Maintenance and Evolution of Grammarware by Grammar Transformation

IPA Spring Days on Model-Driven Software Engineering
Vadim Zaytsev, SWAT, CWI
© 2012
Grammarware
import types.*;
import org.antlr.runtime.*;
import java.io.*;
public class TestEvaluator {
    public static void main(String[] args) throws Exception {
        ANTLRFileStream input = new ANTLRFileStream(args[0]);
        FLLexer lexer = new FLLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        FLParser parser = new FLParser(tokens);
        Program program = parser.program();
        input = new ANTLRFileStream(args[1]);
        lexer = new FLLexer(input);
        tokens = new CommonTokenStream(lexer);
        parser = new FLParser(tokens);
        Expr expr = parser.expr();
        Evaluator eval = new Evaluator(program);
        int expected = Integer.parseInt(args[2]);
        assert expected == eval.evaluate(expr); 
    }
<?xml version="1.0" encoding="UTF-8"?>
<bgf:grammar xmlns:bgf="http://planet-sl.org/bgf">
  <root>Program</root>
  <root>Fragment</root>
  <bgf:production>
    <nonterminal>Program</nonterminal>
    <bgf:expression>
      <plus>
        <bgf:expression>
          <selectable>
            <selector>function</selector>
            <bgf:expression>
              <nonterminal>Function</nonterminal>
            </bgf:expression>
          </selectable>
        </bgf:expression>
      </plus>
    </bgf:expression>
  </bgf:production>
</bgf:grammar>
From languages to grammars

• Grammar
  • finite formal definition of a language
  • defines the structure of allowed language instances

• Classical definition
  • nonterminals, terminals, production rules
  • statement ::= “if” expression “then” statement

• Grammarware
  • grammar-based software
Grammar example (EBNF)

```
compilationUnit ::=
    topLevelDefinition* EOF

topLevelDefinition ::=  
    classDefinition
    interfaceDefinition
    functionTypeAlias
    functionSignature functionBody
    returnType? getOrSet identifier formalParameterList functionBody
    "final" type? staticFinalDeclarationList ";"
    variableDeclaration ";"

classDefinition ::= 
    "class" identifier typeParameters? superclass? interfaces? "{" memberDef* "}"

typeParameters ::= 
    "<" typeParameter ("","" typeParameter)* ">"

superclass ::= 
    "extends" type

interfaces ::= 
    "implements" typeList
```
“Grammar” (syntax diagram)
“Grammar” (parser spec)

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)}
“Grammar” (metamodel)

- Program
  - function : Function
- Function
  - name : EString
  - argument : Argument
  - definition : Exp
- Argument
  - name : EString
- LiteralExp → Exp
- ArgumentExp → Exp
- IfThenElseExp → Exp
- ApplyExp → Exp
- BinaryExp → Exp
- PlusExp → BinaryExp
- MinusExp → BinaryExp
- EqualExp → BinaryExp

- Function
  - name : EString
  - argument : Argument
  - definition : Exp
- Argument
  - name : EString
- LiteralExp → Exp
  - (↑) Exp
  - value : EInt
- ArgumentExp → Exp
  - (↑) Exp
  - argument : Argument
- IfThenElseExp → Exp
  - (↑) Exp
  - if : Exp
  - then : Exp
  - else : Exp
- ApplyExp → Exp
  - (↑) Exp
  - function : Function
  - argument : Exp
- BinaryExp → Exp
  - (↑) Exp
  - left : Exp
  - right : Exp
“Grammar” (relation diagram)
Grammarware examples

- Parser
- Compiler
- Interpreter
- Pretty-printer
- Scanner
- Browser
- Static checker
- Structural editor
- IDE
- DSL framework
- Preprocessor
- Postprocessor
- Model checker
- Refactorer
- Code slicer
- API
- XMLware
- Modelware
- Language workbench
- Reverse engineering tool
- Benchmark
- Recommender
- Renovation tool
Grammar Transformations
Motivation

• Why transform?
  • Grammar adaptation
  • Grammar beautification
  • Inconsistency management
  • Version control

• Documented, well-understood, compositional change

• Any difference can be a transformation

• Good for representing relationships
Transformations

- Transparent
  - Full automation
  - Happening behind the scenes
  - Usually optimisations

- Programmable
  - Full control
  - Manually programmed
  - Generated from other artefacts
Transformation components

```
<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>f(a)</th>
<th>G'</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Transformation components

- known semantics, well-defined algorithm
- rename, fold, factor, inject, remove, ...

Operator

G \rightarrow F \rightarrow G'
Transformation components

Arguments

• what exactly to rename/factor/inject/...?
Transformation components

Input grammar

• determines applicability
Transformation components
Transformation components

- **Operator**
  - known semantics, well-defined algorithm
  - rename, fold, factor, inject, remove, ...

- **Arguments**
  - what exactly to rename/factor/inject/...?

- **Input grammar**
  - determines applicability
Example 1:
all three components
Example 1: all three components

• Suppose we know the operator(s), the argument(s), the input

• We can execute the transformation
  • obtain the transformed grammar automatically

• We can verify applicability

• We can coevolve language instances
  • transform both the grammar and trees conforming to it

• We can test transformations with constraints
  • change impact analysis
Grammar refactoring

BGF (read2)

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

XBGF (grammar refactoring)

deypaccify(ClassBodyDeclarations);
inline(ClassBodyDeclarations);
massage(
  ClassBodyDeclaration+? ,
  ClassBodyDeclaration* );
Example 2: just operators
Example 2: just operators

• Suppose we know the operator(s) used in the script
• We do not know/care about their arguments
• We do not know/care about the input grammar
• We still know the semantics
  • $\Rightarrow$ we know certain properties of the transformation
  • $\Rightarrow$ we know the relationship between input & output
## Java grammar convergence

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

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

- We can derive arguments after seeing the grammar

- Grammar mutation
  - Disciplined rename (switch naming convention)
  - Remove all terminal symbols (minimalistic implode)
  - Reroot to top (if starting symbol is undefined/wrong)
  - Eliminate top (remove unconnected components)
  - Extract subgrammar (isolate one component)
  - Remove lazy nonterminals (inline or unchain)
  - Deyaccify all yaccified production rules (A:B; A:AB;)

Tough stuff
TS1: Grammar recovery

- Extraction by abstraction
- Notation-parametric automation
- Many bugs are fixed automatically, but not all
- Documentation is incomplete, incorrect, inconsistent
- Existing grammars smell bad
Grammar revision

BGF \((impl2, impl3)\)

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

XBGF (grammar correction)

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

TS2: Grammar convergence

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

A ::= X Y Z;
B ::= X Y Z;
A ::= X G Z;
A ::= X Y Z;
A ::= X Y Z;
B ::= X Y Z;
TS3: Grammar product lines

- Usual framework:
  - baseline grammar
  - transformation scripts to derive other grammars

- If the baseline grammar changes
  - reapply transformations (modulo applicability fixes)

- If a derived grammar changes
  - reestablish relationships with guided convergence
Guided grammar convergence
Guided convergence of FL

<table>
<thead>
<tr>
<th>Issue</th>
<th>antlr</th>
<th>dcg</th>
<th>sdf</th>
<th>rascal</th>
<th>txl</th>
<th>.ecore</th>
<th>.ecore2</th>
<th>xsd</th>
<th>jaxb</th>
<th>om</th>
</tr>
</thead>
<tbody>
<tr>
<td>One to many nonterminals</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>Nominal mismatches</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>More liberal definitions</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Superfluous nonterminals</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>Disconnected nonterminals</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Maximum number of versions</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Chain production rules</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Permutations</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>±</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Reflexive chain rules</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>Undefined matched as...</td>
<td>ε</td>
<td>ε</td>
<td>ε</td>
<td>ε</td>
<td>ε</td>
<td>φ</td>
<td>ε</td>
<td>ε</td>
<td>ε</td>
<td>ε</td>
</tr>
<tr>
<td>Aggregation</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>Layered definitions</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>Meaningful chain rules</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

Table 1. Different issues of guided grammar convergence in the case study: columns correspond to the grammars of FLs, rows in the upper part relate to subsections of §4, the bottom part is for special cases discussed in §5. We see that some issues arise almost everywhere, like nominal nonterminal mismatches or matching undefined nonterminals as $\varepsilon$s, while others persist only in a specific group of grammars, for example, superfluous nonterminals in FL always refer to function separating newliness so they need to be explicitly discarded only for concrete syntax definitions. Disconnected nonterminals unsurprisingly occur in grammars that are extracted from generated artefacts, they are framework traces like Visitor or ObjectFactory. Permutations are commonly found in abstract syntax definitions. The most problematic grammar turned out to be the one extracted from the handcrafted Ecore models and the most problematic feature for guided grammar convergence was layered definitions with explicitly hardcoded priorities. We do not have enough data to draw any conclusions about whether this was due to the peculiar structure of the grammars themselves or about how common such problems will occur in other case studies. However, at this point we can already conclude that guided grammar convergence is at least as efficient as normal grammar convergence and it both employs more automation and gives more opportunities for further automation.
To summarise

• Software languages are everywhere
• Grammars are finite executable descriptions
• Grammarware is software based on a grammar
• Transformations are grammar differences
To summarise

- Programmable transformation
  - operator
  - arguments
  - input grammar
To summarise

- Operator + arguments + grammar $\Rightarrow$ verify, execute, coevolve, ...
- Operators $\Rightarrow$ relationship
- Operators + grammar $\Rightarrow$ grammar mutation
To summarise

- Grammar recovery:
  - notation-parametric
  - correction + smell removal

- Grammar convergence:
  - take related grammars
  - transform until equal

- Grammar product lines:
  $\rightarrow$ transformations
  $\leftarrow$ guided coevolution
Bibliography


Questions

vadim@grammarware.net