

Context Free Grammars and Syntax

- **Making syntactic sense of a stream of tokens**
- **First we need terminology and theory**

Spring 2008

CIS 461/561 - Syntax and Grammars

1

Definition of Context Free Grammar

- *Terminals* - An alphabet (like regular expressions, only now the symbols are whole tokens, not characters), including ϵ
- *Non-terminals* - A set of *names* for structures (like *statement*, *expression*, *definition* – sometimes called the *variables* of the grammar)
- *Production rules* - The grammar *rules* specify substitutions of non-terminals by strings of terminals and non-terminals – these express the structure of the names
- A *start* symbol (the name of the most general structure — *compilation_unit* in C or Java)

Spring 2008

CIS 461/561 - Syntax and Grammars

2

CFGs and Languages

- Think of the grammar rules as a series of definitions of the elements of the languages
 - Often have multiple choices for definitions
- Beginning with start symbol (the program), choose a definition (rule)
- Then choose definitions for each new structure element that has appeared
- Repeat this process until only terminals remain
- You have generated a program from the grammar!

Spring 2008

CIS 461/561 - Syntax and Grammars

3

Basic Example: Simple integer arithmetic expressions

$$exp \rightarrow exp\ op\ exp \mid (exp) \mid \mathbf{number}$$

$$op \rightarrow + \mid - \mid *$$

2 non-terminals

6 terminals

6 productions (3 on each line)

In what way does such a CFG differ from a regular expression?

digit = 0|1|...|9
number = *digit digit**

Recursion!

Recursive rules

“Base” rule

Spring 2008

CIS 461/561 - Syntax and Grammars

4

Recursion

- Recursion is a natural way to define programs
 - Expressions are made up of expressions
 - Structure of nested blocks is recursively defined...
 - Data structures are collections of data structures...
 - Multi-dimensional arrays are arrays of arrays...
 - Nested parentheses, braces, brackets...
- Thus, we need a CFG

Spring 2008

CIS 461/561 - Syntax and Grammars

5

CFGs compared to DFAs

- DFAs use finite memory (the number of states)
- CFGs require unbounded memory (a stack of indeterminate length)
- Real programming languages are too complicated for regular expressions (DFAs), but not too complex for a grammar

Spring 2008

CIS 461/561 - Syntax and Grammars

6

CFG as a Program Recognizer

- CFGs can generate programs, but how to determine if a given program is generated?
 - This is the activity of parsing
- Requires backtracking
- Algorithms use a stack – many different approaches
- More difficult to eliminate the non-determinism
- Two basic flavors – top down and bottom up parsing

Spring 2008

CIS 461/561 - Syntax and Grammars

7

Parse trees and grammars

Express matching of a string

“(34 - 3) * 42” by a *derivation*:

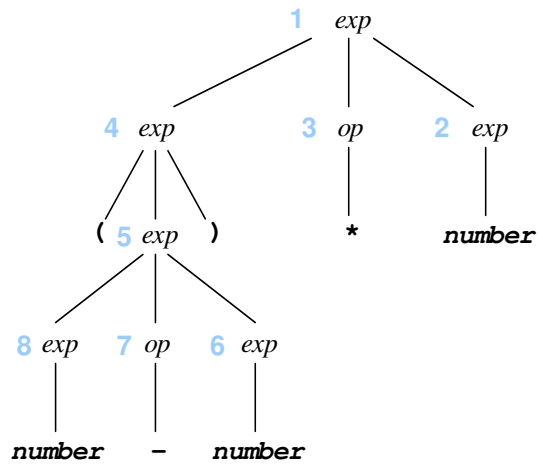
(1)	$exp \Rightarrow exp\ op\ exp$	$[exp \rightarrow exp\ op\ exp]$
(2)	$\Rightarrow exp\ op\ number$	$[exp \rightarrow number]$
(3)	$\Rightarrow exp\ *\ number$	$[op \rightarrow *]$
(4)	$\Rightarrow (exp)\ *\ number$	$[exp \rightarrow (exp)]$
(5)	$\Rightarrow (exp\ op\ exp)\ *\ number$	$[exp \rightarrow exp\ op\ exp]$
(6)	$\Rightarrow (exp\ op\ number)\ *\ number$	$[exp \rightarrow number]$
(7)	$\Rightarrow (exp - number)\ *\ number$	$[op \rightarrow -]$
(8)	$\Rightarrow (number - number)\ *\ number$	$[exp \rightarrow number]$

Spring 2008

CIS 461/561 - Syntax and Grammars

8

Abstract the structure of a derivation to a parse tree:



Spring 2008

CIS 461/561 - Syntax and Grammars

9

Derivations can vary, even when the parse tree doesn't:

leftmost derivation (previous - *rightmost*)

- | | | |
|-----|--|--|
| (1) | $\text{exp} \Rightarrow \text{exp op exp}$ | $[\text{exp} \rightarrow \text{exp op exp}]$ |
| (2) | $\Rightarrow (\text{exp}) \text{ op exp}$ | $[\text{exp} \rightarrow (\text{exp})]$ |
| (3) | $\Rightarrow (\text{exp op exp}) \text{ op exp}$ | $[\text{exp} \rightarrow \text{exp op exp}]$ |
| (4) | $\Rightarrow (\text{number op exp}) \text{ op exp}$ | $[\text{exp} \rightarrow \text{number}]$ |
| (5) | $\Rightarrow (\text{number - exp}) \text{ op exp}$ | $[\text{op} \rightarrow -]$ |
| (6) | $\Rightarrow (\text{number - number}) \text{ op exp}$ | $[\text{exp} \rightarrow \text{number}]$ |
| (7) | $\Rightarrow (\text{number - number}) * \text{exp}$ | $[\text{op} \rightarrow *]$ |
| (8) | $\Rightarrow (\text{number - number}) * \text{number}$ | $[\text{exp} \rightarrow \text{number}]$ |

Spring 2008

CIS 461/561 - Syntax and Grammars

10

Derivations and Parsing

A leftmost derivation corresponds to a (top-down) preorder traversal of the parse tree.

A rightmost derivation corresponds to a (bottom-up) postorder traversal, but in reverse.

Top-down parsers construct leftmost derivations.

(LL = Left-to-right traversal of input, constructing a Leftmost derivation)

Bottom-up parsers construct rightmost derivations in reverse order.

(LR = Left-to-right traversal of input, constructing a Rightmost derivation)

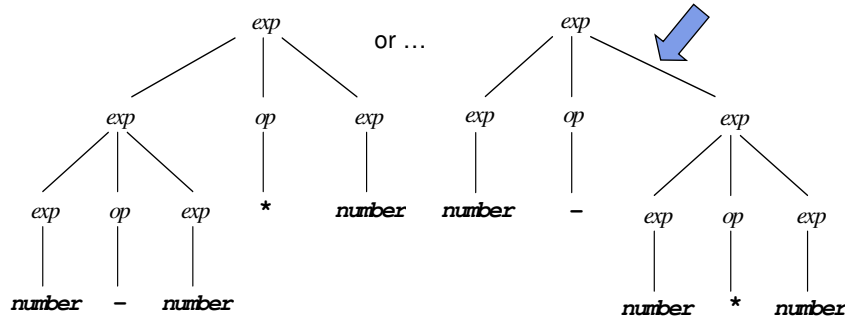
Spring 2008

CIS 461/561 - Syntax and Grammars

11

What is the parse tree if there are *no* parens: $34 - 3 * 42$

Correct one



The grammar is **ambiguous**, but is it a problem?

Yes ... *Semantics!*

Spring 2008

CIS 461/561 - Syntax and Grammars

12

Principle of Syntax-directed Semantics

- Parse tree is the basic model
- Semantic content is attached to the tree
- Thus the tree should reflect the structure of the eventual semantics

- Could describe as *semantics-based syntax*

Spring 2008

CIS 461/561 - Syntax and Grammars

13

Sources of Ambiguity

- Associativity and precedence of operators
- Sequencing (e.g., lists)
- Extent of a nested structure (dangling else)
- “Obscure” recursion (unusual)
 - $exp \rightarrow exp\ exp$

Spring 2008

CIS 461/561 - Syntax and Grammars

14

Dealing with ambiguity

- Change the language
 - Only feasible if language is being designed
 - Only makes sense if language is improved
- Change the grammar (but not the language!)
- Disambiguating rules
 - Almost like extensions of the grammar
- Can all ambiguity be removed?
 - Backtracking can handle it, but expense is great
 - Some specs of language left undefined

Spring 2008

CIS 461/561 - Syntax and Grammars

15

Standard Arithmetic Example

$$exp \rightarrow exp \text{ addop } term \mid term$$

$$\text{addop} \rightarrow + \mid -$$

$$term \rightarrow term \text{ mulop } factor \mid factor$$

$$\text{mulop} \rightarrow * \mid /$$

$$factor \rightarrow (exp) \mid \mathbf{number}$$

- This is a precedence “cascade”
- Also handles associativity of $1 + 2 + 3$

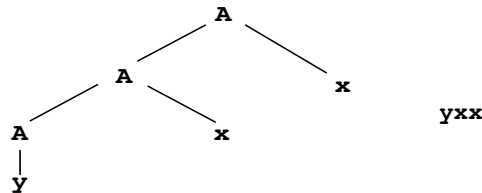
Spring 2008

CIS 461/561 - Syntax and Grammars

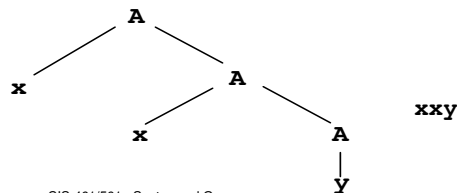
16

Repetition and Recursion

- Left recursion: $A \rightarrow A x \mid y$



- Right recursion: $A \rightarrow x A \mid y$



Spring 2008

CIS 461/561 - Syntax and Grammars

17

Repetition and Recursion

- Sometimes we care which way recursion goes: operator associativity
- Sometimes we don't: statement and expression sequences
 - Tree can descend to right or left, but order remains the same
- Parsing always has to pick a way!
- The tree may remove this information

Spring 2008

CIS 461/561 - Syntax and Grammars

18

Sequence Examples

- *one* or more stmts *separated* by a semicolon
 $stmt-seq \rightarrow stmt ; stmt-seq \mid stmt$
- *zero* or more stmts *terminated* by a semicolon
 $stmt-seq \rightarrow stmt ; stmt-seq \mid \epsilon$
- *one* or more stmts *separated* by a semicolon
 $stmt-seq \rightarrow stmt-seq ; stmt \mid stmt$
- *zero* or more stmts *preceded* by a semicolon
 $stmt-seq \rightarrow stmt-seq ; stmt \mid \epsilon$

Spring 2008

CIS 461/561 - Syntax and Grammars

19

Abstract Syntax Trees

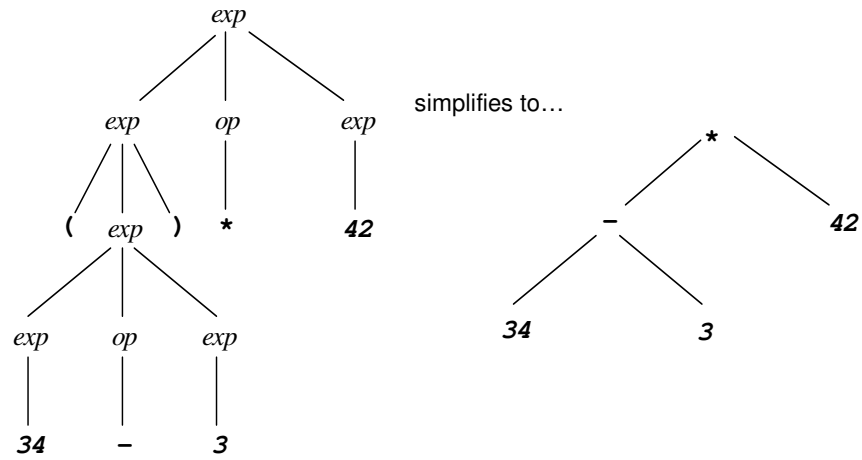
- Retain only the essential structure of the parse tree
- Omit parens, cascades, and “don’t-care” repetitive associativity
- Corresponds to actual internal tree structure produced by parser
- Use sibling lists for “don’t care” repetition
 - I.e., don’t retain grouping information

Spring 2008

CIS 461/561 - Syntax and Grammars

20

First Example (34 - 3) * 42

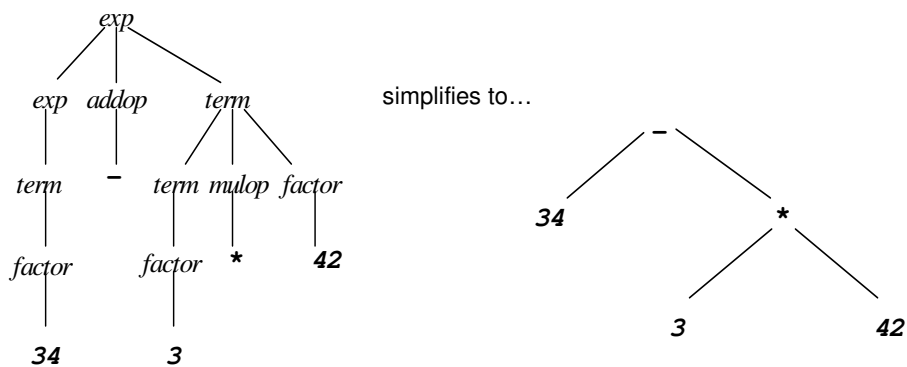


Spring 2008

CIS 461/561 - Syntax and Grammars

21

Last Example 34 - 3 * 42



Spring 2008

CIS 461/561 - Syntax and Grammars

22

Another Ambiguity Example

Incorrect attempt to add unary minus:

$$exp \rightarrow exp \text{ addop } term \mid term \mid - exp$$

$$\text{addop} \rightarrow + \mid -$$

$$term \rightarrow term \text{ mulop } factor \mid factor$$

$$\text{mulop} \rightarrow *$$

$$factor \rightarrow (exp) \mid number$$

Spring 2008

CIS 461/561 - Syntax and Grammars

23

Fixing the grammar

- Better: (but only one at beg. of an exp)

$$exp \rightarrow exp \text{ addop } term \mid term \mid - term$$

- Or maybe: (many at beginning of term)

$$term \rightarrow - term \mid term1$$

$$term1 \rightarrow term1 \text{ mulop } factor \mid factor$$

- Or maybe: (many anywhere)

$$factor \rightarrow (exp) \mid number \mid - factor$$

Spring 2008

CIS 461/561 - Syntax and Grammars

24

Another Ambiguity Example

Fragment of a grammar for conditional statements in C
(parentheses omitted)

$$\begin{aligned}
 \text{if-stmt} &\rightarrow \text{if } \text{expr } \text{stmt} \\
 &\quad | \text{if } \text{expr } \text{stmt } \text{else } \text{stmt} \\
 \text{stmt} &\rightarrow \text{if-stmt} \quad | \quad \text{S1} \quad | \quad \text{S2}
 \end{aligned}$$

Consider the statement

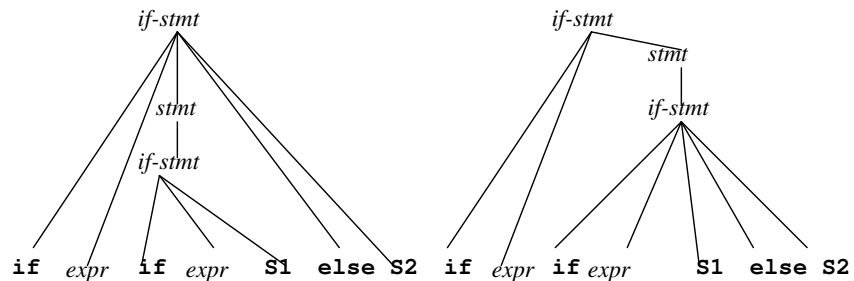
if expr if expr S1 else S2

Spring 2008

CIS 461/561 - Syntax and Grammars

25

Dangling else Example



if expr
 if expr S1
else S2

if expr
 if expr S1
else S2

Spring 2008

CIS 461/561 - Syntax and Grammars

26

How to fix dangling else?

- Add the keyword 'endif' to constrain the clause
 - But this would change the language and may not be acceptable
- Don't allow an if without an else
 - This really changes the language
 - ML does this, but if-else is an expression, not a statement there
- Use add hoc rules in the parsing and document the behavior (e.g., else goes with nearest if)
- Fix the grammar (hard, but elegant) ...

Spring 2008

CIS 461/561 - Syntax and Grammars

27

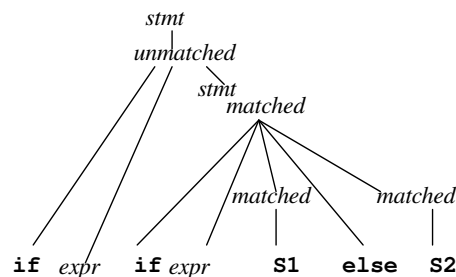
Unambiguous if-else

matched → **if** *expr* **matched** **else** *matched* | **S1** | **S2**

unmatched → **if** *expr* *stmt*

 | **if** *expr* **matched** **else** *unmatched*

stmt → *matched* | *unmatched*



Spring 2008

CIS 461/561 - Syntax and Grammars

28

Another Ambiguity in C

```

cast_expression → unary_expression
                  | ( type_name ) cast_expression
unary_expression → postfix_expression | ...
postfix_expression → primary_expression | ...
primary_expression → IDENTIFIER | CONSTANT
                   | STRING_LITERAL | ( expression )
type_name → ... | TYPE_NAME

```

Example:

<pre> typedef double x; printf("%d\n", (int) (x)-2); </pre>	<pre> int x = 1; printf("%d\n", (int) (x)-2); </pre>
---	--

Spring 2008

CIS 461/561 - Syntax and Grammars

29

Removing the cast ambiguity

- TYPE_IDs must be distinguished from other IDs *in the scanner*.
- Parser must build the symbol table (at least partially) to indicate whether an ID is a typedef or not.
- Scanner must consult the symbol table; if an ID is found as a typedef, return TYPE_ID, if not return ID.

Spring 2008

CIS 461/561 - Syntax and Grammars

30

Object Oriented Hierarchy

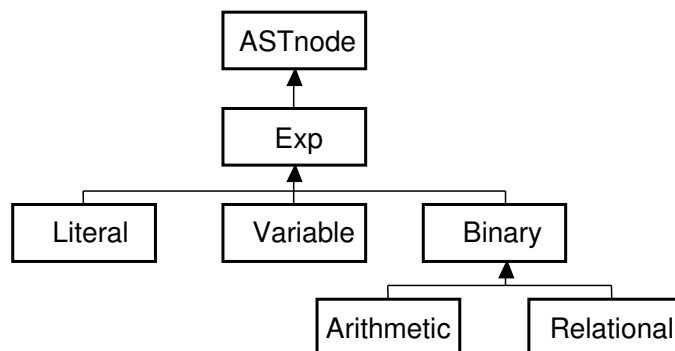
- Represent AST with hierarchy of classes
- Abstract base class of ASTnode
 - Derives from Token
- Subclasses for Program, Statements, Expressions
 - Hierarchy of Statements
 - Hierarchy of Expressions

Spring 2008

CIS 461/561 - Syntax and Grammars

31

Expression Hierarchy

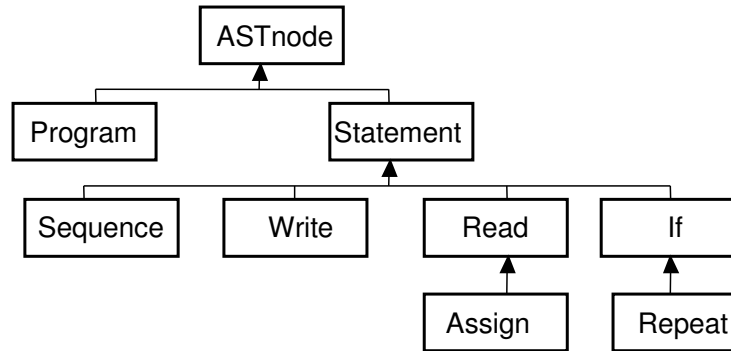


Spring 2008

CIS 461/561 - Syntax and Grammars

32

Statement Hierarchy



Spring 2008

CIS 461/561 - Syntax and Grammars

33

TINY Grammar

```

program → stmt_seq
stmt_seq → stmt_seq ; stmt | stmt
stmt → if exp then stmt_seq end
      | if exp then stmt_seq else stmt_seq end
      | repeat stmt_seq until exp
      | variable := exp
      | read variable
      | write exp
exp → simple_exp < simple_exp | simple_exp = simple_exp | simple_exp
simple_exp → simple_exp + term | simple_exp - term | term
term → term * factor | term / factor | factor
factor → ( exp ) | NUM | variable
variable → ID
  
```

Spring 2008

CIS 461/561 - Syntax and Grammars

34