Attribute Grammars for Modular Disambiguation

Valentin David\textsuperscript{1}  Akim Demaille\textsuperscript{2}  Olivier Gournet\textsuperscript{2}

\textsuperscript{1}Bergen University — Norway

\textsuperscript{2}EPITA Research and Development Laboratory (LRDE) — France

International Conference on
Intelligent Computer Communication and Processing
2006
1. Modular Front-Ends

2. Attribute Grammars for Disambiguation

3. Conclusion
Language Extensions

- domain specific extensions [4]
- embedded SQL
- design by contract (pre-/post-conditions) [1]
- syntactic sugar [5, 7]
- language evolution prototyping
- etc.
- and composition of them!
Language Extensions

- domain specific extensions [4]
- embedded SQL
- design by contract (pre-/post-conditions) [1]
- syntactic sugar [5, 7]
- language evolution prototyping

- etc.
- and composition of them!
Language Extensions

- domain specific extensions [4]
- embedded SQL
- design by contract (pre-/post-conditions) [1]
- syntactic sugar [5, 7]
- language evolution prototyping
- etc.
- and composition of them!
Language Extensions

- domain specific extensions [4]
- embedded SQL
- design by contract (pre-/post-conditions) [1]
- syntactic sugar [5, 7]
- language evolution prototyping
- etc.

and composition of them!
What's this?

\[ a \times b; \]
C/C++ Have Bad Syntactic Properties

What's this?

```c
a * b;
```

A value product

```c
int a, b;
a * b;
```
C/C++ Have Bad Syntactic Properties

What's this?

a * b;

A value product

```c
int a, b;
a * b;
```

A variable declaration

```c
typedef int a;
a * b;
```
What's this?

(a) - (b);
What's this?

(a) - (b);

A subtraction

```c
int a, b;
(a) - (b);
```
What’s this?

(a) - (b);

A subtraction

```c
int a, b;
(a) - (b);
```

A cast

```c
typedef int a;
(a) - (b);
```
To overcome context sensitivity, use “lexical tie-ins” [6]

- Use LALR(1) generators
- Use LR(1) generators
- None of these techniques is closed under union!
- So use GLR [10, 11]
Modular Parser Generation

- To overcome context sensitivity, use “lexical tie-ins” [6]
- Use LALR(1) generators
- Use LR(1) generators
- None of these techniques is closed under union!
- So use GLR [10, 11]
To overcome context sensitivity, use “lexical tie-ins” [6]
Use LALR(1) generators
Use LR(1) generators
None of these techniques is closed under union!
So use GLR [10, 11]
Modular Parser Generation

- To overcome context sensitivity, use “lexical tie-ins” [6]
- Use LALR(1) generators
- Use LR(1) generators
- None of these techniques is closed under union!
- So use GLR [10, 11]
Generalized LR Parsing

- accepts the full class of context-free languages
- including ambiguous grammars
  - so accept a superset of context-sensitive languages
- and filter the resulting “parse-forest”
Generalized LR Parsing

- accepts the full class of context-free languages
- including ambiguous grammars
- so accept a superset of context-sensitive languages
- and filter the resulting “parse-forest”
Generalized LR Parsing

- accepts the full class of context-free languages
- including ambiguous grammars
- so accept a superset of context-sensitive languages
- and filter the resulting “parse-forest”
The Big Picture

Concrete syntax to abstract syntax

V. David et al. (Bergen U. & LRDE)
The Big Picture

Concrete syntax to abstract syntax

C source → SGLR parser → Parse forest → Disambiguation → Parse tree

AST Transformations → AST → Pretty printer → C source → Standard C compiler

Need for modular disambiguation.
1 Modular Front-Ends

2 Attribute Grammars for Disambiguation

3 Conclusion
A Simple Ambiguous Grammar

context-free syntax

"true" -> Bool
"false" -> Bool
Bool "|" Bool -> Bool

**Figure:** Boolean Expressions (Ambiguous) [3]
Parse Forest for true | true | true
Techniques for Disambiguation

Dedicated Code

Poor modularity, more imperative, less declarative

Algebraic Specification [2]
Modular, declarative, too hard to use

Attribute Grammars (AGs) [9]
Modular, declarative, easy to use
Techniques for Disambiguation

Dedicated Code

- Poor modularity, more imperative, less declarative

Algebraic Specification [2]

- Modular, declarative, too hard to use

Attribute Grammars (AGs) [9]

- Modular, declarative, easy to use
Techniques for Disambiguation

Dedicated Code

Poor modularity, more imperative, less declarative

Algebraic Specification [2]

Modular, declarative, too hard to use

Attribute Grammars (AGs) [9]

Modular, declarative, easy to use
A Simple AG Example

context-free syntax
"true" | "false" -> Bool
{attributes(assoc:
    root.is_atomic := true
)}

lhs:Bool "|" rhs:Bool -> Bool
{attributes(assoc:
    root.is_atomic := false
    root.ok := rhs.is_atomic
)}

Figure: Boolean Expressions Disambiguated
Disambiguated Parse Forest

V. David et al. (Bergen U. & LRDE)
Application to ISO C99 [8]

- 126 symbols
- 356 rules
- 53 modules
  - 10 attribute kinds
  - 190 attribute rules
  - completed to 1183 rules
Application to ISO C99 [8]

- 126 symbols
- 356 rules
- 53 modules
- 10 attribute kinds
- 190 attribute rules
- completed to 1183 rules
Application to ISO C99 [8]

- 126 symbols
- 356 rules
- 53 modules
- 10 attribute kinds
- 190 attribute rules
- completed to 1183 rules
<table>
<thead>
<tr>
<th></th>
<th>HelloW</th>
<th>Lemon</th>
<th>Eval</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lines of code</strong></td>
<td>448</td>
<td>4135</td>
<td>28392</td>
</tr>
<tr>
<td><strong>Ambiguities</strong></td>
<td>103</td>
<td>6410</td>
<td>68195</td>
</tr>
<tr>
<td><strong>Duration (s)</strong></td>
<td>3.8</td>
<td>28.0</td>
<td>322.5</td>
</tr>
</tbody>
</table>
1. Modular Front-Ends

2. Attribute Grammars for Disambiguation

3. Conclusion
Attribute Grammars for Modular Disambiguation

Pros
- Declarativity
- Modularity
- Simplicity

Cons
- Slow
- Hard to debug
- Poor genericity
- External data
Attribute Grammars for Modular Disambiguation

Pros
- Declarativity
- Modularity
- Simplicity

Cons
- Slow
- Hard to debug
- Poor genericity
- External data
Future Work

- Completion of C++ tool-chain
- Various C++ syntactic sugar
- Parse time disambiguation
Future Work

- Completion of C++ tool-chain
- Various C++ syntactic sugar
- Parse time disambiguation
Future Work

- Completion of C++ tool-chain
- Various C++ syntactic sugar
- Parse time disambiguation
Questions?

Mark van den Brand, Steven Klusener, Leon Moonen, and Jurgen J. Vinju.
Generalized parsing and term rewriting: Semantics driven disambiguation.

Martin Bravenboer, René de Groot, and Eelco Visser.
Metaborg in action: Examples of domain-specific language embedding and assimilation using Stratego/XT.
A static C++ object-oriented programming (SCOOP) paradigm mixing benefits of traditional OOP and generic programming.
In Proceedings of the Workshop on Multiple Paradigm with OO Languages (MPOOL), Anaheim, CA, USA, October 2003.

Bison: GNU LALR(1) and GLR parser generator, 2003.
Akim Demaille, Sylvain Peyronnet, and Benoît Sigoure. Modeling of sensor networks using XRM. In *Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA’06)*, Coral Beach Resort, Paphos, Cyprus, November 2006. Accepted.


Masaru Tomita.  
*Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*.  

Eelco Visser.  
*Scannerless generalized-LR parsing*.  