Improving modularity in GLL

Ana-Maria Farcasi

Department of Mathematics and Computer Science, Technical University of Eindhoven

March 19, 2013
Table of Contents

1. Problem description
2. Plan of action
3. Done so far
4. Conclusions
<table>
<thead>
<tr>
<th></th>
<th>Problem description</th>
<th>Plan of action</th>
<th>Done so far</th>
<th>Conclusions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Problem description</td>
<td>Plan of action</td>
<td>Done so far</td>
<td>Conclusions</td>
</tr>
</tbody>
</table>
The need for a discipline of programming for grammars
Problem context

- The need for a discipline of programming for grammars
- Concept of modularity in grammars not yet clearly understood
Problem context

- The need for a discipline of programming for grammars
- Concept of modularity in grammars not yet clearly understood
- Concerned about grammar modularity in the parsing arena
Problem context

- The need for a discipline of programming for grammars
- Concept of modularity in grammars not yet clearly understood
- Concerned about grammar modularity in the parsing arena
- Modularity in the GLL (General LL) parsing algorithm
Possible benefits of modularity in grammars

- Increasing productivity in grammars’ development
Possible benefits of modularity in grammars

- Increasing productivity in grammars’ development
- Cleaner development and easier debugging
Possible benefits of modularity in grammars

- Increasing productivity in grammars’ development
- Cleaner development and easier debugging
- Faster and better understanding of a grammar as a whole
Possible benefits of modularity in grammars

- Increasing productivity in grammars’ development
- Cleaner development and easier debugging
- Faster and better understanding of a grammar as a whole
- Reduced costs
Practical examples of modularity in grammars

- Grammar evolution
Practical examples of modularity in grammars

- Grammar evolution
- Creation of a grammar from components
Practical examples of modularity in grammars

- Grammar evolution
- Creation of a grammar from components
- Embedding languages
Problem statement

- Suitability of modularity for grammars
Problem statement

- Suitability of modularity for grammars
- Different layouts for embedded languages
Problem statement

- Suitability of modularity for grammars
- Different layouts for embedded languages
- Most suitable operators for combining grammars
Problem statement

- Suitability of modularity for grammars
- Different layouts for embedded languages
- Most suitable operators for combining grammars
- Modularity composition decisions creating least ambiguities
Why is modularity in grammars a hard problem?

- Hard to separate the grammar in low coupling modules
Why is modularity in grammars a hard problem?

- Hard to separate the grammar in low coupling modules
- General framework for modular grammars hard to establish
  - Namespace handling
  - Import type
  - Use of parameters and renaming
<table>
<thead>
<tr>
<th></th>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Problem description</td>
</tr>
<tr>
<td>2</td>
<td>Plan of action</td>
</tr>
<tr>
<td>3</td>
<td>Done so far</td>
</tr>
<tr>
<td>4</td>
<td>Conclusions</td>
</tr>
</tbody>
</table>
Use cases

**UC1** Creating a grammar from components with glue code (language unification)
Use cases

UC1  Creating a grammar from components with glue code (language unification)

UC2  Adding components to a grammar (language extension)
Use cases

UC1 Creating a grammar from components with glue code (language unification)

UC2 Adding components to a grammar (language extension)

UC3 Extending a grammar with a new alternative for an existing production rule
Use cases

UC1 Creating a grammar from components with glue code (language unification)

UC2 Adding components to a grammar (language extension)

UC3 Extending a grammar with a new alternative for an existing production rule

UC4 Extending a grammar by modifying alternatives of existing production rules
Use cases

**UC1** Creating a grammar from components with glue code (language unification)

**UC2** Adding components to a grammar (language extension)

**UC3** Extending a grammar with a new alternative for an existing production rule

**UC4** Extending a grammar by modifying alternatives of existing production rules

**UC5** Extending a grammar by overriding a production rule
Use cases

UC1  Creating a grammar from components with glue code (language unification)
UC2  Adding components to a grammar (language extension)
UC3  Extending a grammar with a new alternative for an existing production rule
UC4  Extending a grammar by modifying alternatives of existing production rules
UC5  Extending a grammar by overriding a production rule
UC6  Embedding a language into another language
Use cases

UC1 Creating a grammar from components with glue code (language unification)

UC2 Adding components to a grammar (language extension)

UC3 Extending a grammar with a new alternative for an existing production rule

UC4 Extending a grammar by modifying alternatives of existing production rules

UC5 Extending a grammar by overriding a production rule

UC6 Embedding a language into another language

UC7 Embedding a language into another language at program level
Use cases

**UC1** Creating a grammar from components with glue code (language unification)

**UC2** Adding components to a grammar (language extension)

**UC3** Extending a grammar with a new alternative for an existing production rule

**UC4** Extending a grammar by modifying alternatives of existing production rules

**UC5** Extending a grammar by overriding a production rule

**UC6** Embedding a language into another language

**UC7** Embedding a language into another language at program level

**UC8** Removing a construct from a grammar
Use cases

**UC1** Creating a grammar from components with glue code (language unification)

**UC2** Adding components to a grammar (language extension)

**UC3** Extending a grammar with a new alternative for an existing production rule

**UC4** Extending a grammar by modifying alternatives of existing production rules

**UC5** Extending a grammar by overriding a production rule

**UC6** Embedding a language into another language

**UC7** Embedding a language into another language at program level

**UC8** Removing a construct from a grammar

**UC9** Removing a feature from a grammar
<table>
<thead>
<tr>
<th>Problem description</th>
<th>Plan of action</th>
<th>Done so far</th>
<th>Conclusions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace handling</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Mixing of namespaces at import
Namespace handling

- Mixing of namespaces at import
- Separation namespaces at import
Modularization level

Extremes:
- One big module

Further classification:
Modularization level

Extremes:

- One big module
- Small modules: every production rule is a separate module.

Further classification:
Modularization level

Extremes:
- One big module
- Small modules: every production rule is a separate module.

Further classification:
- Modules as in general software development: module comprises functionality.
Import graphs

- The imported productions are not being reexported.
The imported productions are not being reexported.
The imported productions can be reexported.
Modularity operators

- Add alternative to an existing production rule
Modularity operators

- Add alternative to an existing production rule
- Remove alternative from a production rule
Modularity operators

- Add alternative to an existing production rule
- Remove alternative from a production rule
- Import a module
Modularity operators

- Add alternative to an existing production rule
- Remove alternative from a production rule
- Import a module
- Extend a module
Modularity operators

- Add alternative to an existing production rule
- Remove alternative from a production rule
- Import a module
- Extend a module
- Use parameters in a module
Modularity operators

- Add alternative to an existing production rule
- Remove alternative from a production rule
- Import a module
- Extend a module
- Use parameters in a module
- Use renaming in a module
<table>
<thead>
<tr>
<th>Problem description</th>
<th>Plan of action</th>
<th>Done so far</th>
<th>Conclusions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Further steps</td>
<td>Choose list of modularity operators</td>
<td>Implement modularity in JGLL</td>
<td>Experiment with modularity in JGLL</td>
</tr>
</tbody>
</table>
Further steps

- Choose list of modularity operators
Further steps

- Choose list of modularity operators
- Implement modularity in JGLL
Further steps

- Choose list of modularity operators
- Implement modularity in JGLL
- Experiment with modularity in JGLL
Ana-Maria Farcasi

Improving modularity in GLL
Installed the GLL framework for Scala
Done so far

- Installed the GLL framework for Scala
- Experimented with modules for a simple expressions language
Done so far

- Installed the GLL framework for Scala
- Experimented with modules for a simple expressions language
- Experimented with modules for the COOL programming language
Done so far

- Installed the GLL framework for Scala
- Experimented with modules for a simple expressions language
- Experimented with modules for the COOL programming language
- Read papers on modularity in grammars.
Table of Contents

1 Problem description

2 Plan of action

3 Done so far

4 Conclusions
Questions to be answered

- Does modularity in grammars really brings value? Does it ease the development of new languages or of evolving languages?
Questions to be answered

- Does modularity in grammars really bring value? Does it ease the development of new languages or of evolving languages?
- What are the most relevant and useful operators for modularity in grammars?
Questions to be answered

- Does modularity in grammars really bring value? Does it ease the development of new languages or of evolving languages?
- What are the most relevant and useful operators for modularity in grammars?
- How does modularity integrate with the GLL parsing algorithm?