Language Evolution, Metasyntactically
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
Metalanguage evolution

- Notation correction
  - Misused notation
  - Overused notation
- Notation evolution
  - Notations are languages, they evolve
- Mapping between notations
  - Many notations are equivalent
  - Most are (almost EBNF) + (little extra)
Metasyntactic evolution megamodel
Megamodel

Syntactic notation specification
EBNF Dialect Definition

- List of indicators
- Together form a notation specification

Zaytsev, *What Have We Done About the Unnecessary Diversity of Notation for Syntactic Definitions*, SAC/PL 2012.
**EDD example**

<table>
<thead>
<tr>
<th>defining metasymbol</th>
<th>:</th>
<th>definition separator metasymbol</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>terminator metasymbol</td>
<td>;</td>
<td>postfix optionality metasymbol</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>postfix star metasymbol</td>
<td>*</td>
<td>postfix plus metasymbol</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>start terminal metasymbol</td>
<td>&quot;</td>
<td>end terminal metasymbol</td>
<td>&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Megamodel

Syntactic notation specification

Generated notation grammar

extract

infer
module LLL
import util::IDE; // needed only for advanced IDE support (see last two lines)

start syntax LLLGrammar = LLLLayoutList LLLProduction* LLLLayoutList;
syntax LLLProduction = LLLNonterminal ":" {LLLDefinition ":"+ ";";

syntax LLLDefinition = LLLSymbol+;

syntax LLLSymbol
= @category="Identifier" nonterminal: LLLNonterminal
| @category="Constant" terminal: LLLTerminal
| group: "(" LLLDefinition ")"
| optional: LLLSymbol ":?"
| start: LLLSymbol ":*"
| plus: LLLSymbol ":+"
| sepliststar: ":{" LLLSymbol LLLSymbol ":}" ":*"
| seplistplus: ":{" LLLSymbol LLLSymbol ":}" ":+
lexical LLLTerminal = "\"" LLLTerminalSymbol "\"";
lexical LLLTerminalSymbol = !\[\",\];
lexical LLLNonterminal = [A-Za-z_0-9-\/]+ !>> [A-Za-z_0-9-\/];
layout LLLLayoutList = LLLLayout* !>> [\t-n \r \ ] !>> "#";
lexical LLLLayout = [\t-n \r \ ] | LLLComment ;
lexical LLLComment = @category="Comment" ":#" !\[\n\] * \[\n\];

Tree getLLL(str s,loc z) = parse(#LLLGrammar,z);
public void registerLLL() = registerLanguage("LLL","lll",getLLL);
# Grammar internal representation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rs</td>
<td>start symbols</td>
</tr>
<tr>
<td>Ps</td>
<td>productions</td>
</tr>
<tr>
<td>mapoptlist</td>
<td>mapoptlist(n,Rs), maplist(prod,Ps).</td>
</tr>
<tr>
<td>maplist</td>
<td>maplist(prod,Ps).</td>
</tr>
<tr>
<td>prod</td>
<td>prod(p(L,N,X)) = mapopt(label,L), atom(N), expr(X).</td>
</tr>
<tr>
<td>label</td>
<td>label(l(X)) = atom(X).</td>
</tr>
<tr>
<td>expr</td>
<td>expr(true).</td>
</tr>
<tr>
<td>expr</td>
<td>expr(fail).</td>
</tr>
<tr>
<td>expr</td>
<td>expr(a).</td>
</tr>
<tr>
<td>expr</td>
<td>expr(t(T)) = atom(T).</td>
</tr>
<tr>
<td>expr</td>
<td>expr(n(N)) = atom(N).</td>
</tr>
<tr>
<td>expr</td>
<td>expr(';'(Xs)) = maplist(expr,Xs).</td>
</tr>
<tr>
<td>expr</td>
<td>expr('?'(X)) = expr(X).</td>
</tr>
<tr>
<td>expr</td>
<td>expr('∗'(X)) = expr(X).</td>
</tr>
<tr>
<td>expr</td>
<td>expr('+'(X)) = expr(X).</td>
</tr>
<tr>
<td>expr</td>
<td>expr(slp(X,Y)) = expr(X), expr(Y).</td>
</tr>
<tr>
<td>expr</td>
<td>expr(sls(X,Y)) = expr(X), expr(Y).</td>
</tr>
<tr>
<td>expr</td>
<td>expr(s(S,X)) = atom(S), expr(X).</td>
</tr>
</tbody>
</table>

- 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

Generated notation grammar (in BGF)

LLL1Grammar:
    LLL1Production*
LLL1Production:
    LLL1Nonterminal ":" {LLL1Definition "|"}+ ";"
LLL1Definition:
    LLL1Symbol+
[nonterminal] LLL1Symbol:
    LLL1Nonterminal
[terminal] LLL1Symbol:
    LLL1Terminal
[optional] LLL1Symbol:
    LLL1Symbol "?"
[star] LLL1Symbol:
    LLL1Symbol "*"
[plus] LLL1Symbol:
    LLL1Symbol "+"
Beautified notation grammar (in BGF)

grammar:
  rule+
rule:
  sort ":" alts ";"
alts:
  alt alts-tail*
alts-tail:
    "|" alt
alt:
  term*
term:
  basis repetition?
basis:
  literal
  sort
repetition:
  "*"
  "+"
  "?"
Megamodel

Syntactic notation specification → Generated notation grammar

extract

Generated notation grammar ← Beautified notation grammar

adapt

infer
Bidirectional grammar adaptation

- $\text{XBGF} \Rightarrow \text{EBGF}$:
  - 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

Megamodel

- Syntactic notation specification
- Generated notation grammar
- Beautified notation grammar
- Notation “in itself”

Flow:
- extract → adapt → pretty-print
- infer ↔ adapt
- recover
Megamodel

Syntactic notation specification → Generated notation grammar → Beautified notation grammar → Notation “in itself”

extract → adapt → pretty-print

infer → ↔ • adapt ↔ • recover

Syntactic notation specification
Generated notation grammar
Beautified notation grammar
Notation “in itself”
Megamodel

- Syntactic notation specification
- Generated notation grammar
- Beautified notation grammar
- Notation “in itself”

Diagrams:
- Extract
- Adapt
- Pretty-print
- Recover
- Transform notation
Notation transformation

- EDD — notation, consists of metasymbols
- XEDD — transformation language
  - rename-metasymbol\((s, v_1, v_2)\)
    - e.g., change defining metasymbol from “:” to “::=”
  - introduce-metasymbol\((s, v)\)
    - e.g., bring a terminator metasymbol to a notation
  - eliminate-metasymbol\((s, v)\)
Megamodel
Grammar convergence

source grammar

grammar transformation

target grammar

source grammar

grammar transformation

source grammar

bidirectional grammar transformation

= relationship
Megamodel

Syntactic notation specification

Generated notation grammar

Beautified notation grammar

Notation “in itself”

extract

infer

adapt

pretty-print

transform notation

convergence relationship

coupled mutations

recover

recover

pretty-print
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:
  \[ \mu = \langle \{c_i\}, \{t_i\}, c_{\text{post}} \rangle. \]

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

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

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

- A notation evolution step $\Delta$ consists of the following coupled components:
  - $\sigma$, a bidirectional notation transformation that changes the notation itself
  - $\delta$, a convergence relationship that can transform the notation grammars
  - $\gamma$, a bidirectional grammar adaptation that prepares a beautified readable version of $N'$
  - $\mu$, an unidirectional coupled grammar mutation that migrates the grammarbase according to notation changes
  - possibly $\mu'$, an unidirectional coupled grammar mutation that migrates the grammarbase according to the inverse of the intended notation changes
Evaluation
Megamodel: case study

LLL1.edd ➔ edd2rsc+ rsc2bgf ➔ LLL1. spec. bgf

LLL1. spec. bgf ➔ LLL1. spec2doc. ξ bgf ➔ LLL1. doc. bgf

LLL1. doc. bgf ➔ bgfpp ➔ LLL1.LLL1

LLL1. spec. bgf ➔ LLL1to2.ξ bgf ➔ LLL1to2.xedd ➔ LLL1.edd

LLL2.edd ➔ edd2rsc+ rsc2bgf ➔ LLL2. spec. bgf

LLL2. spec. bgf ➔ LLL2. spec2doc. ξ bgf ➔ LLL2. doc. bgf

LLL2. doc. bgf ➔ bgfpp ➔ LLL2.LLL2

LLL2. spec. bgf ➔ LLL2to2.ξ bgf ➔ LLL2to2.xedd ➔ LLL2.edd

LLL1.LLL1 ➔ Grammar Hunter ➔ *.LLL1

LLL2.LLL2 ➔ Grammar Hunter ➔ *.LLL2

LLL1to2.xedd ➔ EliminateGroup.rsc ➔ LLL1to2.ξ bgf

LLL1to2.ξ bgf ➔ LLL1to2.xedd

LLL1. spec2doc. ξ bgf ➔ LLL1. spec. bgf

LLL2. spec2doc. ξ bgf ➔ LLL2. spec. bgf
Megamodel: previously
LLL1 in itself

grammar : rule+;
rule : sort "::" alts "";
alts : alt alts-tail*;
alts-tail : "|" alt;
alt : term*;
term : basis repetition?;
basis : literal | sort;
repetition : "*" | "+" | "?";

LLL2 in itself

specification : rule+;
rule          : ident "::" disjunction ";;";
disjunction  : \{ conjunction "|" \} +;
conjunction   : term*;
term          : basis repetition?;
basis         : ident | literal
               | alternation | group;
repetition    : "+" | "*" | "?";
alternation   : \{ "basis basis " \} repetition;
group         : "(" disjunction ")" ;
Megamodel: manually
Megamodel: automated

LLL1.edd

LLL1.spec bgf

LLL1.doc bgf

LLL1.LLL1

LLL2.edd

LLL2.spec bgf

LLL2.doc bgf

LLL2.LLL2

LLL1to2.xedd

LLL1to2.ξbgf

LLL1.spec2doc.ξbgf

LLL2.spec2doc.ξbgf

Grammar Hunter

EliminateGroup.rsc

*LLL1

*LLL2

edd2rsc+ rsc2bgf

bgfpp

edd2rsc+ rsc2bgf
Applying coupled mutation
eliminate-metasymbol(group)
to Grammar Zoo

<table>
<thead>
<tr>
<th>Language</th>
<th>Value1</th>
<th>Language</th>
<th>Value2</th>
<th>Language</th>
<th>Value3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ada-kellogg</td>
<td>108</td>
<td>csharp-iso-23270-2003</td>
<td>0</td>
<td>java-1-jls-read</td>
<td>0</td>
</tr>
<tr>
<td>ada-kempe</td>
<td>89</td>
<td>csharp-iso-23270-2006</td>
<td>0</td>
<td>java-2-jls-impl</td>
<td>36</td>
</tr>
<tr>
<td>ada-laemmel-verhoef</td>
<td>79</td>
<td>csharp-msft-ls-1.0</td>
<td>0</td>
<td>java-2-jls-read</td>
<td>0</td>
</tr>
<tr>
<td>ada-lncs-2219</td>
<td>89</td>
<td>csharp-msft-ls-1.2</td>
<td>0</td>
<td>java-5-habelitz</td>
<td>65</td>
</tr>
<tr>
<td>ada-lncs-4348</td>
<td>109</td>
<td>csharp-msft-ls-3.0</td>
<td>0</td>
<td>java-5-jls-impl</td>
<td>60</td>
</tr>
<tr>
<td>c-iso-9899-1999</td>
<td>0</td>
<td>csharp-msft-ls-4.0</td>
<td>0</td>
<td>java-5-jls-read</td>
<td>1</td>
</tr>
<tr>
<td>c-iso-9899-tc2</td>
<td>0</td>
<td>csharp-zaytsev</td>
<td>23</td>
<td>java-5-parr</td>
<td>95</td>
</tr>
<tr>
<td>c-iso-9899-tc3</td>
<td>0</td>
<td>dart-google</td>
<td>58</td>
<td>java-5-stahl</td>
<td>92</td>
</tr>
<tr>
<td>cpp-iso-14882-1998</td>
<td>0</td>
<td>dart-spec-0.01</td>
<td>56</td>
<td>java-5-studman</td>
<td>91</td>
</tr>
<tr>
<td>cpp-iso-n2723</td>
<td>0</td>
<td>dart-spec-0.05</td>
<td>62</td>
<td>mediawiki-bnf</td>
<td>32</td>
</tr>
<tr>
<td>csharp-ecma-334-1</td>
<td>0</td>
<td>eiffel-bezault</td>
<td>45</td>
<td>mediawiki-ebnf</td>
<td>30</td>
</tr>
<tr>
<td>csharp-ecma-334-2</td>
<td>0</td>
<td>eiffel-iso-25436-2006</td>
<td>345</td>
<td>modula-sdf</td>
<td>50</td>
</tr>
<tr>
<td>csharp-ecma-334-3</td>
<td>0</td>
<td>fortran-derricks</td>
<td>101</td>
<td>modula-src-052</td>
<td>65</td>
</tr>
<tr>
<td>csharp-ecma-334-4</td>
<td>0</td>
<td>java-1-jls-impl</td>
<td>0</td>
<td>w3c-xpath1</td>
<td>3</td>
</tr>
</tbody>
</table>
Conclusion
Conclusion

• Extended XBGF to bidirectionality, resulting in $\Xi$BGF.
• Proposed EDD and XEDD for notation & its evolution.
• Presented a case study of LLL evolution (GDK).
• Generalised transformers and generators to transformations and mutations; also formalised them.
• Implemented an XEDD processor for evolution, coevolution, change propagation and mutation.
Discussion

grammarware.net
slps.sf.net