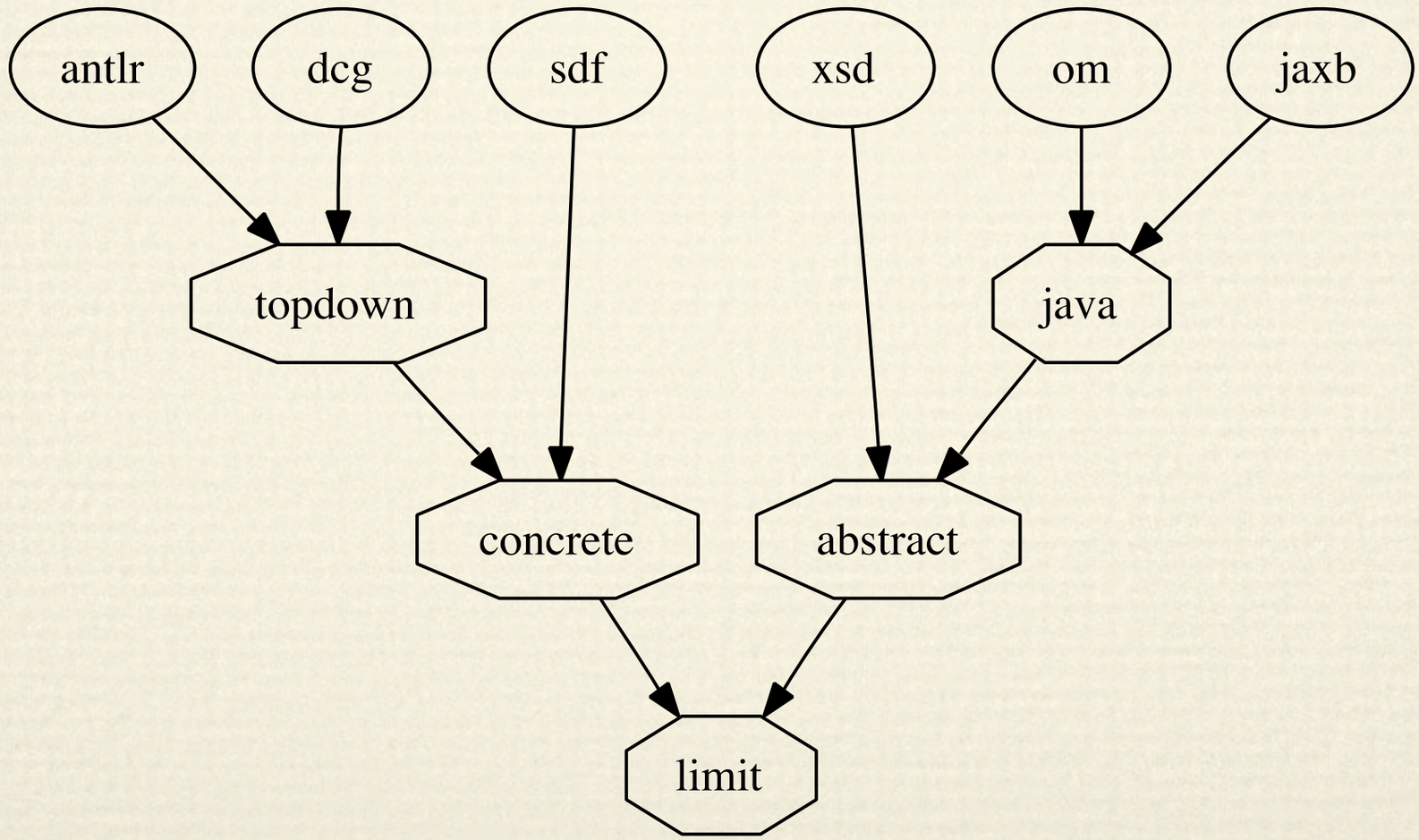
# Grammar Convergence

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

# What is grammar convergence?

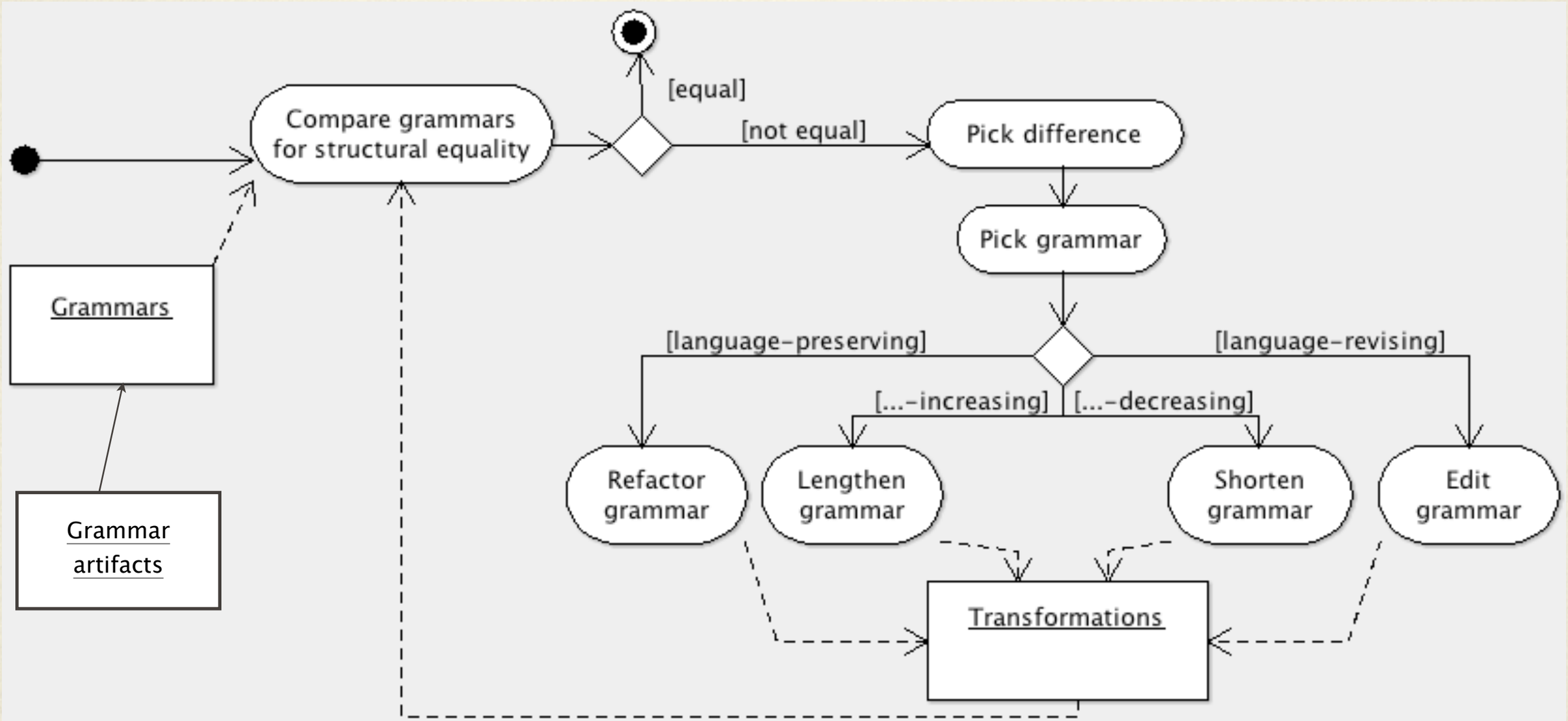
*Think of scattered grammar knowledge (say, in language documentation, parsers, object models, etc.)* ***how to establish relationships between the grammars, how to verify that these relationships are preserved****?*



# What is grammar convergence?

- ★ 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

# How grammar convergence works



# 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
- ★ Extraction is a generic, partial operation.

# An extractor for SDF

```
context-free syntax
    Function+                                  -> Program
    Name Name+ "=" Expr Newline+               -> Function
    Expr Ops Expr                              -> Expr {left,prefer,cons(binary)}
    Name Expr+                                 -> Expr {avoid,cons(apply)}
    "if" Expr "then" Expr "else" Expr          -> Expr {cons(ifThenElse)}
    "(" Expr ")"                               -> Expr {bracket}
    Name                                       -> Expr {cons(argument)}
    Int                                        -> Expr {cons(literal)}

    "-"                                        -> Ops {cons(minus)}
    "+"                                        -> Ops {cons(plus)}
    "=="                                       -> Ops {cons(equal)}
```

# An extractor for SDF

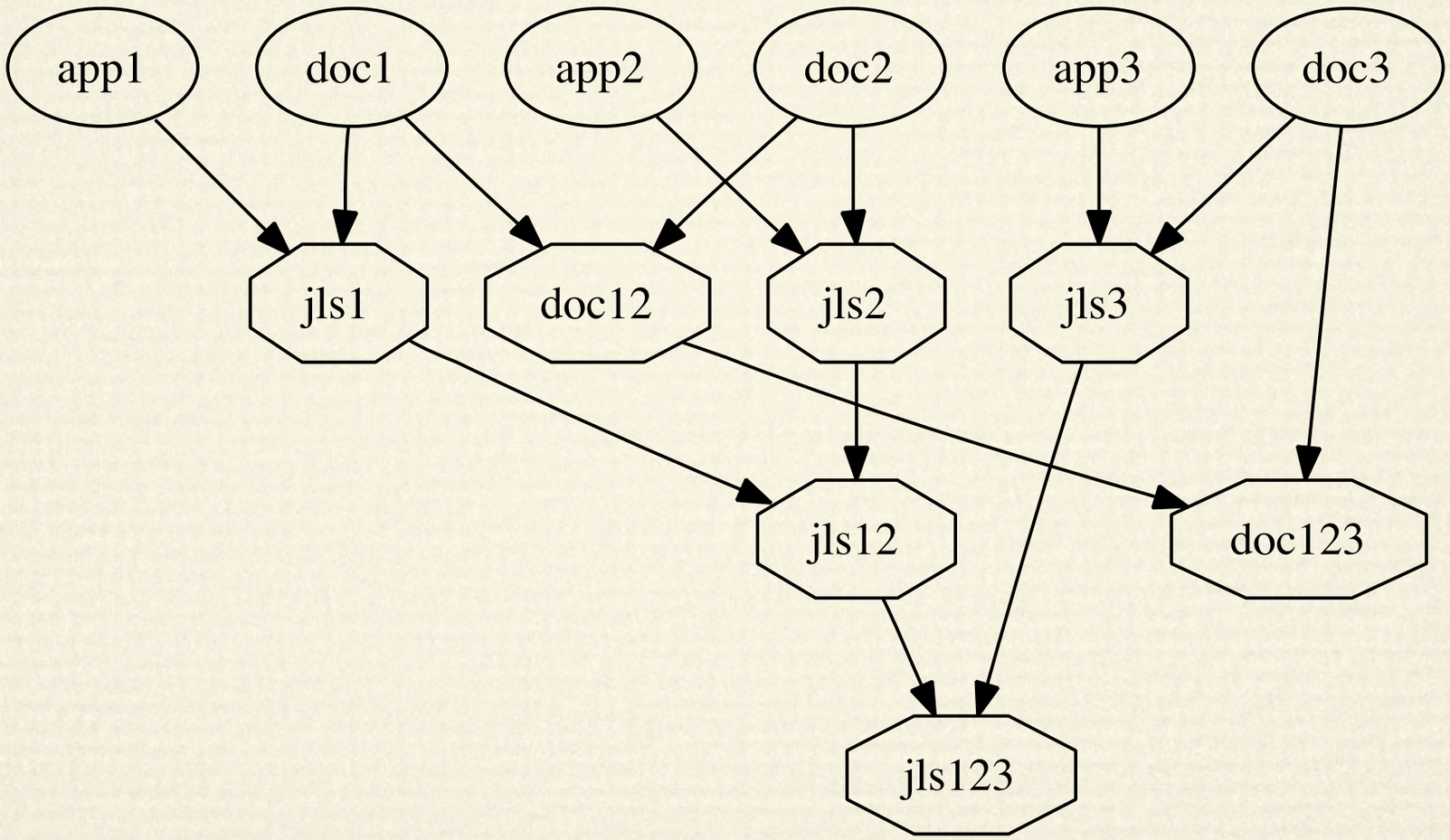
- SDF basics:
  - SDF=Syntax Def. Formalism
  - SDF has S–G–LR as semantics.
  - Computations over SDF:
    - ASF
    - Stratego
    - ...
- Extractor option:
  - Use SDF of SDF.
  - Use ASF over it.
  - Construct BGF via XML.

```
[transform-a-production]
&C1 := sort2chardata(&N1),
&E2 := trafoSymbols(&Ss1)
=======================
trafoProd ( &Ss1 -> &N1 &As1 ) =
<bgf:production>
 <nonterminal>&C1</nonterminal>
 &E2
</bgf:production>

[transform-empty-definition-of-nonterminal]
trafoSymbols() =
<bgf:expression>
 <epsilon/>
</bgf:expression>

[transform-definition-that-is-not-a-sequence]
trafoSymbols(&S1) = trafoSymbol(&S1)

[transform-a-definition-that-is-a-nontrivial-sequence]
&S1 &S2 &S*3 := &Ss1,
&C*1 := mapTrafoSymbol(&Ss1)
=======================
trafoSymbols(&Ss1) =
<bgf:expression>
 <sequence>
  &C*1
 </sequence>
</bgf:expression>
```

# Available extractors

- ✓ ANTLR
- ✓ SDF
- ✓ DCG
- ✓ Java object models
- ✓ XML Schemas
- ✓ Language specifications
- ✓ …

# Applying grammar convergence to the Java Language Specification




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

# Basic properties of the JLS sources

| | **Grammar class** | **Iteration style** |
|---|---|---|
| *app1* | LALR(1) | left-recursive |
| *doc1* | none | left-recursive |
| *app2* | unclear | EBNF |
| *doc2* | none | left-recursive |
| *app3* | “nearly” LL(k) | EBNF |
| *doc3* | none | left-recursive |

| | 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 |

# Grammar extraction for JLS

★ 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 |

# 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* | 347 | 152 | 1 | 7 |
| *doc123* | 440 | 201 | 1 | 7 |

# Grammar comparison

★ Compare grammars structurally.

★ Apply simple algebraic laws on grammars.

★ Provide suggestive input for transformation.

# Grammar transformation

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

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

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

Need to project away “(“ & “)”

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

- ★ Semantics–preserving (refactoring)
  - ✦ rename, introduce, eliminate
  - ✦ fold, unfold, extract, inline
  - ✦ factor, distribute, horizontal, vertical
  - ✦ yaccify, deyaccify, massage
  - ✦ designate, unlabel
  - ✦ ...

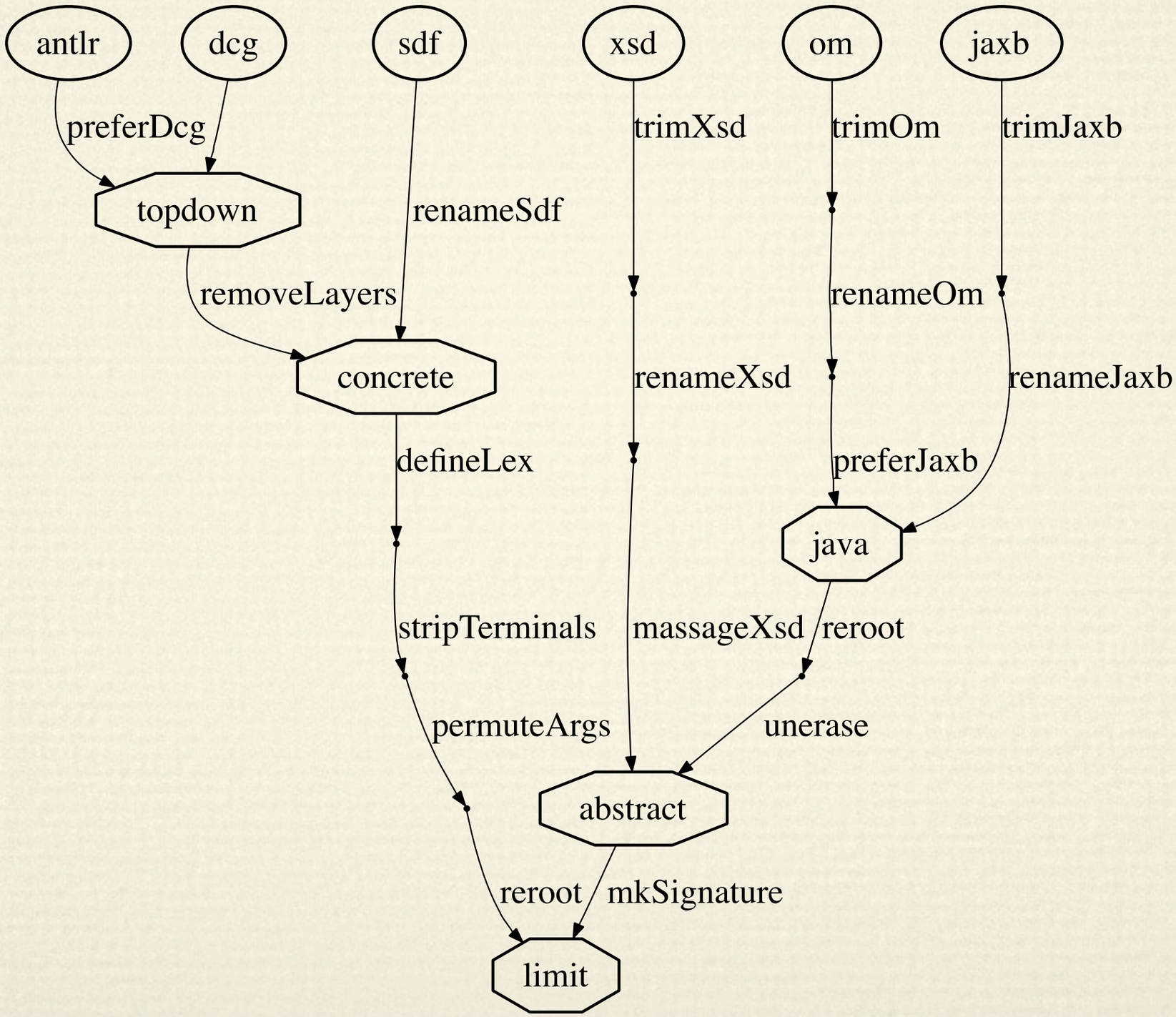
# XBGF Operator Suite

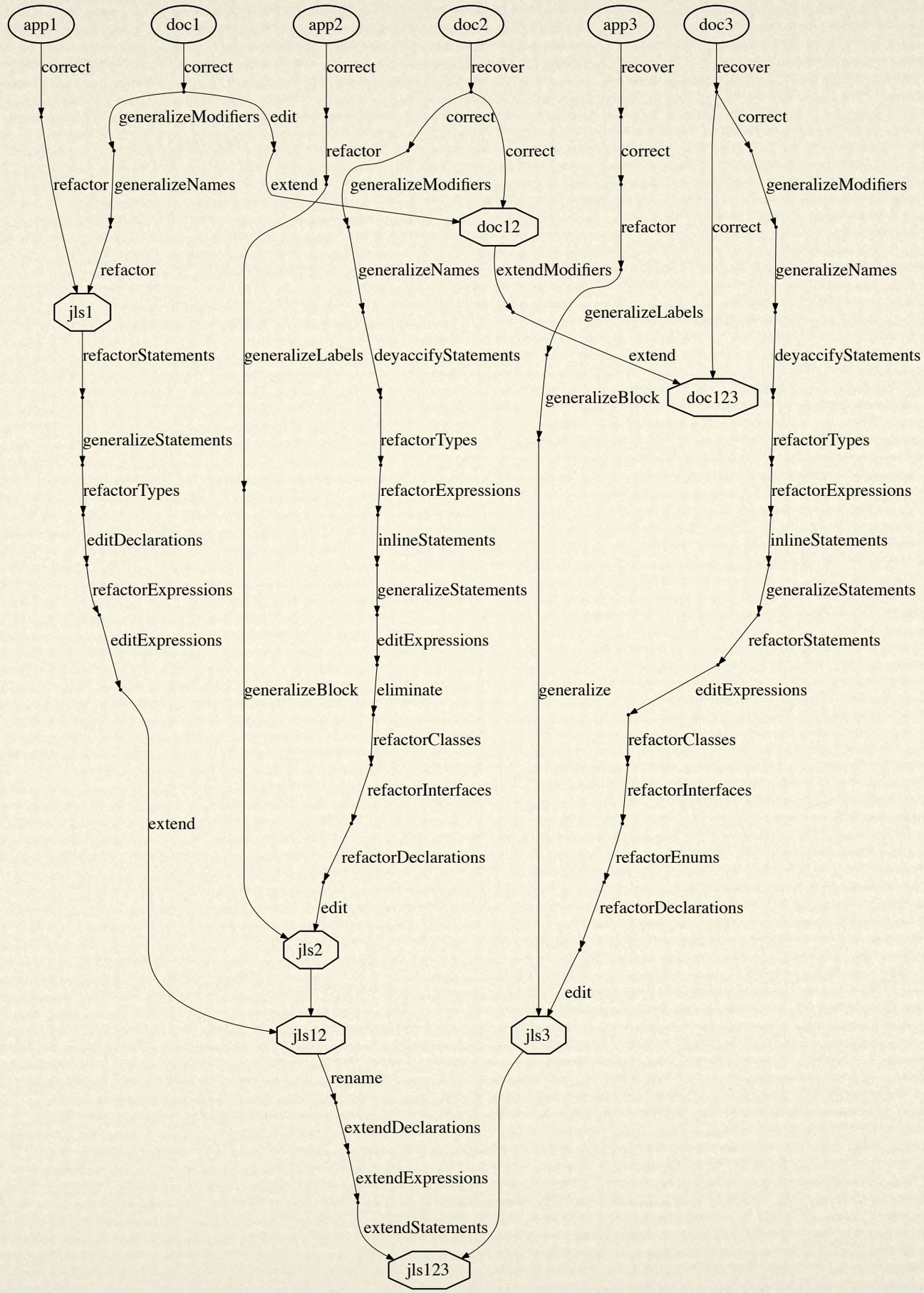
- ★ Semantics–increasing/–decreasing
  - ✦ appear, disappear
  - ✦ narrow, widen
  - ✦ add, remove
  - ✦ upgrade, downgrade
  - ✦ unite
  - ✦ ...

# XBGF Operator Suite

- ★ Semantics–revising
  - ✦ undefine, define, redefine
  - ✦ inject, project, permute
  - ✦ abstractize, concretize
  - ✦ replace

# A more detailed convergence tree




app1
doc1
app2
doc2
app3
doc3
correct
correct
correct
recover
recover
recover
generalizeModifiers
edit
refactor
correct
correct
correct
refactor
generalizeNames
extend
generalizeModifiers
doc12
refactor
correct
generalizeModifiers
refactor
generalizeNames
extendModifiers
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
refactorEnums
refactorDeclarations
refactorDeclarations
edit
jls2
edit
jls12
jls3
rename
extendDeclarations
extendExpressions
extendStatements
jls123

# Transformation statistics for JLS

| | **jls1** | **jls2** | **jls3** | **jls12** | **jls123** | **doc12** | **doc123** | **Total** |
|---|---|---|---|---|---|---|---|---|
| Number of lines | 600 | 4807 | 9469 | 4285 | 2934 | 1491 | 3072 | 26658 |
| Number of transformations | 62 | 367 | 538 | 287 | 120 | 70 | 133 | 1577 |
| ○ semantics-preserving | 40 | 278 | 398 | 235 | 87 | 25 | 73 | 1136 |
| ○ semantics-increasing or -decreasing | 22 | 78 | 127 | 50 | 32 | 38 | 56 | 403 |
| ○ semantics-revising | — | 11 | 13 | 2 | 1 | 7 | 4 | 38 |
| Number of issues | 8 | 38 | 47 | 25 | 17 | 32 | 40 | 207 |
| ○ recoveries | — | 7 | 8 | — | — | 7 | 4 | 26 |
| ○ corrections | 5 | 22 | 22 | 2 | — | 10 | 7 | 68 |
| ○ extensions | — | — | — | 17 | 14 | 15 | 28 | 74 |
| ○ optimizations | 3 | 9 | 17 | 6 | 3 | — | 1 | 39 |

# Conclusion and future work

★ Synchronise scattered grammar knowledge

★ Further consolidation of operator suite

★ Co–transformation of parse–trees possible

★ Semi–automatic approach desirable

★ Additional techniques for priorities

★ Alignment with metamodeling–based work

# Thank you!

★ Questions?

★ Comments?

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