Class Information

- Second homework will be posted today or early tomorrow. Due next Tuesday before class.

Review: Context Free Grammars (CFGs)

- A formalism for describing languages
- A CFG $\mathcal{G}$ is a quadruple $\mathcal{G} =< T, N, P, S >$:
  1. A set T of terminal symbols (**tokens**)
  2. A set N of nonterminal symbols
  3. A set P production (rewrite) rules
  4. A special start symbol S
- The language $L(\mathcal{G})$ is the set of sentences of terminal symbols in T^* that can be derived from the start symbol S: $L(\mathcal{G}) = \{w \in T^* | S \Rightarrow^* w\}$

CFGs are rewrite systems with restrictions on the form of rewrite (production) rules that can be used

Review: BNF Syntax

Terminal Symbol: **Symbol-In-Boldface**

Non-Terminal Symbol: *Symbol-In-Angle-Brackets*

Production Rule:

Non-Terminal ::= Sequence of Symbols

or

Non-Terminal ::= Sequence | Sequence | ...

Alternative Symbol: |

Empty String: ϵ

Grammars and Programming Languages

Many grammars may correspond to one programming language.

Good grammars:

- capture the logical structure of the language
  $\Rightarrow$ structure carries some semantic information
  (example: expression grammar)
- use meaningful names
- are easy to read,
- are unambiguous
- ...

What's the problem with ambiguity?

Review: Ambiguous Grammars

"Time flies like an arrow; fruit flies like a banana."

A grammar $\mathcal{G}$ is ambiguous iff there exist a $w \in L(\mathcal{G})$ such that there are

1. two distinct parse trees for w, or
2. two distinct leftmost derivations for w, or
3. two distinct rightmost derivations for w.

We want a unique semantics of our programs, which typically requires a unique syntactic structure.

Arithmetic Expression Grammar

<start> ::= <expr>

<expr> :: = <expr> + <expr> |
<expr> - <expr> |
<expr> * <expr> |
<expr> / <expr> |
<expr> ^ <expr>|
<d> | <l>

<d> :: = 0 | 1 | 2 | 3 | ...| 9

<l> :: = a | b | c | ...| z

Changing the Grammar to Impose Precedence

<expr> ::= <expr> $-$ <expr> |

<term>

<term> ::= <term> $*$ <term> |

<factor>

<factor> ::= 0 | 1 | 2 | 3 | … | 9

Grouping In Parse Tree Now Reflects Precedence

Parse "$8 - 3 * 2$":

Precedence

- Low Precedence:

  Addition $+$ and Subtraction $-$
- Medium Precedence:

  Multiplication $*$ and Division $/$
- Highest Precedence:

  Exponentiation ^

$\Rightarrow$ Ordered lowest to highest in grammar.

Still Have Ambiguity...

$3 - 2 - 1$ still a problem:

- Grouping of operators of same precedence not disambiguated.
- Non-commutative operators: only one parse tree correct.

Imposing Associativity

Simple grammars with left/right recursion for $-$:

our choices:

<expr> ::= <d> $-$ <expr> |
<d>
<d> ::= 0 | 1 | 2 | 3 | ... | 9

or

<expr> ::= <expr> $-$ <d> |
<d>
<d> ::= 0 | 1 | 2 | 3 | ... | 9

Associativity

- Deals with operators of same precedence
- Implicit grouping or parenthesizing
- Left to Right: $*$, $/$, $+$, $-$
- Right to Left: ^

Complete, Unambiguous Arithmetic Expression Grammar

<start> ::= <e>

<e> :: = <e> + <t> | <e> − <t> |<t>

<t> :: = <t> * <f> | <t> / <f> |<f>

<f> :: = <g> ^ <f> | <g>

<g> :: = (<e>) | <n> | <i>

<n> :: = 0 | 1 | 2 | ... | 9

<i> :: = a | b | c | ... | z

Dealing with Ambiguity

1. Can't *always* remove an ambiguity from a grammar by restructuring productions
2. An inherently ambiguous language does not possess an unambiguous grammar
3. There is no algorithm that can examine an arbitrary context-free grammar and tell if it is ambiguous, i.e., detecting ambiguity in context-free grammars is an *undecidable* problem

Abstract versus Concrete Syntax

Concrete Syntax:

representation of a construct in a particular language, including placement of keywords and delimiters

Abstract Syntax:

structure of meaningful components of each language construct

Abstract versus Concrete Syntax

Same abstract syntax, different concrete syntax:

Pascal

```
while x <> A[i] do
     i := i + 1
end
```

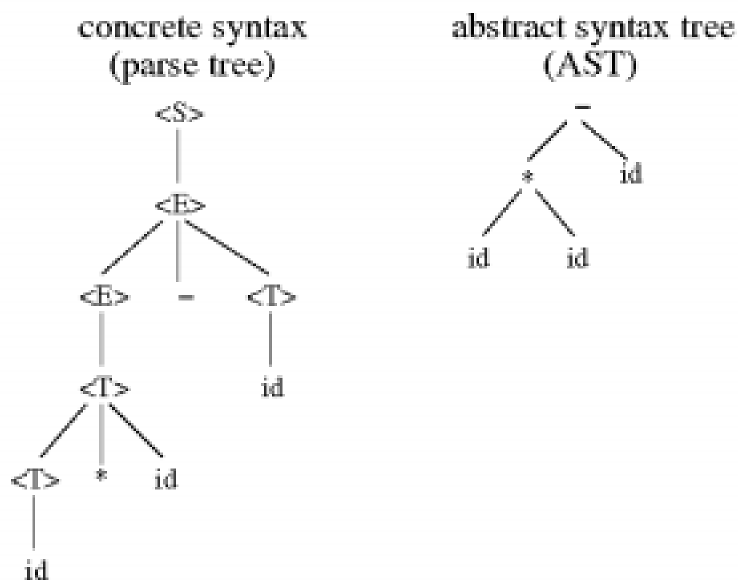
C

```
while ( x != A[i] )
     i = i + 1;
```

Example

<S> ::= <E>
<E> ::= <E> - <T> | <T>
<T> ::= <T> * **id** | **id**

Consider `A*B-C`:



Regular vs. Context Free

- All regular languages are context free languages
- Not all context free languages are regular languages

Example:

N ::= X | Y
X ::= a | X b
Y ::= c | Y c

is equivalent to:
$\mathbf{ab}^*|\mathbf{c}^+$

Is $\{\mathbf{a}^n\mathbf{b}^n | n \geq 0\}$ *a context free language?*

Is $\{\mathbf{a}^n\mathbf{b}^n | n \geq 0\}$ *a regular language?*

Regular Grammars

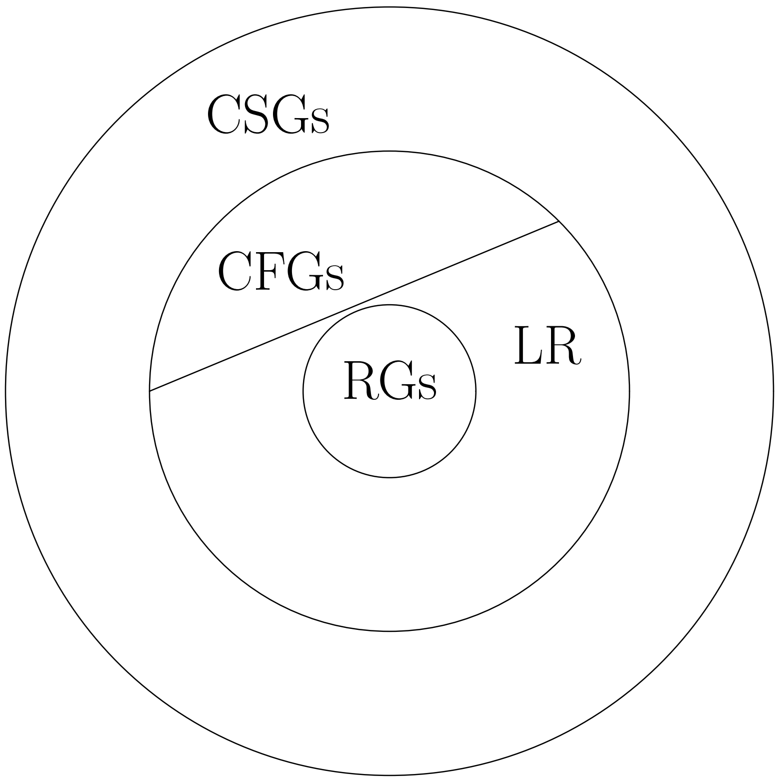
CFGs with restrictions on the shapes of production rules.

Left-linear:
N ::= X a b
X ::= a | X b

Right-linear:
N ::= b | b Y
Y ::= a b | a b Y

Complexity of Parsing

<u>Classification of languages that can be recognized by specific grammars</u>



Complexity:

Regular grammars	*dfa*s	$\mathbf{O}(n)$
LR grammars	Knuth's algorithm	$\mathbf{O}(n)$
Arbitrary CFGs	Early's algorithm	$\mathbf{O}(n^3)$
Arbitrary CSGs	*lba*s	*P-SPACE COMPLETE*