Discussion #2

Grammars and Languages

Topics

- Set basics
- Programming languages / compilers
- Grammars as set generators
- Languages as defined by grammars

Set Basics

- Set = unordered collection of distinct items
  - Notation: curly braces enclose items, e.g. {0, a, xy}
  - Unordered: {0, a, xy} = {a, xy, 0}
  - Distinct: {0, 0, a, xy} is not a set; contains duplicates
  - Can name sets: A = {0, a, xy}; B = {a, b}
- Empty set: { } or $\varnothing$
- Common operations
  - Element: $0 \in \{0, a, xy\}$; $0 \in A$; $0 \notin B$; $aa \notin A$
  - Count or cardinality: $|\{0, a, xy\}| = 3$; $|A| = 3$; $|B| = 2$
  - Union: $A \cup B = \{0, a, xy, b\}$
  - Intersection: $A \cap B = \{a\}$
  - Difference: $A - B = \{0, xy\}$; $B - A = \{b\}$
- Subset
  - Subset: $\{0, xy\} \subseteq \{0, a, xy\}$; $\varnothing \subseteq A$; $A \subseteq A$; $B \not\subseteq A$
  - Proper Subset: $\{0, xy\} \subset \{0, a, xy\}$; $\varnothing \subset A$; $A \not\subset A$

Programming Language Specification

- Define the alphabet—a set of symbols that can be used to construct programs
- Define the set of all correct programs
- Define the “meaning” of all correct programs

Alphabets

- An alphabet (or vocabulary) V is a nonempty set of symbols.
- Examples:
  - $V_1 = \{0, 1\}$
  - $V_2 = \{a, b, \ldots, z, \ldots\}$

Strings

- An element of an alphabet set is called a *letter*, *character*, or *symbol*
- A *string* over an alphabet is a sequence of symbols from the alphabet
- A string is also called a *sequence*, *word*, or *sentence*
- Length of a string α is denoted by $\#\alpha$ or $|\alpha|$.

Strings

- A string of *m* symbols is called a string of *length m.*
- If *m* = 0, the string is called the empty string and is denoted by ε.
- The set of strings over an alphabet *V* of length *n* is denoted by V^n.
- The set of all strings is V^*.
- The set of all nonempty strings is V^+.
- If *V* = {0, 1, 2, 3, *x*, *y*, *z*, +, *,), (} then
  - $V^1 = V$
  - V^2 = {00, 01, … 0(, … (()
  - $V^0 = \{\varepsilon\}$
  - V^* = All combinations of all lengths
  - $V^+ = V^* - \{\varepsilon\}$

Languages

- Definition: A *language L* is a subset of V^*, i.e. $L \subseteq V^*$.
  - A programming language can be thought of as the set of all possible programs, where a program is a valid string (a very long string).
  - Programs with syntax errors are not in the set.
- A language can be finite or infinite.
  - Programming languages are infinite.
  - i.e. there are an infinite number of programs.

Language Representation

- A finite language can be specified by enumerating all of its sentences.
- An infinite language cannot be specified by enumeration, but can be specified by a generative device called a *grammar.*

Grammars

- A grammar is a way to specify the set of all legal sentences of a language (i.e. to specify the set of all legal programs of a programming language).
- Grammars are defined recursively (i.e. some elements are defined in terms of themselves). Recursive definitions are also called inductive definitions (i.e. they induce, rather than enumerate, the set of strings).

Grammar: Inductive Definition

Let's define a simple kind of arithmetic expression, A.

- Basis Rules:
  - A Variable is an A.
  - An Integer is an A.
- Inductive Rules:
  - If E_1 and E_2 are A's, so is $(E_1 + E_2)$.
  - If E_1 and E_2 are A's, so is $(E_1 * E_2)$.

Examples:

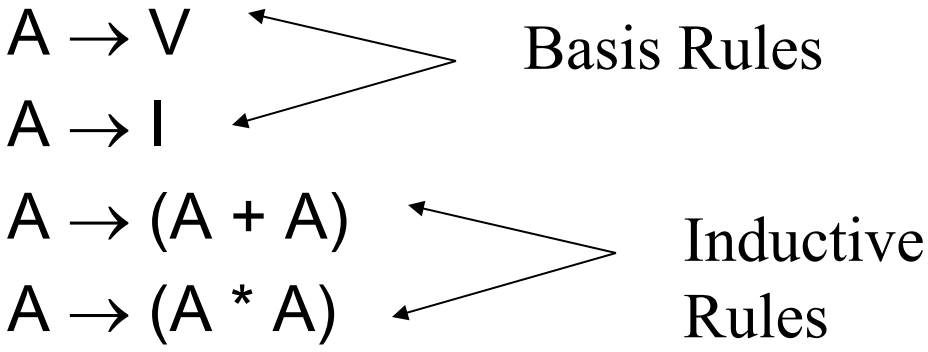
x, y, 3, 12, (x + y), (z * (x + y)), ((z * (x + y)) + 12)

Not Examples:

*3, (x ++ y), x + y

Writing Inductive Definitions as Productions

- *Productions* use *terminal symbols*, *non-terminal symbols* (also called *syntactical categories*) and *meta-symbols* to define basis and inductive rules.
- For our example:

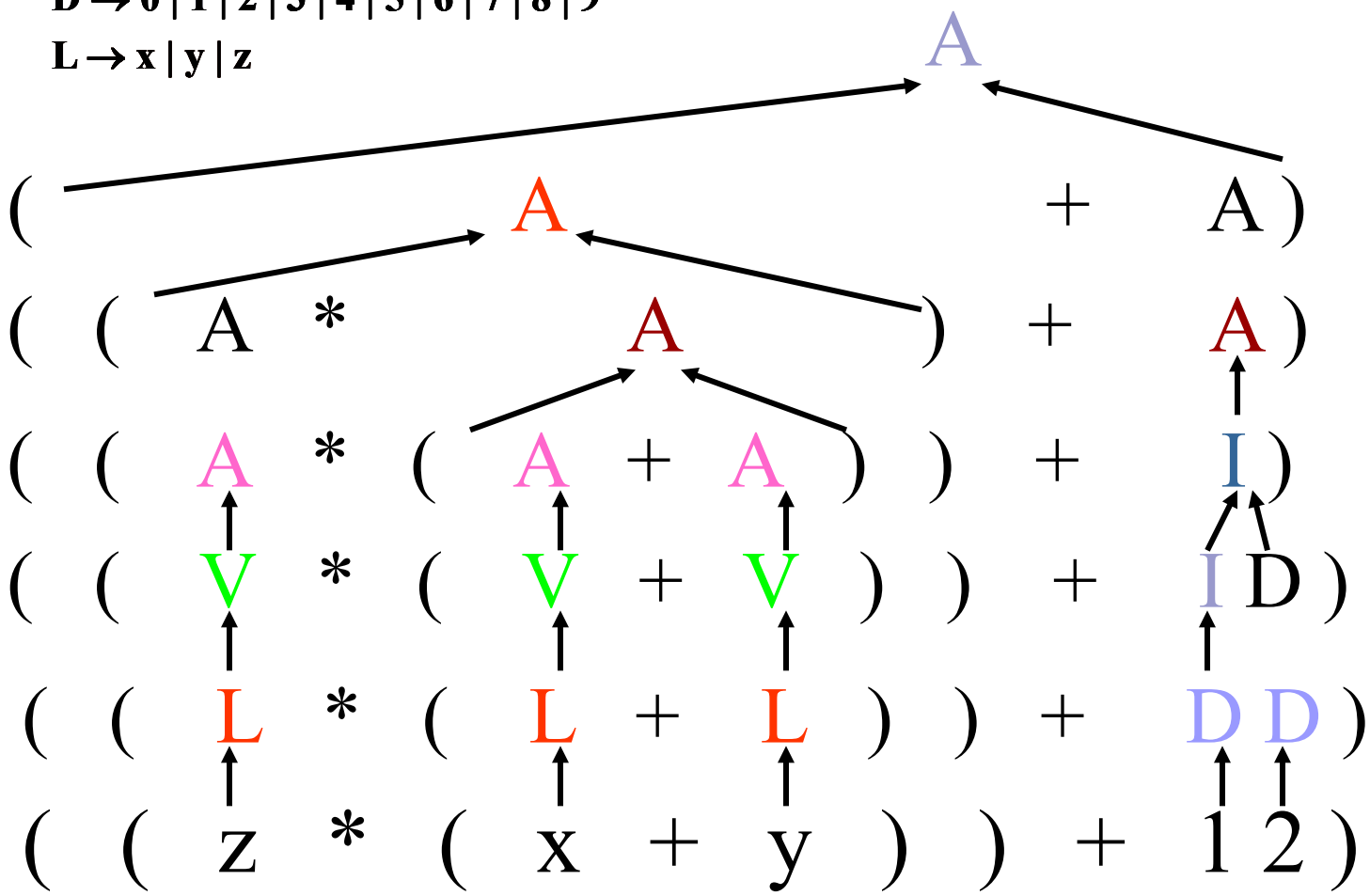


Full Grammar for Simple Arithmetic Expressions

Let's define all non-terminals:

1. $A \rightarrow V \mid I \mid (A + A) \mid (A * A)$
2. $V \rightarrow L \mid VL \mid VD$
3. $I \rightarrow D \mid ID$
4. $D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
5. $L \rightarrow x \mid y \mid z$

1. A → V | I | (A + A) | (A * A)
2. V → L | VL | VD
3. I → D | ID
4. D → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
5. L → x | y | z



Lexical Analyzers and Parsers

- Lexical analyzers
  - Input: symbols of length 1
  - Output: classified tokens
- Parsers
  - Input: classified tokens
  - Output: parse tree (syntactically correct program)

Simple English Grammar

- Symbols:
  - S: sentence
  - V: verb
  - O: object
  - A: article
  - N: noun
  - SP: subject phrase
  - VP: verb phrase
  - NP: noun phrase
- Rules:
  - $S \rightarrow SP\ VP$
  - $SP \rightarrow A\ N$
  - $A \rightarrow a \mid the$
  - $N \rightarrow monkey \mid banana \mid tree$
  - $VP \rightarrow V\ O$
  - $V \rightarrow ate \mid climbs$
  - $O \rightarrow NP$
  - $NP \rightarrow A\ N$

Parsing an English sentence