Parsing

Left Recursion and Left Factoring

Why use Context Free Grammars

Every construct that can be described by a regular expression can be described by a grammar, but not vice-versa. Alternatively, every regular language is a context-free language, but not vice-versa.

- L={ $a^n b^n$ | $n \geq 1$}

- Show that L can be described by a grammar not by a regular expression
- Construct a DFA D with k states to accept L
- For $a^n b^n$ ($n>k$) some state (s_i) of D must be entered twice

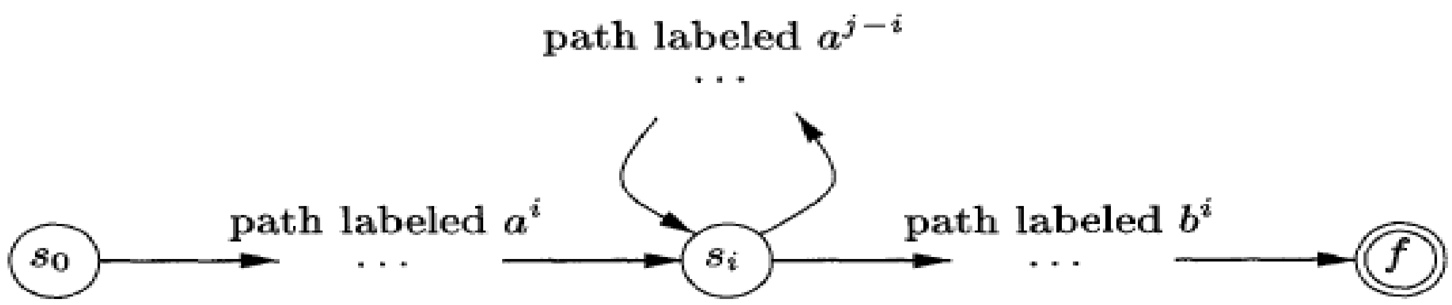


Figure 4.6: DFA D accepting both $a^i b^i$ and $a^j b^i$.

Left Recursion – Why is it bad

- A grammar is left recursive if it can be represented in the form:

$$A \Rightarrow^{+} A\alpha$$

- Why is this a problem for top down parsing?

Left Recursion – Why is it bad

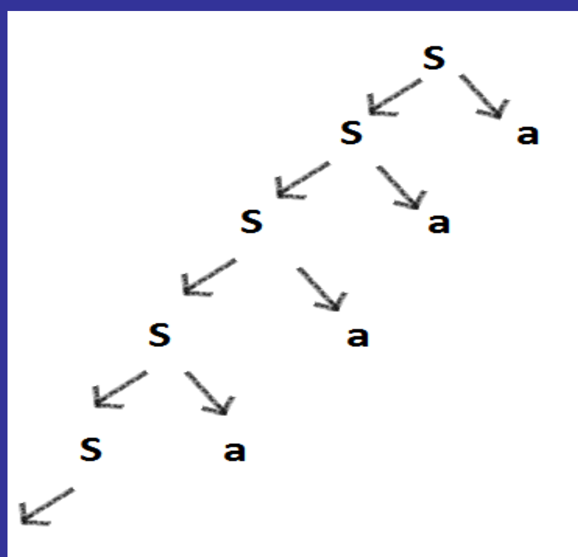
BRAC UNIVERSITY
Inspiring Excellence

- Example:

Left recursive grammar

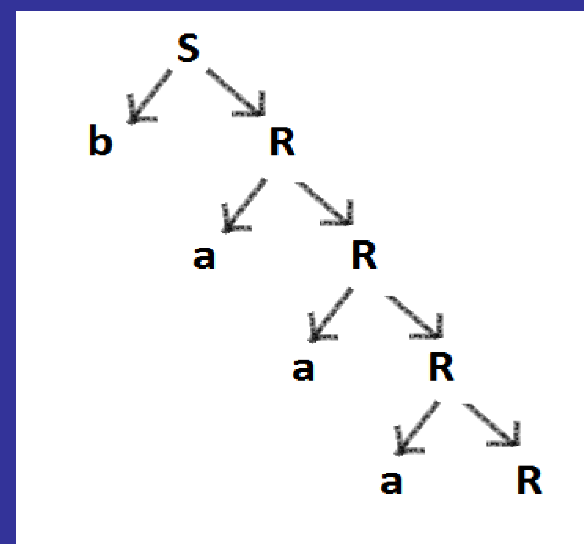
S -> Sa | b

To derive baa:



Equivalent Right Recursive grammar

S -> bR
R -> aR | €



Left and Right Recursive Grammar

(a) $A \rightarrow A\alpha \mid \beta$

(b)
$$\begin{aligned} A &\rightarrow \beta R \\ R &\rightarrow \alpha R \mid \epsilon \end{aligned}$$

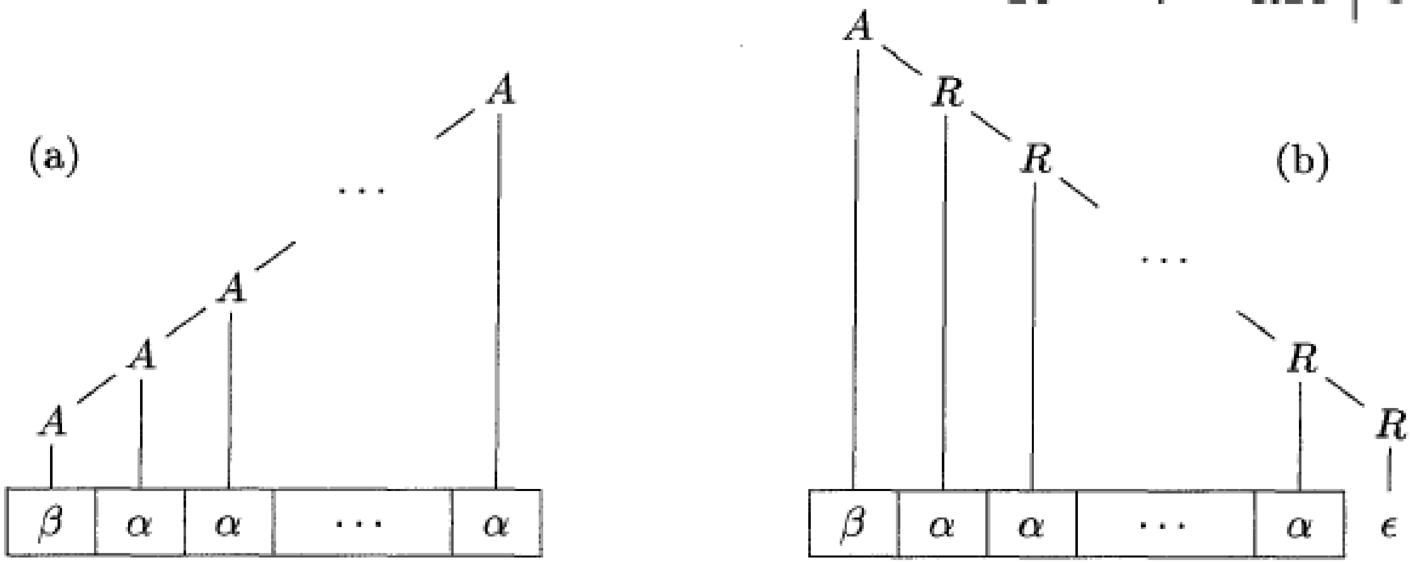


Figure 2.20: Left- and right-recursive ways of generating a string

Left Recursion Removal

Whenever

$$A \Rightarrow^{+} A\alpha$$

Simplest Case: Immediate Left Recursion

Given:

$$A \rightarrow A\alpha \mid \beta$$

Transform into:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \varepsilon$$

where A' is a new nonterminal

More General (but still immediate):

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \mid \ldots \mid \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots$$

Transform into:

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \beta_3 A' \mid \ldots$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \ldots \mid \varepsilon$$

Immediate Left Recursion Elimination: example

- Grammar

  $E \rightarrow E + T \mid T$

  $T \rightarrow T * F \mid F$

  $F \rightarrow (E) \mid$ **id**

Given:

$$A \rightarrow A\alpha \mid \beta$$

Transform into:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \varepsilon$$

- **Left recursion Eliminated (step 1)**

  $E \rightarrow T\,E'$

  $E' \rightarrow + T\,E' \mid \varepsilon$

Immediate Left Recursion Elimination: example

- Grammar

  $E \rightarrow E + T \mid T$

  $T \rightarrow T * F \mid F$

  $F \rightarrow (E) \mid \textbf{id}$

Given:

$$A \rightarrow A\alpha \mid \beta$$

Transform into:

$$A \rightarrow \beta A'$$
$$A' \rightarrow \alpha A' \mid \varepsilon$$

- **Left recursion Eliminated (step 2)**

  $E \rightarrow T E'$

  $E' \rightarrow + T E' \mid \varepsilon$

  $T \rightarrow F T'$

  $T' \rightarrow * F T' \mid \varepsilon$

Immediate Left Recursion Elimination: example

- Grammar

  E → E + T | T

  T → T * F | F

  F → (E) | **id**

Given:

$$A \rightarrow A\alpha \mid \beta$$

Transform into:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \varepsilon$$

- **Left recursion Eliminated (step 3)**

  E → T E'

  E' → + T E' | ε

  T → F T'

  T' → * F T' | ε

  F → (E) | **id**

Left Recursion in More Than One Step

Example:

$S \rightarrow Af \mid b$

$A \rightarrow Ac \mid Sd \mid e$

Is A left recursive? Yes.

Is S left recursive? Yes, but not immediate left recursion. $S \Rightarrow Af \Rightarrow Sdf$

Approach:

Look at the rules for S only (ignoring other rules)... No left recursion.

Look at the rules for A...

Do any of A's rules start with S? Yes.

$A \rightarrow Sd$

Get rid of the S. Substitute in the righthand sides of S.

$A \rightarrow Afd \mid bd$

The modified grammar:

$S \rightarrow Af \mid b$

$A \rightarrow Ac \mid Afd \mid bd \mid e$

Now eliminate immediate left recursion involving A.

$S \rightarrow Af \mid b$

$A \rightarrow bdA' \mid eA'$

$A' \rightarrow cA' \mid fdA' \mid \varepsilon$

Left Recursion in More Than One Step

The Original Grammar:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow A\underline{c} \mid S\underline{d} \mid B\underline{e}$

$B \rightarrow A\underline{g} \mid S\underline{h} \mid \underline{k}$

So Far:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow \underline{bd}A' \mid B\underline{e}A'$

$A' \rightarrow \underline{c}A' \mid \underline{fd}A' \mid \varepsilon$

Left Recursion in More Than One Step

The Original Grammar:

$S \rightarrow Af \mid b$

$A \rightarrow Ac \mid Sd \mid Be$

$B \rightarrow Ag \mid Sh \mid k$

So Far:

$S \rightarrow Af \mid b$

$A \rightarrow bdA' \mid BeA'$

$A' \rightarrow cA' \mid fdA' \mid \varepsilon$

$B \rightarrow Ag \mid Sh \mid k$

Look at the B rules next; Does any righthand side start with "S"?

Left Recursion in More Than One Step

The Original Grammar:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow A\underline{c} \mid S\underline{d} \mid B\underline{e}$

$B \rightarrow A\underline{g} \mid S\underline{h} \mid \underline{k}$

So Far:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow \underline{bd}A' \mid B\underline{e}A'$

$A' \rightarrow \underline{c}A' \mid \underline{fd}A' \mid \varepsilon$

$B \rightarrow A\underline{g} \mid A\underline{fh} \mid \underline{bh} \mid \underline{k}$

Substitute, using the rules for "S"

$A\underline{f}\ldots \mid \underline{b}\ldots$

Left Recursion in More Than One Step

The Original Grammar:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow A\underline{c} \mid S\underline{d} \mid B\underline{e}$

$B \rightarrow A\underline{g} \mid S\underline{h} \mid \underline{k}$

So Far:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow \underline{bd}A' \mid B\underline{e}A'$

$A' \rightarrow \underline{c}A' \mid \underline{fd}A' \mid \varepsilon$

$B \rightarrow A\underline{g} \mid A\underline{fh} \mid \underline{bh} \mid \underline{k}$

Does any righthand side start with "A"?

Left Recursion in More Than One Step

The Original Grammar:

S → Af | b
A → Ac | Sd | Be
B → Ag | Sh | k

So Far:

S → Af | b
A → bdA' | BeA'
A' → cA' | fdA' | ε
B → bdA'g | BeA'g | Afh | bh | k

Substitute, using the rules for "A"
bdA'... | BeA'...

Left Recursion in More Than One Step

The Original Grammar:

$S \rightarrow Af \mid b$

$A \rightarrow Ac \mid Sd \mid Be$

$B \rightarrow Ag \mid Sh \mid k$

So Far:

$S \rightarrow Af \mid b$

$A \rightarrow bdA' \mid BeA'$

$A' \rightarrow cA' \mid fdA' \mid \varepsilon$

$B \rightarrow bdA'g \mid BeA'g \mid bdA'fh \mid BeA'fh \mid bh \mid k$

Substitute, using the rules for "A"

$bdA' \ldots \mid BeA' \ldots$

Left Recursion in More Than One Step

The Original Grammar:

S → Af | b
A → Ac | Sd | Be
B → Ag | Sh | k

So Far:

S → Af | b
A → bdA' | BeA'
A' → cA' | fdA' | ε
B → bdA'g | BeA'g | bdA'fh | BeA'fh | bh | k

Finally, eliminate any immediate Left recursion involving "B"

Next Form

S → Af | b
A → bdA' | BeA'
A' → cA' | fdA' | ε
B → bdA'gB' | bdA'fhB' | bhB' | kB'
B' → eA'gB' | eA'fhB' | ε

Left Recursion in More Than One Step

The Original Grammar:

S → A**f** | **b**

A → A**c** | S**d** | B**e** | C

B → A**g** | S**h** | **k**

C → B**km**A | AS | **j**

If there is another nonterminal, then do it next.

So Far:

S → A**f** | **b**

A → **bd**A' | B**e**A' | CA'

A' → **c**A' | **fd**A' | ε

B → **bd**A'**g**B' | **bd**A'**fh**B' | **bh**B' | **k**B' | CA'**g**B' | CA'**fh**B'

B' → **e**A'**g**B' | **e**A'**fh**B' | ε

Algorithm for Eliminating Left Recursion

BRAC UNIVERSITY
Inspiring Excellence

Assume the nonterminals are ordered $A_1, A_2, A_3, \ldots$
(In the example: S, A, B)

for each nonterminal A_i (for i = 1 to N) do
  for each nonterminal A_j (for j = 1 to i-1) do
    Let $A_j \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots \mid \beta_N$ be all the rules for A_j
    if there is a rule of the form
      $A_i \rightarrow A_j\alpha$
    then replace it by
      $A_i \rightarrow \beta_1\alpha \mid \beta_2\alpha \mid \beta_3\alpha \mid \ldots \mid \beta_N\alpha$
    endIf
  endFor
  Eliminate immediate left recursion
    among the A_i rules
endFor

A_1
A_2
A_3
...
A_j
...
A_7
A_i

Inner Loop

Outer Loop

Left Factoring

- Consider the following grammar:

$$S \rightarrow i\,E\,t\,S \mid i\,E\,t\,S\,e\,S \mid a$$
$$E \rightarrow b$$

Suppose a top down parser is trying to generate the string: ***ibtaea***

How will a top down parser work?

Remember that a parser has limited look ahead symbols (1 look ahead usually)

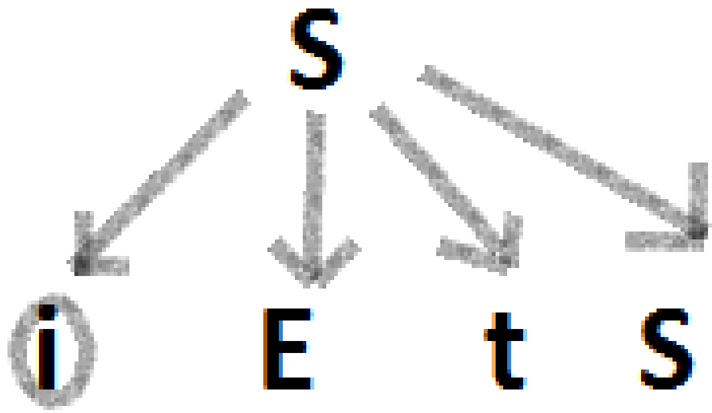
Left Factoring

S

$$S \rightarrow i\,E\,t\,S \mid i\,E\,t\,S\,e\,S \mid a$$
$$E \rightarrow b$$

Given string: ***ibtaea***

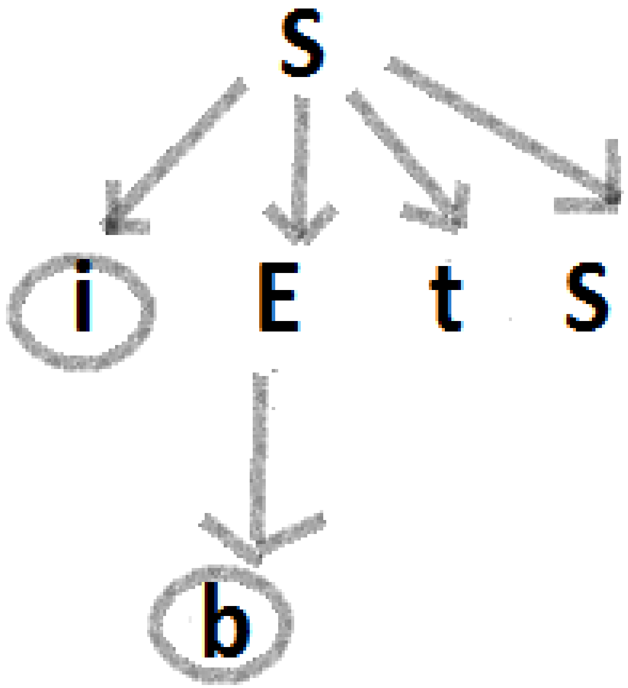
Left Factoring



$$S \rightarrow iEtS \mid iEtSeS \mid a$$
$$E \rightarrow b$$

Given string: ***ibtaea***

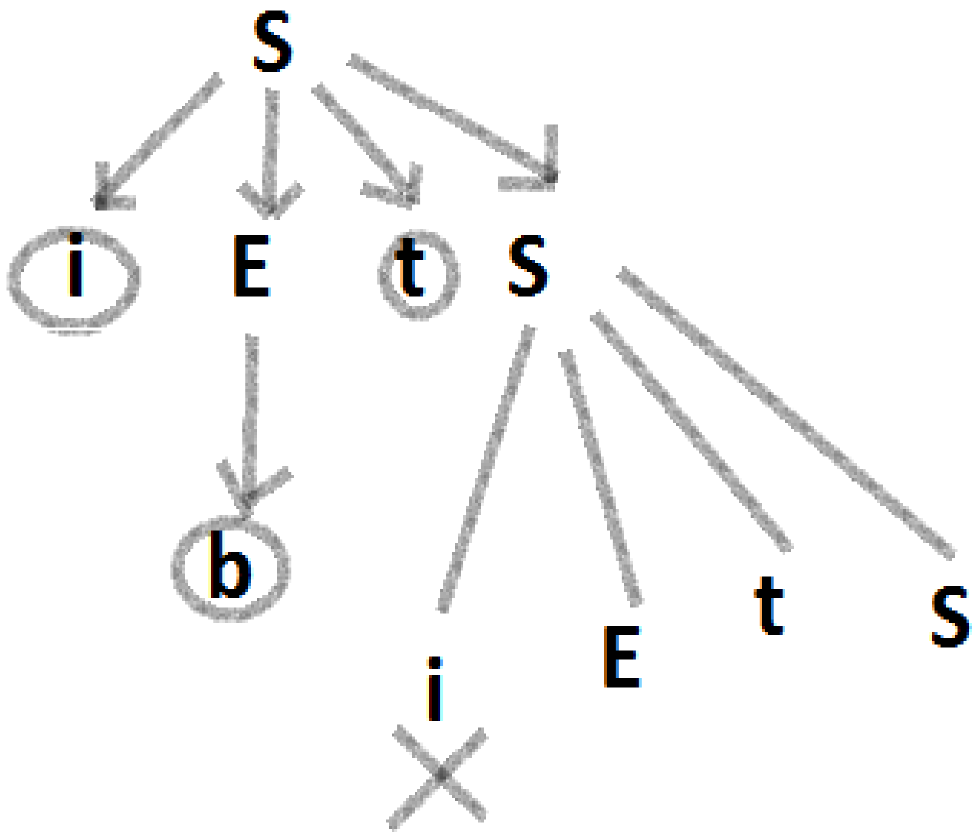
Left Factoring



$$S \rightarrow i\,E\,t\,S \mid i\,E\,t\,S\,e\,S \mid a$$
$$E \rightarrow b$$

Given string: ***ibtaea***

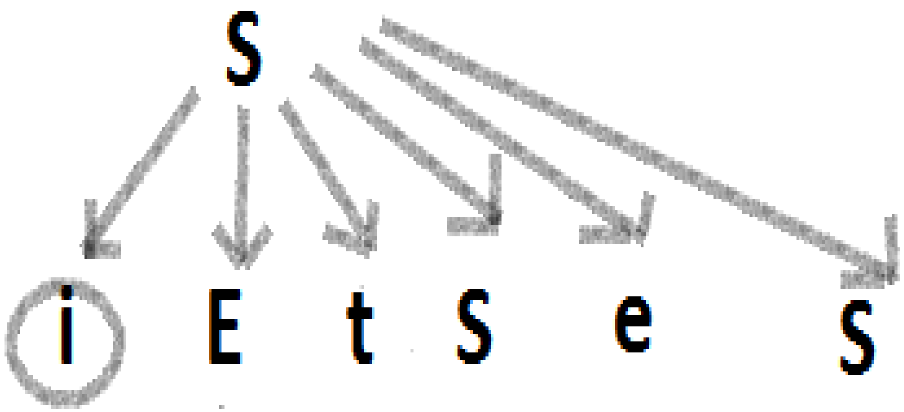
Left Factoring



$$S \rightarrow iEtS \mid iEtSeS \mid a$$
$$E \rightarrow b$$

Given string: ***ibtaea***

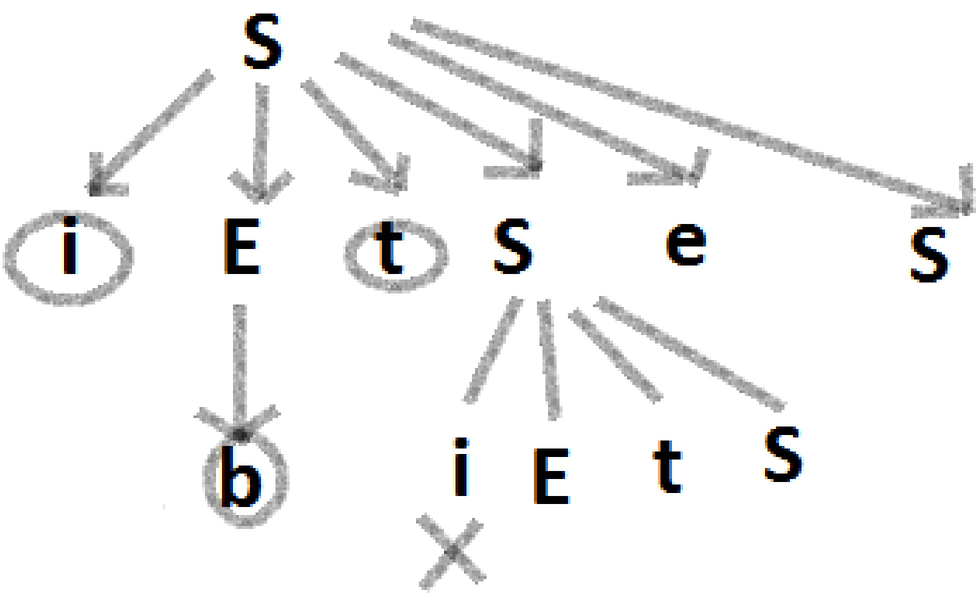
Left Factoring



$$S \rightarrow iEtS \mid iEtSeS \mid a$$
$$E \rightarrow b$$

Given string: ***ibtaea***

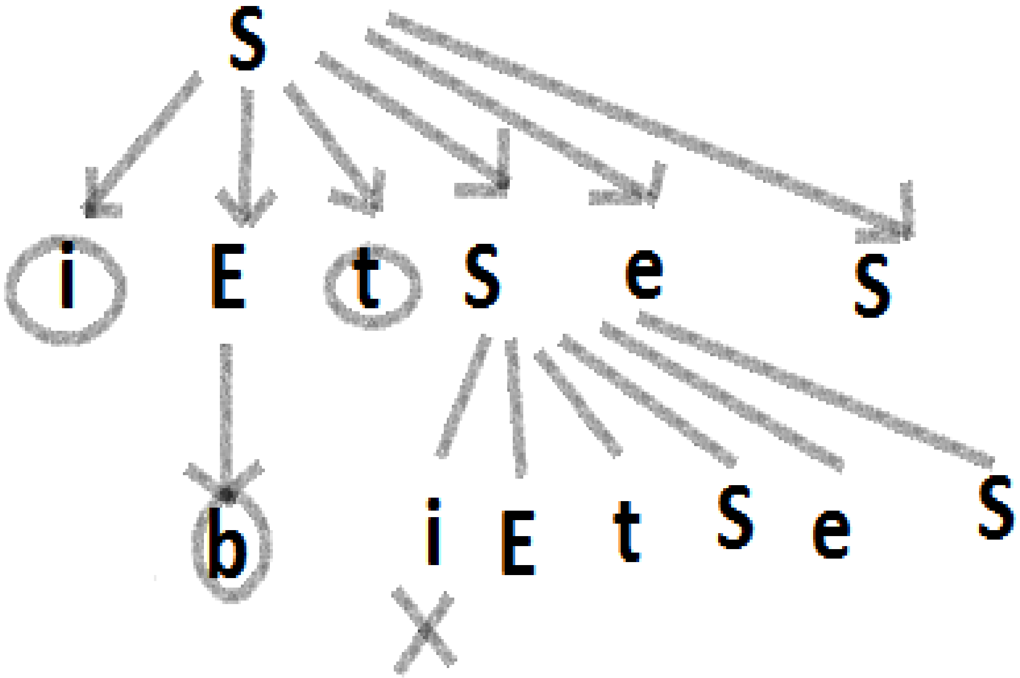
Left Factoring



$$S \rightarrow iEtS \mid iEtSeS \mid a$$
$$E \rightarrow b$$

Given string: ***ibtaea***

Left Factoring

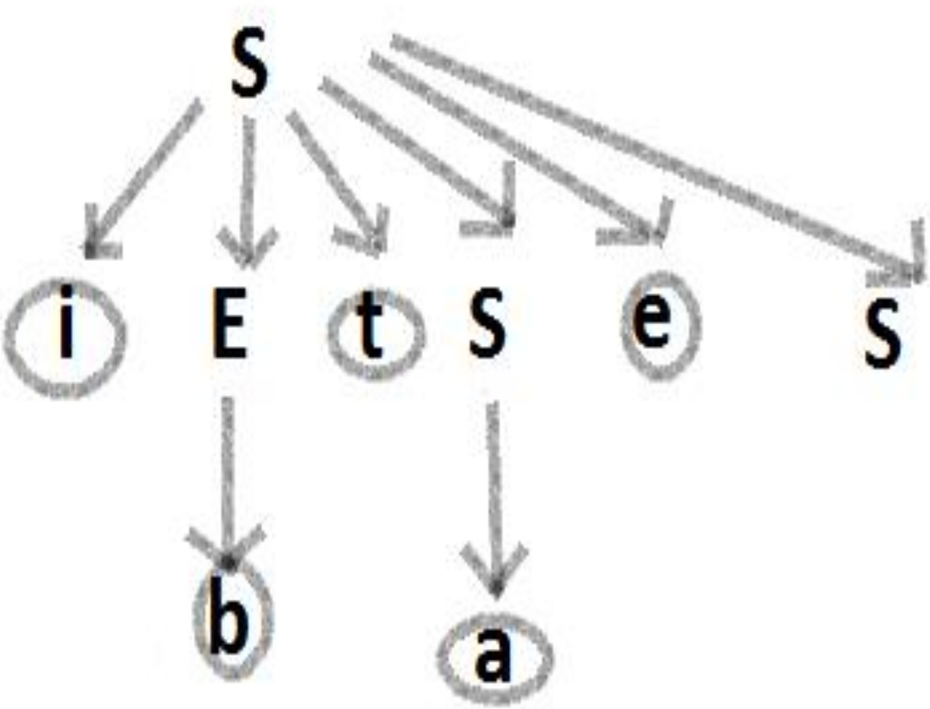


$$S \rightarrow iEtS \mid iEtSeS \mid a$$

$$E \rightarrow b$$

Given string: ***ibtaea***

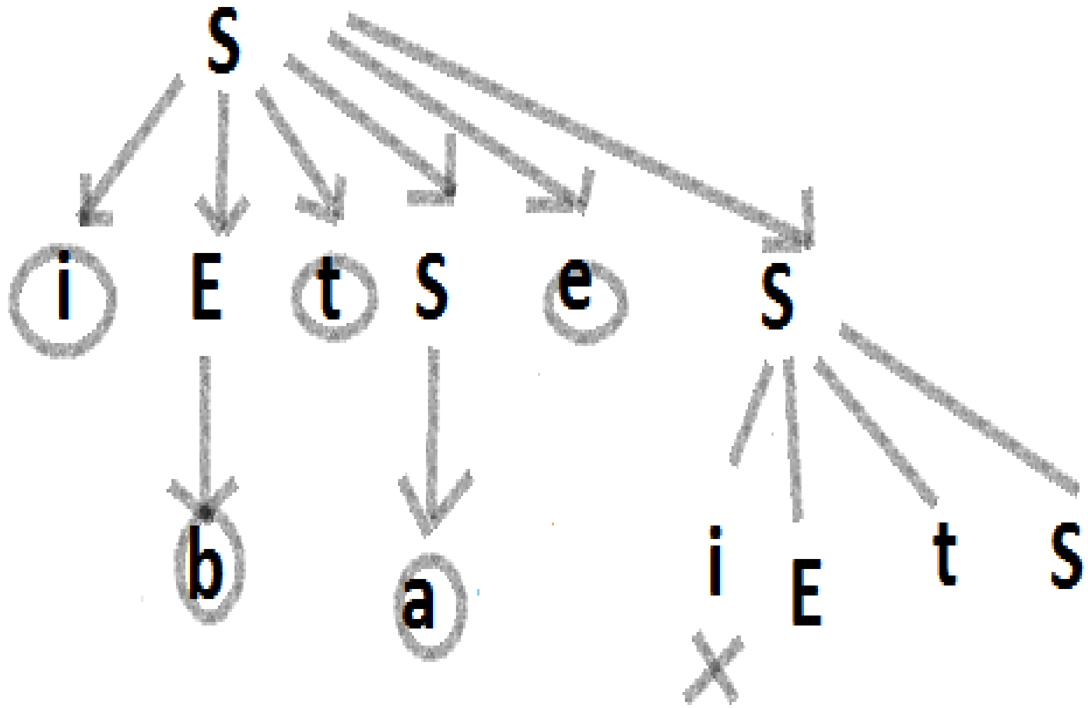
Left Factoring



$$S \rightarrow iEtS \mid iEtSeS \mid a$$
$$E \rightarrow b$$

Given string: ***ibtaea***

Left Factoring



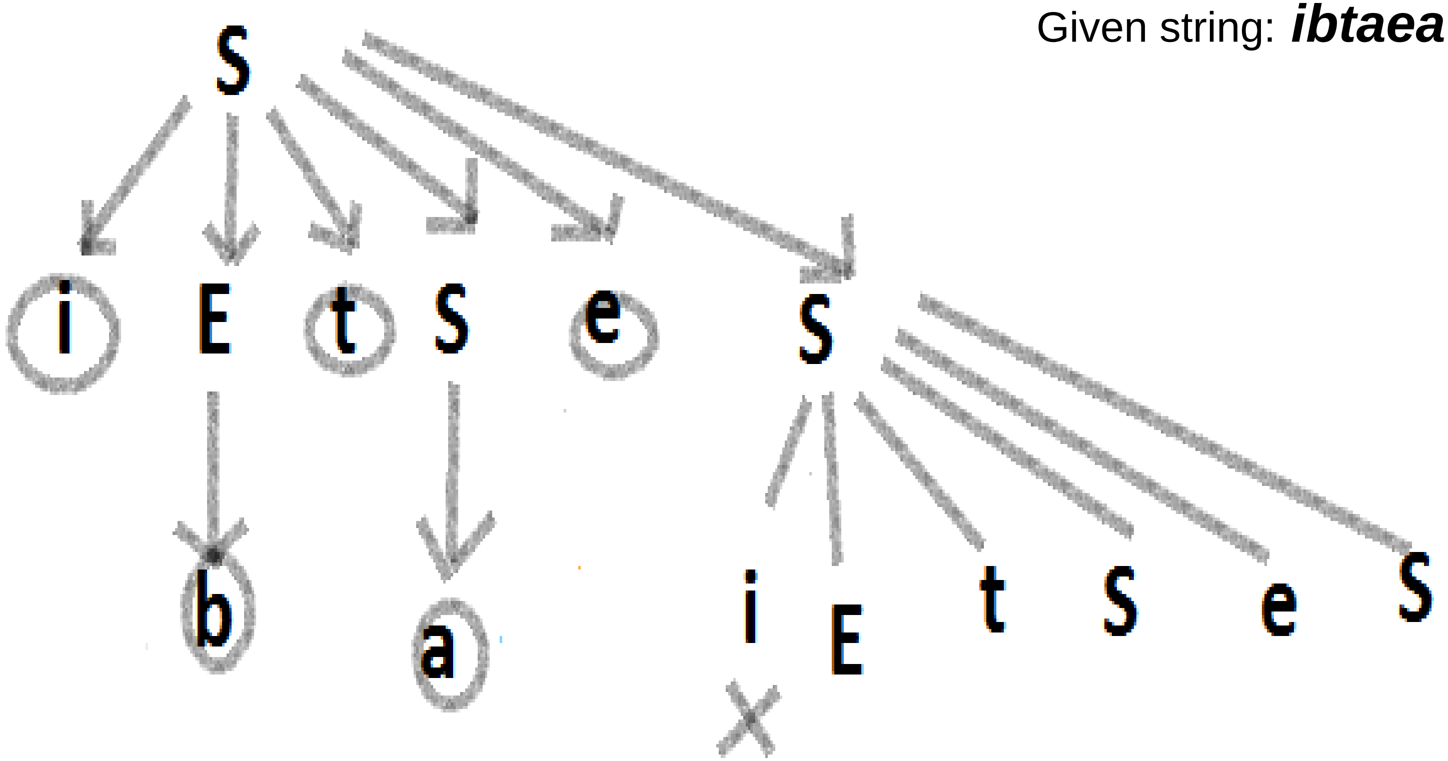
$$S \rightarrow iEtS \mid iEtSeS \mid a$$
$$E \rightarrow b$$

Given string: ***ibtaea***

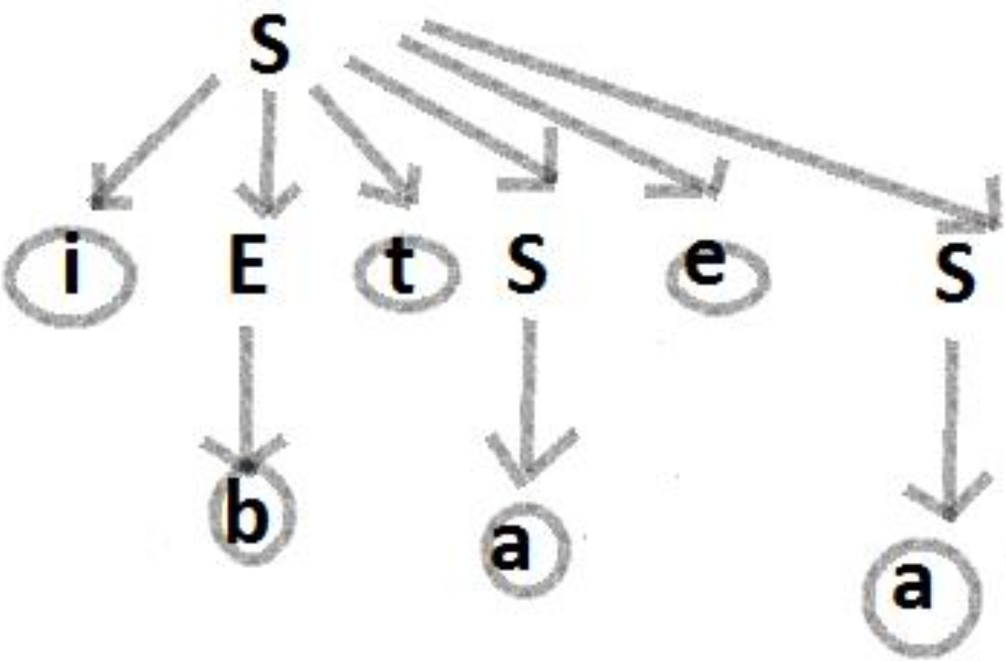
Left Factoring

$$S \to iEtS \mid iEtSeS \mid a$$
$$E \to b$$

Given string: ***ibtaea***



Left Factoring



$$S \to iEtS \mid iEtSeS \mid a$$
$$E \to b$$

Given string: ***ibtaea***

Left Factoring

- So to eliminate the problem of generating the same grammar string multiple times, take a common grammar string and factor it out:

Given Grammar:

$$S \rightarrow i\,E\,t\,S \mid i\,E\,t\,S\,e\,S \mid a$$
$$E \rightarrow b$$

Left Factored Equivalent of the grammar:

$$S \rightarrow i\ E\ t\ S\ S' \mid a$$
$$S' \rightarrow e\ S \mid \epsilon$$
$$E \rightarrow b$$

Left Factoring

In general, if $A \to \alpha\beta_1 \mid \alpha\beta_2$ are two A-productions, and the input begins with a nonempty string derived from α, we do not know whether to expand A to $\alpha\beta_1$ or $\alpha\beta_2$. However, we may defer the decision by expanding A to $\alpha A'$. Then, after seeing the input derived from α, we expand A' to β_1 or to β_2. That is, left-factored, the original productions become

$$
\begin{aligned}
A &\to \alpha A' \\
A' &\to \beta_1 \mid \beta_2
\end{aligned}
$$