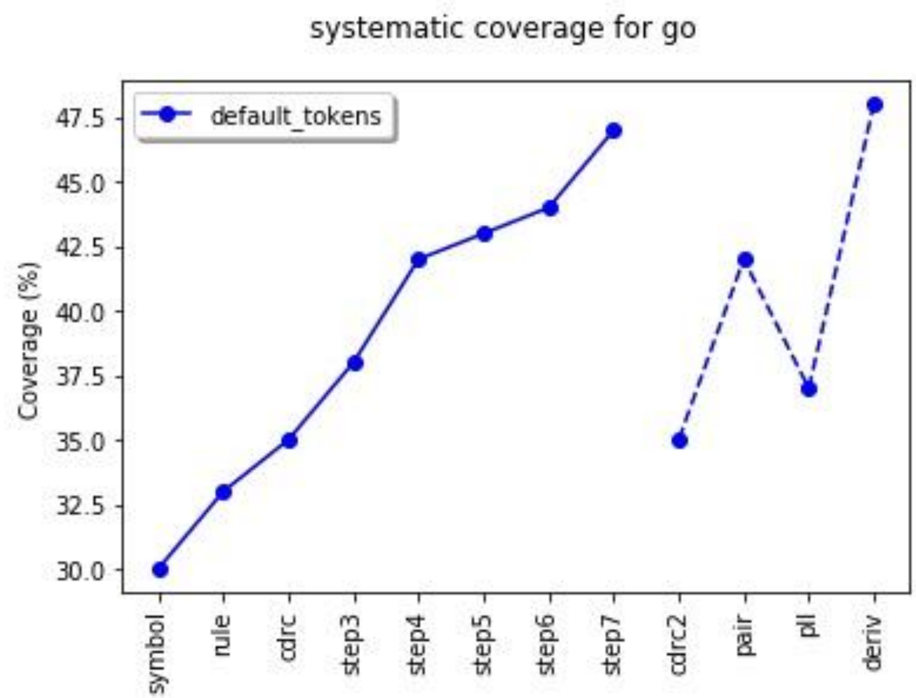
Pair: A word w covers an adjacent pair $(X,Y) \in V \times V$ iff $S \Rightarrow^* \alpha XY \omega \Rightarrow^* w$. TS satisfies **adjacent pair coverage** iff each pair (X,Y) with $Y \in \text{follow}(X)$ is covered by a word $w \in TS$.

Experimental Results - Coverage

Go (the programming language, that is):

- BNF: $|N| = 158$, $|T| = 83$, $|P| = 323$
- evaluated over gcc-go v8.2.0, $|go1_C| = 31034$





Experimental Results - Bug finding

Forced crash of gcc-go v8.2.0:

foo.go

```
package A; var A[A] A;
```

```
$ gccgo-8.2 -c foo.go
gccgo-8.2: internal compiler error: Segmentation
fault signal terminated program go1
Please submit a full bug report,
with preprocessed source if appropriate.
See <https://gcc.gnu.org/bugs/> for instructions.
```

I can recreate the compiler crash with GCC 8 branch, but it is fixed on trunk. On trunk I get

```
foo.go:3:7: error: array bound is not constant
    3 | var A[A] A
      |       ^
foo.go:3:7: error: expected type
```

Random Test Suite Generation

Basic algorithm:

start with the sentential form $\alpha = S$

repeatedly pick a random non-terminal symbol A such that $\alpha = \beta A \gamma$

expand A with a random rule $A \rightarrow \delta \in P$

continue until $\alpha = \beta\delta\gamma \in T^*$

Many variations:

- force termination
  - replace remaining non-terminals by fixed yield
- repeated depth-first
  - pick $A \in \delta$, if impossible randomly restart
- breadth-first
  - start with ß = ϵ, pick $A \in \gamma$, if impossible restart

Experimental Results - Bug finding

Forced crash of gcc-go v8.2.0:

foo.go

```
package A; func(*A) A(); type A(*A); type(A A; A A;);
```

Fixed on trunk by revision 270658.

```
$ gccgo-8.2 -c foo.go
go1: internal compiler error: in func_value, at
go/gofrontend/gogo.h:2583
0x9d0bfb Named_object::func_value()
        ../../gcc-8.2.0/gcc/go/gofrontend/gogo.h:2583
0xb1a03d Type_declaration::define_methods(Named_type*)
        ../../gcc-8.2.0/gcc/go/gofrontend/gogo.cc:7099
[...]
0xad4a71 go_langhook_parse_file
        ../../gcc-8.2.0/gcc/go/go-lang.c:329
Please submit a full bug report, with preprocessed
source if appropriate.
Please include the complete backtrace with any
bug report.
```

Systematic construction of negative test suites

what if $L(G) \subseteq L(U)$?

Grammar-Based Testing

grammar *G*

$prog \rightarrow$ **module** $prio$ id = $block$.
$prio \rightarrow$ [num]
$block \rightarrow$ **begin** $(decl\ ;)^*$ $(stmt\ ;)^*$ **end**
$decl \rightarrow$ **var** id : $type$
$type \rightarrow$ **bool** | **int**
$stmt \rightarrow$ **if** $expr$ **then** $stmt$ (**else** $stmt$)? | **while** $expr$ **do** $stmt$ | id = $expr$ | $block$
$expr \rightarrow expr = expr \mid expr + expr \mid$ ($expr$) | id | num

sentence generation →

test suite $TS \subseteq L(G)$

```
module[1] x = begin begin end; end.
module[2] y = begin end.
module[3] z = begin x = (y); end.
module[1] z = begin x = x + y; end.
module[2] x = begin y = z; end.
module[3] z = begin x = z = y; end.
module[1] y = begin y = 1; end.
module[2] y = begin if x then begin end; end.
module[3] y = begin var x : bool; end.
module[2] z = begin var z : int; end.
module[1] x = begin while x do begin end; end.
...
```

Test suites with only positive test cases fail to find many errors:

- gratuitous optionals

  $prog \rightarrow$ **module** $prio?$ id = $block$. $decl \rightarrow$ **var** id (, id$)^*$: $type$

- superfluous alternatives

  $type \rightarrow$ **bool** | **int** | **long**

- unwarranted over-generalization

  $prio \rightarrow$ ([$epxr$])

- order violations

  $block \rightarrow$ **begin** $((decl\ ;) \mid (stmt\ ;))^*$ **end**

Mutation-Based Language Fuzzing

Key observation #1

If $w = uabv$ and $b \notin \text{follow}(a)$, then $w \notin L(G)$.

Key observation #2

We can use this to identify locations for string editing operations (insert, delete, substitute, transpose) that fuzz an existing positive test suite into a negative test suite.

Key observation #3

We can lift these ideas from tokens and words to symbols and rules.

Basic Notations

Poisoned pair (i.e., symbols that cannot be next to each other)

- $(X, Y) \in PP(G)$ iff $X \notin \text{precede}(Y)$ or $Y \notin \text{follow}(X)$

Left / right sets (i.e., terminals that can occur left / right to the designated position in an item $A \rightarrow \alpha\bullet\beta$ for $A \rightarrow \alpha\beta \in P$)

- $\text{left}(A \longrightarrow \alpha\bullet\beta) = \begin{cases} (\text{last}(\alpha) \cup \text{precede}(A)) \cap T & \textit{if } \alpha \textit{ nullable} \\ (\text{last}(\alpha) \cap T & \textit{otherwise} \end{cases}$

- $\text{right}(A \longrightarrow \alpha\bullet\beta) = \begin{cases} (\text{first}(\beta) \cup \text{follow}(A)) \cap T & \textit{if } \beta \textit{ nullable} \\ (\text{first}(\beta)) \cap T & \textit{otherwise} \end{cases}$

Word Mutation Operators

Token deletion:

- $uabcv \in L(G)$, $(a,c) \in PP(G) \Rightarrow uacv \notin L(G)$

Token insertion:

- $uacv \in L(G)$, $(a,d) \in PP(G)$ or $(d,c) \in PP(G) \Rightarrow uadcv \notin L(G)$

Token substitution:

- $uabcv \in L(G)$, $(a,d) \in PP(G)$ or $(d,c) \in PP(G) \Rightarrow uadcv \notin L(G)$

Token transposition:

- $uabcdv \in L(G)$, $(a,c) \in PP(G)$ or $(c,b) \in PP(G)$ or $(b,d) \in PP(G)$ $\Rightarrow uacbdv \notin L(G)$

Note: higher-order mutations are not guaranteed to produce negative test cases.

Word Mutation Algorithm

```
...
module[2] x = begin y = z; end.
...
```

```
...
      [2] x = begin y = z; end.
module 2] x = begin y = z; end.
module[ ] x = begin y = z; end.
module[2  x = begin y = z; end.
module[2]   = begin y = z; end.

          x = begin y module z; end.
          x = begin y [ z; end.
          x = begin y ] z; end.
          x = begin y begin z; end.
          x = begin y end z; end.
          x = begin y var z; end.
          x = begin y : z; end.
          x = begin y bool z; end.
     e[2] x = begin y int z; end.
modu      x = begin y if z; end.
module[2] x = begin y while z; end.
module[2] x = begin y ( z; end.
module[2] x = begin y ) z; end.
            = begin y x z; end.
            = begin y 0 z; end.

      [2] x = begin y module = z; end.
...
module[2] x = begin y 0 = z; end.
module[2] x = begin y then = z; end.
module[2] x = begin y else = z; end.
module[2] x = begin y do = z; end.
module[2] x = begin y + = z; end.
module[2] x = begin y = = z; end.
...
```

foreach $w \in TS$:

  foreach i in $|w|$:

    foreach operator m:

      if $\text{pre}_m(w,i)$

      then print $m(w,i)$

no replacement by **then**, **else**, **do**, **+** and **=** since they do not produce a PP in remaining context …

… but do in context with **=**

Rule Mutation Operators

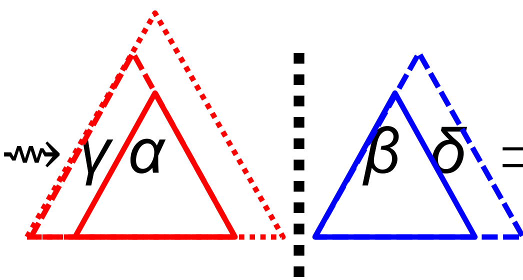
Symbol deletion: Let $p = A \rightarrow \alpha \bullet X\beta \in P^{\bullet}$. If

- $\text{follow}(\text{left}(A \rightarrow \alpha \bullet \beta)) \cap \text{right}(A \rightarrow \alpha \bullet \beta) = \emptyset$, or
- $\text{left}(A \rightarrow \alpha \bullet \beta) \cap \text{precede}(\text{right}(A \rightarrow \alpha \bullet \beta)) = \emptyset$

then any $w \notin L(G)$ if $S \Rightarrow^{*} \gamma A \delta \rightsquigarrow \gamma \alpha \beta \delta \Rightarrow^{*} w$

Intuition:

$$S \Rightarrow^{*} \gamma A \delta \rightsquigarrow \gamma \; \alpha \; \beta \; \delta \Rightarrow^{*} w$$



Symbol insertion: Let $p = A \rightarrow \alpha \bullet \beta \in P^{\bullet}$, $X \in V$. If

- $\text{follow}(\text{left}(A \rightarrow \alpha \bullet X\beta)) \cap \text{right}(A \rightarrow \alpha \bullet X\beta) = \emptyset$, or
- $\text{left}(A \rightarrow \alpha \bullet X\beta) \cap \text{precede}(\text{right}(A \rightarrow \alpha \bullet X\beta)) = \emptyset$

then any $w \notin L(G)$ if $S \Rightarrow^{*} \gamma A \delta \rightsquigarrow \gamma \alpha X \beta \delta \Rightarrow^{*} w$

Experimental Results

- Simpl - small imperative language (like Ampl)
- student grammars, yacc encoding from given EBNF
- differential testing
  - test cases generated from grammar, using cover algorithm
  - tested on golden parser

														False negatives		False positives		
Grammar	$\|N\|$	$\|T\|$	$\|P\|$		*DL(cdrc)*		*DL(pll)*		total_{DL}	*rule-mut*		total	overlap	*cdrc*	*pll*	*DL(cdrc)*	*DL(pll)*	*rule-mut*
11	46	47	88	0.6	139331 (166)	1.0	36135 (50)	0.3	143049	8984	3.3	144959	78.7%	0	0	0	0	7
13	42	45	80	0.6	138102 (171)	1.0	32505 (46)	0.5	141645	8376	2.6	143625	76.4%	17	1	0	0	6
15	64	47	107	0.9	182205 (223)	1.5	37097 (51)	0.3	185946	8923	5.0	187809	79.1%	47	3	0	0	5
17	47	47	90	0.7	145761 (174)	1.4	36609 (51)	0.2	149479	9004	3.3	151380	78.9%	15	3	0	0	6
19	46	47	88	0.6	116062 (139)	0.9	36135 (50)	0.3	119780	8984	3.1	121694	78.7%	0	0	0	0	7
21	68	47	110	0.8	139331 (166)	1.7	36135 (50)	0.2	143049	8984	5.1	144959	78.7%	0	0	0	0	7
23	73	47	115	0.8	129130 (152)	1.1	36135 (50)	0.3	132849	8984	5.9	134759	78.7%	7	1	458	142	35
25	46	47	88	0.6	139331 (166)	1.0	36135 (50)	0.2	143049	8984	3.3	144959	78.7%	0	0	0	0	7
27	46	47	88	0.6	139331 (166)	1.0	36135 (50)	0.2	143049	8984	3.1	144959	78.7%	0	0	0	0	7
29	92	46	136	0.9	115566 (141)	0.8	35227 (50)	0.2	119188	9344	8.2	121490	75.4%	15	2	0	0	5
31	70	47	112	1.0	139331 (166)	1.3	36135 (50)	0.3	143049	8984	8.4	144959	78.7%	0	0	0	0	7
33	47	47	89	0.7	139331 (166)	1.7	36135 (50)	0.3	143049	8984	3.5	144959	78.7%	0	0	0	0	7

Conclusions

- Better grammar coverage gives better system coverage
  - token construction mechanism makes large difference
  - specialized criteria can outperform simple k-step for small k
- Random test suites outperform simple k-step for large k
- Negative test cases can be generated constructively
  - number of edit-based mutants grows very large
  - number of rule-based mutants remains reasonable
  - mutations allow precise oracles (location / error type)