Grammar Convergence

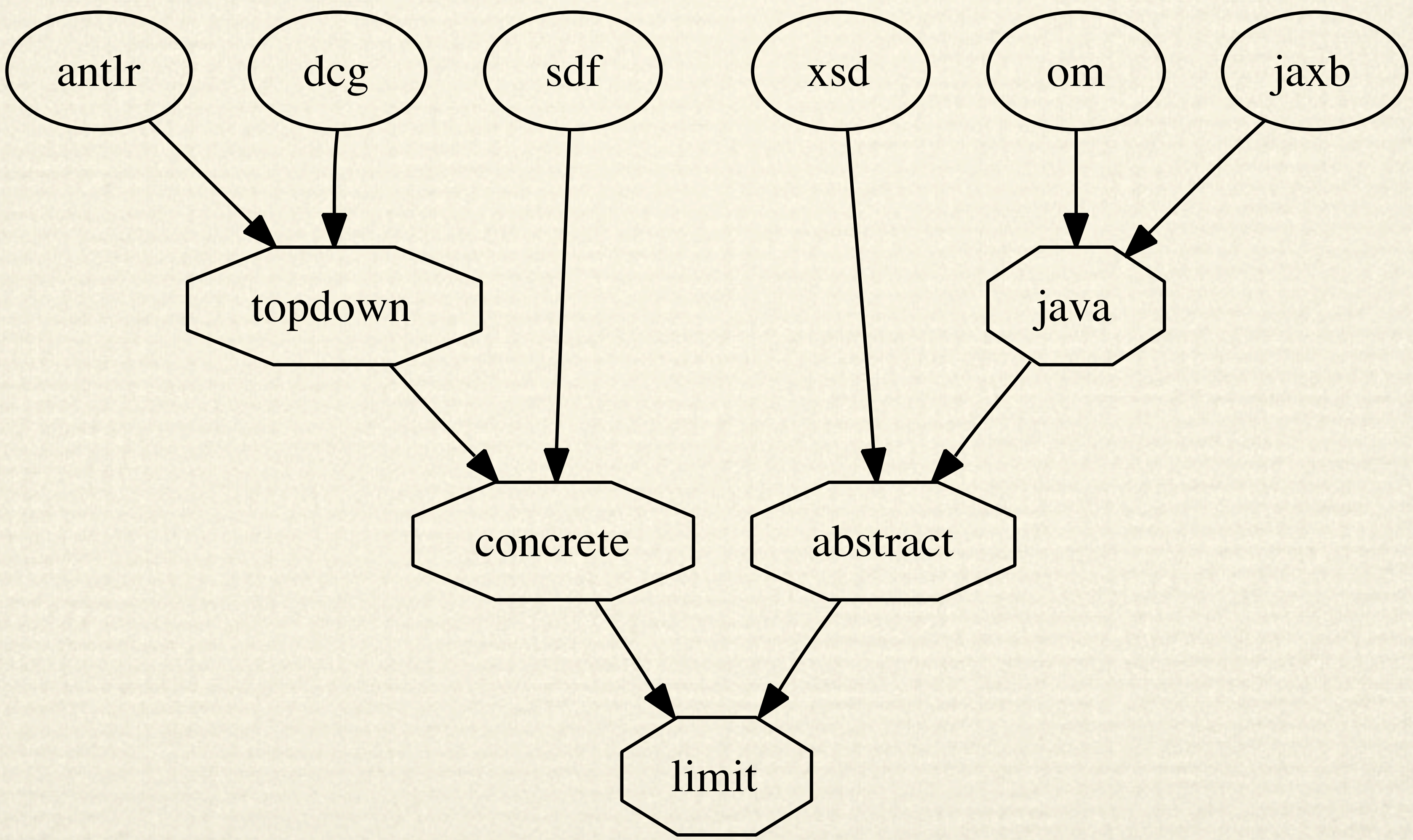
Ralf Lämmel and Vadim Zaytsev
Software Languages Team
Universität Koblenz-Landau

What is grammar convergence?

- ★ Distributed grammar knowledge
- ★ Surface and maintain relationships
- ★ Transform grammars until convergence
- ★ Lightweight verification

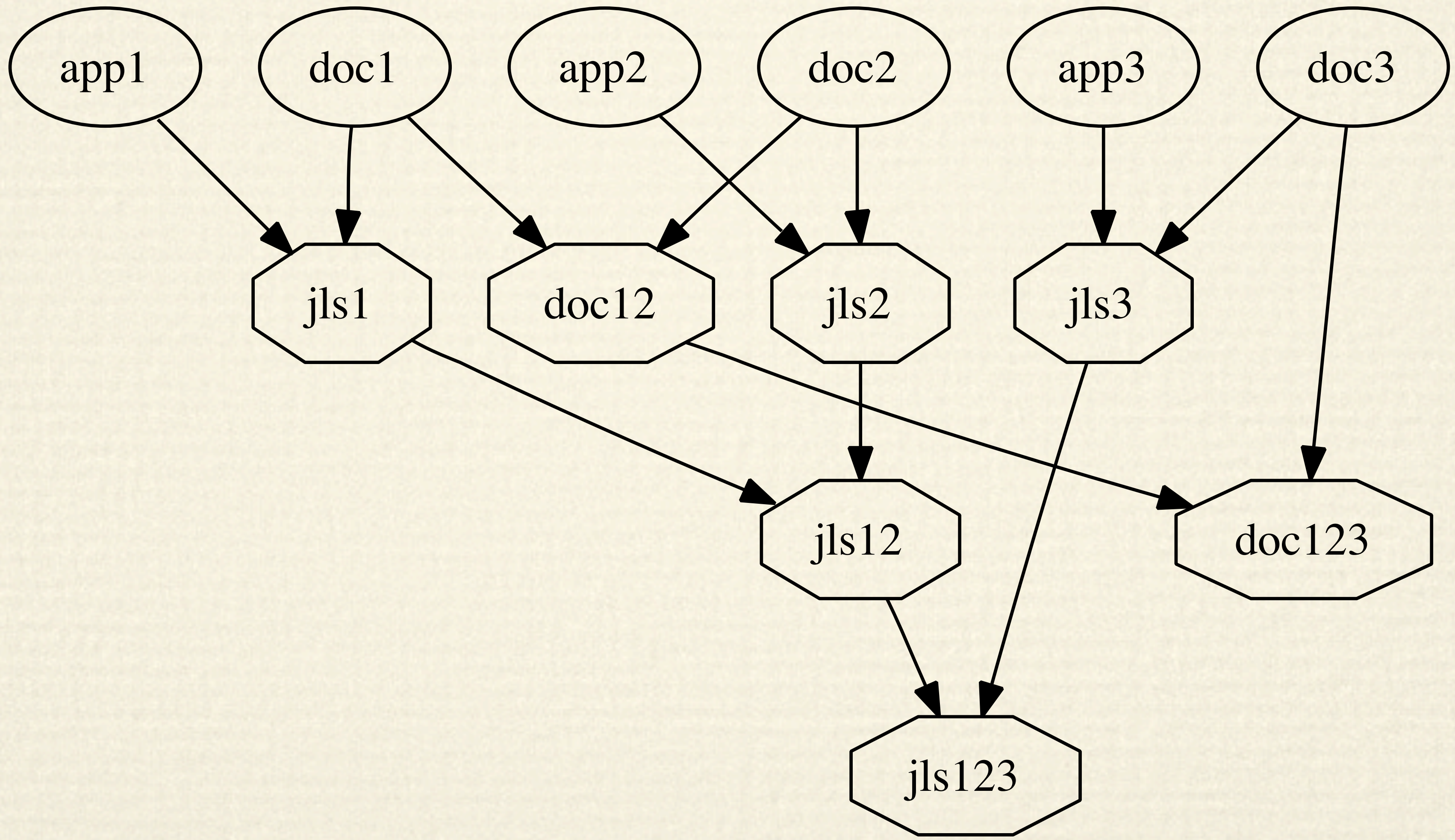
Grammar convergence scenario

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



Grammar convergence scenario

Different version of a language documented by specifications



Ralf Lämmel and Vadim Zaytsev, *Consistency of the Java Language Specification*, submitted draft, http://www.uni-koblenz.de/~laemmel/jls/

Grammar convergence framewowk

- Grammar *format* to abstract from idiosyncrasies
- Grammar *extraction* to feed into the format
- Grammar *comparison* for spotting grammar deviations
- Grammar *transformation*:
  - Refactoring
  - Extension / restriction
  - Revision

BGF: BNF–like Grammar Format

- BNF: symbols, composition
- EBNF: *, +, ?
- Production labels
- Expression selectors
- Universal type
- Namespaces

Grammar extract: ANTLR

```
g( [], [
  p([], program, +n(function)),
  p([], function, (n('ID'), +n('ID'), t(=), n(expr), +n('NEWLINE'))),
  p([], expr, (n(binary);n(apply);n(ifThenElse))),
  p([], binary, (n(atom), *((n(ops), n(atom))))),
  p([], apply, (n('ID'), +n(atom))),
  p([], ifThenElse, (t(if), n(expr), t(then), n(expr), t(else), n(expr))),
  p([], atom, (n('ID');n('INT');t('('), n(expr), t(')'))),
  p([], ops, (t(==);t(+);t(–)))
])
```

Grammar extract: XSD

```
g( ['Program', 'Fragment'], [
  p([], 'Program', +s(function, n('Function'))),
  p([], 'Fragment', n('Expr')),
  p([], 'Function', (s(name, v(string)), +s(arg, v(string)), s(rhs, n('Expr')))),
  p([], 'Expr', (n('Literal');n('Argument');n('Binary');n('IfThenElse');n('Apply'))),
  p([], 'Literal', s(info, v(int))),
  p([], 'Argument', s(name, v(string))),
  p([], 'Binary', (s(ops, n('Ops')), s(left, n('Expr')), s(right, n('Expr')))),
  p([], 'Ops', (s('Equal', true);s('Plus', true);s('Minus', true))),
  p([], 'IfThenElse', (s(ifExpr, n('Expr')), s(thenExpr, n('Expr')), s(elseExpr, n('Expr')))),
  p([], 'Apply', (s(name, v(string)), +s(arg, n('Expr'))))
])
```

Grammar extraction

- Get out of a source format
  - Can be ANTLR, SDF, Java, XSD, HTML
- Abstract from idiosyncrasies
  - XML–isms, semantic actions, etc
- Specific for the source format, not for the source

Available extractors

- Grammars for ANTLR parser generator: ANTLR self-application
- Definite clause grammars in Prolog: Prolog
- Java classes: reflection with `java.lang.reflect` or `com.sun.source.tree`
- Syntax Definition Formalism: term rewriting with ASF+SDF MetaEnvironment or Stratego/XT
- (E)BNF in HTML: stateful scanner in Python
- XML Schema schemata: Prolog

Grammar extraction for Java Language Specification

- ★ Use HTML representation (instead of PDF)
- ★ Many markup/well–formedness problems
- ★ Some syntax errors
- ★ Many obvious semantic errors

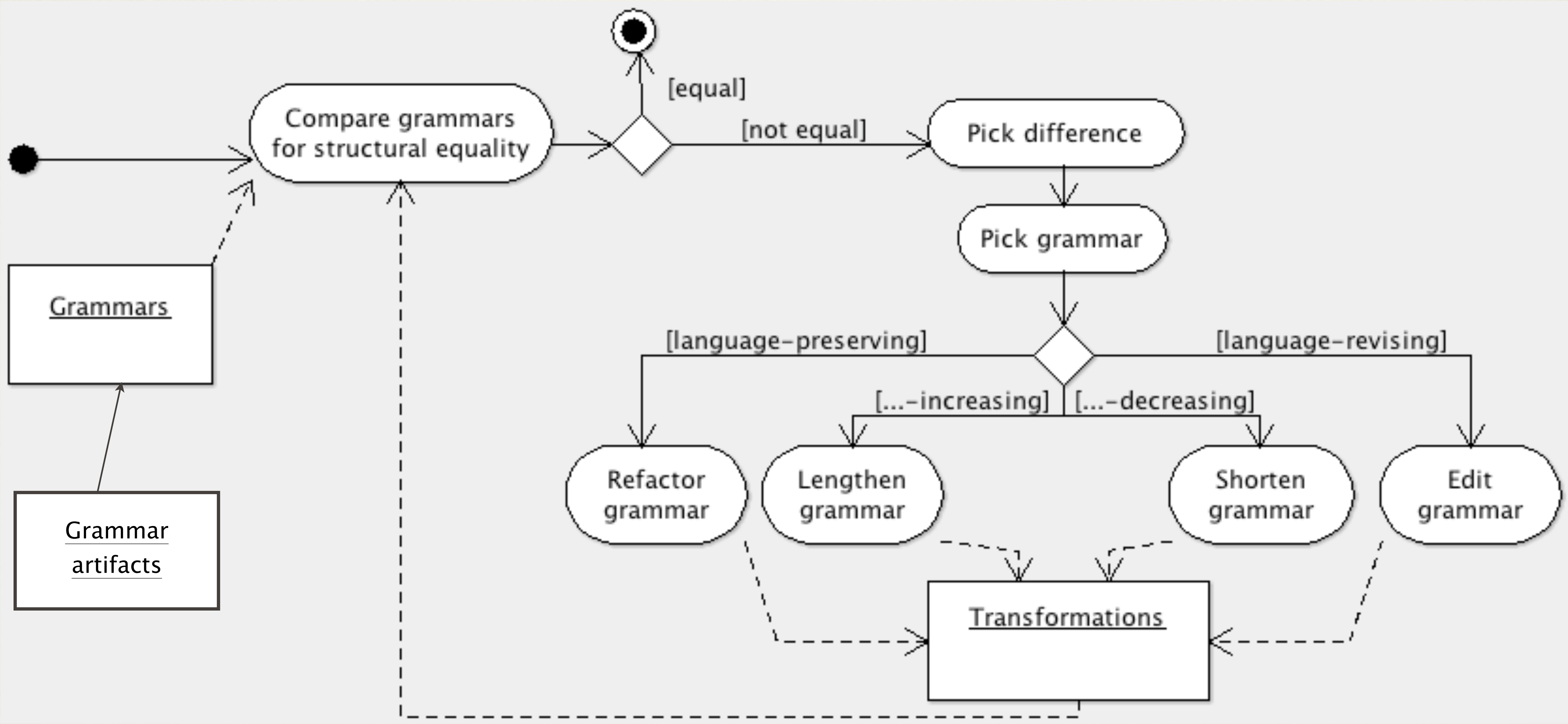
JLS irregularities in extraction

	app1	app2	app3	doc1	doc2	doc3	Total
Arbitrary lexical decisions	2	109	60	1	90	161	423
Well-formedness violations	5	0	7	4	11	4	31
Indentation violations	1	2	7	1	4	8	23
Recovery rules	3	12	18	2	59	47	141
◦ Match parentheses	0	3	6	0	0	0	9
◦ Metasymbol to terminal	0	1	7	0	27	7	42
◦ Merge adjacent symbols	1	0	0	1	1	0	3
◦ Split compound symbol	0	1	1	0	3	8	13
◦ Nonterminal to terminal	0	7	3	0	8	11	29
◦ Terminal to nonterminal	1	0	1	1	17	13	33
◦ Recover optionality	1	0	0	0	3	8	12
Purge duplicate definitions	0	0	0	16	17	18	51
Total	11	123	92	24	181	238	669

Grammar comparison

- ★ Compare grammars structurally.
- ★ Apply simple algebraic laws on grammars.
- ★ Provide suggestive input for transformation.

How grammar convergence works



Grammar transformation

- Performing post-extraction activities
- Refactoring for structural equivalence
- Extension to cover missing language construct
- Restriction to abstract away “irrelevant” constructs
- Replacement to fix accidental deviations
- Capture and document language differences

A fragment of concrete syntax.
What if we want to derive the abstract syntax?

Need to project away “(“ & “)”

```
expr : ...;
atom : ID | INT  | '(' expr ')';
```

Need to merge “expr” & “atom”

Alternative needs to go entirely

A transformation sequence

expr : ...;
atom : ID | INT | '(' expr ')';

abstractize

expr : ...;
atom : ID | INT | **expr**;

vertical

expr : ...;
atom : ID;
atom : INT;
atom : expr;

unite

expr : ...;
expr : ID;
expr : INT;
expr : expr;

abridge

expr : ...;
expr : ID;
expr : INT;

XBGF Operator Suite

$$L(G_1) = L(G_2)$$

- Semantics–preserving (refactoring)
  - rename, import, introduce, eliminate
  - fold, unfold, extract, inline
  - factor, distribute, horizontal, <u>vertical</u>
  - yaccify, deyaccify, massage
  - designate, unlabel
  - ...

XBGF Operator Suite

$$L(G_1) \subseteq L(G_2)$$
$$\vee$$
$$L(G_2) \subseteq L(G_1)$$

- Semantics–increasing/–decreasing
  - appear, disappear
  - narrow, widen
  - add, remove
  - upgrade, downgrade
  - <u>unite</u>
  - ...

XBGF Operator Suite

$$L(G_1) \not\subseteq L(G_2)$$

$$\wedge$$

$$L(G_2) \not\subseteq L(G_1)$$

- Semantics–revising
  - undefine, define
  - inject, project, permute
  - <u>abstractize</u>, concretize
  - replace, redefine

Grammar refactoring example

BGF (*doc2*)

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

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

XBGF (*grammar refactoring*)

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

Grammar extension example

BGF (*doc2*)

```
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 (*app2*, *app3*)

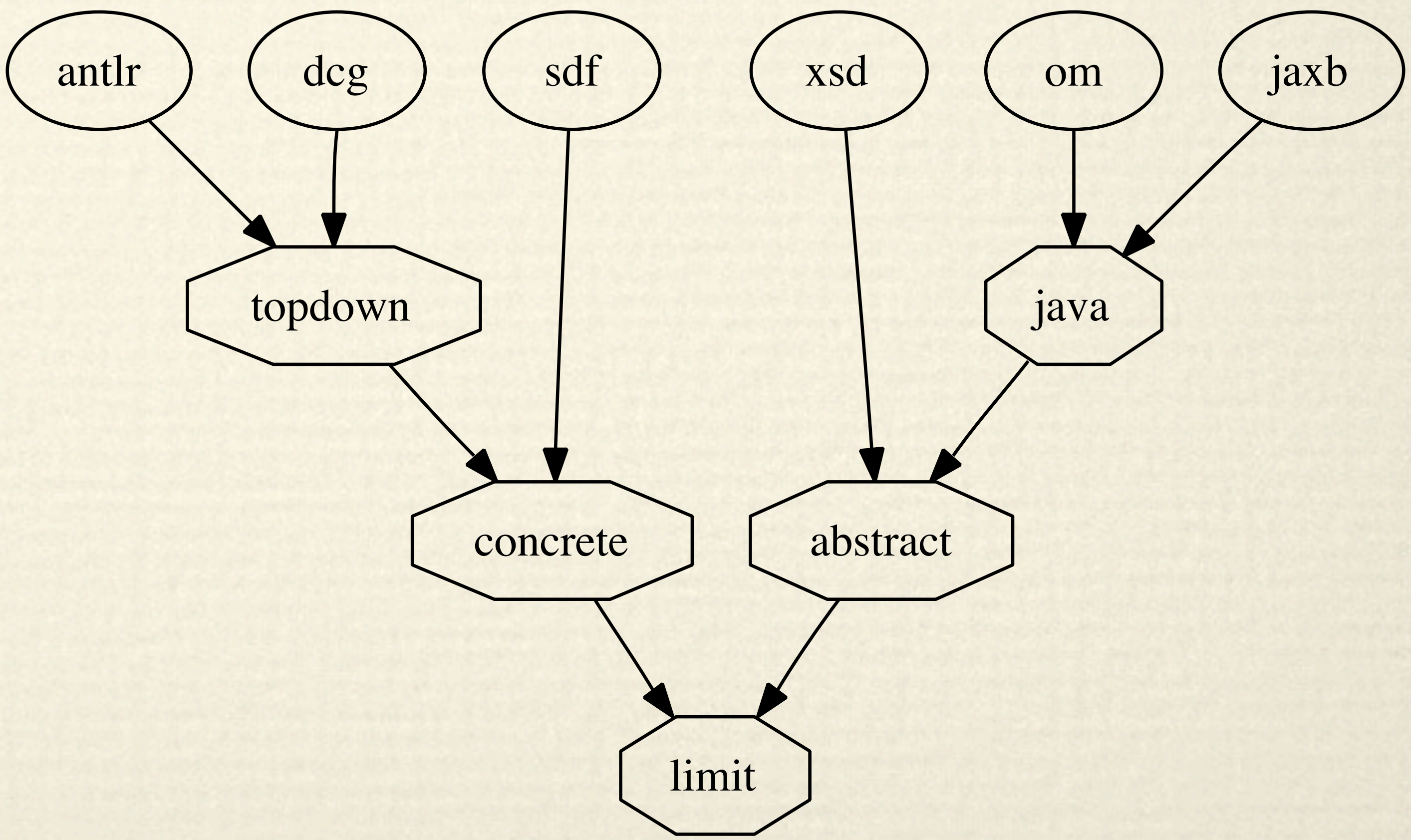
Expression2:
  Expression3 Expression2Rest ?
Expression2Rest:
  (Infixop Expression3) *
Expression2Rest:
  ~~Expression3~~ "instanceof" Type

XBGF (*grammar correction*)

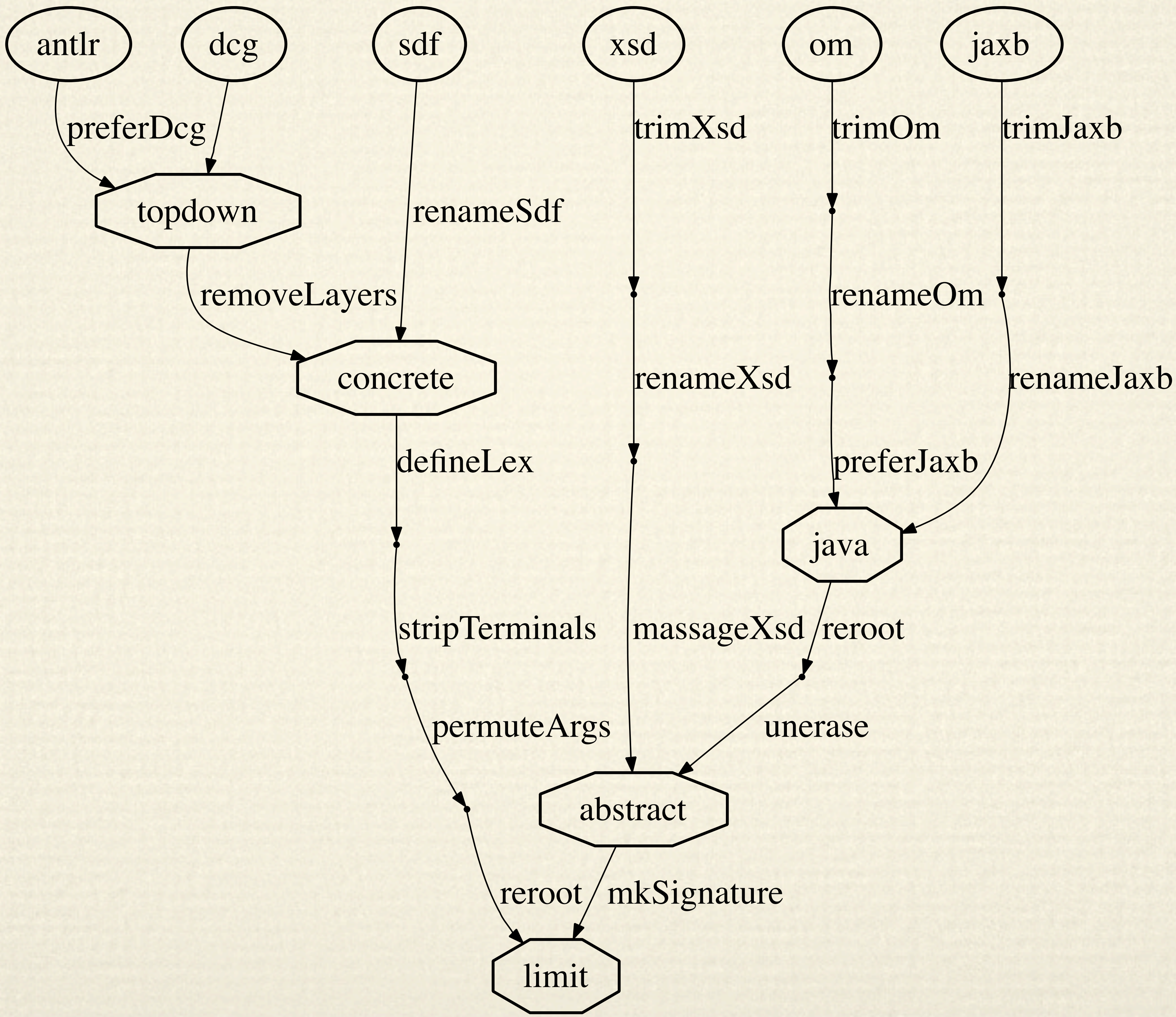
```
project(
 Expression2Rest:
     < Expression3 > "instanceof" Type
);
```

A DSL for grammar convergence

- Domain concepts of the little language
  - Defining “sources” of the convergence tree.
  - Defining “targets” (non–leafs) of that tree.
  - Pursuing
    - Extraction
    - Validation
    - Comparison
    - Transformation



A more detailed convergence tree




app1
doc1
app2
doc2
app3
doc3
correct
correct
correct
recover
recover
recover
generalizeModifiers
edit
correct
correct
correct
refactor
refactor
generalizeNames
extend
generalizeModifiers
refactorPackageNames
generalizeModifiers
doc12
refactor
refactor
generalizeNames
extendModifiers
refactorPackageNames
generalizeNames
jls1
generalizeLabels
refactorStatements
generalizeLabels
deyaccifyStatements
extend
deyaccifyStatements
generalizeBlock
doc123
generalizeStatements
refactorTypes
refactorTypes
refactorTypes
refactorExpressions
refactorExpressions
editDeclarations
inlineStatements
inlineStatements
refactorExpressions
generalizeStatements
generalizeStatements
editExpressions
editExpressions
refactorStatements
generalizeBlock
eliminate
generalize
editExpressions
refactorClasses
refactorClasses
refactorInterfaces
refactorInterfaces
extend
refactorDeclarations
refactorEnums
edit
refactorDeclarations
jls2
edit
jls12
jls3
rename
extendDeclarations
extendExpressions
extendStatements
jls123

Consolidation of basic metrics

	Productions	Nonterminals	Tops	Bottoms
app1	282	135	1	7
doc1	315	148	1	9
app2	185	80	6	11
doc2	346	151	1	11
app3	245	114	2	12
doc3	435	197	3	14

	Productions	Nonterminals	Tops	Bottoms
jls1	278	132	1	7
jls2	182	75	1	7
jls3	236	109	1	7
jls12	182	75	1	7
jls123	236	109	1	7
doc12	345	152	1	7
doc123	438	201	1	7

Transformation statistics for JLS

	jls1	jls2	jls3	jls12	jls123	doc12	doc123	Total
Number of lines	602	6339	9899	4681	2917	1597	2813	28848
Number of transformations	62	390	539	293	120	76	119	1599
○ semantics-preserving	40	278	385	234	87	33	59	1116
○ semantics-increasing or -decreasing	22	102	141	58	32	36	56	447
○ semantics-revising	—	10	13	1	1	7	4	36
Number of issues	8	40	48	25	17	32	40	210
○ recoveries	—	7	8	—	—	7	4	26
○ corrections	5	23	21	2	—	10	7	68
○ extensions	—	1	—	17	14	15	28	75
○ optimizations	3	9	19	6	3	—	1	41

Conclusion and future work

- ★ Synchronise scattered grammar knowledge
- ★ Further consolidation of operator suite
- ★ Co–transformation of parse–trees possible
- ★ Semi–automatic approach desirable

Thank you!

* Questions?
* Comments?

* **S**oftware **L**anguage **P**rocessing **S**uite is here: http://slps.sf.net/