Background

★ BX 2012: First International Workshop on Bidirectional Transformations (satellite event of ETAPS 2012)

★ “Language Evolution, Metasyntactically”

★ Submitted to BX 2012: 21 December 2011

★ Notification: 24 January 2012

★ Camera-ready copy: 5 February 2012

★ Conditionally accepted with 2×SA and 2×WR
Bidirectional transformations community

A cross-discipline field:

- Model-Driven Software Development: sync views
- Graphical User Interfaces: model-view-controller
- Visualization with Direct Manipulation: animation
- Relational Databases: updatable views
- Data Transformation, Integration and Exchange: map data and merge it
- Data Synchronizers: bridge the gap between replicas in different formats
- Macro Systems: give feedback in terms of original program elements
- Domain-Specific Languages: runtime mapping in embedded interpreters
- Structure Editors: interfaces for editing complicated data sources
- Serializers: map external data to structured objects

Czarnecki, Foster, Hu, Lämmel, Schürr, Terwilliger, GRACE
Introduction
Introduction

★ Every language document employs its own notation
★ We focus on metalanguage evolution
  ★ the language itself does not evolve
  ★ the notation in which it is written, does
★ We limit ourselves to grammarware technical space
★ Working prototypes are a part of SLPS
Motivating example
LLL in itself [LDTA’02]

grammar    : rule+;
rule       : sort "::" alts ";";
alts       : alt alts-tail*;
alts-tail  : "|" alt;
alts-tail  : "|" alt;
alt        : term*;
term       : basis repetition?;
basis      : literal | sort;
repetition : "*" | "+" | "?";
LLL in itself [GDK]

specification  :  rule+;
rule           :  ident "::" disjunction ";";
disjunction    :  { conjunction "|" }+;
conjunction    :  term*;
term           :  basis repetition?;
basis          :  ident | literal 
                |  alternation |  group;
repetition     :  "+" | "*" | "?";
alternation    :  "{" basis basis "}" repetition;
group          :  "(" disjunction ")" ;
<table>
<thead>
<tr>
<th>Metasymbol</th>
<th>Definition</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>defining metasymbol</td>
<td>:</td>
<td></td>
<td>definition separator</td>
</tr>
<tr>
<td>terminator metasymbol</td>
<td>;</td>
<td></td>
<td>postfix optionality</td>
</tr>
<tr>
<td>postfix star metasymbol</td>
<td>*</td>
<td></td>
<td>postfix plus metasymbol</td>
</tr>
<tr>
<td>start terminal metasymbol</td>
<td>&quot;</td>
<td></td>
<td>end terminal metasymbol</td>
</tr>
</tbody>
</table>
Δ between LLL1 and LLL2

<table>
<thead>
<tr>
<th>start group metasymbol</th>
<th>(</th>
<th>end group metasymbol</th>
<th>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>start separator list star metasymbol</td>
<td>{</td>
<td>end separator list star metasymbol</td>
<td>}*</td>
</tr>
<tr>
<td>start separator list plus metasymbol</td>
<td>{</td>
<td>end separator list plus metasymbol</td>
<td>}+</td>
</tr>
</tbody>
</table>
Metasyntactic evolution megamodel
Judging by how terminal symbols are written in the LLL definitions themselves, we can conclude that terminal symbols are supposed to be double quoted in both versions:

```
start terminal metasymbol
"end terminal metasymbol
```

Features new to L with respect to L:

- are grouping of symbols and separator lists:
  - start group metasymbol
  - (end group metasymbol
  - start separator list
  - start separator list
  - start separator list

Since the introduction of these features is precisely the linguistic difference between the two versions, it represents language evolution. In the remainder of the paper, we aim at investigating how the changes we observe in these notation specifications are related to various coevolving artefacts; how such changes can be propagated through the system; and how far we can push automation of these activities.

### 3.1 Grammar internal representation

For this paper, we used a slightly enhanced dialect of BGF, a Grammar Format inherited from [LZ.92:999]. Its logic programming-based specification follows:

```
grammar(Rs,Ps) ⇐ mapoptlist(n,Rs), maplist(prod,Ps).
prod(p(L,N,X)) ⇐ mapopt(label,L), atom(N), expr(X).
label(l(X)) ⇐ atom(X).
expr(true).
expr(fail).
expr(a).
expr(t(T)) ⇐ atom(T).
expr(n(N)) ⇐ atom(N).
expr(’,(Xs)) ⇐ maplist(expr,Xs).
expr(’,,(Xs)) ⇐ maplist(expr,Xs).
expr(’?,(X)) ⇐ expr(X).
expr(’∗,(X)) ⇐ expr(X).
expr(’+,(X)) ⇐ expr(X).
expr(slp(X,Y)) ⇐ expr(X), expr(Y).
expr(sls(X,Y)) ⇐ expr(X), expr(Y).
expr(s(S,X)) ⇐ atom(S), expr(X).
```

- grammar = start symbols + productions
- production = label + lhs + rhs
- production labels
- ε
- empty language
- universal type
- terminal symbols
- nonterminal symbols
- sequential composition
- choice
- optionality
- Kleene star
- transitive closure
- Y-separated list with 1 or more elements
- Y-separated list with 0 or more elements
- selectable expressions
Toward bidirectional grammar transformation

★ XBGF ⇒ EBGF:

★ renameN, factor, etc: flip arguments
★ addV/removeV, narrow/widen: form pairs
★ extract/inline, unlabel/designate: asymmetry
★ distribute: removed from the language
★ unite, equate: tricky, superposition of others

★ BX is a stable way to represent grammar relationship
Toward transformable notation specification

★ EDD [Z12a]

★ confix metaconstructs
★ infix, prefix, postfix metasymbols
★ predefined sets (e.g., built-in nonterminals)
★ conventions (e.g., naming, whitespace reliability)

★ XEDD:

★ rename-metasymbol(s, v1, v2)
★ introduce-metasymbol(s, v)
★ eliminate-metasymbol(s)
Toward in-notation grammar transformation

★ Concrete syntax transformations
★ Avoiding discussion on propagation of CTS elements
★ bgfreformat tool:
  ★ extract the grammar from the given notation
  ★ manipulate (transform) the internal representation
  ★ pretty-print the grammar in the desired notation
★ Alternatively, parse with grammar for grammars
Grammar transformation vs. grammar mutation

★ A grammar transformation operator $\tau$ can be formalised as a triplet:

$\tau = \langle c_{pre}, t, c_{post} \rangle$.

★ A grammar transformation then is $\tau_{a_i}(G)$, resulting in $G'$.

★ if $a_i$ are of incorrect types and quantity than expected by $t$
   $\Rightarrow \tau$ is incorrectly called;

★ if the constraint $c_{pre}$ does not hold on $G$
   $\Rightarrow \tau_{a_i}$ is inapplicable to $G$;

★ if the constraint $c_{post}$ holds on $G$
   $\Rightarrow \tau_{a_i}$ is vacuous on $G$;

★ if the constraint $c_{pre}$ holds on $G$ and $c_{post}$ does not hold on $G'$
   $\Rightarrow t$ is incorrectly implemented;

★ if $c_{pre}$ holds on $G$, $c_{post}$ holds on $G'$
   $\Rightarrow \tau$ has been applied correctly with arguments $a_i$ to $G$ resulting in $G'$.
Grammar transformation VS. grammar mutation

★ A grammar mutation does not have a single precondition

★ It has a set of preconditions that serve as triggers:
\[ \mu = \langle \{c_i\}, \{t_i\}, c_{post} \rangle. \]

★ The mutation terminates once no trigger \( c_i \) holds and the postcondition \( c_{post} \) is met.

★ A bidirectional grammar mutation:
\[ \mu_{bx} = \langle c_{pre}, \{c_i\}, \{t_i\}, c_{post} \rangle \]
will be an instantiation of a grammar mutation

★ The family of spawned BMs does not define the original:
i.e., \( \forall \mu \exists G \exists G' \not\exists \mu_{bx}, G' = \mu(G) \land G' = \mu_{bx}(G) \land G = \mu^{-1}(G') \).
The megamodel
Notation evolution summary

To conclude, a notation evolution step $\Delta$ consists of the following components:

- $\sigma$, a bidirectional notation specification transformation that changes the notation itself
- $\delta$, a bidirectional coupled grammar transformation that converges the notation grammars
- $\mu$, an unidirectional coupled grammar mutation that migrates the grammarbase according to notation changes.
- a mechanism to propagate naming changes to form $\gamma = \delta^{-1} \circ \beta$

• $\sigma$, a bidirectional notation specification transformation that changes the notation itself
• $\delta$, a bidirectional coupled grammar transformation that converges the notation grammars
• $\mu$, an unidirectional coupled grammar mutation that migrates the grammarbase according to notation changes.
• a mechanism to propagate naming changes to form $\gamma = \delta^{-1} \circ \beta$
LLL case study
The megamodel
The megamodel
$$\Delta = \langle \sigma, \delta, \mu \rangle$$

The $\sigma$ between $L_1^3$ and $L_2^3$, expressed in XEDD, looks like this (see $1111to2.xedd$):

```
introduce-metasyMBOL (group, '(', ')');
introduce-metasyMBOL (seplist-star, '{', '}' +');
introduce-metasyMBOL (seplist-plus, '{', '} +');
```

The coupled $\delta$ generated by the `xedd` processor produces the following $\Xi$BGF:

```
rename-rename (LLL1Grammar, LLL2genGrammar);
rename-rename (LLL1Production, LLL2genProduction);
rename-rename (LLL1Definition, LLL2genDefinition);
rename-rename (LLL1Symbol, LLL2genSymbol);
rename-rename (LLL1Nonterminal, LLL2genNonterminal);
rename-rename (LLL1Terminal, LLL2genTerminal);
add-remove(p(l(group), LLL2genSymbol, ','(t(''),slp(LLL2genDefinition,'|')'),t(''))));
add-remove(p(l(sepliststar), LLL2genSymbol, ','(t('{'),n(LLL2genSymbol),n(LLL2genSymbol),t('}{', '*'))));
add-remove(p(l(seplistplus), LLL2genSymbol, ','(t('{'),n(LLL2genSymbol),n(LLL2genSymbol),t('}{', '+'))));
```

Propagation of nominal refactorings from $\delta$ to $\beta$ to form $\gamma$ is performed by an XSLT script $\xi_{bgf}^2$. In general, propagating structural changes is hard and sometimes impossible (for some transformations, there is no easy way to express their permutation in XBGF), and in this particular scenario is even undesirable. We save space in the paper by reserving it for future work.
Applying coupled mutation
eliminate-metasymbol(group)
to Grammar Zoo

<table>
<thead>
<tr>
<th>Language</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ada-kellogg</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>ada-kempe</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>ada-laemmel-verhoef</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>ada-lncs-2219</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>ada-lncs-4348</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>c-iso-9899-1999</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c-iso-9899-tc2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c-iso-9899-tc3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>cpp-iso-14882-1998</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>cpp-iso-n2723</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>csharp-ecma-334-1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>csharp-ecma-334-2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>csharp-ecma-334-3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>csharp-ecma-334-4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>csharp-isoo-23270-2003</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>csharp-isoo-23270-2006</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>csharp-msft-1.0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>csharp-msft-1.2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>csharp-msft-3.0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>dart-google</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>dart-spec-0.01</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>dart-spec-0.05</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>eiffel-bezault</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>eiffel-iso-25436-2006</td>
<td>345</td>
<td></td>
</tr>
<tr>
<td>fortran-derricks</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>java-1-jls-read</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>java-2-jls-impl</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>java-2-jls-read</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>java-5-habelitz</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>java-5-jls-impl</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>java-5-jls-read</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>java-5-parr</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>java-5-stahl</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>java-5-studman</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>mediawiki-bnf</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>mediawiki-ebnf</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>modula-sdf</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>modula-src-052</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>w3c-xpath1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Related and future work
Related work

★ Cicchetti et al: coevolution of models/metamodels (syntax/metasyntax) with language evolution and language coevolution happening simultaneously.

★ Wachsmuth: MDA/MOF solution, close to us.

★ Stevens: formulated properties like correctness and hippocraicness; need further investigation.
Future work

- Extract reference grammars from compiler sources
  - rare enabling precondition
  - known to be successful at least once [C500LP]
- Derive $\beta$ from inline edits of the definition “in itself”
  - possible if edits are purely decorational
  - makes sense in context of IDE (structural editors?)
- Propagate all refactorings from $\delta$ to $\beta$ to form $\gamma$
Conclusion
Conclusion

★ We extended XBGF to bidirectionality, resulting in EBGF.

★ We proposed EDD and XEDD for notation & its evolution.

★ We presented a case study of LLL evolution (GDK).

★ We generalised transformers and generators to transformations and mutations; also formalised them.

★ We implemented an XEDD processor for evolution, coevolution, change propagation and mutation.
Bibliography
Bibliography

[THIS]
Zaytsev,
*Language Evolution, Metasyntactically*
BX and hopefully ECEASST

[GRACE]
Czarnecki, Foster, Hu, Lämmel, Schürr, Terwilliger,
*Bidirectional Transformations: A Cross-Discipline Perspective*
GRACE 2008 meeting notes, state of the art, and outlook

[C500LP]
Lämmel, Verhoef,
*Cracking the 500-Language Problem*
IEEE Software, November/December 2001

[Z12a]
Zaytsev,
*BNF WAS HERE: What Have We Done About the Unnecessary Diversity of Notation for Syntactic Definitions*
SAC/PL 2012

[LDTA02]
Kort, Lämmel, Verhoef,
*The Grammar Deployment Kit: System Demonstration*
LDTA 2002 and ENTCS 65

[GDK]
Kort,
*Grammar Deployment Kit Reference Manual*
Universiteit van Amsterdam, May 2003
Discussion