

CS 460

Programming Languages

Fall 2008

Dr. Watts

(Week 5)



Grading

- Review/Tutorial Exercises: 20 points each
- Projects: 50 points each
- 10 % off per day for late assignments
- No more than 50 % for assignments that crash
- No more than 25 % for assignments that do not compile
- No more than 0% for assignments that can not be read
- 5 points for homework assignments – due the next class meeting



Describing Syntax



- Language is a set of strings of characters from some alphabet.
- The strings of a language are called sentences or statements.
- Smallest units of the statements are words or lexemes.
- Syntax rules of a language describe what words are in the language and how they should be ordered.
- Natural languages (such as English) have a complex and extensive set of syntactical rules.
- Programming languages have a relatively simple set of syntactical rules.

Context Free Grammars and Backus-Naur Form



- Context Free Grammar (CFG) – order of syntactical elements is important – meaning is not.
- Meaning is determined by context semantics.
- Backus-Naur Form (BNF)
 - ALGOL 58
 - John Backus – 1959
 - Peter Naur – 1960
- BNF is a natural notation for describing syntax



BNF

- Set of terminal symbols (T)
- Set of non-terminal symbols (N)
- Start symbol ($S \in N$)
- Set of production rules (P)
 - $\langle \text{non-terminal} \rangle \rightarrow$ string of terminal and $\langle \text{non-terminal} \rangle$ symbols
 - $\langle \text{non-terminal} \rangle ::=$ string of terminal and $\langle \text{non-terminal} \rangle$ symbols



BNF Grammar (Example 1)

- $T = \{=, A, B, C, +, *, (,)\}$
- $N = \{\langle \text{assign} \rangle, \langle \text{id} \rangle, \langle \text{expr} \rangle\}$
- $S = \langle \text{assign} \rangle$
- $P = \{$
 - $\langle \text{assign} \rangle \rightarrow \langle \text{id} \rangle = \langle \text{expr} \rangle$
 - $\langle \text{id} \rangle \rightarrow A \mid B \mid C$
 - $\langle \text{expr} \rangle \rightarrow$
 - $\langle \text{id} \rangle + \langle \text{expr} \rangle$
 - $\langle \text{id} \rangle * \langle \text{expr} \rangle$
 - $(\langle \text{expr} \rangle)$
 - $\langle \text{id} \rangle$

Derivation of $A = B + (C * A)$



$\langle \text{assign} \rangle$

$\Rightarrow \langle \text{id} \rangle = \langle \text{expr} \rangle$

$\Rightarrow A = \langle \text{expr} \rangle$

$\Rightarrow A = \langle \text{id} \rangle + \langle \text{expr} \rangle$

$\Rightarrow A = B + \langle \text{expr} \rangle$

$\Rightarrow A = B + (\langle \text{expr} \rangle)$

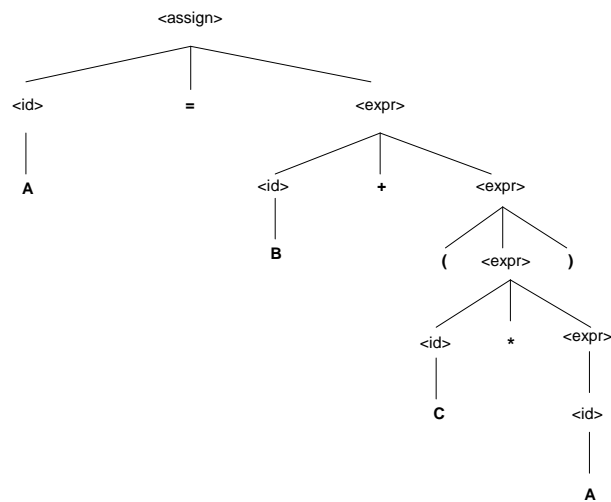
$\Rightarrow A = B + (\langle \text{id} \rangle * \langle \text{expr} \rangle)$

$\Rightarrow A = B + (C * \langle \text{expr} \rangle)$

$\Rightarrow A = B + (C * \langle \text{id} \rangle)$

$\Rightarrow A = B + (C * A)$

Parse tree for $A = B + (C * A)$



Taunt Generator Grammar



```
< taunt > ::= < sentence > | < taunt > < sentence > | < noun >!
< sentence > ::= < past-rel > < noun-phrase > | < present-rel > < noun-phrase >
| < past-rel > < article > < noun >
< noun-phrase > ::= < article > < modified-noun >
< modified-noun > ::= < noun > | < modifier > < noun >
< modifier > ::= < adjective > | < adverb > < adjective >
< present-rel > ::= your < present-person > < present-verb >
< past-rel > ::= your < past-person > < past-verb >
< present-person > ::= steed | king | first-born
< past-person > ::= mother | father | grandmother | grandfather | godfather
< noun > ::= hamster | coconut | duck | herring | newt | peril | chicken | vole
| parrot | mouse | twit
< present-verb > ::= is | "masquerades as"
< past-verb > ::= was | personified
< article > ::= a
< adjective > ::= silly | wicked | sordid | naughty | repulsive | malodorous
| ill-tempered
< adverb > ::= conspicuously | categorically | positively | cruelly |
incontrovertibly
```

Team 1



- a. your grandmother was a hamster chicken!
- b. newt! coconut! herring! your godfather masquerades as a naughty chicken
- c. your mother is a repulsive duck
- d. your grandmother personified a repulsive twit, your first-born masquerades as a conspicuously silly newt

Team 2



- a. Twit! Your first born masquerades as a mouse.
- b. Your mother is a silly duck
- c. Your steed is a categorically sordid coconut.
- d. Newt! Your grandfather was a conspicuously naughty chicken.

Team 3



- a. Hamster! Your firstborn masquerades as a coconut. Your father was a parrot.
- b. Your firstborn was a twit. Your mother is a hamster. Your king masquerades as a herring.
- c. Your steed personified a incontrovertible naughty chicken.
- d. Twit! Your steed personified an ill-tempered herring.

Team 4



- A. Twit! Your grandfather masquerades as a herring.
- B. Coconut! Your king was a hamster.
- C. Vole! Your father personified a mouse your grandfather was a chicken.
- D. Your first-born is a positively naughty parrot.

Team 5



- 1) Coconut!
- 2) Your grandfather personified a twit.
- 3) Your first-born personified a sordid hamster.
- 4) Coconut! Your mother was a newt.

Team 6



- a. Hamster! Hamster! Hamster! Hamster!
- b. Your ill-tempered mother masquerades as a conspicuously sordid newt.
- c. Your steed is a positively repulsive coconut.
- d. Your father was a naughty newt. Your mother was a wicked twit!

Team 7



- a. your firstborn masquerades as a naughty coconut
- b. your mother is a newt
- c. your father personified a peril
- d. your grandmother masquerades as a hamster your mother personified a coconut duck!



Team 8

- a. Your first-born masquerades as a categorically sordid newt.
- b. Your godmother personified a positively naughty newt.
- c. Your father was a conspicuously maodorous coconut.
- d. Your king is a duck.



BNF Grammar (Example 2)

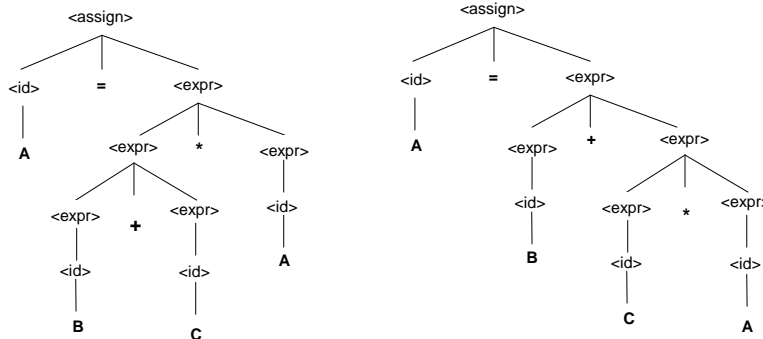
- $T = \{=, A, B, C, +, *, (,)\}$
- $N = \{<assign>, <id>, <expr>\}$
- $S = <assign>$
- $P = \{$

```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> →      <expr> + <expr>
          |      <expr> * <expr>
          |      ( <expr> )
          |      <id>
```

- $\}$
- Parse tree for $A = B + C * A$?

Example 2

Parse Trees for $A = B + C * A$



- Ambiguous

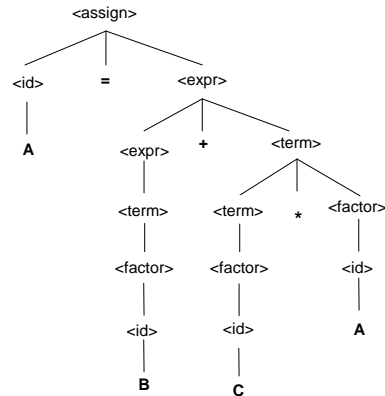
BNF Grammar (Example 3)



- $T = \{=, A, B, C, +, *, (,)\}$
- $N = \{\langle \text{assign} \rangle, \langle \text{id} \rangle, \langle \text{expr} \rangle, \langle \text{term} \rangle, \langle \text{factor} \rangle\}$
- $S = \langle \text{assign} \rangle$
- $P = \{$
 - $\langle \text{assign} \rangle \rightarrow \langle \text{id} \rangle = \langle \text{expr} \rangle$
 - $\langle \text{id} \rangle \rightarrow A \mid B \mid C$
 - $\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle + \langle \text{term} \rangle \mid \langle \text{term} \rangle$
 - $\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle * \langle \text{factor} \rangle \mid \langle \text{factor} \rangle$
 - $\langle \text{factor} \rangle \rightarrow (\langle \text{expr} \rangle) \mid \langle \text{id} \rangle$
- Parse tree for $A = B + C * A$?

Example 3

Parse Tree for $A = B + C * A$



- Operator precedence

Semantic Analysis



- The meaning of the expressions, statements and program units.
- Static semantics
 - At compile time
- Dynamic semantics
 - At run time
- Attribute grammars
- Denotational semantics

LL(1) Grammar for a Small Programming Language



T = {begin, end, ;, =, A, B, C, +, *}
 N = {<program>, <stmt_list>, <stmt>, <var>, <stmt_tail>, <expression>, <expr_tail>}
 Start = <program>

P =
 {
 1) <program> → **begin** <stmt_list> **end**
 2) <stmt_list> → <stmt> <stmt_tail>
 3) <stmt_tail> → ; <stmt_list>
 4) <stmt_tail> → λ
 5) <stmt> → <var> = <expression>
 6) <var> → **A**
 7) <var> → **B**
 8) <var> → **C**
 9) <expression> → <var><expr_tail>
 10) <expr_tail> → + <var><expr_tail>
 11) <expr_tail> → * <var><expr_tail>
 12) <expr_tail> → λ
 }

Example 1:

```
begin
    A = B + C;
    C = A * B
end
```

Example 2:

```
begin
    A = B + A * C;
    C = A * B;
end
```

Derivation of Example 1



```
Start <program>
1 begin <stmt_list> end
2 <stmt> <stmt_tail> end
5 <var> = <expression> <stmt_tail> end
6 A = <expression> <stmt_tail> end
9 <var> <expr_tail> <stmt_tail> end
7 B <expr_tail> <stmt_tail> end
10 + <var> <expr_tail> <stmt_tail> end
8 C <expr_tail> <stmt_tail> end
12 λ <stmt_tail> end
3 ; <stmt_list> end
2 <stmt> <stmt_tail> end
5 <var> = <expression> <stmt_tail> end
8 C = <expression> <stmt_tail> end
9 <var> <expr_tail> <stmt_tail> end
6 A <expr_tail> <stmt_tail> end
11 * <var> <expr_tail> <stmt_tail> end
7 B <expr_tail> <stmt_tail> end
12 λ <stmt_tail> end
4 λ end
```

• Done

Parsing



- Top Down
 - Recursive Descent
- Bottom Up
 - Shift Reduce