Attribute Grammars for Modular Disambiguation

Valentin David[1] Akim Demaille[2] Olivier Gournet[2]

[1]Bergen University — Norway

[2]EPITA Research and Development Laboratory (LRDE) — France

International Conference on
Intelligent Computer Communication and Processing
2006

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!

C/C++ Have Bad Syntactic Properties

What's this?

```
a * b;
```

C/C++ Have Bad Syntactic Properties

What's this?

```
a * b;
```

A value product

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

C/C++ Have Bad Syntactic Properties

What's this?

```
a * b;
```

A value product

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

A variable declaration

```
typedef int a;
a * b;
```

C/C++ Have Bad Syntactic Properties

What's this?

```
(a) - (b);
```

C/C++ Have Bad Syntactic Properties

What's this?

```
(a) - (b);
```

A subtraction

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

C/C++ Have Bad Syntactic Properties

What's this?

```
(a) - (b);
```

A subtraction

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

A cast

```
typedef int a;
(a) - (b);
```

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]

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]

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]

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"

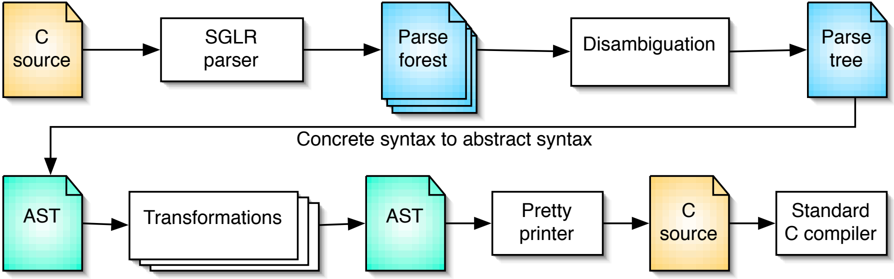
Generalized LR Parsing

- 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”

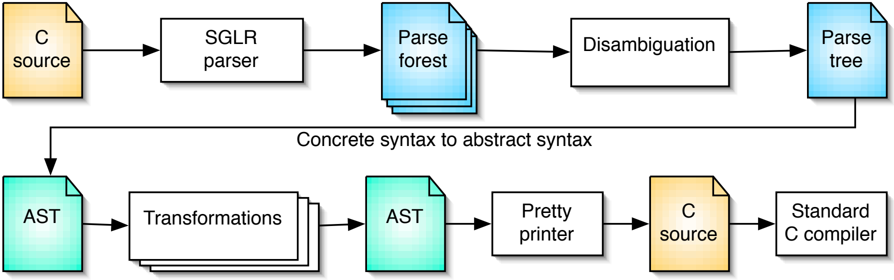
Generalized LR Parsing

- 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



The Big Picture



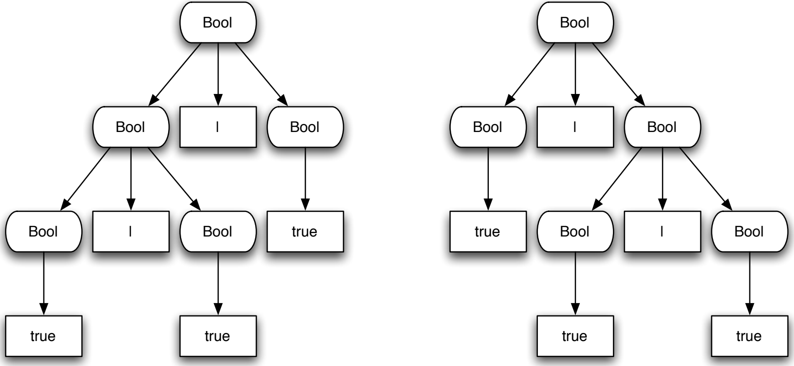
Need for modular disambiguation.

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

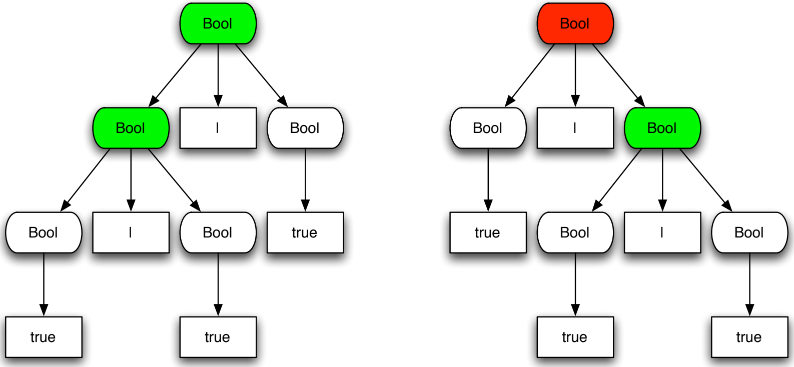
A Simple AG Example

```
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



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

C Disambiguation

	HelloW	Lemon	Eval
Lines of code	448	4 135	28 392
Ambiguities	103	6 410	68 195
Duration (s)	3.8	28.0	322.5

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?

Bibliography I

Alexandre Borghi, Valentin David, and Akim Demaille.
C-transformers - a framework to write C program transformations.
ACM Crossroads, 12(3), Spring 2006.
http://www.acm.org/crossroads/xrds12-3/contractc.html.

Mark van den Brand, A. van Deursen, J. Heering, H. de Jonge, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P. Olivier, J. Scheerder, J. Vinju, E. Visser, and J. Visser.
The ASF+SDF Meta-Environment: a component-based language development environment.
In R. Wilhelm, editor, *Compiler Construction 2001 (CC'2001)*, volume 2027 of *LNCS*, pages 365 – 370. Springer-Verlag, 2001.

Bibliography II

Mark van den Brand, Steven Klusener, Leon Moonen, and Jurgen J. Vinju.
Generalized parsing and term rewriting: Semantics driven disambiguation.
volume 82 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2003.

Martin Bravenboer, René de Groot, and Eelco Visser.
Metaborg in action: Examples of domain-specific language embedding and assimilation using Stratego/XT.
In *Proceedings of the Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE'05)*, Braga, Portugal, July 2005.

Bibliography III

Nicolas Burrus, Alexandre Duret-Lutz, Thierry Géraud, David Lesage, and Raphaël Poss.
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.

Robert Corbett, Richard Stallman, and Paul Hilfinger.
Bison: GNU LALR(1) and GLR parser generator, 2003.
`http://www.gnu.org/software/bison/bison.html`.

Bibliography IV

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.

ISO/IEC.
ISO/IEC 14882:2003 (E). *Programming languages - C++*, 2003.

Donald E. Knuth.
Semantics of context-free languages.
Journal of Mathematical System Theory, pages 127–145, 1968.

Bibliography V

Masaru Tomita.
Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems.
Kluwer Academic Publishers, 1985.

Eelco Visser.
Scannerless generalized-LR parsing.
Technical Report P9707, Programming Research Group, University of Amsterdam, July 1997.