



Syntactic Analysis / Parsing

- Goal: Convert token stream to abstract syntax
- Abstract syntax tree (AST):
 - Captures the structural features of the program
 - Primary data structure for remainder of compilation
- Three Part Plan
 - Study how context-free grammars specify syntax
 - Study algorithms for parsing / building ASTs
 - Study the miniJava Implementation



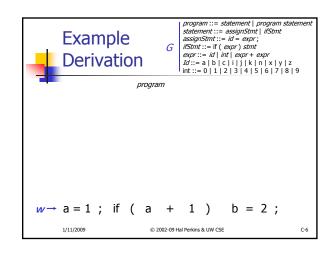
Context-free Grammars

- The syntax of most programming languages can be specified by a *context-free grammar* (CGF)
- Compromise between
 - REs, can't nest or specify recursive structure
- General grammars, too powerful, undecidable
- Context-free grammars are a sweet spot
 - Powerful enough to describe nesting, recursion Easy to parse; but also allow restrictions for speed
- Not perfect
 - Cannot capture semantics, as in, "variable must be declared," requiring later semantic pass
 - Can be ambiguous
- EBNF, Extended Backus Naur Form, is popular notation



Derivations and Parse Trees

- Derivation: a sequence of expansion steps, beginning with a start symbol and leading to a sequence of terminals
- Parsing: inverse of derivation
 - Given a sequence of terminals (a\k\a tokens) want to recover the nonterminals representing structure
- Can represent derivation as a parse tree, that is, the concrete syntax tree





Parsing

- Parsing: Given a grammar G and a sentence W in L(G), traverse the derivation (parse tree) for W in some standard order and do something useful at each node
 - The tree might not be produced explicitly, but the control flow of a parser corresponds to a traversal

1/11/2009

© 2002-09 Hal Perkins & UW CSF



"Standard Order"

- For practical reasons we want the parser to be *deterministic* (no backtracking), and we want to examine the source program from *left to right*.
 - (i.e., parse the program in linear time in the order it appears in the source file)

1/11/2009

© 2002-09 Hal Perkins & LIW CSF



Common Orderings

- Top-down
 - Start with the root
 - Traverse the parse tree depth-first, left-to-right (leftmost derivation)
 - LL(k)
- Bottom-up
 - Start at leaves and build up to the root
 Effectively a rightmost derivation in reverse(!)
 - LR(k) and subsets (LALR(k), SLR(k), etc.)

1/11/2009

© 2002-09 Hal Perkins & UW CSE



"Something Useful"

- At each point (node) in the traversal, perform some semantic action
 - Construct nodes of full parse tree (rare)
 - Construct abstract syntax tree (common)
 - Construct linear, lower-level representation (more common in later parts of a modern compiler)
 - Generate target code on the fly (1-pass compiler; not common in production compilers – can't generate very good code in one pass – but great if you need a quick 'n dirty working compiler)

1/11/2009

© 2002-09 Hal Perkins & UW CSE

C-10

C-12



Context-Free Grammars

- Formally, a grammar G is a tuple $\langle N, \Sigma, P, S \rangle$ where
 - N a finite set of non-terminal symbols
 - Σ a finite set of terminal symbols
 - P a finite set of productions
 - A subset of $N \times (N \cup \Sigma)^*$
 - S the start symbol, a distinguished element of N
 - If not specified otherwise, this is usually assumed to be the non-terminal on the left of the first production

1/11/2009

© 2002-09 Hal Perkins & UW CSE



Standard Notations

- a, b, c elements of Σ
- w, x, y, z elements of Σ*
- A, B, C elements of N
- X, Y, Z elements of $N \cup \Sigma$
- α , β , γ elements of $(N \cup \Sigma)^*$
- $A \rightarrow \alpha$ or $A ::= \alpha$ if $\langle A, \alpha \rangle$ in P

1/11/2009

© 2002-09 Hal Perkins & UW CSE



Derivation Relations (1)

- $\alpha A \gamma => \alpha \beta \gamma$ iff $A ::= \beta$ in P
 - derives
- A =>* w if there is a chain of productions starting with A that generates w
 - transitive closure

1/11/2009

© 2002-09 Hal Perkins & UW CSE



Derivation Relations (2)

- w A $\gamma =>_{lm}$ w $\beta \gamma$ iff A ::= β in P
 - derives leftmost
- $\alpha \land w =>_{rm} \alpha \beta w \text{ iff } A ::= \beta \text{ in } P$
 - derives rightmost
- We will only be interested in leftmost and rightmost derivations – not random orderings

1/11/2009

© 2002-09 Hal Perkins & UW CSE



Languages

- For A in N, $L(A) = \{ w \mid A = > * w \}$
- If S is the start symbol of grammar G, define L(G) = L(S)

1/11/2009

© 2002-09 Hal Perkins & UW CSE



C-13

C-15

Reduced Grammars

 Grammar G is reduced iff for every production A ::= α in G there is a derivation

$$S = > * x A z = > x \alpha z = > * xyz$$

- i.e., no production is useless
- Convention: we will use only reduced grammars

1/11/2009

© 2002-09 Hal Perkins & UW CSE



Ambiguity

- Grammar G is unambiguous iff every w in L(G) has a unique leftmost (or rightmost) derivation
 - Fact: unique leftmost or unique rightmost implies the other
- A grammar without this property is ambiguous
 - Note that other grammars that generate the same language may be unambiguous
- We need unambiguous grammars for parsing

1/11/2009

© 2002-09 Hal Perkins & UW CSE

C-17



Example: Ambiguous Grammar for Arithmetic Expressions

expr ::= expr + expr | expr - expr| expr * expr | expr | expr | int

int ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

- Exercise: show that this is ambiguous
 - How? Show two different leftmost or rightmost derivations for the same string
 - Equivalently: show two different parse trees for the same string

1/11/2009

© 2002-09 Hal Perkins & UW CSE

C-18

C-16

