Grammar Comparison Techniques

Vadim Zaytsev, SWAT, CWI

Research Colloquium @ SERG

2011

Questions?

What this talk is not about

- (Un)decidability of grammar equivalence
- Model synchronisation
- Tree diffing
- Graph diffing

Claims we all hear

- “This compiler implements that language”
- “This appendix contains an [*insert parsing technique here*] optimised grammar of the language”
- “This grammarware produces data suitable to use with that grammarware”
- “These are 100 implementations of one language”
- “This language is a subset/superset of that language”
- “This version of a compiler is backward compatible”

Grammar differences

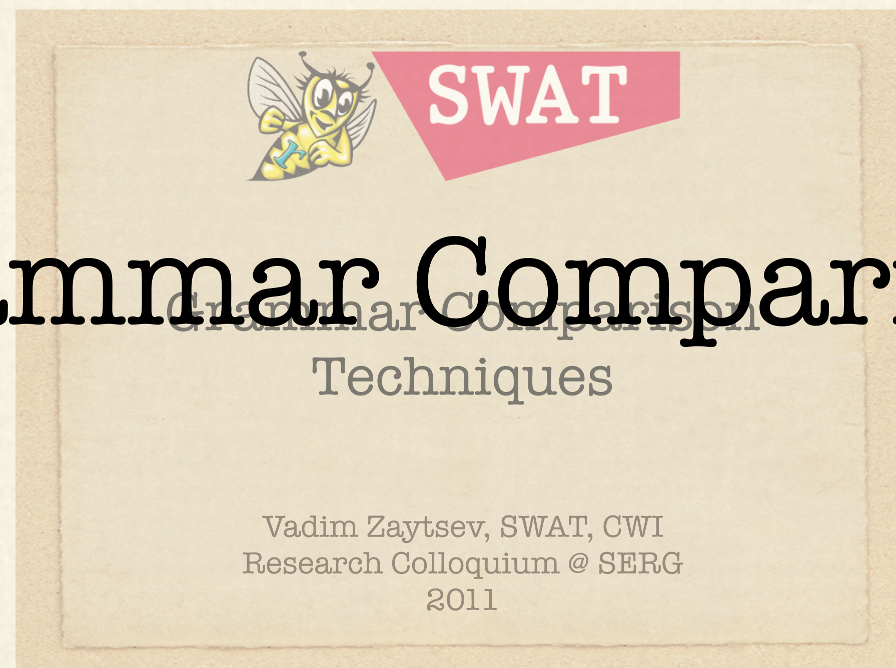
- ★ intended vs. accidental
- ★ result of grammar adaptation
- ★ result of grammar evolution
- ★ idiosyncrasies thanks to metanotation
- ★ idiosyncrasies thanks to parsing technology
- ★ presentation and understandability
- ★ misspelling
- ★ …etc

Grammar (non)equivalence

- Undecidable
- Non-existent
- Can we cheat?
- Approach (non)equivalence
- Establish and maintain grammar relationships
- Consistence throughout language incarnations!

Conventions

Language Comparison



Grammar Comparison

Assumed infrastructure

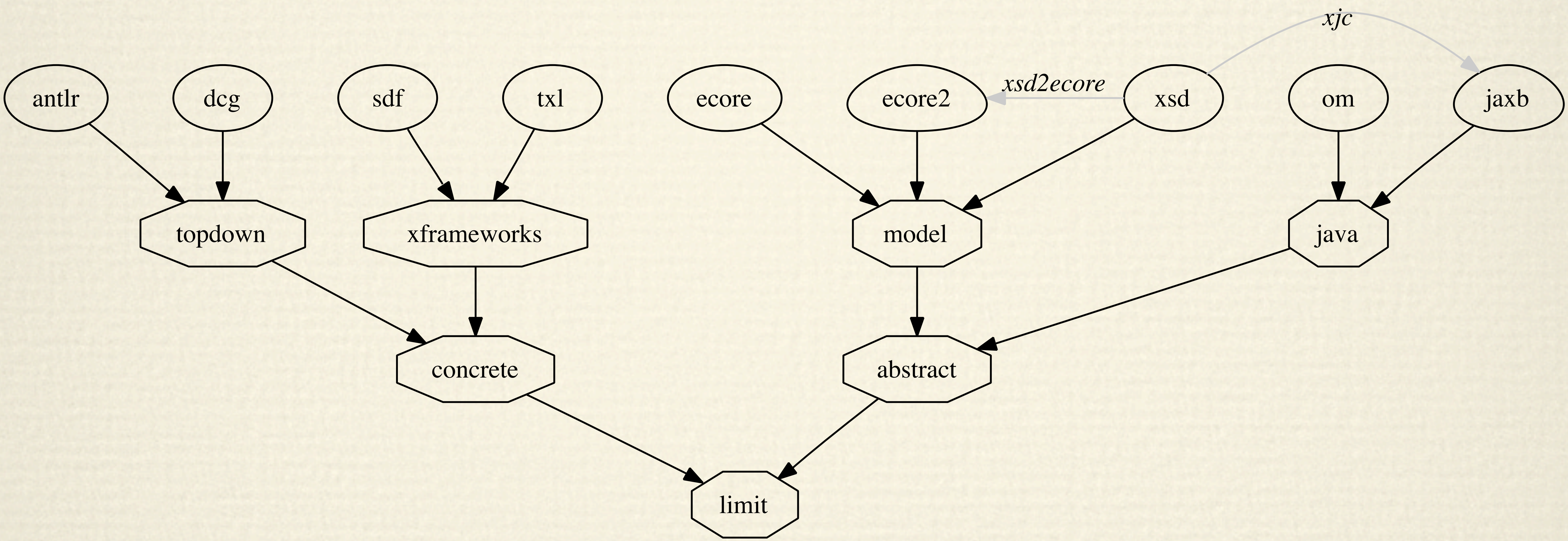
- Grammar *format* free from idiosyncrasies
- (Parse) *tree format* containing productions & literals
- Grammar *extraction* for notation mapping:
  - Abstraction from predicates, sem.actions, rew.rules
- Grammar *comparison* for spotting grammar differences
  - Nominal differences; structural differences
- Grammar *transformation* suite:
  - Refactoring; extension / restriction; revision
- Grammar *export* for connecting to different toolsets

Straightforward comparison

- Equivalence as equality
- Nominal differences
  - `A ::= X Y Z;` `B ::= X Y Z;`
- Structural differences
  - `A ::= X Y Z;` `A ::= X   Z;`
- Deliberately limited comparator is useful

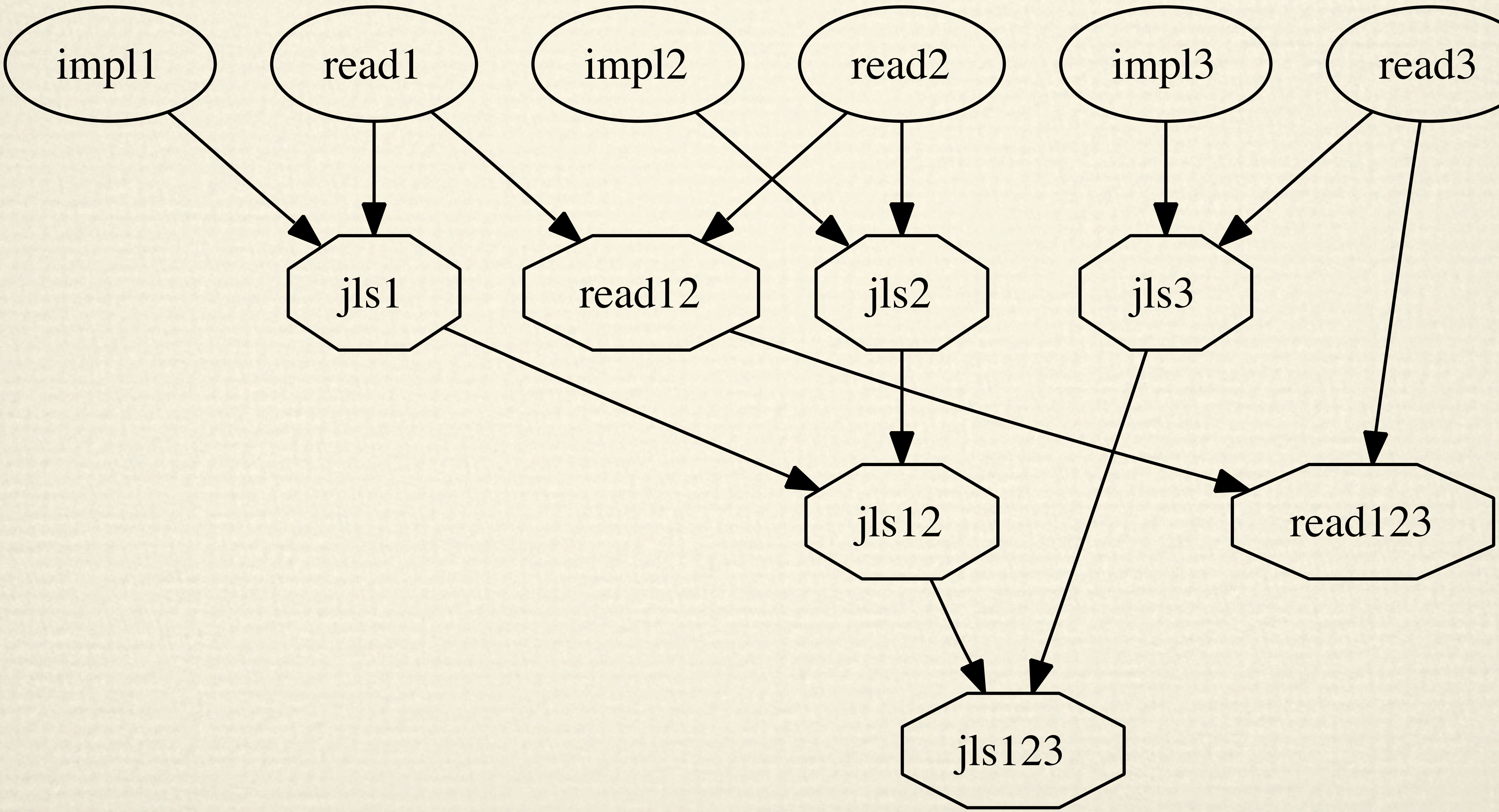
Complicated scenario

Different implementations of the same language
(parsers, data models, etc.)

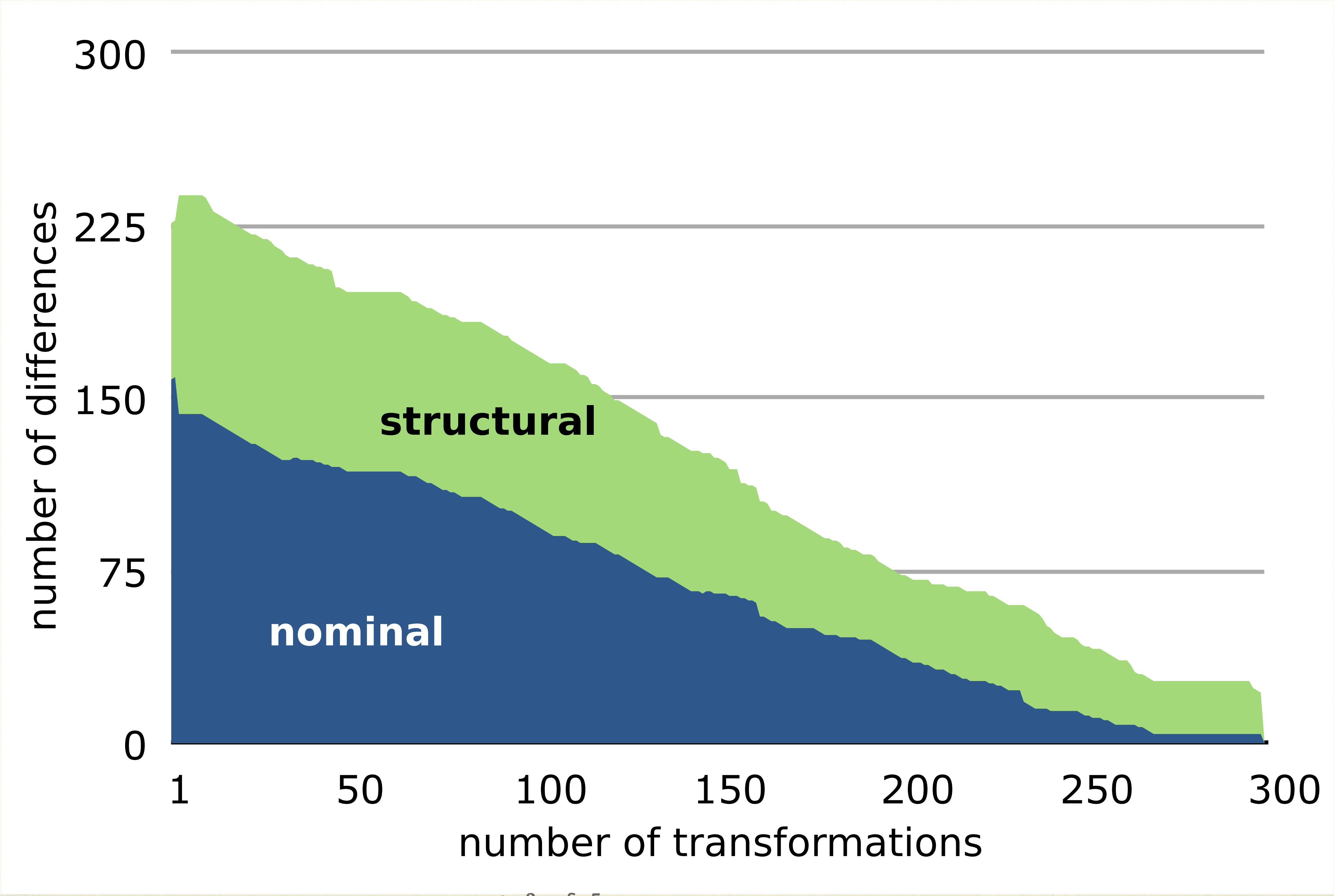


Alternative scenario

Different versions of a language as documented by specifications



Transform until equal



Grammar refactoring example

BGF (*read2*)

```
ClassBodyDeclarations:
    ClassBodyDeclaration
ClassBodyDeclarations:
    ClassBodyDeclarations ClassBodyDeclaration
ClassBody:
    "{" ClassBodyDeclarations? "}"
```

```
ClassBody:
    "{" ClassBodyDeclaration* "}"
```

XBGF (*grammar refactoring*)

deyaccify(ClassBodyDeclarations);
inline(ClassBodyDeclarations);
massage(
 ClassBodyDeclaration+? ,
 ClassBodyDeclaration*);

Grammar extension example

BGF (*read2*)

```
ClassModifier:
    "public"
    "protected"
    "private"
    "abstract"
    "static"
    "final"
    "strictfp"
```

```
FieldModifier:
    "public"
    "protected"
    "private"
    "static"
    "final"
    "transient"
    "volatile"
```

```
MethodModifier:
    "public"
    "protected"
    "private"
    "abstract"
    "static"
    "final"
    "synchronized"
    "native"
    "strictfp"
```

XBGF (grammar optimisation)

unite(InterfaceModifier, Modifier);
unite(ConstructorModifier, Modifier);
unite(MethodModifier, Modifier);
unite(FieldModifier, Modifier);
... ...

Grammar revision example

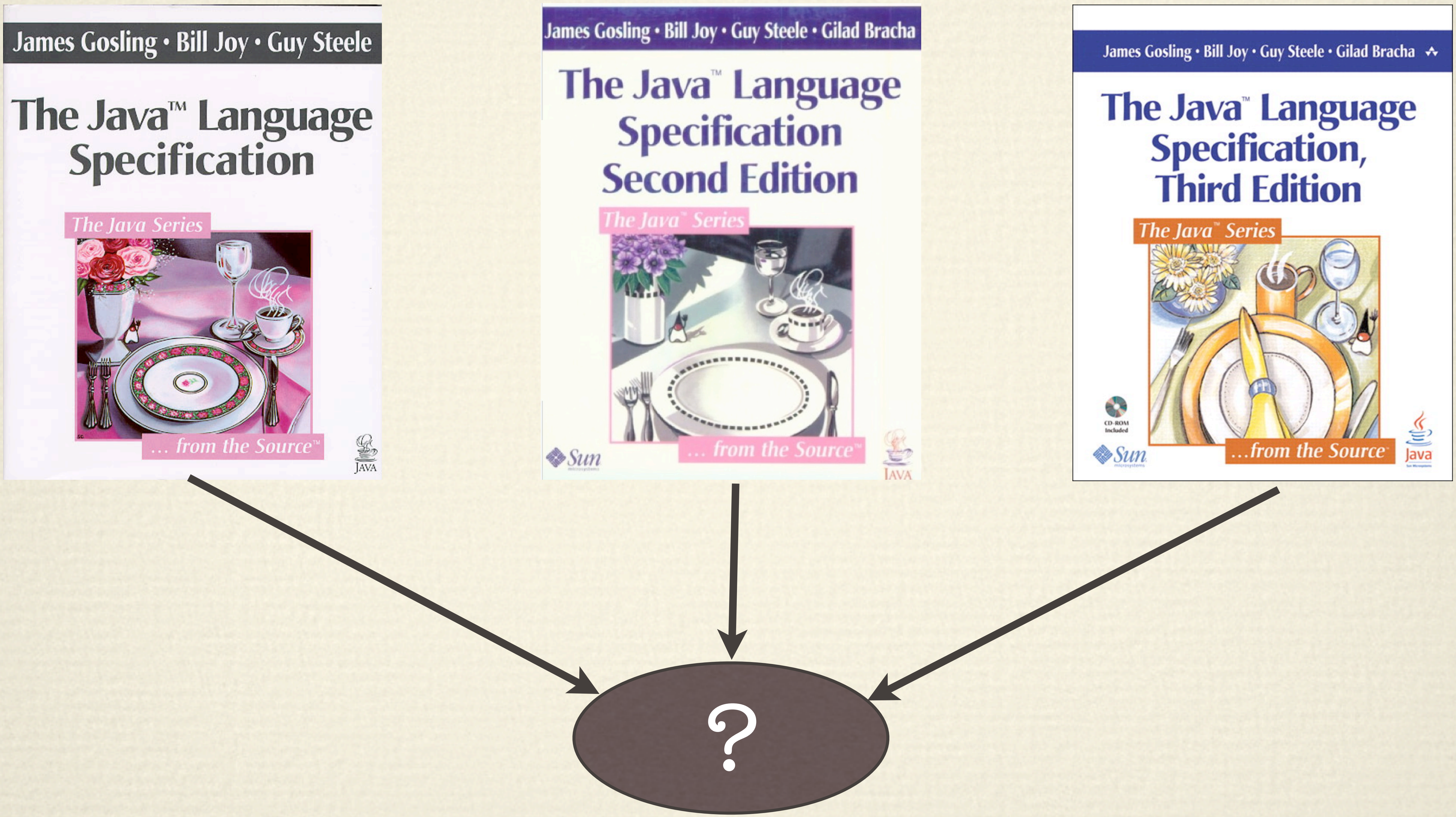
BGF (*impl2*, *impl3*)

Expression2:
  Expression3 Expression2Rest ?
Expression2Rest:
  (Infixop Expression3) *
Expression2Rest:
  ~~Expression3~~ "instanceof" Type

XBGF (*grammar correction*)

project(
 Expression2Rest:
  < Expression3 > "instanceof" Type
);

Case study: JLS



JLS convergence results

	jls1	jls12	jls123	jls2	jls3	read12	read123	Total
Number of lines	682	5114	2847	6774	10721	1639	3082	30859
Number of transformations	67	290	111	387	544	77	135	1611
◦ Semantics-preserving (§4.2.2)	45	231	80	275	381	31	78	1121
◦ Semantics-increasing/-decreasing	22	58	31	102	150	39	53	455
◦ Semantics-revising	—	1	—	10	13	7	4	35
Preparation phase (§4.2.1)	1	—	—	15	24	11	14	65
◦ Known bugs	—	—	—	1	11	—	4	16
◦ Post-extraction	—	—	—	7	8	7	5	27
◦ Initial correction	1	—	—	7	5	4	5	22
Resolution phase	21	59	31	97	139	35	43	425
◦ Extension (§4.2.3)	—	17	26	—	—	31	38	112
◦ Relaxation (§4.2.4)	18	39	5	75	112	—	2	251
◦ Correction (§4.2.5)	3	3	—	22	27	4	3	62

Convergence reveals relationships

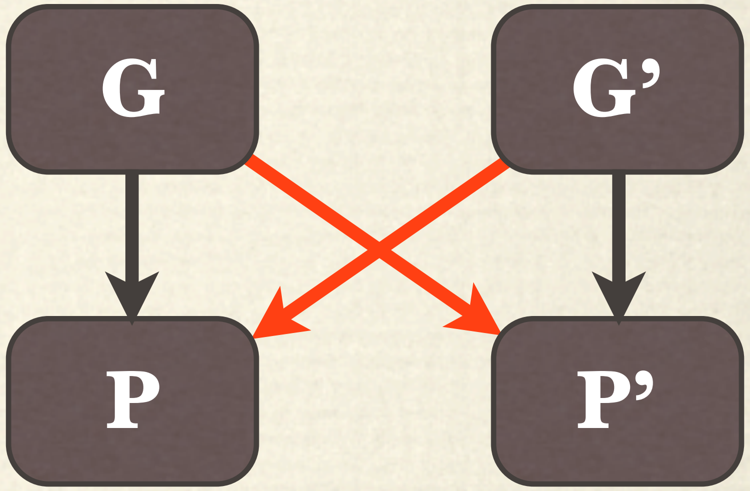
	jls1	jls12	jls123	jls2	jls3	read12	read123	Total
∘ *rename*	9	4	2	9	10	—	2	36
∘ *reroot*	2	—	—	2	2	2	1	9
∘ *unfold*	1	10	8	11	13	2	3	48
∘ *fold*	4	11	4	11	13	2	5	50
∘ *inline*	3	67	8	71	100	—	1	250
∘ *extract*	—	17	5	18	30	—	5	75
∘ *chain*	1	—	2	—	—	1	4	8
∘ *massage*	2	13	—	15	32	5	3	70
∘ *distribute*	3	4	2	3	6	—	—	18
∘ *factor*	1	7	3	5	24	3	1	44
∘ *deyaccify*	2	20	—	25	33	4	3	87
∘ *yaccify*	—	—	—	—	1	—	1	2
∘ *eliminate*	1	8	1	14	22	—	—	46
∘ *introduce*	—	1	30	4	13	3	34	85
∘ *import*	—	—	2	—	—	—	1	3
∘ *vertical*	5	7	7	8	22	5	8	62
∘ *horizontal*	4	19	5	17	31	4	4	84
∘ *add*	1	14	13	7	20	28	20	103
∘ *appear*	—	8	11	8	25	2	17	71
∘ *widen*	1	3	—	1	8	1	3	17
∘ *upgrade*	—	8	—	14	20	2	2	46
∘ *unite*	18	2	—	18	21	5	4	68
∘ *remove*	—	10	1	11	18	—	1	41
∘ *disappear*	—	7	4	11	11	—	—	33
∘ *narrow*	—	—	1	—	4	—	—	5
∘ *downgrade*	—	2	—	8	3	—	—	13
∘ *define*	—	6	—	4	9	1	6	26
∘ *undefine*	—	3	—	5	3	—	—	11
∘ *redefine*	—	3	—	8	7	6	2	26
∘ *inject*	—	—	—	2	4	—	1	7
∘ *project*	—	1	—	1	2	—	—	4
∘ *replace*	3	1	2	3	6	1	1	17
∘ *unlabel*	—	—	—	—	—	—	2	2

Resources

- R. Lämmel, V. Zaytsev, *An Introduction to Grammar Convergence*, iFM 2009, LNCS 5423.
- R. Lämmel, V. Zaytsev, *Reverse Engineering Grammar Relationships*, WSR 2010.
- V. Zaytsev, *Language Convergence Infrastructure*, GTTSE 2009, LNCS 6491.
- R. Lämmel, V. Zaytsev, *Recovering Grammar Relationships for the JLS*, SCAM 2009, SQJ 19:2, arXiv abs/1008.4188.

Grammar-based testing

- ★ Asymmetric comparison:
  - ✦ Reference grammar vs. parser under test
- ★ Symmetric comparison:
  - ✦ Differential testing
- ★ Systematic test data generation
  - ✦ Controlled combinatorial coverage
- ★ Larger sets of smaller test data items



Test data generation (1/4)

$grammar(Ps)$ $\Leftarrow maplist(prod,Ps).$	$prod(p(L,N,X))$ $\Leftarrow mapopt(atom,L),\ atom(N),\ expr(X).$
$expr(true).$ $expr(t(T)) \Leftarrow atom(T).$ $expr(n(N)) \Leftarrow atom(N).$ $expr(','(Xs)) \Leftarrow maplist(expr,Xs).$ $expr(';'(Xs)) \Leftarrow maplist(expr,Xs).$ $expr('?'(X)) \Leftarrow expr(X).$ $expr('*'(X)) \Leftarrow expr(X).$ $expr('+'(X)) \Leftarrow expr(X).$	$tree(true).$ $tree(t(T)) \Leftarrow atom(T).$ $tree(n(P,T)) \Leftarrow prod(P),\ tree(T).$ $tree(','(Ts)) \Leftarrow maplist(tree,Ts).$ $tree(';'(X,T)) \Leftarrow expr(X),\ tree(T).$ $tree('?'(Ts)) \Leftarrow mapopt(tree,Ts).$ $tree('*'(Ts)) \Leftarrow maplist(tree,Ts).$ $tree('+'(Ts)) \Leftarrow maplist1(tree,Ts).$

Test data generation (2/4)

```
mark(C,p(L,N,X1),p(L,N,X2)) ⇐
    mark(C,X1,X2).

mark(uc,n(N),{n(N)}).
mark(bc,';'(Xs),{';'(Xs)}).
mark(bc,'?'(X),{'?'(X)}).
mark(bc,'*'(X),{'*'(X)}).
mark(bc,'+'(X),{'+'(X)}).

mark(C,'?'(X1),'?'(X2)) ⇐
    mark(C,X1,X2).
mark(C,'*'(X1),'*'(X2)) ⇐
    mark(C,X1,X2).
mark(C,'+'(X1),'+'(X2)) ⇐
    mark(C,X1,X2).

mark(C,','(Xs1),','(Xs2)) ⇐
    append(Xs1a,[X1|Xs1b],Xs1),
    append(Xs1a,[X2|Xs1b],Xs2),
    mark(C,X1,X2).

mark(C,';'(Xs1),';'(Xs2)) ⇐
    append(Xs1a,[X1|Xs1b],Xs1),
    append(Xs1a,[X2|Xs1b],Xs2),
    mark(C,X1,X2).
```

Marked productions are essentially marked expressions.

A nonterminal occurrence provides a focus for unfolding coverage. The EBNF forms ‘;’, ‘?’, ‘*’, ‘+’ provide foci for branch coverage.

Foci for BC and UC may also be found by recursing into subexpressions.

Sequences and choices combine multiple expressions, and foci are found by considering one subexpression at the time.

Coverage criteria

- ★ **Trivial** coverage: if the test data set is not empty.
- ★ **Nonterminal** coverage: if each nonterminal is exercised at least once.
- ★ **Production** coverage: if each production in the grammar is exercised at least once.
- ★ **Branch** coverage: each branch of ?|*+
- ★ **Unfolding** coverage: each production of each right hand side nonterminal occurrence
- ★ **Context-dependent branch coverage**!

Test data generation (3/4)

$$
\begin{array}{l}
\mathit{vary}(G,\{n(N)\},n(P,T)) \Leftarrow \\
\quad \mathit{def}(G,N,Ps), \\
\quad \mathit{member}(P,Ps), \\
\quad P = p(\_,\_,X), \\
\quad \mathit{complete}(G,X,T).
\end{array}
$$

A nonterminal occurrence in focus is varied so that all productions are exercised. (The complete spec also deals with chain productions and top-level choices in a manner that increases variation in a reasonable sense.)

$$
\begin{array}{l}
\mathit{vary}(G,\{\text{';'}(Xs)\},\text{';'}(X,T)) \Leftarrow \\
\quad \mathit{member}(X,Xs), \\
\quad \mathit{complete}(G,X,T).
\end{array}
$$

A choice in focus is varied so that all branches are exercised.

$$
\begin{array}{l}
\mathit{vary}(\_,\{\text{'?'}(\_)\},\text{'?'}([])). \\
\mathit{vary}(G,\{\text{'?'}(X)\},\text{'?'}([T])) \Leftarrow \\
\quad \mathit{complete}(G,X,T). \\
\mathit{vary}(\_,\{\text{'}{*}\text{'}(\_)\},\text{'}{*}\text{'}([])). \\
\mathit{vary}(G,\{\text{'}{*}\text{'}(X)\},\text{'}{*}\text{'}([T])) \Leftarrow \\
\quad \mathit{complete}(G,X,T). \\
\mathit{vary}(G,\{\text{'+'}(X)\},\text{'+'}([T])) \Leftarrow \\
\quad \mathit{complete}(G,X,T). \\
\mathit{vary}(G,\{\text{'+'}(X)\},\text{'+'}([T1,T2])) \Leftarrow \\
\quad \mathit{complete}(G,X,T1), \\
\quad \mathit{complete}(G,X,T2).
\end{array}
$$

An optional expression and a '*' repetition in focus are varied so that the cases for no tree and one tree are exercised. A '+' repetition is varied so that the cases for sequences of length 1 and 2 are exercised.

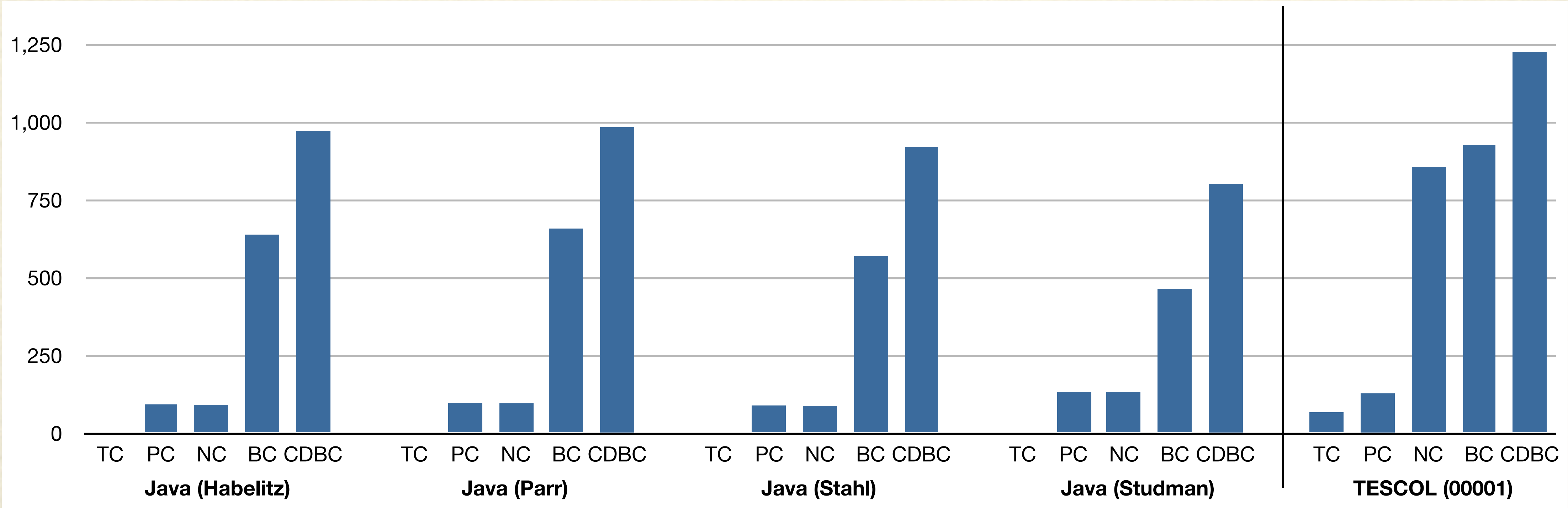
We omit all clauses for recursing into compound expressions; they mimic shortest completion but they are directed in a way that they reach the focus.

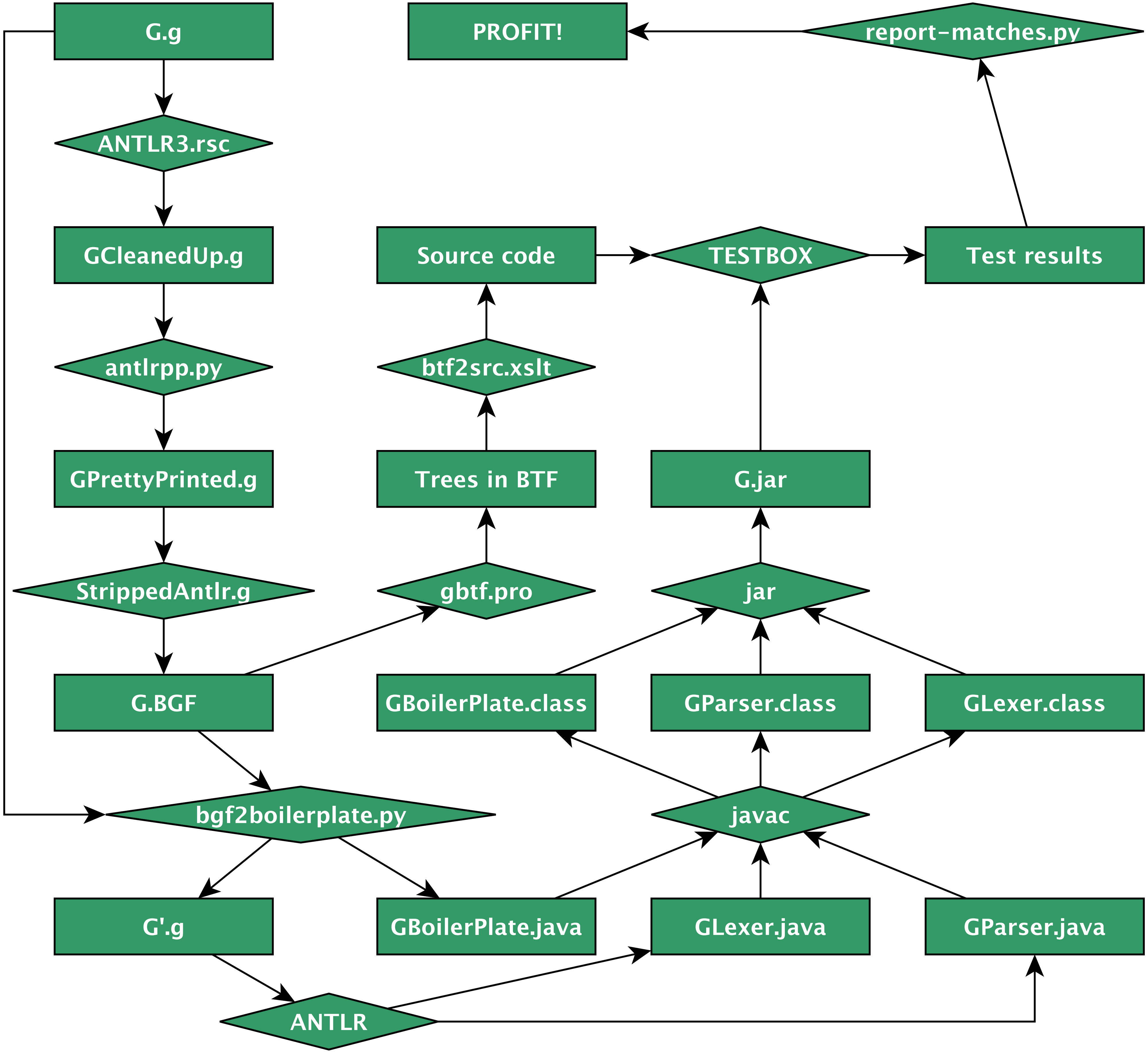
Test data generation (4/4)

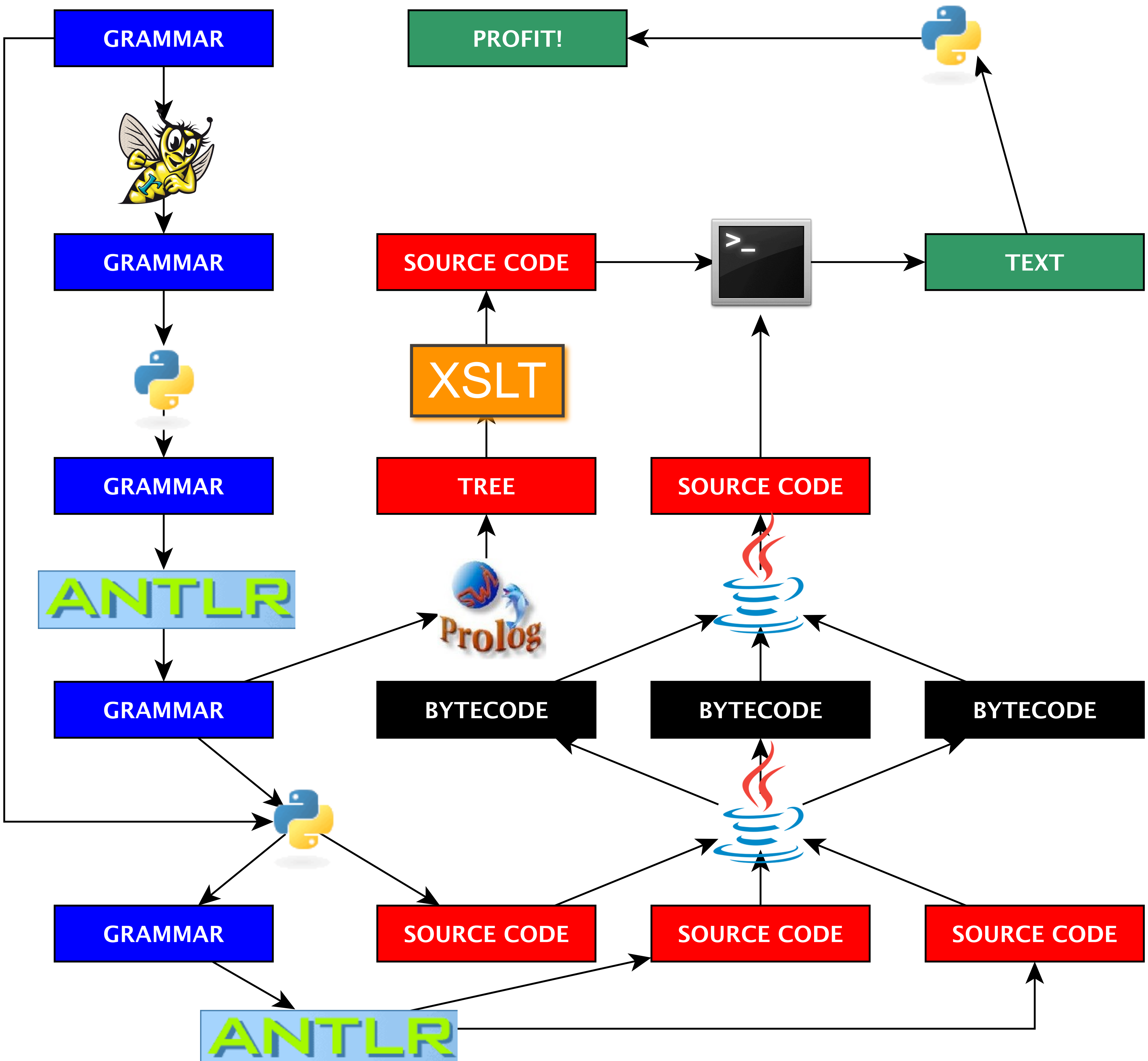
$tc(G,R,T)$
$\quad \Leftarrow def(G,R,\_),\ complete(G,n(R),T).$
$nc(G,R,T)$
$\quad \Leftarrow def(G,R,\_),\ dist(G,R,H,\_),\ hole(G,n(R),H,T,V),\ complete(G,n(H),V).$
$pc(G,R,T)$
$\quad \Leftarrow def(G,R,Ps),\ member(P,Ps),\ complete(G,P,T).$
$pc(G,R,T)$
$\quad \Leftarrow def(G,R,\_),\ dist(G,R,H,\_),\ hole(G,n(R),H,T,V),\ pc(G,H,V).$
$bc(G,R,T)$
$\quad \Leftarrow cdbc(bc,G,R,T).$
$uc(G,R,T)$
$\quad \Leftarrow cdbc(uc,G,R,T).$
$cdbc(C,G,R,T)$
$\quad \Leftarrow def(G,R,Ps),\ member(P,Ps),\ mark(C,P,F),\ vary(G,F,T).$
$cdbc(C,G,R,T)$
$\quad \Leftarrow def(G,R,\_),\ dist(G,R,H,\_),\ hole(G,n(R),H,T,V),\ cdbc(C,G,H,V).$

Grammar equivalence study: Java

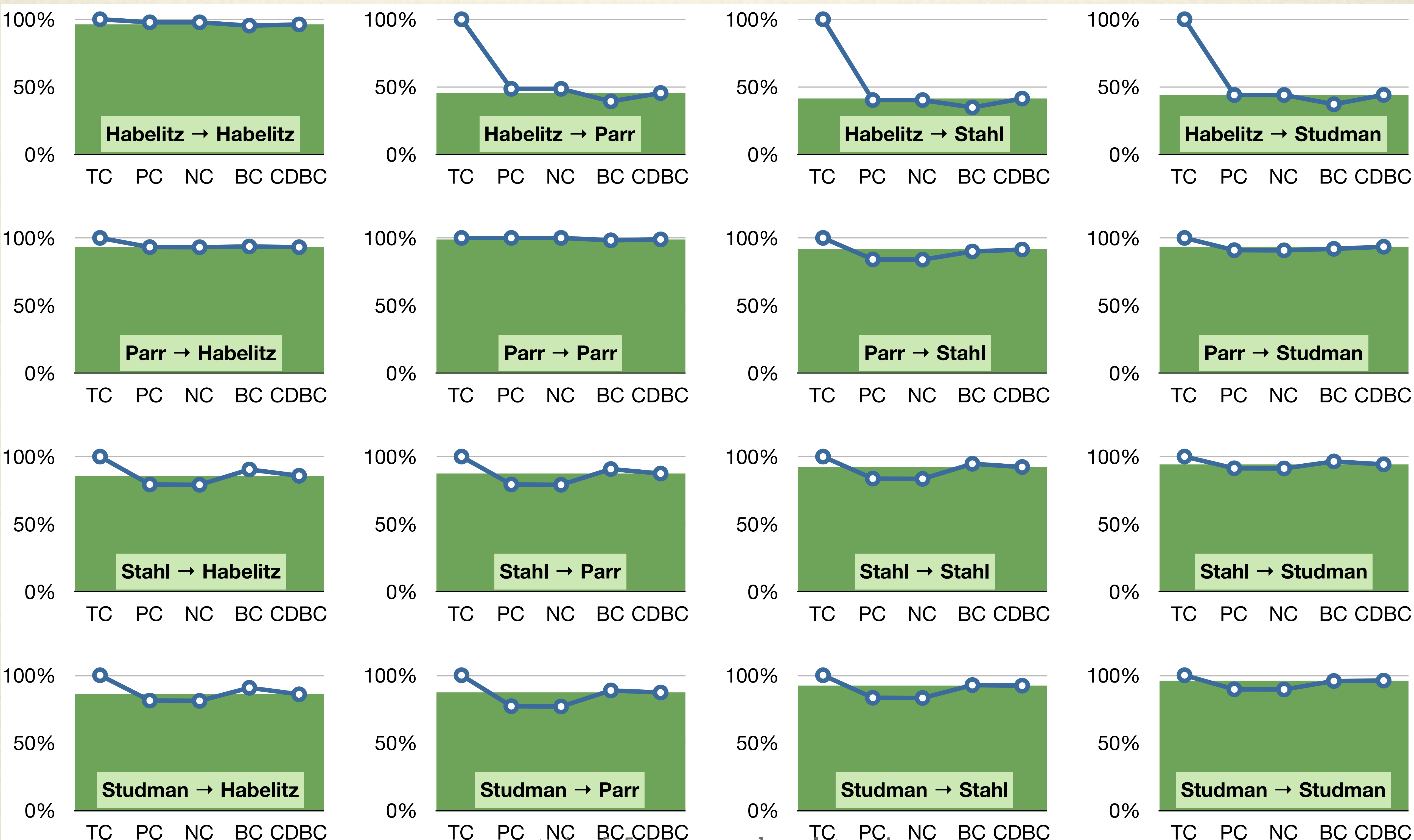
Codename	Tech	Author	year	PROD	VAR	TERM	…
Habelitz	ANTLR3	Dieter Habelitz	2008	397	226	166	…
Parr	ANTLR3	Terence Parr	2006	425	151	157	…
Stahl	ANTLR2	Michael Stahl	2004	262	155	167	…
Studman	ANTLR2	Michael Studman	2004	267	161	168	…




G.g
PROFIT!
report-matches.py
ANTLR3.rsc
GCleanedUp.g
Source code
TESTBOX
Test results
antlrpp.py
btf2src.xslt
GPrettyPrinted.g
Trees in BTF
G.jar
StrippedAntlr.g
gbtf.pro
jar
G.BGF
GBoilerPlate.class
GParser.class
GLexer.class
bgf2boilerplate.py
javac
G'.g
GBoilerPlate.java
GLexer.java
GParser.java
ANTLR


GRAMMAR
PROFIT!
GRAMMAR
SOURCE CODE
TEXT
XSLT
GRAMMAR
TREE
SOURCE CODE
ANTLR
Prolog
GRAMMAR
BYTECODE
BYTECODE
BYTECODE
GRAMMAR
SOURCE CODE
SOURCE CODE
SOURCE CODE
ANTLR

Grammar equivalence study: Java

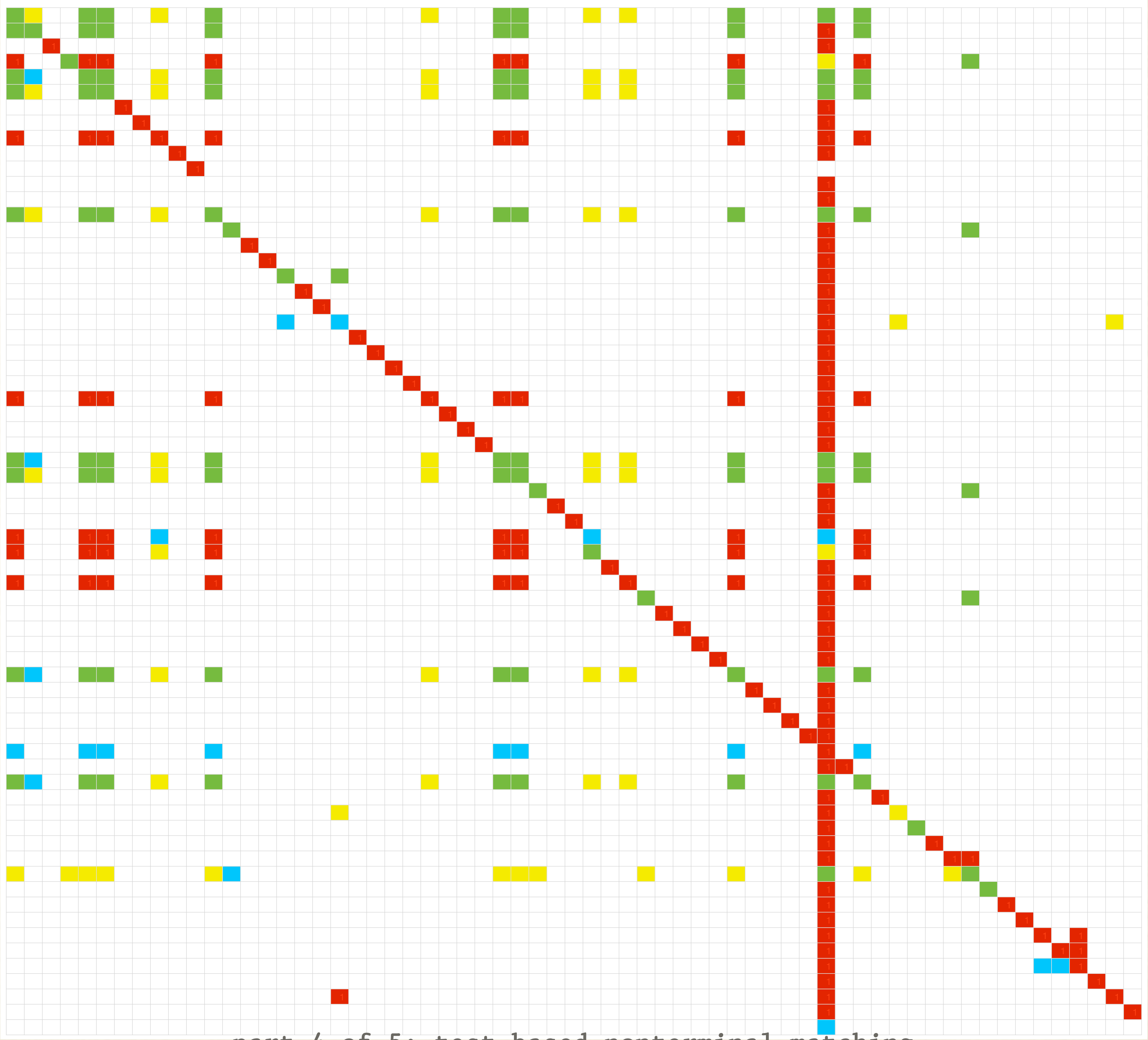


Resources

- ★ B. Fischer, R. Lämmel, V. Zaytsev, *Comparison of Context-free Grammars Based on Parsing Generated Test Data*, SLE 2011, LNCS 6940.
- ★ ~~A. Davenport, B. Fischer, CC 2012 draft.~~

- http://softlang.uni-koblenz.de/testmatch
- http://slps.sourceforge.net/testmatch
- http://grammarware.net/text/2011/testmatch.pdf

Nonterminal matching



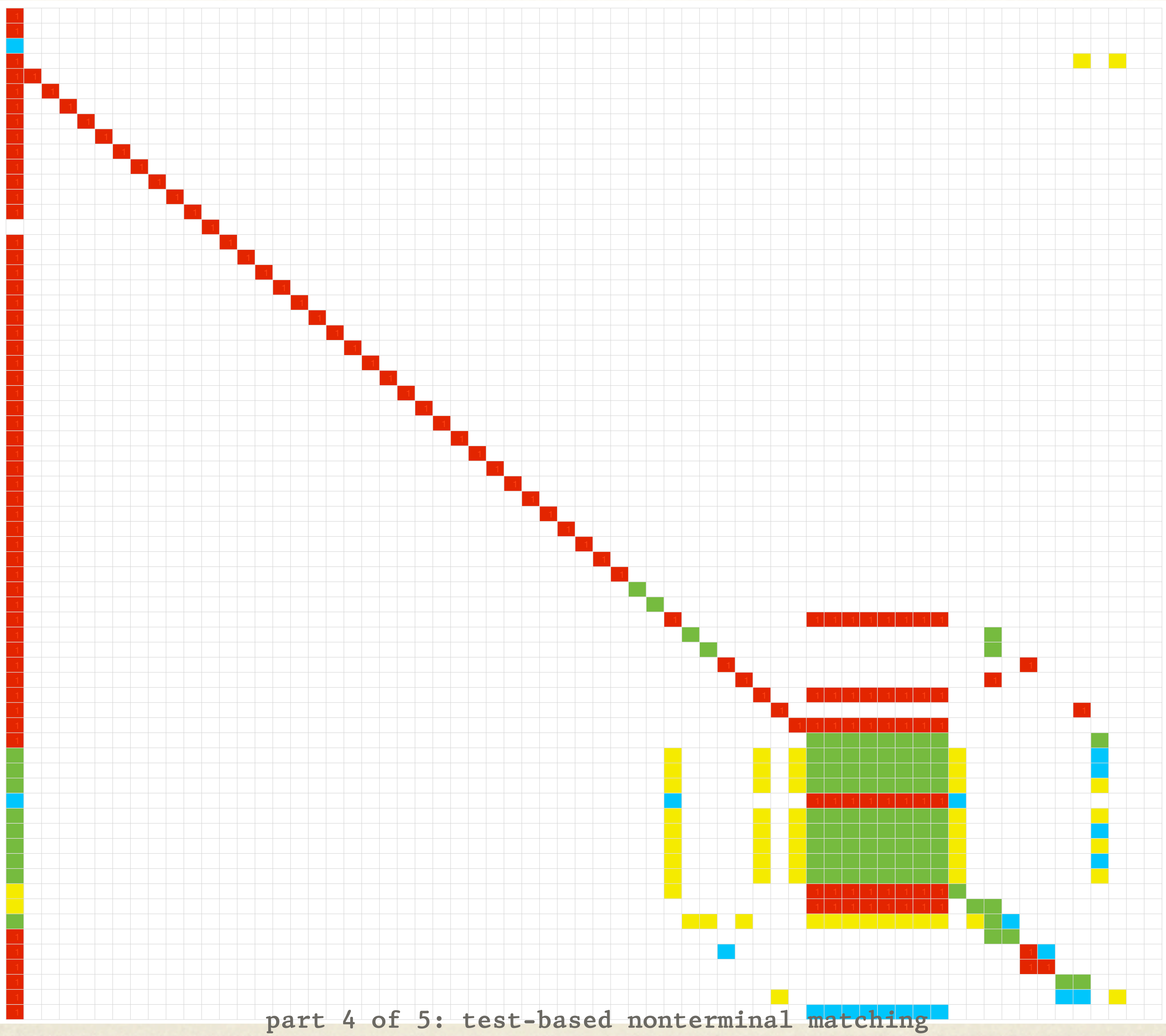
part 4 of 5: test-based nonterminal matching

Name matching study: TESCOL

Codename	Tech	Author	year	PROD	VAR	TERM	...
00000	ANTLR3	[obfuscated]	2010	126	74	107	…
00001	ANTLR3	[obfuscated]	2010	79	67	107	…
00010	ANTLR3	[obfuscated]	2010	101	73	108	…
00011	ANTLR3	[obfuscated]	2010	84	63	107	…
00100	ANTLR3	[obfuscated]	2010	93	76	108	…
00101	ANTLR3	[obfuscated]	2010	94	76	107	…
00110	ANTLR3	[obfuscated]	2010	92	75	120	…
00111	ANTLR3	[obfuscated]	2010	84	71	108	…
01000	ANTLR3	[obfuscated]	2010	85	67	107	…
...	…	…	…	…	…	…	…

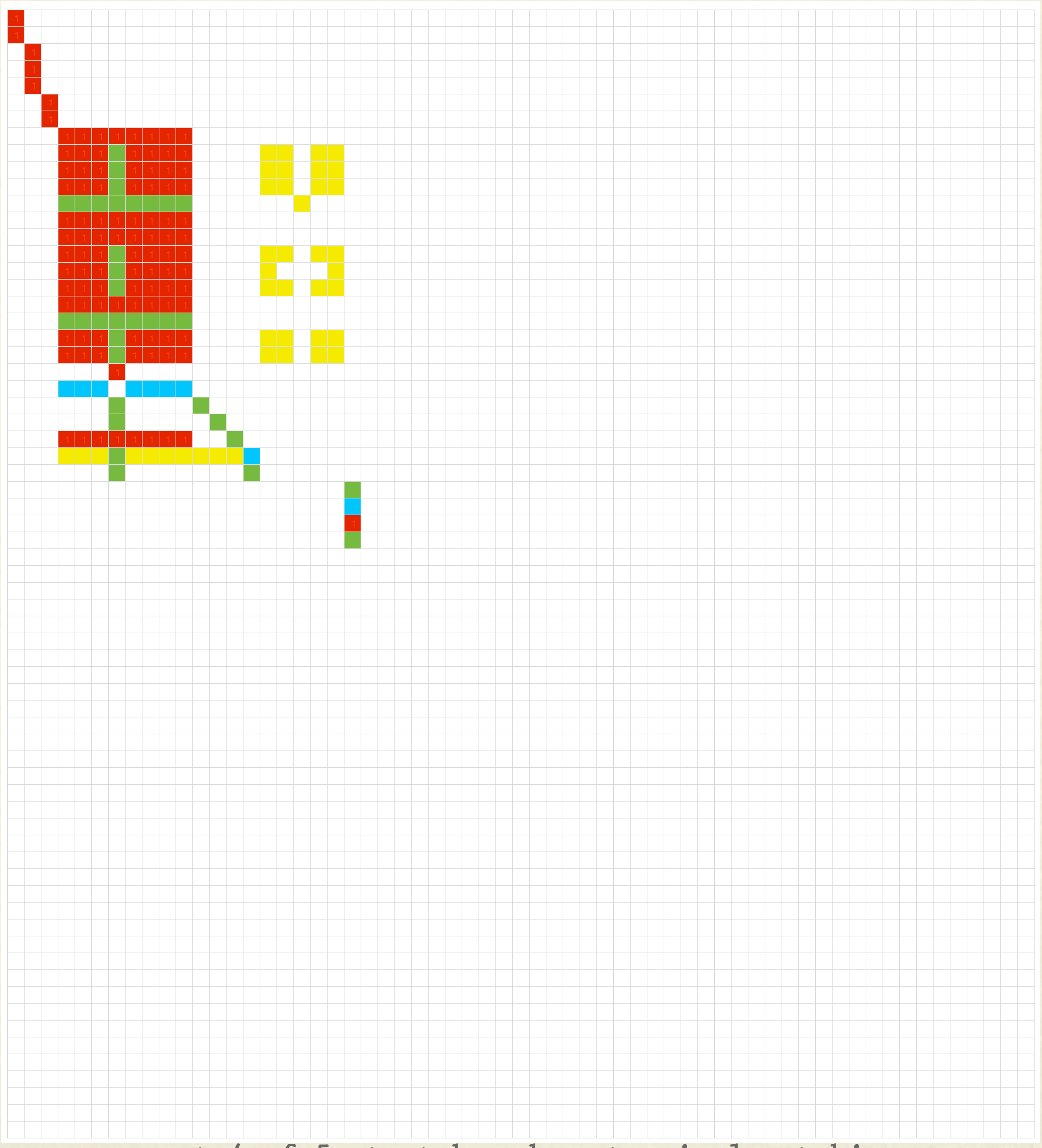
Nonterminal matching

Nonterminal matching



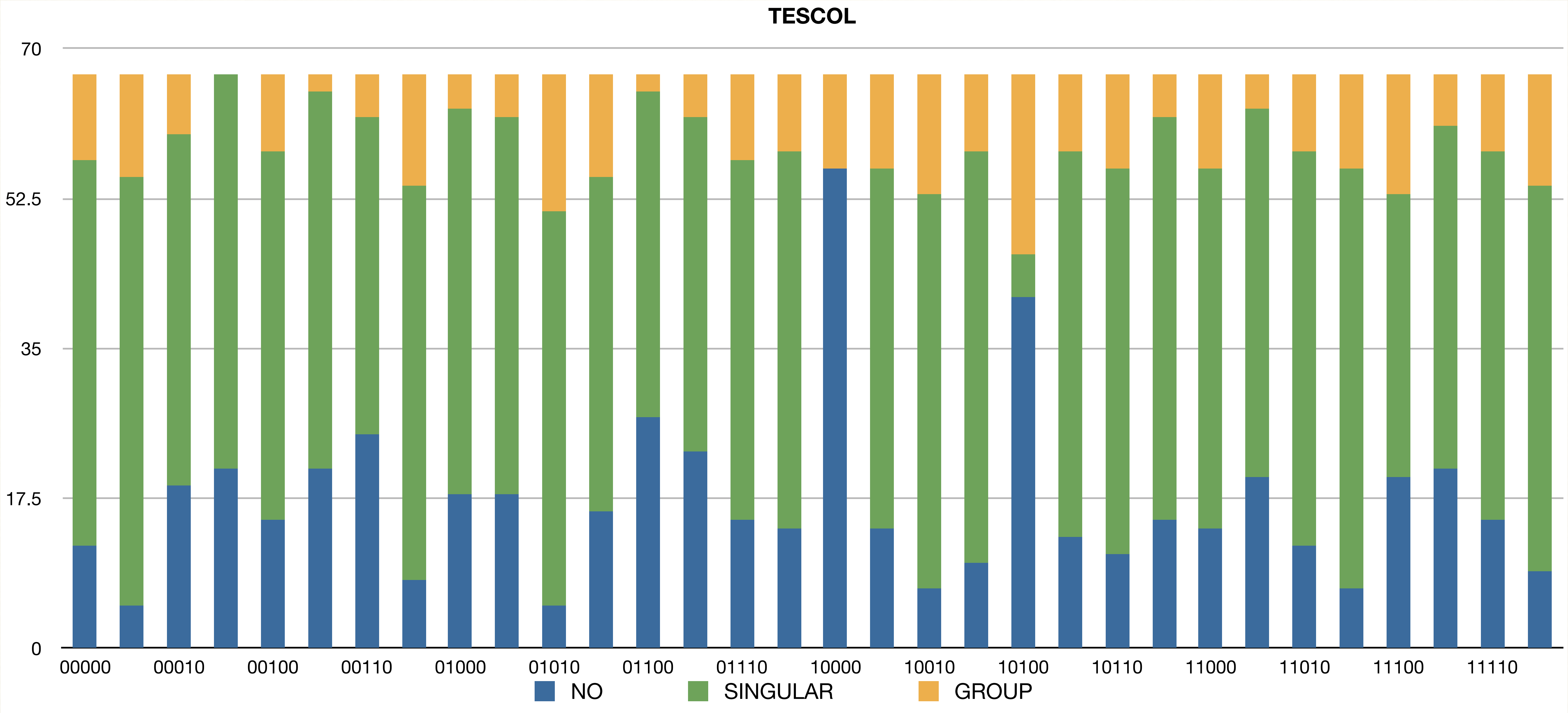
part 4 of 5: test-based nonterminal matching

Nonterminal matching



part 4 of 5: test-based nonterminal matching

Nonterminal matching



part 4 of 5: test-based nonterminal matching

Resources

- ★ B. Fischer, R. Lämmel, V. Zaytsev, *Comparison of Context-free Grammars Based on Parsing Generated Test Data*, SLE 2011, LNCS 6940.
- ★ …

- http://softlang.uni-koblenz.de/testmatch
- http://slps.sourceforge.net/testmatch
- http://slps.sourceforge.net/tank/#tescol
- http://grammarware.net/text/2011/testmatch.pdf

Roadmap

- ★ Motivation & methodology
- ★ Techniques not covered by the talk
- ★ Straightforward grammar comparison
- ★ Grammar convergence
- ★ Grammar-based testing
- ★ Test-based nonterminal matching

Thank you!

- http://grammarware.net/slides/2011/comparison-serg.pdf
- http://grammarware.net
- @grammarware
- …