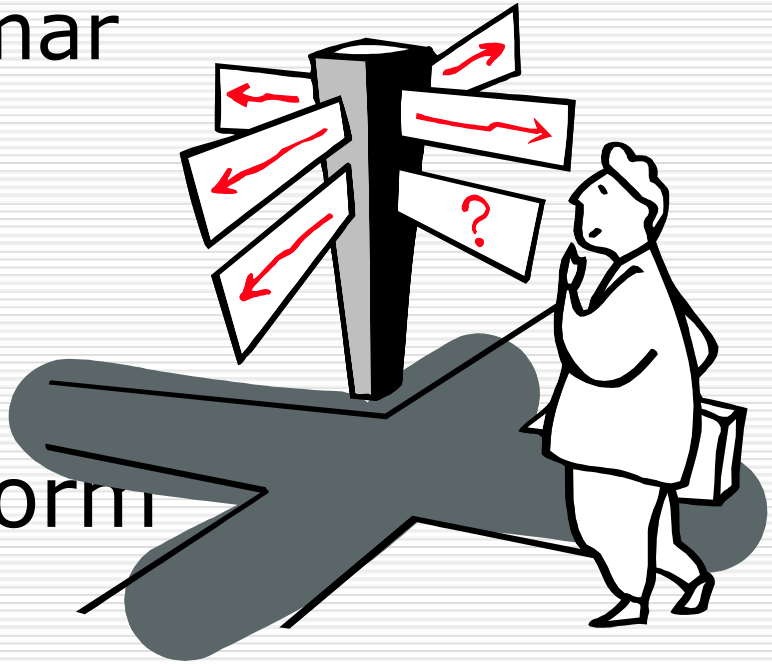
2. Context-Free Languages

- 2.1 Context-free Grammars
  - Derivation
  - Parse tree
  - Ambiguity
  - Chomsky normal form
- 2.2 Pushdown Automata
  - CFG = PDA
  - Deterministic PDA
- 2.3 Properties of Context-free Languages
  - Pumping lemma for CFL's
  - Closure properties
  - Decision properties

2.1 Context-Free Grammars

- Context-free Grammar
- Derivations
- Parse Trees
- Ambiguity
- Chomsky normal form



Context-Free Grammar

- Context-Free Languages (CFL's) played a central role in natural languages since the 1950's and in compilers since the 1960's
- Context-Free Grammars (CFG's) are the basis of BNF syntax
- Today CFL's are increasingly important for XML and their DTD's

Palindromes

- $L_{pal} = \{w \in \Sigma^* \mid w = w^R\}$
  - e.g. otto $\in L_{pal}$, 1001 $\in L_{pal}$
- L_{pal} is not regular, proven by Pumping Lemma testing
- Inductively define L_{pal} over $\Sigma = \{0, 1\}$ as
  - Basis: ε, 0, 1 are palindromes
  - Induction: if w is a palindrome, so are 0w0 and 1w1. Nothing else is a palindrome.

Grammar for Palindromes

- Use context-free grammar to formally express recursive definitions of palindromes

  1. $P \rightarrow \varepsilon$
  2. $P \rightarrow 0$
  3. $P \rightarrow 1$
  4. $P \rightarrow 0P0$
  5. $P \rightarrow 1P1$

  - 0, 1 are terminals
  - P is a variable (or non-terminal, or syntactic category)
  - P is in this grammar also a start variable
  - 1-5 are productions (or rules)

Formal Definition of CFG's

- A context-free grammar is a quadruple

  $G = \{V, T, P, S\}$

  Where

  V is a finite set of variables

  T is a finite set of terminals

  P is a finite set of productions of the form
  $A \rightarrow \alpha$, where A is a variable and $\alpha \in (V \cup T)^*$

  $S \in V$ is the start variable

Examples

- $G_{pal}=(\{P\}, \{0,1\}, A, P)$, where $A=\{P\rightarrow\varepsilon, P\rightarrow 0, P\rightarrow 1, P\rightarrow 0P0, P\rightarrow 1P1\}$
- Sometimes we group productions with the same head, e.g. $P \rightarrow \varepsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$.

Context-Free Grammars

- generate strings by repeated replacement of non-terminals with string of terminals and non-terminals
  - write down start variable (non-terminal)
  - replace a non-terminal with the right-hand-side of a rule that has that non-terminal as its left-hand-side.
  - repeat above until no more non-terminals

Example

- A simple grammar generates strings of 0's and 1's such that each block of 0's is followed by at least as many 1's.

  $S \rightarrow AS \mid \varepsilon$

  $A \rightarrow 0A1 \mid A1 \mid 01$

- $S \Rightarrow AS \Rightarrow A \Rightarrow 0A1 \Rightarrow 0A11 \Rightarrow 00111$
- a derivation of the string 00111

Derivations

Let $G=(V, T, P, S)$ be a CFG, $A\in V$, $\alpha,\beta,\gamma\in(V\cup T)^*$, and $A\rightarrow\gamma \in P$

- Then we can write $\alpha A\beta \Rightarrow \alpha\gamma\beta$, and say that $\alpha A\beta$ **derives** $\alpha\gamma\beta$
  - E.g. $011AS \Rightarrow 0110A1S$
- We define $\Rightarrow^*$ be the reflexive and transitive closure of $\Rightarrow$:
  - Basis: $\alpha \Rightarrow^* \alpha$
  - Induction: if $\alpha \Rightarrow^* \beta$ and $\beta \Rightarrow \gamma$, then $\alpha \Rightarrow^* \gamma$

Example

- $011AS \Rightarrow^* 011AS$
- $011AS \Rightarrow^* 0110A1S$
- $011AS \Rightarrow 011A1S \Rightarrow 011011S \Rightarrow 011011$
- Note: at each step, we might have a choice of variable to replace, and we might have several rules to apply for the variable to be replaced

Leftmost and Rightmost Derivations

- Leftmost derivation: at each step, replace the leftmost variable by its production body
  - $S \underset{lm}{\Rightarrow} AS \underset{lm}{\Rightarrow} A1S \underset{lm}{\Rightarrow} 011S \underset{lm}{\Rightarrow} 011AS \underset{lm}{\Rightarrow} 0110A1S \underset{lm}{\Rightarrow} 0110011S \underset{lm}{\Rightarrow} 0110011$
- Rightmost derivation: at each step, replace the rightmost variable by its production body
  - $S \underset{rm}{\Rightarrow} AS \underset{rm}{\Rightarrow} AAS \underset{rm}{\Rightarrow} AA \underset{rm}{\Rightarrow} A0A1 \underset{rm}{\Rightarrow} A0011 \underset{rm}{\Rightarrow} A10011 \underset{rm}{\Rightarrow} 0110011$

Language of a CFG

- The language of G = (V, T, P, S), denoted L(G) is:

  $$\{w \in \Sigma^* : S \Rightarrow^* w\}$$

  I.e. the set of strings over T* derivable from the start variable.
- If G is a CFG, we call L(G) a context-free language.

Aside: Notation

- a, b, … are terminals.
- …, y, z are strings of terminals.
- Greek letters are strings of variables and/or terminals, often called sentential forms.
- A, B, … are variables.
- …, Y, Z are variables or terminals.
- S is typically the start variable.

CFG Example

- Arithmetic expressions over {+, *, (,), a}
  - (a + a) * a
  - a * a + a + a + a + a

- A CFG generating this language:

  <expr> → <expr> * <expr>

  <expr> → <expr> + <expr>

  <expr> → (<expr>) | a

CFG Example

- A derivation of the string: a+a*a

  <expr> ⇒ <expr> * <expr>

  ⇒ <expr> + <expr> * <expr>

  ⇒ a + <expr> * <expr>

  ⇒ a + a * <expr>

  ⇒ a + a * a

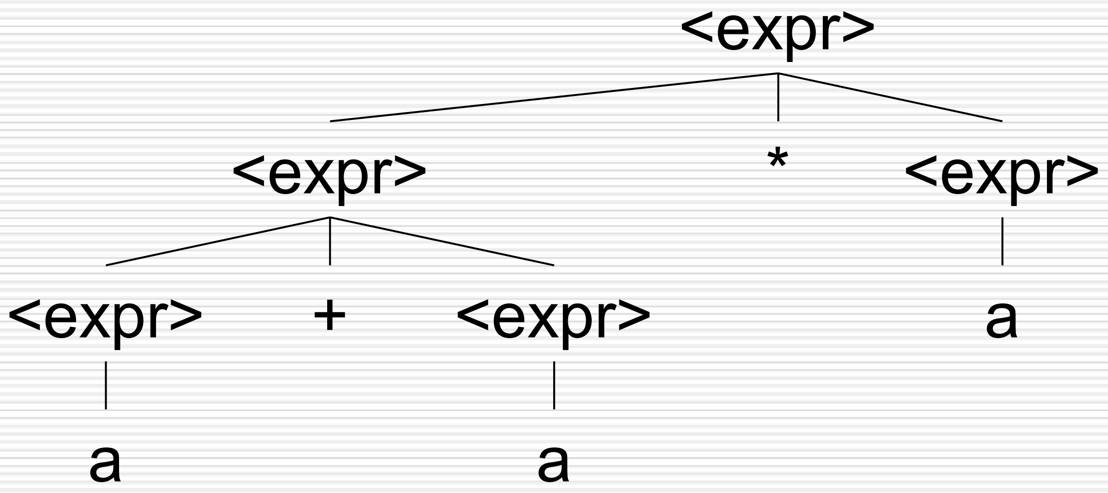
<expr> → <expr> * <expr>

<expr> → <expr> + <expr>

<expr> → (<expr>) | a

Parse Tree

- Easier way to picture derivation: parse tree



- grammar encodes grouping information; this is captured in the parse tree.

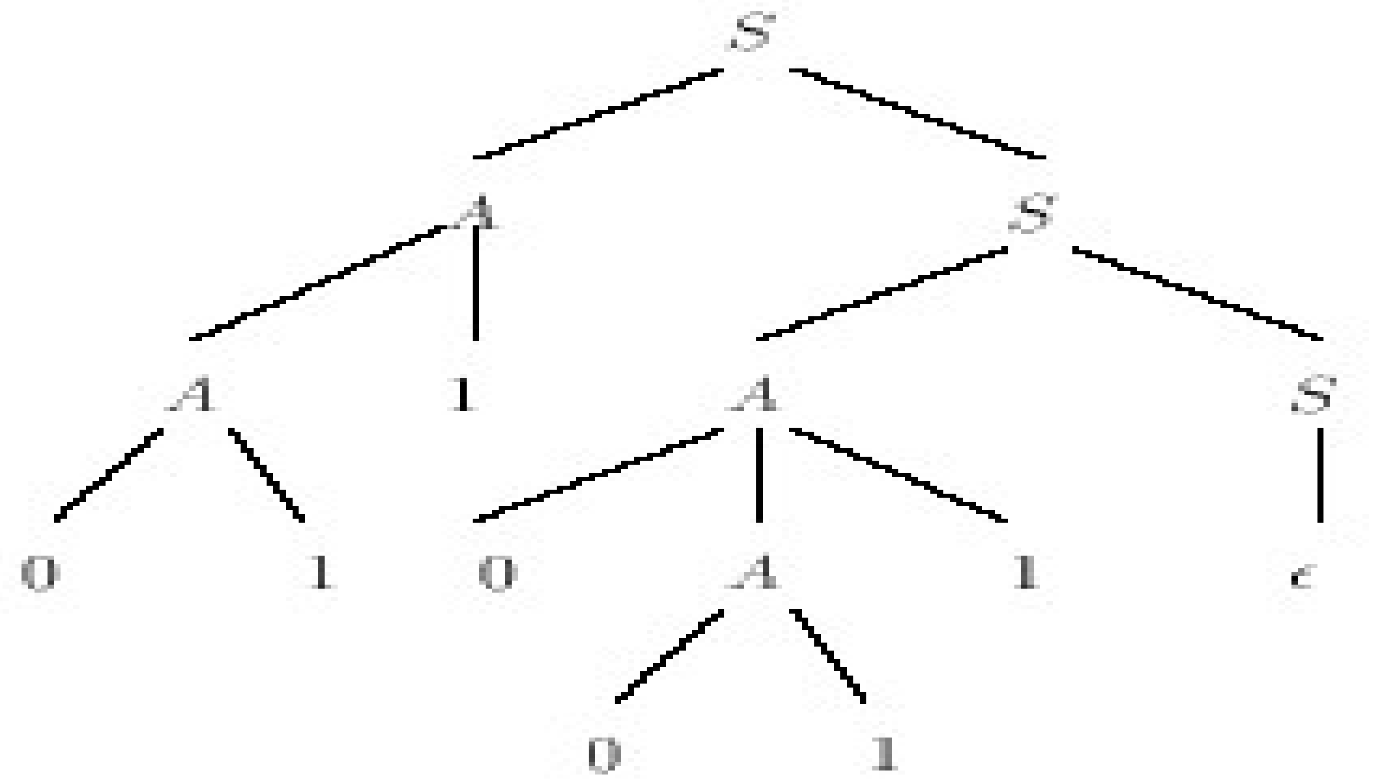
Constructing Parse Tree

- Nodes = variables, terminals, or ε.
  - Interior nodes are variables
  - Leaf nodes are variables, terminals, or ε
  - A leaf can be ε only if it is the only child of its parent.
- A node and its children (from left to right) must form the head and body of a production

The Yield of a Parse Tree

- The yield of a parse tree is the string of leaves from left to right.
- Important are those parse trees where:
  - The yield is a terminal string
  - The root is labeled by the start variable
- We shall see the yields of these important parse trees is the language of the grammar.

Example



Equivalence of Parse Trees and Derivations

- The following about a grammar G = (V, T, P, S) and a terminal string w are all equivalent:
  - $S \Rightarrow^* w$ (i.e., w is in L(G))
  - $S \underset{lm}{\Rightarrow}^* w$
  - $S \underset{rm}{\Rightarrow}^* w$
  - There is a parse tree for G with root S and yield w.

From Trees to Derivations

- Induction on the height of the parse tree.
  - **Basis**: (Height = 1) Tree is root A and leaves w = $a_1, a_2, \ldots, a_k$. Then $A \to w$ must be a production, so $A \underset{lm}{\Rightarrow} w$ and $A \underset{rm}{\Rightarrow} w$.
  - **Induction**: (Height > 1) Tree is root A with children $X_1, X_2, \ldots, X_k$. Those X_i's that are variables are roots of shorter trees.
    - Thus, the IH says that they have LM derivations of their yields. Construct a LM derivation of w from A by starting with $A \underset{lm}{\Rightarrow} X_1X_2\ldots X_k$, then using LM derivations from each X_i that is a variable, in the order from left to right.
    - RM derivation analogous.

From Derivations to Trees

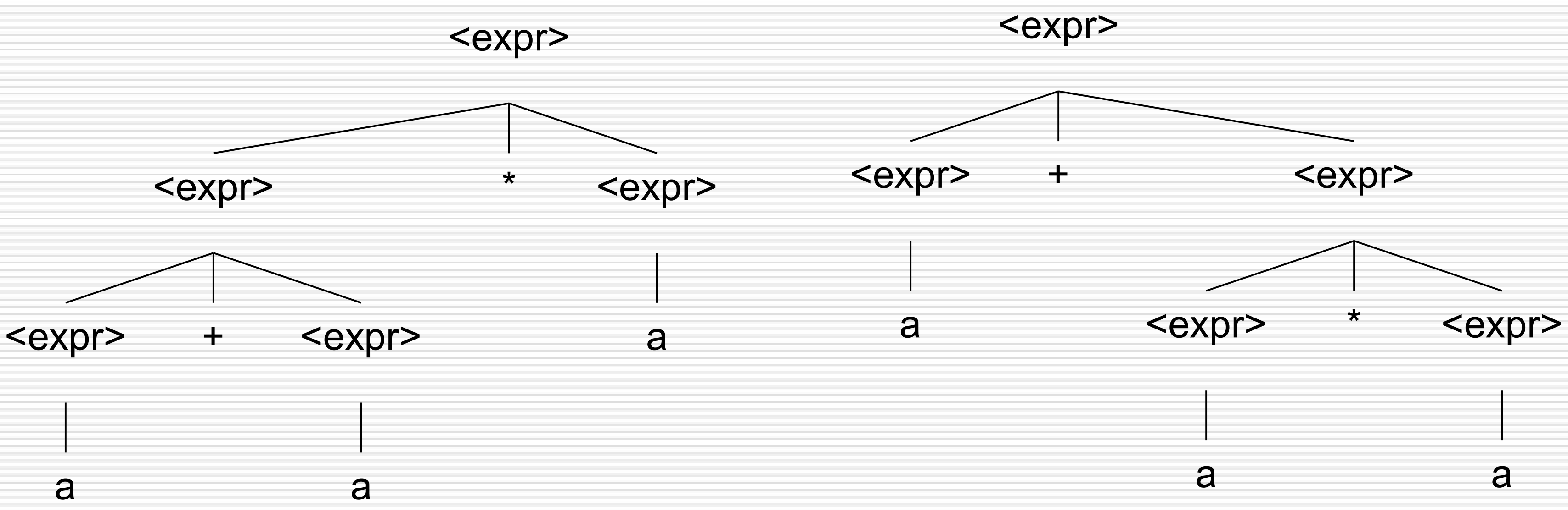
- Induction on length of the derivation.
  - **Basis**: One step. There is an obvious parse tree.
  - **Induction**: More than one step.
    - Let the first step be $A \Rightarrow X_1X_2\ldots X_k$.
    - Subsequent changes can be reordered so that all changes to X_1 and the sentential forms that replace it are done first, then those for X_2, and so on (i.e., we can rewrite the derivation as a LM derivation).
    - The derivations from those X_i's that are variables are all shorter than the given derivation, so the IH applies.
    - By the IH, there are parse trees for each of these derivations.
    - Make the roots of these trees be children of a new root labeled A.

Example

- Consider derivation $S \Rightarrow AS \Rightarrow AAS \Rightarrow AA \Rightarrow A1A \Rightarrow A10A1 \Rightarrow 0110A1 \Rightarrow 0110011$
  - Subderivation from A is: $A \Rightarrow A1 \Rightarrow 011$
  - Subderivation from S is: $S \Rightarrow AS \Rightarrow A \Rightarrow 0A1 \Rightarrow 0011$
  - Each has a parse tree, put them together with new root S.

Ambiguity

- A CFG is ambiguous if there is a terminal string that has multiple leftmost derivations from the start variable.
  - Equivalently: multiple rightmost derivations, or multiple parse trees.



Remove Ambiguity

- Good news: sometimes we can remove ambiguity "by hand"
- Bad news: there is no algorithm to do it
- More bad news: some CFL's have only ambiguous CFG's

- E.g. forces correct precedence in parse tree grouping

<expr> → <expr> + <term> | <term>

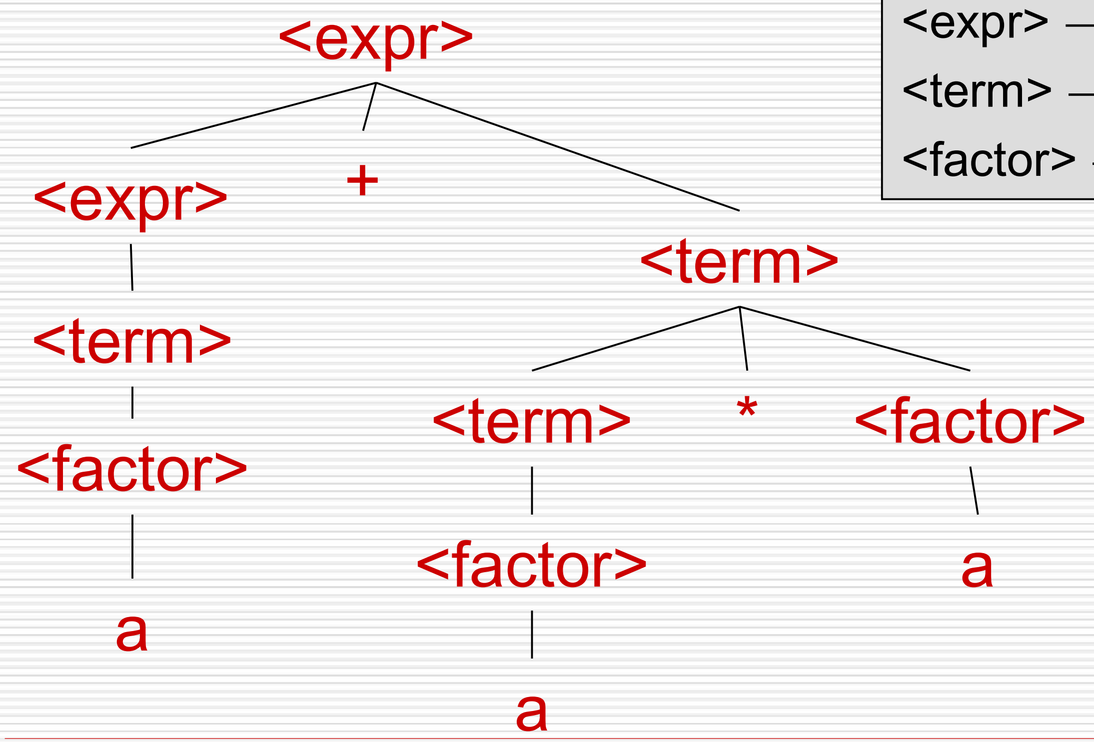
<term> → <term> * <factor> | <factor>

<factor> → (<expr>) | a

Example

- parse tree for a + a * a in new grammar:

<expr> → <expr> + <term> | <term>
<term> → <term> * <factor> | <factor>
<factor> → (<expr>) | a



Inherent Ambiguity

- A CFL L is inherently ambiguous if every CFG for L is ambiguous.
- Ambiguity of the grammar implies that at least some strings in its language have different structures (parse trees).
  - Thus, such a grammar is unlikely to be useful for a programming language, because two structures for the same string (program) implies two different meanings.
  - Common example: the easiest grammars for arithmetic expressions are ambiguous and need to be replaced by more complex, unambiguous grammars.
  - An inherently ambiguous language would be absolutely unsuitable as a programming language, because we would not have any way of fixing a unique structure for all its programs.

Example

- There indeed exist inherent ambiguous languages. E.g.

  $L=\{a^nb^nc^md^m \mid n\geq 1, m\geq 1\} \cup \{a^nb^mc^md^n \mid n\geq 1, m\geq 1\}$

  $S \rightarrow AB \mid C$

  $A \rightarrow aAb \mid ab$

  $B \rightarrow cBd \mid cd$

  $C \rightarrow aCd \mid aDd$

  $D \rightarrow bDc \mid bc$

Chomsky Normal Form

- Useful to deal only with CFGs in a simple normal form
- Most common: Chomsky Normal Form (CNF)
- Definition: every production has form

$$A \rightarrow BC \quad \text{or}$$

$$A \rightarrow a \quad \text{or} \quad S \rightarrow \varepsilon$$

where A, B, C are any non-terminals (and B, C are not S) and a is any terminal.

Chomsky Normal Form

Theorem: Every CFL is generated by a CFG in Chomsky Normal Form.

Proof: Transform any CFG into an equivalent CFG in CNF. Four steps:

- add a new start symbol
- eliminate "ε-productions" $A \rightarrow \varepsilon$
- eliminate "unit productions" $A \rightarrow B$
- convert remaining rules into proper form

Chomsky Normal Form

- add a new start symbol
  - add production $S_0 \rightarrow S$
- remove "ε-productions" $A \rightarrow \varepsilon$
  - for each production with A on rhs, add productions with each occurrence of A removed. E.g. for the rule $R \rightarrow uAvAw$, add $R \rightarrow uvAw \mid uAvw \mid uvw$
- eliminate "unit productions" $A \rightarrow B$
  - for each production with B on lhs: $B \rightarrow u$, add the rule $A \rightarrow u$

Chomsky Normal Form

- convert remaining rules into proper form
  - replace production of form:

$$A \rightarrow u_1 U_2 u_3 \ldots u_k$$

with:

$A \rightarrow U_1 A_1$ $\qquad$ $U_1 \rightarrow u_1$

$A_1 \rightarrow U_2 A_2$

$A_2 \rightarrow U_3 A_3$ $\qquad$ $U_3 \rightarrow u_3$

$\vdots$

$A_{k-2} \rightarrow U_{k-1} U_k$ $\qquad$ $U_{k-1} \rightarrow u_{k-1}$

$U_k \rightarrow u_k$

U_2 is already a non-terminal

Example 2.7

$S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \varepsilon$

Cleaning Up Grammars

- Eliminate useless symbols. In order for a symbol X to be useful, it must:
  - 1. Derive some terminal string (possibly X is a terminal).
  - 2. Be reachable from the start symbol; i.e., $S \Rightarrow^* \alpha X \beta$.
  - Note that X wouldn't really be useful if α or β included a symbol that didn't satisfy (1), so it is important that (1) be tested first, and symbols that don't derive terminal strings be eliminated before testing (2).

Eliminate Useless Symbols（1）

- Finding symbols that don't derive any terminal string
- Recursive construction:
  - Basis: A terminal surely derives a terminal string.
  - Induction: If A is the head of a production whose body is $X_1X_2 \ldots X_k$, and each X_i is known to derive a terminal string, then surely A derives a terminal string.
- Keep going until no more symbols that derive terminal strings are discovered.

Example

$$S \rightarrow AB \mid C$$

$$A \rightarrow 0B \mid C$$

$$B \rightarrow 1 \mid A0$$

$$C \rightarrow AC \mid C1$$

Eliminate Useless Symbols（2）

- Finding symbols that cannot be derived from the start symbol
- Another recursive algorithm:
  - Basis: S is “in”.
  - Induction: If variable A is in, then so is every symbol in the production bodies for A.
- Keep going until no more symbols derivable from S can be found.