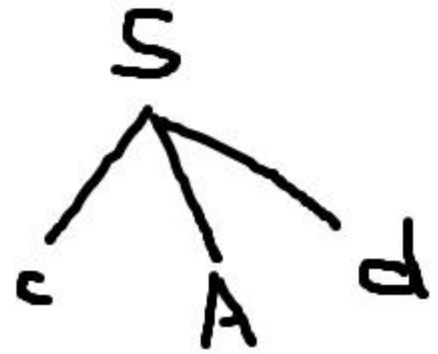
TOP Down parsing

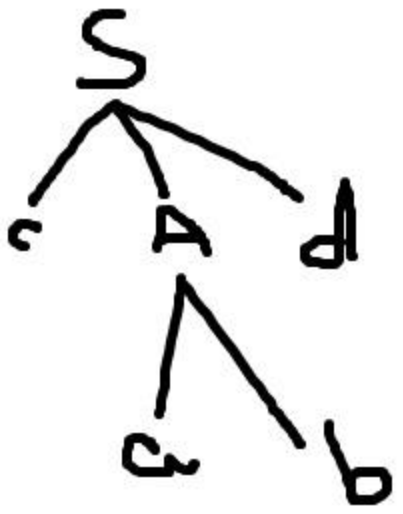
Introduction

- It can be viewed as an attempt to find a leftmost derivation for an input string.
- It can also be viewed as attempting to construct a parse tree for the input starting from the root and creating nodes of the parse tree in preorder

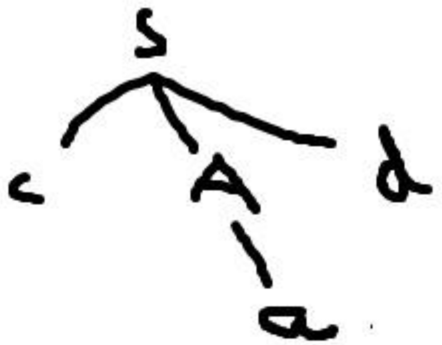
- $S \rightarrow cAd$
- $A \rightarrow ab \mid a$
- W=cad

Constructing a parse tree for w=cad





Backtracking



Difficulties with Top down parsing

1. First problem concerns Left recursion
2. The second concerns backtracking
3. The third is that the order in which alternates are tried can affect the language accepted
   - E.g., we used a and then ab as the order of the alternates for A
   - We could fail to accept **cabd**

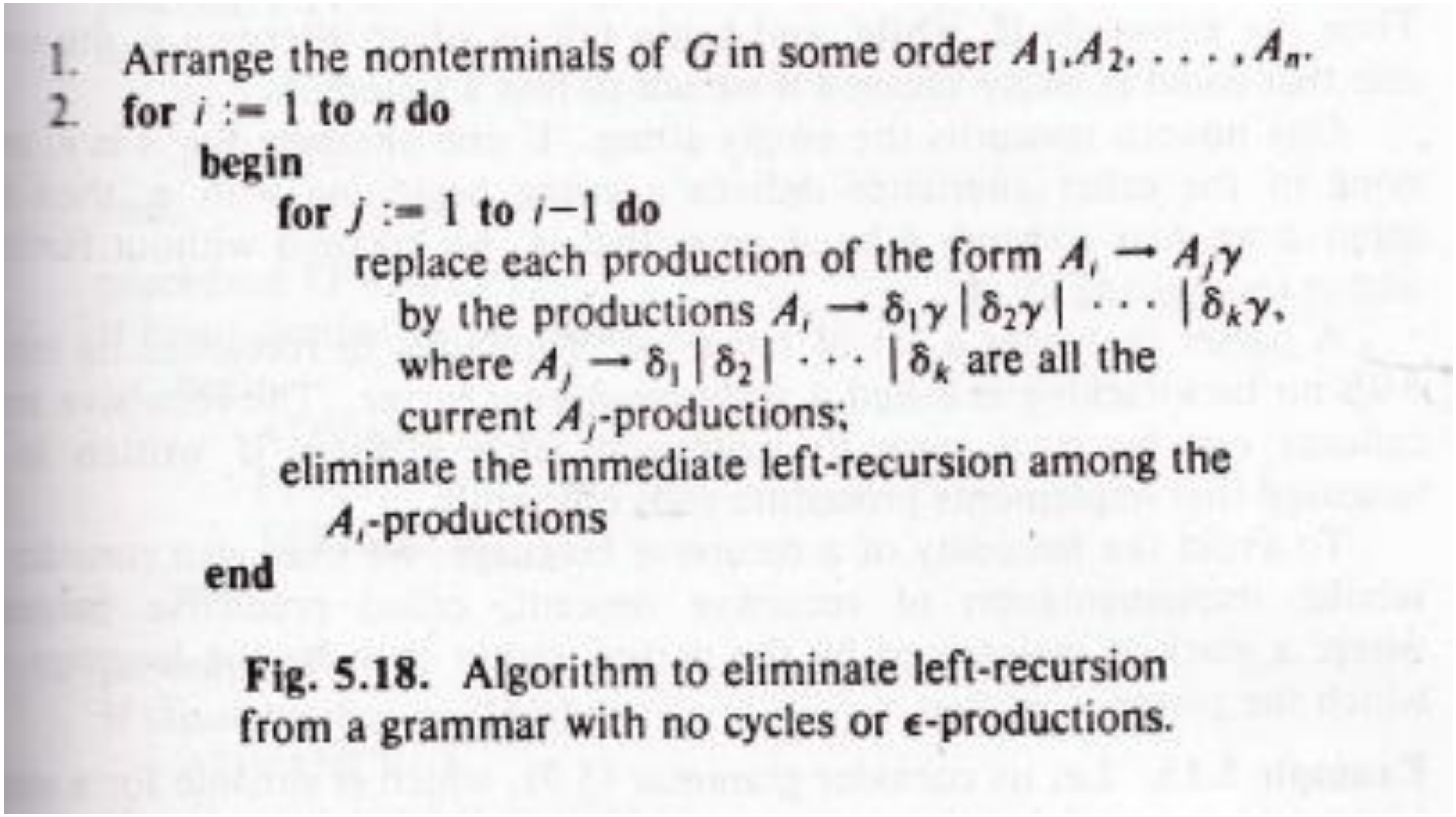
Elimination of left recursion

- $A \rightarrow A\alpha \mid \beta$ where β does not begin with an A
- Can be eliminated by
- $A \rightarrow \beta A'$
- $A' \rightarrow \alpha A' \mid \in$

Consider the grammar

- $E \rightarrow E+T \mid T$
- $T \rightarrow T * F \mid F$
- $F \rightarrow (E) \mid id$

Algorithm



```
1. Arrange the nonterminals of $G$ in some order $A_1, A_2, \ldots, A_n$.
2. for $i$ := 1 to $n$ do
       begin
           for $j$ := 1 to $i-1$ do
               replace each production of the form $A_i \rightarrow A_j\gamma$
                   by the productions $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_k\gamma$,
                   where $A_j \rightarrow \delta_1 \mid \delta_2 \mid \cdots \mid \delta_k$ are all the
                   current $A_j$-productions;
           eliminate the immediate left-recursion among the
               $A_i$-productions
       end
```

Fig. 5.18. Algorithm to eliminate left-recursion from a grammar with no cycles or ϵ-productions.