Fault Localization and Repair for Grammarware

Moeketsi Raselimo

Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

Grammarware is software that processes structured data defined by an implicit or explicit (context-free) grammar.

Grammarware is software that processes structured data defined by an implicit or explicit (context-free) grammar.

(Internal and external) data formats

AVI MP4 MP3 HTML5

PDF FLV MKV 3GP MPEG

Grammarware is software that processes structured data defined by an implicit or explicit (context-free) grammar.

(Internal and external) data formats

Generic interchange formats

Grammarware is software that processes structured data defined by an implicit or explicit (context-free) grammar.

(Internal and external) data formats

AVI MP4 MP3 HTML5

PDF FLV MKV 3GP MPEG

Programming languages

JS

XML {JSON} YAML

Generic interchange formats

Grammarware is software that processes structured data defined by an implicit or explicit (context-free) grammar.

(Internal and external) data formats

AVI MP4 MP3 HTML5

PDF FLV MKV 3GP MPEG

Programming languages

XML {JSON} YAML

Generic interchange formats

CUP LALR Parser Generator for Java™ Scott Hudson, GVU Center, Georgia Tech

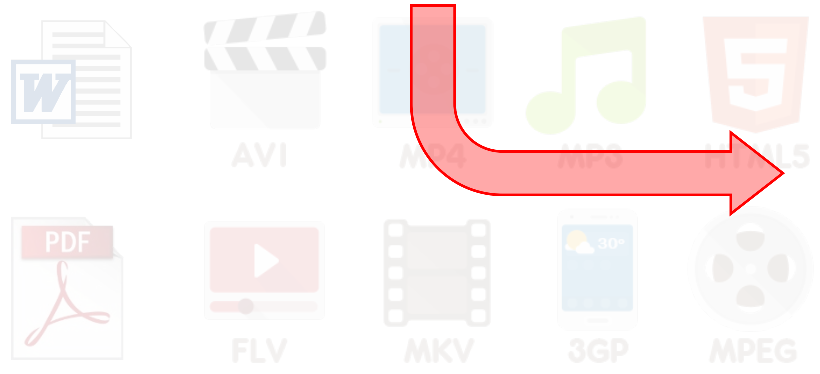
ANTLR

Language specifications

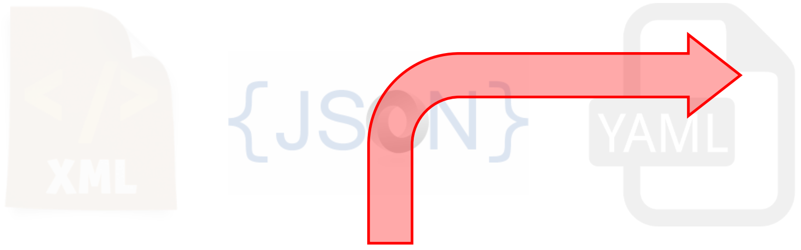
Grammarware is software that processes structured data defined by an implicit or explicit (context-free) grammar.

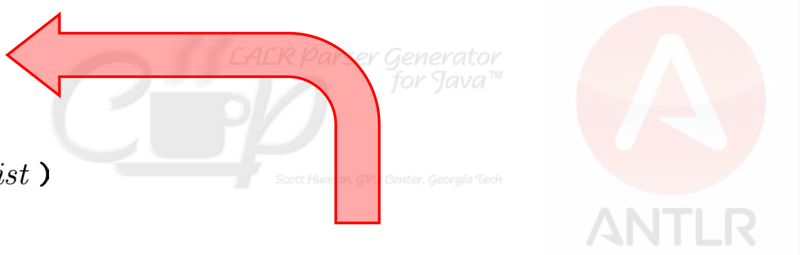
(Internal and external) data formats

Programming languages



```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( name namelist )
namelist  → namelist , name | ε
```





Generic interchange formats

Language specifications

Grammars are software. Software contains bugs.

Grammars are software. Software contains bugs.
But how do you *find* and *fix* grammar bugs?

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)

```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( name namelist )
namelist  → namelist , name | ε
```

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)

```
prog      → program id body
           | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
           | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( name namelist )
namelist  → namelist , name | ϵ
```

LALR Parser Generator for Java™

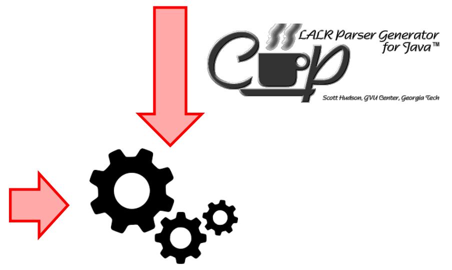
Scott Hudson, GVU Center, Georgia Tech

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)

```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( name  namelist )
namelist  → namelist , name | ϵ
```

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

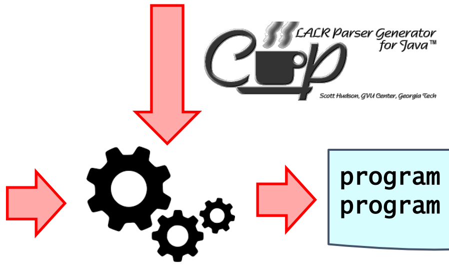

LALR Parser Generator for Java™
CUP
Scott Hudson, GVU Center, Georgia Tech

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)

```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( name namelist )
namelist  → namelist , name | ϵ
```

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```


LALR Parser Generator for Java™
CUP
Scott Hudson, GVU Center, Georgia Tech

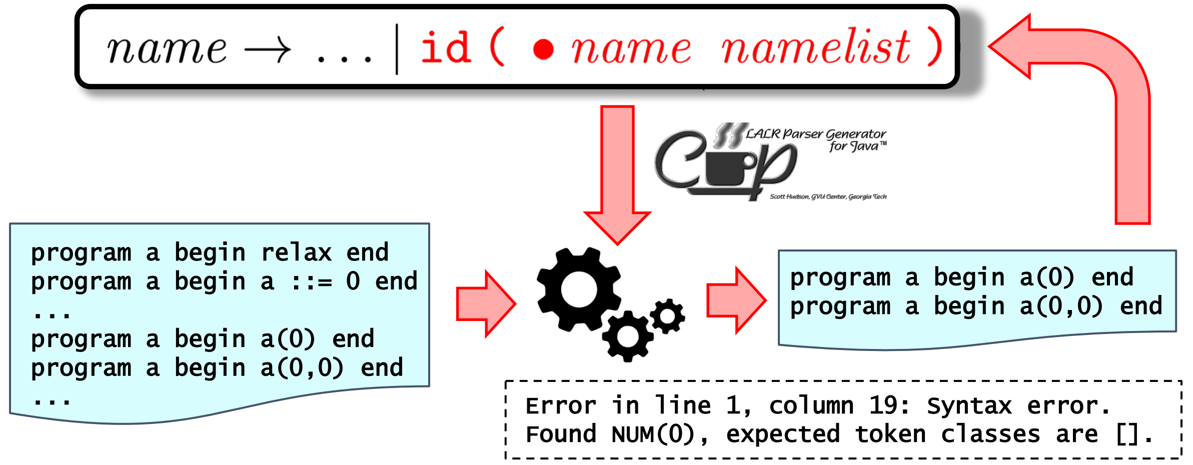
```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)

```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
```


LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

$name \rightarrow \ldots \mid$ id (• *name namelist*)

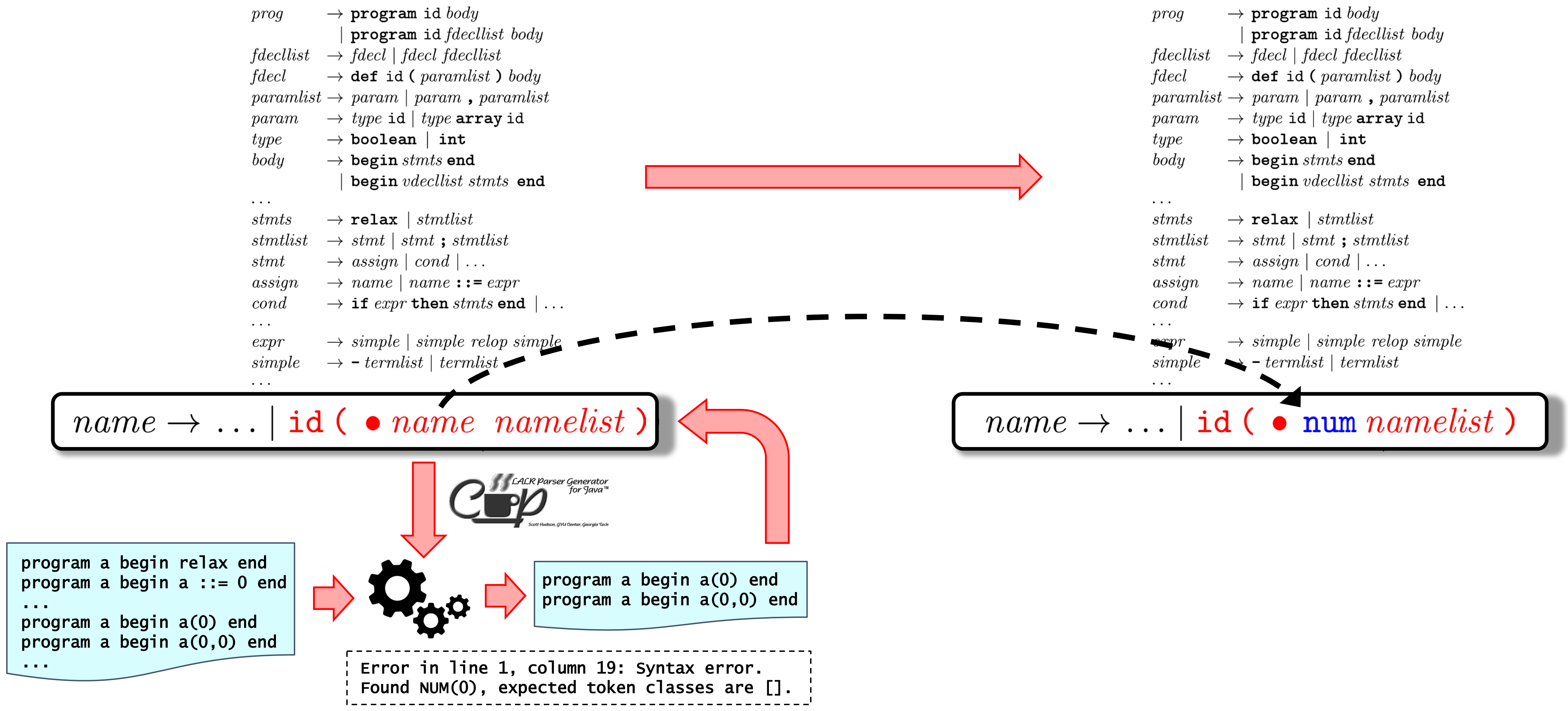
```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)



```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
```

$name \rightarrow \ldots \mid$ id (● *name namelist*)

$name \rightarrow \ldots \mid$ id (● num *namelist*)

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

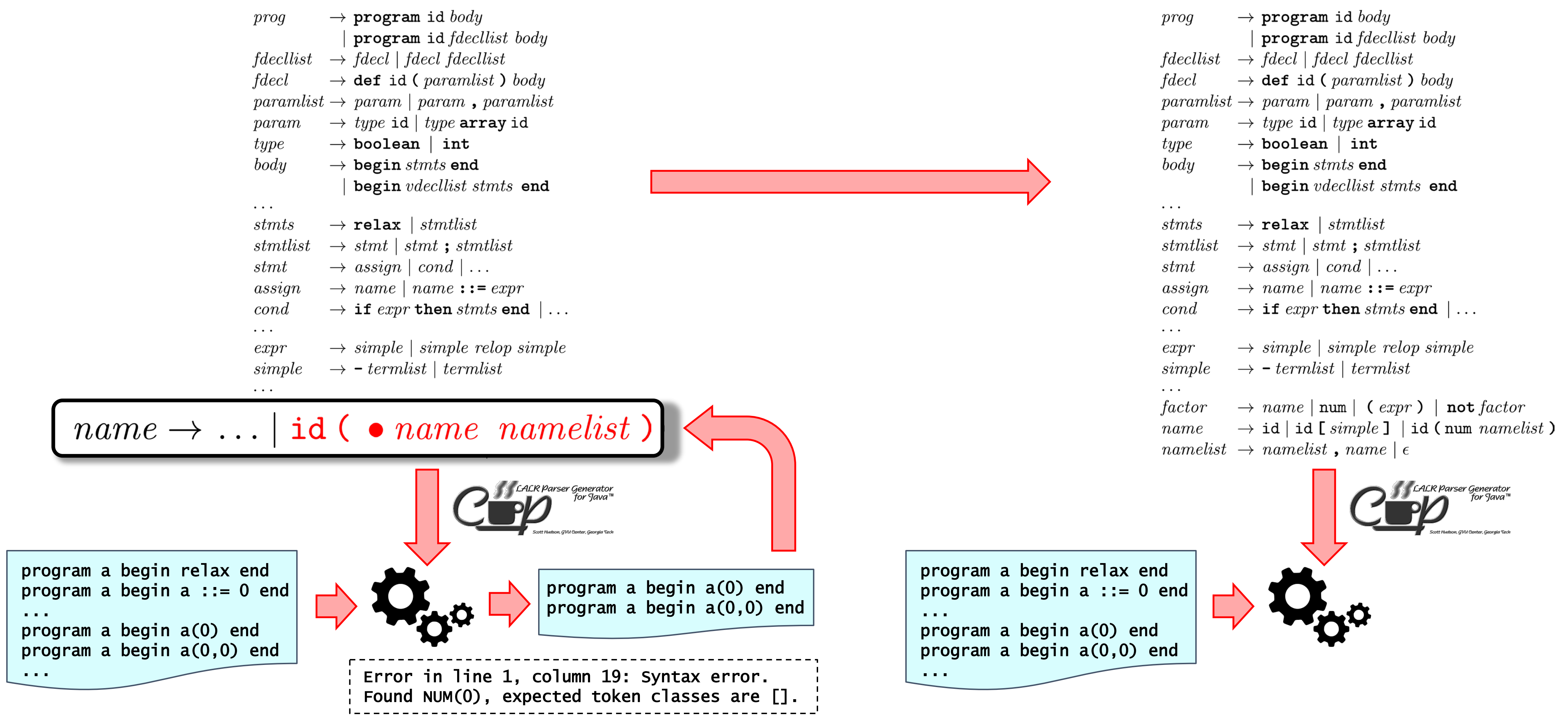
```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)



```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
```

$name \rightarrow \ldots \mid$ id (• *name* *namelist*)

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

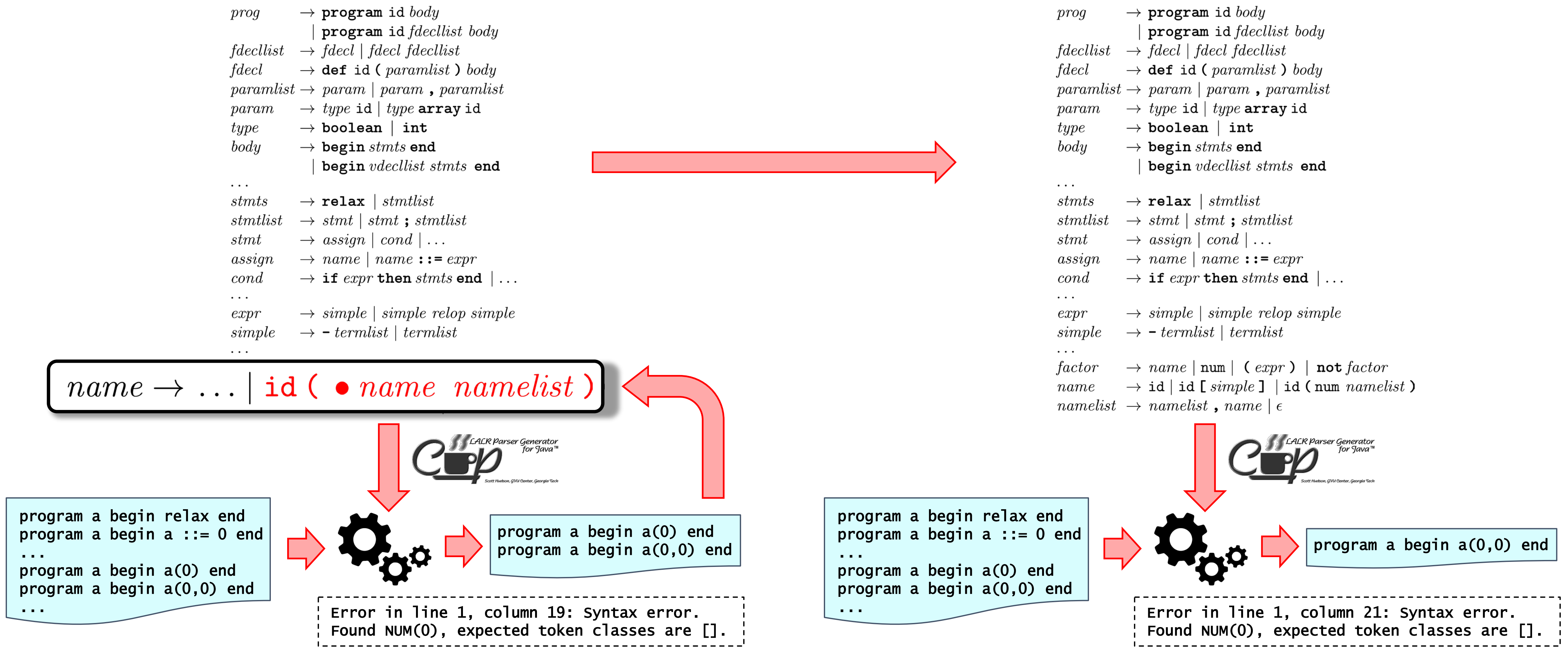
```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( num namelist )
namelist  → namelist , name | ϵ
```

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech
LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)



```
prog      → program id body
           | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
           | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
```

$name \rightarrow \ldots \mid$ id (• *name namelist*)

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

```
prog      → program id body
           | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
           | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( num namelist )
namelist  → namelist , name | ϵ
```

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

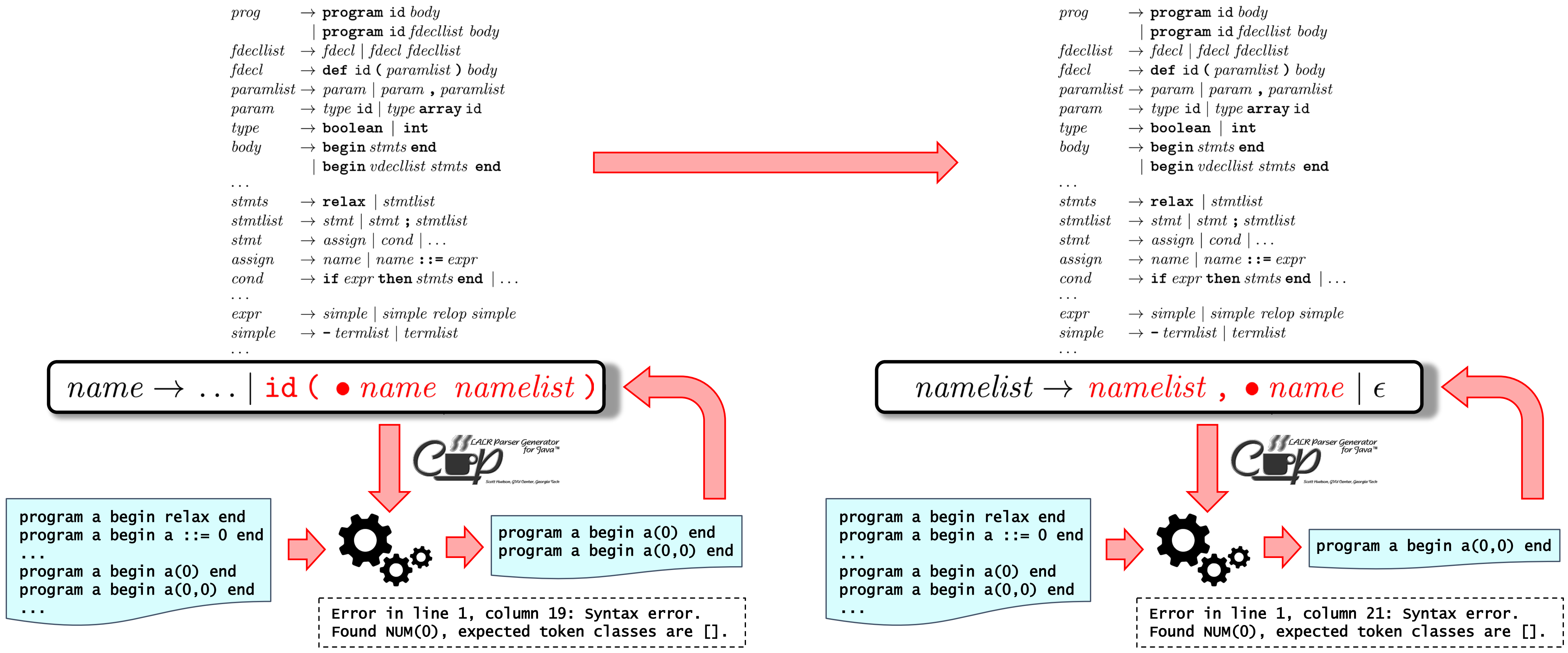
```
program a begin a(0,0) end
```

```
Error in line 1, column 21: Syntax error.
Found NUM(0), expected token classes are [].
```

LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)



```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
```

$name \rightarrow \ldots \mid$ **id (• *name namelist*)**

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
```

$namelist \rightarrow$ ***namelist* , • *name*** $\mid \epsilon$

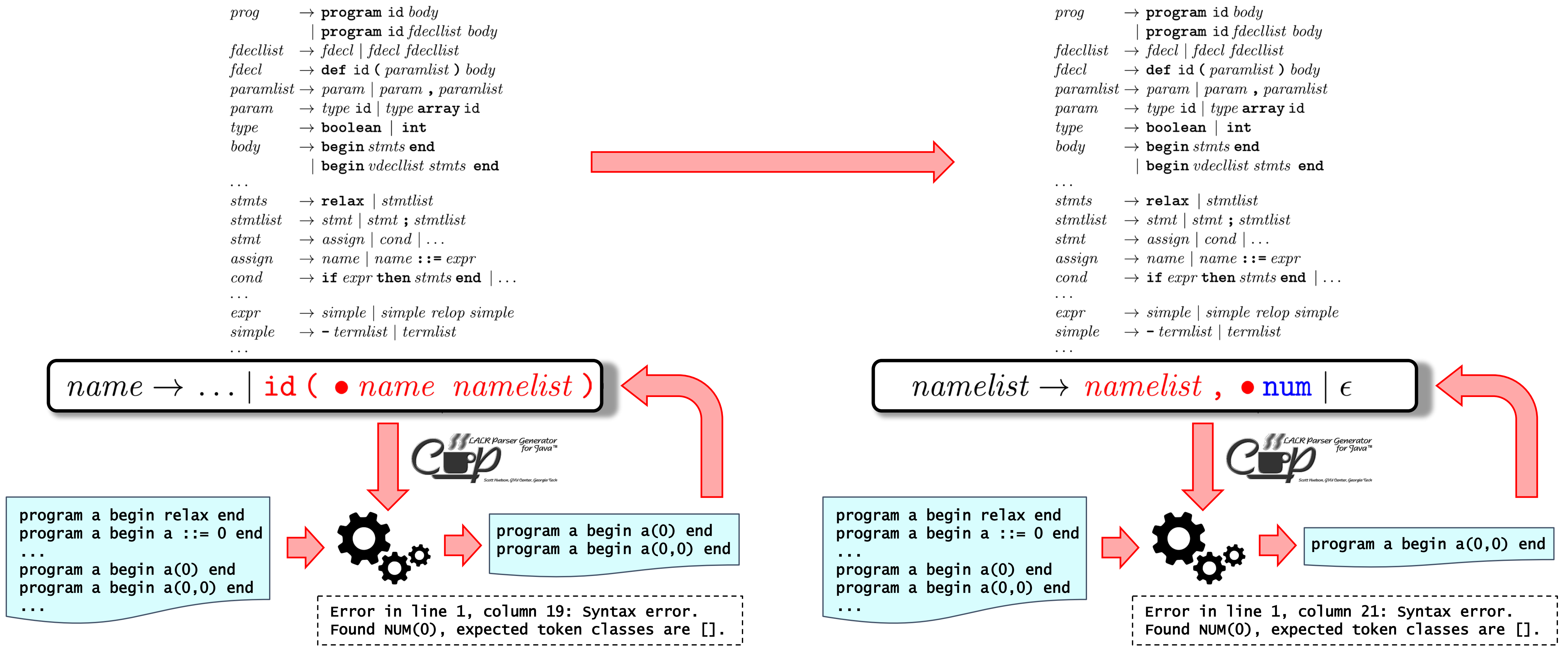
```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0,0) end
```

```
Error in line 1, column 21: Syntax error.
Found NUM(0), expected token classes are [].
```

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)



```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
```

$$name \to \dots \mid \texttt{id}\ (\ \bullet\ name\ \ namelist\ )$$

LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
```

$$namelist \to namelist\ ,\ \bullet\ \texttt{num} \mid \epsilon$$

LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0,0) end
```

```
Error in line 1, column 21: Syntax error.
Found NUM(0), expected token classes are [].
```

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)

```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
```

$name \rightarrow \ldots \mid$ id (● *name namelist*)

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

. . .

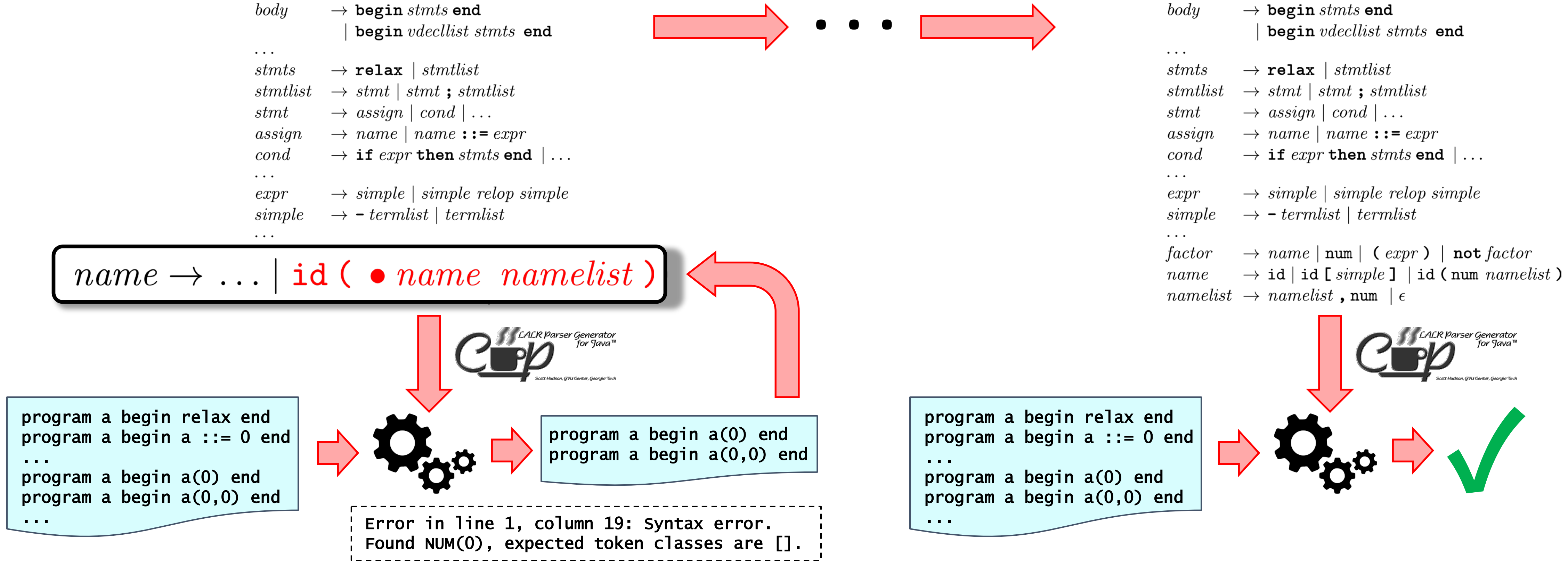
```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( num namelist )
namelist  → namelist , num | ϵ
```

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs?

Manual test-driven find-and-fix loop (***grammar debugging***)

```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
```

```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( num namelist )
namelist  → namelist , num | ε
```



. . .
name → . . . | id (● name namelist)
CUP LALR Parser Generator for Java
Scott Hudson, GVU Center, Georgia Tech
✓

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

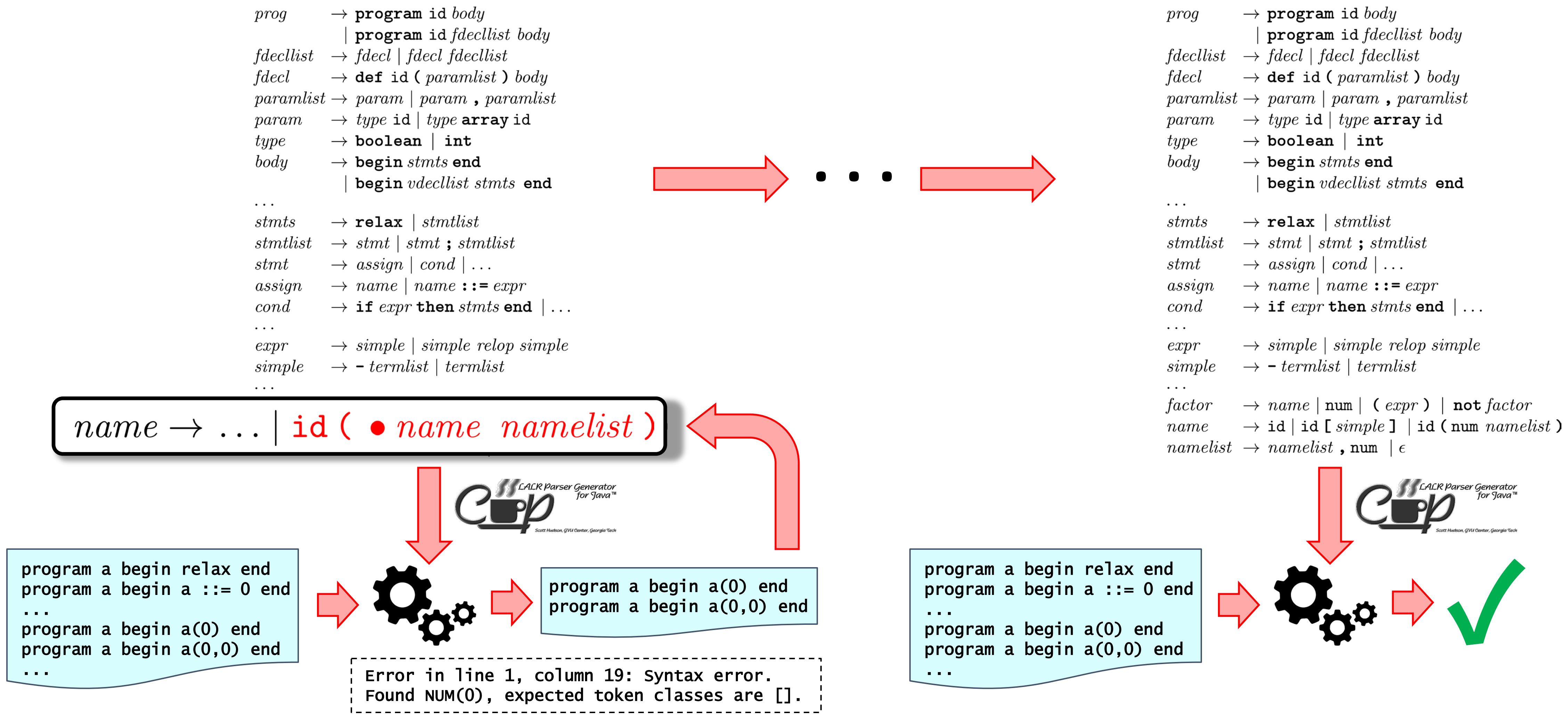
```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

Grammars are software. Software contains bugs. But how do you *find* and *fix* grammar bugs *automatically*?

Manual test-driven find-and-fix loop (***grammar debugging***)



```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
```

$name \rightarrow \ldots \mid$ id (● $name\ \ namelist$)

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

. . .

```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( num namelist )
namelist  → namelist , num | ε
```

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

CUP LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech
CUP LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech
✓

Fault Localization

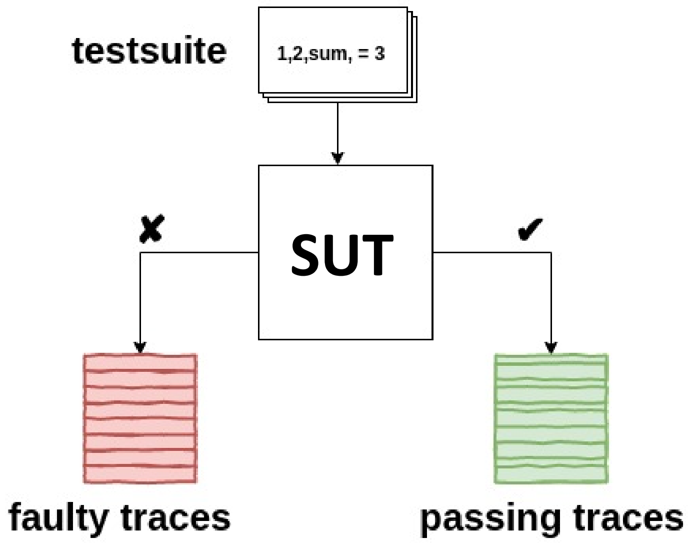
Spectrum-based fault localization (SBFL) tries to predict faulty statements from failing tests.

SBFL is a **heuristic**, **coverage-based**, **dynamic** method to identify faulty program elements (typically statements or methods):

Spectrum-based fault localization (SBFL) tries to predict faulty statements from failing tests.

SBFL is a **heuristic**, **coverage-based**, **dynamic** method to identify faulty program elements (typically statements or methods):

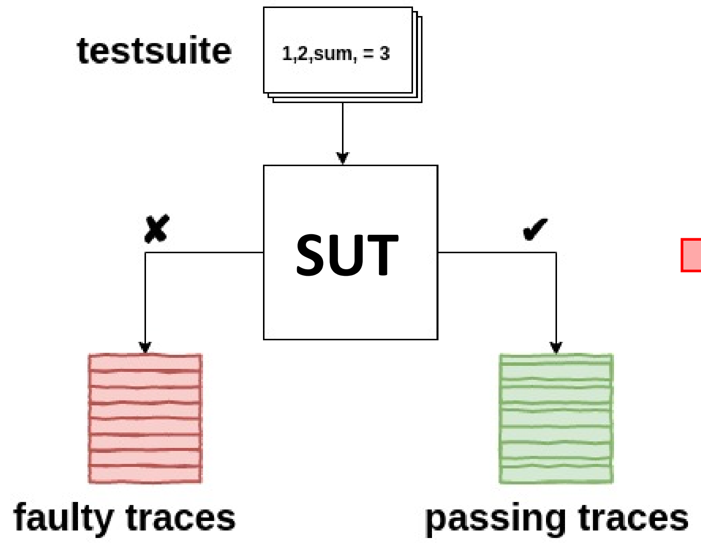
1. Execute system under test (SUT) over testsuite, collect ***coverage*** for each individual test case



Spectrum-based fault localization (SBFL) tries to predict faulty statements from failing tests.

SBFL is a **heuristic**, **coverage-based**, **dynamic** method to identify faulty program elements (typically statements or methods):

1. Execute system under test (SUT) over testsuite, collect ***coverage*** for each individual test case
2. Correlate coverage with outcomes, aggregate into ***spectrum***

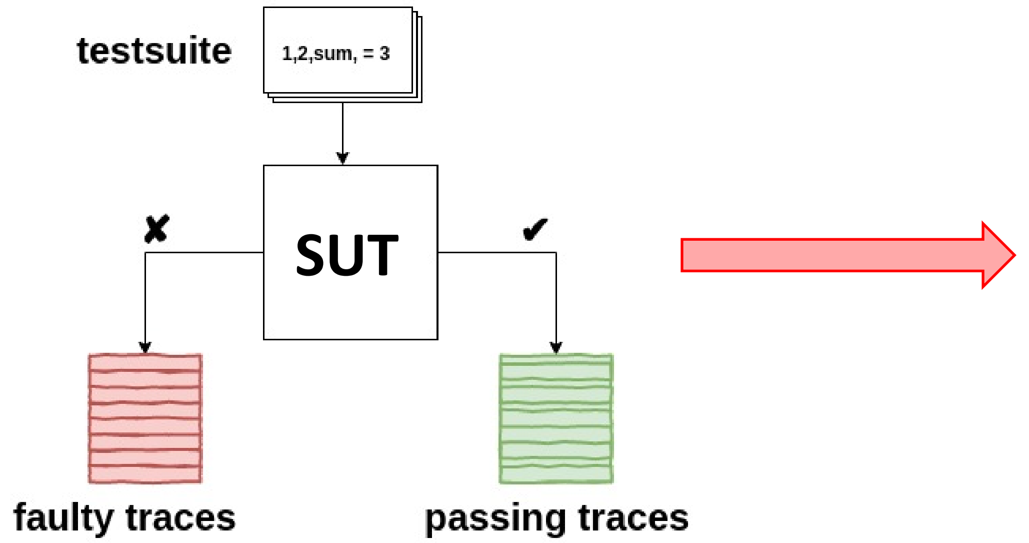


#	program	t1	t2	t3	t4	t5	t6
1	read(a);	✘	✘	✔	✔	✔	✔
2	read(b);	✘	✘	✔	✔	✔	✔
3	read(op);	✘	✘	✔	✔	✔	✔
4	if (op == "sum")	✘	✘	✔	✔	✔	✔
5	res = a - b; //fault	✘	✘	✔			
6	else if (op == "average")				✔	✔	✔
7	res = (a + b)/2;				✔	✔	✔
8	print(res);	✘	✘	✔	✔	✔	✔

Spectrum-based fault localization (SBFL) tries to predict faulty statements from failing tests.

SBFL is a **heuristic**, **coverage-based**, **dynamic** method to identify faulty program elements (typically statements or methods):

1. Execute system under test (SUT) over testsuite, collect ***coverage*** for each individual test case
2. Correlate coverage with outcomes, aggregate into ***spectrum***
3. Compute ***suspiciousness score*** for elements and ***rank***: higher scores and ranks indicate higher bug likelihood



#	program	t1	t2	t3	t4	t5	t6	sus	rank
1	read(a);	✘	✘	✔	✔	✔	✔	0.33	2
2	read(b);	✘	✘	✔	✔	✔	✔	0.33	2
3	read(op);	✘	✘	✔	✔	✔	✔	0.33	2
4	if (op == "sum")	✘	✘	✔	✔	✔	✔	0.33	2
5	res = a - b; //fault	✘	✘	✔				0.66	1
6	else if (op == "average")				✔	✔	✔	0	7
7	res = (a + b)/2;				✔	✔	✔	0	7
8	print(res);	✘	✘	✔	✔	✔	✔	0.33	2

How are scores computed?

How are scores computed?

1. Reduce spectra into four basic counts for each element *e*:

ep(e): # ***passed*** tests in which ***e*** is ***executed***
ef(e): # ***failed*** tests in which ***e*** is ***executed***
np(e): # ***passed*** tests in which ***e*** is ***not executed***
nf(e): # ***failed*** tests in which ***e*** is ***not executed***

How are scores computed?

1. Reduce spectra into four basic counts for each element *e*:

ep(e):	# ***passed*** tests in which **e** is ***executed***
ef(e):	# ***failed*** tests in which **e** is ***executed***
np(e):	# ***passed*** tests in which **e** is ***not executed***
nf(e):	# ***failed*** tests in which **e** is ***not executed***

2. Define ***ranking metric*** using basic counts

Tarantula

$$\frac{\frac{ef(e)}{ef(e)+nf(e)}}{\frac{ef(e)}{ef(e)+nf(e)}+\frac{ep(e)}{ep(e)+np(e)}}$$

Jaccard

$$\frac{ef(e)}{ef(e)+nf(e)+ep(e)}$$

Ochiai

$$\frac{ef(e)}{\sqrt{(ef(e)+nf(e))\cdot(ef(e)+ep(e))}}$$

D*

$$\frac{ef(e)^n}{nf(e)+ep(e)}$$

SBFL for Context-Free Grammars

Key insight: framework applies with minimal change

"executed" grammar elements instead of program statements
⇒ ***grammar spectra***

SBFL for Context-Free Grammars

Key insight: framework applies with minimal change

"executed" grammar elements instead of program statements
⇒ ***grammar spectra***

grammar rules and positions within rules

SBFL for Context-Free Grammars

Key insight: framework applies with minimal change

"executed" grammar elements instead of program statements
⇒ ***grammar spectra***

grammar rules and positions within rules

Advantage #1: low-cost approach

- reuse existing framework and tooling

Advantage #2: domain-specific approach

- higher precision: ignores parser boilerplate code
- higher precision: works for table-driven parser implementations
- higher utility: localization at grammar level
  ⇒ no tracking through code required

Rule-level Grammar Spectra

Given a ***grammar*** $G = (N,T,P,S)$ and a ***test suite*** $TS \subseteq T^*$, the **rule spectrum** for TS is the sets of rules $\boldsymbol{R} \subseteq \boldsymbol{P}$ (partially) applied when each $w \in TS$ is parsed.

Rule-level Grammar Spectra

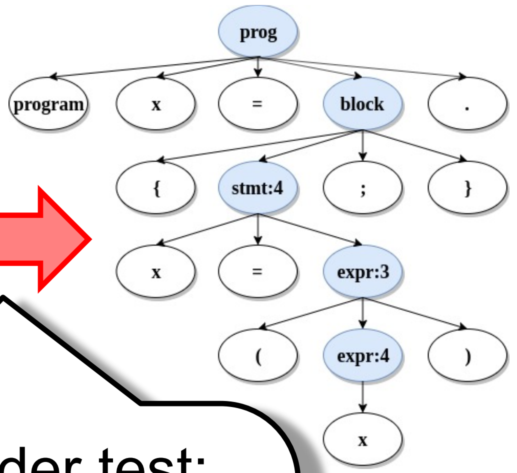
Given a ***grammar*** $G = (N,T,P,S)$ and a ***test suite*** $TS \subseteq T^*$, the **rule spectrum** for TS is the sets of rules $\boldsymbol{R} \subseteq \boldsymbol{P}$ (partially) applied when each $w \in TS$ is parsed.

```
...
program x={x=(x);}.
...
```

Rule-level Grammar Spectra

Given a ***grammar*** *G* = (*N*,*T*,*P*,*S*) and a ***test suite*** *TS* ⊆ *T**, the **rule spectrum** for *TS* is the sets of rules ***R* ⊆ *P*** (partially) applied when each *w* ∈ *TS* is parsed.

...
program x={x=(x);}.
...



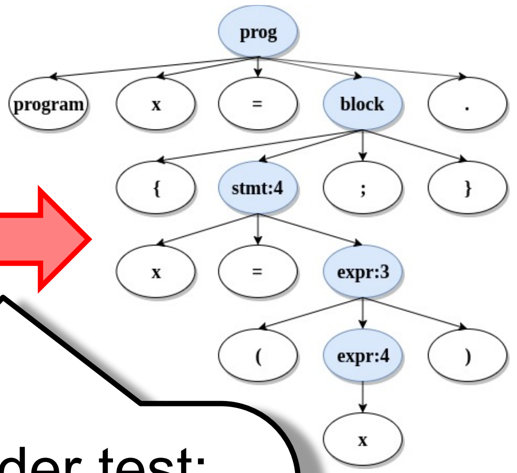
parse with grammar under test:

```
prog    → program id = block .
block   → { (decl ;)∗ (stmt ;)∗ }
decl    → var id : type
type    → bool | int
stmt    → sleep | if expr then stmt else stmt
          | while expr do block | id = expr | block
expr    → expr = expr | expr + expr | (expr ) | id | num
```

Rule-level Grammar Spectra

Given a ***grammar*** *G* = (*N*,*T*,*P*,*S*) and a ***test suite*** *TS* ⊆ *T**, the **rule spectrum** for *TS* is the sets of rules ***R*** ⊆ ***P*** (partially) applied when each *w* ∈ *TS* is parsed.

...
program x={x=(x);}.
...



prog
block
stmt:4
expr:3
expr:4

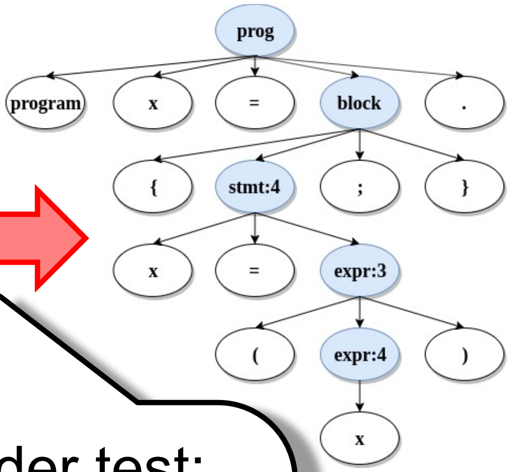
parse with grammar under test:

```
prog   → program id = block .
block  → { (decl ;)∗ (stmt ;)∗ }
decl   → var id : type
type   → bool | int
stmt   → sleep | if expr then stmt else stmt
         | while expr do block | id = expr | block
expr   → expr = expr | expr + expr | (expr ) | id | num
```

Item-level Grammar Spectra

Given a ***grammar*** $G = (N, T, P, S)$ and a ***test suite*** $TS \subseteq T^*$, the **item spectrum** for TS is the sets of positions within these rules $\boldsymbol{R^\bullet} \subseteq \boldsymbol{P^\bullet}$ processed successfully.

```
...
program x={x=(x);}.
...
```



parse with grammar under test:

prog → **program** id **=** block **.**
block → **{** (decl **;**)* (stmt **;**)* **}**
decl → **var id :** type
type → **bool** | **int**
stmt → **sleep** | **if** expr **then** stmt **else** stmt
| **while** expr **do** block | **id =** expr | block
expr → expr **=** expr | expr **+** expr | **(**expr **)** | **id** | **num**

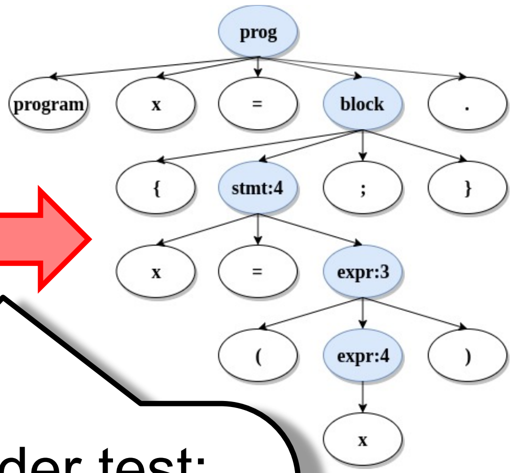
Item-level Grammar Spectra

Given a ***grammar*** $G = (N,T,P,S)$ and a ***test suite*** $TS \subseteq T^*$, the **item spectrum** for TS is the sets of positions within these rules $\boldsymbol{R^\bullet} \subseteq \boldsymbol{P^\bullet}$ processed successfully.

...
program x={x=(x);}.
...

parse with grammar under test:

prog → **program** id **=** block **.**
block → **{** (decl **;**)* (stmt **;**)* **}**
decl → **var id :** type
type → **bool** | **int**
stmt → **sleep** | **if** expr **then** stmt **else** stmt
| **while** expr **do** block | **id =** expr | block
expr → expr **=** expr | expr **+** expr | **(**expr **)** | **id** | **num**



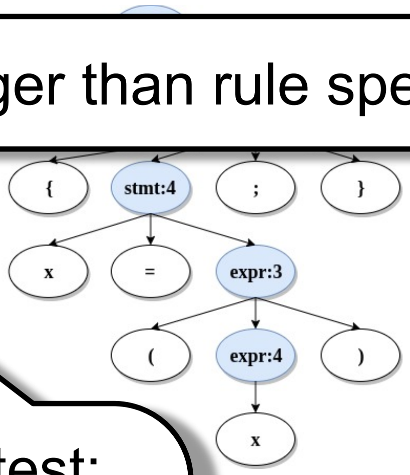
prog:1:1
…
prog:1:6
block:1:1
…
block:1:5
stmt:4:1
…
stmt:4:4
expr:3:1
…
expr:3:4
expr:4:1
expr:4:2

Item-level Grammar Spectra

Given a ***grammar*** $G = (N,T,P,S)$ and a ***test suite*** $TS \subseteq T^*$, the **item spectrum** for TS is the sets of positions within these rules $\boldsymbol{R^\bullet} \subseteq \boldsymbol{P^\bullet}$ processed successfully.

...
program x={x=(x);}.
...

larger than rule spectrum



prog:1:1
…
prog:1:6
block:1:1
…
block:1:5
stmt:4:1
…
stmt:4:4
expr:3:1
…
expr:3:4
expr:4:1
expr:4:2

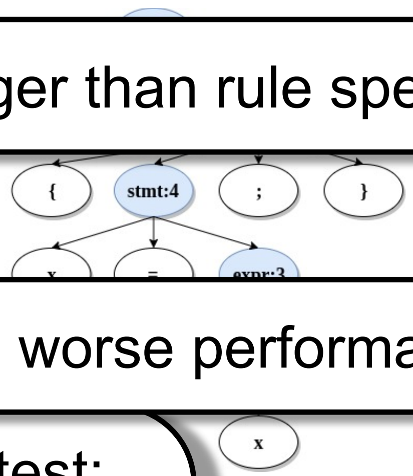
parse with grammar under test:

prog → **program** id **=** block **.**
block → **{** (decl **;**)∗ (stmt **;**)∗ **}**
decl → **var id :** type
type → **bool** | **int**
stmt → **sleep** | **if** expr **then** stmt **else** stmt
| **while** expr **do** block | **id =** expr | block
expr → expr **=** expr | expr **+** expr | **(**expr **)** | **id** | **num**

Item-level Grammar Spectra

Given a ***grammar*** $G = (N,T,P,S)$ and a ***test suite*** $TS \subseteq T^*$, the **item spectrum** for TS is the sets of positions within these rules $\boldsymbol{R^\bullet} \subseteq \boldsymbol{P^\bullet}$ processed successfully.

...
program x={x=(x);}.
...



larger than rule spectrum

≠ worse performance

parse with grammar under test:

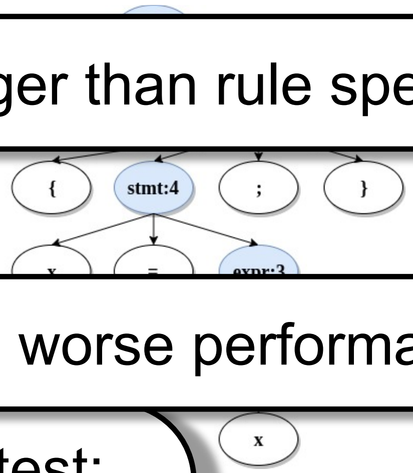
prog	→ **program** id **=** block **.**
block	→ **{** (decl **;**)* (stmt **;**)* **}**
decl	→ **var id :** type
type	→ **bool** | **int**
stmt	→ **sleep** | **if** expr **then** stmt **else** stmt
	| **while** expr **do** block | **id =** expr | block
expr	→ expr **=** expr | expr **+** expr | **(**expr **)** | **id** | **num**

prog:1:1
…
prog:1:6
block:1:1
…
block:1:5
stmt:4:1
…
stmt:4:4
expr:3:1
…
expr:3:4
expr:4:1
expr:4:2

Item-level Grammar Spectra

Given a ***grammar*** $G = (N,T,P,S)$ and a ***test suite*** $TS \subseteq T^*$, the **item spectrum** for TS is the sets of positions within these rules $\boldsymbol{R^\bullet \subseteq P^\bullet}$ processed successfully.

...
program x={x=(x);}.
...



larger than rule spectrum

≠ worse performance

more natural for repair

parse with grammar under test:

prog → **program** id **=** block **.**
block → **{** (decl **;**)∗ (stmt **;**)∗ **}**
decl → **var id :** type
type → **bool** | **int**
stmt → **sleep** | **if** expr **then** stmt **else** stmt
| **while** expr **do** block | **id =** expr | block
expr → expr **=** expr | expr **+** expr | **(**expr **)** | **id** | **num**

prog:1:1
…
prog:1:6
block:1:1
…
block:1:5
stmt:4:1
…
stmt:4:4
expr:3:1
…
expr:3:4
expr:4:1
expr:4:2

Grammar Spectra and Localization

```
program x={ x = (x); }.
program x={ x = x + x; }.
program x={ x = x; }.
program x={ x = x = x; }.
program x={ x = 0; }.
program x={ if x then sleep; }.
program x={ if x then sleep else sleep; }.
program x={ sleep; }.
program x={ var x : bool; }.
program x={ var x : int; }.
program x={ while x do sleep; }.
program x={ { }; }.
program x={ }.
```

parse with grammar under test:

prog	→ **program** id **=** block **.**
block	→ **{** (decl **;**)∗ (stmt **;**)∗ **}**
decl	→ **var id :** type
type	→ **bool** | **int**
stmt	→ **sleep** | **if** expr **then** stmt **else** stmt
	| **while** expr **do** block | **id =** expr | block
expr	→ expr **=** expr | expr **+** expr | **(**expr **)** | **id** | **num**

two faults

Grammar Spectra and Localization

program x={ x = (x); }.
program x={ x = x + x; }.
program x={ x = x; }.
program x={ x = x = x; }.
program x={ x = 0; }.
program x={ if x then sleep; }.
program x={ if x then sleep else sleep; }.
program x={ sleep; }.
program x={ var x : bool; }.
program x={ var x : int; }.
program x={ while x do sleep; }.
program x={ { }; }.
program x={ }.

parse with grammar under test:

prog → **program** id **=** block **.**
block → **{** (decl **;**)∗ (stmt **;**)∗ **}**
decl → **var id :** type
type → **bool** | **int**
stmt → **sleep** | **if** expr **then** stmt **else** stmt
| **while** expr **do** block | **id =** expr | block
expr → expr **=** expr | expr **+** expr | **(**expr **)** | **id** | **num**

two faults

rule	1	2	3	4	5	6	7	8	9	10	11	12	13
prog	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓
block	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓
decl									✓	✓			
type:1										✓			
type:2									✓				
stmt:1						✗	✓	✓					
stmt:2						✗	✓						
stmt:3											✗		
stmt:4	✓	✓	✓	✓	✓								
stmt:5												✓	
expr:1				✓									
expr:2		✓											
expr:3	✓												
expr:4	✓	✓	✓	✓		✗	✓				✗		
expr:5					✓								

Grammar Spectra and Localization

```
program x={ x = (x); }.
program x={ x = x + x; }.
program x={ x = x; }.
program x={ x = x = x; }.
program x={ x = 0; }.
program x={ if x then sleep; }.
program x={ if x then sleep else sleep; }.
program x={ sleep; }.
program x={ var x : bool; }.
program x={ var x : int; }.
program x={ while x do sleep; }.
program x={ { }; }.
program x={ }.
```

parse with grammar under test:

prog → **program** id **=** block **.**
block → **{** (decl **;**)∗ (stmt **;**)∗ **}**
decl → **var id :** type
type → **bool** | **int**
stmt → **sleep** | **if** expr **then** stmt **else** stmt
  | **while** expr **do** block | **id =** expr | block
expr → expr **=** expr | expr **+** expr | **(**expr **)** | **id** | **num**

two faults

rule	1	2	3	4	5	6	7	8	9	10	11	12	13	*ep*	*np*	*ef*	*nf*
prog	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	11	0	2	0
block	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	11	0	2	0
decl									✓	✓				2	9	0	2
type:1										✓				1	10	0	2
type:2									✓					1	10	0	2
stmt:1						✗	✓	✓						2	9	1	1
stmt:2						✗	✓							**1**	**10**	**1**	**1**
stmt:3											✗			**0**	**11**	**1**	**1**
stmt:4	✓	✓	✓	✓	✓									5	6	0	2
stmt:5												✓		1	10	0	2
expr:1				✓										1	10	0	2
expr:2		✓												1	10	0	2
expr:3	✓													1	10	0	2
expr:4	✓	✓	✓	✓		✗	✓				✗			5	6	2	0
expr:5					✓									1	10	0	2

Grammar Spectra and Localization

```
program x={ x = (x); }.
program x={ x = x + x; }.
program x={ x = x; }.
program x={ x = x = x; }.
program x={ x = 0; }.
program x={ if x then sleep; }.
program x={ if x then sleep else sleep; }.
program x={ sleep; }.
program x={ var x : bool; }.
program x={ var x : int; }.
program x={ while x do sleep; }.
program x={ { }; }.
program x={ }.
```

parse with grammar under test:

prog → **program** id **=** block **.**
block → **{** (decl **;**)$*$ (stmt **;**)$*$ **}**
decl → **var id :** type
type → **bool** | **int**
stmt → **sleep** | **if** expr **then** stmt **else stmt**
| **while** expr **do** block | **id =** expr | block
expr → expr **=** expr | expr **+** expr | **(**expr **)** | **id** | **num**

two faults

rule	1	2	3	4	5	6	7	8	9	10	11	12	13	*ep*	*np*	*ef*	*nf*	Tarantula		Ochiai		Jaccard		DStar	
prog	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	11	0	2	0	0.50	=5	0.39	=4	0.15	=5	0.36	=5
block	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	11	0	2	0	0.50	=5	0.39	=4	0.15	=5	0.36	=5
decl									✓	✓				2	9	0	2	0.00	-	0.00	-	0.00	-	0.00	-
type:1										✓				1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
type:2									✓					1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
stmt:1						✗	✓	✓						2	9	1	1	0.69	4	0.17	6	0.25	4	0.67	4
stmt:2						✗	✓							**1**	**10**	**1**	**1**	**0.85**	**2**	**0.50**	**3**	**0.33**	**2**	**2.00**	**2**
stmt:3											✗			**0**	**11**	**1**	**1**	**1.00**	**1**	**0.71**	**1**	**0.50**	**1**	**4.00**	**1**
stmt:4	✓	✓	✓	✓	✓									5	6	0	2	0.00	-	0.00	-	0.00	-	0.00	-
stmt:5												✓		1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:1				✓										1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:2		✓												1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:3	✓													1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:4	✓	✓	✓	✓		✗	✓				✗			5	6	2	0	0.79	3	0.53	2	0.29	3	0.80	3
expr:5					✓									1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-

Grammar Spectra and Localization

```
program x={ x = (x); }.
program x={ x = x + x; }.
program x={ x = x; }.
program x={ x = x = x; }.
program x={ x = 0; }.
program x={ if x then sleep; }.
program x={ if x then sleep else sleep; }.
program x={ sleep; }.
program x={ var x : bool; }.
program x={ var x : int; }.
program x={ while x do sleep; }.
program x={ { }; }.
program x={ }.
```

parse with grammar under test:

prog	→ **program** id **=** block **.**
block	→ **{** (decl **;**)∗ (stmt **;**)∗ **}**
decl	→ **var id :** type
type	→ **bool** | **int**
stmt	→ **sleep** | **if** expr **then** stmt **else** stmt
	| **while** expr **do** block | **id =** expr | block
expr	→ expr **=** expr | expr **+** expr | **(**expr **)** | **id** | **num**

two faults

rule	1	2	3	4	5	6	7	8	9	10	11	12	13	*ep*	*np*	*ef*	*nf*	Tarantula		Ochiai		Jaccard		DStar	
prog	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	11	0	2	0	0.50	=5	0.39	=4	0.15	=5	0.36	=5
block	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	11	0	2	0	0.50	=5	0.39	=4	0.15	=5	0.36	=5
decl									✓	✓				2	9	0	2	0.00	-	0.00	-	0.00	-	0.00	-
type:1										✓				1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
type:2									✓					1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
stmt:1						✗	✓	✓						2	9	1	1	0.69	4	0.17	6	0.25	4	0.67	4
stmt:2						✗	✓							**1**	**10**	**1**	**1**	**0.85**	**2**	**0.50**	**3**	**0.33**	**2**	**2.00**	**2**
stmt:3											✗			**0**	**11**	**1**	**1**	**1.00**	**1**	**0.71**	**1**	**0.50**	**1**	**4.00**	**1**
stmt:4	✓	✓	✓	✓	✓									5	6	0	2	0.00	-	0.00	-	0.00	-	0.00	-
stmt:5												✓		1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:1														1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:2		✓												1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:3	✓													1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:4	✓	✓	✓	✓		✗	✓				✗			5	6	2	0	0.79	3	0.53	2	0.29	3	0.80	3
expr:5					✓									1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-

while expr **do** block

Grammar Spectra and Localization

```
program x={ x = (x); }.
program x={ x = x + x; }.
program x={ x = x; }.
program x={ x = x = x; }.
program x={ x = 0; }.
program x={ if x then sleep; }.
program x={ if x then sleep else sleep; }.
program x={ sleep; }.
program x={ var x : bool; }.
program x={ var x : int; }.
program x={ while x do sleep; }.
program x={ { }; }.
program x={ }.
```

parse with grammar under test:

prog	→	**program** id **=** block **.**
block	→	**{** (decl **;**)* (stmt **;**)* **}**
decl	→	**var id :** type
type	→	**bool** | **int**
stmt	→	**sleep** | **if** expr **then** stmt **else stmt**
		| **while** expr **do** block | **id =** expr | block
expr	→	expr **=** expr | expr **+** expr | **(**expr **)** | **id** | **num**

two faults

rule	1	2	3	4	5	6	7	8	9	10	11	12	13	*ep*	*np*	*ef*	*nf*	Tarantula		Ochiai		Jaccard		DStar	
prog	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	11	0	2	0	0.50	=5	0.39	=4	0.15	=5	0.36	=5
block	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	11	0	2	0	0.50	=5	0.39	=4	0.15	=5	0.36	=5
decl									✓	✓				2	9	0	2	0.00	-	0.00	-	0.00	-	0.00	-
type:1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]			1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
type:2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]			1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
stmt:1						✗	✓	✓						2	9	1	1	0.69	4	0.17	6	0.25	4	0.67	4
stmt:2						✗	✓							**1**	**10**	**1**	**1**	**0.85**	**2**	**0.50**	**3**	**0.33**	**2**	**2.00**	**2**
stmt:3											✗			**0**	**11**	**1**	**1**	**1.00**	**1**	**0.71**	**1**	**0.50**	**1**	**4.00**	**1**
stmt:4	✓	✓	✓	✓	✓									5	6	0	2	0.00	-	0.00	-	0.00	-	0.00	-
stmt:5	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]				✓		1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]						1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:2		✓												1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:3	✓													1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:4	✓	✓	✓	✓		✗	✓				✗			5	6	2	0	0.79	3	0.53	2	0.29	3	0.80	3
expr:5					✓									1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-

if expr **then** stmt **else stmt**

while expr **do** block

Grammar Spectra and Localization

```
program x={ x = (x); }.
program x={ x = x + x; }.
program x={ x = x; }.
program x={ x = x = x; }.
program x={ x = 0; }.
program x={ if x then sleep; }.
program x={ if x then sleep else sleep; }.
program x={ sleep; }.
program x={ var x : bool; }.
program x={ var x : int; }.
program x={ while x do sleep; }.
program x={ { }; }.
program x={ }.
```

parse with grammar under test:

prog	→ **program** id **=** block **.**
block	→ **{** (decl **;**)* (stmt **;**)* **}**
decl	→ **var id :** type
type	→ **bool** | **int**
stmt	→ **sleep** | **if** expr **then** stmt **else** stmt
	| **while** expr **do** block | **id =** expr | block
expr	→ expr **=** expr | expr **+** expr | **(**expr **)** | **id** | **num**

two faults

rule	1	2	3	4	5	6	7	8	9	10	11	12	13	*ep*	*np*	*ef*	*nf*	Tarantula		Ochiai		Jaccard		DStar	
prog	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	11	0	2	0	0.50	=5	0.39	=4	0.15	=5	0.36	=5
block	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	11	0	2	0	0.50	=5	0.39	=4	0.15	=5	0.36	=5
decl									✓	✓				2	9	0	2	0.00	-	0.00	-	0.00	-	0.00	-
type:1														1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
type:2														1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
stmt:1						✗	✓	✓						2	9	1	1	0.69	4	0.17	6	0.25	4	0.67	4
stmt:2						✗	✓							**1**	**10**	**1**	**1**	**0.85**	**2**	**0.50**	**3**	**0.33**	**2**	**2.00**	**2**
stmt:3											✗			**0**	**11**	**1**	**1**	**1.00**	**1**	**0.71**	**1**	**0.50**	**1**	**4.00**	**1**
stmt:4	✓		✓	✓	✓									5	6	0	2	0.00	-	0.00	-	0.00	-	0.00	-
stmt:5												✓		1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:1														1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:2		✓												1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:3	✓													1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-
expr:4	✓	✓	✓	✓		✗	✓				✗			5	6	2	0	0.79	3	0.53	2	0.29	3	0.80	3
expr:5					✓									1	10	0	2	0.00	-	0.00	-	0.00	-	0.00	-

if expr **then** stmt **else stmt**

while expr **do** block

id

Fault Localization Evaluation

synthetic rule spectrum

Goals:

- evaluate effectiveness (spectra from LL and LR parsers; from test suites also)
- evaluate effects of different test suites
- compare both levels of fault localization

Fault seeding approach:

- full mutation of all rules in “golden” grammar (single mutants)
- keep only compiling grammars

Measurement:

- average predicted rank of mutated rule

Fault Localization Evaluation

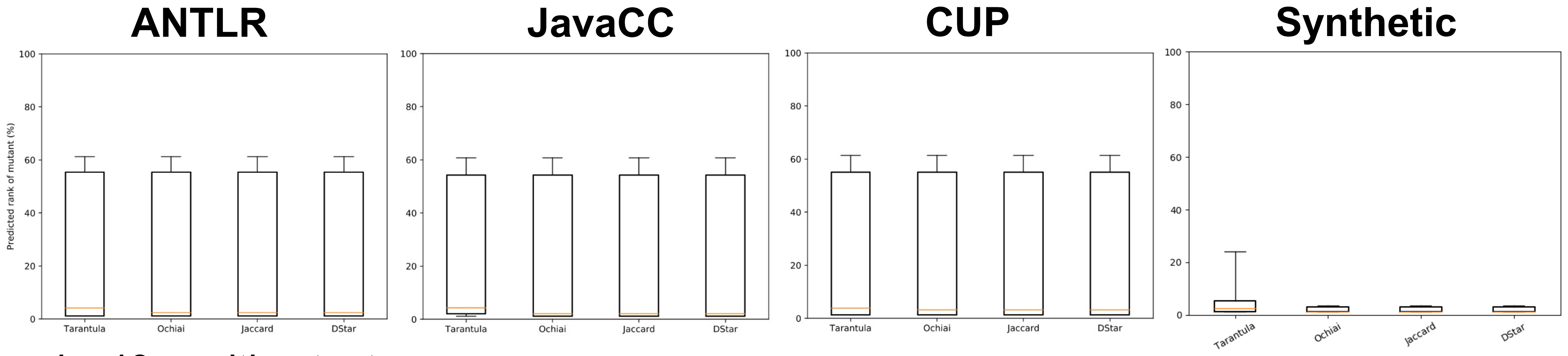
Subject:

- SIMPL grammar from 2nd-year computer architecture course

Test suites:

- generated from golden grammar
  - positive only: rule, cdrc
  - large: positive and mutation-based negative tests (word and rule mutation)

Rule Localization Results (I)

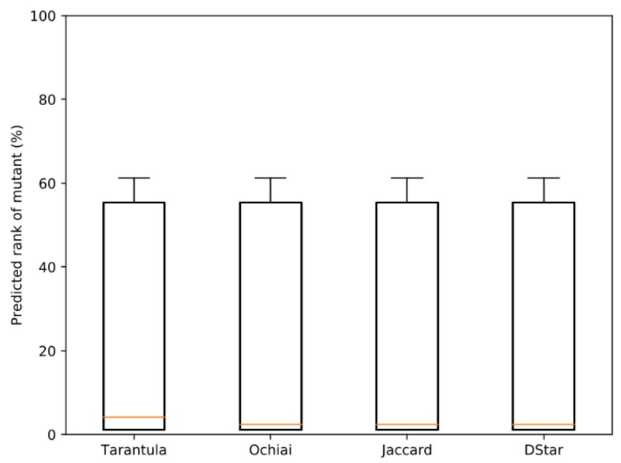


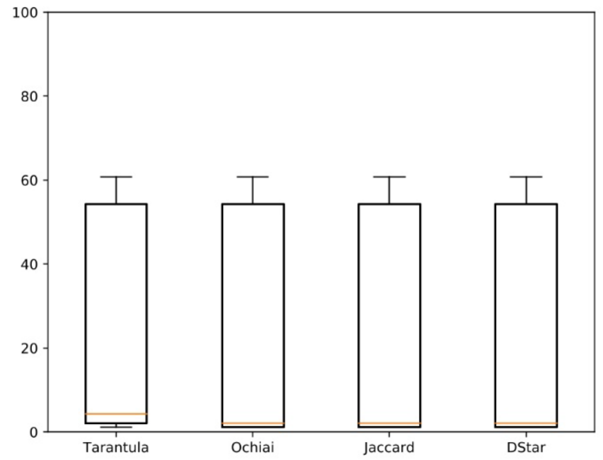
rule: 43 positive test cases

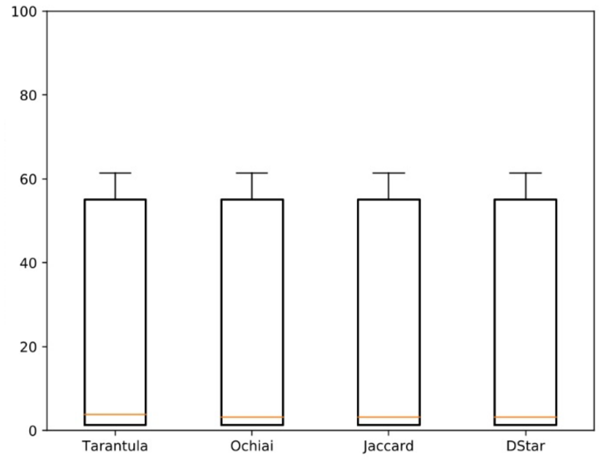
- *ANTLR:* ~80% killed, median rank 4 rules (~5%), ~25% rank #1, ~45% rank #5
- *JavaCC:* ~80% killed, median rank 4 rules (~4%), ~25% rank #1, ~50% rank #5
- *CUP:* ~90% killed, median rank 3 rules, ~40% rank #1
- *Synthetic:* ~85% rank #5
- Tarantula performs slightly worse, not much difference on others

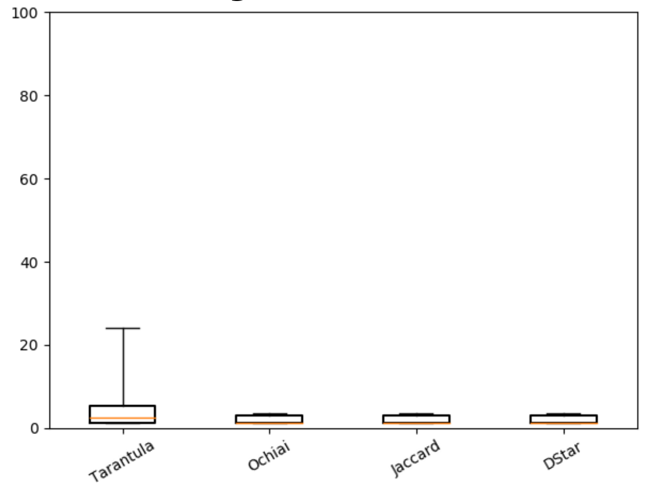
Rule Localization Results (I)

ANTLR **JavaCC** **CUP** **Synthetic**

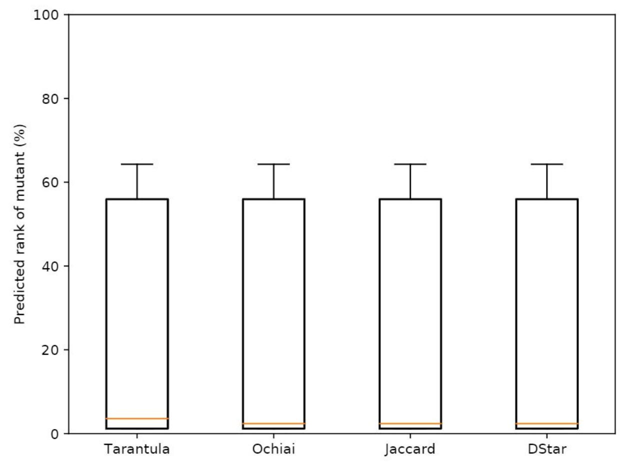


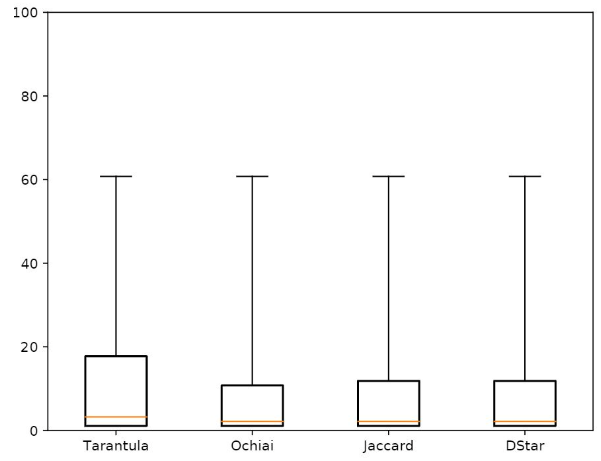


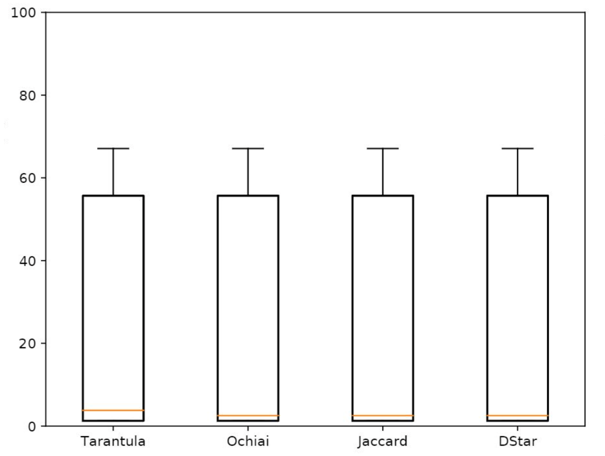


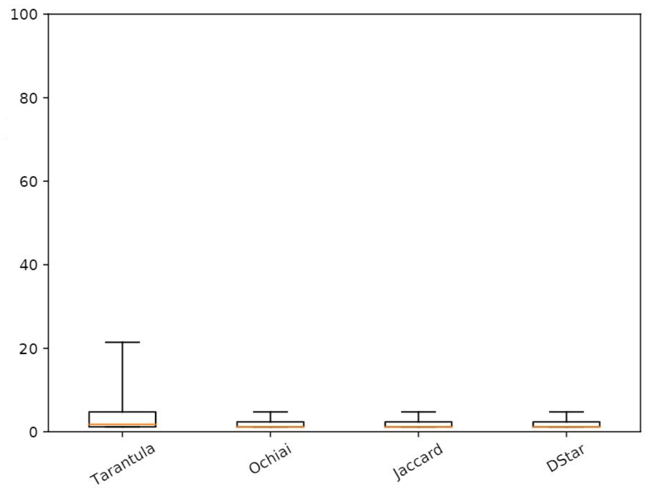


rule: 43 positive test cases



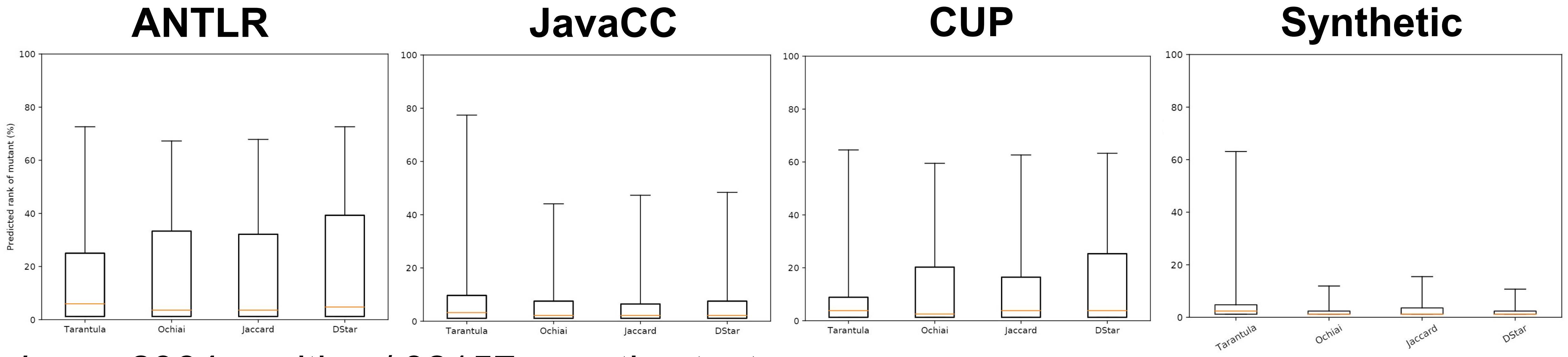






cdrc: 86 positive test cases (minor improvements)

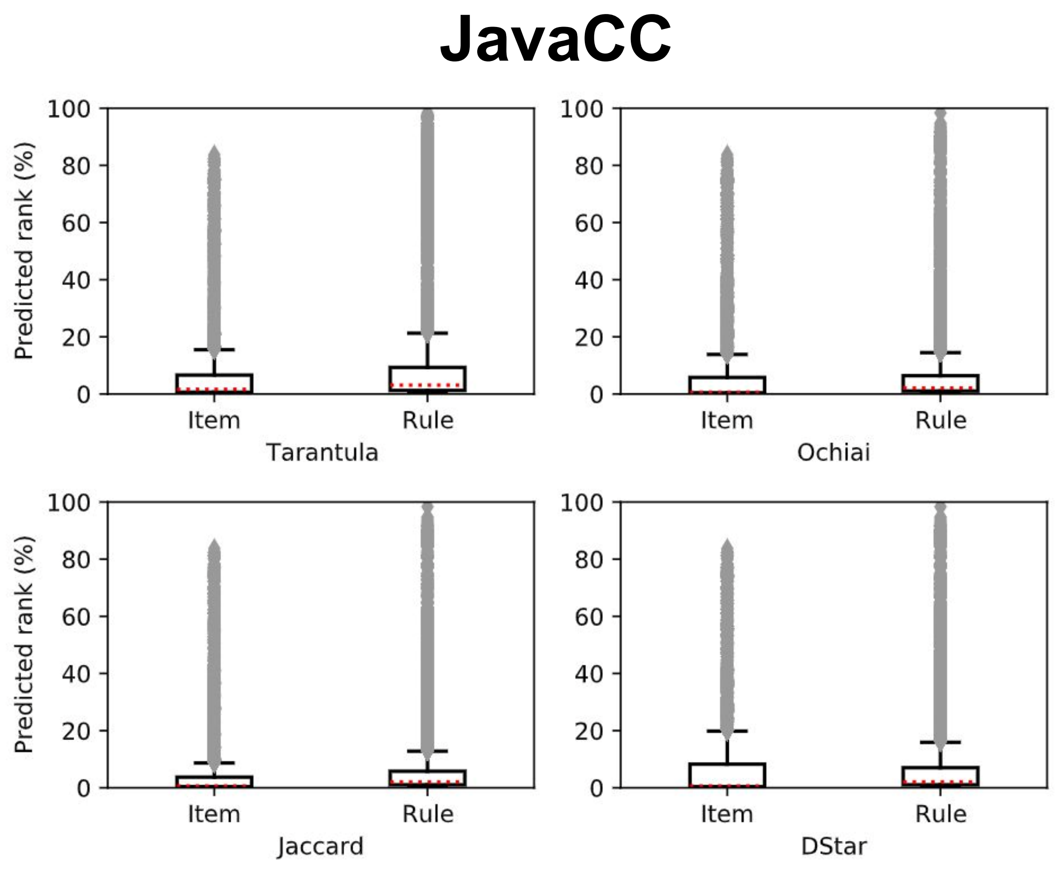
Rule Localization Results (I)

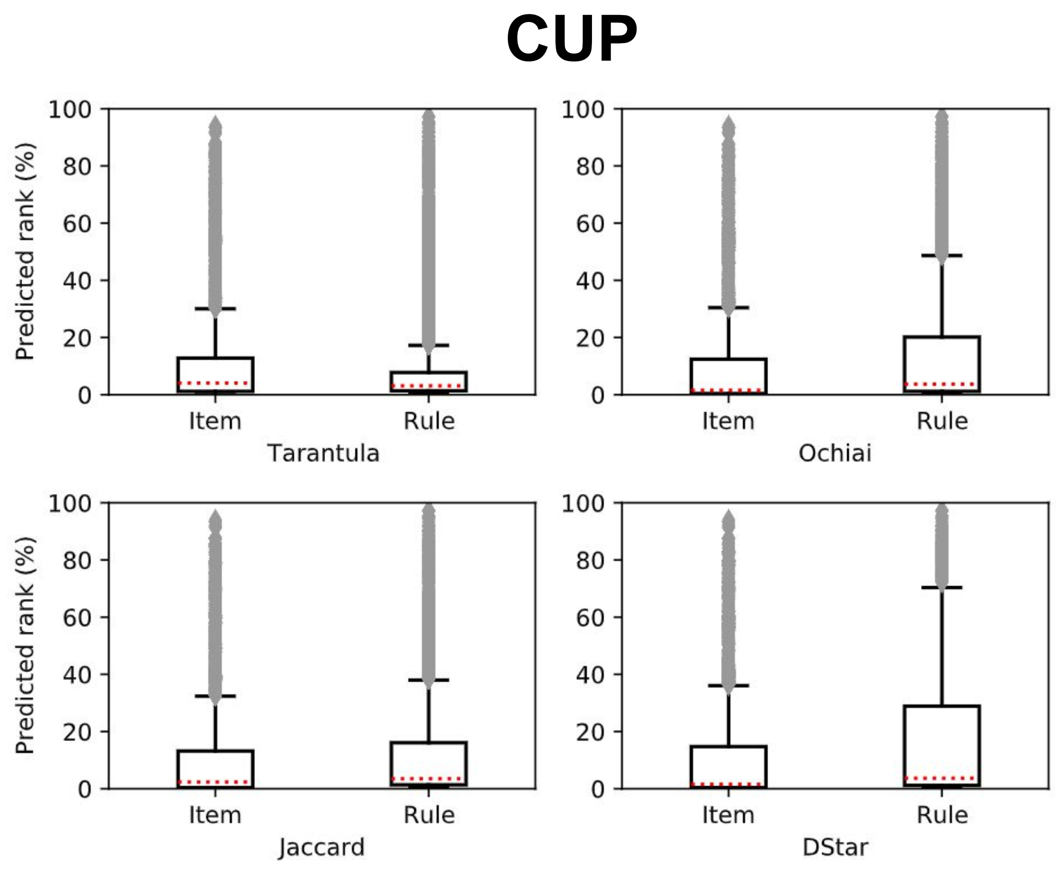


large: 2964 positive / 32157 negative test cases

- increased kill (prediction) rate
- large increases in rank #1 predictions

Item-level localization reduces repair search space.





large: 2964 positive / 32157 negative test cases

- *JavaCC:* median rank ~2% for item spectra and ~2.5% for rule spectra
- *CUP:* median rank ~3% for item spectra and ~4% for rule spectra

Automatic Grammar Repair

Grammar repair automates the find-and-fix cycle.

We follow a **generate-and-validate** approach that takes as faulty input grammar G and construct a grammar G' that passes all test from a **specification test suite**.

Grammar repair automates the find-and-fix cycle.

We follow a **generate-and-validate** approach that takes as faulty input grammar *G* and construct a grammar *G′* that passes all test from a **specification test suite**.

1. localize: identifies **repair sites** using item-level localization

suspicious items

Grammar repair automates the find-and-fix cycle.

We follow a **generate-and-validate** approach that takes as faulty input grammar *G* and construct a grammar *G′* that passes all test from a **specification test suite**.

1. localize: identifies **repair sites** using item-level localization

suspicious items

2. transform: apply small scale **patches** at these sites

Grammar repair automates the find-and-fix cycle.

We follow a **generate-and-validate** approach that takes as faulty input grammar *G* and construct a grammar *G′* that passes all test from a **specification test suite**.

1. localize: identifies **repair sites** using item-level localization

   suspicious items

2. transform: apply small scale **patches** at these sites
3. validate: check **patch pre-** and **postconditions**, re-run patched grammars on tests, maintain a queue to keep **improving the most promising** candidate grammars

Grammar repair automates the find-and-fix cycle.

We follow a **generate-and-validate** approach that takes as faulty input grammar *G* and construct a grammar *G′* that passes all test from a **specification test suite**.

1. localize: identifies **repair sites** using item-level localization

   suspicious items

2. transform: apply small scale **patches** at these sites
3. validate: check **patch pre-** and **postconditions**, re-run patched grammars on tests, maintain a queue to keep **improving the most promising** candidate grammars
4. control: alternate between localization, transformation, and validation (until fixed)

Grammar repair automates the find-and-fix cycle.

We follow a **generate-and-validate** approach that takes as faulty input grammar *G* and construct a grammar *G′* that passes all test from a **specification test suite**.

1. localize: identifies **repair sites** using item-level localization

   suspicious items

2. transform: apply small scale **patches** at these sites
3. validate: check **patch pre-** and **postconditions**, re-run patched grammars on tests, maintain a queue to keep **improving the most promising** candidate grammars
4. control: alternate between localization, transformation, and validation (until fixed)

Passive repair

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

fixed test suite

Grammar repair automates the find-and-fix cycle.

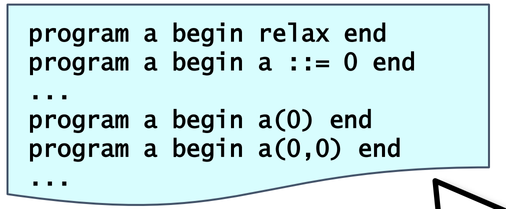
We follow a **generate-and-validate** approach that takes as faulty input grammar *G* and construct a grammar *G′* that passes all test from a **specification test suite**.

1. localize: identifies **repair sites** using item-level localization

suspicious items

2. transform: apply small scale **patches** at these sites
3. validate: check **patch pre-** and **postconditions**, re-run patched grammars on tests, maintain a queue to keep **improving the most promising** candidate grammars
4. control: alternate between localization, transformation, and validation (until fixed)

Passive repair



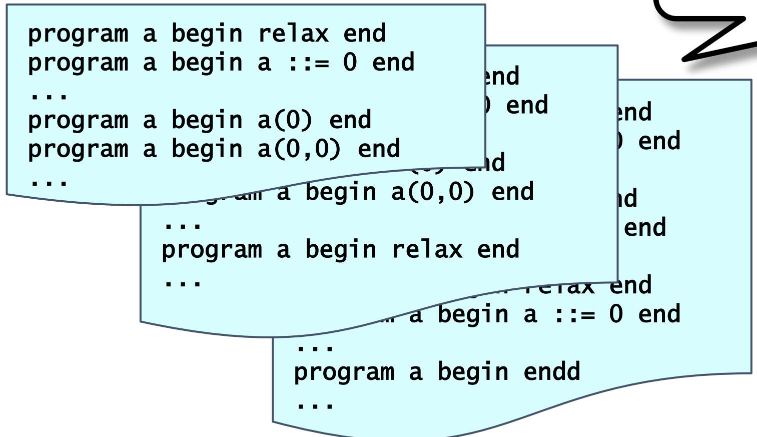
```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

fixed test suite

Active repair

evolving test suite

new tests derived from generated candidates

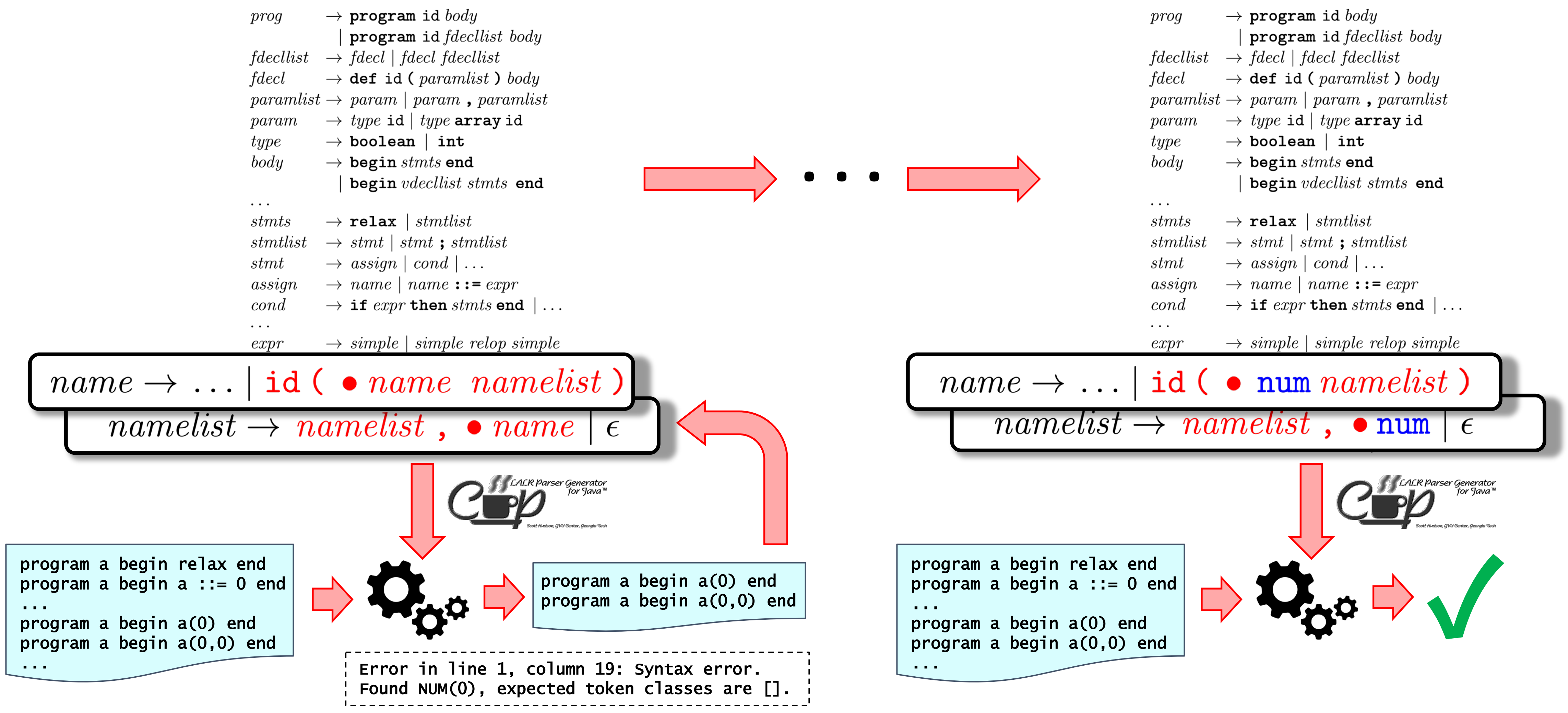


```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
...
program a begin relax end
...
```

```
...
program a begin endd
...
```

Passive repair repairs against a fixed test suite.



```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
```

$name \rightarrow \ldots \mid$ id (● $name$ $namelist$)

$namelist \rightarrow namelist$, ● $name \mid \epsilon$

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

CUP LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
```

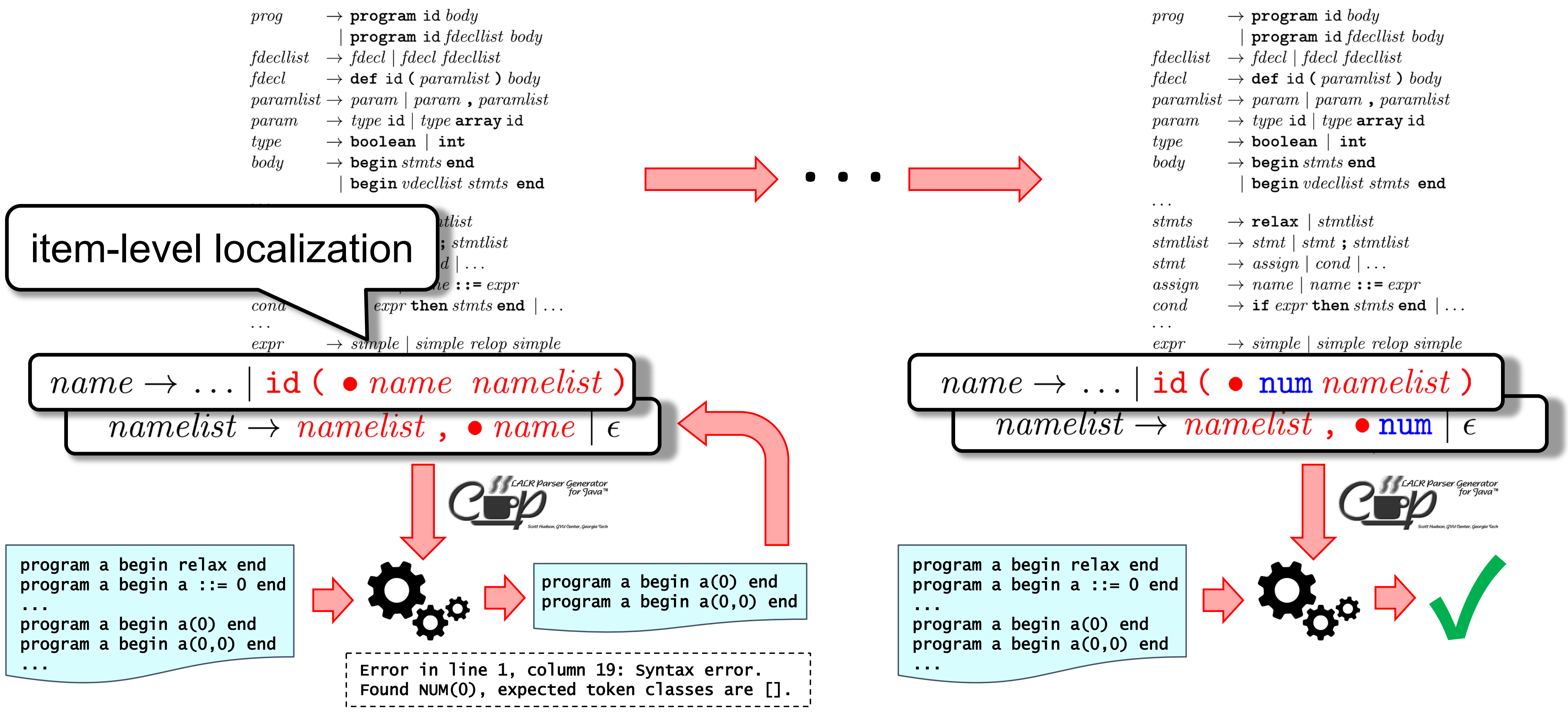
$name \rightarrow \ldots \mid$ id (● num $namelist$)

$namelist \rightarrow namelist$, ● num $\mid \epsilon$

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

CUP LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech
✓

Passive repair repairs against a fixed test suite.



item-level localization
LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech
...

```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
```

$name \to \ldots \mid$ id (• *name namelist*)

$namelist \to$ *namelist* , • *name* $\mid \epsilon$

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

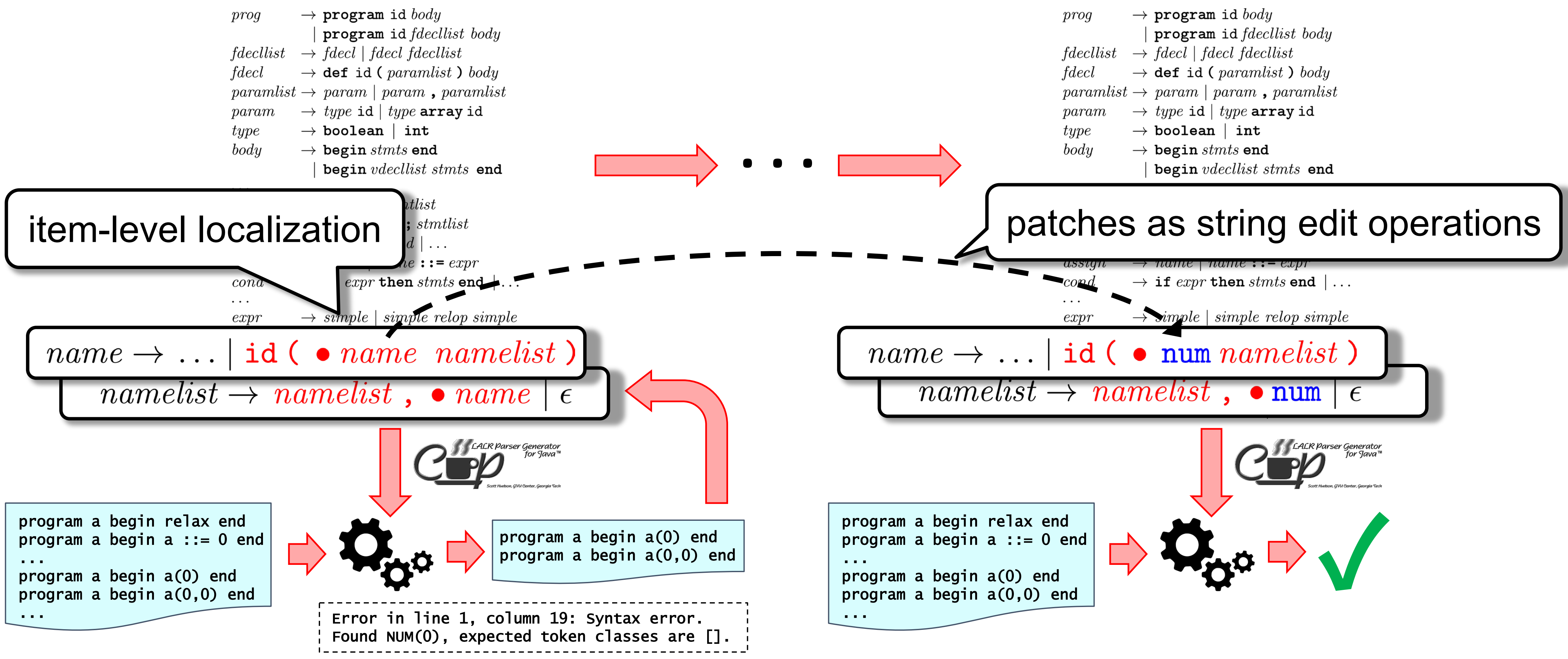
$name \to \ldots \mid$ id (• num *namelist*)

$namelist \to$ *namelist* , • num $\mid \epsilon$

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

✓

Passive repair repairs against a fixed test suite.


item-level localization
patches as string edit operations
. . .
LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech
✓

```
prog       → program id body
           | program id fdecllist body
fdecllist  → fdecl | fdecl fdecllist
fdecl      → def id ( paramlist ) body
paramlist  → param | param , paramlist
param      → type id | type array id
type       → boolean | int
body       → begin stmts end
           | begin vdecllist stmts end
...
cond       → ... expr then stmts end | ...
...
expr       → simple | simple relop simple
```

$name \rightarrow \ldots \mid$ id ($\bullet$ $name$ $namelist$)

$namelist \rightarrow namelist$, $\bullet$ $name$ $\mid \epsilon$

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

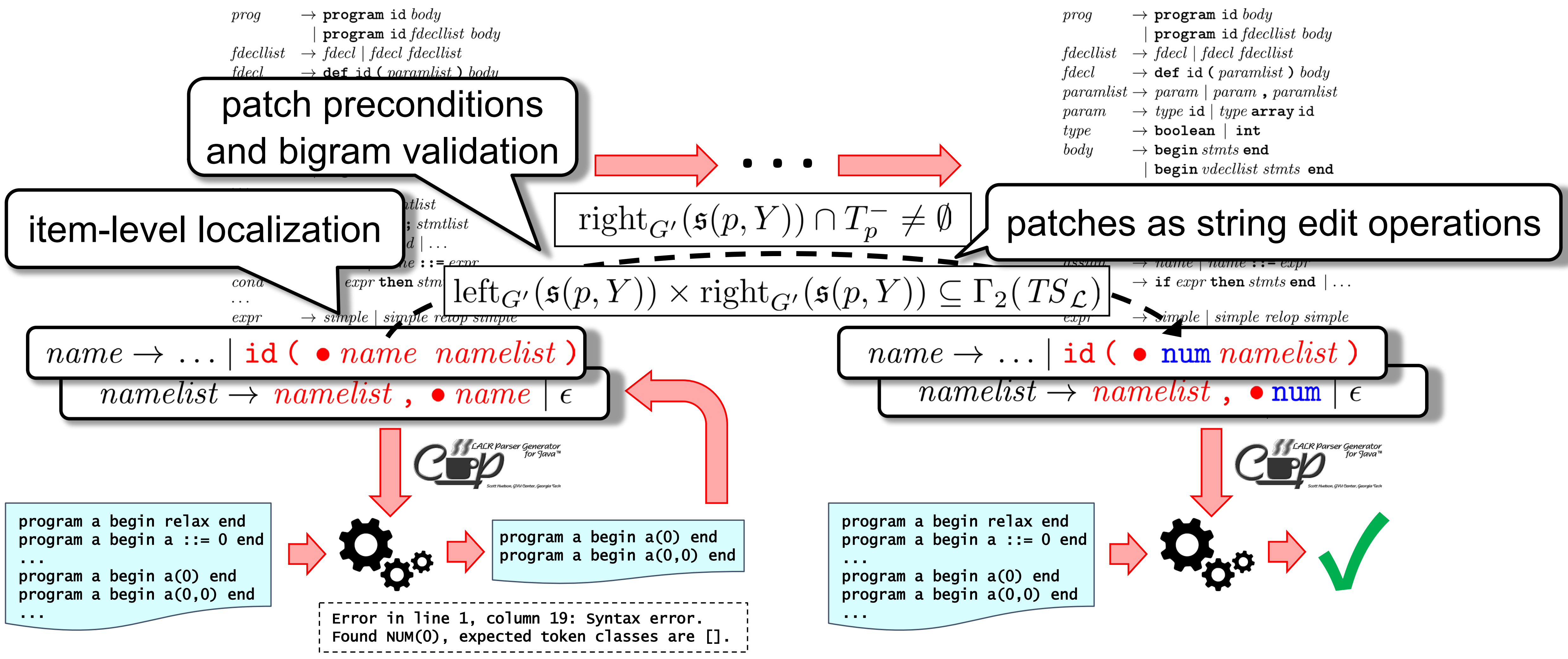
```
prog       → program id body
           | program id fdecllist body
fdecllist  → fdecl | fdecl fdecllist
fdecl      → def id ( paramlist ) body
paramlist  → param | param , paramlist
param      → type id | type array id
type       → boolean | int
body       → begin stmts end
           | begin vdecllist stmts end
assign     → name | name ::= expr
cond       → if expr then stmts end | ...
...
expr       → simple | simple relop simple
```

$name \rightarrow \ldots \mid$ id ($\bullet$ num $namelist$)

$namelist \rightarrow namelist$, $\bullet$ num $\mid \epsilon$

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

Passive repair repairs against a fixed test suite.



patch preconditions and bigram validation
item-level localization
patches as string edit operations
$\mathrm{right}_{G'}(\mathfrak{s}(p,Y)) \cap T_p^- \neq \emptyset$
$\mathrm{left}_{G'}(\mathfrak{s}(p,Y)) \times \mathrm{right}_{G'}(\mathfrak{s}(p,Y)) \subseteq \Gamma_2(TS_{\mathcal{L}})$
$name \to \dots \mid$ id (• $name$ $namelist$)
$namelist \to namelist$, • $name \mid \epsilon$
$name \to \dots \mid$ id (• num $namelist$)
$namelist \to namelist$, • num $\mid \epsilon$
$prog \to$ program id $body$ | program id $fdecllist$ $body$
$fdecllist \to fdecl \mid fdecl\ fdecllist$
$fdecl \to$ def id ($paramlist$) $body$
$paramlist \to param \mid param$, $paramlist$
$param \to type$ id | $type$ array id
$type \to$ boolean | int
$body \to$ begin $stmts$ end | begin $vdecllist$ $stmts$ end
$cond$ … $expr$ then $stmts$ end | …
$expr \to simple \mid simple\ relop\ simple$
. . .
CUP LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech
✓

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

```
program a begin a(0) end
program a begin a(0,0) end
```

```
Error in line 1, column 19: Syntax error.
Found NUM(0), expected token classes are [].
```

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
```

Grammar patches via symbol string edit operations

Grammar patches via symbol string edit operations

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

Grammar patches via symbol string edit operations

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$name \rightarrow \texttt{id}\ \texttt{(}\ \bullet\ name\ namelist\ \texttt{)} \leadsto_{\mathfrak{s}(\texttt{num})}\ name \rightarrow \texttt{id}\ \texttt{(}\ \bullet\ \texttt{num}\ namelist\ \texttt{)}$$

Grammar patches via symbol string edit operations

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$\mathit{name} \rightarrow \texttt{id} \; \texttt{(} \; \bullet \; \mathit{name} \; \mathit{namelist} \; \texttt{)} \rightsquigarrow_{\mathfrak{s}(\texttt{num})} \; \mathit{name} \rightarrow \texttt{id} \; \texttt{(} \; \bullet \; \texttt{num} \; \mathit{namelist} \; \texttt{)}$$

- collect ***lexical information*** around ***parse error*** locations (per item)

Grammar patches via symbol string edit operations

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$name \to \texttt{id}\,(\,\bullet\, name\ namelist\,) \leadsto_{\mathfrak{s}(\texttt{num})} name \to \texttt{id}\,(\,\bullet\,\texttt{num}\ namelist\,)$$

- collect ***lexical information*** around ***parse error*** locations (per item)

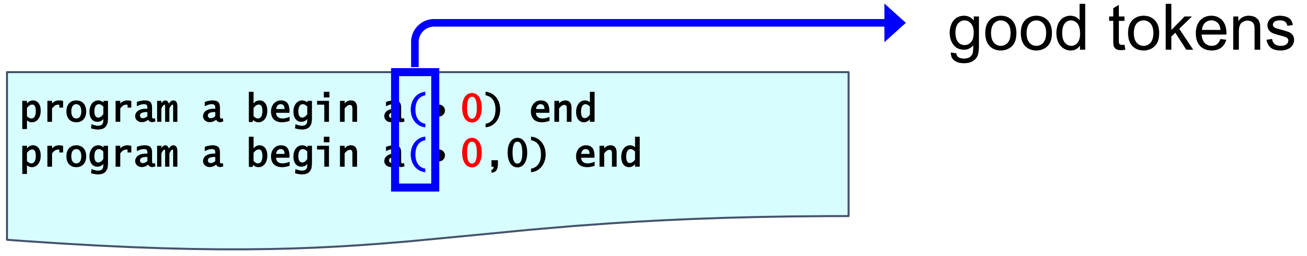
```
program a begin a(• 0) end
program a begin a(• 0,0) end
```

Grammar patches via symbol string edit operations

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$name \to \mathtt{id}\ (\ \bullet\ name\ namelist\ ) \rightsquigarrow_{\mathfrak{s}(\mathtt{num})}\ name \to \mathtt{id}\ (\ \bullet\ \mathtt{num}\ namelist\ )$$

- collect ***lexical information*** around ***parse error*** locations (per item)



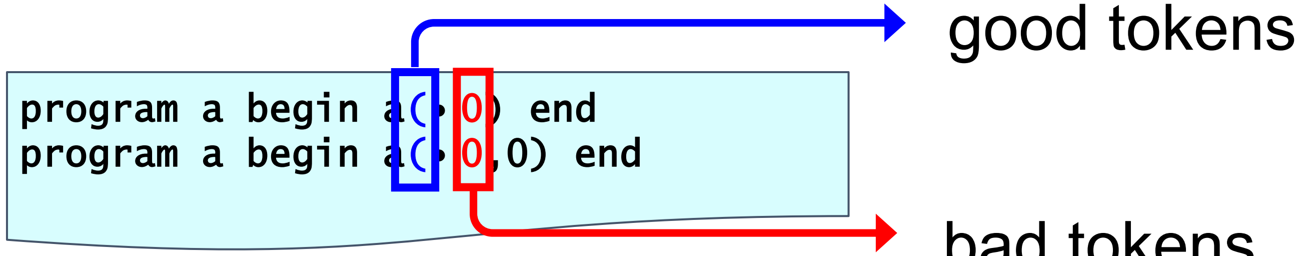
```
program a begin a(• 0) end
program a begin a(• 0,0) end
```

Grammar patches via symbol string edit operations

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$\mathit{name} \rightarrow \texttt{id} \; (\, \bullet \; \mathit{name} \; \mathit{namelist} \,) \leadsto_{\mathfrak{s}(\texttt{num})} \mathit{name} \rightarrow \texttt{id} \; (\, \bullet \; \texttt{num} \; \mathit{namelist} \,)$$

- collect ***lexical information*** around ***parse error*** locations (per item)



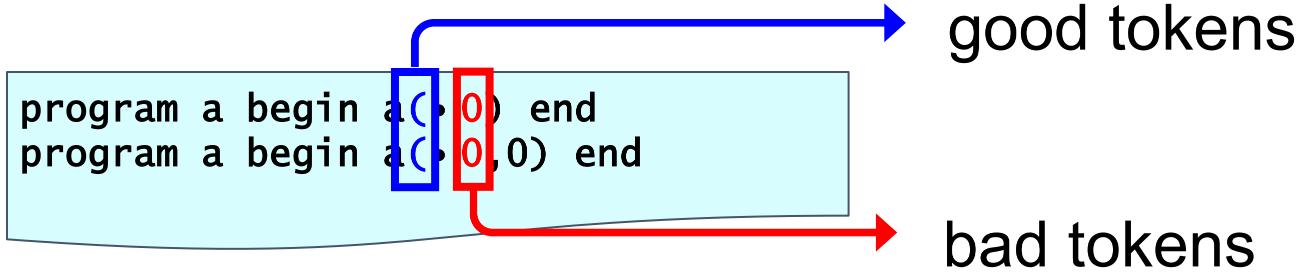
```
program a begin a(•0) end
program a begin a(•0,0) end
```

Grammar patches via symbol string edit operations

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$name \rightarrow \texttt{id}\ (\ \bullet\ name\ namelist\ ) \rightsquigarrow_{\mathfrak{s}(\textbf{num})}\ name \rightarrow \texttt{id}\ (\ \bullet\ \textbf{num}\ namelist\ )$$

- collect ***lexical information*** around ***parse error*** locations (per item)



```
program a begin a(• 0) end
program a begin a(• 0,0) end
```

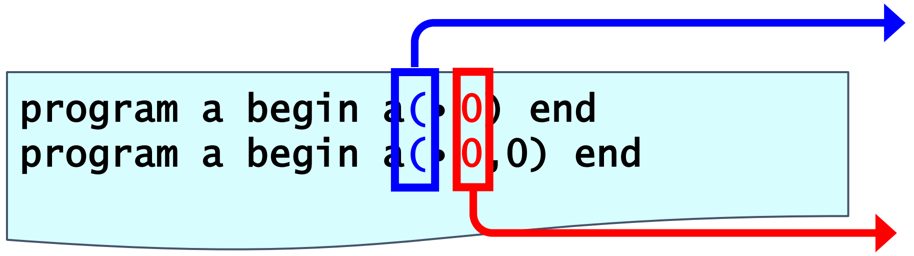
- … use it to ***eliminate*** candidate ***items*** that do not directly reflect error location

Grammar patches via symbol string edit operations

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$name \rightarrow \texttt{id}\ (\ \bullet\ name\ namelist\ ) \rightsquigarrow_{\mathfrak{s}(\texttt{num})} name \rightarrow \texttt{id}\ (\ \bullet\ \texttt{num}\ namelist\ )$$

- collect ***lexical information*** around ***parse error*** locations (per item)



```
program a begin a(•0) end
program a begin a(•0,0) end
```

good tokens T_p^+

bad tokens T_p^-

$$\text{left}(A \rightarrow \alpha \bullet \beta) = \begin{cases} \text{last}(\alpha) \cup \text{precede}(A) & \text{if } \alpha \text{ nullable} \\ \text{last}(\alpha) & \text{otherwise} \end{cases}$$

$$\text{right}(A \rightarrow \alpha \bullet \beta) = \begin{cases} \text{first}(\beta) \cup \text{follow}(A) & \text{if } \beta \text{ nullable} \\ \text{first}(\beta) & \text{otherwise} \end{cases}$$

- … use it to ***eliminate*** candidate ***items*** that do not direc y reflect error location

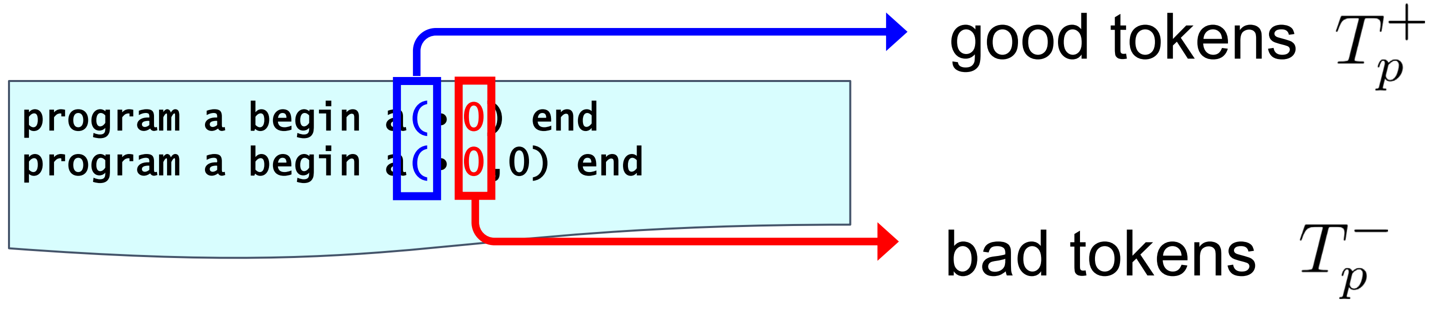
  check $\text{left}_G(p) \subseteq T_p^+$ (or weak form: $\text{left}_G(p) \cap T_p^+ \neq \emptyset$)

Grammar patches via symbol string edit operations

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$name \rightarrow \texttt{id}\ (\ \bullet\ name\ namelist\ ) \rightsquigarrow_{\mathfrak{s}(\texttt{num})}\ name \rightarrow \texttt{id}\ (\ \bullet\ \texttt{num}\ namelist\ )$$

- collect ***lexical information*** around ***parse error*** locations (per item)



```
program a begin a(•0) end
program a begin a(•0,0) end
```

$$\text{left}(A \rightarrow \alpha \bullet \beta) = \begin{cases} \text{last}(\alpha) \cup \text{precede}(A) & \text{if } \alpha \text{ nullable} \\ \text{last}(\alpha) & \text{otherwise} \end{cases}$$

$$\text{right}(A \rightarrow \alpha \bullet \beta) = \begin{cases} \text{first}(\beta) \cup \text{follow}(A) & \text{if } \beta \text{ nullable} \\ \text{first}(\beta) & \text{otherwise} \end{cases}$$

- … use it to ***eliminate*** candidate ***items*** that do not directly reflect error location

  check $\text{left}_G(p) \subseteq T_p^+$ (or weak form: $\text{left}_G(p) \cap T_p^+ \neq \emptyset$)

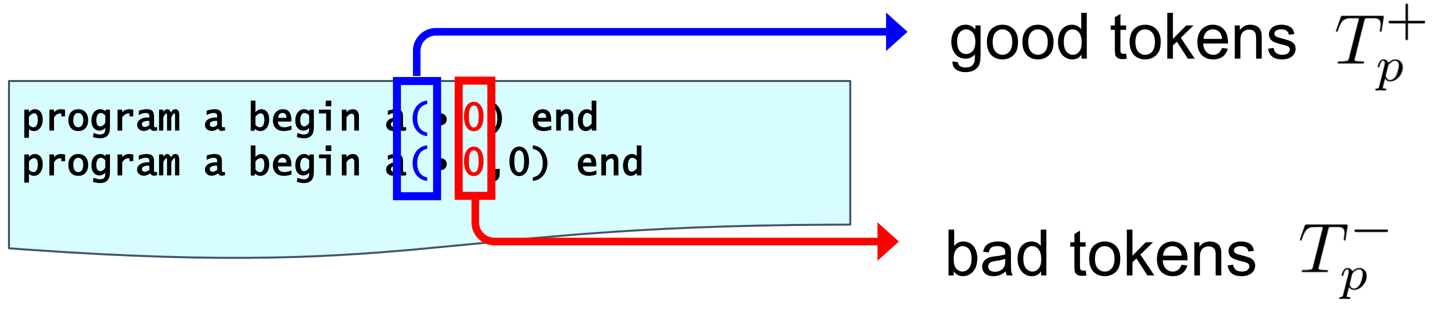
- … use it to ***eliminate*** candidate ***edits*** that cannot improve the grammar

Grammar patches via symbol string edit operations

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$name \to \texttt{id}\ (\ \bullet\ name\ namelist\ )\ \rightsquigarrow_{\mathfrak{s}(\texttt{num})}\ name \to \texttt{id}\ (\ \bullet\ \texttt{num}\ namelist\ )$$

- collect ***lexical information*** around ***parse error*** locations (per item)



```
program a begin a(•0) end
program a begin a(•0,0) end
```

$$\text{left}(A \to \alpha \bullet \beta) = \begin{cases} \text{last}(\alpha) \cup \text{precede}(A) & \text{if } \alpha \text{ nullable} \\ \text{last}(\alpha) & \text{otherwise} \end{cases}$$

$$\text{right}(A \to \alpha \bullet \beta) = \begin{cases} \text{first}(\beta) \cup \text{follow}(A) & \text{if } \beta \text{ nullable} \\ \text{first}(\beta) & \text{otherwise} \end{cases}$$

- … use it to ***eliminate*** candidate ***items*** that do not directly reflect error location

  check $\text{left}_G(p) \subseteq T_p^+$ (or weak form: $\text{left}_G(p) \cap T_p^+ \neq \emptyset$)

- … use it to ***eliminate*** candidate ***edits*** that cannot improve the grammar

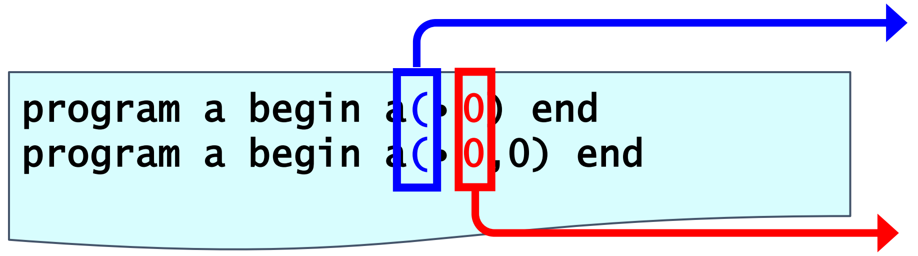
  allow only substitutions $\mathfrak{s}(Y)$ such that $\text{right}_{G'}(\mathfrak{s}(p, Y)) \cap T_p^- \neq \emptyset$

Grammar patches via symbol string edit operations

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

  $name \rightarrow \texttt{id}\ (\ \bullet\ name\ namelist\ )\ \rightsquigarrow_{\mathfrak{s}(\texttt{num})}\ name \rightarrow \texttt{id}\ (\ \bullet\ \texttt{num}\ namelist\ )$

- collect ***lexical information*** around ***parse error*** locations (per item)



```
program a begin a(•0) end
program a begin a(•0,0) end
```

good tokens T_p^+

bad tokens T_p^-

$$\text{left}(A \rightarrow \alpha \bullet \beta) = \begin{cases} \text{last}(\alpha) \cup \text{precede}(A) & \text{if } \alpha \text{ nullable} \\ \text{last}(\alpha) & \text{otherwise} \end{cases}$$

$$\text{right}(A \rightarrow \alpha \bullet \beta) = \begin{cases} \text{first}(\beta) \cup \text{follow}(A) & \text{if } \beta \text{ nullable} \\ \text{first}(\beta) & \text{otherwise} \end{cases}$$

- … use it to ***eliminate*** candidate ***items*** that do not directly reflect error location

  check $\text{left}_G(p) \subseteq T_p^+$ (or weak form: $\text{left}_G(p) \cap T_p^+ \neq \emptyset$)

- … use it to ***eliminate*** candidate ***edits*** that cannot improve the grammar

  allow only substitutions $\mathfrak{s}(Y)$ such that $\text{right}_{G'}(\mathfrak{s}(p, Y)) \cap T_p^- \neq \emptyset$

  can also use $\mathfrak{s}(expr), \mathfrak{s}(simple), \mathfrak{s}(term), \mathfrak{s}(factor), \ldots$

Grammar patches via symbol string edit operations (cont.)

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

  $$name \rightarrow \texttt{id}\ \texttt{(}\ \bullet\ name\ namelist\ \texttt{)} \rightsquigarrow_{\mathfrak{s}(\texttt{num})}\ name \rightarrow \texttt{id}\ \texttt{(}\ \bullet\ \texttt{num}\ namelist\ \texttt{)}$$

- collect ***lexical information*** around ***parse error*** locations (per item)
- … use it to ***eliminate*** candidate ***items*** that do not directly reflect error location
- … use it to ***eliminate*** candidate ***edits*** that cannot not improve the grammar

Grammar patches via symbol string edit operations (cont.)

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$\mathit{name} \rightarrow \mathtt{id}\ \mathtt{(}\ \bullet\ \mathit{name}\ \mathit{namelist}\ \mathtt{)} \rightsquigarrow_{\mathfrak{s}(\mathtt{num})} \mathit{name} \rightarrow \mathtt{id}\ \mathtt{(}\ \bullet\ \mathtt{num}\ \mathit{namelist}\ \mathtt{)}$$

- collect ***lexical information*** around ***parse error*** locations (per item)
- … use it to ***eliminate*** candidate ***items*** that do not directly reflect error location
- … use it to ***eliminate*** candidate ***edits*** that cannot not improve the grammar
- collect ***lexical information*** from ***valid tests*** (***bigrams***)

```
program a begin relax end
program a
  def a(int a) begin relax end
  begin relax end
program a begin a(0) end
...
```

$$\begin{aligned}\Gamma_2(TS_{\mathcal{L}}) = \{&(\mathbf{program}, \mathtt{id}),\\ &(\mathtt{id}, \mathbf{begin}), (\mathtt{id}, \mathbf{def}), (\mathtt{id}, \mathbf{(}\,),\\ &(\mathbf{(}\,, \mathbf{bool}), (\mathbf{(}\,, \mathbf{int}), (\mathbf{(}\,, \mathtt{id}), (\mathbf{(}\,, \mathtt{num}),\\ &(\mathtt{num}, \mathbf{)}\,), (\mathtt{num}, \mathbf{,}\,), (\mathtt{num}, \mathbf{*}\,), (\mathtt{num}, \mathbf{+}\,), \ldots\}\end{aligned}$$

Grammar patches via symbol string edit operations (cont.)

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$\mathit{name} \rightarrow \texttt{id}\ (\ \bullet\ \mathit{name}\ \mathit{namelist}\ )\ \rightsquigarrow_{\mathfrak{s}(\texttt{num})}\ \mathit{name} \rightarrow \texttt{id}\ (\ \bullet\ \texttt{num}\ \mathit{namelist}\ )$$

- collect ***lexical information*** around ***parse error*** locations (per item)
- … use it to ***eliminate*** candidate ***items*** that do not directly reflect error location
- … use it to ***eliminate*** candidate ***edits*** that cannot not improve the grammar
- collect ***lexical information*** from ***valid tests*** (***bigrams***)

```
program a begin relax end
program a
  def a(int a) begin relax end
  begin relax end
program a begin a(0) end
...
```

$$\Gamma_2(TS_{\mathcal{L}}) = \{(\texttt{program}, \texttt{id}), (\texttt{id}, \texttt{begin}), (\texttt{id}, \texttt{def}), (\texttt{id}, \texttt{(}\,), (\texttt{(}\,, \texttt{bool}), (\texttt{(}\,, \texttt{int}), (\texttt{(}\,, \texttt{id}), (\texttt{(}\,, \texttt{num}), (\texttt{num}, \texttt{)}\,), (\texttt{num}, \texttt{,}\,), (\texttt{num}, \texttt{*}\,), (\texttt{num}, \texttt{+}\,), \ldots\}$$

- … use it to ***eliminate patches*** (***patch validation***)

Grammar patches via symbol string edit operations (cont.)

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$name \rightarrow \mathtt{id}\ \mathtt{(}\ \bullet\ name\ namelist\ \mathtt{)} \rightsquigarrow_{\mathfrak{s}(\mathtt{num})} name \rightarrow \mathtt{id}\ \mathtt{(}\ \bullet\ \mathtt{num}\ namelist\ \mathtt{)}$$

- collect ***lexical information*** around ***parse error*** locations (per item)
- … use it to ***eliminate*** candidate ***items*** that do not directly reflect error location
- … use it to ***eliminate*** candidate ***edits*** that cannot not improve the grammar
- collect ***lexical information*** from ***valid tests*** (***bigrams***)

```
program a begin relax end
program a
  def a(int a) begin relax end
  begin relax end
program a begin a(0) end
...
```

$$\Rightarrow$$

$$\begin{aligned}\Gamma_2(TS_{\mathcal{L}}) = \{&(\mathtt{\textbf{program}}, \mathtt{id}),\\ &(\mathtt{id}, \mathtt{\textbf{begin}}), (\mathtt{id}, \mathtt{\textbf{def}}), (\mathtt{id}, \mathtt{\textbf{(}}),\\ &(\mathtt{\textbf{(}}, \mathtt{\textbf{bool}}), (\mathtt{\textbf{(}}, \mathtt{\textbf{int}}), (\mathtt{\textbf{(}}, \mathtt{id}), (\mathtt{\textbf{(}}, \mathtt{num}),\\ &(\mathtt{num}, \mathtt{\textbf{)}}), (\mathtt{num}, \mathtt{\textbf{,}}), (\mathtt{num}, \mathtt{\textbf{*}}), (\mathtt{num}, \mathtt{\textbf{+}}), \ldots\}\end{aligned}$$

- … use it to ***eliminate patches*** (***patch validation***)

$$\mathrm{left}_{G'}(\mathfrak{s}(p, Y)) \times \mathrm{right}_{G'}(\mathfrak{s}(p, Y)) \subseteq \Gamma_2(TS_{\mathcal{L}}) \qquad \{(\mathtt{\textbf{(}}, \mathtt{num})\}$$

Grammar patches via symbol string edit operations (cont.)

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

$$\mathit{name} \to \mathtt{id}\ \mathtt{(}\ \bullet\ \mathit{name}\ \mathit{namelist}\ \mathtt{)} \rightsquigarrow_{\mathfrak{s}(\mathtt{num})} \mathit{name} \to \mathtt{id}\ \mathtt{(}\ \bullet\ \mathtt{num}\ \mathit{namelist}\ \mathtt{)}$$

- collect ***lexical information*** around ***parse error*** locations (per item)
- … use it to ***eliminate*** candidate ***items*** that do not directly reflect error location
- … use it to ***eliminate*** candidate ***edits*** that cannot not improve the grammar
- collect ***lexical information*** from ***valid tests*** (***bigrams***)

```
program a begin relax end
program a
  def a(int a) begin relax end
  begin relax end
program a begin a(0) end
...
```

$$\begin{aligned}\Gamma_2(TS_{\mathcal{L}}) = \{&(\mathbf{program}, \mathtt{id}),\\ &(\mathtt{id}, \mathbf{begin}), (\mathtt{id}, \mathbf{def}), (\mathtt{id}, \mathbf{(}\,),\\ &(\mathbf{(}\,, \mathbf{bool}), (\mathbf{(}\,, \mathbf{int}), (\mathbf{(}\,, \mathtt{id}), (\mathbf{(}\,, \mathtt{num}),\\ &(\mathtt{num}, \mathbf{)}\,), (\mathtt{num}, \mathbf{,}\,), (\mathtt{num}, \mathbf{*}\,), (\mathtt{num}, \mathbf{+}\,), \ldots\}\end{aligned}$$

- … use it to ***eliminate patches*** (***patch validation***)

$$\mathrm{left}_{G'}(\mathfrak{s}(p, Y)) \times \mathrm{right}_{G'}(\mathfrak{s}(p, Y)) \subseteq \Gamma_2(TS_{\mathcal{L}}) \quad \{(\mathbf{(}\,, \mathtt{num})\}$$

$$\mathrm{left}_{G'}(\mathfrak{s}(p, Y)') \times \mathrm{right}_{G'}(\mathfrak{s}(p, Y)') \subseteq \Gamma_2(TS_{\mathcal{L}}) \quad \{(\mathtt{num}, \mathbf{)}\,), (\mathtt{num}, \mathbf{,}\,)\}$$

$$\boxed{\mathit{name} \to \mathtt{id}\ \mathtt{(}\ \mathtt{num}\ \bullet\ \mathit{namelist}\ \mathtt{)}}$$

Grammar patches via symbol string edit operations (cont.)

- apply ***string edit operations*** $\mathfrak{e}$ at designated position of suspicious items

  $name \rightarrow \texttt{id} \; \texttt{(} \bullet \; name \; namelist \; \texttt{)} \rightsquigarrow_{\mathfrak{s}(\texttt{num})} name \rightarrow \texttt{id} \; \texttt{(} \bullet \; \texttt{num} \; namelist \; \texttt{)}$

- collect ***lexical information*** around ***parse error*** locations (per item)
- … use it to ***eliminate*** candidate ***items*** that do not directly reflect error location
- … use it to ***eliminate*** candidate ***edits*** that cannot not improve the grammar
- collect ***lexical information*** from ***valid tests*** (***bigrams***)

```
program a begin relax end
program a
  def a(int a) begin relax end
  begin relax end
program a begin a(0) end
...
```

$$\Gamma_2(TS_{\mathcal{L}}) = \{(\textbf{program}, \texttt{id}), (\texttt{id}, \textbf{begin}), (\texttt{id}, \textbf{def}), (\texttt{id}, \textbf{(}), (\textbf{(}, \textbf{bool}), (\textbf{(}, \textbf{int}), (\textbf{(}, \texttt{id}), (\textbf{(}, \texttt{num}), (\texttt{num}, \textbf{)}), (\texttt{num}, \textbf{,}), (\texttt{num}, \textbf{*}), (\texttt{num}, \textbf{+}), \ldots\}$$

- … use it to ***eliminate patches*** (***patch validation***)

$$\text{left}_{G'}(\mathfrak{s}(p,Y)) \times \text{right}_{G'}(\mathfrak{s}(p,Y)) \subseteq \Gamma_2(TS_{\mathcal{L}}) \quad \{(\textbf{(}, \texttt{num})\}$$

$$\text{left}_{G'}(\mathfrak{s}(p,Y)') \times \text{right}_{G'}(\mathfrak{s}(p,Y)') \subseteq \Gamma_2(TS_{\mathcal{L}}) \quad \{(\texttt{num}, \textbf{)}), (\texttt{num}, \textbf{,})\}$$

✓

$\boxed{name \rightarrow \texttt{id} \; \texttt{(} \; \texttt{num} \bullet namelist \; \texttt{)}}$

Active repair generates the test suites used for repair.

```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( name namelist )
namelist  → namelist , name | ϵ
```

program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...

initial test suite (optional)

Active repair generates the test suites used for repair.

```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( name namelist )
namelist  → namelist , name | ϵ
```

program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...

initial test suite (optional)

Active repair generates the test suites used for repair.

sentence generation

```
prog      → program id body
            list body
            ist
            t ) body
            paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( name namelist )
namelist  → namelist , name | ϵ
```

program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .

initial test suite (optional)

Active repair generates the test suites used for repair.

sentence generation

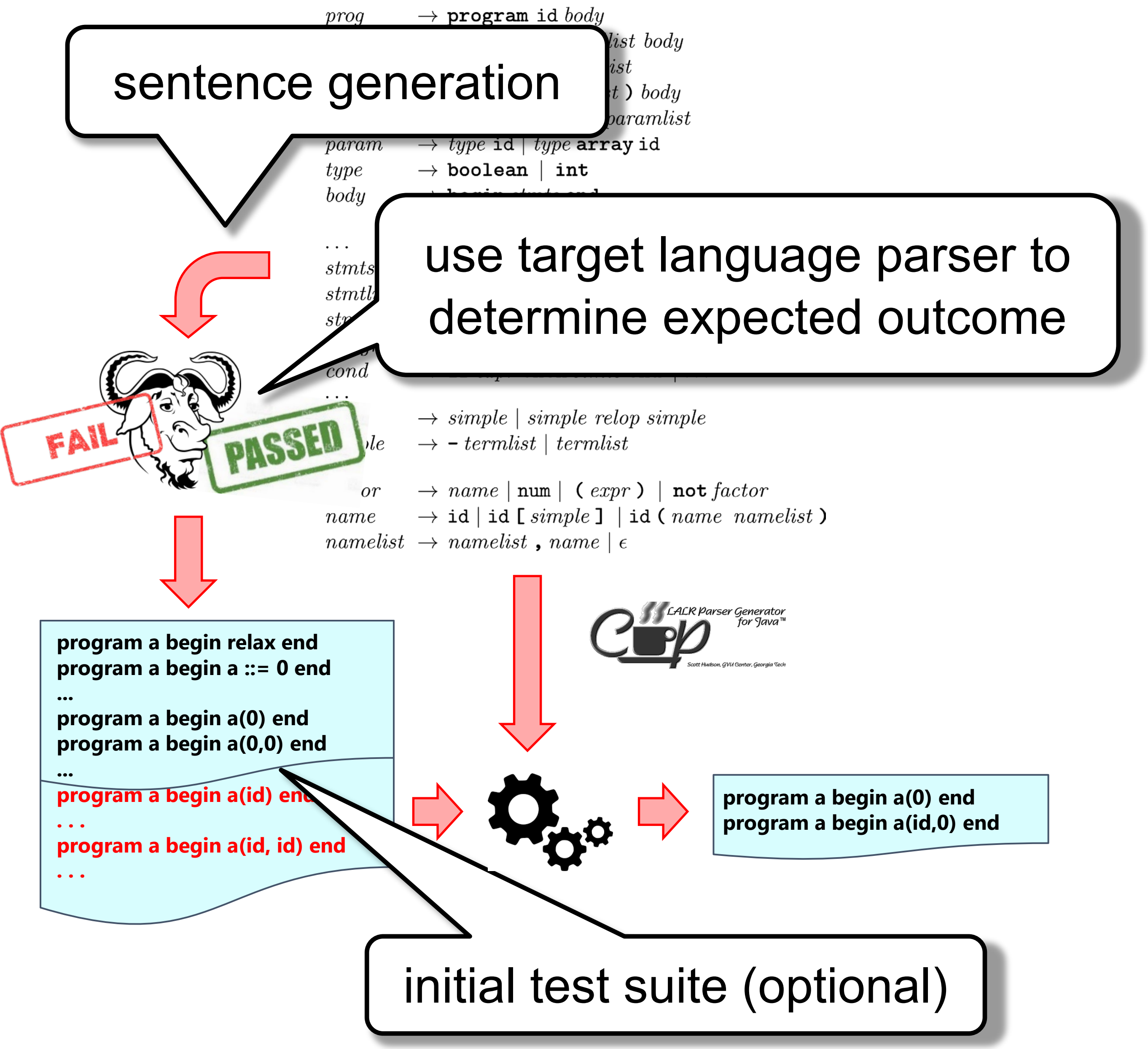
```
prog      → program id body
                         list body
                         ist
                         t ) body
                         paramlist
param     → type id | type array id
type      → boolean | int
body
...
stmts
stmtl
st
cond
...
          → simple | simple relop simple
   le     → - termlist | termlist

    or    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( name  namelist )
namelist  → namelist , name | ϵ
```

use target language parser to determine expected outcome

program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .

initial test suite (optional)

Active repair generates the test suites used for repair.



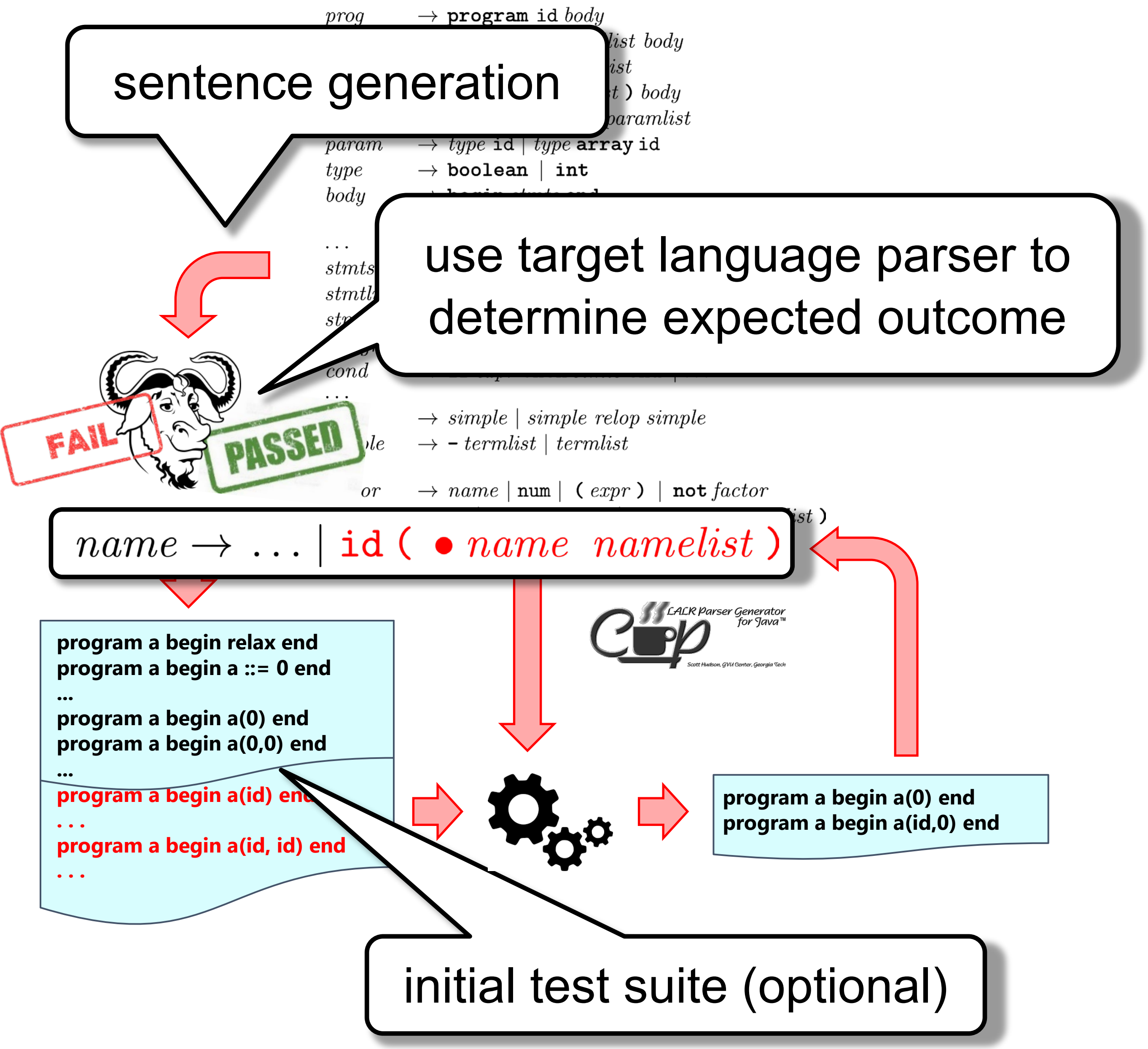
sentence generation
use target language parser to determine expected outcome
initial test suite (optional)
FAIL
PASSED
CUP LALR Parser Generator for Java

```
prog      → program id body
param     → type id | type array id
type      → boolean | int
body
...
stmts
cond
...
          → simple | simple relop simple
          → - termlist | termlist
          → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( name namelist )
namelist  → namelist , name | ε
```

program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .

program a begin a(0) end
program a begin a(id,0) end

Active repair generates the test suites used for repair.



prog → program id body
type → boolean | int
param → type id | type array id
→ simple | simple relop simple
→ - termlist | termlist
→ name | num | (expr) | not factor
FAIL
PASSED
CUP LALR Parser Generator for Java™

sentence generation

use target language parser to determine expected outcome

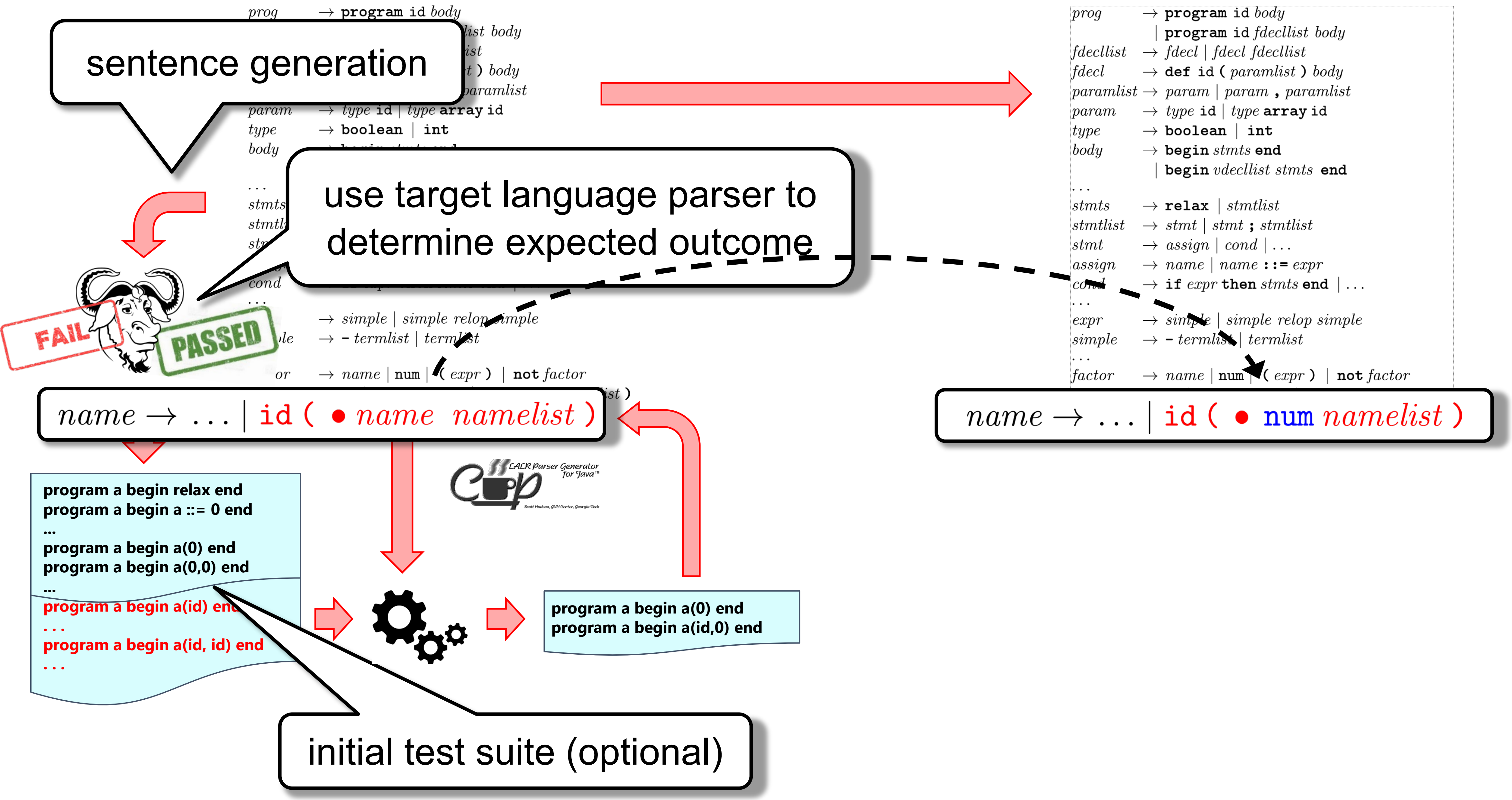
$name \rightarrow \ldots \mid$ id ($\bullet$ $name$ $namelist$)

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
```

```
program a begin a(0) end
program a begin a(id,0) end
```

initial test suite (optional)

Active repair generates the test suites used for repair.


sentence generation
use target language parser to determine expected outcome
FAIL
PASSED
CUP LALR Parser Generator for Java™
initial test suite (optional)

```
prog       → program id body
           | program id fdecllist body
fdecllist  → fdecl | fdecl fdecllist
fdecl      → def id ( paramlist ) body
paramlist  → param | param , paramlist
param      → type id | type array id
type       → boolean | int
body       → begin stmts end
           | begin vdecllist stmts end
...
stmts      → relax | stmtlist
stmtlist   → stmt | stmt ; stmtlist
stmt       → assign | cond | ...
assign     → name | name ::= expr
cond       → if expr then stmts end | ...
...
expr       → simple | simple relop simple
simple     → - termlist | termlist
...
factor     → name | num | ( expr ) | not factor
```

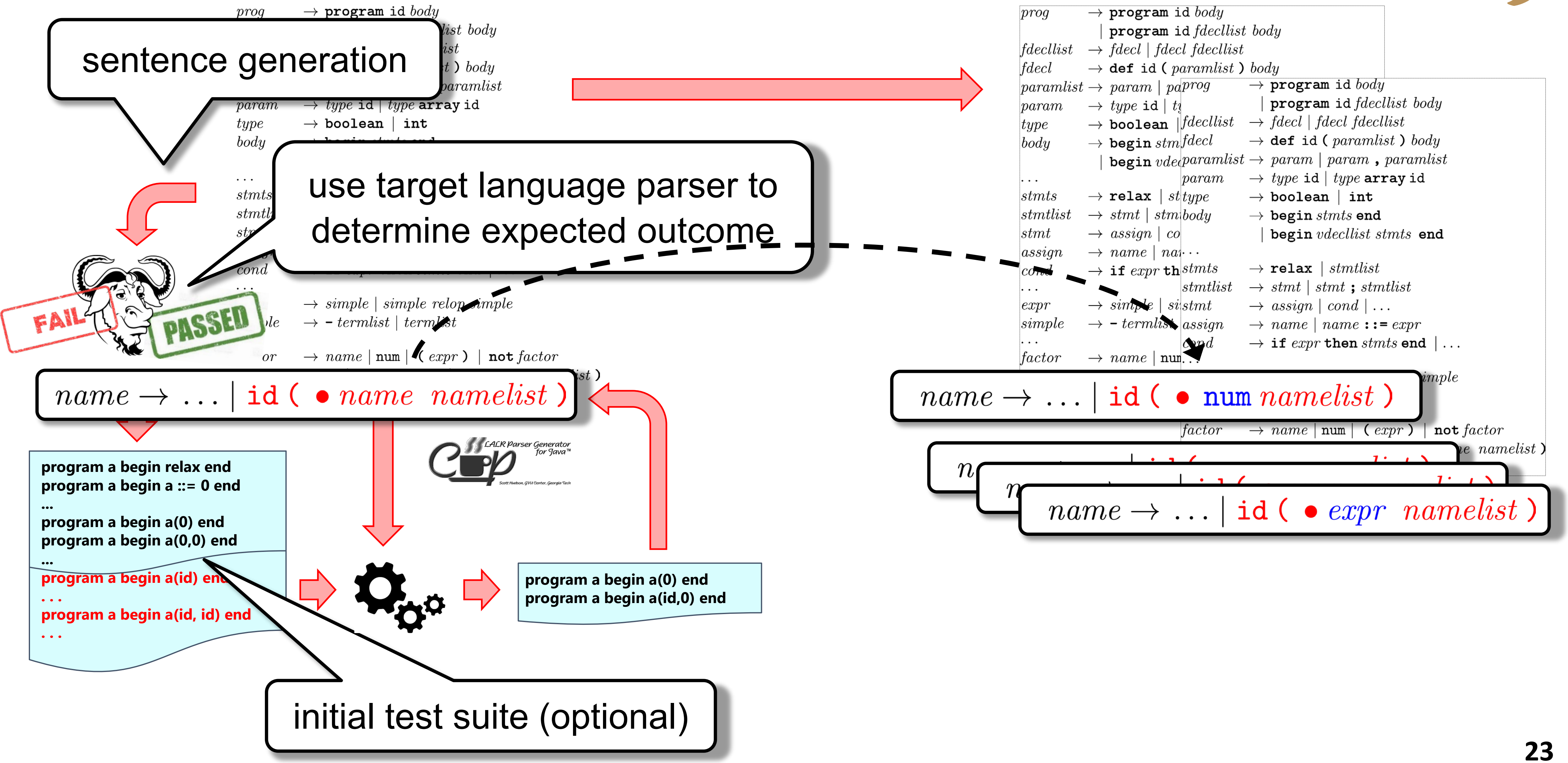
$name \rightarrow \ldots \mid \texttt{id}\ (\ \bullet\ name\ \ namelist\ )$

$name \rightarrow \ldots \mid \texttt{id}\ (\ \bullet\ \texttt{num}\ namelist\ )$

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
```

```
program a begin a(0) end
program a begin a(id,0) end
```

Active repair generates the test suites used for repair.


sentence generation
use target language parser to determine expected outcome
FAIL
PASSED
LALR Parser Generator for Java™
initial test suite (optional)

$name \rightarrow \ldots \mid \texttt{id}\ (\ \bullet\ name\ \ namelist\ )$

program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .

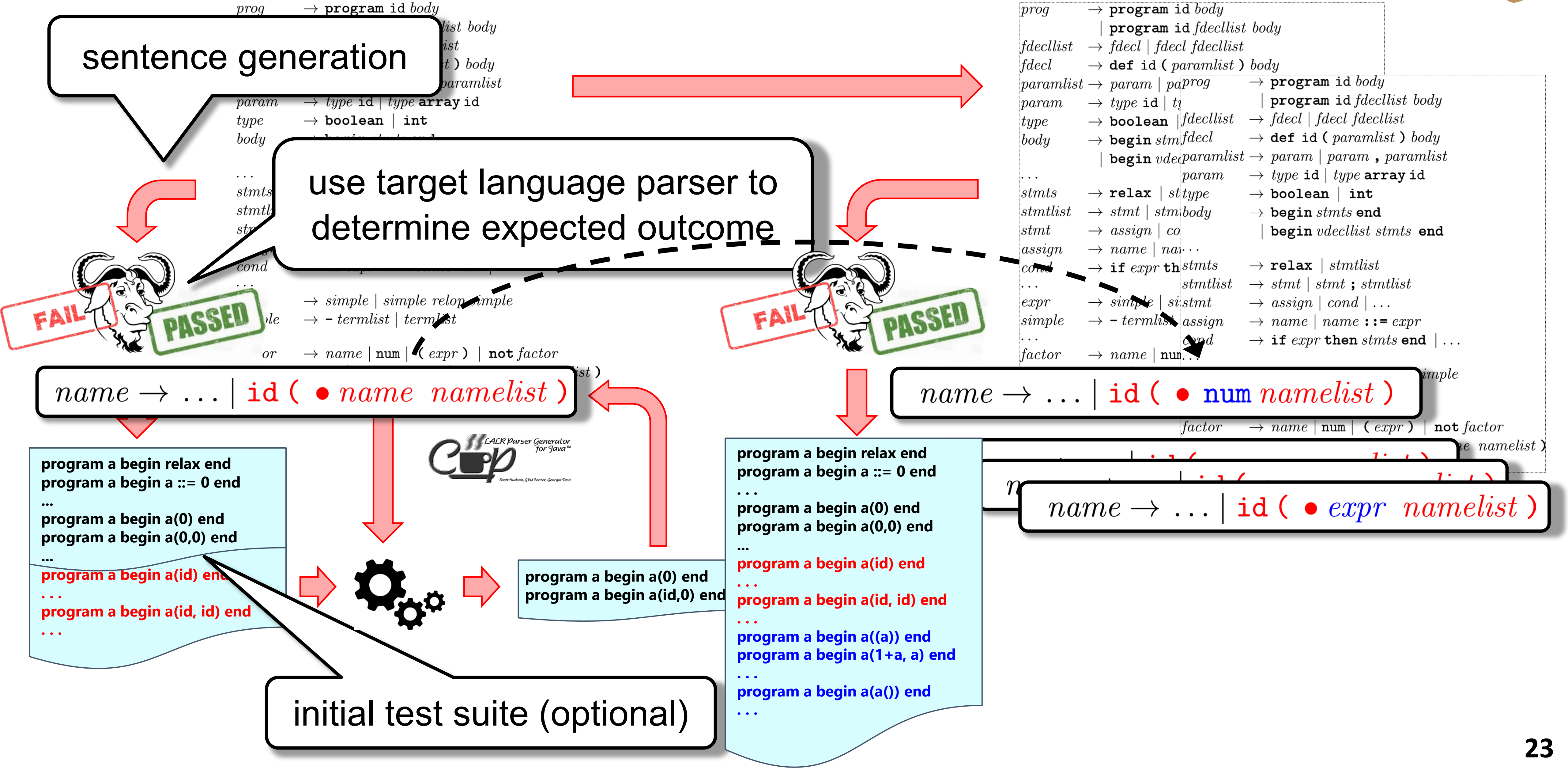
program a begin a(0) end
program a begin a(id,0) end

$name \rightarrow \ldots \mid \texttt{id}\ (\ \bullet\ \texttt{num}\ namelist\ )$

$name \rightarrow \ldots \mid \texttt{id}\ (\ \bullet\ expr\ \ namelist\ )$

prog	→	**program** id *body*
		| **program** id *fdecllist body*
fdecllist	→	*fdecl* | *fdecl fdecllist*
fdecl	→	**def** id (*paramlist*) *body*
paramlist	→	*param* | *param* , *paramlist*
param	→	*type* id | *type* **array** id
type	→	**boolean** | **int**
body	→	**begin** *stmts* **end**
		| **begin** *vdecllist stmts* **end**
. . .		
stmts	→	**relax** | *stmtlist*
stmtlist	→	*stmt* | *stmt* ; *stmtlist*
stmt	→	*assign* | *cond* | . . .
assign	→	*name* | *name* ::= *expr*
cond	→	**if** *expr* **then** *stmts* **end** | . . .
. . .		
factor	→	*name* | num | (*expr*) | **not** *factor*

Active repair generates the test suites used for repair.



sentence generation
use target language parser to determine expected outcome
initial test suite (optional)
FAIL
PASSED
CUP LALR Parser Generator for Java™
name → … | id (• name namelist)
name → … | id (• num namelist)
name → … | id (• expr namelist)

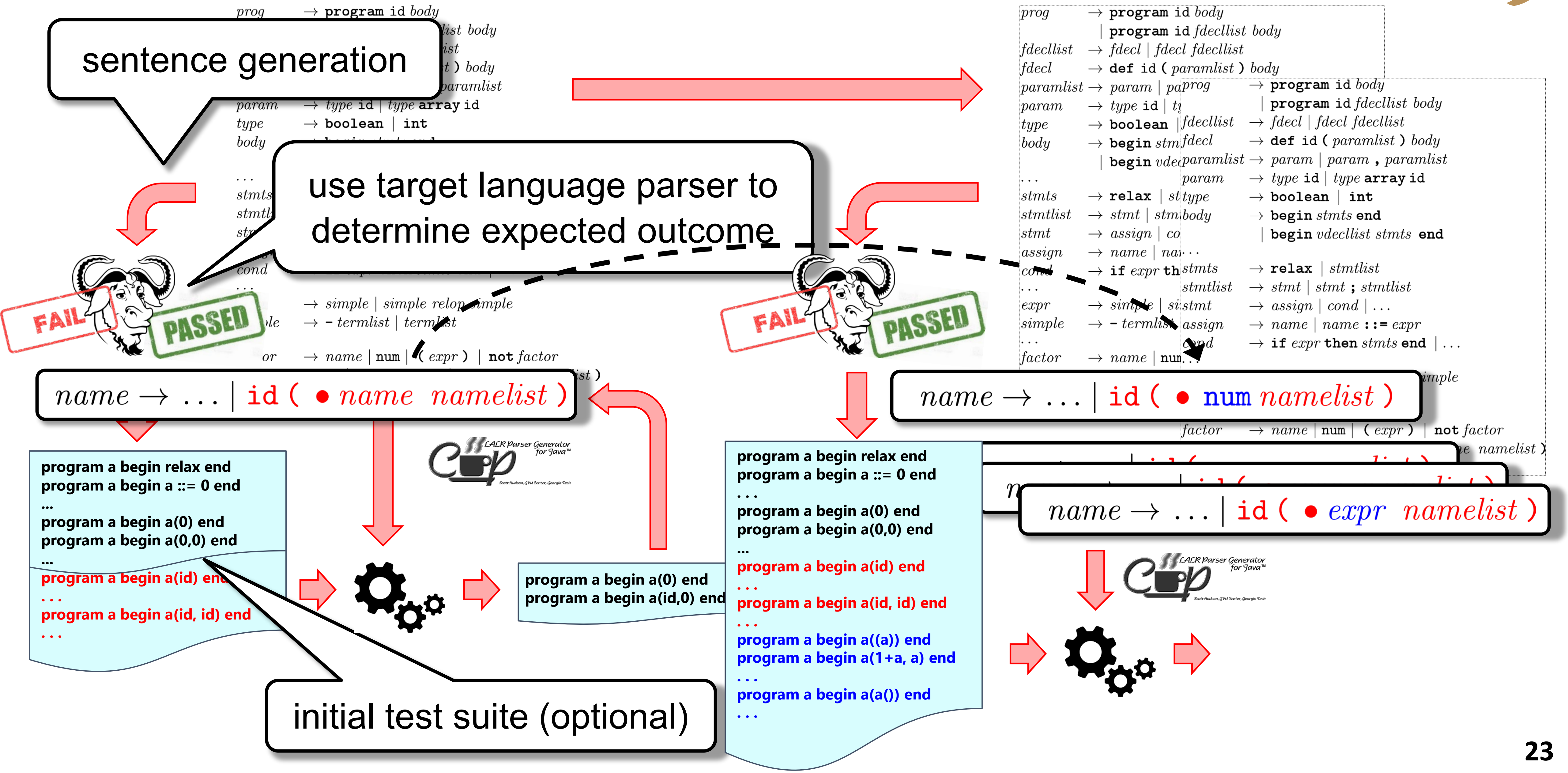
```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | …
assign    → name | name ::= expr
cond      → if expr then stmts end | …
expr      → simple | simple relop simple
simple    → - termlist | termlist
factor    → name | num | ( expr ) | not factor
```

program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .

program a begin a(0) end
program a begin a(id,0) end

program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .

Active repair generates the test suites used for repair.



sentence generation
use target language parser to determine expected outcome
initial test suite (optional)
FAIL
PASSED
CUP LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
```

name → ... | id (• name namelist)

name → ... | id (• num namelist)

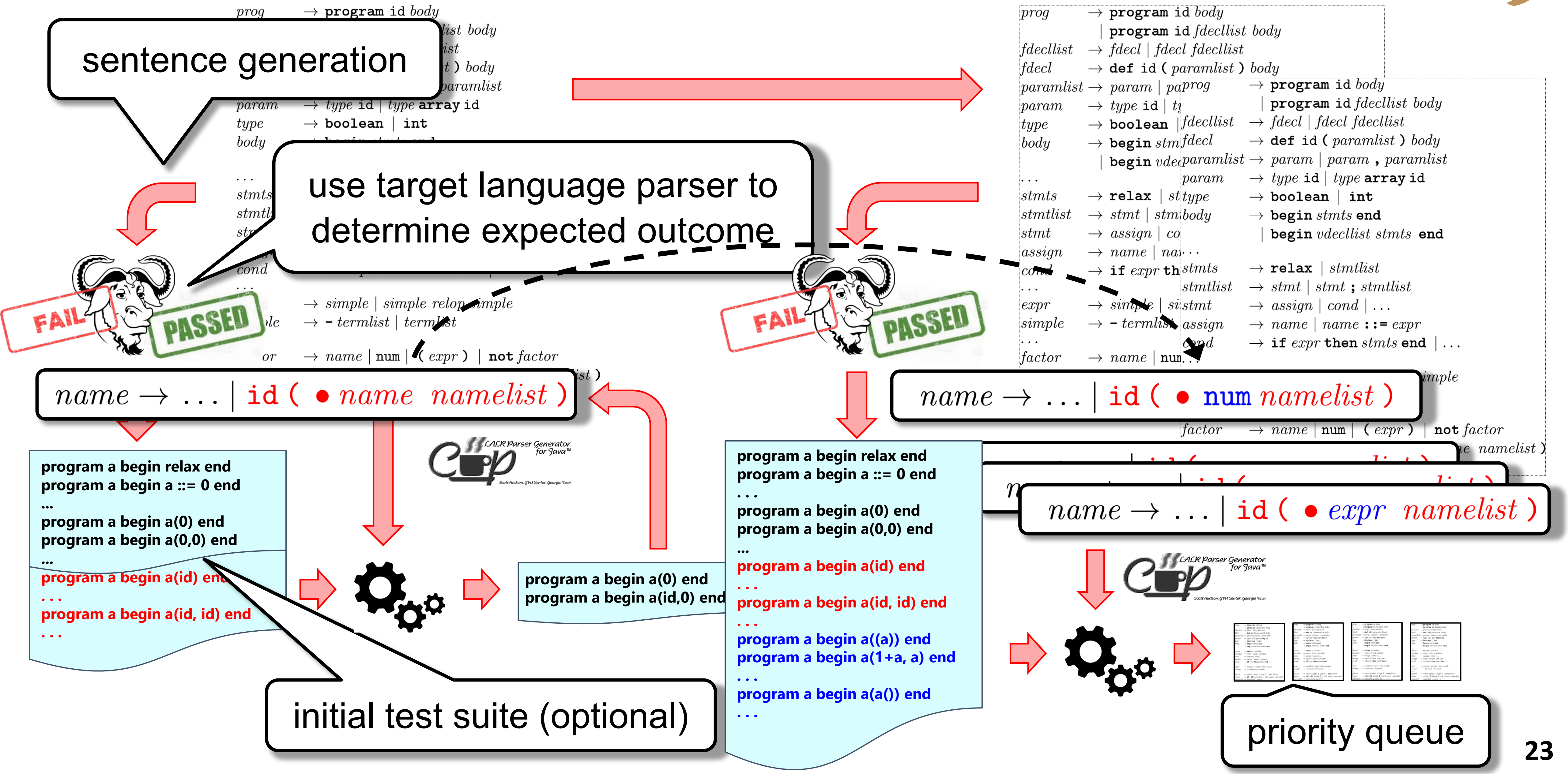
name → ... | id (• expr namelist)

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
```

```
program a begin a(0) end
program a begin a(id,0) end
```

```
program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .
```

Active repair generates the test suites used for repair.


sentence generation
use target language parser to determine expected outcome
FAIL
PASSED
FAIL
PASSED
LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech
initial test suite (optional)
priority queue

```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
```

$name \rightarrow \dots \mid$ id (• *name namelist*)

$name \rightarrow \dots \mid$ id (• num *namelist*)

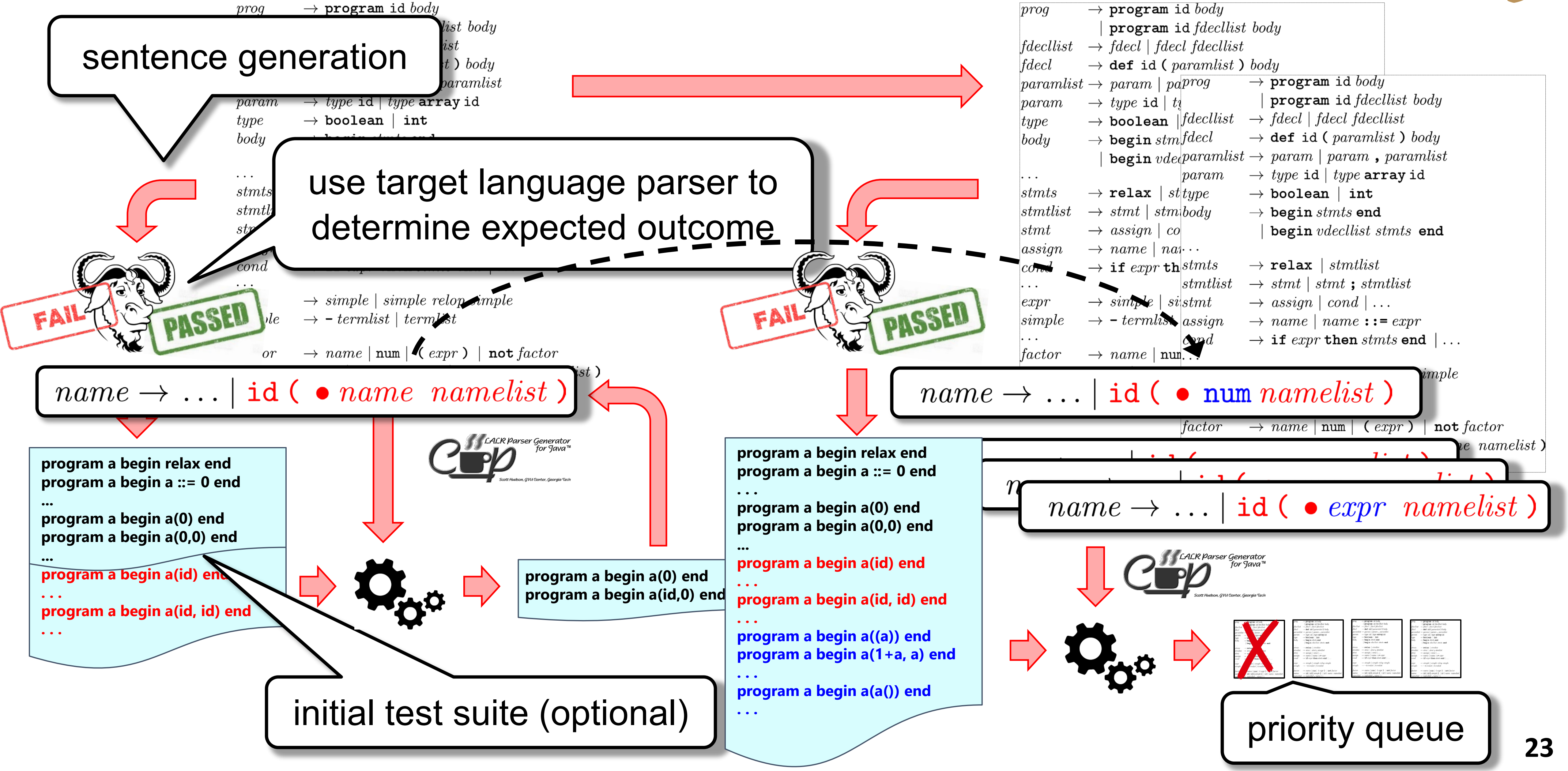
$name \rightarrow \dots \mid$ id (• *expr namelist*)

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
```

```
program a begin a(0) end
program a begin a(id,0) end
```

```
program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .
```

Active repair generates the test suites used for repair.



sentence generation
use target language parser to determine expected outcome
FAIL
PASSED
initial test suite (optional)
priority queue
CUP LALR Parser Generator for Java™

```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
...
factor    → name | num | ( expr ) | not factor
```

name → . . . | id (• name namelist)

name → . . . | id (• num namelist)

name → . . . | id (• expr namelist)

```
program a begin relax end
program a begin a ::= 0 end
...
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
```

```
program a begin a(0) end
program a begin a(id,0) end
```

```
program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .
```

Active repair generates the test suites used for repair.

```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
```

$\boxed{\textit{name} \rightarrow \ \ldots \ | \ \texttt{id}\ (\ \bullet\ \textit{expr}\ \ \textit{namelist}\ )}$

```
namelist  → namelist , name | ϵ
```

program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .

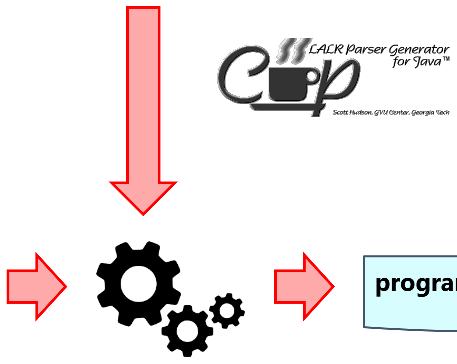
Active repair generates the test suites used for repair.

$$
\begin{array}{lll}
\textit{prog} & \to & \texttt{program id } \textit{body} \\
 & | & \texttt{program id } \textit{fdecllist body} \\
\textit{fdecllist} & \to & \textit{fdecl} \mid \textit{fdecl fdecllist} \\
\textit{fdecl} & \to & \texttt{def id (} \textit{ paramlist } \texttt{)} \textit{ body} \\
\textit{paramlist} & \to & \textit{param} \mid \textit{param} \texttt{ , } \textit{paramlist} \\
\textit{param} & \to & \textit{type} \texttt{ id} \mid \textit{type} \texttt{ array id} \\
\textit{type} & \to & \texttt{boolean} \mid \texttt{int} \\
\textit{body} & \to & \texttt{begin } \textit{stmts} \texttt{ end} \\
 & | & \texttt{begin } \textit{vdecllist stmts} \texttt{ end} \\
\ldots & & \\
\textit{stmts} & \to & \texttt{relax} \mid \textit{stmtlist} \\
\textit{stmtlist} & \to & \textit{stmt} \mid \textit{stmt} \texttt{ ; } \textit{stmtlist} \\
\textit{stmt} & \to & \textit{assign} \mid \textit{cond} \mid \ldots \\
\textit{assign} & \to & \textit{name} \mid \textit{name} \texttt{ ::= } \textit{expr} \\
\textit{cond} & \to & \texttt{if } \textit{expr} \texttt{ then } \textit{stmts} \texttt{ end} \mid \ldots \\
\ldots & & \\
\textit{expr} & \to & \textit{simple} \mid \textit{simple relop simple} \\
\textit{simple} & \to & \texttt{-} \textit{ termlist} \mid \textit{termlist} \\
\end{array}
$$

$$\boxed{\textit{name} \to \ldots \mid \texttt{id (} \bullet \textit{ expr} \;\; \textit{namelist} \texttt{)}}$$

$$\textit{namelist} \to \textit{namelist} \texttt{ , } \textit{name} \mid \epsilon$$

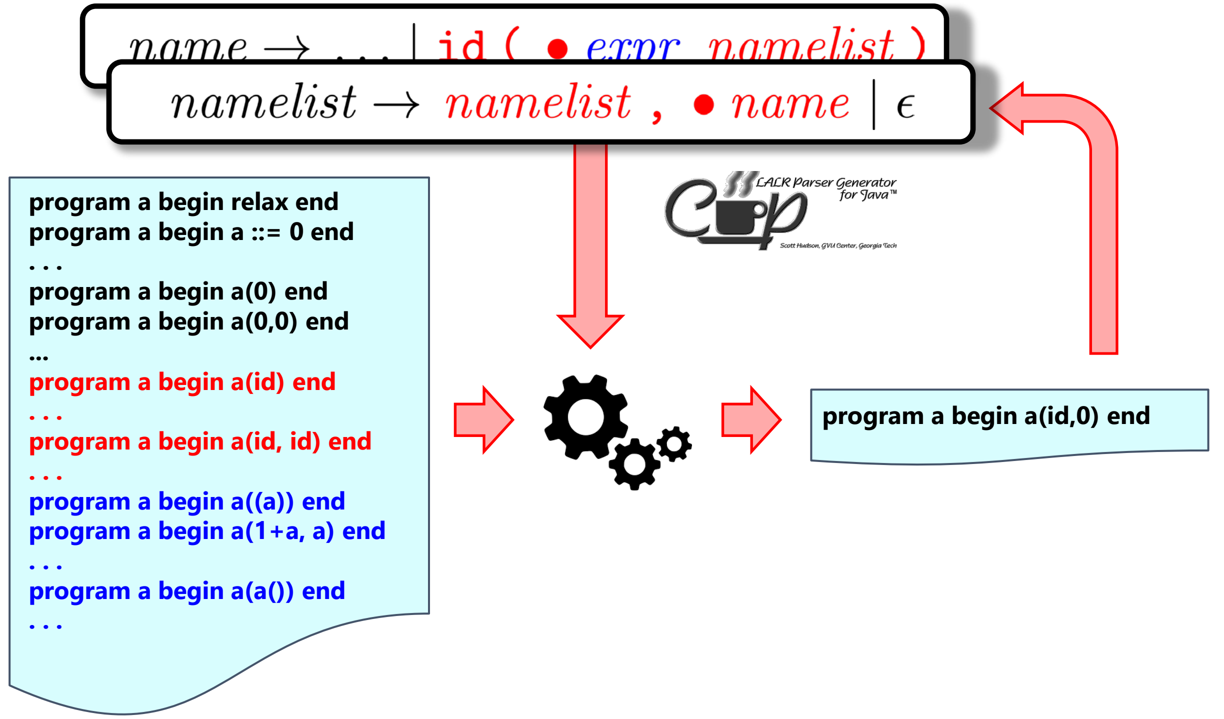
program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .


LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

program a begin a(id,0) end

Active repair generates the test suites used for repair.

```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
. . .
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | . . .
assign    → name | name ::= expr
cond      → if expr then stmts end | . . .
. . .
expr      → simple | simple relop simple
simple    → - termlist | termlist
```



LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

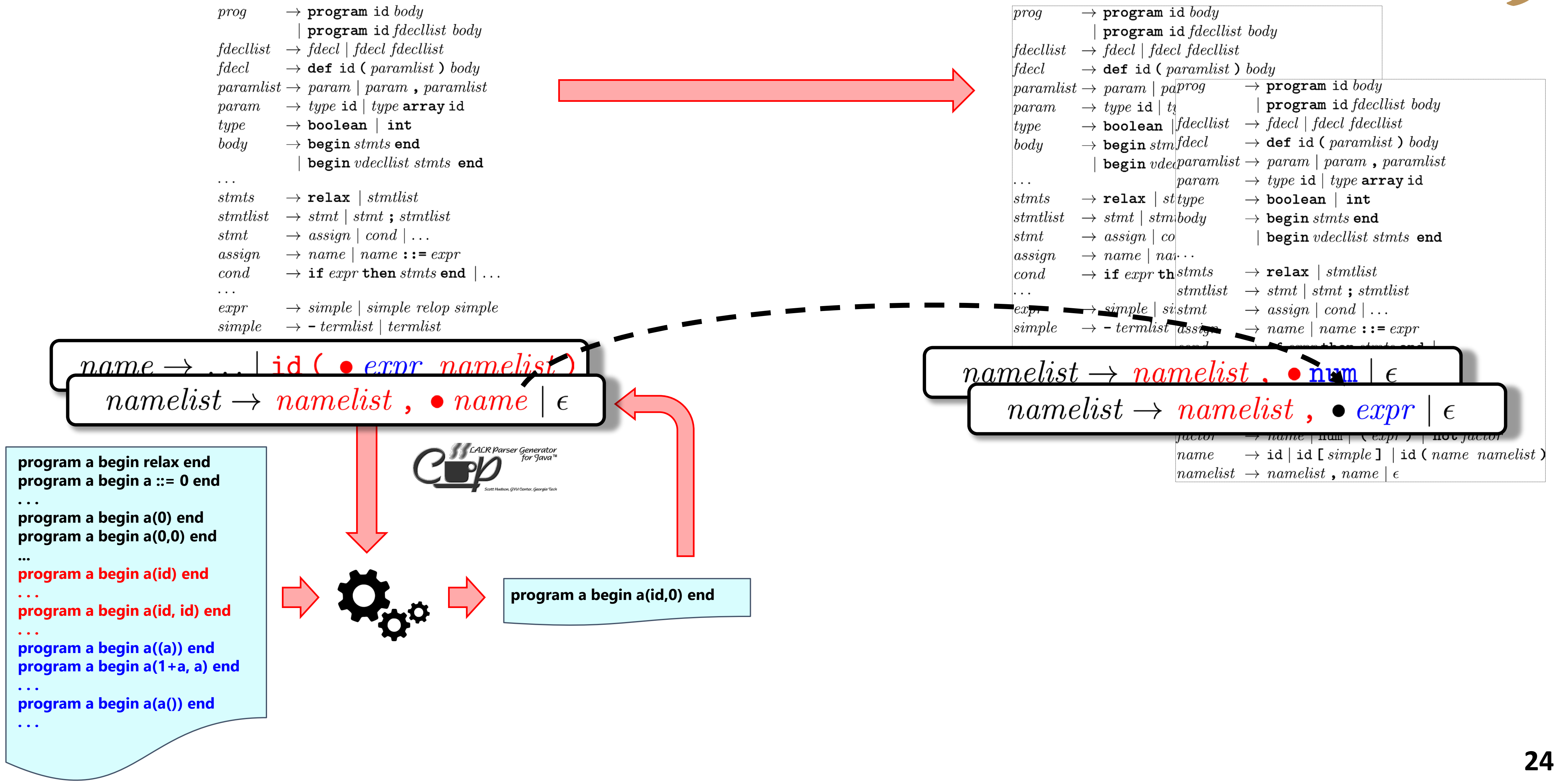
$name \rightarrow \ldots \mid \text{id} \; (\; \bullet \; expr \;\; namelist \;)$

$namelist \rightarrow namelist \; , \; \bullet \; name \mid \epsilon$

program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .

program a begin a(id,0) end

Active repair generates the test suites used for repair.



```
prog       → program id body
           | program id fdecllist body
fdecllist  → fdecl | fdecl fdecllist
fdecl      → def id ( paramlist ) body
paramlist  → param | param , paramlist
param      → type id | type array id
type       → boolean | int
body       → begin stmts end
           | begin vdecllist stmts end
. . .
stmts      → relax | stmtlist
stmtlist   → stmt | stmt ; stmtlist
stmt       → assign | cond | . . .
assign     → name | name ::= expr
cond       → if expr then stmts end | . . .
. . .
expr       → simple | simple relop simple
simple     → - termlist | termlist
```

$name \rightarrow \ldots \mid$ id (● $expr$ $namelist$)

$namelist \rightarrow namelist$, ● $name \mid \epsilon$

LALR Parser Generator for Java™

program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .

program a begin a(id,0) end

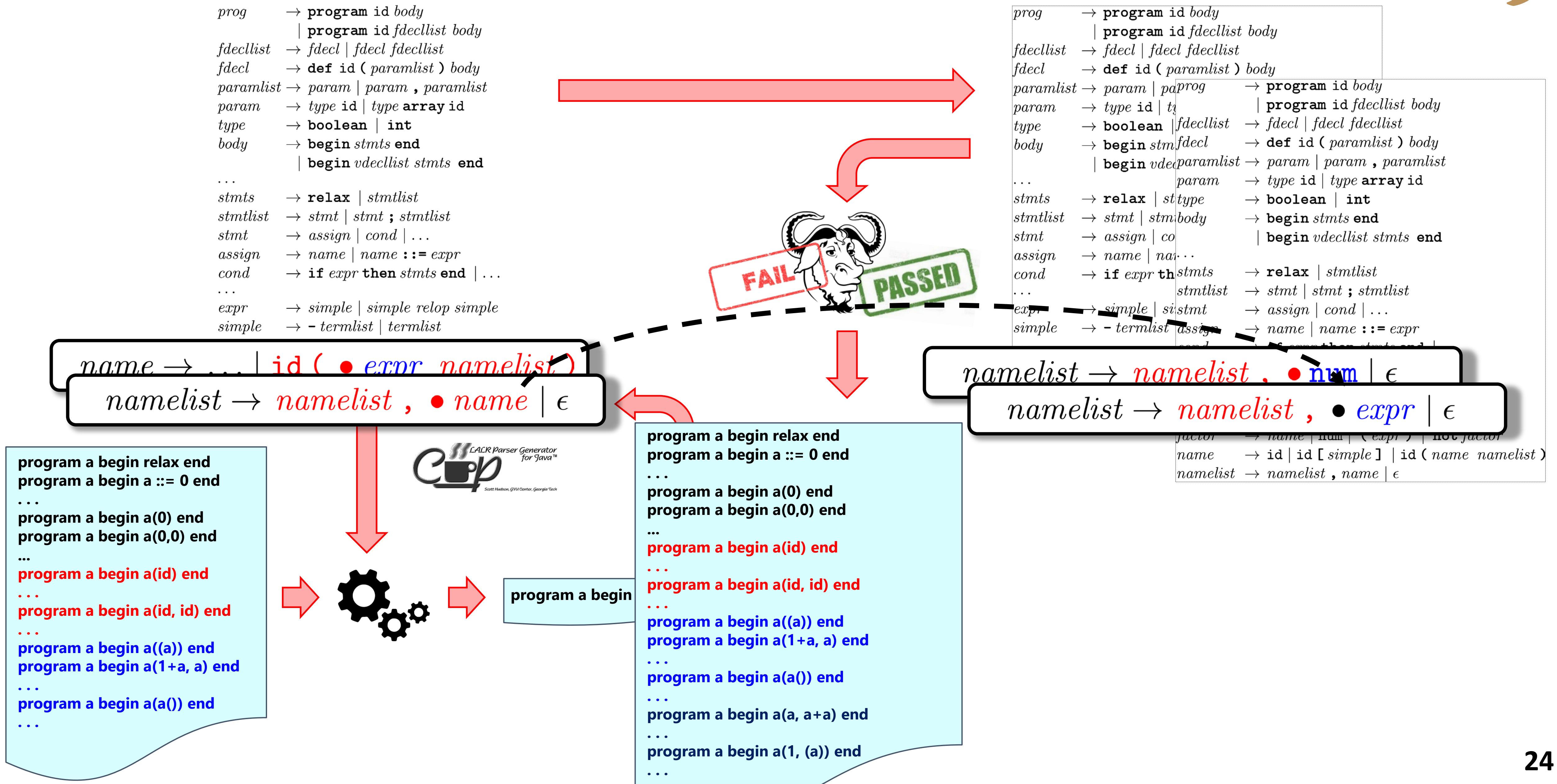
```
prog       → program id body
           | program id fdecllist body
fdecllist  → fdecl | fdecl fdecllist
fdecl      → def id ( paramlist ) body
paramlist  → param | pa
param      → type id | ty
type       → boolean |
body       → begin stm
           | begin vde
. . .
stmts      → relax | st
stmtlist   → stmt | stm
stmt       → assign | co
assign     → name | na
cond       → if expr th
. . .
expr       → simple | si
simple     → - termlist
```

```
prog       → program id body
           | program id fdecllist body
fdecllist  → fdecl | fdecl fdecllist
fdecl      → def id ( paramlist ) body
paramlist  → param | param , paramlist
param      → type id | type array id
type       → boolean | int
body       → begin stmts end
           | begin vdecllist stmts end
. . .
stmts      → relax | stmtlist
stmtlist   → stmt | stmt ; stmtlist
stmt       → assign | cond | . . .
assign     → name | name ::= expr
. . .
factor     → name | num | ( expr ) | not factor
name       → id | id [ simple ] | id ( name namelist )
namelist   → namelist , name | ε
```

$namelist \rightarrow namelist$, ● num $\mid \epsilon$

$namelist \rightarrow namelist$, ● $expr \mid \epsilon$

Active repair generates the test suites used for repair.



```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
```

$name \rightarrow \ldots \mid id \; (\; \bullet \; expr \;\; namelist \;)$

$namelist \rightarrow namelist \; , \; \bullet \; name \mid \epsilon$

```
program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(()) end
. . .
```

FAIL
PASSED
LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

```
program a begin
```

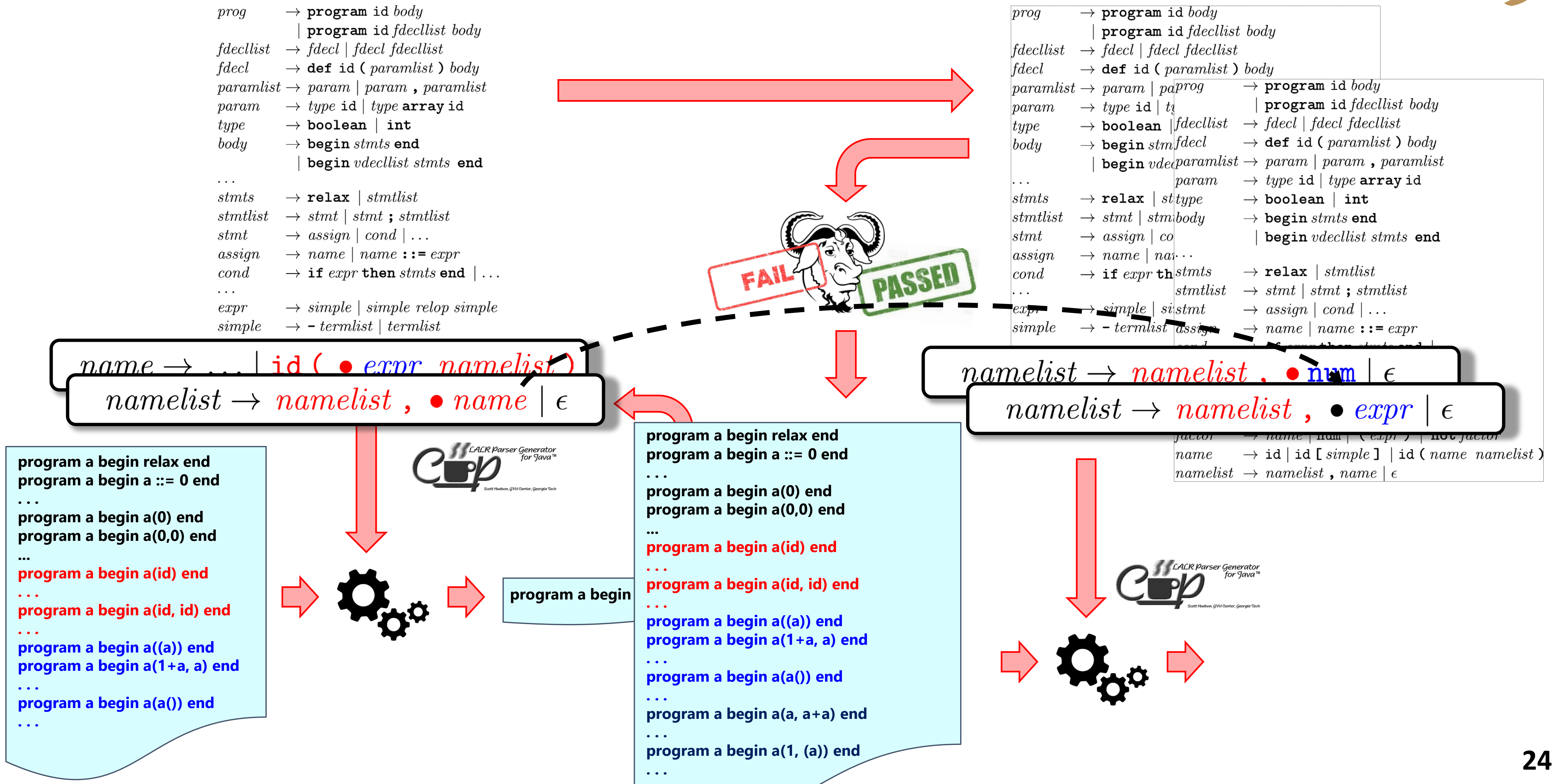
```
program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(()) end
. . .
program a begin a(a, a+a) end
. . .
program a begin a(1, (a)) end
. . .
```

```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
...
name      → id | id [ simple ] | id ( name  namelist )
namelist  → namelist , name | ϵ
```

$namelist \rightarrow namelist \; , \; \bullet \; num \mid \epsilon$

$namelist \rightarrow namelist \; , \; \bullet \; expr \mid \epsilon$

Active repair generates the test suites used for repair.



```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
. . .
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | . . .
assign    → name | name ::= expr
cond      → if expr then stmts end | . . .
. . .
expr      → simple | simple relop simple
simple    → - termlist | termlist
```

$name \to \ldots \mid$ id ($\bullet$ $expr$ $namelist$)

$namelist \to namelist$, $\bullet$ $name \mid \epsilon$

program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(()) end
. . .

program a begin

program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .
program a begin a(a, a+a) end
. . .
program a begin a(1, (a)) end
. . .

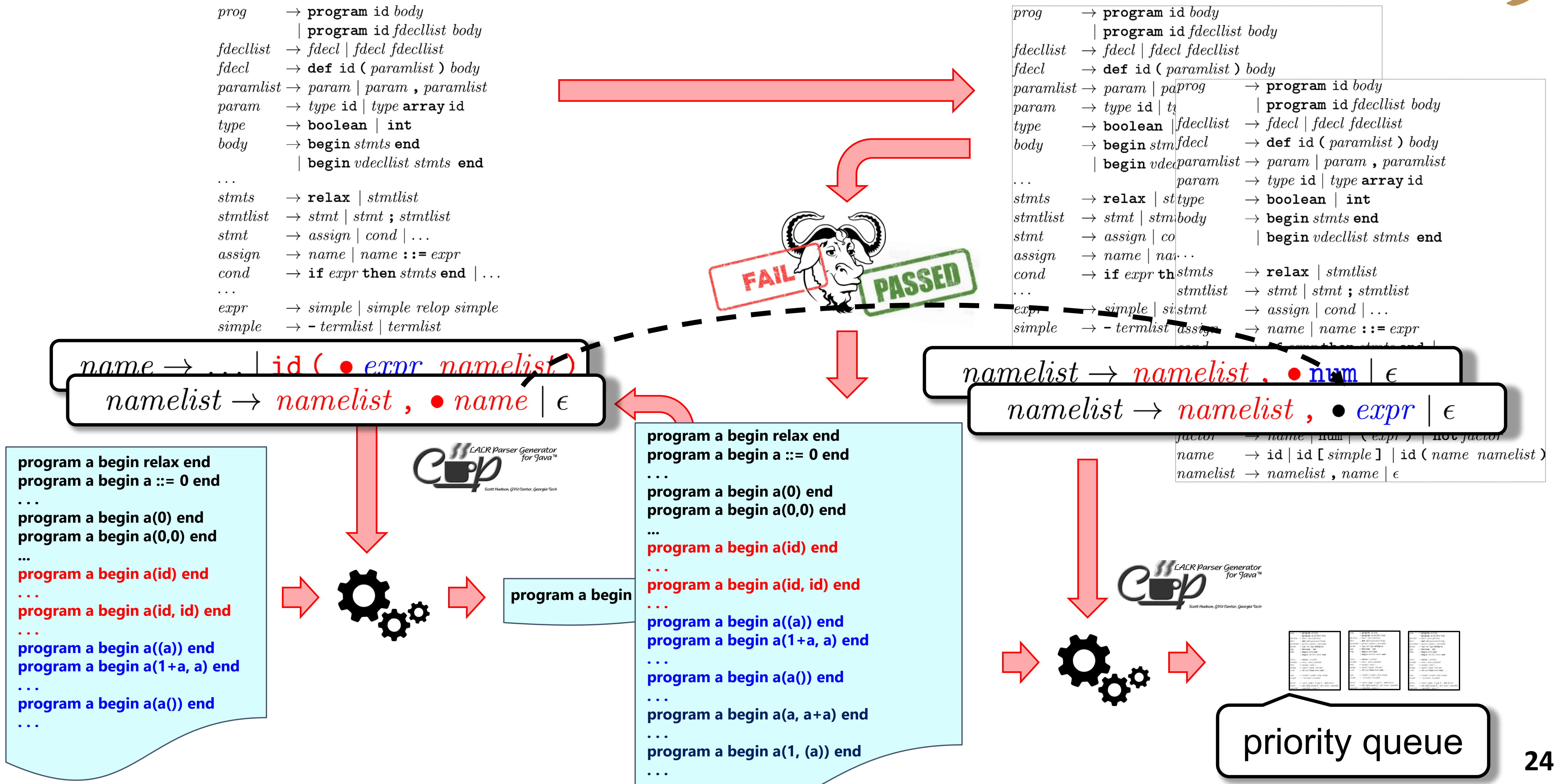
```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | pa
param     → type id | ty
type      → boolean |
body      → begin stm
          | begin vdec
. . .
stmts     → relax | st
stmtlist  → stmt | stm
stmt      → assign | co
assign    → name | na
cond      → if expr th
. . .
expr      → simple | si
simple    → - termlist
```

```
prog      → program id body
          | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
          | begin vdecllist stmts end
. . .
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | . . .
assign    → name | name ::= expr
. . .
name      → id | id [ simple ] | id ( name namelist )
namelist  → namelist , name | ε
```

$namelist \to namelist$, $\bullet$ num $\mid \epsilon$

$namelist \to namelist$, $\bullet$ $expr \mid \epsilon$

Active repair generates the test suites used for repair.



```
prog       → program id body
            | program id fdecllist body
fdecllist  → fdecl | fdecl fdecllist
fdecl      → def id ( paramlist ) body
paramlist  → param | param , paramlist
param      → type id | type array id
type       → boolean | int
body       → begin stmts end
            | begin vdecllist stmts end
...
stmts      → relax | stmtlist
stmtlist   → stmt | stmt ; stmtlist
stmt       → assign | cond | ...
assign     → name | name ::= expr
cond       → if expr then stmts end | ...
...
expr       → simple | simple relop simple
simple     → - termlist | termlist
```

$name \rightarrow \ldots \mid$ id ($\bullet$ $expr$ $namelist$)

$namelist \rightarrow namelist$, $\bullet$ $name \mid \epsilon$

$namelist \rightarrow namelist$, $\bullet$ num $\mid \epsilon$

$namelist \rightarrow namelist$, $\bullet$ $expr \mid \epsilon$

```
factor     → name | num | ( expr ) | not factor
name       → id | id [ simple ] | id ( name namelist )
namelist   → namelist , name | ε
```

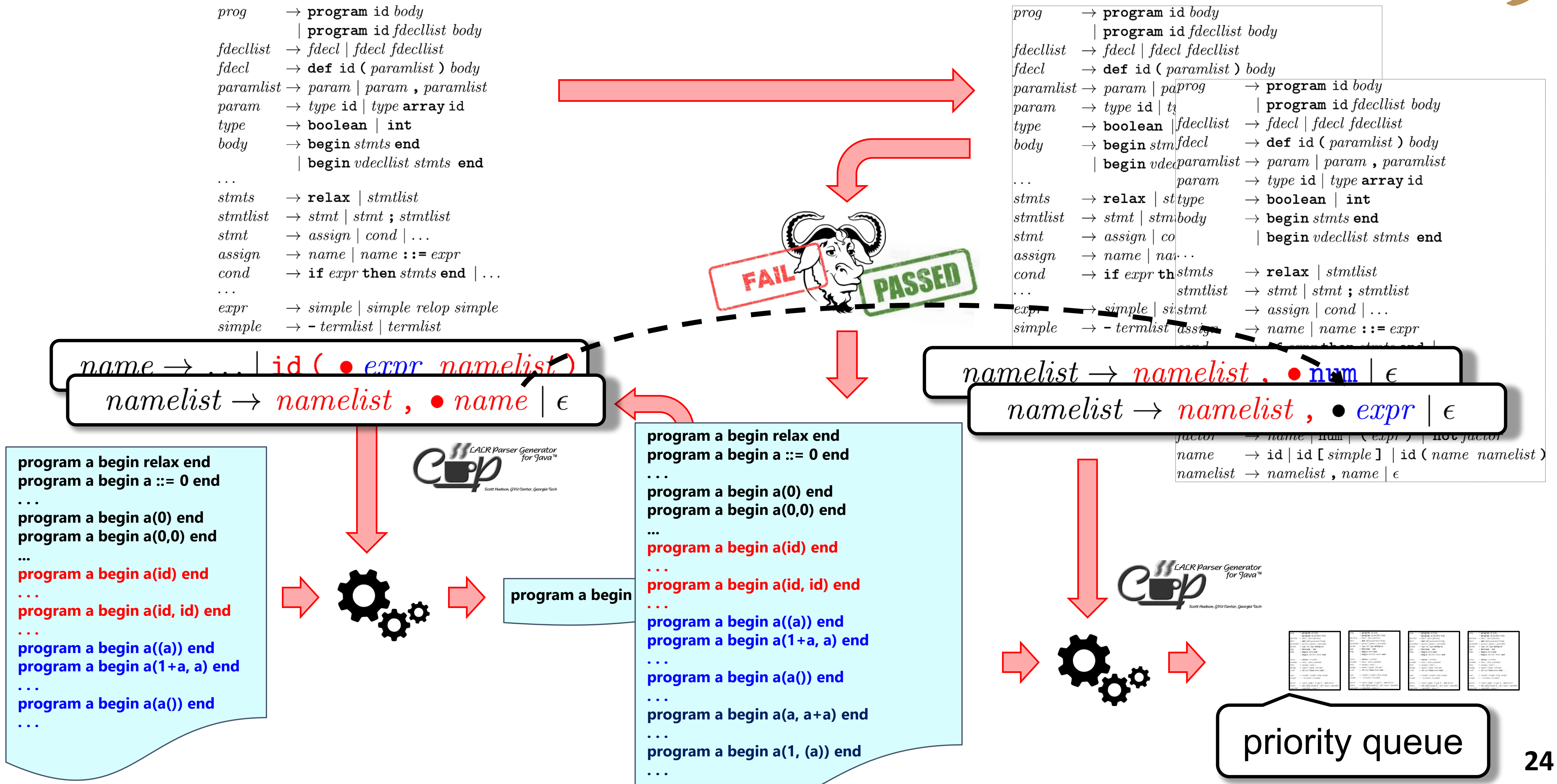
program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .

program a begin

program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .
program a begin a(a, a+a) end
. . .
program a begin a(1, (a)) end
. . .

FAIL
PASSED
CUP LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech
priority queue

Active repair generates the test suites used for repair.


FAIL
PASSED
LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech
LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech
priority queue

```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
cond      → if expr then stmts end | ...
...
expr      → simple | simple relop simple
simple    → - termlist | termlist
```

$name \rightarrow \ldots \mid \text{id} \; (\; \bullet \; expr \;\; namelist \;)$

$namelist \rightarrow namelist \; , \; \bullet \; name \mid \epsilon$

$namelist \rightarrow namelist \; , \; \bullet \; \text{num} \mid \epsilon$

$namelist \rightarrow namelist \; , \; \bullet \; expr \mid \epsilon$

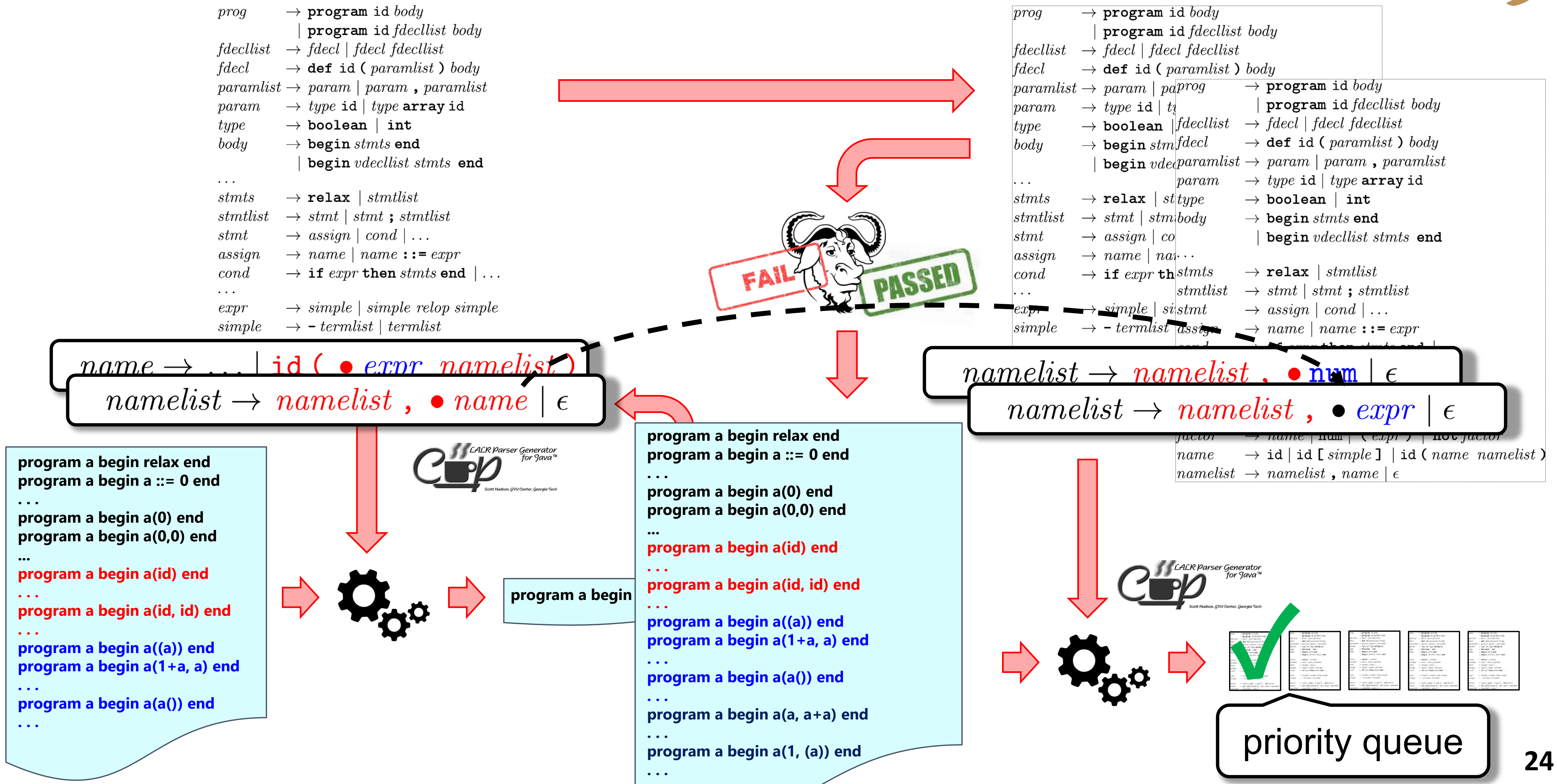
```
prog      → program id body
            | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
            | begin vdecllist stmts end
...
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | ...
assign    → name | name ::= expr
...
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( name namelist )
namelist  → namelist , name | ε
```

program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(()) end
. . .

program a begin

program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .
program a begin a(a, a+a) end
. . .
program a begin a(1, (a)) end
. . .

Active repair generates the test suites used for repair.



```
prog      → program id body
           | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
           | begin vdecllist stmts end
. . .
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | . . .
assign    → name | name ::= expr
cond      → if expr then stmts end | . . .
. . .
expr      → simple | simple relop simple
simple    → - termlist | termlist
```

$name \rightarrow \ldots \mid id\ (\ \bullet\ expr\ \ namelist\ )$

$namelist \rightarrow namelist\ ,\ \bullet\ name \mid \epsilon$

FAIL
PASSED
LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(()) end
. . .

program a begin

program a begin relax end
program a begin a ::= 0 end
. . .
program a begin a(0) end
program a begin a(0,0) end
...
program a begin a(id) end
. . .
program a begin a(id, id) end
. . .
program a begin a((a)) end
program a begin a(1+a, a) end
. . .
program a begin a(a()) end
. . .
program a begin a(a, a+a) end
. . .
program a begin a(1, (a)) end
. . .

$namelist \rightarrow namelist\ ,\ \bullet\ num \mid \epsilon$

$namelist \rightarrow namelist\ ,\ \bullet\ expr \mid \epsilon$

```
prog      → program id body
           | program id fdecllist body
fdecllist → fdecl | fdecl fdecllist
fdecl     → def id ( paramlist ) body
paramlist → param | param , paramlist
param     → type id | type array id
type      → boolean | int
body      → begin stmts end
           | begin vdecllist stmts end
. . .
stmts     → relax | stmtlist
stmtlist  → stmt | stmt ; stmtlist
stmt      → assign | cond | . . .
assign    → name | name ::= expr
factor    → name | num | ( expr ) | not factor
name      → id | id [ simple ] | id ( name  namelist )
namelist  → namelist , name | ϵ
```

LALR Parser Generator for Java™
Scott Hudson, GVU Center, Georgia Tech

priority queue

Grammar Repair Evaluation

Goals:

- evaluate effectiveness
- compare passive repair and active repair flavours

Evaluation subjects:

- 33 medium-sized grammars containing real and multiple faults

Test suites:

- cdrc test suite generated from target grammar as specification
- cdrc test suite generated from each candidate patch in case of active repair
- validation against much stronger test suites (including random and negative)

Grammar Repair Evaluation

Evaluation metrics:

- recall: do the patches generalize to unseen tests?
- precision: how closely do the patches approximate the target?

Grammar Repair Evaluation

Evaluation metrics:

- recall: do the patches generalize to unseen tests?
- precision: how closely do the patches approximate the target?

prog → **program** id **=** block **.**
block → **{** (decl **;**)∗ (stmt **;**)∗ **}**
decl → **var id :** type
type → **bool** | **int**
stmt → **sleep**
| **if** expr **then** stmt (**else** stmt)?
| **while** expr **do** block
| **id =** expr | block
expr → expr **=** expr | expr **+** expr
| (expr **)** | **id** | **num**

patched grammar G’

prog → **program** id **=** block **.**
block → **{** (decl **;**)∗ (stmt **;**)∗ **}**
decl → **var id :** type
type → **bool** | **int**
stmt → **sleep**
| **if** expr **then** stmt (**else** stmt)?
| **while** expr **do** stmt
| **id =** expr | block
expr → expr **=** expr | expr **+** expr
| (expr **)** | **id** | **num**

target grammar G_T

Grammar Repair Evaluation

Evaluation metrics:

- recall: do the patches generalize to unseen tests?
- precision: how closely do the patches approximate the target?

generate parser

```
prog   → program id = block .
block  → { (decl ;)* (stmt ;)* }
decl   → var id : type
type   → bool | int
stmt   → sleep
         | if expr then stmt (else stmt)?
         | while expr do block
         | id = expr | block
expr   → expr = expr | expr + expr
         | (expr ) | id | num
```

patched grammar G'

recall

$$\frac{|L(G') \cap T(G_T)|}{|T(G_T)|}$$

generate tests

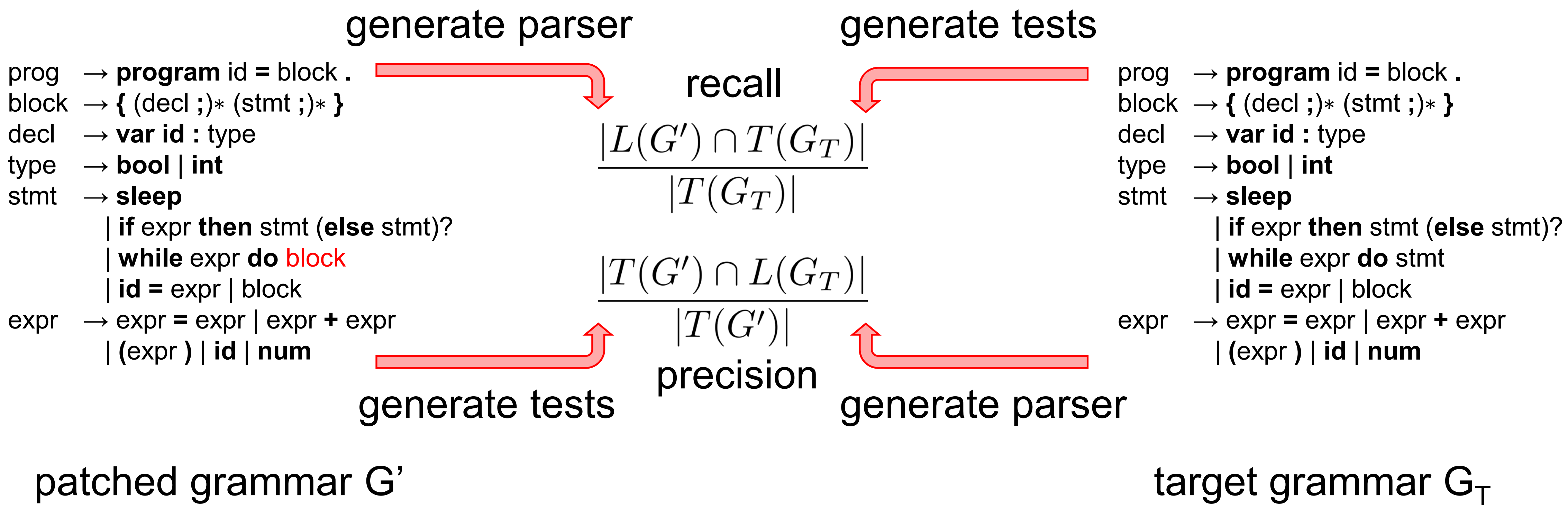
```
prog   → program id = block .
block  → { (decl ;)* (stmt ;)* }
decl   → var id : type
type   → bool | int
stmt   → sleep
         | if expr then stmt (else stmt)?
         | while expr do stmt
         | id = expr | block
expr   → expr = expr | expr + expr
         | (expr ) | id | num
```

target grammar G_T

Grammar Repair Evaluation

Evaluation metrics:

- recall: do the patches generalize to unseen tests?
- precision: how closely do the patches approximate the target?



```
prog   → program id = block .
block  → { (decl ;)∗ (stmt ;)∗ }
decl   → var id : type
type   → bool | int
stmt   → sleep
         | if expr then stmt (else stmt)?
         | while expr do block
         | id = expr | block
expr   → expr = expr | expr + expr
         | (expr ) | id | num
```

recall

$$\frac{|L(G') \cap T(G_T)|}{|T(G_T)|}$$

$$\frac{|T(G') \cap L(G_T)|}{|T(G')|}$$

precision

```
prog   → program id = block .
block  → { (decl ;)∗ (stmt ;)∗ }
decl   → var id : type
type   → bool | int
stmt   → sleep
         | if expr then stmt (else stmt)?
         | while expr do stmt
         | id = expr | block
expr   → expr = expr | expr + expr
         | (expr ) | id | num
```

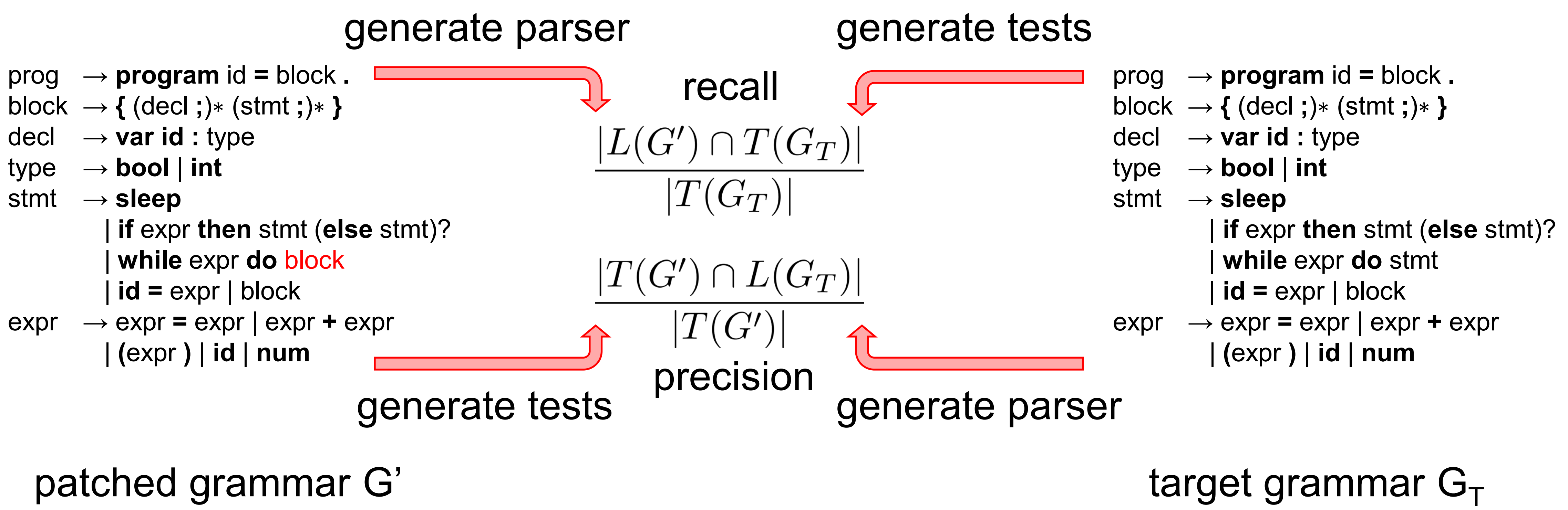
patched grammar G’

target grammar G_T

Grammar Repair Evaluation

Evaluation metrics:

- recall: do the patches generalize to unseen tests?
- precision: how closely do the patches approximate the target?
- F1 score: combined measure of the quality



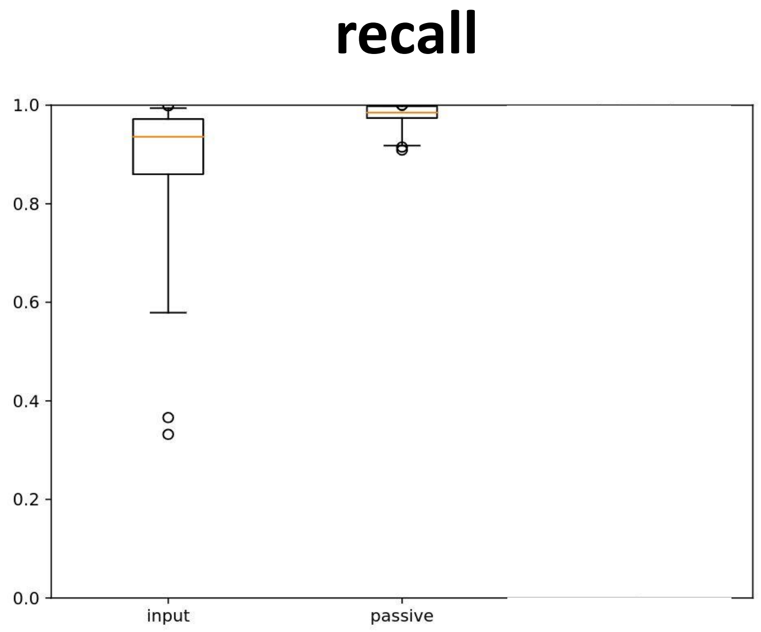
```
prog    → program id = block .
block   → { (decl ;)* (stmt ;)* }
decl    → var id : type
type    → bool | int
stmt    → sleep
          | if expr then stmt (else stmt)?
          | while expr do block
          | id = expr | block
expr    → expr = expr | expr + expr
          | (expr ) | id | num
```

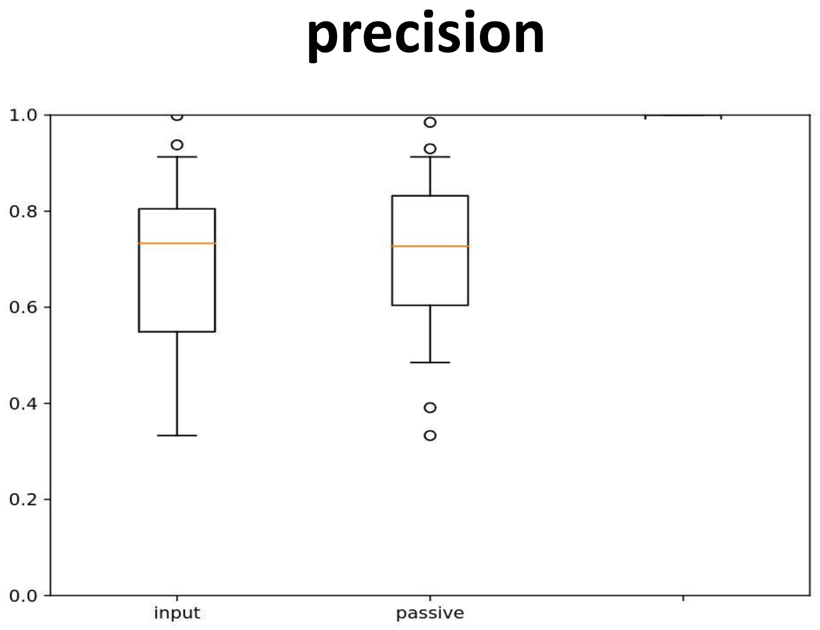
$$\frac{|L(G') \cap T(G_T)|}{|T(G_T)|}$$

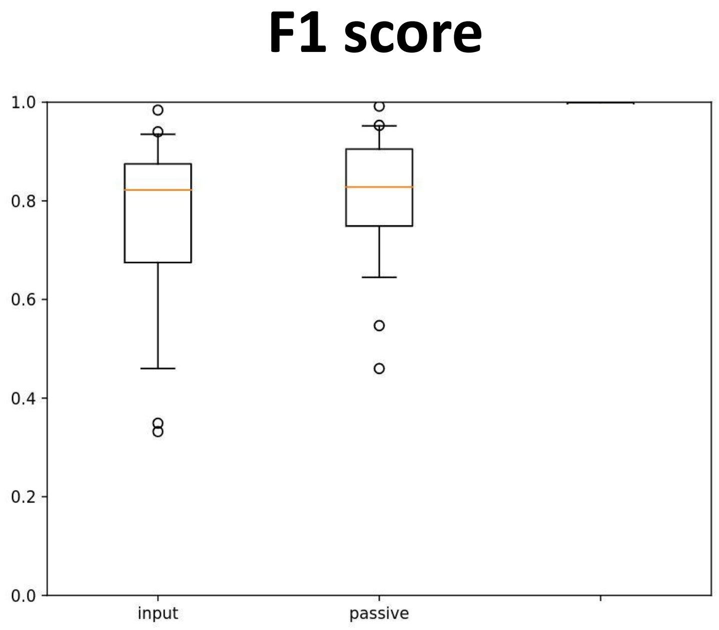
$$\frac{|T(G') \cap L(G_T)|}{|T(G')|}$$

```
prog    → program id = block .
block   → { (decl ;)* (stmt ;)* }
decl    → var id : type
type    → bool | int
stmt    → sleep
          | if expr then stmt (else stmt)?
          | while expr do stmt
          | id = expr | block
expr    → expr = expr | expr + expr
          | (expr ) | id | num
```

Passive repair is effective.

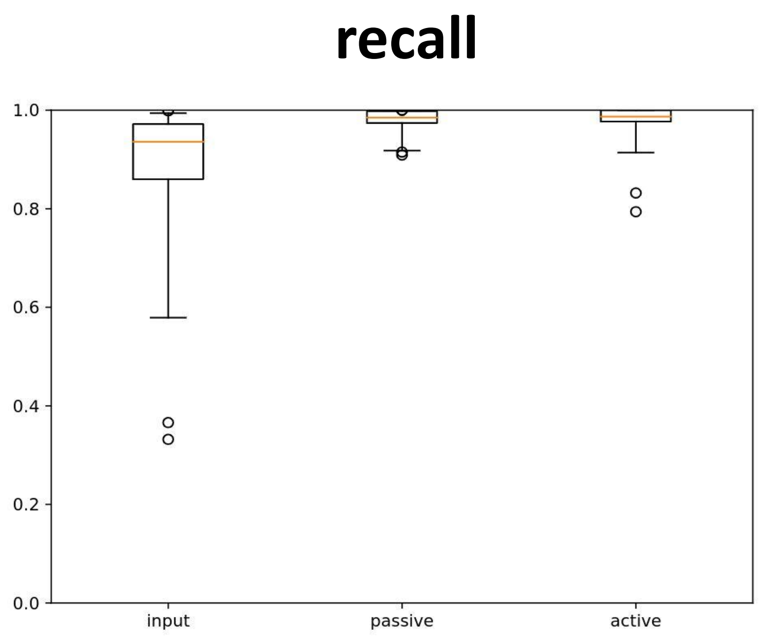


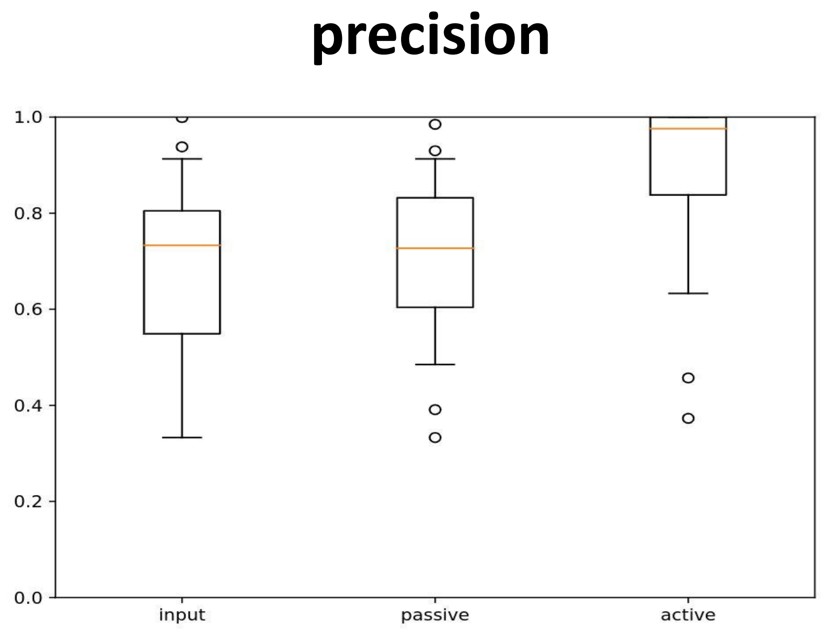


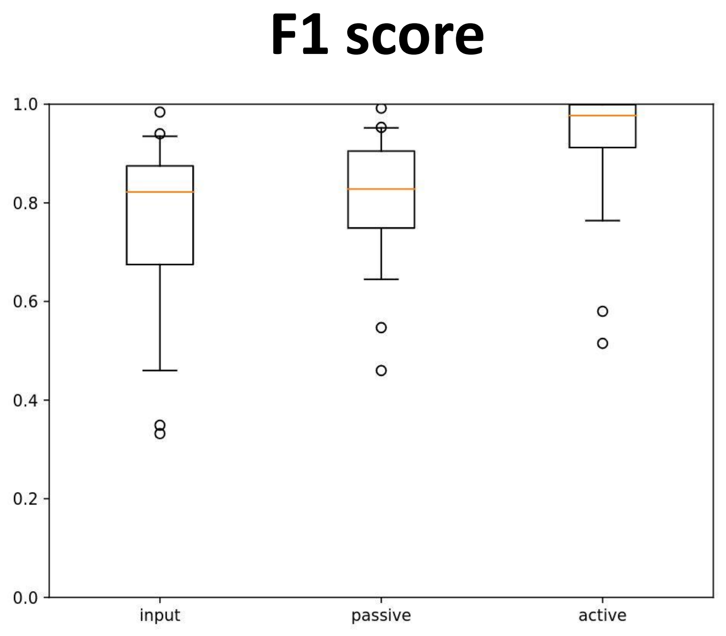


- 25 full repairs; 4 partial repairs;
  4 grammars unrepairable with input test suite (no failing tests)
- fixes generalize to new unseen tests (improvements in recall)
- minor improvements in precision; sometimes drops

Active repair is slower but much better!







- 27 full repairs; 6 partial repairs
- tighter patches
- mostly high F1 scores
- many perfect patches (100% F1)

Conclusions

Conclusion #1: SBFL can find bugs in grammars.

- works at two levels of granularity
- ranks seeded faults on average in 15%-25% of rules; pinpoints in 10%-40% of cases
- can handle real and multiple bugs

Conclusion #2: Automatic grammar repair is possible.

- relies on item-level fault localization
- use small-scale transformations with explicit pre- and post-conditions as ***patches***
- implemented and evaluated two grammar repair approaches
- successfully repaired student grammars against test suites

Future Work

- migrate to modern compiler-compiler tools
- investigate multiple-bugs-at-a-time approach
- generate-localize-repair approach for grammar mining

Future Work

- migrate to modern compiler-compiler tools
- investigate multiple-bugs-at-a-time approach
- generate-localize-repair approach for grammar mining

Thank you!