Concrete and Abstract Syntax

Prof. Evan Chang
Meeting 4, CSCI 3155, Fall 2009

Announcements

• Assignment 1 due tonight at 11:55pm
  - Find a partner
• Assignment 2 out tonight
  - Start early!
  - PL-Detective User ID sent out by e-mail.
  Problems?

Languages and Grammars

• A language is a set of strings over some alphabet (=sentences)
• A context-free grammar (=BNF) is a notation for describing languages
• A derivation exhibits a sentence in a language
• We typically consider an alphabet of tokens rather than of characters

Grammars and Derivations

• Formally, a grammar consists of
  - an alphabet $\Sigma$ of terminals (usually, tokens)
  - a finite set $N$ of non-terminals
  - a finite set of productions $n ::= s$ where $n \in N$ and $s \in (\Sigma \cup N)^*$
• Example:
  $e ::= \text{num} | e + e | e * e$
• Derivation for $1 + 2 + 3$?
End of Review
On to Concrete and Abstract Syntax

One-Slide Summary
- Concrete syntax is the surface level of a language (think, strings)
- Abstract syntax is the deep structure of a language (think, trees/terms)
- Parsers convert concrete syntax into abstract syntax and have to deal with ambiguity
- Precedence and associativity are some ways to deal with ambiguity

Ambiguity
\[ e ::= \text{num} \mid e + e \mid e \ast e \]

- What does \(1 + 2 \ast 3\) evaluate to (i.e., what is the semantics of \(1 + 2 \ast 3\))?
  \begin{align*}
  (1+2) \ast 3 &= 9 \\
  \text{or} \\
  1 + (2 \ast 3) &= 7
  \end{align*}

Ambiguity
The same string can be read in two different ways!

- "Need parentheses with ambiguous grammars"

Derivations and Parse Trees
\[ e ::= \text{num} \mid e + e \mid e \ast e \]

- Let's write a derivation and a parse tree for \(1 + 2 \ast 3\)
  \begin{align*}
  e &\Rightarrow e + e \\
  &\Rightarrow e + e + e \\
  &\Rightarrow \ldots \\
  &\Rightarrow \text{num} + \text{num} + \text{num} \\
  &\Rightarrow (1) + (2) + (3) \\
  (42) \ast 3 &\Rightarrow 1 + (2 \ast 3)
  \end{align*}
Derivations and Parse Trees

unambiguous grammar

= every string has a unique parse tree

Example

e ::= num | e + e | e * e

- Is the above grammar ambiguous?

7 - 3 - 2
(7 - 3) - 2 OR 7 - (3 - 2)

Associativity

Hasti: wonyon exthe

e ::= num | e - e

- Try rewriting the grammar to get "left associativity"

```
(7 - 3) - 2
```

```
e - e
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
e - num
```

```
```
Better Idea:
Concrete Syntax vs. Abstract Syntax

- Use trees directly!
- Separate readability concerns (concrete) from structural ones (abstract)
- Concrete syntax = strings
- Abstract syntax = terms (~parse trees)

Abstract Syntax

\[ e ::= \text{num} | \text{plus}(e,e) | \text{times}(e,e) \]

- We can see the structure immediately
- What do the two versions of "1 + 2 * 3" become?

Parser

- Converts a string into a tree
- Maps concrete syntax into abstract syntax
- Does syntactic analysis once and for all
Are unambiguous grammars better?

- Compare
  \[ e ::= \text{num} | e + e | e \cdot e \] ambiguous
  \[ e ::= t | t + e \]
  \[ t ::= \text{num} | \text{num} \cdot t \]
  \[ \text{hurts to read} \]
  \[ \text{assume abstract syntax} \]
  \[ \text{view abstract syntax} \]

Often write ambiguous grammars and view them as abstract syntax

For Next Time

- Reading
- Online discussion forum
  - \( \geq 1 \) substantive question, comment, or answer each week
- Homework assignment 2
  - Start early!