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

Grammar convergence framework

★ 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
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(−)))  
])
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

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

★ Compare grammars structurally.
★ Apply simple algebraic laws on grammars.
★ Provide suggestive input for transformation.
How grammar convergence works

Grammar artifacts

Compare grammars for structural equality

[equal] [not equal]

Pick difference

Pick grammar

[language-preserving] [language-revising]

[...-increasing] [...-decreasing]

Refactor grammar
Lengthen grammar
Shorten grammar
Edit grammar

Transformations

Grammars
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?

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;

abridge

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

expr : ...
expr : expr;

unite
XBGF Operator Suite

Semantics–preserving (refactoring)

- rename, import, introduce, eliminate
- fold, unfold, extract, inline
- factor, distribute, horizontal, vertical
- yaccify, deyaccify, massage
- designate, unlabel
- ...

$L(G_1) = L(G_2)$

Productions

Nonterminals

Tops

Bottoms

app1

app2

app3

doc1

doc2

doc3

Figure 1: Metrics for the JLS grammars.
Semantics—increasing/—decreasing
- appear, disappear
- narrow, widen
- add, remove
- upgrade, downgrade
- unite
- ...

\[ L(G_1) \subseteq L(G_2) \]
\[ \lor \]
\[ L(G_2) \subseteq L(G_1) \]
XBGF Operator Suite

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

$L(G_1) \not\subseteq L(G_2)$
\land
$L(G_2) \not\subseteq L(G_1)$
Grammar refactoring example

BGF \((doc2)\)

\[
\begin{align*}
\text{ClassBodyDeclarations:} & \text{ ClassBodyDeclaration} \\
\text{ClassBodyDeclarations:} & \text{ ClassBodyDeclarations ClassBodyDeclaration} \\
\text{ClassBody:} & \text{ ClassBodyDeclarations ? } \\
& \text{ "\{ " ClassBodyDeclarations * "\}"
\end{align*}
\]

XBGF \(\text{grammar refactoring}\)

\[
\begin{align*}
\text{deyaccify(ClassBodyDeclarations);} \\
\text{inline(ClassBodyDeclarations);} \\
\text{massage(} \\
\text{ ClassBodyDeclaration + ?,} \\
\text{ ClassBodyDeclaration * \);} \\
\end{align*}
\]
## Grammar extension example

### BGF (*doc2*)

<table>
<thead>
<tr>
<th>ClassModifier:</th>
<th>FieldModifier:</th>
<th>MethodModifier:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;public&quot;</td>
<td>&quot;public&quot;</td>
<td>&quot;public&quot;</td>
</tr>
<tr>
<td>&quot;protected&quot;</td>
<td>&quot;protected&quot;</td>
<td>&quot;protected&quot;</td>
</tr>
<tr>
<td>&quot;private&quot;</td>
<td>&quot;private&quot;</td>
<td>&quot;private&quot;</td>
</tr>
<tr>
<td>&quot;abstract&quot;</td>
<td>&quot;static&quot;</td>
<td>&quot;abstract&quot;</td>
</tr>
<tr>
<td>&quot;static&quot;</td>
<td>&quot;final&quot;</td>
<td>&quot;static&quot;</td>
</tr>
<tr>
<td>&quot;final&quot;</td>
<td>&quot;transient&quot;</td>
<td>&quot;final&quot;</td>
</tr>
<tr>
<td>&quot;strictfp&quot;</td>
<td>&quot;volatile&quot;</td>
<td>&quot;synchronized&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;native&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;strictfp&quot;</td>
</tr>
</tbody>
</table>

### XBGF (grammar optimisation)

```java
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

ANTLR → DCG → Topdown → Concrete → Abstract → Limit

Prefer DCG → Remove Layers → Concrete → Define Lex → Strip Terminals → Permute Args → Reroot / MKSignature

Trim XSD → Rename SDF → Concrete → Define Lex → Strip Terminals → Permute Args → Reroot

Trim OM → Rename OM → Concrete → Define Lex → Strip Terminals → Permute Args → Reroot

Trim JAXB → Rename JAXB → Concrete → Define Lex → Strip Terminals → Permute Args → Reroot
2.2 Simple grammar metrics

<table>
<thead>
<tr>
<th>Productions</th>
<th>Nonterminals</th>
<th>Tops</th>
<th>Bottoms</th>
</tr>
</thead>
<tbody>
<tr>
<td>app1</td>
<td>282</td>
<td>135</td>
<td>1</td>
</tr>
<tr>
<td>doc1</td>
<td>315</td>
<td>148</td>
<td>1</td>
</tr>
<tr>
<td>app2</td>
<td>185</td>
<td>80</td>
<td>6</td>
</tr>
<tr>
<td>doc2</td>
<td>346</td>
<td>151</td>
<td>1</td>
</tr>
<tr>
<td>app3</td>
<td>245</td>
<td>114</td>
<td>2</td>
</tr>
<tr>
<td>doc3</td>
<td>435</td>
<td>197</td>
<td>3</td>
</tr>
</tbody>
</table>

2.3 The source format for grammars

There should be only one top nonterminal, the actual start symbol of the Java grammar. Any additional case of an “unused” nonterminal does not make sense. At first glance, there should be only one top nonterminal, the actual start symbol of the Java grammar.

The decrease of numbers for the step from jls1 to jls2 is mainly implied by the different grammar optimizations—revising semantics—increasing or—decreasing semantics—preserving semantics. The obvious trend is that the numbers of productions and nonterminals go up with the version number. In particular from grammars and versions—we collect bits from different documents and sections—in the HTML format here.

The grammar format slightly varies for the different JLS formats. The JLS is available electronically in HTML and PDF format. Neither of these formats was designed with convenient access to the grammars in mind. We will focus on the HTML format here.

2.4 The JLS document is basically a structured text document with embedded grammar sections. The major differences between the numbers of productions and nonterminals for the two grammars of any given version (see Fig. 2) are recoveries. The decrease of numbers for the step from jls1 to jls2 is mainly implied by the different grammar optimizations—revising semantics—increasing or—decreasing semantics—preserving semantics.

The metrics were automatically derived from the extracted grammars. Terminology: a nonterminal that is defined but never used; a nonterminal that is used but never defined; a nonterminal that is defined and used.

We list all spotted grammar class claims—we also add observations about iteration style “lists” that we made during cursory examination. This table makes clear that we need to bridge the gap between different grammar classes—which is more involved—but also relatively simple iteration styles—which is relatively simple but also unclear.

2.5 Figure 3: Effort measurements per target in the convergence graph for the JLS. Figure 4: Metrics for the transformed grammars.
Transformation statistics for JLS

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

Software Language Processing Suite is here: http://slps.sf.net/