

Lecture 8

3.1.3 String parsing

88

String parsing

String parsing

- String inference

String parsing

- String inference
 - $w \in L(G)$?

String parsing

- String inference
 - $w \in L(G)$?
- String parsing

String parsing

- **String inference**
 - $w \in L(G)$?
- **String parsing**
 - **String inference + parse tree**

String parsing

- String inference
 - $w \in L(G)$?
- String parsing
 - String inference + parse tree
- Top – down string parsing

String parsing

- **String inference**
 - $w \in L(G)$?
- **String parsing**
 - **String inference + parse tree**
- **Top – down string parsing**
 - **Building parse tree**
 - Start with the *root* (start symbol of the grammar)
 - Towards *leaves* (terminal symbols of the grammar)

String parsing

- **String inference**
 - $w \in L(G)$?
- **String parsing**
 - **String inference + parse tree**
- **Top – down string parsing**
 - **Building parse tree**
 - Start with the *root* (start symbol of the grammar)
 - Towards *leaves* (terminal symbols of the grammar)
- **Bottom – up string parsing**

String parsing

- **String inference**
 - $w \in L(G)$?
- **String parsing**
 - **String inference + parse tree**
- **Top – down string parsing**
 - **Building parse tree**
 - Start with the *root* (start symbol of the grammar)
 - Towards *leaves* (terminal symbols of the grammar)
- **Bottom – up string parsing**
 - **Building parse tree**
 - Starts with *leaves* (terminal symbols of the grammar)
 - Towards *root* (start symbol of the grammar)

Top – down string parsing

CHEERFUL

FANS ARE WATCHING MOTIVATED PLAYERS .

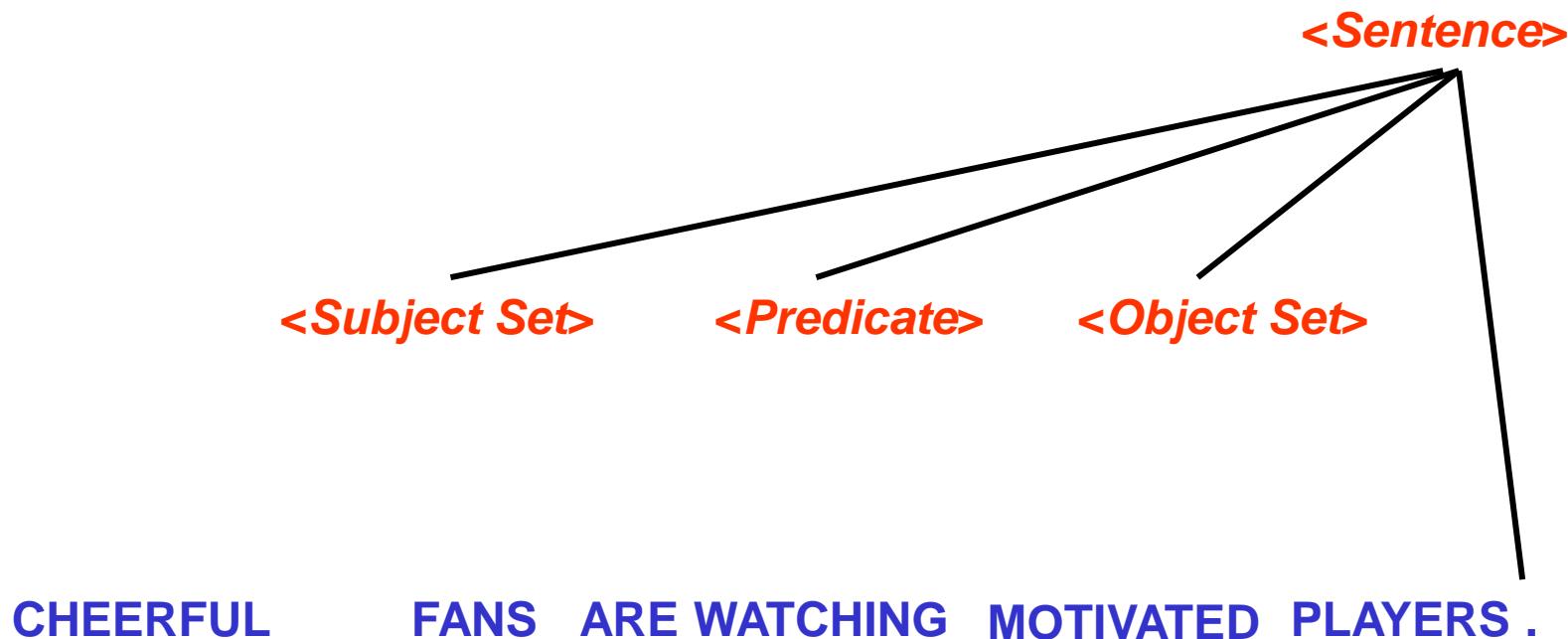
Top – down string parsing

<Sentence>

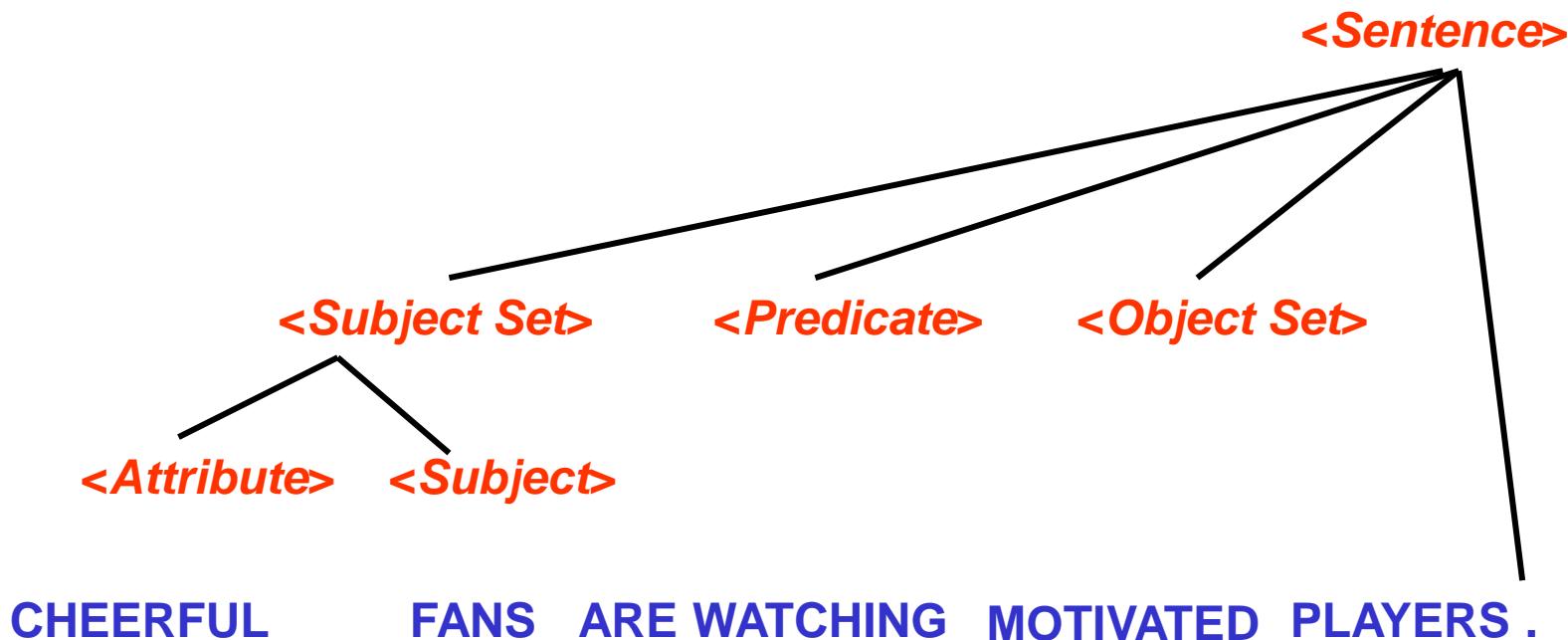
CHEERFUL

FANS ARE WATCHING MOTIVATED PLAYERS .

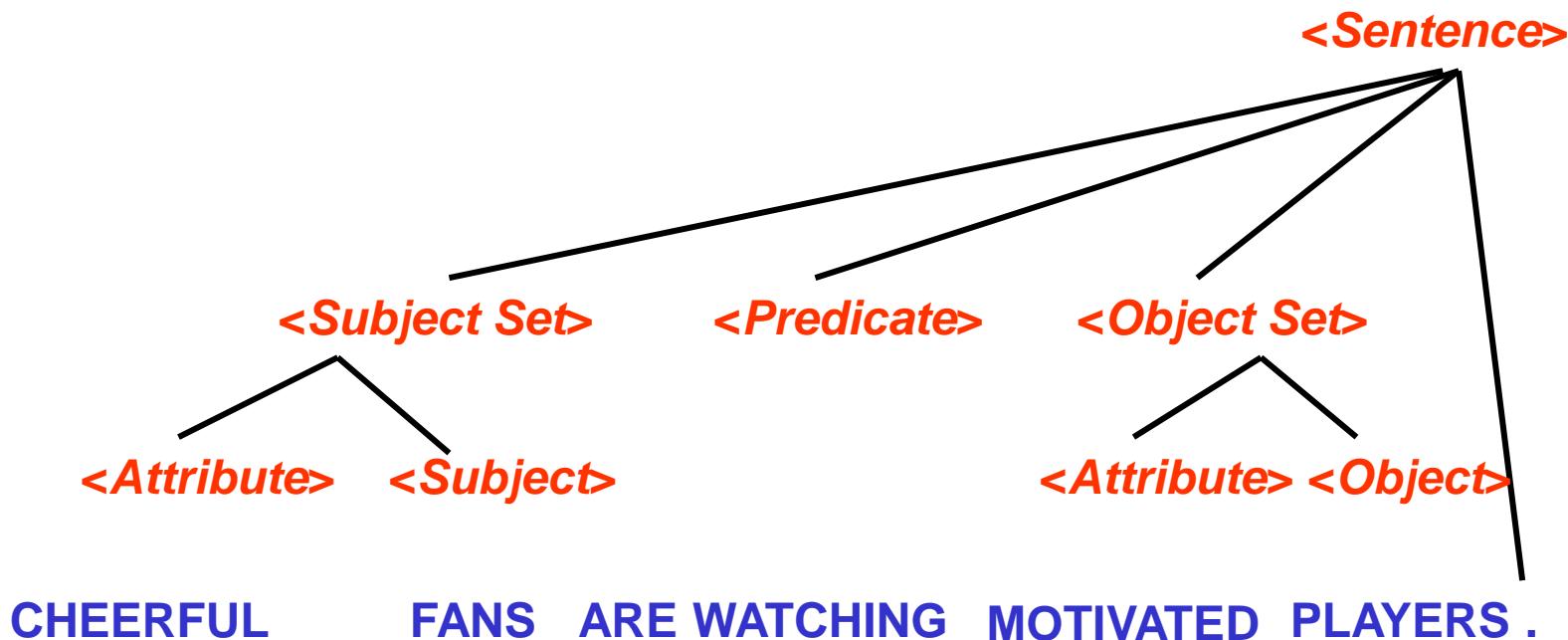
Top – down string parsing



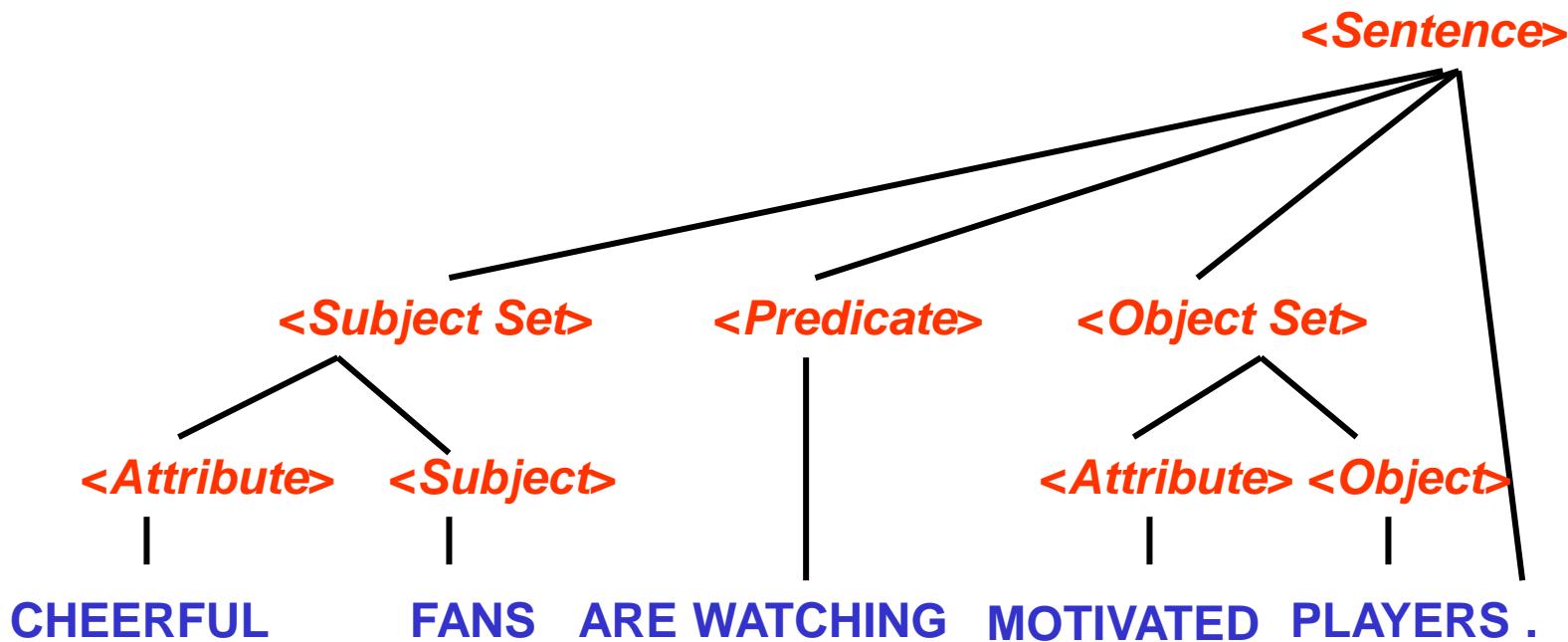
Top – down string parsing



Top – down string parsing



Top – down string parsing



Bottom – up string parsing

CHEERFUL FANS ARE WATCHING MOTIVATED PLAYERS .

Bottom – up string parsing

<Attribute>

|

CHEERFUL

FANS ARE WATCHING MOTIVATED PLAYERS .

Bottom – up string parsing



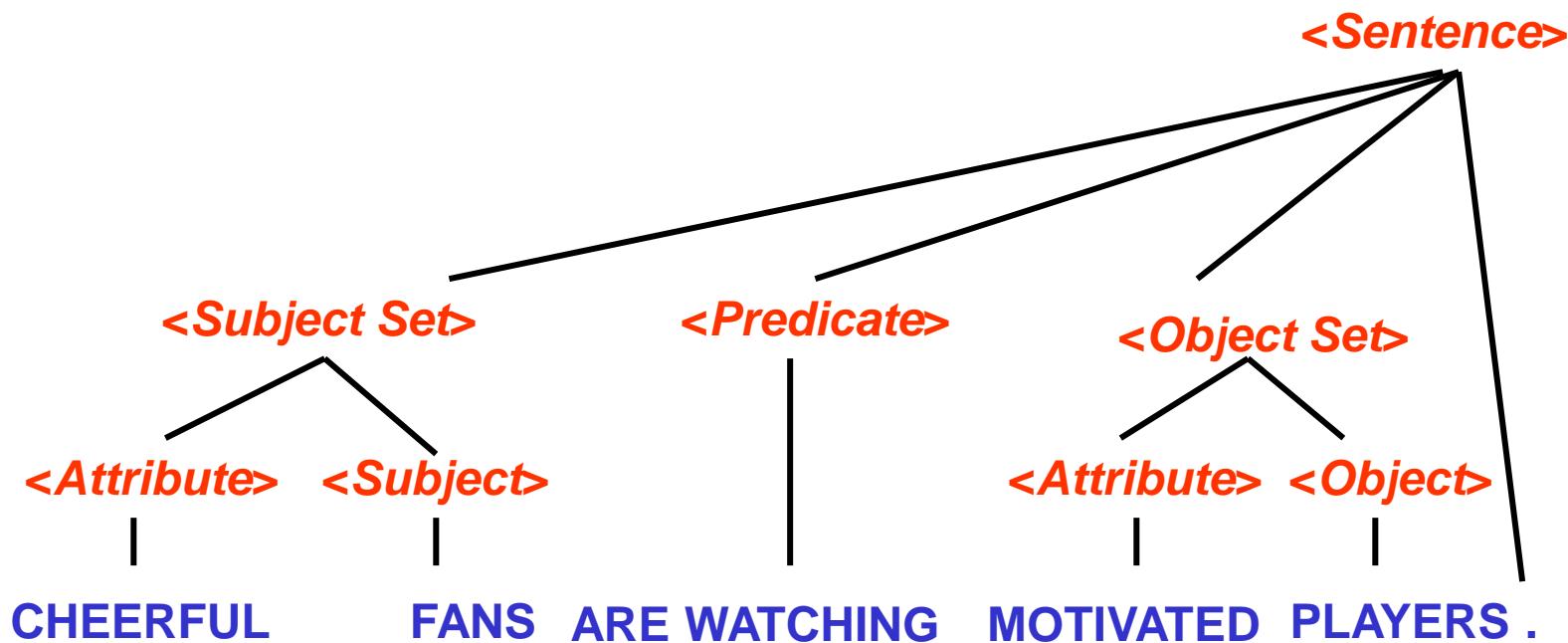
Bottom – up string parsing



Bottom – up string parsing



Bottom – up string parsing



Top – down string parsing

Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$

$S_1 \rightarrow S S_2$

$S_2 \rightarrow ; S_1 \mid \epsilon$

$S \rightarrow \text{var} \ := \ \text{var}$

$S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$

$S \rightarrow \text{begin } S_1 \text{ end}$

Top – down string parsing

```
C → begin S1 end
S1 → S S2
S2 → ; S1 | ε
S → var := var
S → while var ≠ var do S
S → begin S1 end
```

```
begin
    var := var ;
    while var ≠ var do
        var := var
end
```

Top – down string parsing

C

```
C → begin S1 end
S1 → S S2
S2 → ; S1 | ε
S → var := var
S → while var ≠ var do S
S → begin S1 end
```

```
begin
    var := var ;
    while var ≠ var do
        var := var
end
```

Top – down string parsing

C

```
C → begin S1 end
S1 → S S2
S2 → ; S1 | ε
S → var := var
S → while var ≠ var do S
S → begin S1 end
```

begin

```
    var := var ;
    while var ≠ var do
        var := var
```

end

Top – down string parsing

C

$C \rightarrow \text{begin } S_1 \text{ end}$

$S_1 \rightarrow S S_2$

$S_2 \rightarrow ; \mid \epsilon$

$S \rightarrow \text{var} \ := \ \text{var}$

$S \rightarrow \text{while } \text{var} \neq \text{var} \ \text{do } S$

$S \rightarrow \text{begin } S_1 \text{ end}$

begin

 var := var ;

 while var \neq var do

 var := var

end

Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$

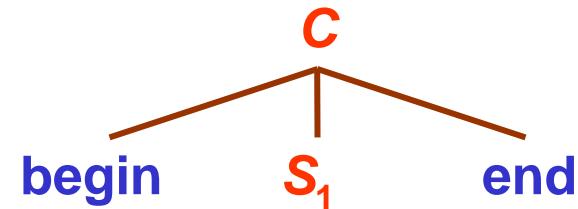
$S_1 \rightarrow S S_2$

$S_2 \rightarrow ; \ S_1 \mid \epsilon$

$S \rightarrow \text{var} \ := \ \text{var}$

$S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$

$S \rightarrow \text{begin } S_1 \text{ end}$



begin

var := var ;

while var ≠ var do

var := var

end

Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$

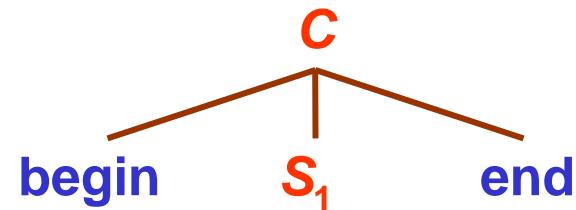
$S_1 \rightarrow S S_2$

$S_2 \rightarrow ; \ S_1 \mid \epsilon$

$S \rightarrow \text{var} \ := \ \text{var}$

$S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$

$S \rightarrow \text{begin } S_1 \text{ end}$



begin

var := var ;

while var ≠ var do

var := var

end

Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$

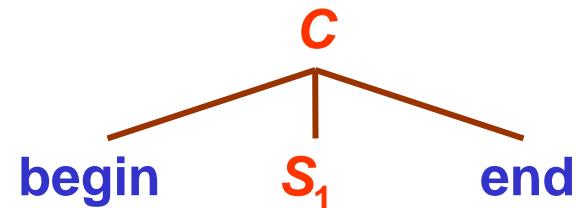
$S_1 \rightarrow S S_2$

$S_2 \rightarrow ; S_1 \mid \epsilon$

$S \rightarrow \text{var} \ := \ \text{var}$

$S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$

$S \rightarrow \text{begin } S_1 \text{ end}$



begin

var := var ;

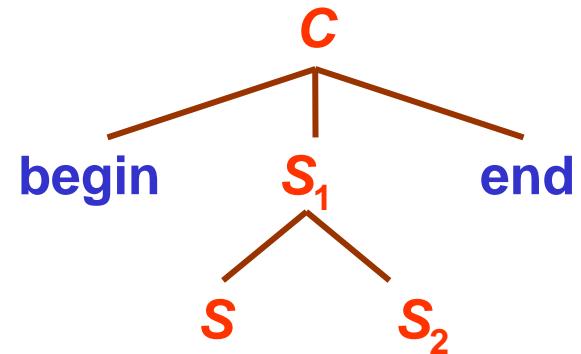
while var ≠ var do

var := var

end

Top – down string parsing

```
C → begin S1 end
S1 → S S2
S2 → ; S1 | ε
S → var := var
S → while var ≠ var do S
S → begin S1 end
```



```
begin
  var := var ;
  while var ≠ var do
    var := var
end
```

Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$

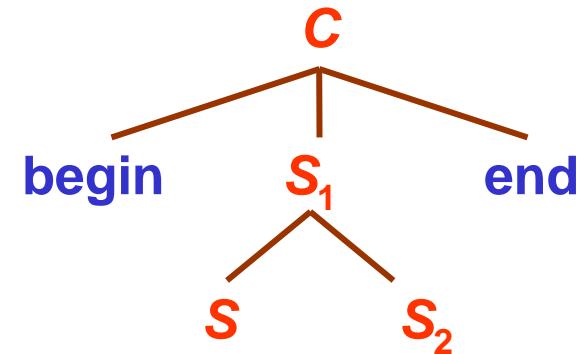
$S_1 \rightarrow S S_2$

$S_2 \rightarrow : S_1 \mid \epsilon$

$S \rightarrow \text{var} \ := \ \text{var}$

$S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$

$S \rightarrow \text{begin } S_1 \text{ end}$



begin

var := var ;

while var \neq var do

var := var

end

Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$

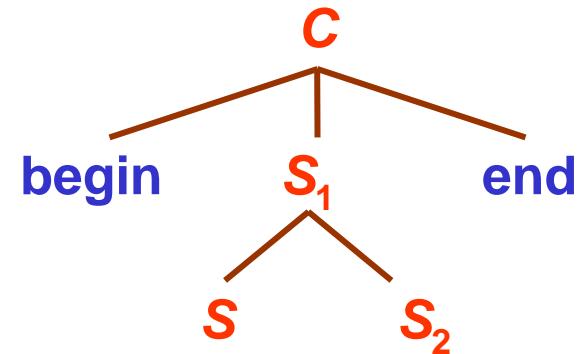
$S_1 \rightarrow S S_2$

$S_2 \rightarrow : S_1 \mid \epsilon$

$S \rightarrow \text{var} \ := \ \text{var}$

$S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$

$S \rightarrow \text{begin } S_1 \text{ end}$



begin

var := var ;

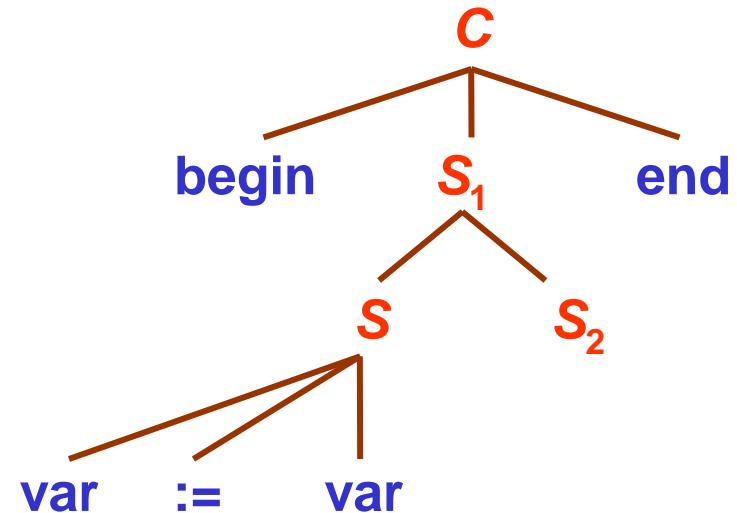
while var ≠ var do

var := var

end

Top – down string parsing

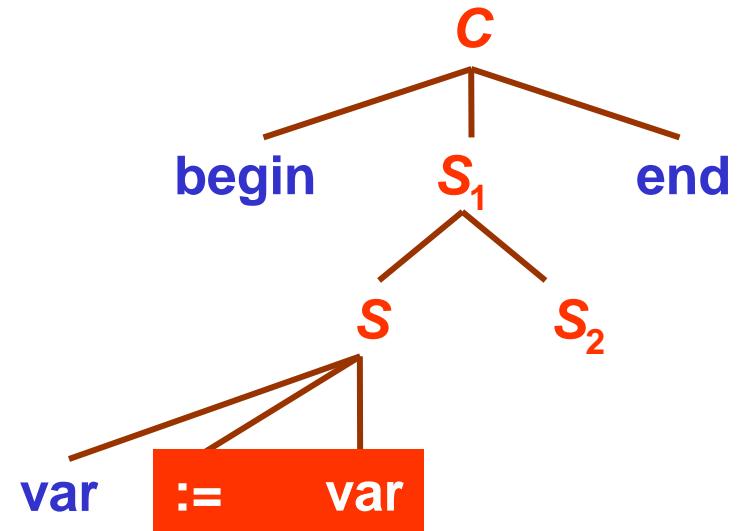
$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \ S_1 \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$



begin
 var := var ;
 while var ≠ var do
 var := var
end

Top – down string parsing

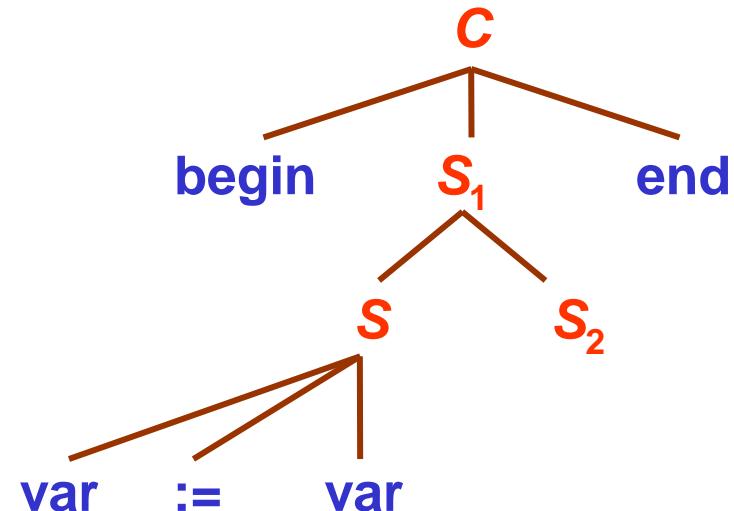
$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \ S_1 \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$



begin
 var := var ;
 while var ≠ var do
 var := var
end

Top – down string parsing

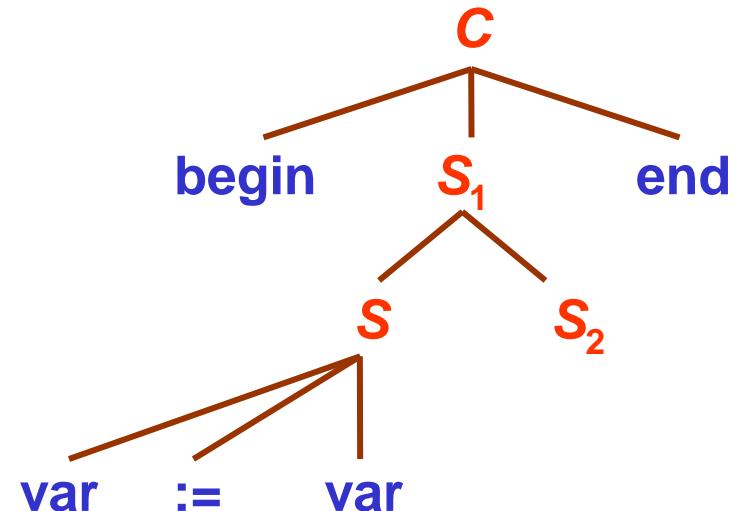
$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$



begin
 var | := var ;
 while var ≠ var do
 var := var
end

Top – down string parsing

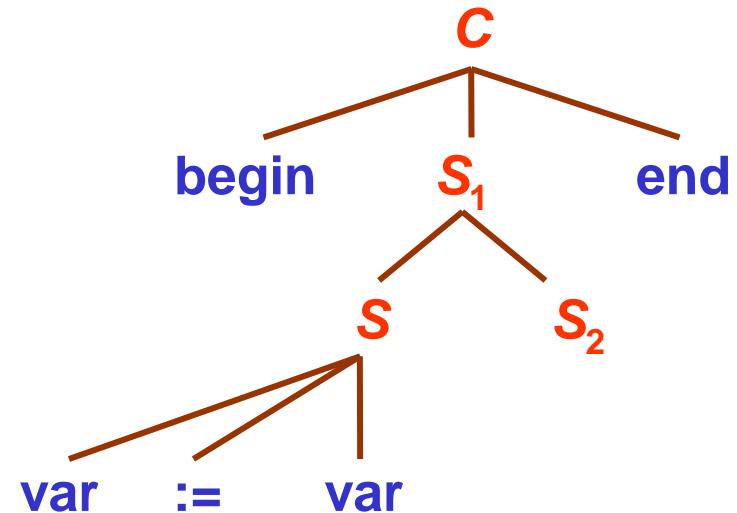
$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$



begin
 var | := var | ;
 while var ≠ var do
 var := var
end

Top – down string parsing

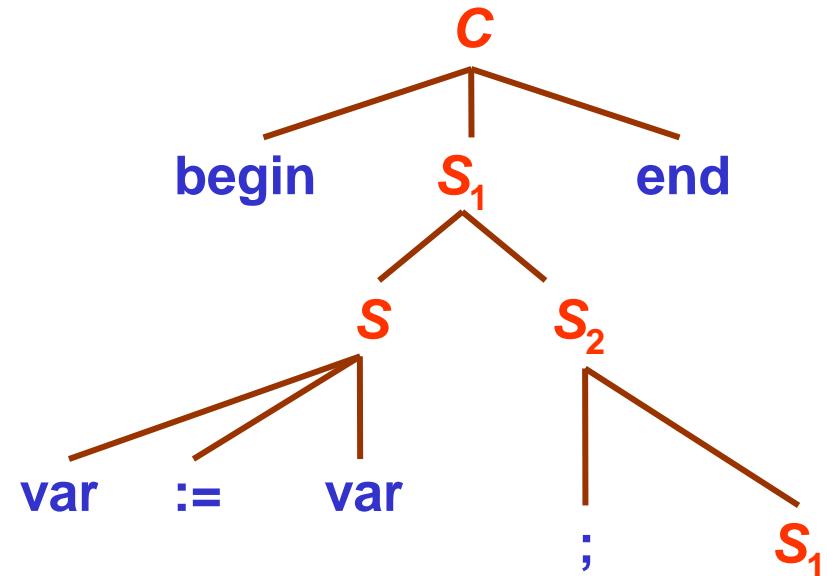
```
C → begin S1 end
S1 → S S2
S2 → ; S1 | ε
S → var := var
S → while var ≠ var do S
S → begin S1 end
```



```
begin
  var := var ;
  while var ≠ var do
    var := var
end
```

Top – down string parsing

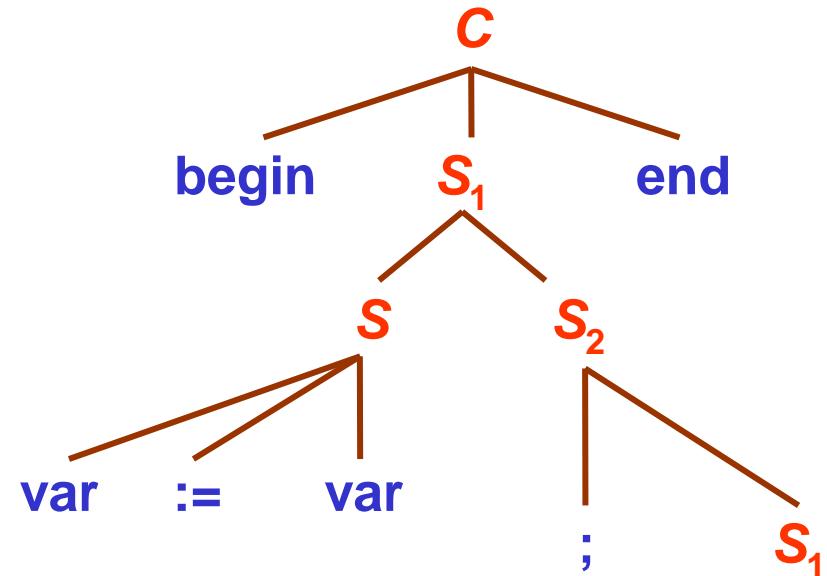
$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$



begin
 var | := var | ;
 while var ≠ var do
 var := var
end

Top – down string parsing

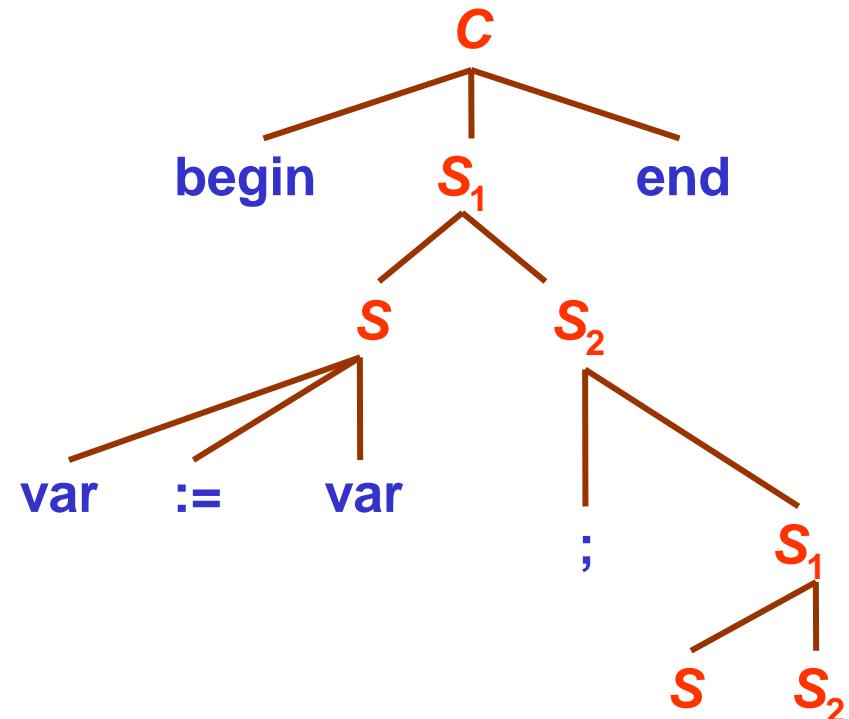
$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$



begin
 var | := var | ;
 while var ≠ var do
 var := var
end

Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$



begin
var | := var | ;
while var ≠ var do
var := var
end

Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$

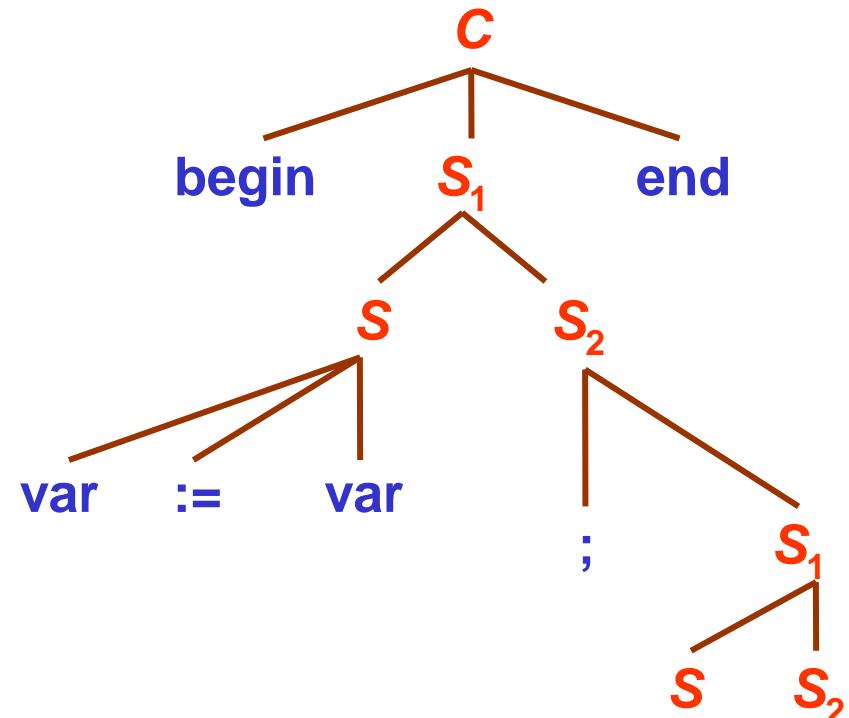
$S_1 \rightarrow S S_2$

$S_2 \rightarrow ; \mid \epsilon$

$S \rightarrow \text{var} \ := \ \text{var}$

$S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$

$S \rightarrow \text{begin } S_1 \text{ end}$



begin

var | := var | ;

while var \neq var do

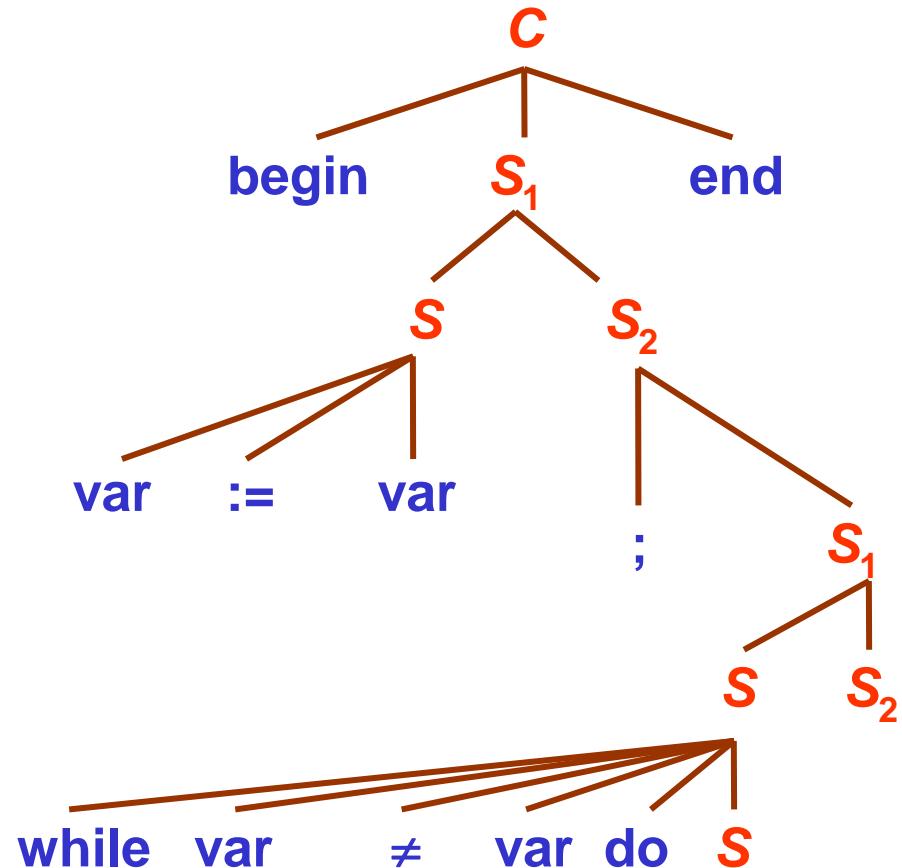
var := var

end

Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \ \text{do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$

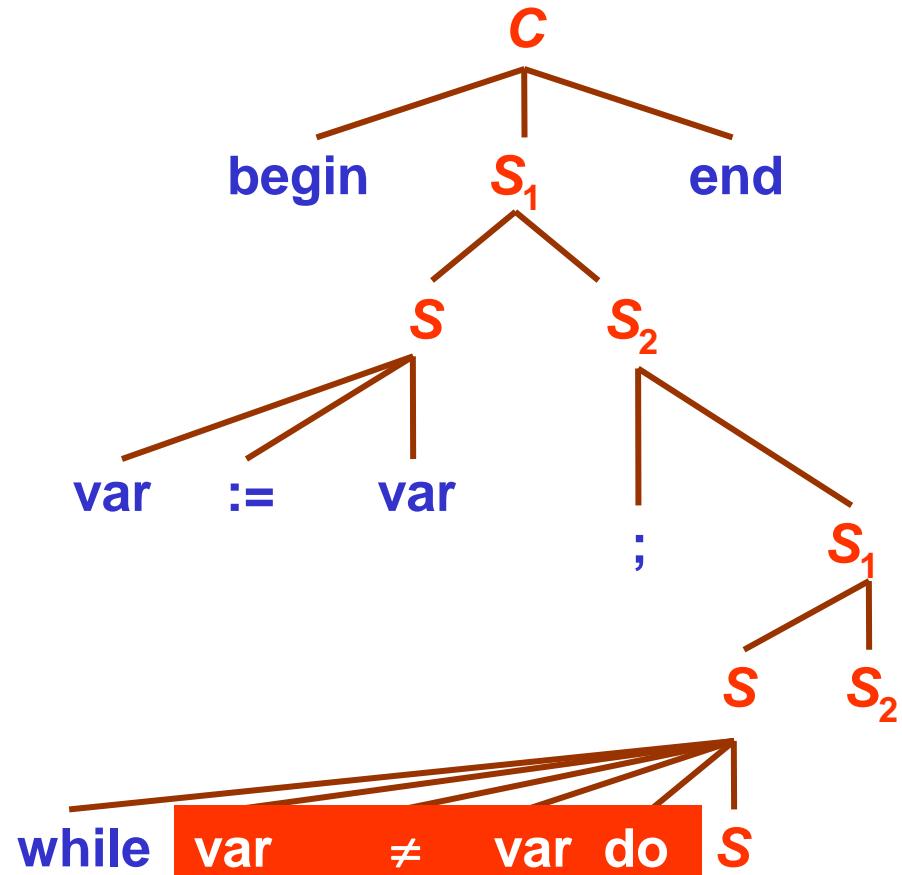
begin
 var | := var | ;
 while var ≠ var do
 var := var
end



Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \ \text{do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$

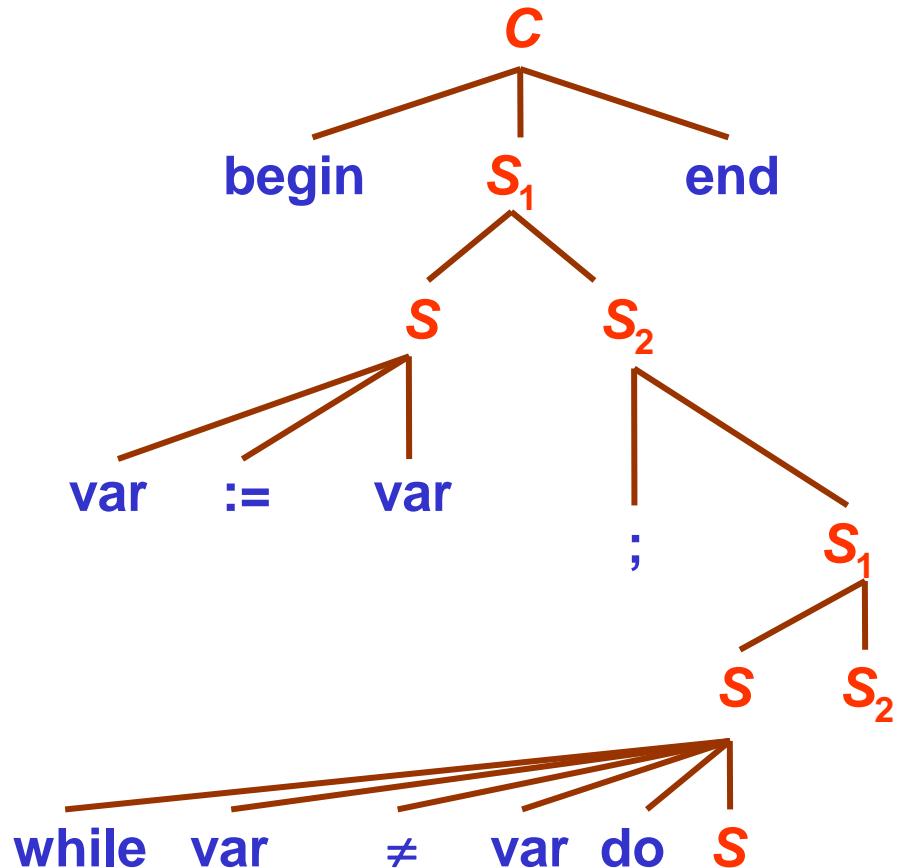
begin
 var | := var | ;
 while var ≠ var do
 var := var
end



Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \ \text{do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$

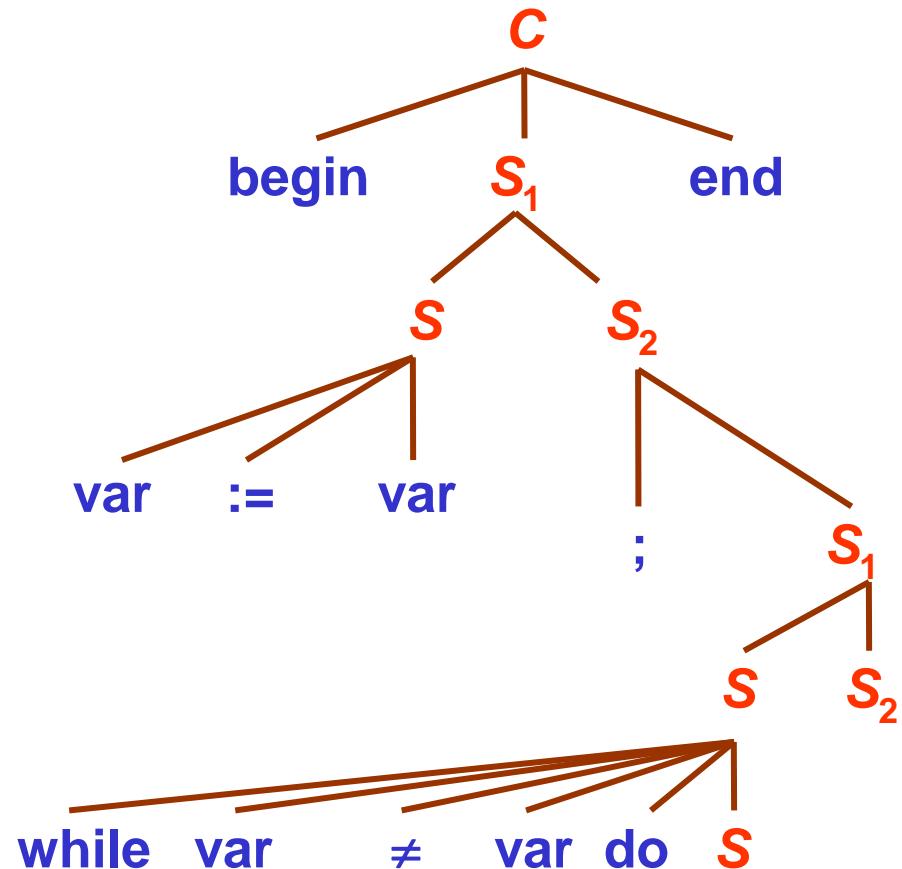
begin
 var | := var | ;
 while var ≠ var do
 var := var
end



Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \ \text{do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$

begin
var | := var | ;
while var ≠ var do
var := var
end



Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$

$S_1 \rightarrow S S_2$

$S_2 \rightarrow : S_1 \mid \epsilon$

$S \rightarrow \text{var} \ := \ \text{var}$

$S \rightarrow \text{while var} \neq \text{var do } S$

$S \rightarrow \text{begin } S_1 \text{ end}$

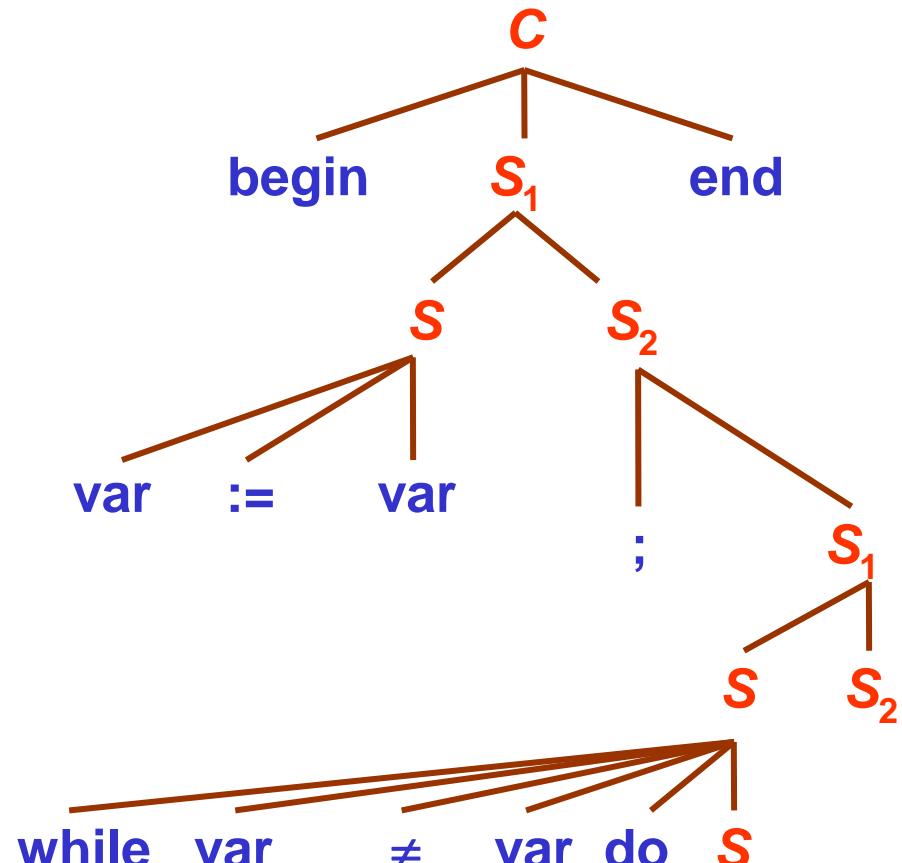
begin

var | := var | ;

while var ≠ var do

var := var

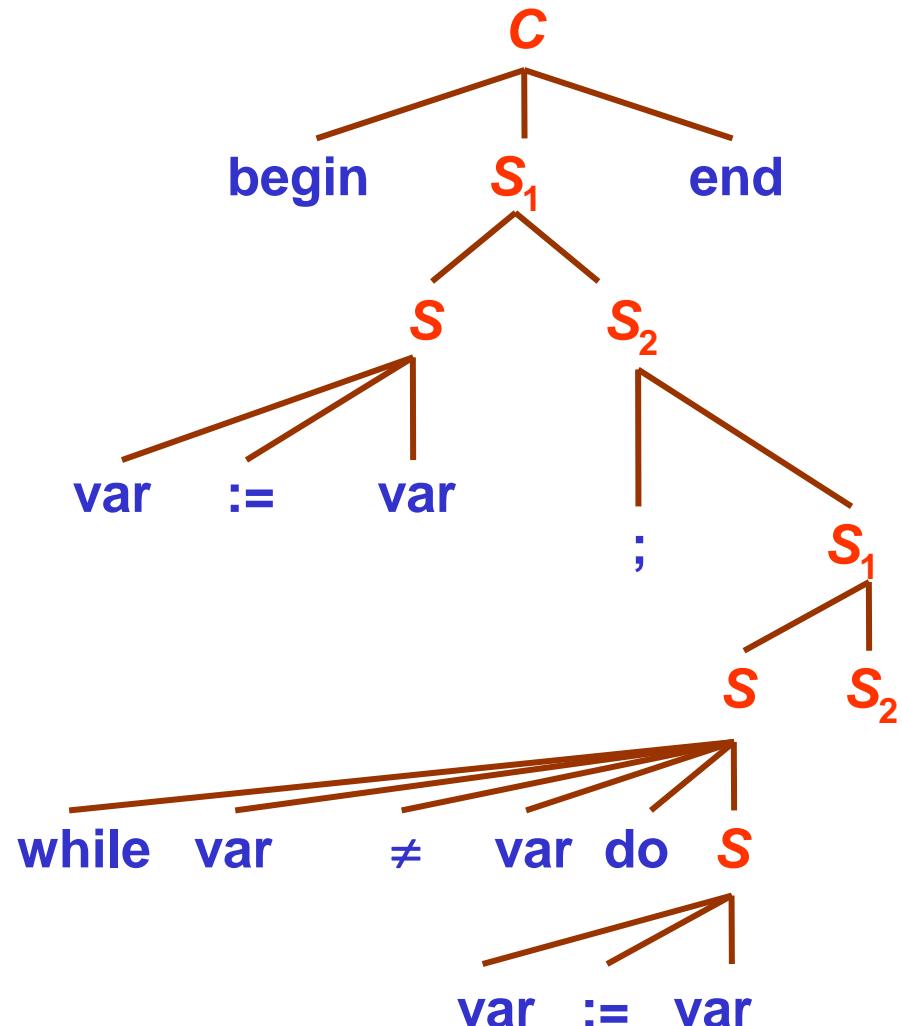
end



Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \ \text{do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$

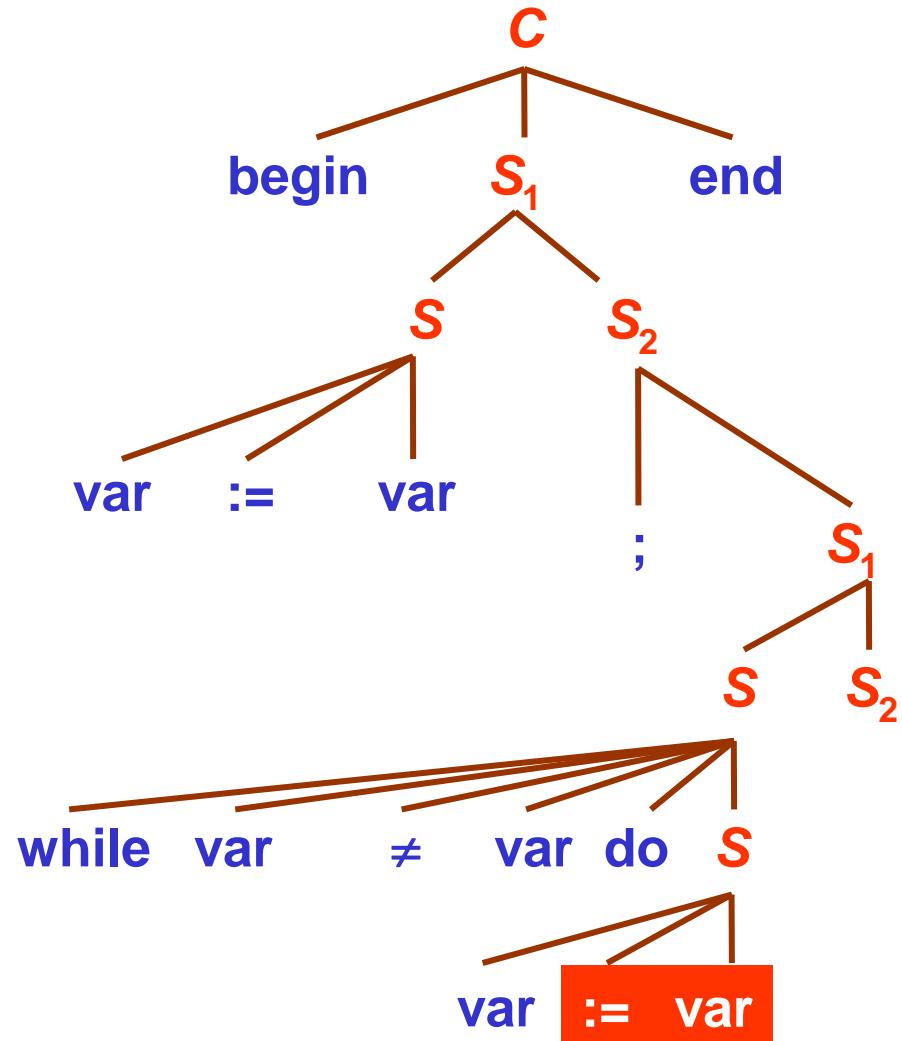
begin
var | := var | ;
while var ≠ var do
var := var
end



Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \ \text{do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$

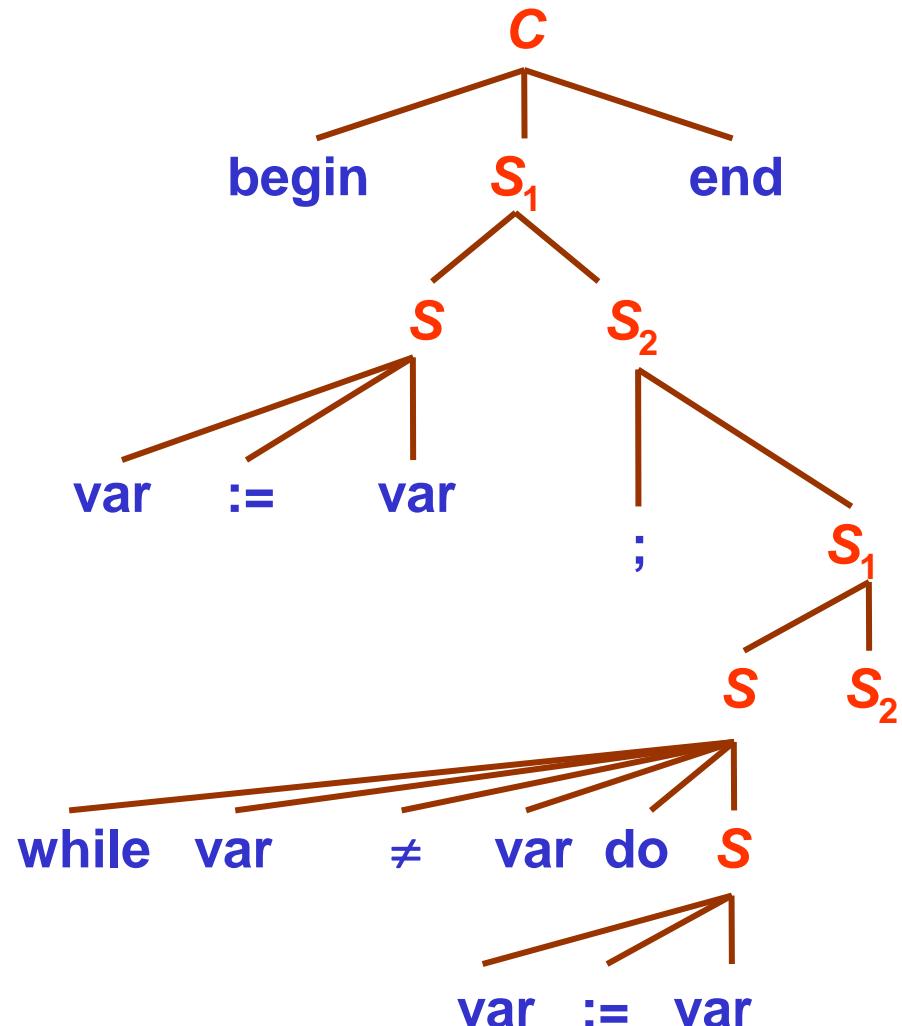
begin
var | := var | ;
while var ≠ var do
var := var
end



Top – down string parsing

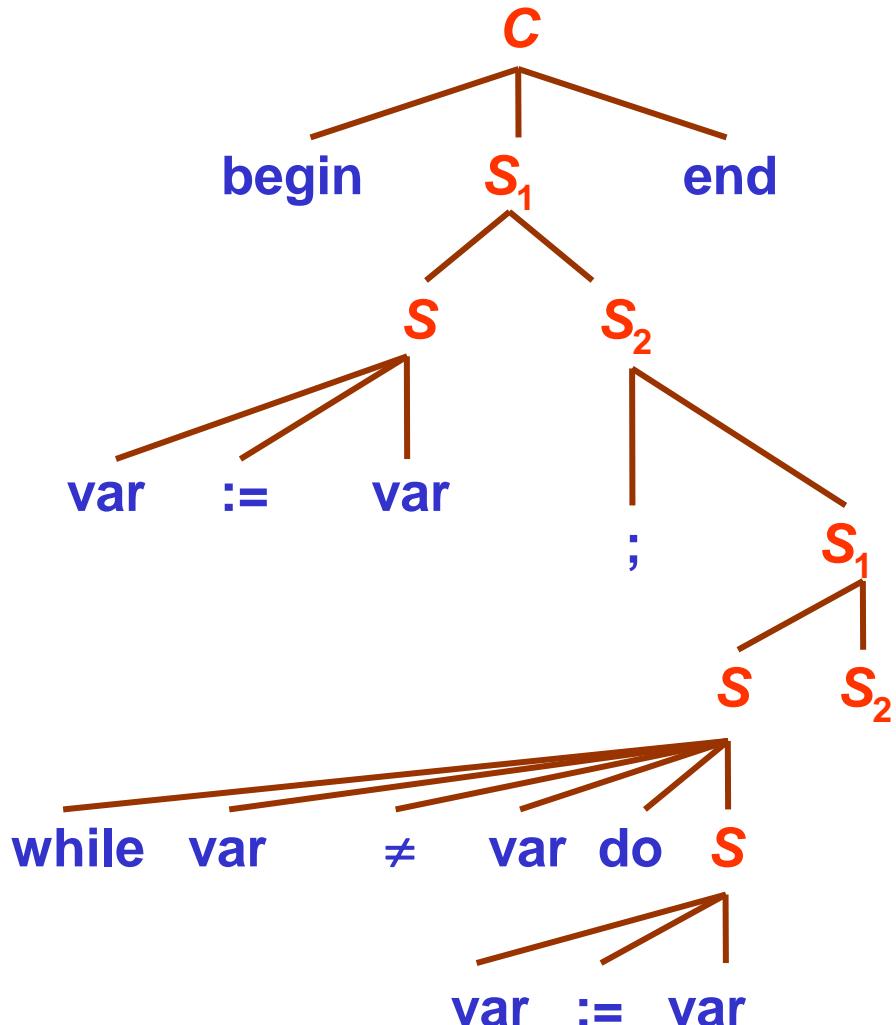
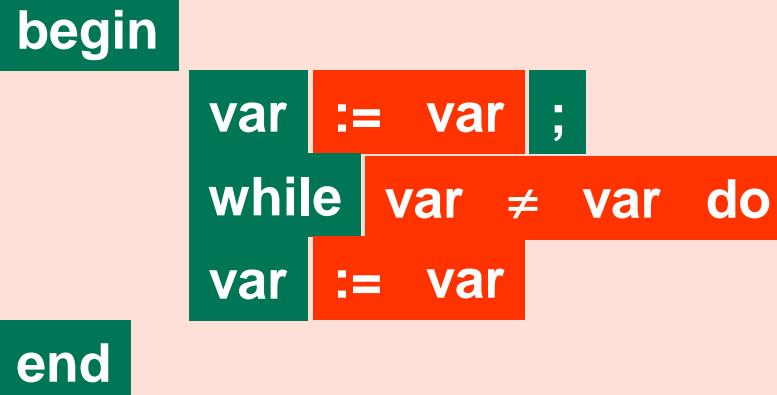
$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \ \text{do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$

begin
var | := var | ;
while var ≠ var do
var | := var
end



Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$



Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$

$S_1 \rightarrow S S_2$

$S_2 \rightarrow ; \mid \epsilon$

$S \rightarrow \text{var} \ := \ \text{var}$

$S \rightarrow \text{while } \text{var} \neq \text{var} \ \text{do } S$

$S \rightarrow \text{begin } S_1 \text{ end}$

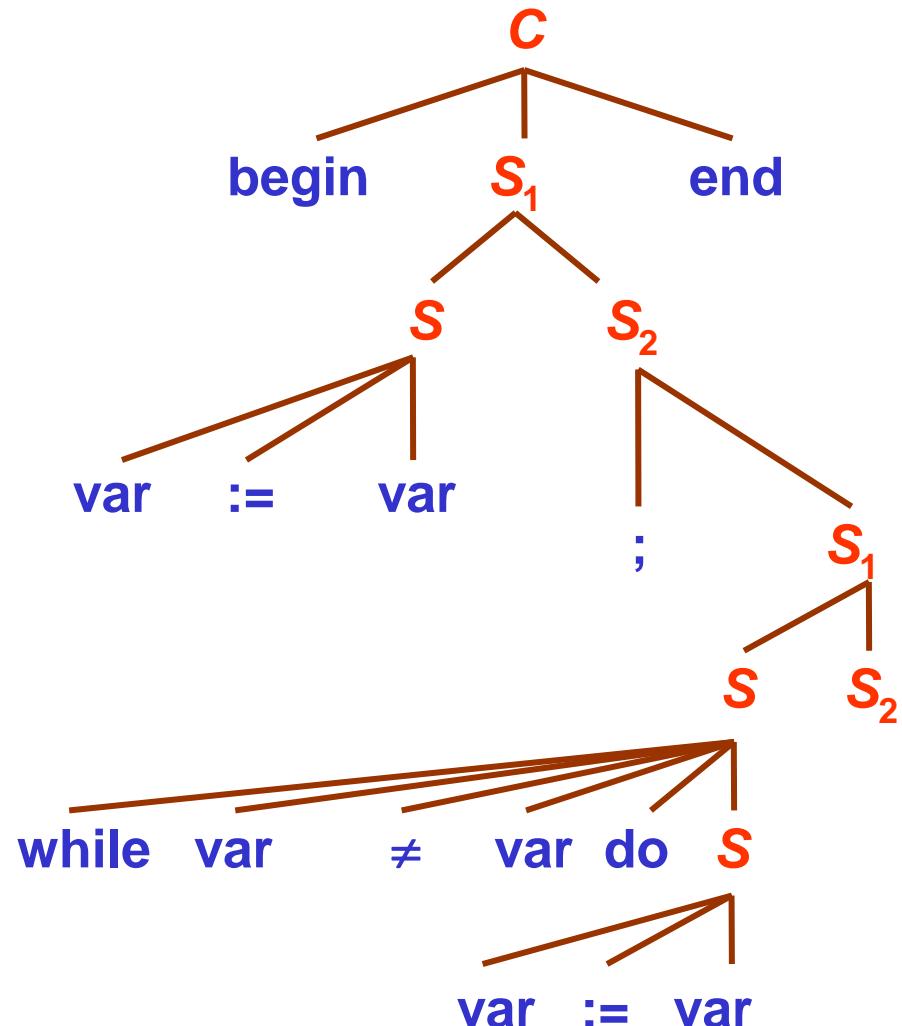
begin

var | := var | ;

while | var | ≠ | var | do

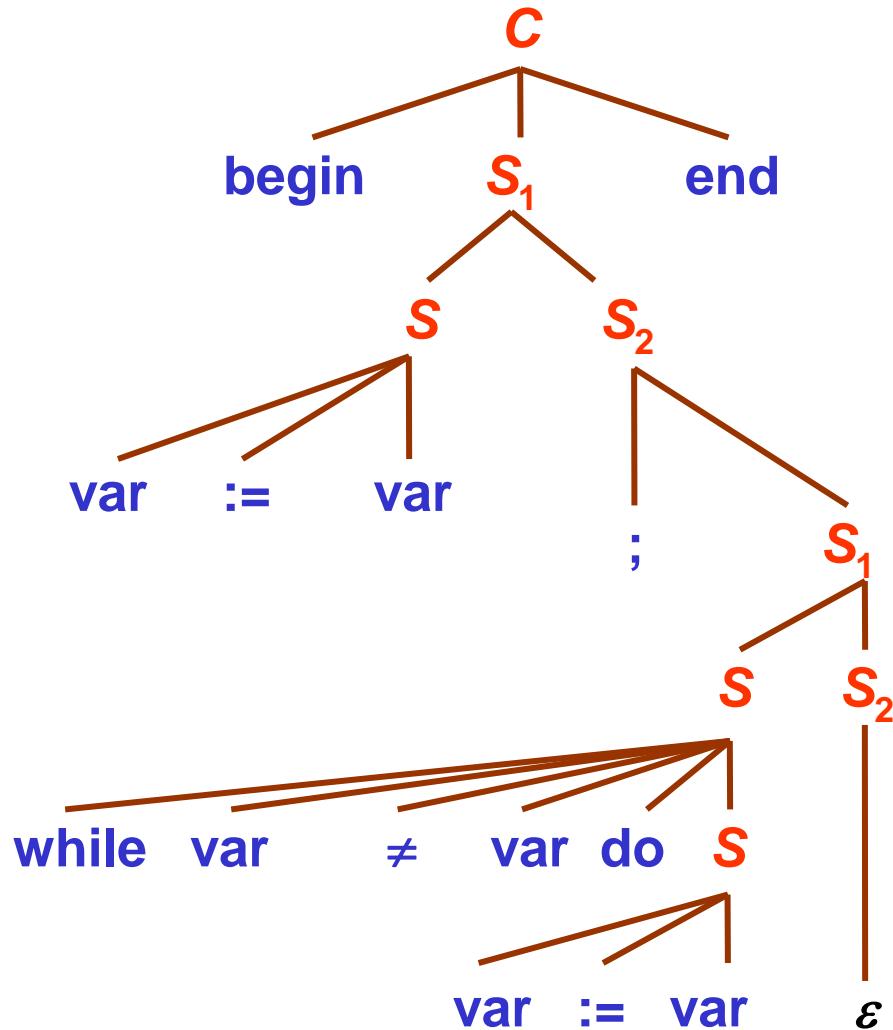
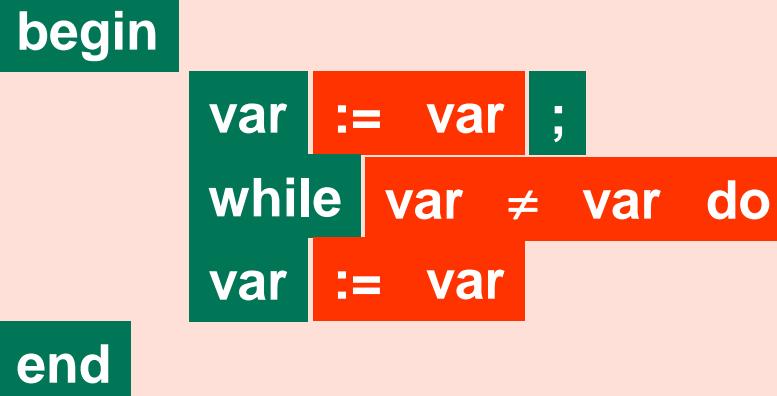
var | := var

end



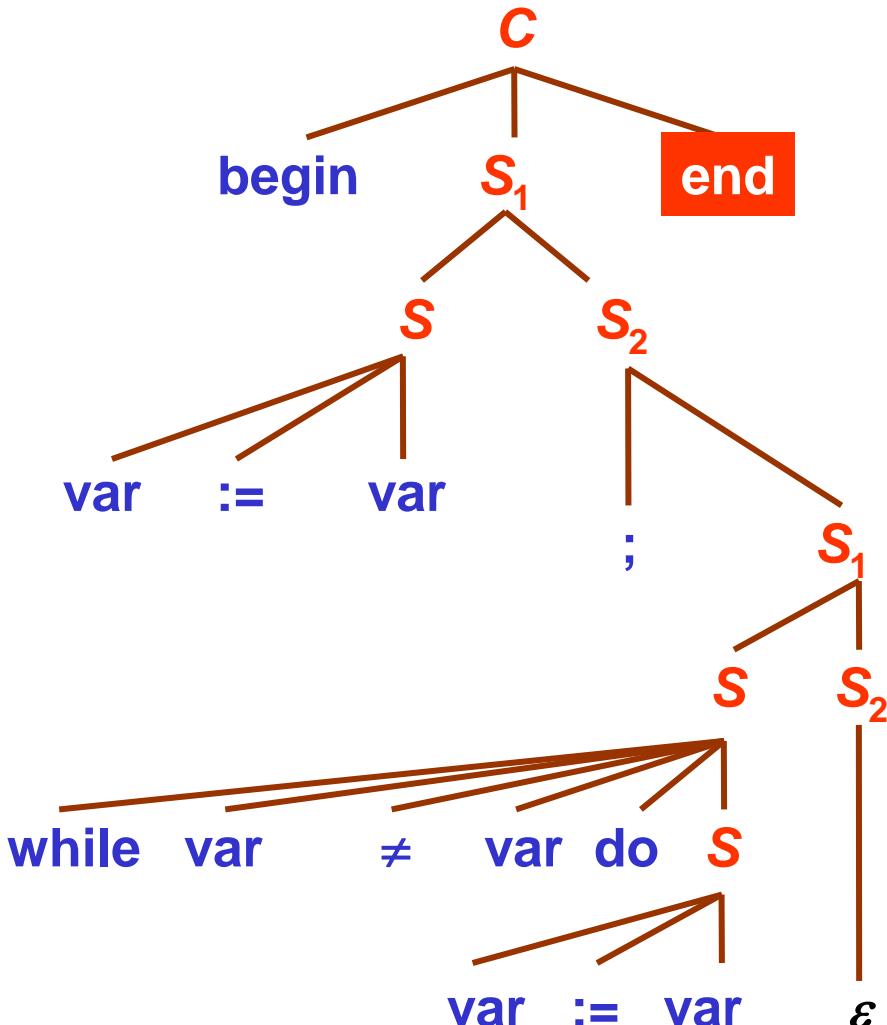
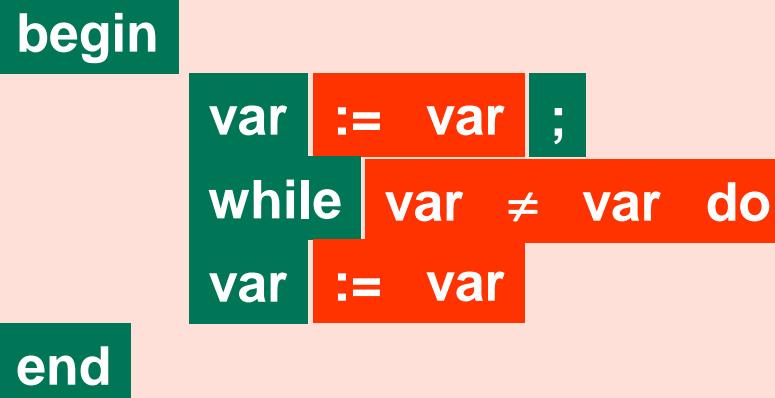
Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \ \text{do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$



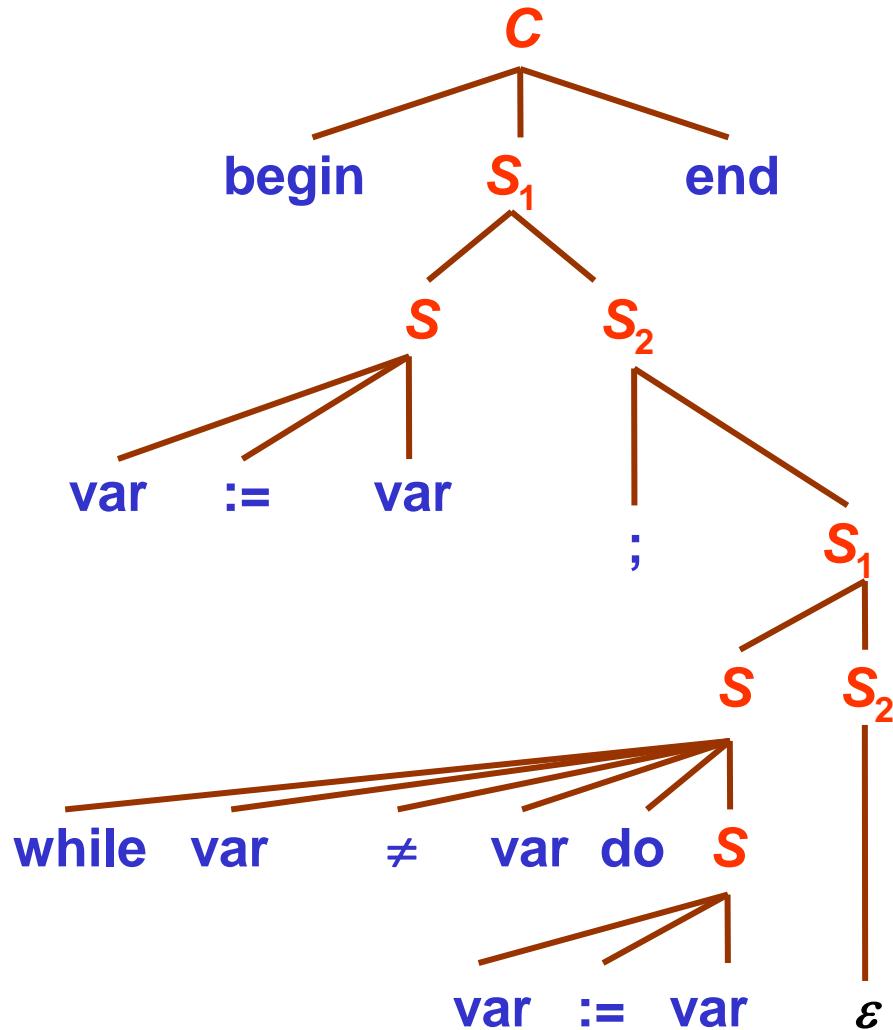
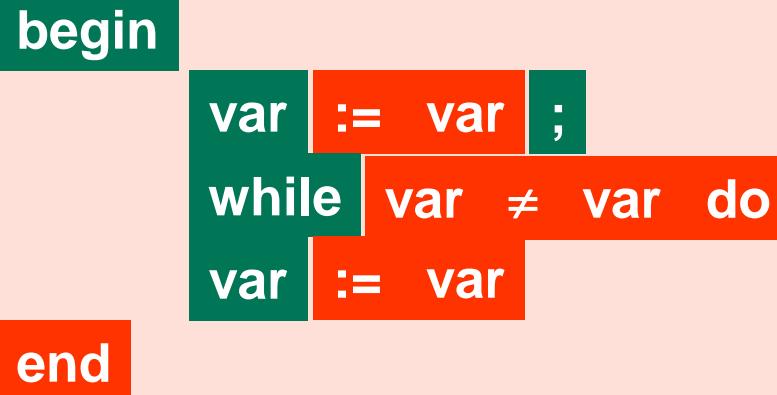
Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \ \text{do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$



Top – down string parsing

$C \rightarrow \text{begin } S_1 \text{ end}$
 $S_1 \rightarrow S S_2$
 $S_2 \rightarrow ; \mid \epsilon$
 $S \rightarrow \text{var} \ := \ \text{var}$
 $S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$
 $S \rightarrow \text{begin } S_1 \text{ end}$



Top – down string parsing

Top – down string parsing

- $LL(k)$ grammar

Top – down string parsing

- $LL(k)$ grammar
- $LL(k)$ parsing

Top – down string parsing

- $LL(k)$ grammar
- $LL(k)$ parsing
 - **first "L"**
 - The string is parsed from left to right (**Left-to-right scanning**)

Top – down string parsing

- $LL(k)$ grammar
- $LL(k)$ parsing
 - **first "L"**
 - The string is parsed from left to right (**Left-to-right scanning**)
 - **second "L"**
 - The Leftmost derivation algorithm is used while generating parse tree (**Leftmost derivation**)

Top – down string parsing

- $LL(k)$ grammar
- $LL(k)$ parsing
 - first " L "
 - The string is parsed from left to right (**Left-to-right scanning**)
 - second " L "
 - The Leftmost derivation algorithm is used while generating parse tree (**Leftmost derivation**)
 - k
 - The decision which production is applied is based on k read symbols
 - $k = 1, 2, 3, 4, \dots$

Top – down string parsing

- $LL(k)$ grammar
- $LL(k)$ parsing
 - first " L "
 - The string is parsed from left to right (**Left-to-right scanning**)
 - second " L "
 - The Leftmost derivation algorithm is used while generating parse tree (**Leftmost derivation**)
 - k
 - The decision which production is applied is based on k read symbols
 - $k = 1, 2, 3, 4, \dots$
- $LL(1)$ – syntax parsing for programming languages

Recursive Descent Parsing

Recursive Descent Parsing

- **Recursive descent parsing is implemented in a programming language**

Recursive Descent Parsing

- **Recursive descent parsing is implemented in a programming language**
 - The programming language need to have a recursive function call feature

Recursive Descent Parsing

- **Recursive descent parsing is implemented in a programming language**
 - The programming language need to have a recursive function call feature
- **Nonterminal grammar symbols**

Recursive Descent Parsing

- **Recursive descent parsing is implemented in a programming language**
 - The programming language need to have a recursive function call feature
- **Nonterminal grammar symbols**
 - Each nonterminal symbol is associated with the function

Recursive Descent Parsing

- **Recursive descent parsing is implemented in a programming language**
 - The programming language need to have a recursive function call feature
- **Nonterminal grammar symbols**
 - Each nonterminal symbol is associated with the function
 - Function checks if the substring of the input string matches the right side of the production associated to the nonterminal symbol on the left side of the production

Recursive Descent Parsing

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

MainProgram()

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

MainProgram()
{

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

MainProgram()
{

Input = Leftmost symbol in string w;

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

MainProgram()
{

Input = Leftmost symbol in string w;

C();

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

MainProgram()
{

Input = Leftmost symbol in string *w*;

C();

if(*Input* != ⊥)

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

MainProgram()
{

Input = Leftmost symbol in string *w*;

C();

if(*Input* != \perp)
Write (“ *w* $\notin L(G)$ ”);

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

MainProgram()
{

Input = Leftmost symbol in string *w*;

C();

if(*Input* != \perp)
 Write (“ *w* $\notin L(G)$ ”);
else

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

MainProgram()
{

Input = Leftmost symbol in string *w*;

C();

if(*Input* != \perp)
Write(“ *w* $\notin L(G)$ ”);

else
Write(“ *w* $\in L(G)$ ”);

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

MainProgram()
{

Input = Leftmost symbol in string *w*;

C();

if(*Input* != \perp)
Write(“ *w* $\notin L(G)$ ”);

else
Write(“ *w* $\in L(G)$ ”);

}

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$
$$S_1 \rightarrow S S_2$$
$$S_2 \rightarrow ; S_1$$
$$S_2 \rightarrow \varepsilon$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$
$$S_1 \rightarrow S S_2$$
$$S_2 \rightarrow ; S_1$$
$$S_2 \rightarrow \varepsilon$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$

C()

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$
$$S_1 \rightarrow S S_2$$
$$S_2 \rightarrow ; S_1$$
$$S_2 \rightarrow \varepsilon$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$
$$C()$$

{

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$
$$S_1 \rightarrow S S_2$$
$$S_2 \rightarrow ; S_1$$
$$S_2 \rightarrow \varepsilon$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$

C()
{
 if(*Input* != **begin**)

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$
$$S_1 \rightarrow S S_2$$
$$S_2 \rightarrow ; S_1$$
$$S_2 \rightarrow \epsilon$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$
$$C()$$

{

if(*Input* != begin)

Write (“ *w* $\notin L(G)$ ”);

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$
$$S_1 \rightarrow S S_2$$
$$S_2 \rightarrow ; S_1$$
$$S_2 \rightarrow \epsilon$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$
$$C()$$

{

if(*Input* != begin)
 Write (“ *w* $\notin L(G)$ ”);

Input = Read the next symbol in string *w*;

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$
$$S_1 \rightarrow S S_2$$
$$S_2 \rightarrow ; S_1$$
$$S_2 \rightarrow \epsilon$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$
$$C()$$

{

if(*Input* != begin)
 Write (“ *w* $\notin L(G)$ ”);

Input = Read the next symbol in string *w*;

$$S_1();$$

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$
$$S_1 \rightarrow S S_2$$
$$S_2 \rightarrow ; S_1$$
$$S_2 \rightarrow \epsilon$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$
$$C()$$

{

if(*Input* != begin)

Write (“ *w* $\notin L(G)$ ”);

Input = Read the next symbol in string *w*;

S1();

if(*Input* != end)

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$
$$S_1 \rightarrow S S_2$$
$$S_2 \rightarrow ; S_1$$
$$S_2 \rightarrow \epsilon$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$
$$C()$$

{

if(*Input* != begin)
 Write (“ *w* $\notin L(G)$ ”);

Input = Read the next symbol in string *w*;

$$S_1();$$

if(*Input* != end)
 Write (“ *w* $\notin L(G)$ ”);

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$

$$S_1 \rightarrow S S_2$$

$$S_2 \rightarrow ; S_1$$

$$S_2 \rightarrow \epsilon$$

$$S \rightarrow \text{var} := \text{var}$$

$$S \rightarrow \text{while var} \neq \text{var do } S$$

$$S \rightarrow \text{begin } S_1 \text{ end}$$

C()

{

if(*Input* != begin)

Write (“ **w** $\notin L(G)$ ”);

Input = Read the next symbol in string **w**;

S1();

if(*Input* != end)

Write (“ **w** $\notin L(G)$ ”);

Input = Read the next symbol in string **w**;

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$
$$S_1 \rightarrow S S_2$$
$$S_2 \rightarrow ; S_1$$
$$S_2 \rightarrow \epsilon$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$
$$C()$$

{

if(*Input* != begin)

Write (“ *w* $\notin L(G)$ ”);

Input = Read the next symbol in string *w*;

S1();

if(*Input* != end)

Write (“ *w* $\notin L(G)$ ”);

Input = Read the next symbol in string *w*;

}

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$
$$S_1 \rightarrow S S_2$$
$$S_2 \rightarrow ; S_1$$
$$S_2 \rightarrow \varepsilon$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var} \text{ do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

S1()

Recursive Descent Parsing

$$C \rightarrow \text{begin } S_1 \text{ end}$$
$$S_1 \rightarrow S S_2$$
$$S_2 \rightarrow ; S_1$$
$$S_2 \rightarrow \epsilon$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var} \text{ do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$

S1()
{

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

S1()
{
S();

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

S1()
{
 S();
 S2();

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S1()
{
    S();
    S2();
}
```

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

$S_2()$

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

S2()
{

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S2()
{
    if( Input == ; )
```

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S2()
{
    if( Input == ; )
    {
```

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

S2()
{

if(*Input* == ;)
{

Input = Read the next symbol in string *w*;

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S2()
{
    if( Input == ; )
    {
```

Input = Read the next symbol in string w;
 $S_1();$

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S2()
{
    if( Input == ; )
    {
```

Input = Read the next symbol in string w;
S1();

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S2()
{
    if( Input == ; )
    {
        Input = Read the next symbol in string w;
        S1();
    }
}
```

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S2()
{
    if( Input == ; )
    {
        Input = Read the next symbol in string w;
        S1();
    }
}
```

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

S()

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

S()
{

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

S()
{
 case (*Input*)

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S()  
{  
    case ( Input )  
    {
```

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S()
{
    case ( Input )
    {
        var:
```

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S()  
{  
    case ( Input )  
    {  
        var:  
    }
```

Source code that checks substring (`:= var`)

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S()
{
    case ( Input )
    {
        var:
```

Source code that checks substring (:= var)

while:

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S()  
{  
    case ( Input )  
    {  
        var:  
    }
```

Source code that checks substring (:= var)

while:

**Source code that checks substring
(var ≠ var do S)**

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S()
{
    case ( Input )
    {
        var:
```

Source code that checks substring (:= var)

```
        while:
```

**Source code that checks substring
(var ≠ var do S)**

```
        begin :
```

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

```
S()  
{  
    case ( Input )  
    {  
        var:  
    }
```

Source code that checks substring (:= var)

while:

**Source code that checks substring
(var ≠ var do S)**

begin :

Source code that checks substring (S₁ end)

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

```
S()
{
    case ( Input )
    {
        var:
            Source code that checks substring ( := var )

        while:
            Source code that checks substring
            ( var ≠ var do S )

        begin :
            Source code that checks substring ( S1 end )
    }
}
```

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S()
{
    case ( Input )
    {
        var:
            Source code that checks substring ( := var )

        while:
            Source code that checks substring
            ( var ≠ var do S )

        begin :
            Source code that checks substring ( S1 end )
    }
}
```

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

S()

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$S \rightarrow \text{var} := \text{var}$$
$$S \rightarrow \text{while var} \neq \text{var do } S$$
$$S \rightarrow \text{begin } S_1 \text{ end}$$

S()

{

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

S()
{
case (Input)

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S()
{
    case ( Input )
{
```

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S()
{
    case ( Input )
    {
        var:
```

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S()
{
    case ( Input )
    {
        var:
```

Input = Read the next symbol in string w;

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

```
S()
{
    case ( Input )
    {
        var:
```

Input = Read the next symbol in string w;
*if(***Input** != **:=**)

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

```
S()
{
    case ( Input )
    {
        var:
```

Input = Read the next symbol in string w;
*if(***Input** != **:=**)
 Write (“ w **notin** *L(G) ”);*

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$S \rightarrow \text{var} := \text{var}$$

$$S \rightarrow \text{while var} \neq \text{var do } S$$

$$S \rightarrow \text{begin } S_1 \text{ end}$$

```
S()
{
    case ( Input )
    {
        var:
```

Input = Read the next symbol in string w;
*if(***Input** != **:=**)
 Write (“ w **notin** *L(G) ”);*

Input = Read the next symbol in string w;

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

```
S()
{
    case ( Input )
    {
        var:
```

Input = Read the next symbol in string w;
if(*Input* != *:=*)
 Write (“ *w* $\notin L(G)$ ”);

Input = Read the next symbol in string w;
if(*Input* != *var*)

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; \ S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$S \rightarrow \text{var} \ := \ \text{var}$$

$$S \rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S$$

$$S \rightarrow \text{begin } S_1 \text{ end}$$

```
S()
{
    case ( Input )
    {
        var:
```

Input = Read the next symbol in string w;
if(*Input* != *:=*)
 Write (“ *w* $\notin L(G)$ ”);

Input = Read the next symbol in string w;
if(*Input* != *var*)
 Write (“ *w* $\notin L(G)$ ”);

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$S \rightarrow \text{var} := \text{var}$$

$$S \rightarrow \text{while var} \neq \text{var do } S$$

$$S \rightarrow \text{begin } S_1 \text{ end}$$

```
S()
{
    case ( Input )
    {
        var:
```

Input = Read the next symbol in string w;
if(*Input* != *:=*)
 Write (“ *w* $\notin L(G)$ ”);

Input = Read the next symbol in string w;
if(*Input* != *var*)
 Write (“ *w* $\notin L(G)$ ”);

Input = Read the next symbol in string w;

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

while:

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \epsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} \ := \ \text{var} \\ S &\rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

while:

Input = Read the next symbol in string w;
if(Input != var)

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} \ := \ \text{var} \\ S &\rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;
if(Input != var)
 Write (“ w $\notin L(G)$ ”);

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} \ := \ \text{var} \\ S &\rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;
if(Input != var)
 Write (“ w ∈ L(G) ”);

Input = Read the next symbol in string w;

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} \ := \ \text{var} \\ S &\rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;
if(Input != var)
 Write (“ w ∈ L(G) ”);

Input = Read the next symbol in string w;
if(Ulaz != ≠)

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} \ := \ \text{var} \\ S &\rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;
if(Input != var)
 Write (“ w $\notin L(G)$ ”);

Input = Read the next symbol in string w;
if(Ulaz != ≠)
 Write (“ w $\notin L(G)$ ”);

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;
if(Input != var)
 Write (“ w $\notin L(G)$ ”);

Input = Read the next symbol in string w;
if(Ulaz != ≠)
 Write (“ w $\notin L(G)$ ”);

Input = Read the next symbol in string w;

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; \ S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} \ := \ \text{var} \\ S &\rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;
if(Input != var)
 Write (“ w ∈ L(G) ”);

Input = Read the next symbol in string w;
if(Ulaz != ≠)
 Write (“ w ∈ L(G) ”);

Input = Read the next symbol in string w;
if(Input != var)

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; \ S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} \ := \ \text{var} \\ S &\rightarrow \text{while } \text{var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;
if(Input != var)
 Write (“ w ∈ L(G) ”);

Input = Read the next symbol in string w;
if(Ulaz != ≠)
 Write (“ w ∈ L(G) ”);

Input = Read the next symbol in string w;
if(Input != var)
 Write (“ w ∈ L(G) ”);

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;
if(Input != var)
 Write (“ w ∈ L(G) ”);

Input = Read the next symbol in string w;
if(Ulaz != ≠)
 Write (“ w ∈ L(G) ”);

Input = Read the next symbol in string w;
if(Input != var)
 Write (“ w ∈ L(G) ”);

Input = Read the next symbol in string w;

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;
if(Input != var)
 Write (“ w ∈ L(G) ”);

Input = Read the next symbol in string w;
if(Ulaz != ≠)
 Write (“ w ∈ L(G) ”);

Input = Read the next symbol in string w;
if(Input != var)
 Write (“ w ∈ L(G) ”);

Input = Read the next symbol in string w;
if(Input != do)

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;
if(Input != var)
 Write (" w ∈ L(G) ");

Input = Read the next symbol in string w;
if(Ulaz != ≠)
 Write (" w ∈ L(G) ");

Input = Read the next symbol in string w;
if(Input != var)
 Write (" w ∈ L(G) ");

Input = Read the next symbol in string w;
if(Input != do)
 Write (" w ∈ L(G) ");

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;
if(Input != var)
 Write (" w ∈ L(G) ");

Input = Read the next symbol in string w;
if(Ulaz != ≠)
 Write (" w ∈ L(G) ");

Input = Read the next symbol in string w;
if(Input != var)
 Write (" w ∈ L(G) ");

Input = Read the next symbol in string w;
if(Input != do)
 Write (" w ∈ L(G) ");

Input = Read the next symbol in string w;

Recursive Descent Parsing

$$\begin{aligned} C &\rightarrow \text{begin } S_1 \text{ end} \\ S_1 &\rightarrow S S_2 \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow ; S_1 \\ S_2 &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S &\rightarrow \text{var} := \text{var} \\ S &\rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S &\rightarrow \text{begin } S_1 \text{ end} \end{aligned}$$

while:

Input = Read the next symbol in string w;
if(Input != var)
 Write (" w ∈ L(G) ");

Input = Read the next symbol in string w;
if(Ulaz != ≠)
 Write (" w ∈ L(G) ");

Input = Read the next symbol in string w;
if(Input != var)
 Write (" w ∈ L(G) ");

Input = Read the next symbol in string w;
if(Input != do)
 Write (" w ∈ L(G) ");

Input = Read the next symbol in string w;
 S();

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$
$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \varepsilon \end{array}$$
$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

begin:

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

begin:

Input = Read the next symbol in string w;

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

begin:

*Input = Read the next symbol in string w ;
 $S1()$;*

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

begin:

Input = Read the next symbol in string w ;
 $S_1();$

if(*Input* != end)

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

begin:

Input = Read the next symbol in string w ;
 $S_1();$

if(*Input* != end)
Write (“ $w \notin L(G)$ ”);

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

begin:

Input = Read the next symbol in string w ;
 $S1();$

if($\text{Input} \neq \text{end}$)
Write (“ $w \notin L(G)$ ”);

Input = Read the next symbol in string w ;

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

begin:

Input = Read the next symbol in string w ;
 $S_1();$

if($\text{Input} \neq \text{end}$)
Write (“ $w \notin L(G)$ ”);

Input = Read the next symbol in string w ;

Other symbols:

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

begin:

Input = Read the next symbol in string w ;
 $S1();$

if($\text{Input} \neq \text{end}$)
 $Write(" w \notin L(G) ");$

Input = Read the next symbol in string w ;

Other symbols:

$Write(" w \notin L(G) ");$

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

begin:

Input = Read the next symbol in string w ;
 $S1();$

if($\text{Input} \neq \text{end}$)
 $Write(" w \notin L(G) ");$

Input = Read the next symbol in string w ;

Other symbols:

$Write(" w \notin L(G) ");$

}

Recursive Descent Parsing

$$\begin{array}{l} C \rightarrow \text{begin } S_1 \text{ end} \\ S_1 \rightarrow S S_2 \end{array}$$

$$\begin{array}{l} S_2 \rightarrow ; S_1 \\ S_2 \rightarrow \epsilon \end{array}$$

$$\begin{array}{l} S \rightarrow \text{var} := \text{var} \\ S \rightarrow \text{while var} \neq \text{var} \text{ do } S \\ S \rightarrow \text{begin } S_1 \text{ end} \end{array}$$

begin:

Input = Read the next symbol in string w ;
 $S_1();$

if($\text{Input} \neq \text{end}$)
 $Write(" w \notin L(G) ");$

Input = Read the next symbol in string w ;

Other symbols:

$Write(" w \notin L(G) ");$

}

begin var := var ; **while** var ≠ var **do** var := var **end** ⊥

Main Program

```
begin var := var ; while var ≠ var do var := var end ⊥
```

Main Program

```
begin var := var ; while var ≠ var do var := var end ⊥
```

Main Program

Input = begin

Function C

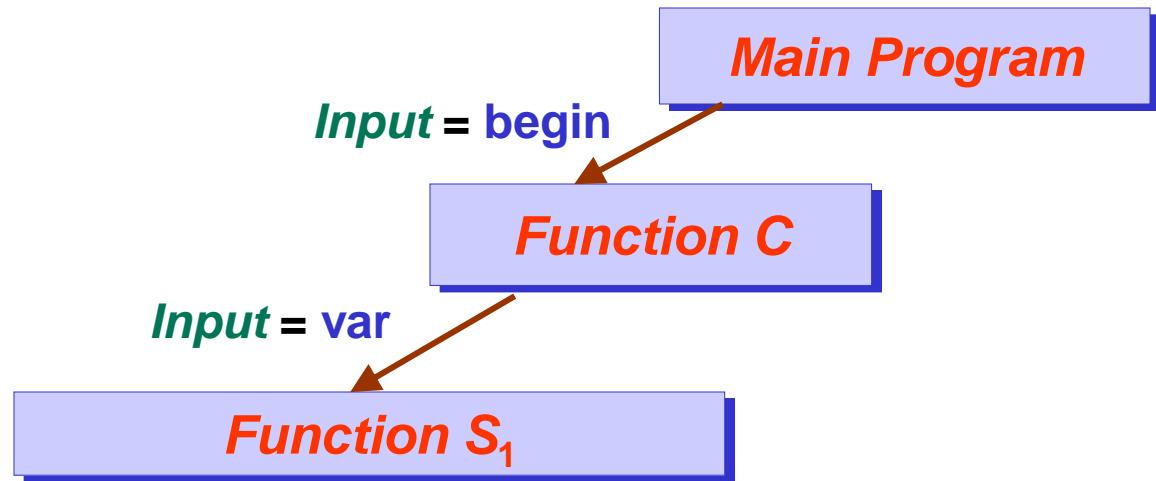
begin var := var ; while var ≠ var do var := var end ⊥

Main Program

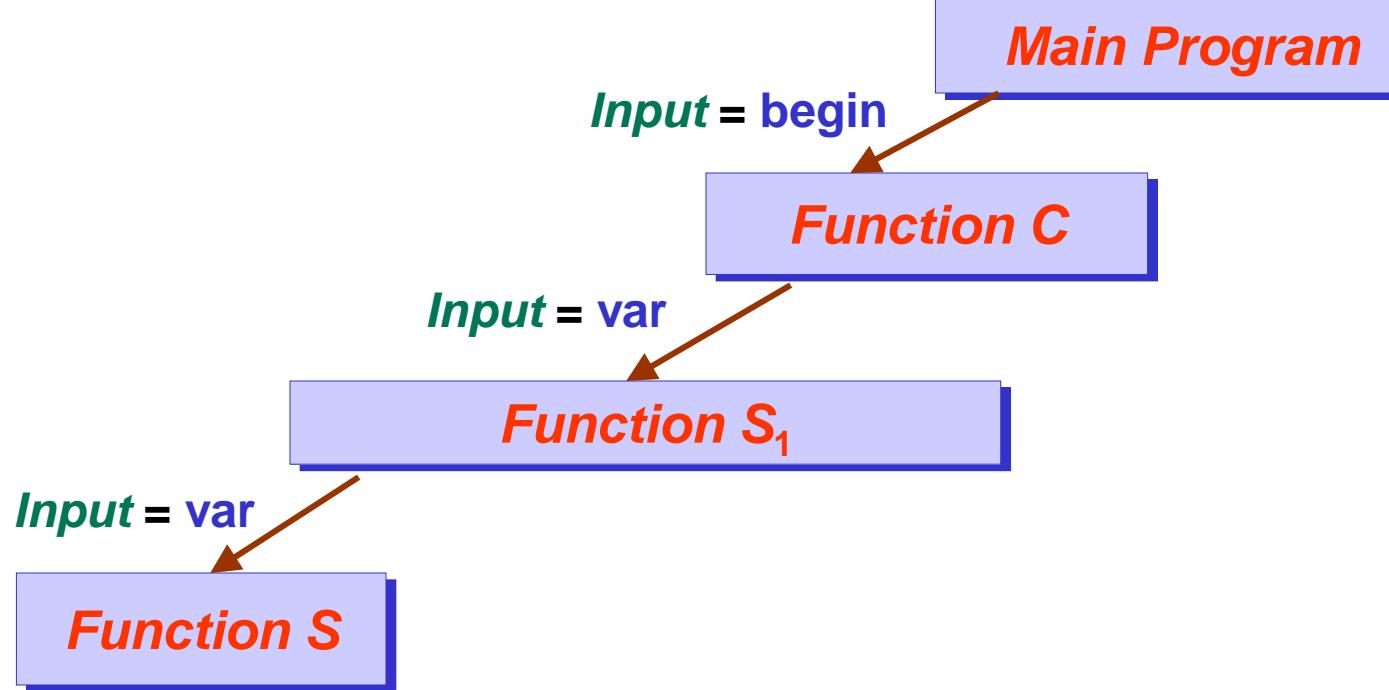
Input = begin

Function C

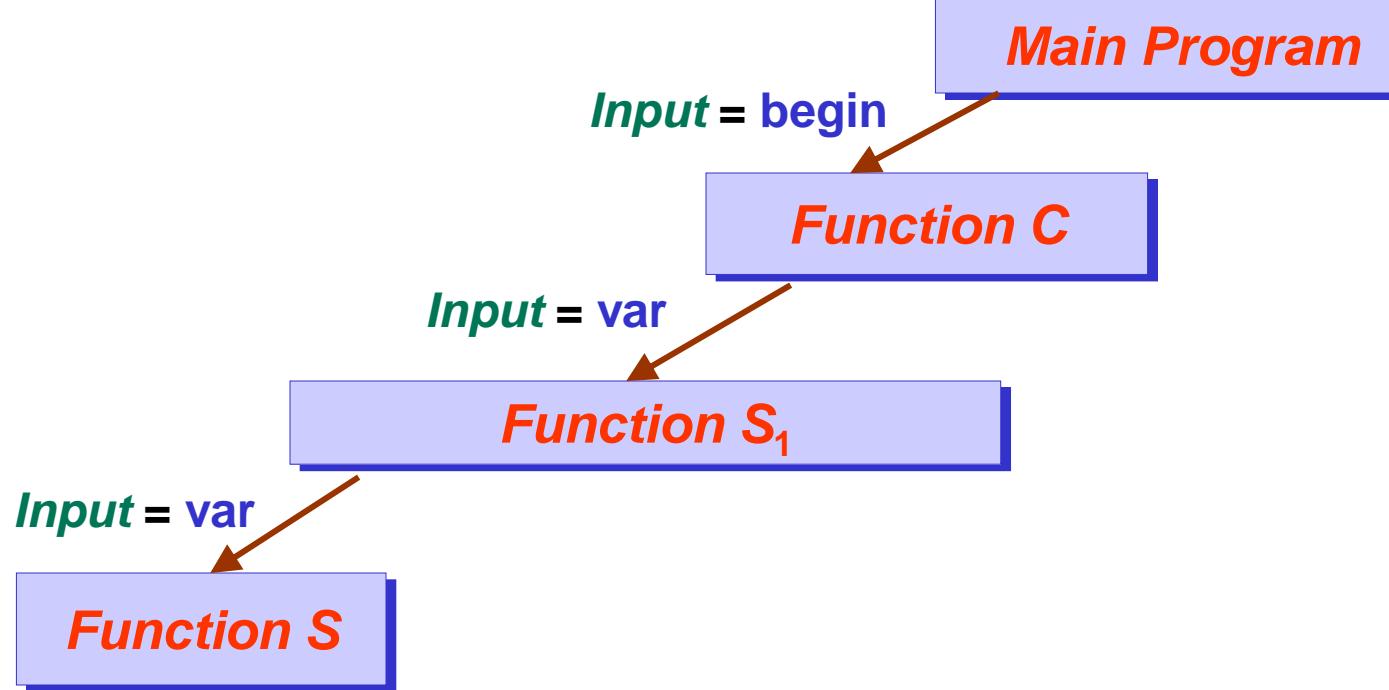
begin | var := var ; while var ≠ var do var := var end ⊥



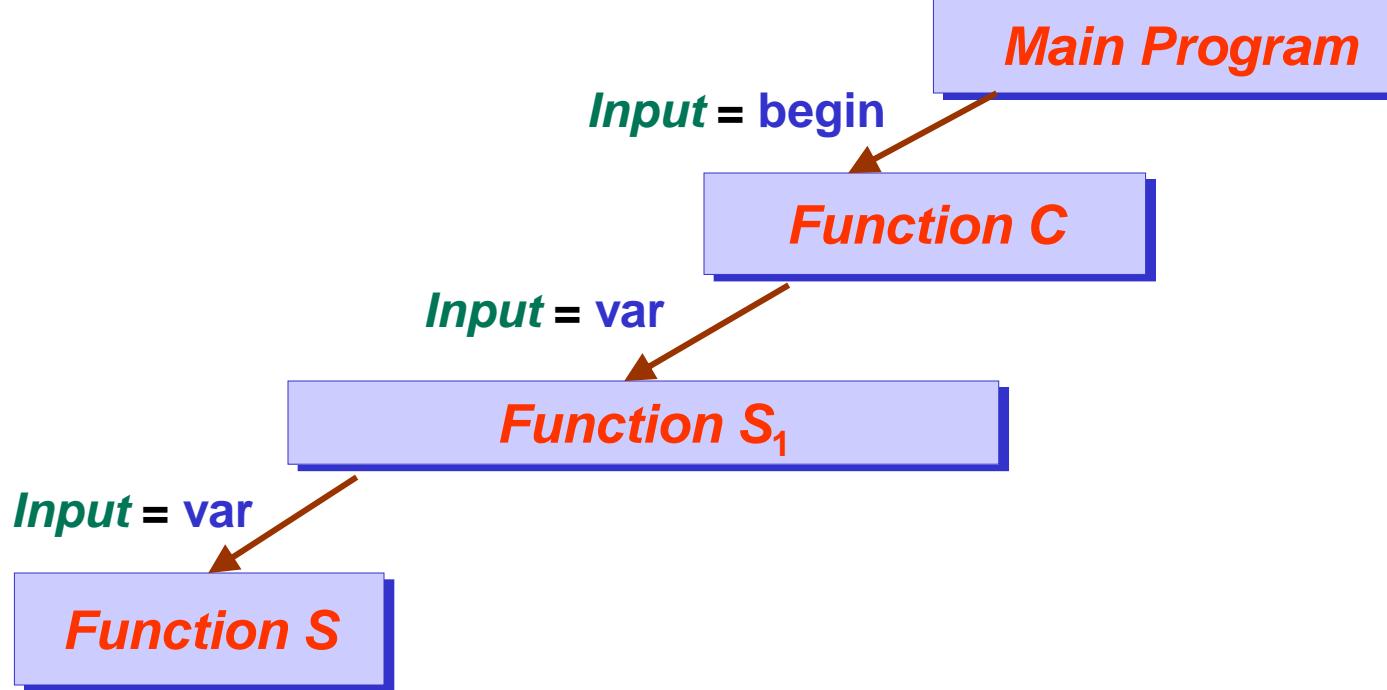
```
begin | var := var ; while var ≠ var do var := var end ⊥
```



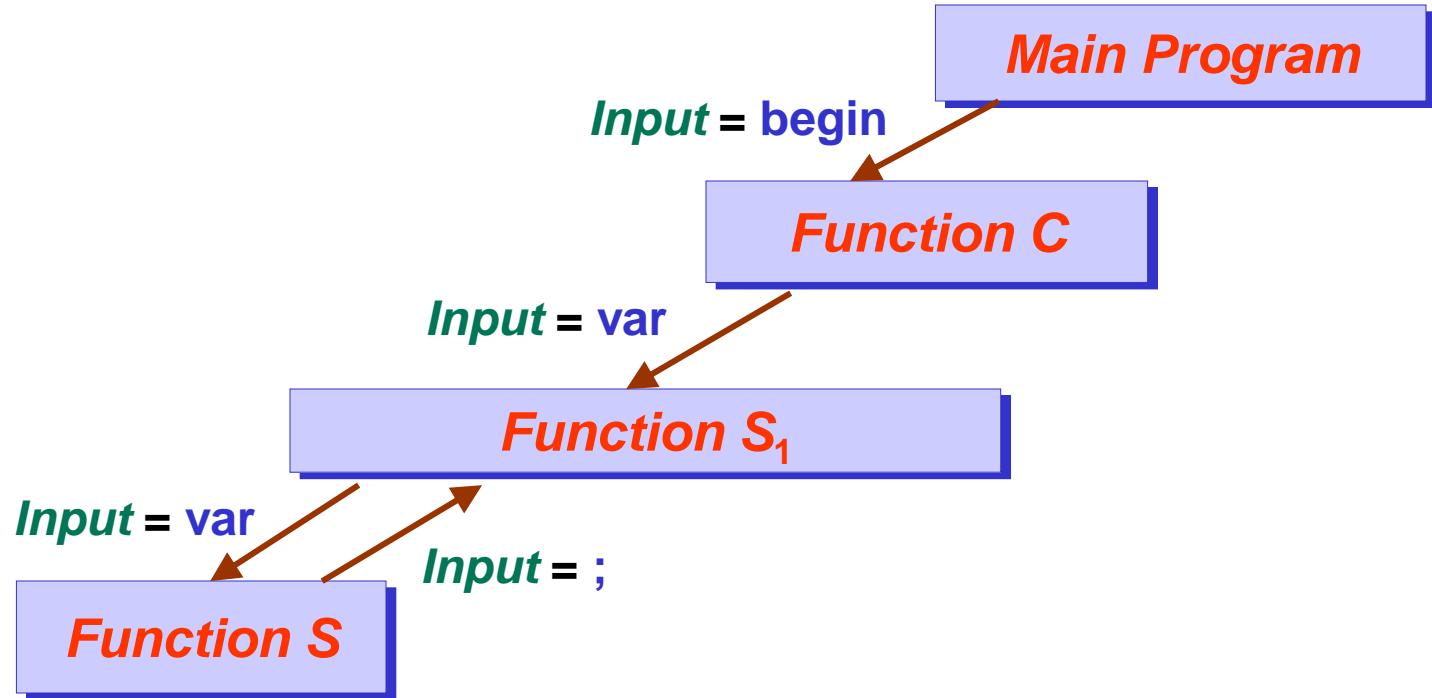
begin	var	:=	var	;	while	var	≠	var	do	var	:=	var	end	l
--------------	------------	-----------	------------	----------	--------------	------------	----------	------------	-----------	------------	-----------	------------	------------	----------



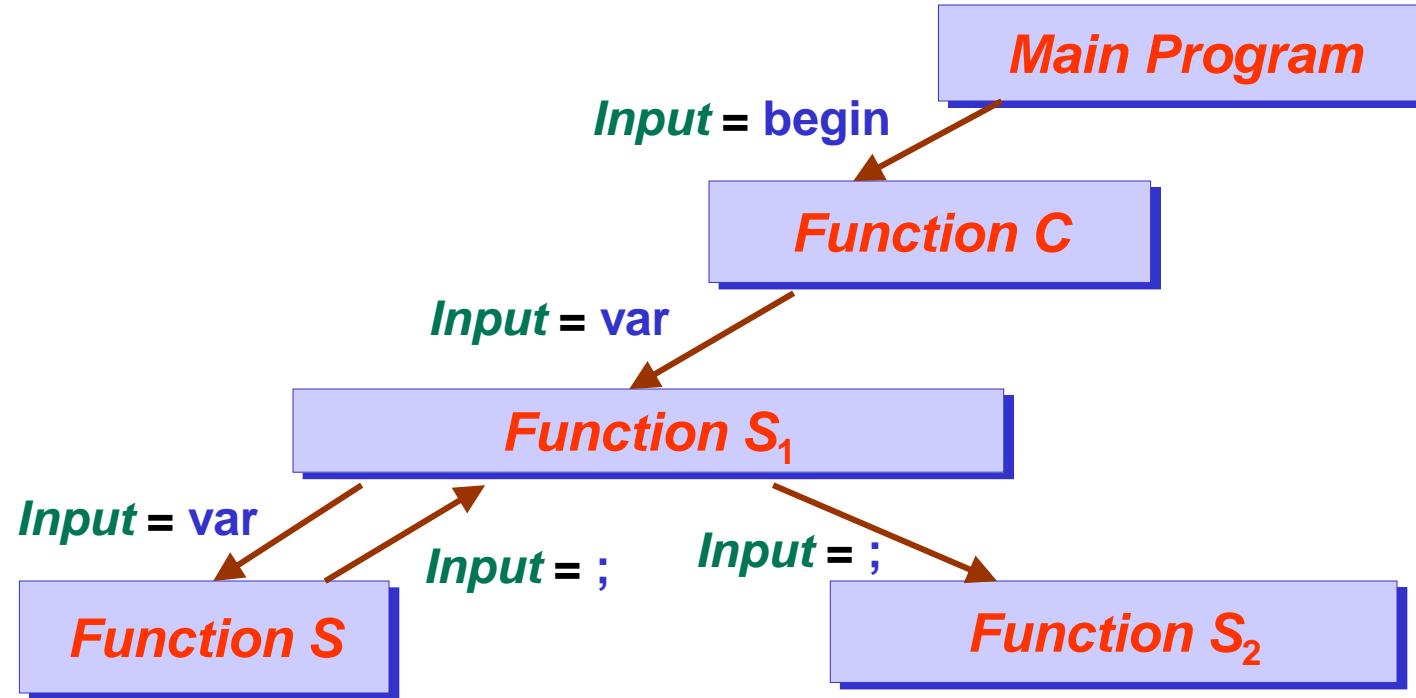
```
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥
```



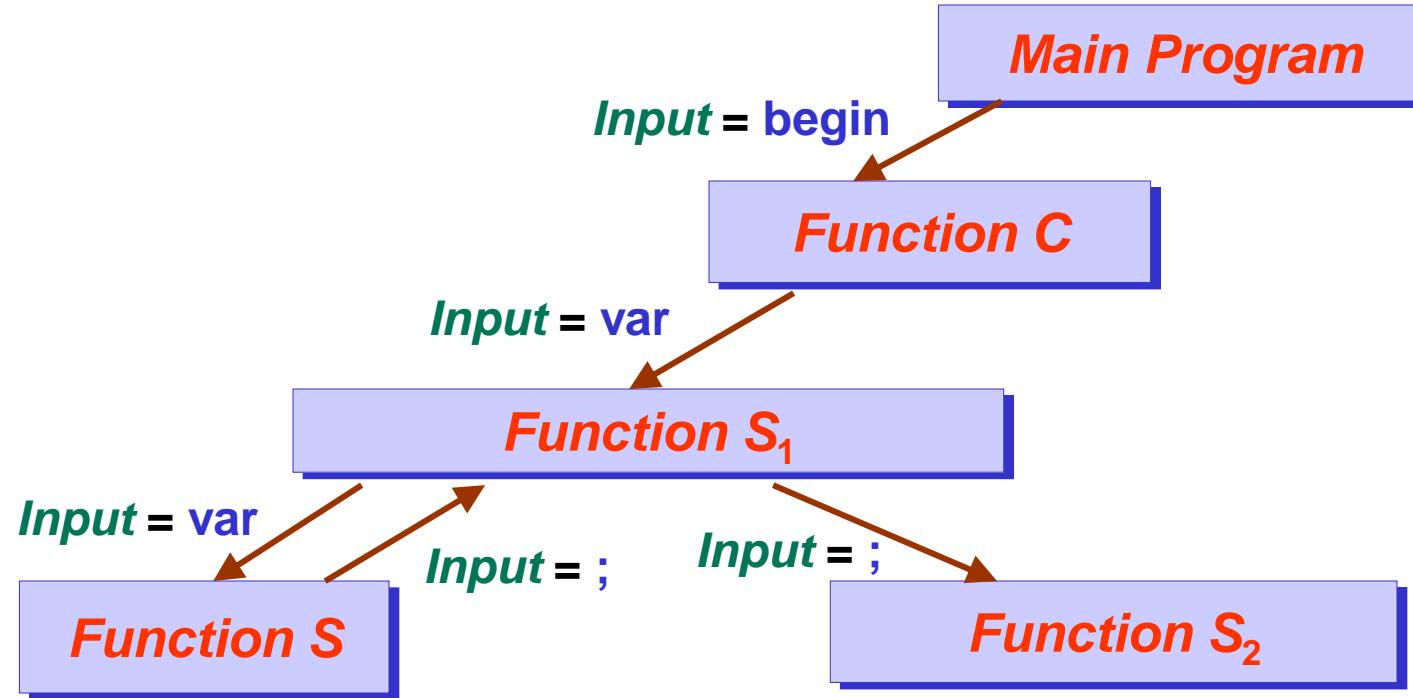
```
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥
```



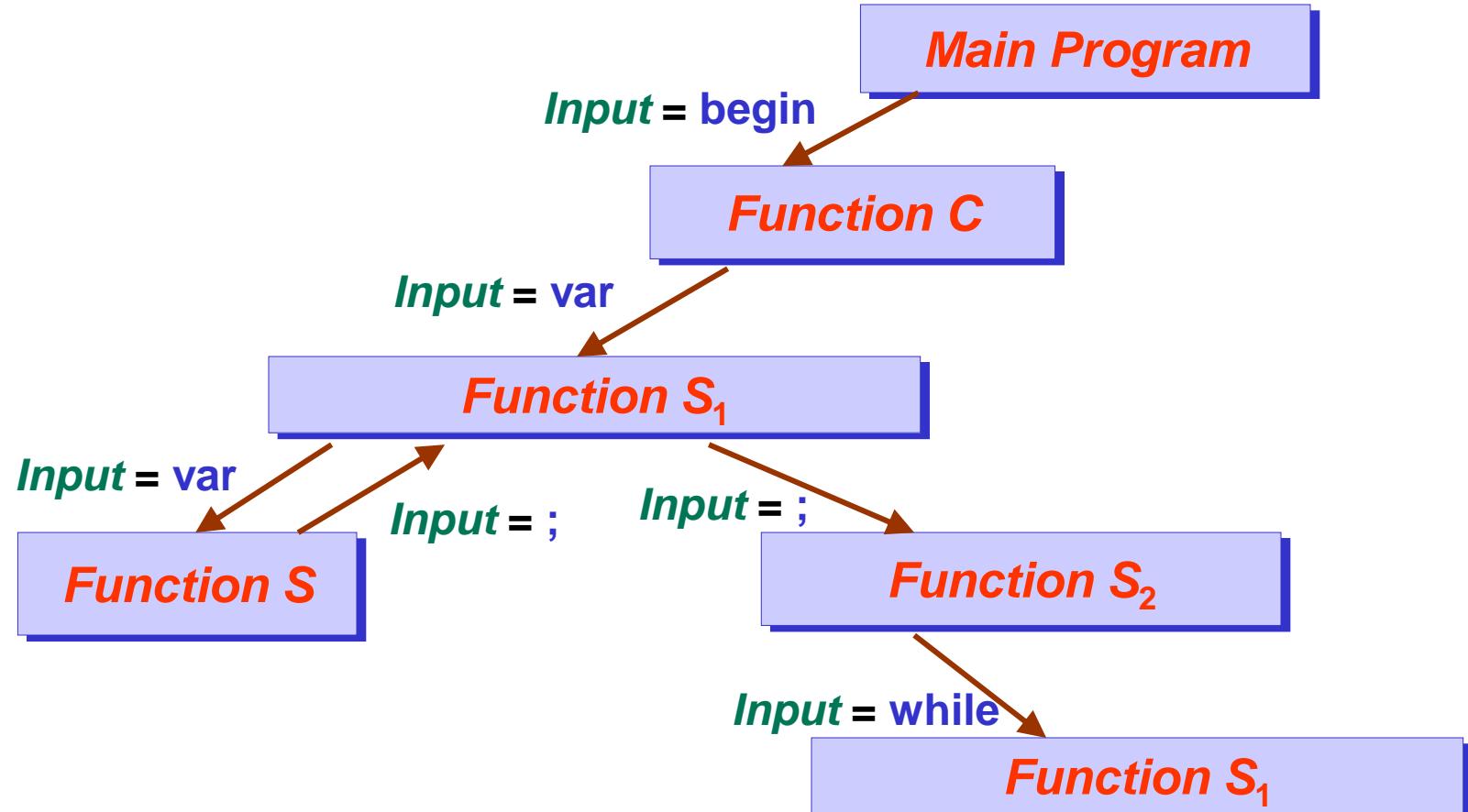
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥



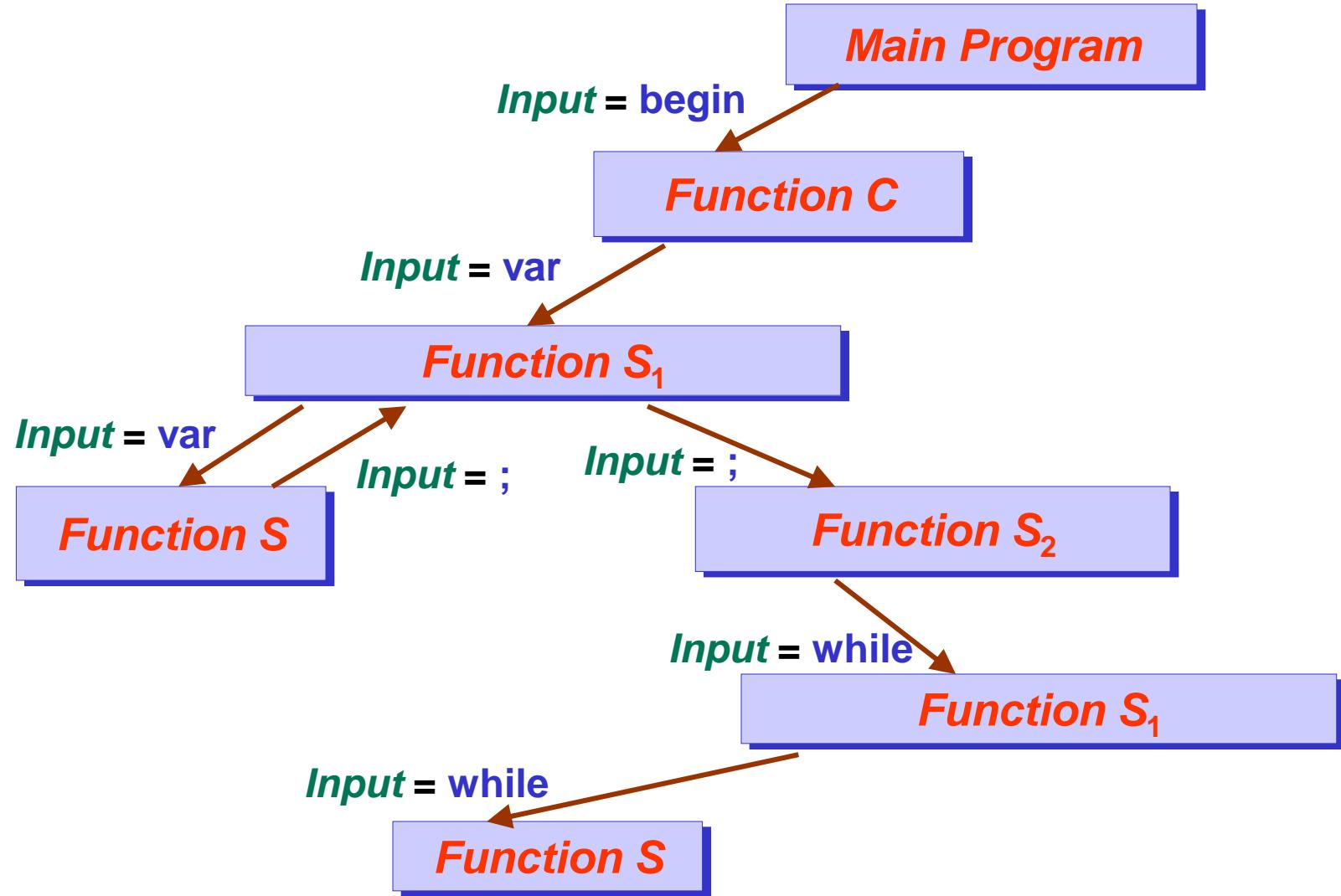
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥



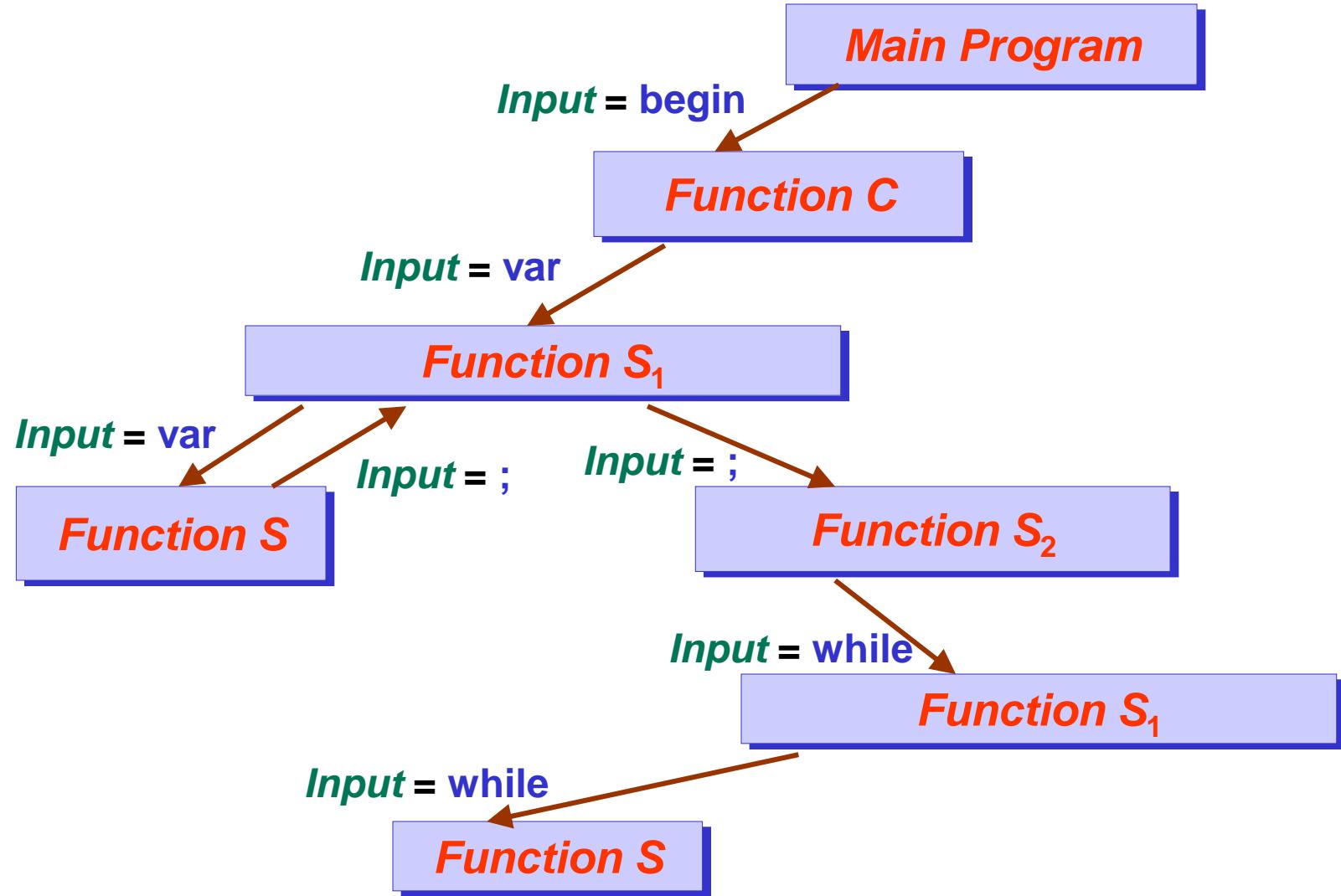
begin	var	:=	var	;	while	var	≠	var	do	var	:=	var	end	⊥
--------------	------------	-----------	------------	----------	--------------	------------	----------	------------	-----------	------------	-----------	------------	------------	----------



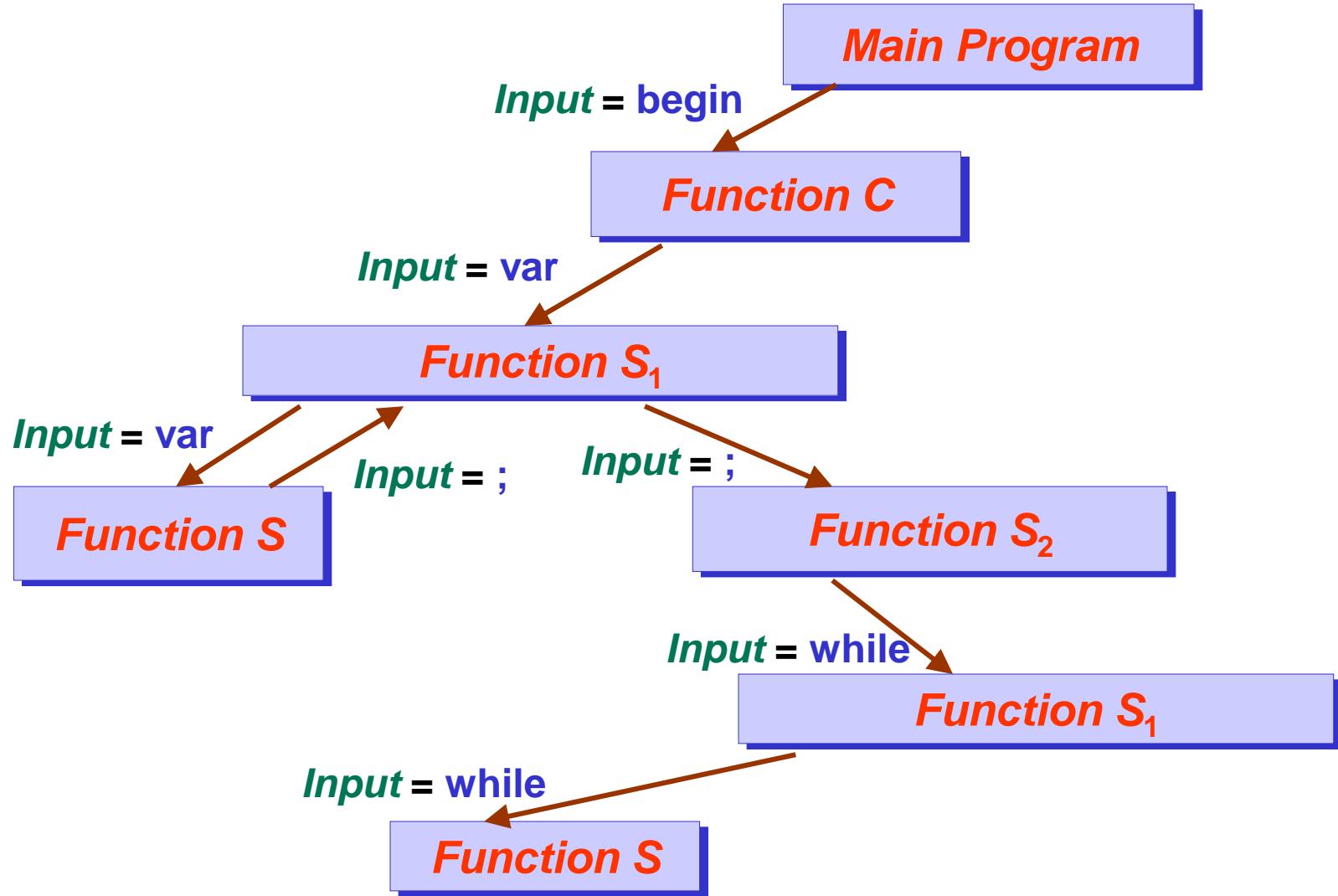
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥



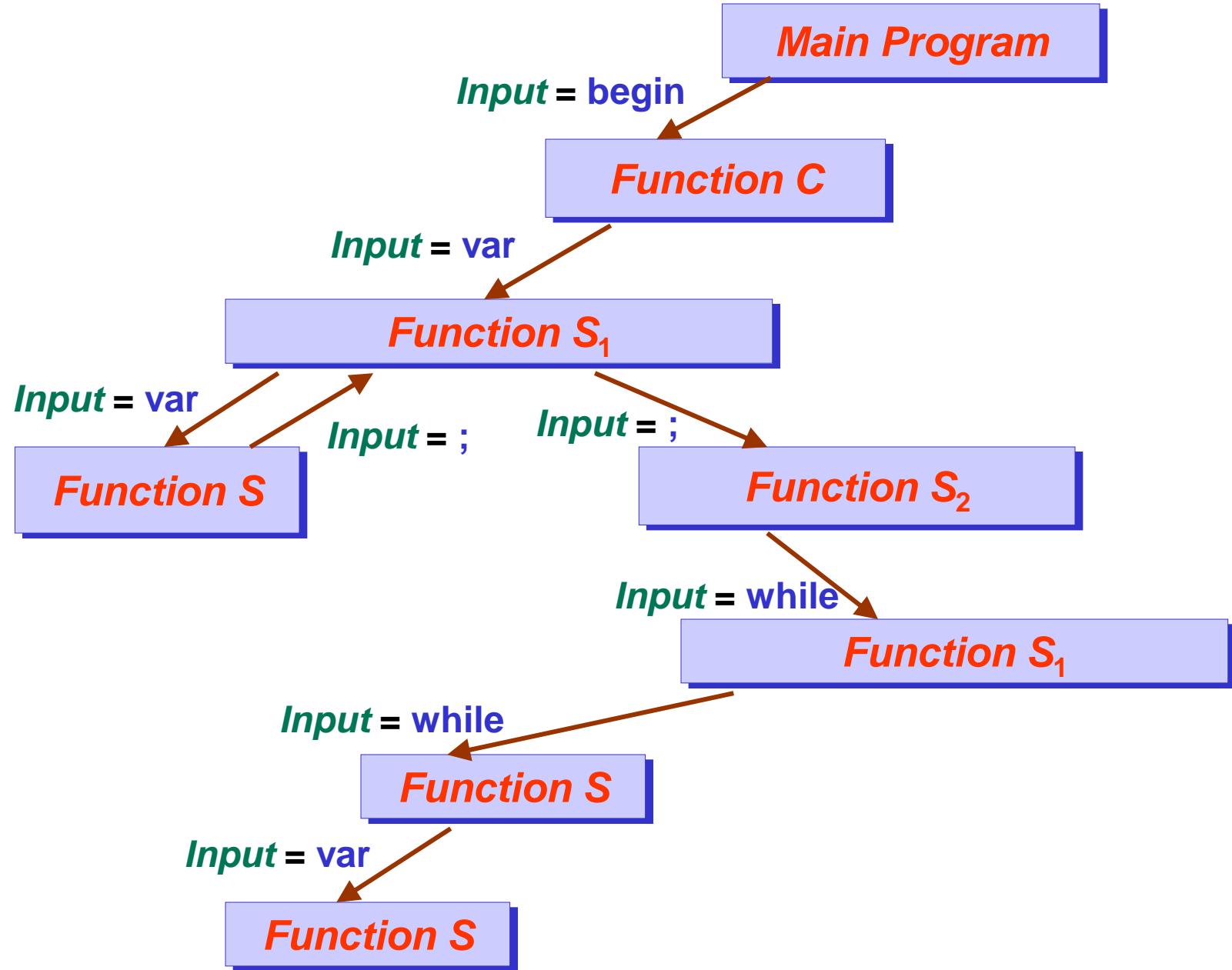
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥



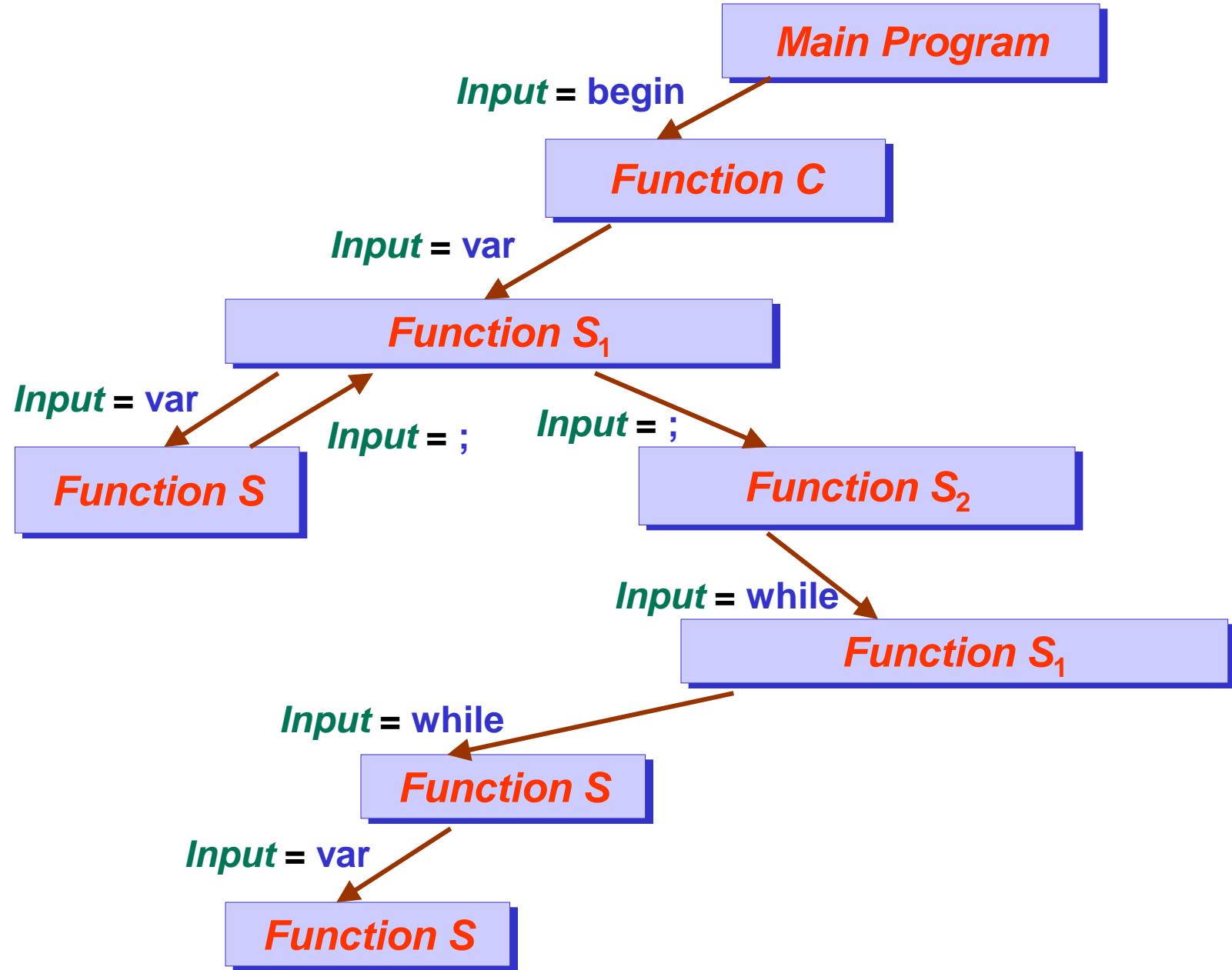
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥



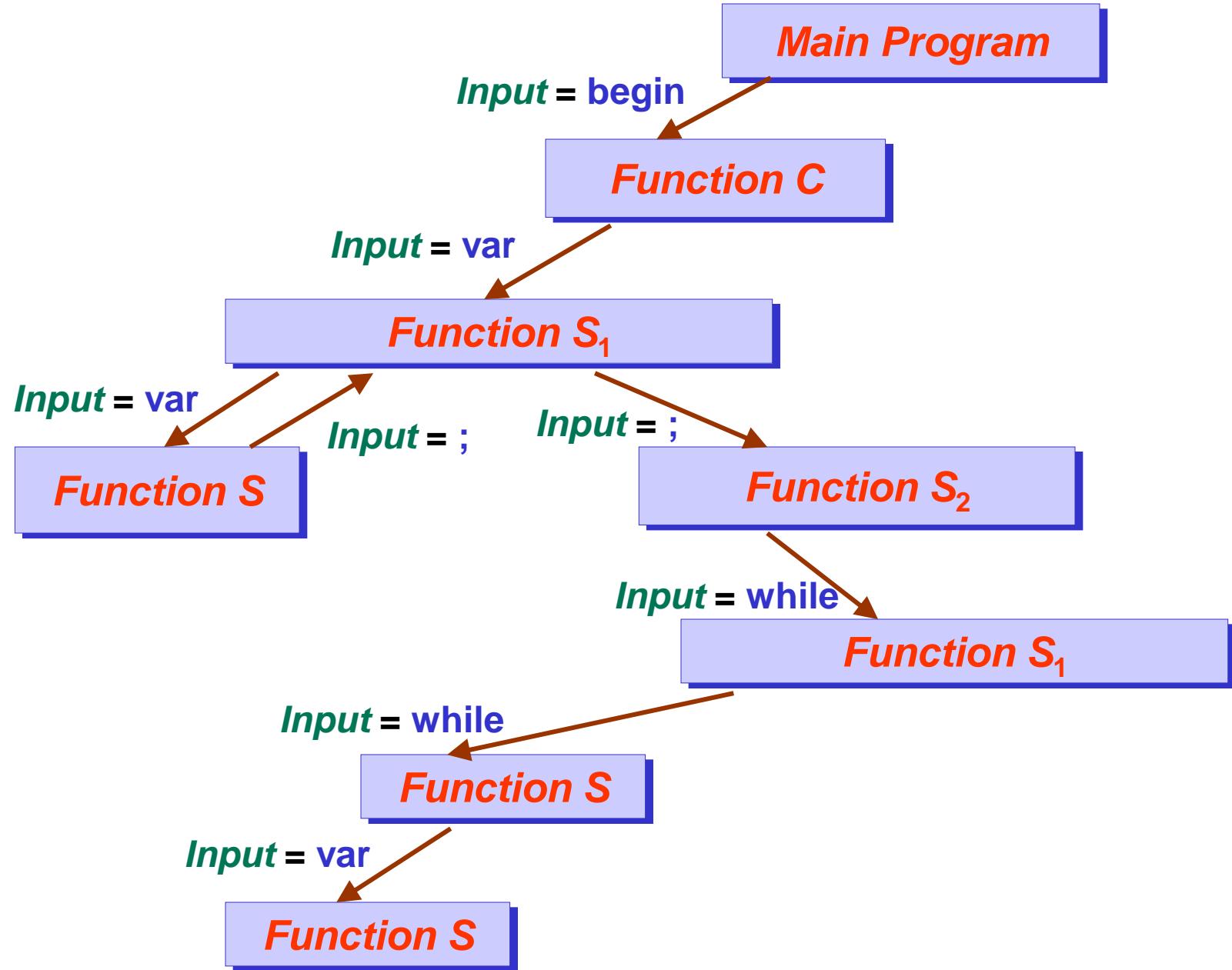
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥



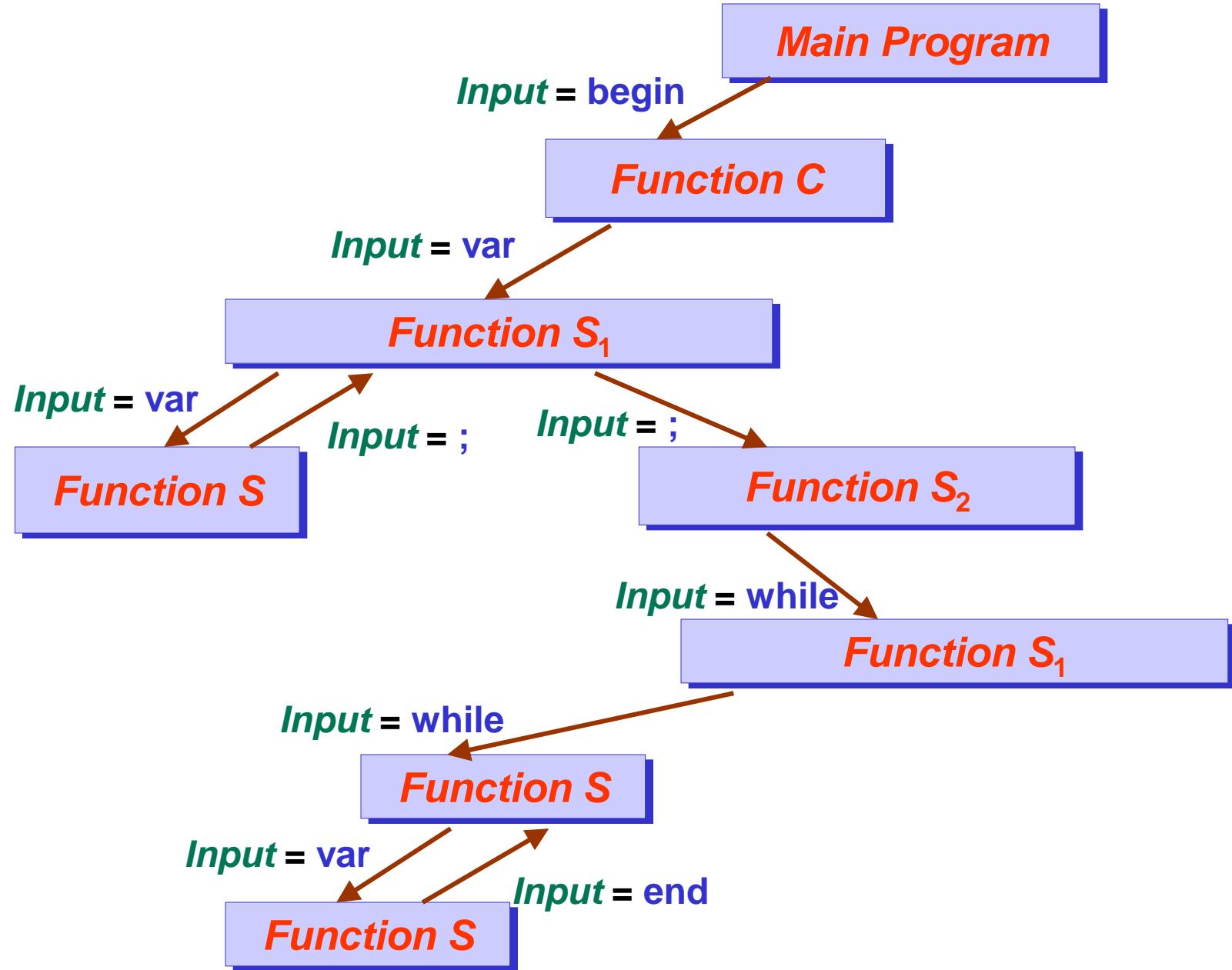
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥



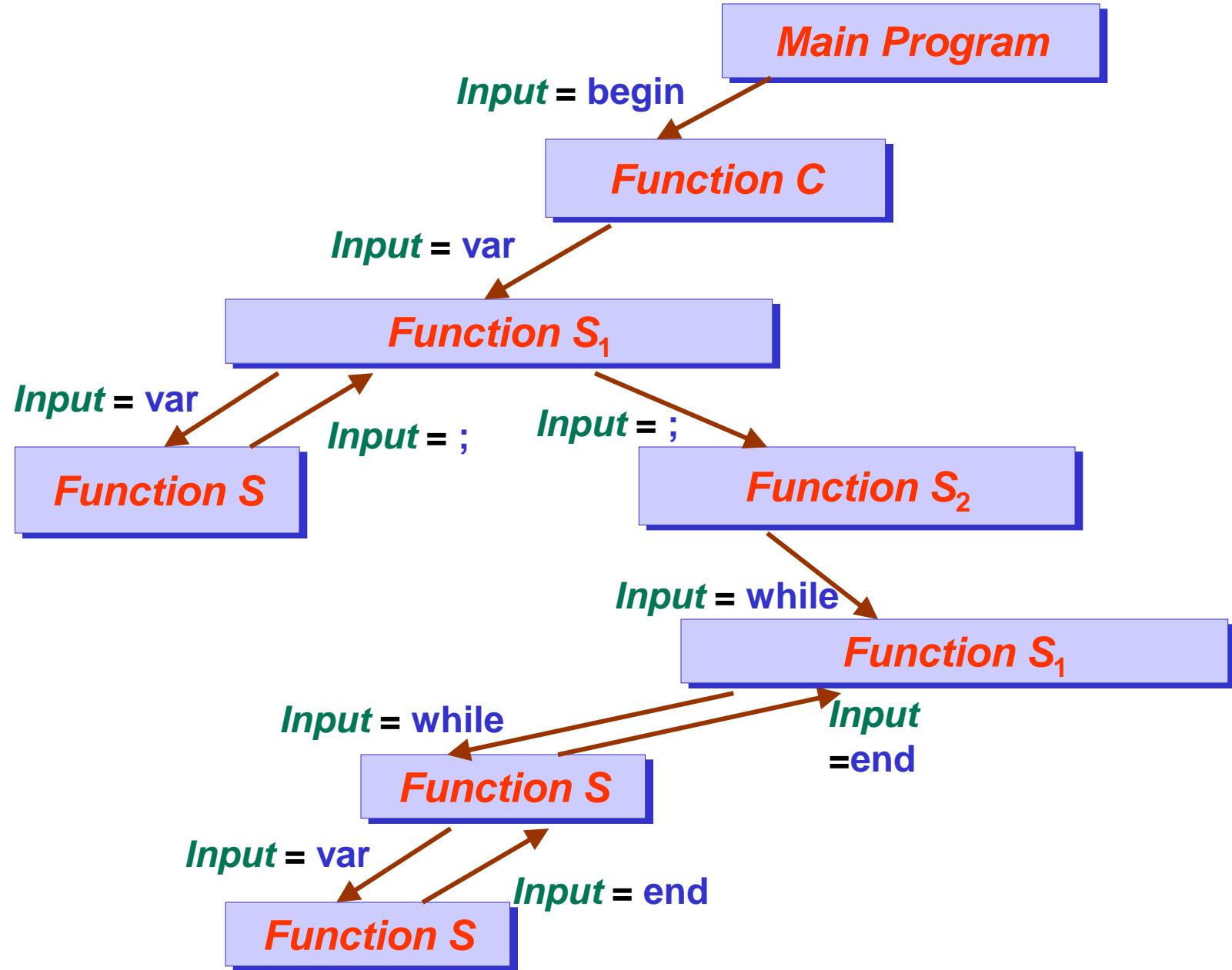
begin	var	\vdash	end	\perp										
-------	-----	----------	-----	----------	-----	----------	-----	----------	-----	----------	-----	----------	-----	---------



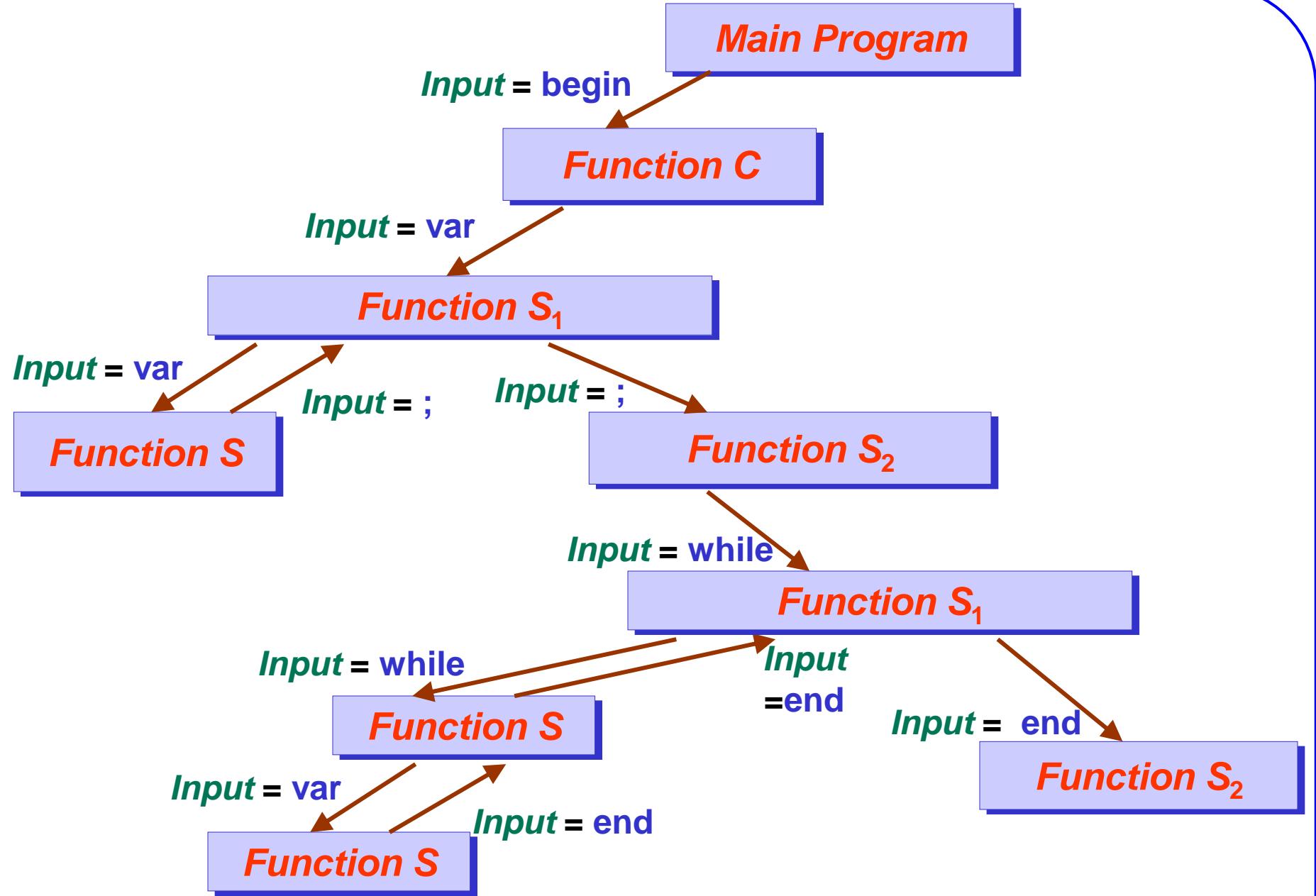
begin	var	\vdash var	;	while	var	\neq	var	do	var	\vdash var	end	\perp
-------	-----	--------------	---	-------	-----	--------	-----	----	-----	--------------	-----	---------



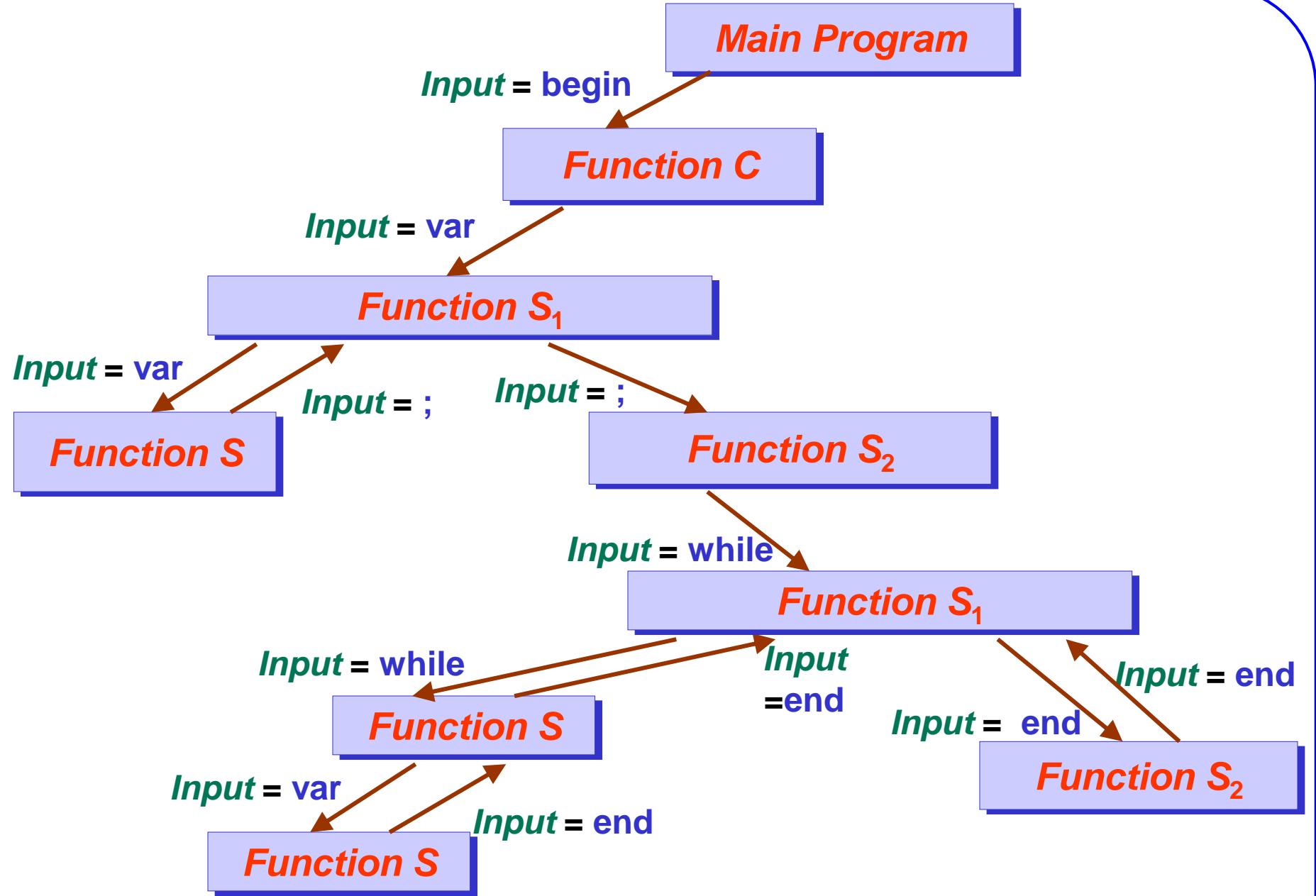
begin	var	\vdash var	;	while	var	\neq	var	do	var	\vdash var	end	\perp
-------	-----	--------------	---	-------	-----	--------	-----	----	-----	--------------	-----	---------



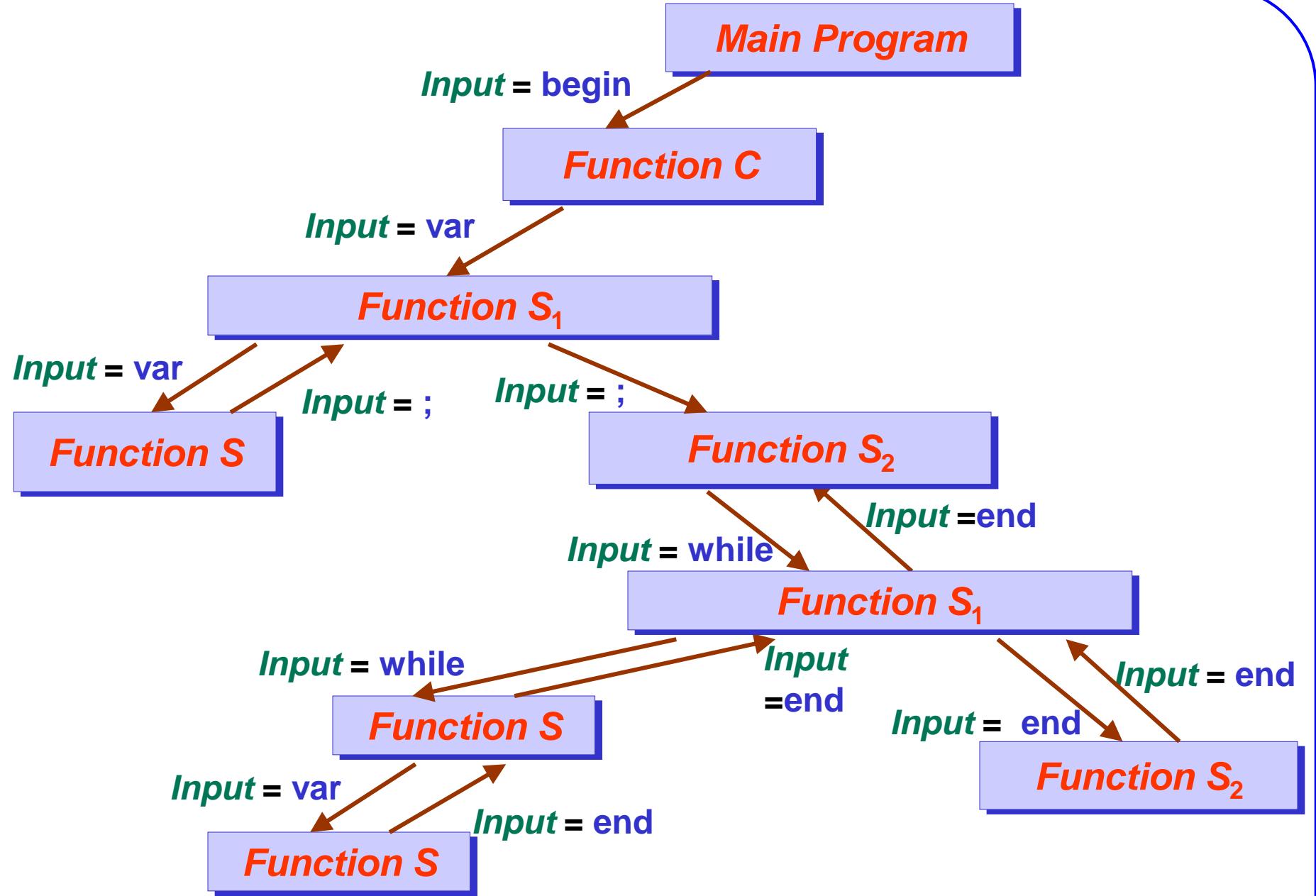
begin	var	\vdash var	;	while	var	\neq	var	do	var	\vdash var	end	\perp
-------	-----	--------------	---	-------	-----	--------	-----	----	-----	--------------	-----	---------



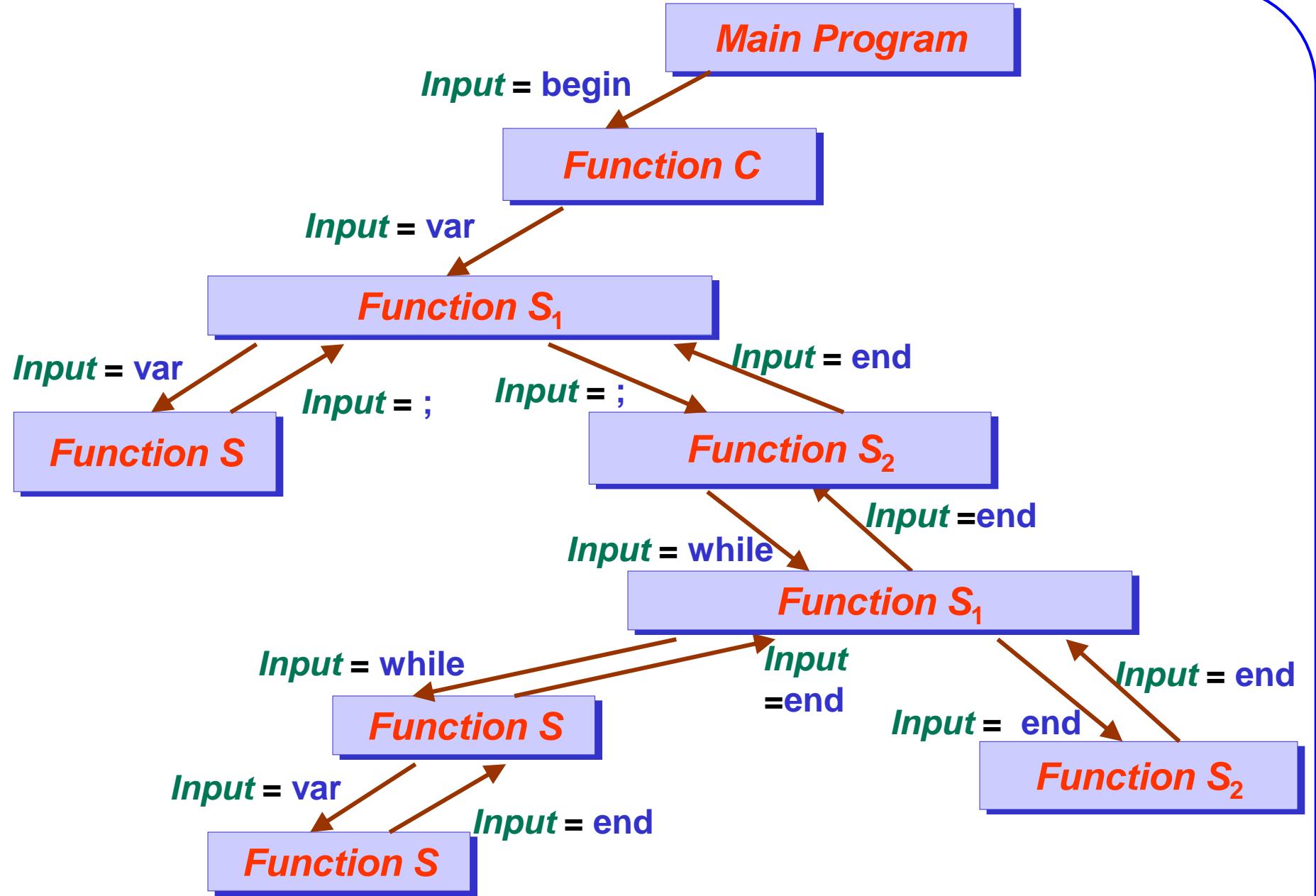
begin	var	\vdash var	;	while	var	\neq	var	do	var	\vdash var	end	1
-------	-----	--------------	---	-------	-----	--------	-----	----	-----	--------------	-----	---



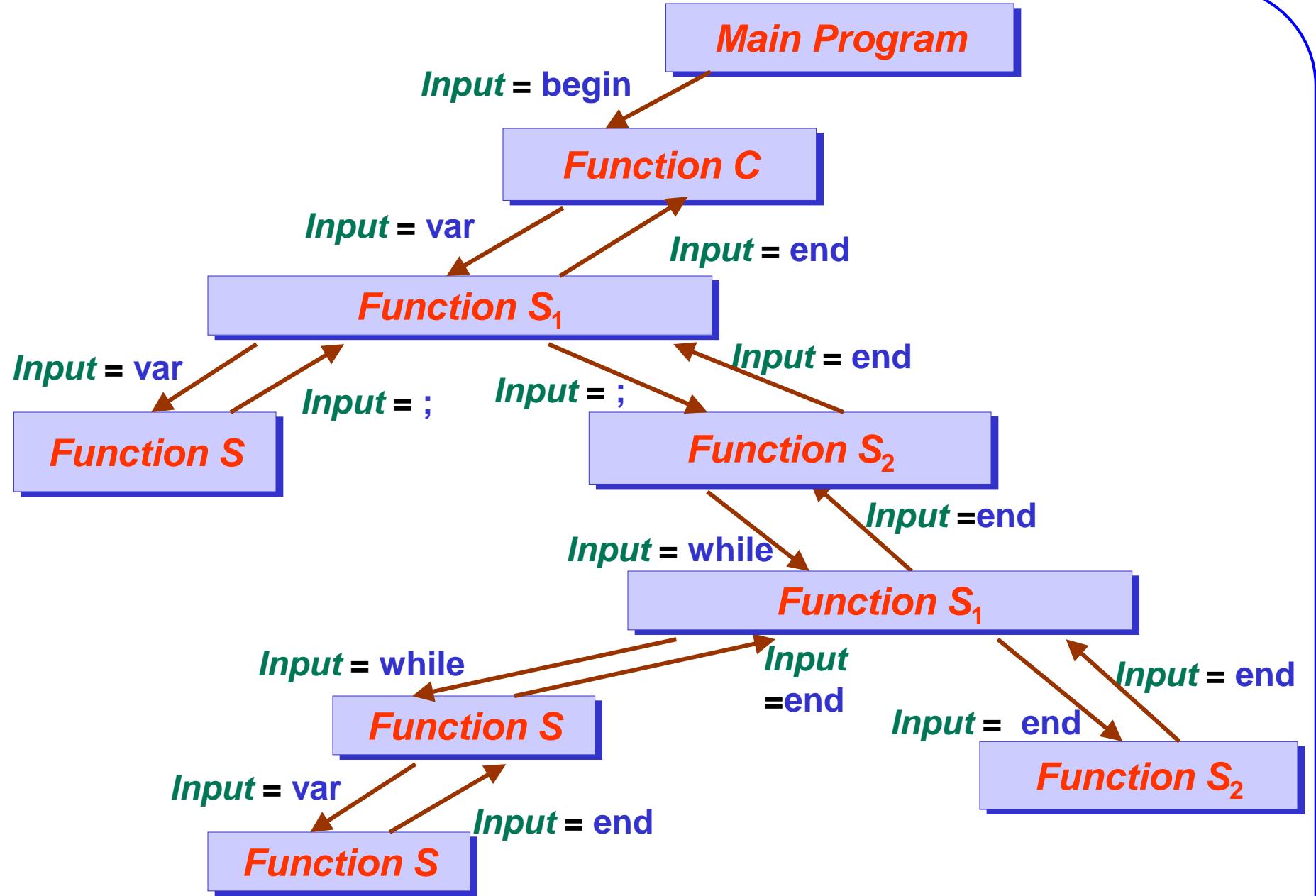
begin	var	\vdash var	;	while	var	\neq	var	do	var	\vdash var	end	\perp
-------	-----	--------------	---	-------	-----	--------	-----	----	-----	--------------	-----	---------



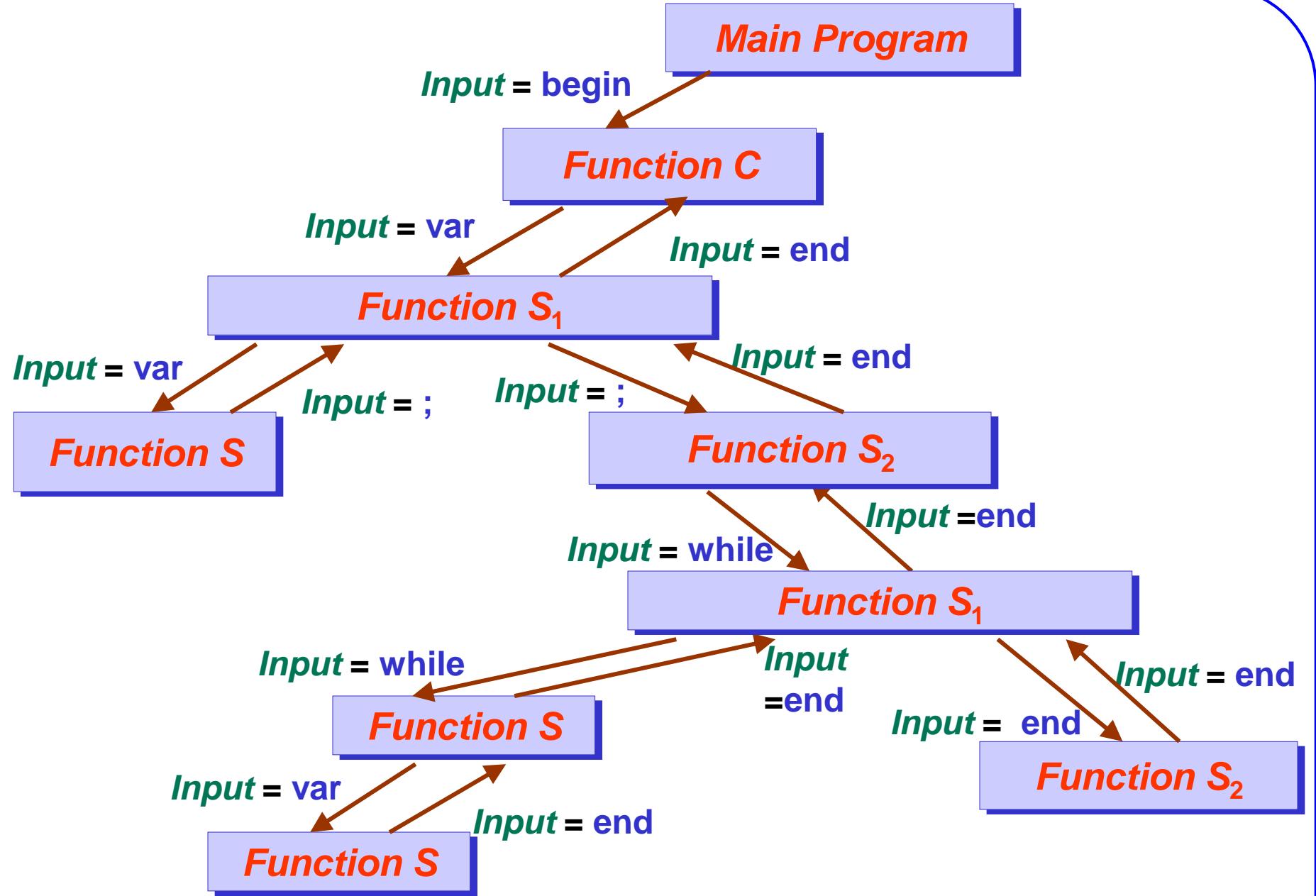
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥



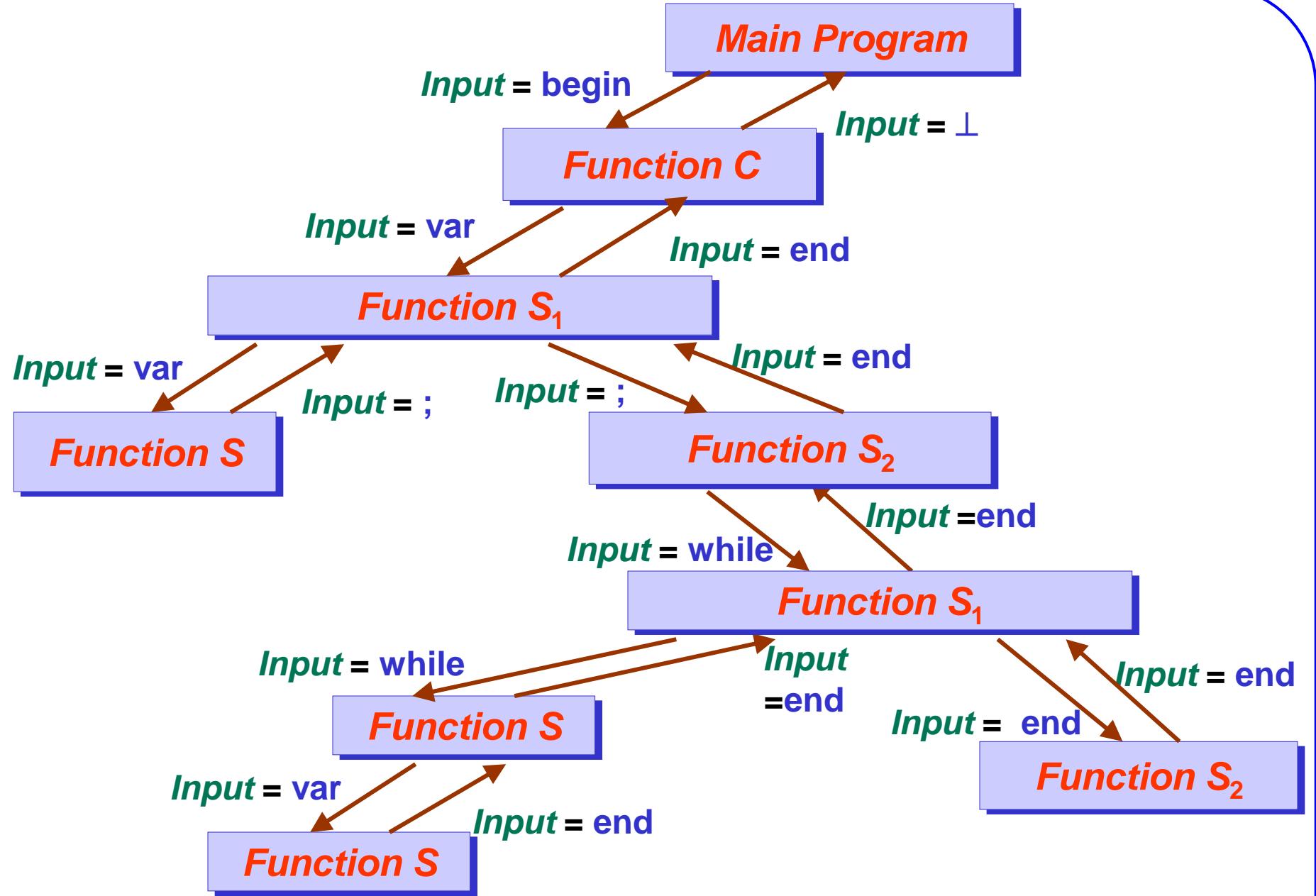
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥



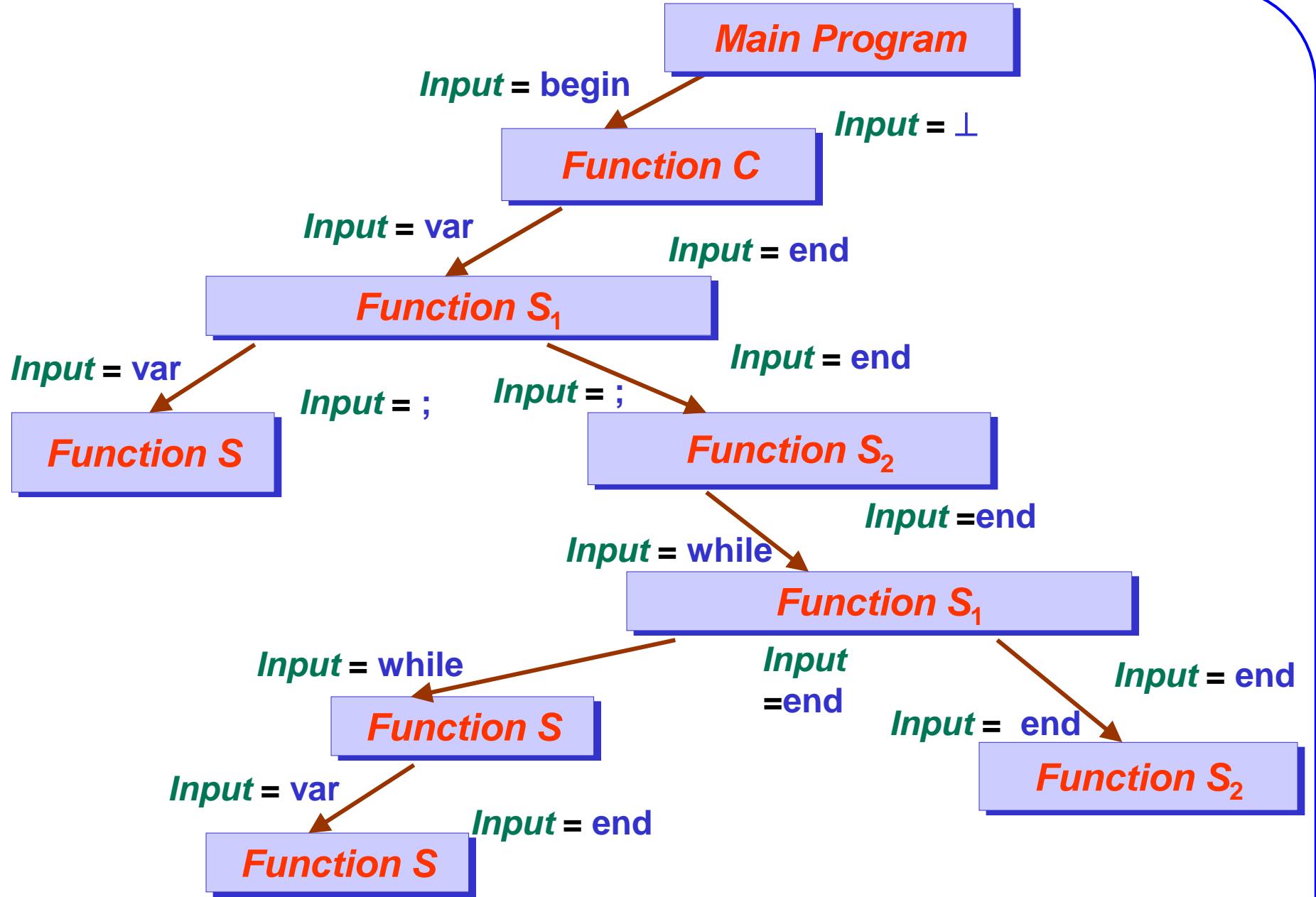
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥



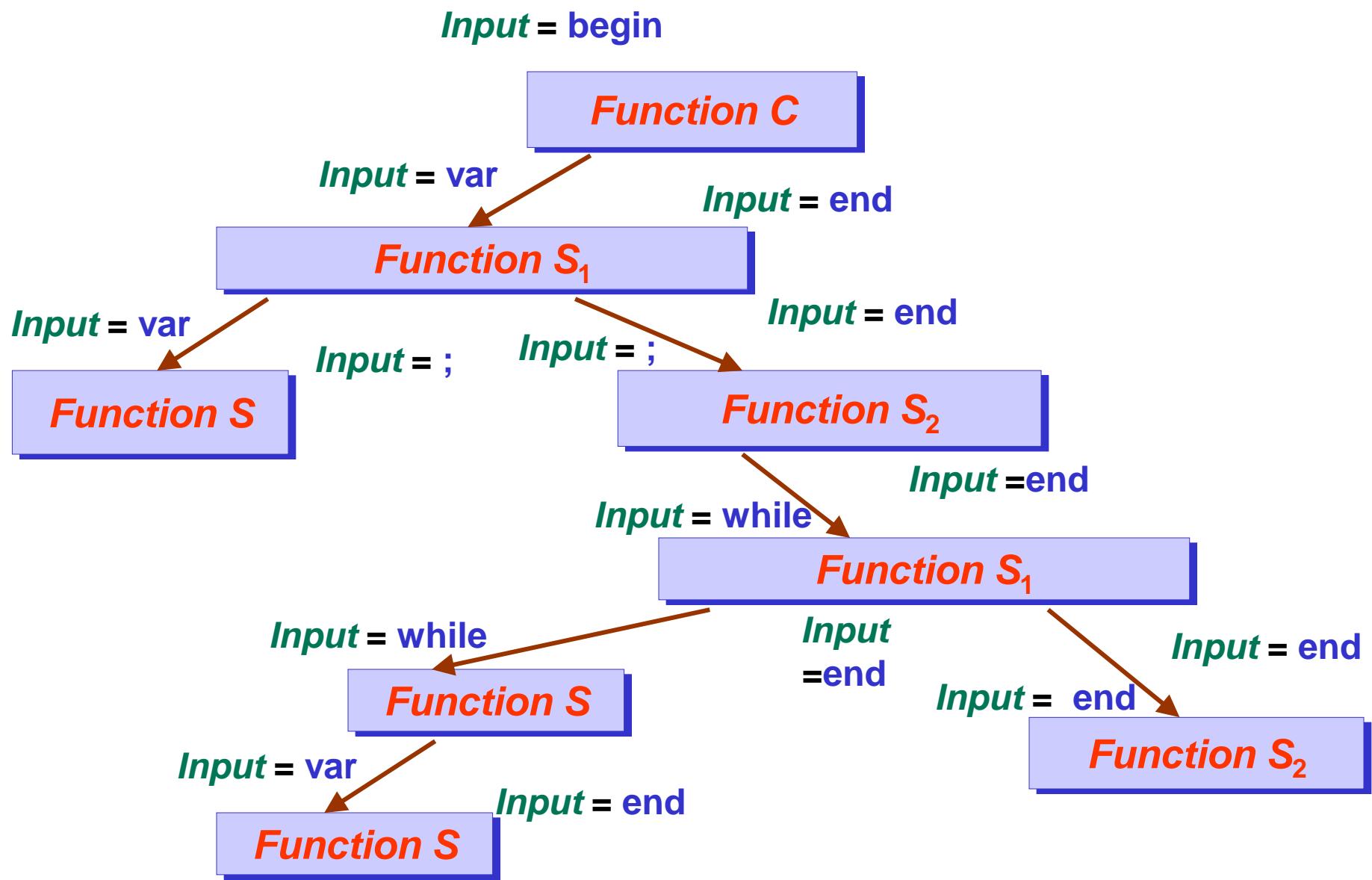
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥



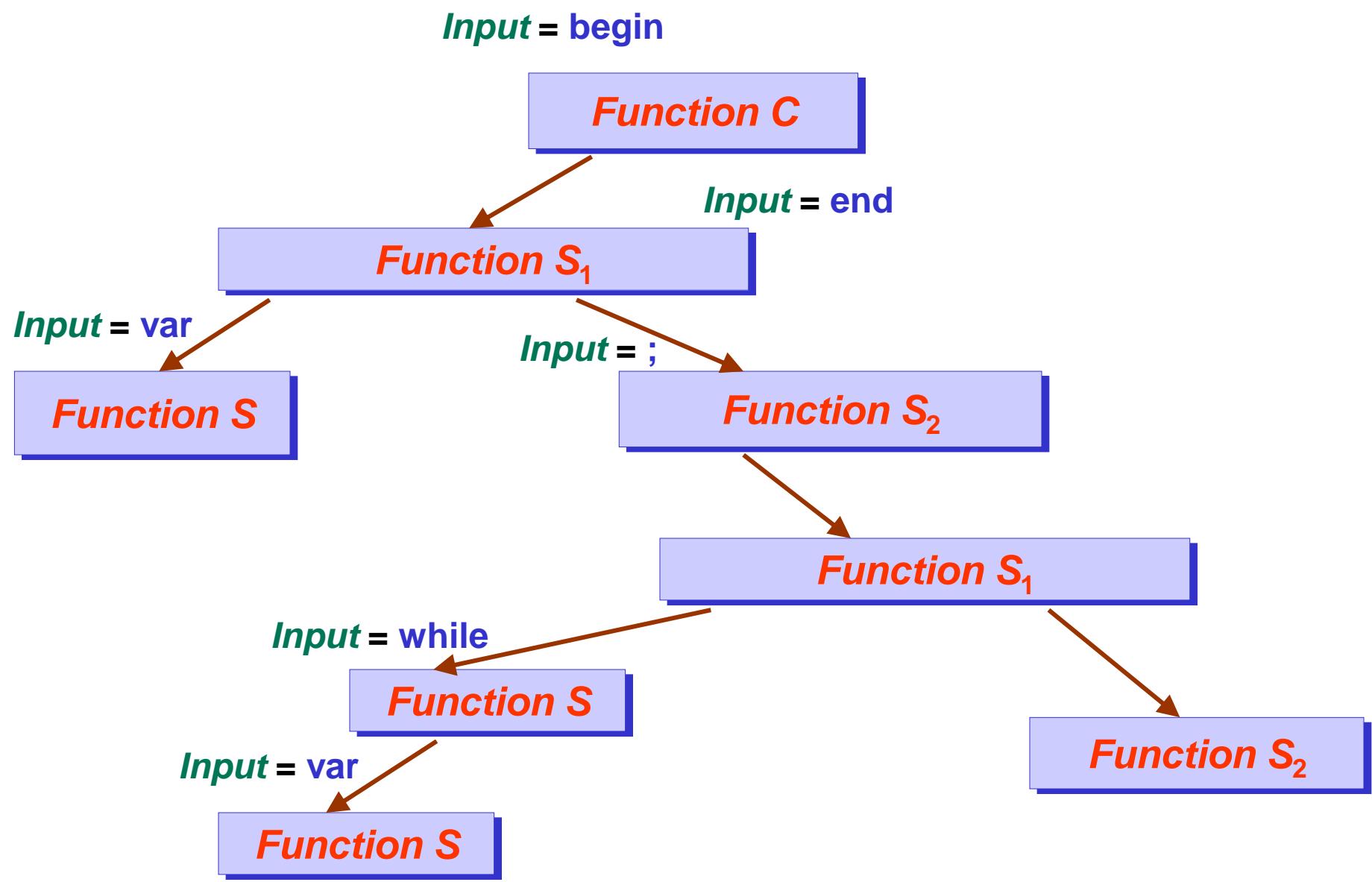
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥



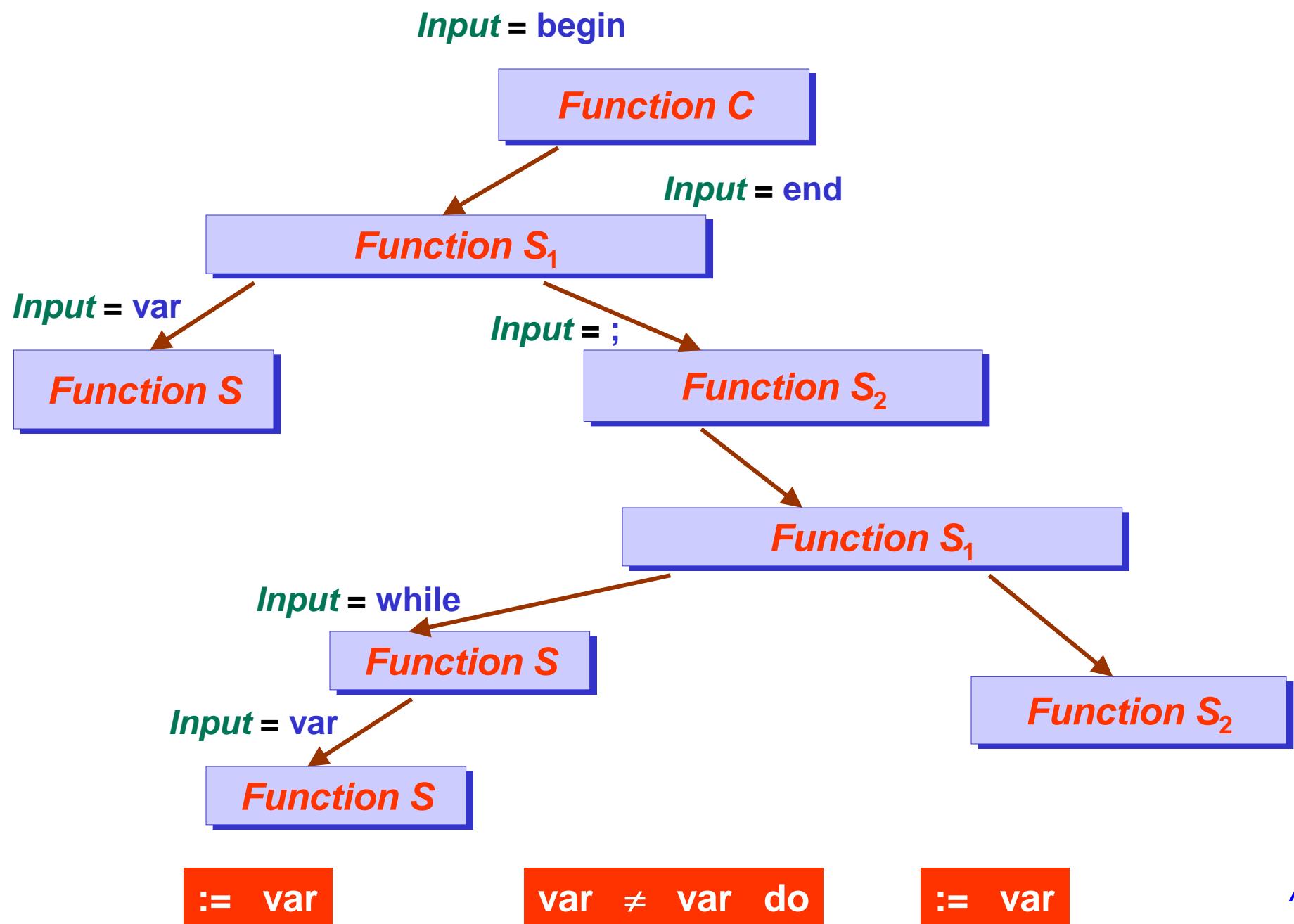
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥

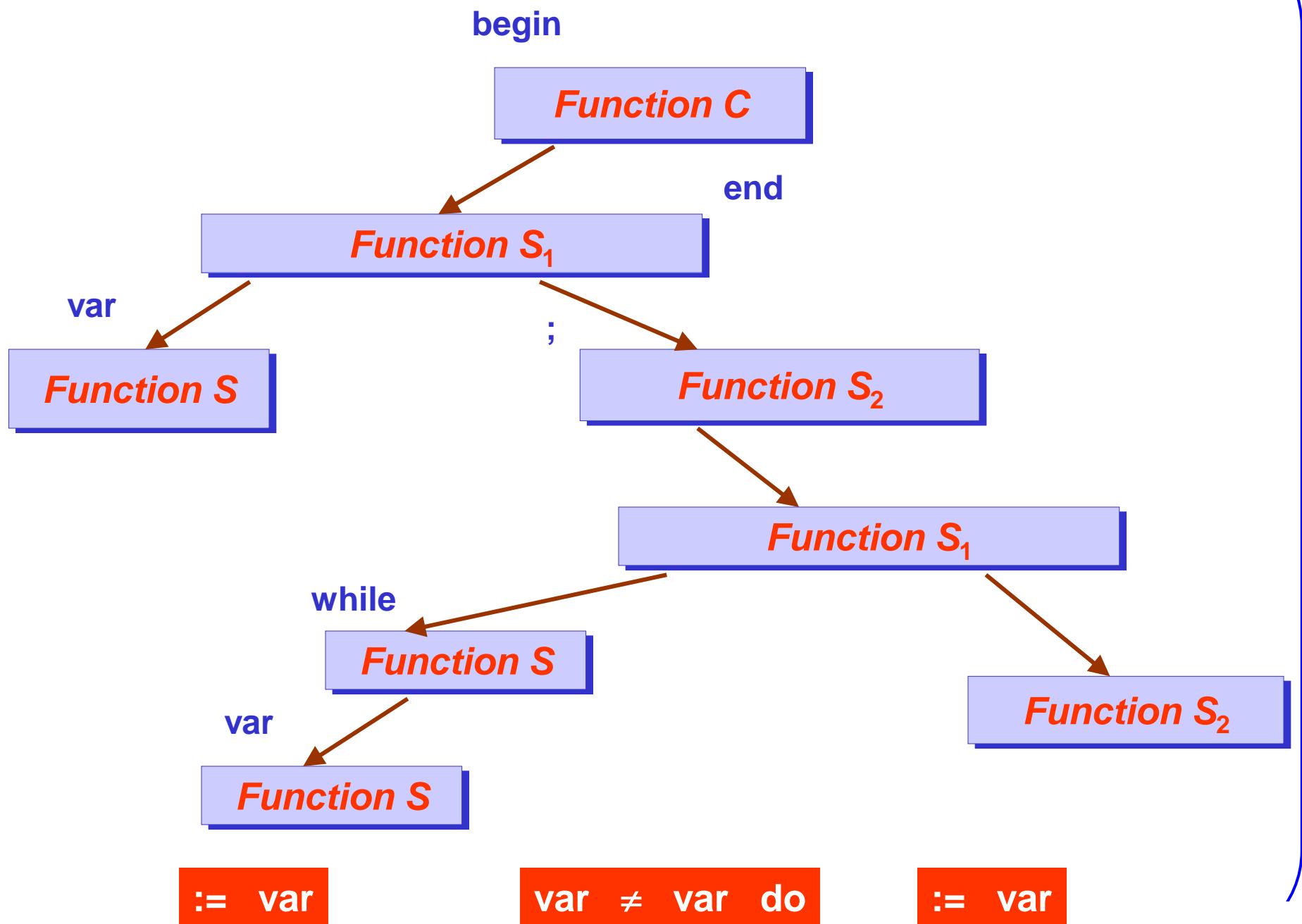


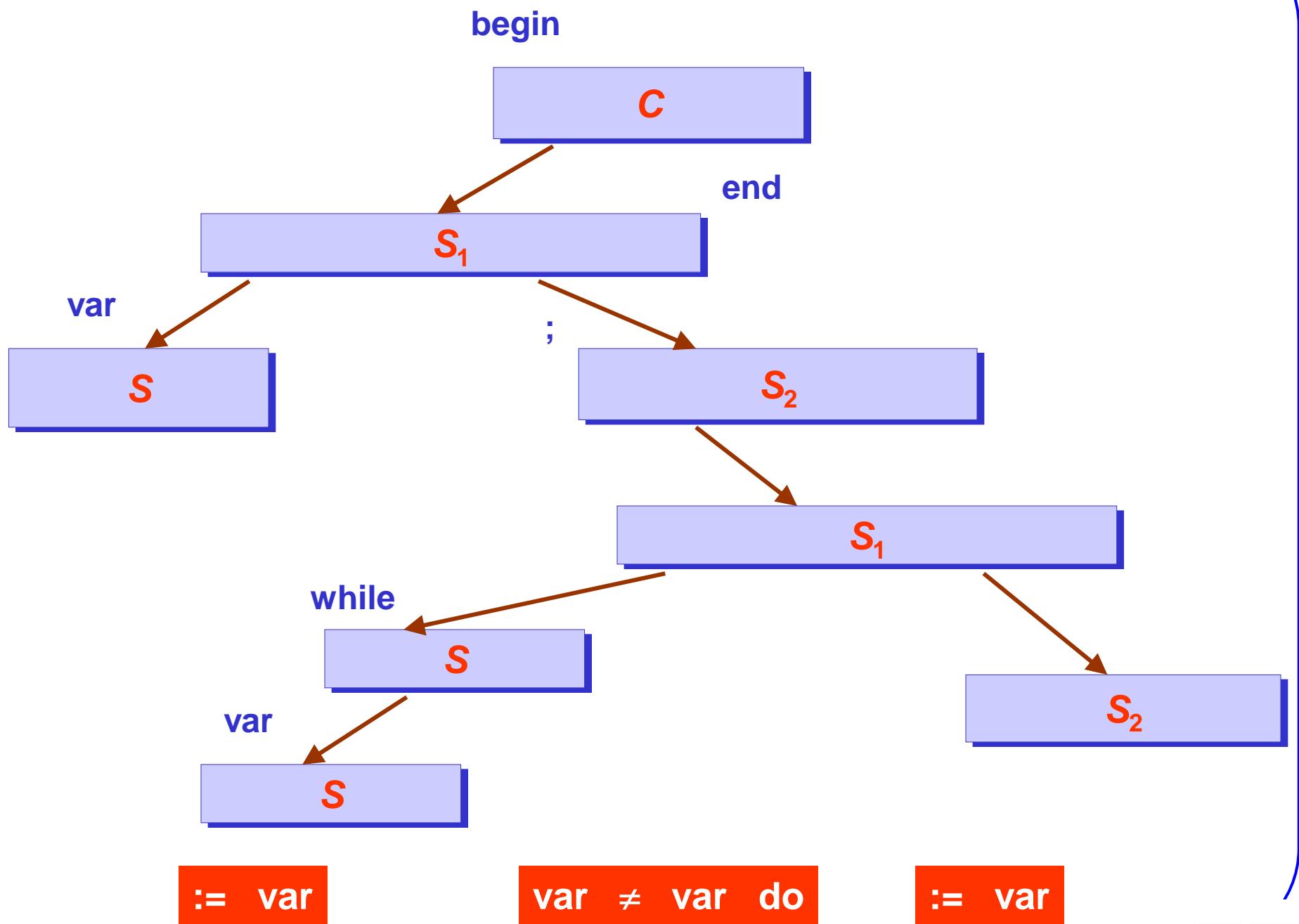
begin | var | := | var | ; | while | var | ≠ | var | do | var | := | var | end | ⊥

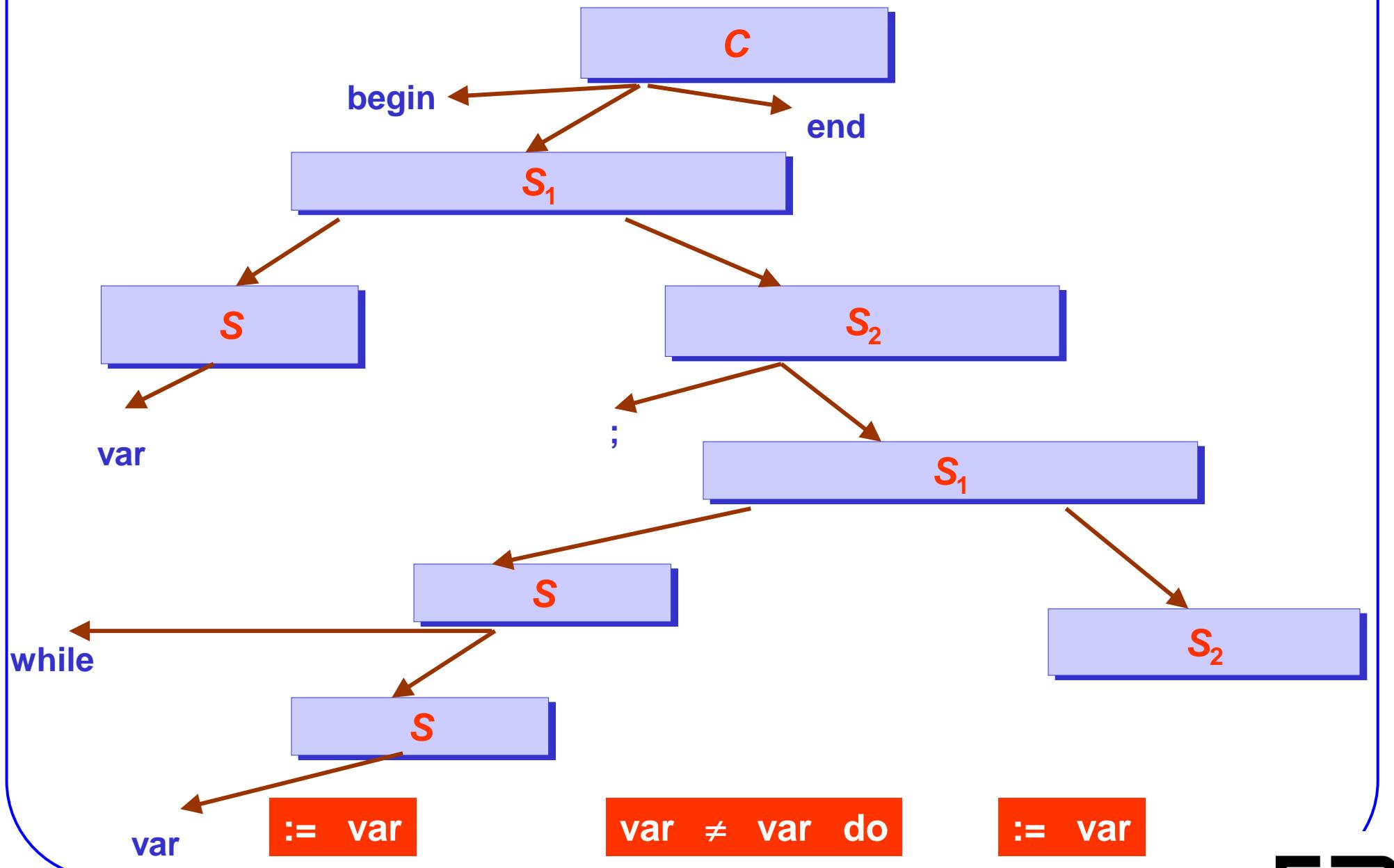


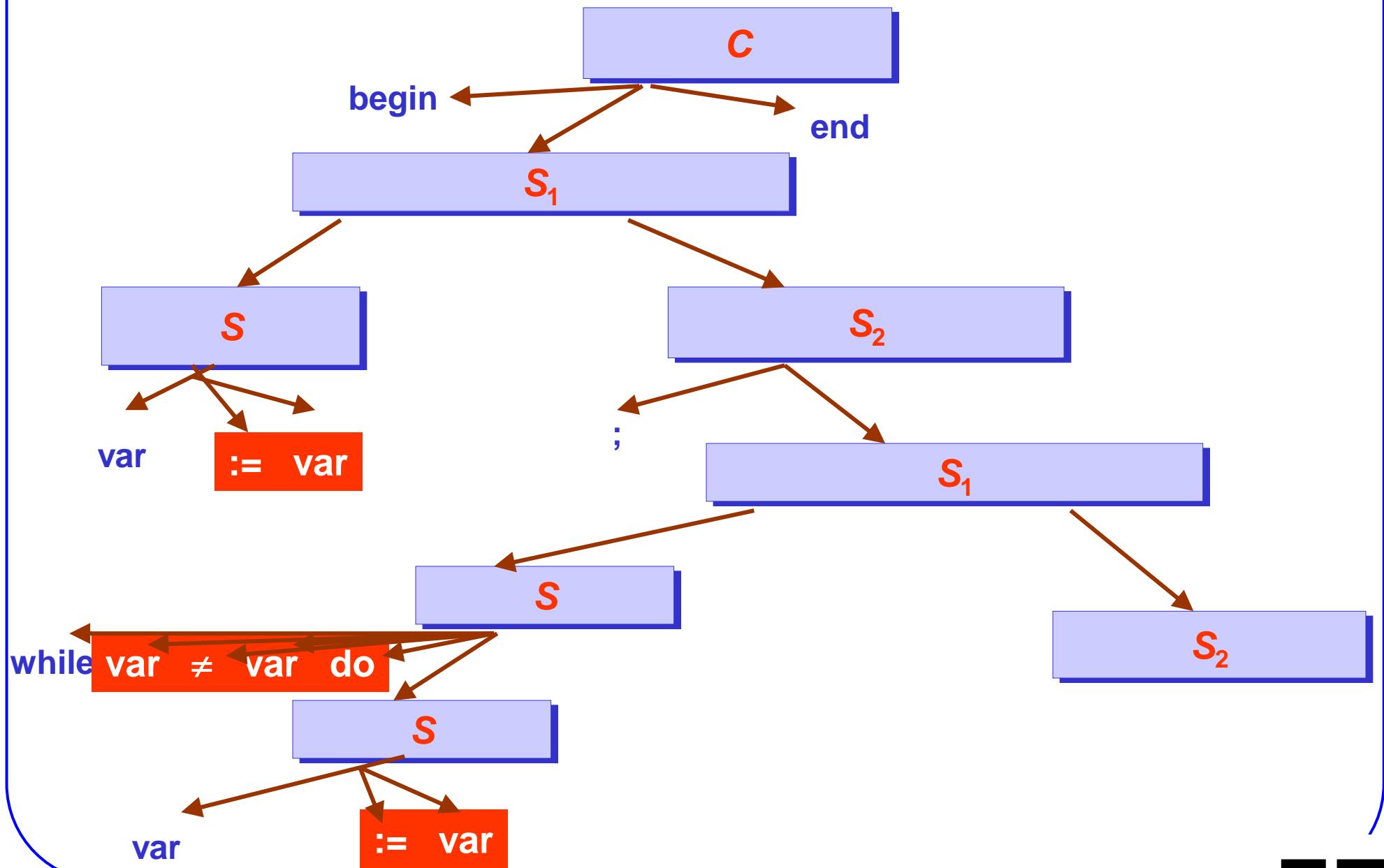
begin	var	:=	var	;	while	var	≠	var	do	var	:=	var	end	\perp
-------	-----	-----------	-----	----------	-------	-----	----------	-----	----	-----	-----------	-----	-----	---------

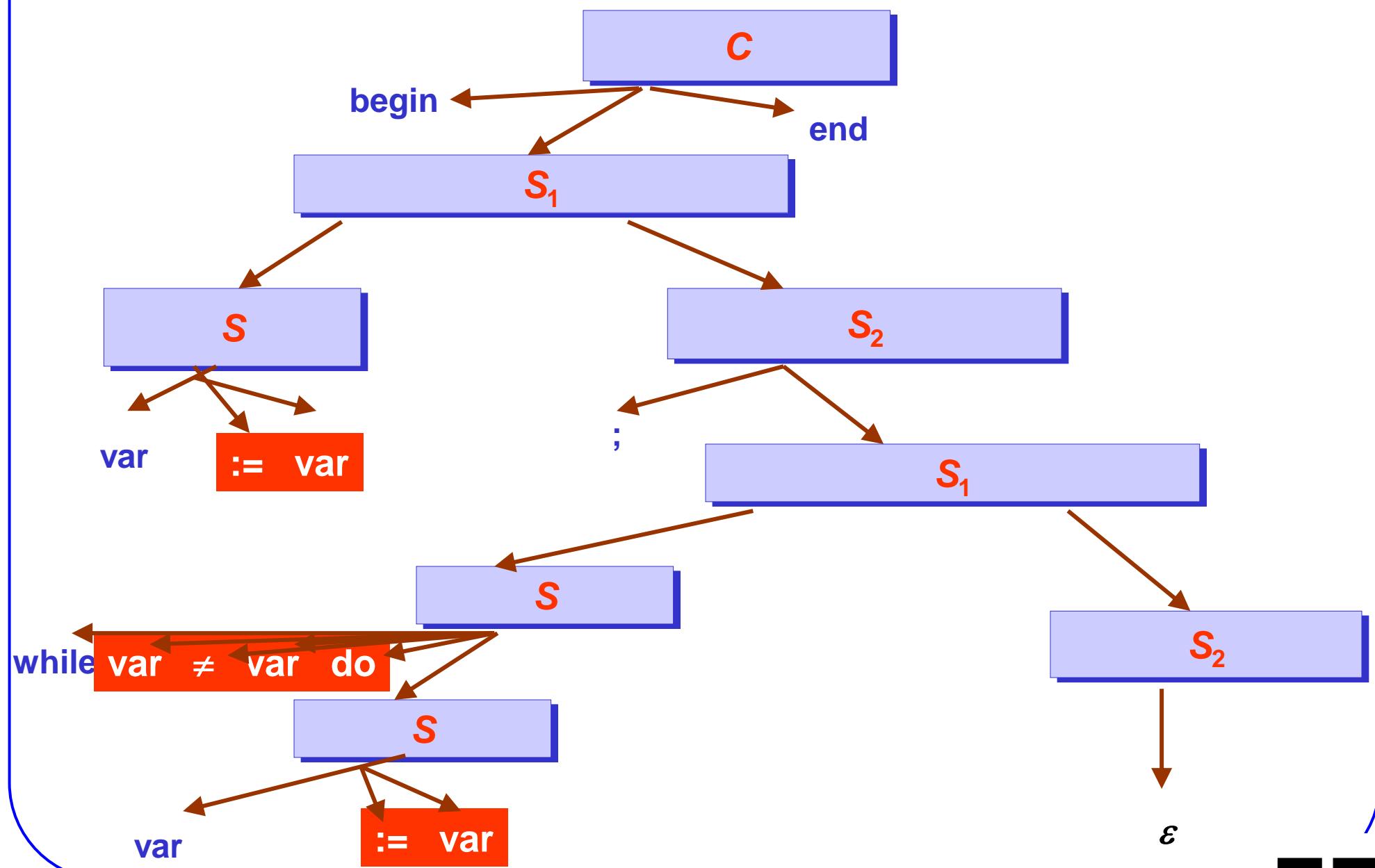


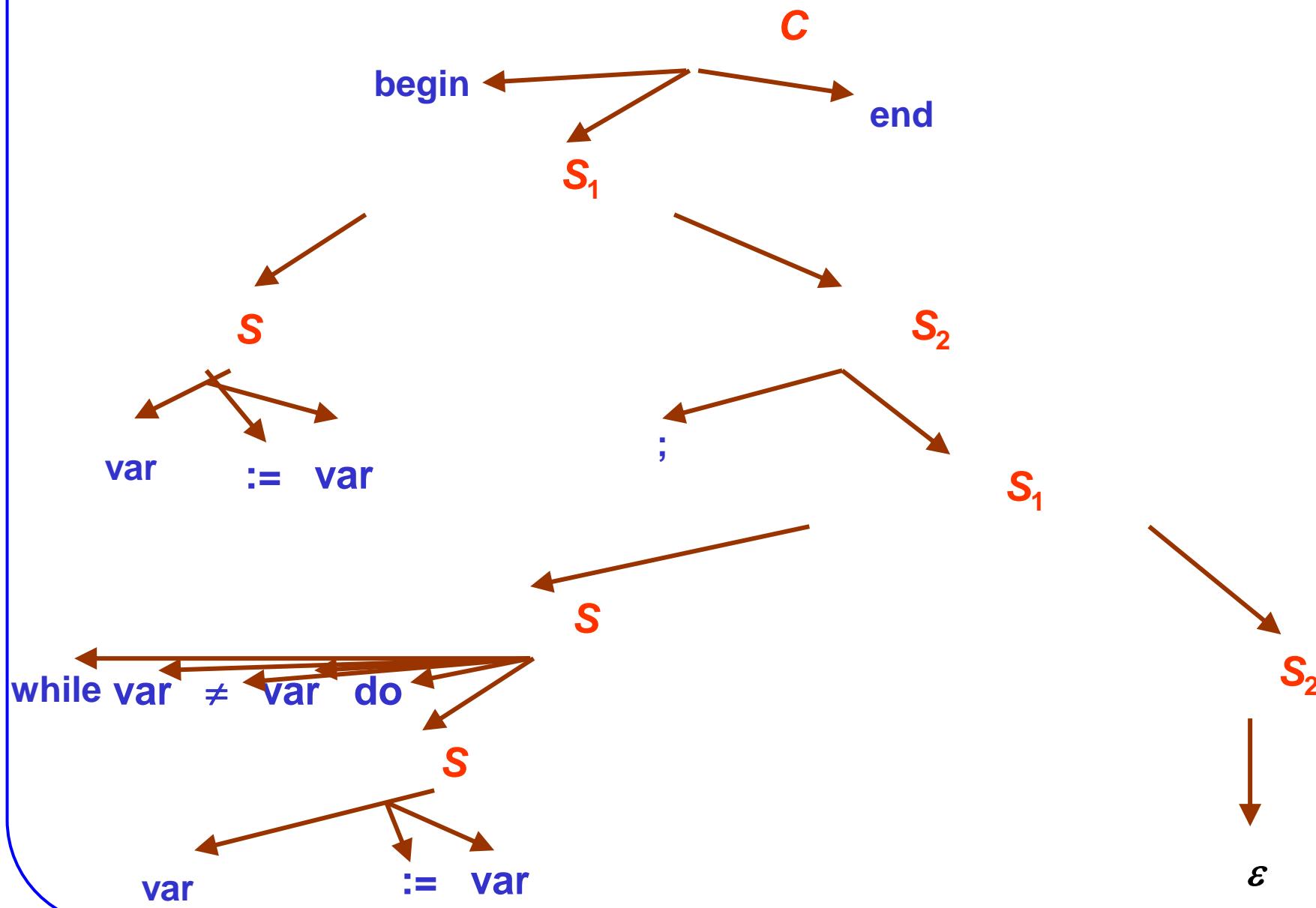


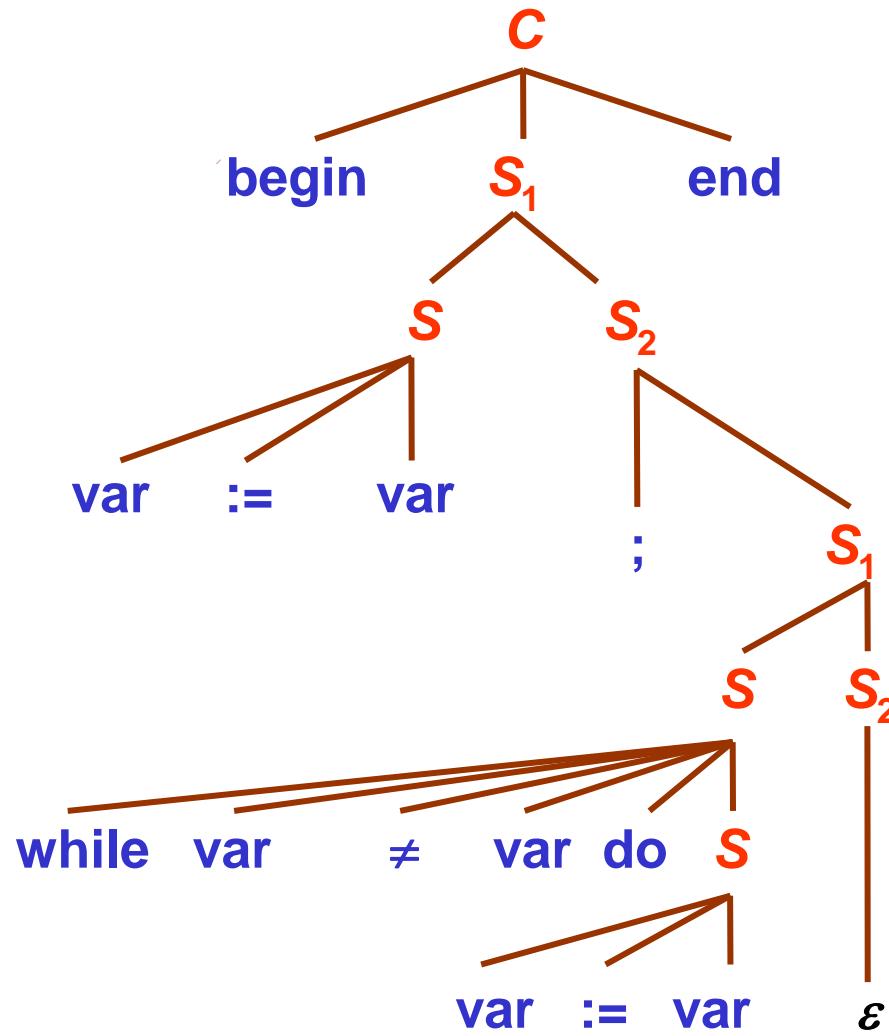












Recursive Descent Parsing

Recursive Descent Parsing

- Main Program

Recursive Descent Parsing

- **Main Program**

MainProgram()

Recursive Descent Parsing

- **Main Program**

```
MainProgram()  
{
```

Recursive Descent Parsing

- Main Program

```
MainProgram()  
{
```

Input = The left most symbol in string w;

Recursive Descent Parsing

- Main Program

```
MainProgram()  
{
```

*Input = The left most symbol in string w;
Call the function associated with the start symbol of the
grammar;*

Recursive Descent Parsing

- Main Program

```
MainProgram()  
{
```

*Input = The left most symbol in string w;
Call the function associated with the start symbol of the
grammar;*

```
if( Input ≠ ⊥ )
```

Recursive Descent Parsing

- Main Program

```
MainProgram()  
{
```

*Input = The left most symbol in string w;
Call the function associated with the start symbol of the
grammar;*

```
if( Input ≠ ⊥ )  
    Write ( “ w ∉ L(G) ” );
```

Recursive Descent Parsing

- Main Program

```
MainProgram()  
{
```

*Input = The left most symbol in string w;
Call the function associated with the start symbol of the
grammar;*

```
if( Input ≠ ⊥ )  
    Write ( “ w ∉ L(G) ” );  
else
```

Recursive Descent Parsing

- Main Program

```
MainProgram()  
{
```

*Input = The left most symbol in string w;
Call the function associated with the start symbol of the
grammar;*

```
if( Input ≠ ⊥ )  
    Write ( “ w ∉ L(G) ” );  
else  
    Write ( “ w ∈ L(G) ” );
```

Recursive Descent Parsing

- Main Program

```
MainProgram()  
{
```

*Input = The left most symbol in string w;
Call the function associated with the start symbol of the
grammar;*

```
    if( Input ≠ ⊥ )  
        Write ( “ w ∈ L(G) ” );  
    else  
        Write ( “ w ∉ L(G) ” );
```

```
}
```

Recursive Descent Parsing

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$
- $A()$

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

$A()$
{

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()
{
    case( Input )
```

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()
{
    case( Input )
{
```

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()  
{  
    case( Input )  
    {  
        a1:
```

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()  
{  
    case( Input )  
    {  
        a1:
```

Check input symbols if they match string β_1 ;

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()  
{  
    case( Input )  
    {  
        a1:  
            Check input symbols if they match string  $\beta_1$ ;  
        a2:  
    }  
}
```

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()  
{  
    case( Input )  
    {  
        a1:  
            Check input symbols if they match string  $\beta_1$ ;  
        a2:  
            Check input symbols if they match string  $\beta_2$ ;  
    }  
}
```

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()  
{  
    case( Input )  
    {  
        a1:  
            Check input symbols if they match string β1;  
        a2:  
            Check input symbols if they match string β2;  
        ...  
    }  
}
```

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()  
{  
    case( Input )  
    {  
        a1:  
        a2:  
        ---  
        an:
```

Check input symbols if they match string β_1 ;

Check input symbols if they match string β_2 ;

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()  
{  
    case( Input )  
    {  
        a1:  
            Check input symbols if they match string  $\beta_1$ ;  
        a2:  
            Check input symbols if they match string  $\beta_2$ ;  
        ---  
        an:  
            Check input symbols if they match string  $\beta_n$ ;  
    }  
}
```

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()
{
    case( Input )
    {
        a1:
            Check input symbols if they match string  $\beta_1$ ;
        a2:
            Check input symbols if they match string  $\beta_2$ ;
        ...
        an:
            Check input symbols if they match string  $\beta_n$ ;
    }
}
```

Other symbols:

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()  
{  
    case( Input )  
    {  
         $a_1$ :  
            Check input symbols if they match string  $\beta_1$ ;  
         $a_2$ :  
            Check input symbols if they match string  $\beta_2$ ;  
        ---  
         $a_n$ :  
            Check input symbols if they match string  $\beta_n$ ;  
    }  
}
```

Other symbols:

Write (“ $w \notin L(G)$ ”);

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()
{
    case( Input )
    {
        a1:
            Check input symbols if they match string  $\beta_1$ ;
        a2:
            Check input symbols if they match string  $\beta_2$ ;
        ...
        an:
            Check input symbols if they match string  $\beta_n$ ;
    }
    Other symbols:
    Write ( “  $w \notin L(G)$  ” );
}
```

Recursive Descent Parsing

- Production: $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n$

```
A()
{
    case( Input )
    {
        a1:
            Check input symbols if they match string  $\beta_1$ ;
        a2:
            Check input symbols if they match string  $\beta_2$ ;
        ...
        an:
            Check input symbols if they match string  $\beta_n$ ;
    }
}
```

Other symbols:
Write (“ $w \notin L(G)$ ”);

Recursive Descent Parsing

Recursive Descent Parsing

- Terminal symbol b on the right side of the production but not on the leftmost place:

Recursive Descent Parsing

- Terminal symbol b on the right side of the production but not on the leftmost place:

$$A \rightarrow a \alpha \underline{b} \gamma$$

Recursive Descent Parsing

- Terminal symbol b on the right side of the production but not on the leftmost place:

$$A \rightarrow a \alpha \underline{b} \gamma$$

$A()$

Recursive Descent Parsing

- Terminal symbol b on the right side of the production but not on the leftmost place:

$$A \rightarrow a \alpha \underline{b} \gamma$$

$A()$
{

Recursive Descent Parsing

- Terminal symbol b on the right side of the production but not on the leftmost place:

$$A \rightarrow a \alpha \underline{b} \gamma$$

$A()$
{

Recursive Descent Parsing

- Terminal symbol b on the right side of the production but not on the leftmost place:

$$A \rightarrow a \alpha \underline{b} \gamma$$

$A()$
{

Input = Read the next symbol in string w ;

Recursive Descent Parsing

- Terminal symbol b on the right side of the production but not on the leftmost place:

$$A \rightarrow a \alpha \underline{b} \gamma$$

$A()$
{

*Input = Read the next symbol in string w ;
if(Input != b)*

Recursive Descent Parsing

- Terminal symbol b on the right side of the production but not on the leftmost place:

$$A \rightarrow a \alpha \underline{b} \gamma$$

A()
{

***Input* = Read the next symbol in string w ;**
if(*Input* != b)
Write (“ $w \notin L(G)$ ”);

Recursive Descent Parsing

- Terminal symbol b on the right side of the production but not on the leftmost place:

$$A \rightarrow a \alpha \underline{b} \gamma$$

```
A()  
{  
---
```

Input = Read the next symbol in string w ;
if(Input != b)
Write (“ $w \notin L(G)$ ”);

```
---
```

Recursive Descent Parsing

- Terminal symbol b on the right side of the production but not on the leftmost place:

$$A \rightarrow a \alpha \underline{b} \gamma$$

```
A()  
{  
    ---  
}  
}
```

*Input = Read the next symbol in string w ;
if(*Input* != b)
 Write (“ $w \notin L(G)$ ”);*

Recursive Descent Parsing

Recursive Descent Parsing

- Nonterminal symbol B on the right of the production but no on the leftmost place

Recursive Descent Parsing

- Nonterminal symbol B on the right of the production but no on the leftmost place

$$A \rightarrow a \alpha \underline{B} \gamma$$

Recursive Descent Parsing

- Nonterminal symbol B on the right of the production but no on the leftmost place

$$A \rightarrow a \alpha \underline{B} \gamma$$

$A()$

Recursive Descent Parsing

- Nonterminal symbol B on the right of the production but no on the leftmost place

$$A \rightarrow a \alpha \underline{B} \gamma$$

$A()$
{

Recursive Descent Parsing

- Nonterminal symbol B on the right of the production but no on the leftmost place

$$A \rightarrow a \alpha \underline{B} \gamma$$

$A()$
{

Recursive Descent Parsing

- Nonterminal symbol B on the right of the production but no on the leftmost place

$$A \rightarrow a \alpha \underline{B} \gamma$$

$A()$
{

Input = Read the next symbol in string w ;

Recursive Descent Parsing

- Nonterminal symbol B on the right of the production but no on the leftmost place

$$A \rightarrow a \alpha \underline{B} \gamma$$

$A()$
{

*Input = Read the next symbol in string w ;
 $B()$;*

Recursive Descent Parsing

- Nonterminal symbol B on the right of the production but no on the leftmost place

$$A \rightarrow a \alpha \underline{B} \gamma$$

$A()$
{

*Input = Read the next symbol in string w ;
 $B()$;*

Recursive Descent Parsing

- Nonterminal symbol B on the right of the production but no on the leftmost place

$$A \rightarrow a \alpha \underline{B} \gamma$$

$A()$
{

 $B()$;

}
}

Input = Read the next symbol in string w ;
 $B()$;

Recursive Descent Parsing

Recursive Descent Parsing

- Nonterminal symbol B on the leftmost place on the right side of the production:

Recursive Descent Parsing

- Nonterminal symbol B on the leftmost place on the right side of the production:

$$A \rightarrow \underline{B} a \alpha \gamma$$

Recursive Descent Parsing

- Nonterminal symbol B on the leftmost place on the right side of the production:

$$A \rightarrow \underline{B} a \alpha \gamma$$

$$A()$$

Recursive Descent Parsing

- Nonterminal symbol B on the leftmost place on the right side of the production:

$$A \rightarrow \underline{B} a \alpha \gamma$$

$A()$
 {

Recursive Descent Parsing

- Nonterminal symbol B on the leftmost place on the right side of the production:

$$A \rightarrow \underline{B} a \alpha \gamma$$

$A()$
{}
 $B();$

Recursive Descent Parsing

- Nonterminal symbol B on the leftmost place on the right side of the production:

$$A \rightarrow \underline{B} a \alpha \gamma$$

$A()$
 {
 $B();$

Recursive Descent Parsing

- Nonterminal symbol B on the leftmost place on the right side of the production:

$$A \rightarrow \underline{B} a \alpha \gamma$$

$A()$
 {
 $B();$

 }

Plan predavanja

3.1.3 String parsing

88

Bottom – up string parsing

Bottom – up string parsing

- **Bottom – up string parsing:**

Bottom – up string parsing

- **Bottom – up string parsing:**
 - Starts the process of building parse tree from the leaves

Bottom – up string parsing

- **Bottom – up string parsing:**
 - Starts the process of building parse tree from the leaves
 - In the derivative substrings of terminal and nonterminal symbols the goal is to match *the right side of some grammar production*

Bottom – up string parsing

- **Bottom – up string parsing:**
 - Starts the process of building parse tree from the leaves
 - In the derivative substrings of terminal and nonterminal symbols the goal is to match *the right side of some grammar production*
 - The replacement pattern

Bottom – up string parsing

- **Bottom – up string parsing:**
 - Starts the process of building parse tree from the leaves
 - In the derivative substrings of terminal and nonterminal symbols the goal is to match *the right side of some grammar production*
 - The replacement pattern
 - The matched substring is replaced with *the left side of the production*

Bottom – up string parsing

- **Bottom – up string parsing:**
 - Starts the process of building parse tree from the leaves
 - In the derivative substrings of terminal and nonterminal symbols the goal is to match *the right side of some grammar production*
 - The replacement pattern
 - The matched substring is replaced with *the left side of the production*
 - Productions rules are applied in reverse – *reductions*

Bottom – up string parsing

Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

var + var * var

Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

var + var * var

Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

var + var * var

Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

var + var * var

Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

var + var * var $\Leftarrow F + \text{var} * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

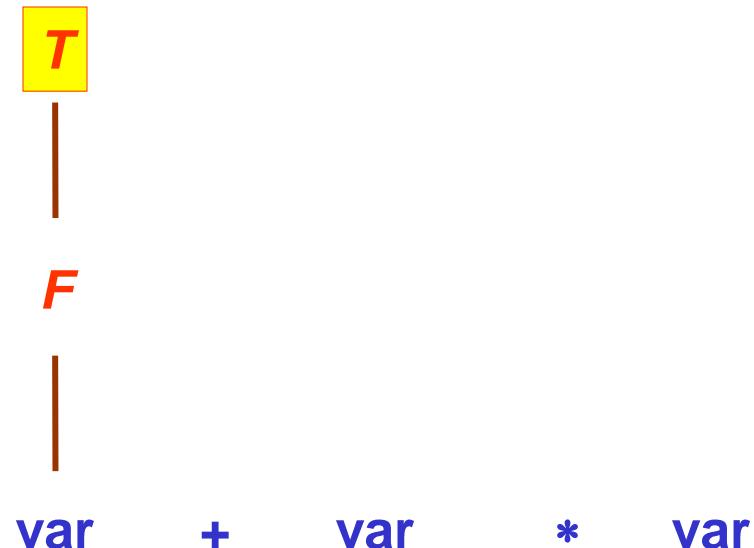
$\Leftarrow T + \text{var} * \text{var}$

T
|
 F
|
 $\text{var} + \text{var} * \text{var}$

Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$
 $\Leftarrow T + \text{var} * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

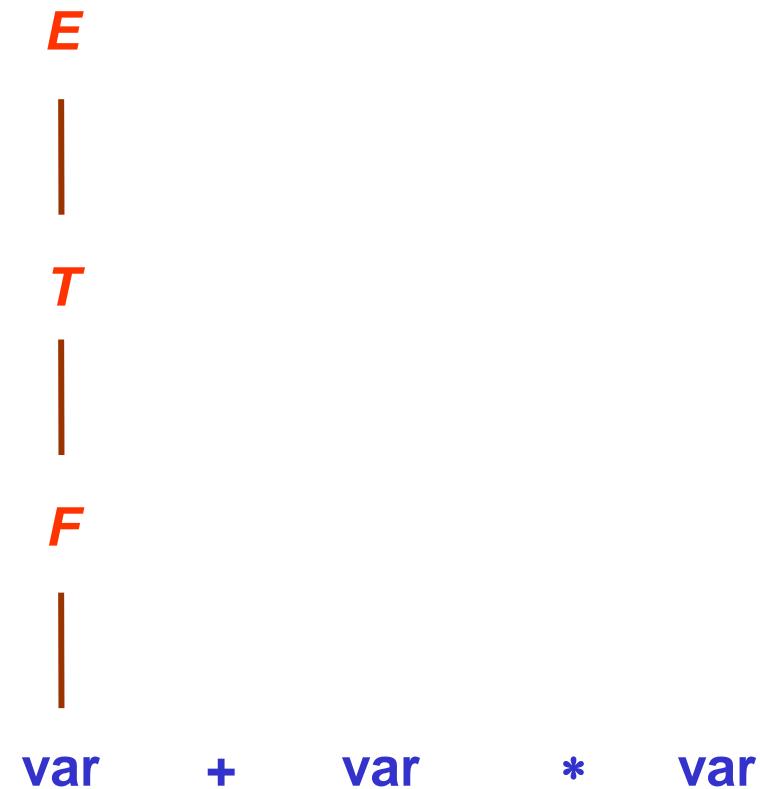
T
|
 F
|
 $\text{var} + \text{var} * \text{var}$

Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

E

T

F

$\text{var} + \text{var} * \text{var}$

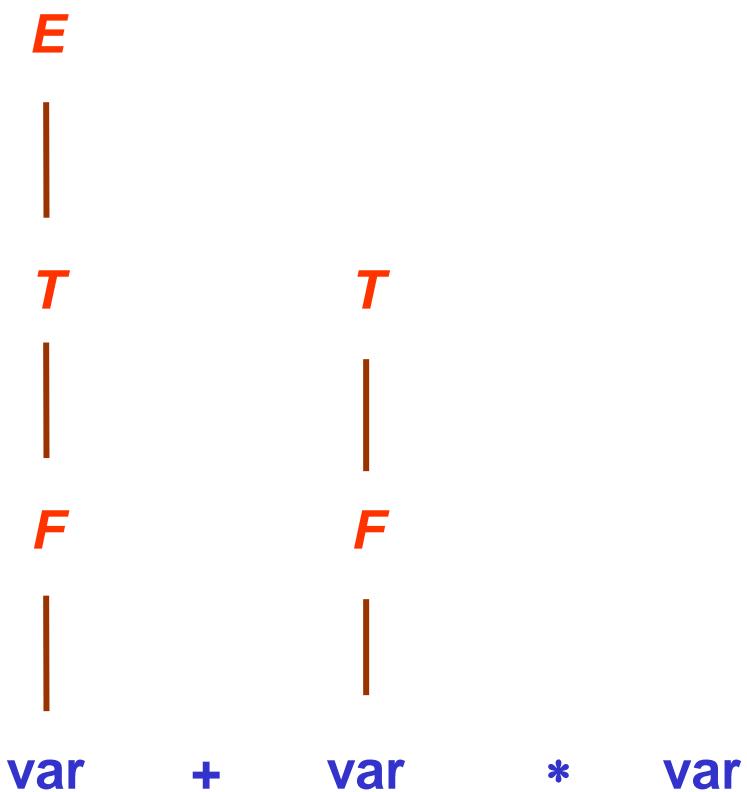
Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$



Bottom – up string parsing

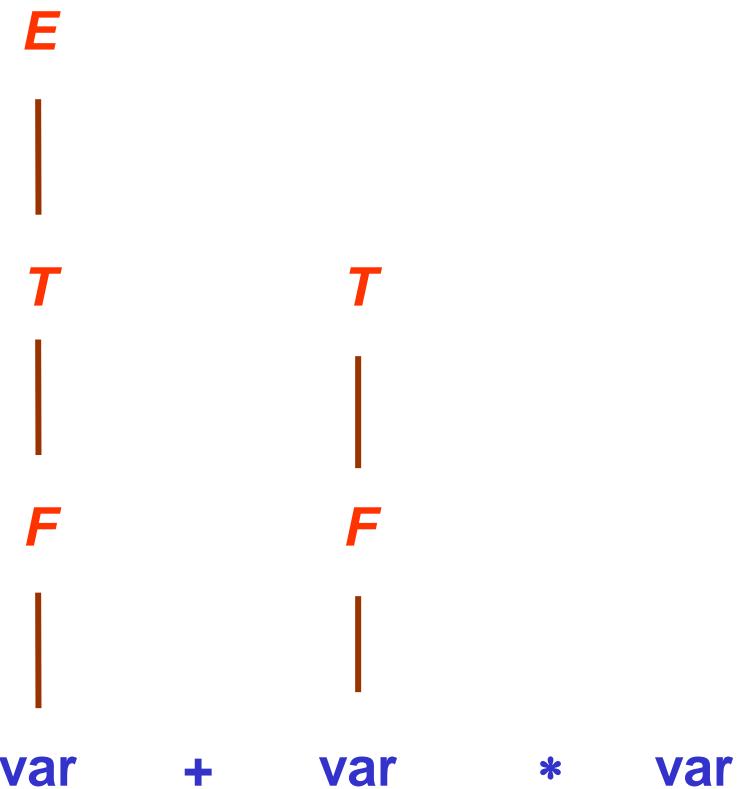
- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$



Bottom – up string parsing

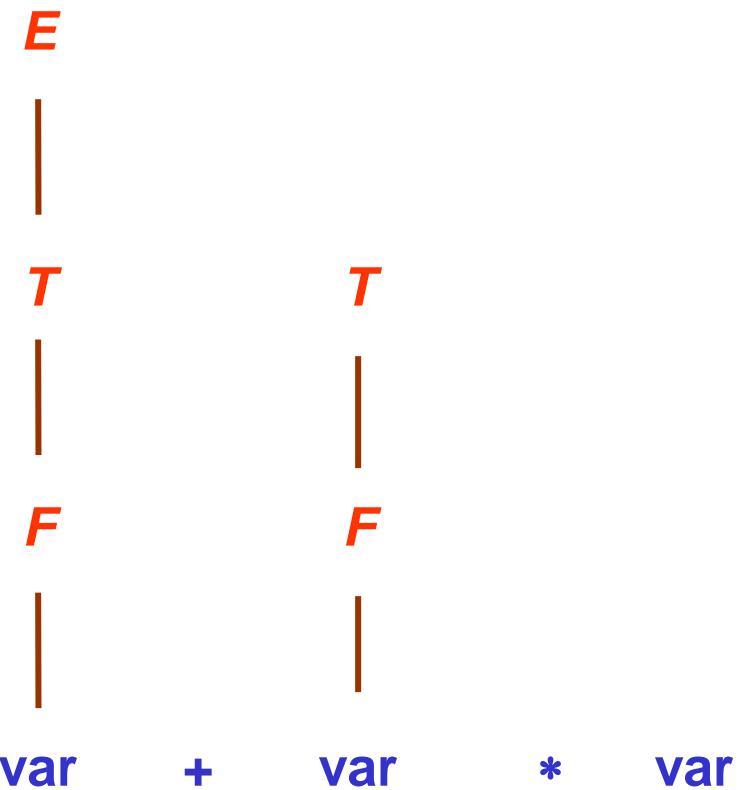
- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$



Bottom – up string parsing

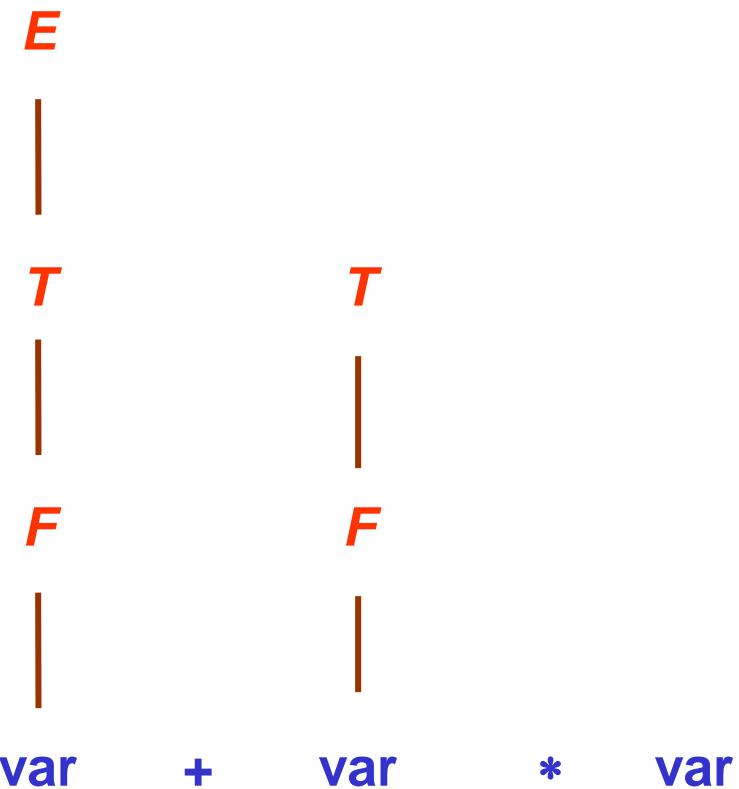
- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

$\Leftarrow E * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

E

T

F

var

$+$

var

T

F

$*$

Bottom – up string parsing

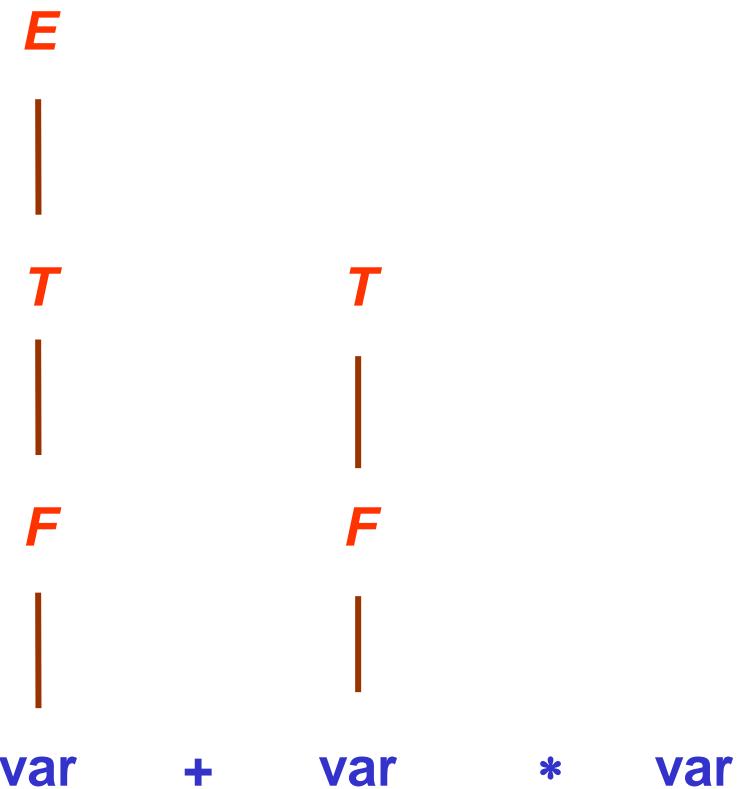
- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

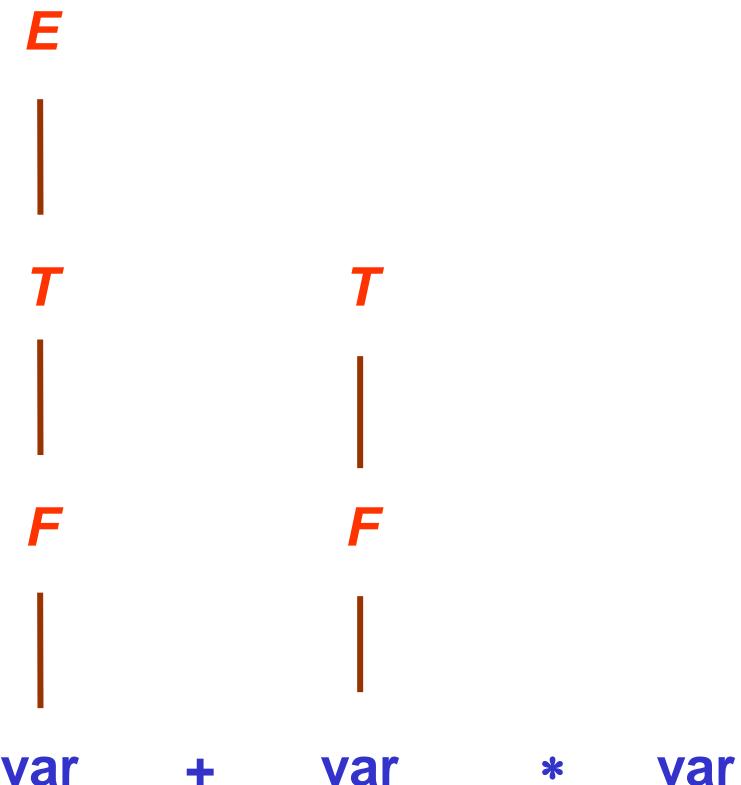
$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

$\Leftarrow E + E * \text{var}$



Bottom – up string parsing

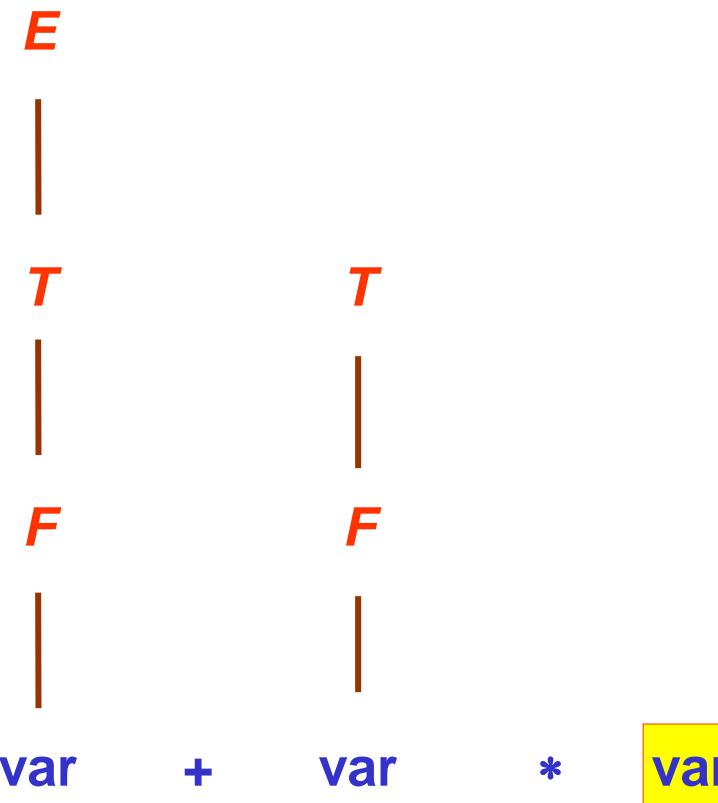
- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

E

T

F

var

$+$

var

T

F

$*$

Bottom – up string parsing

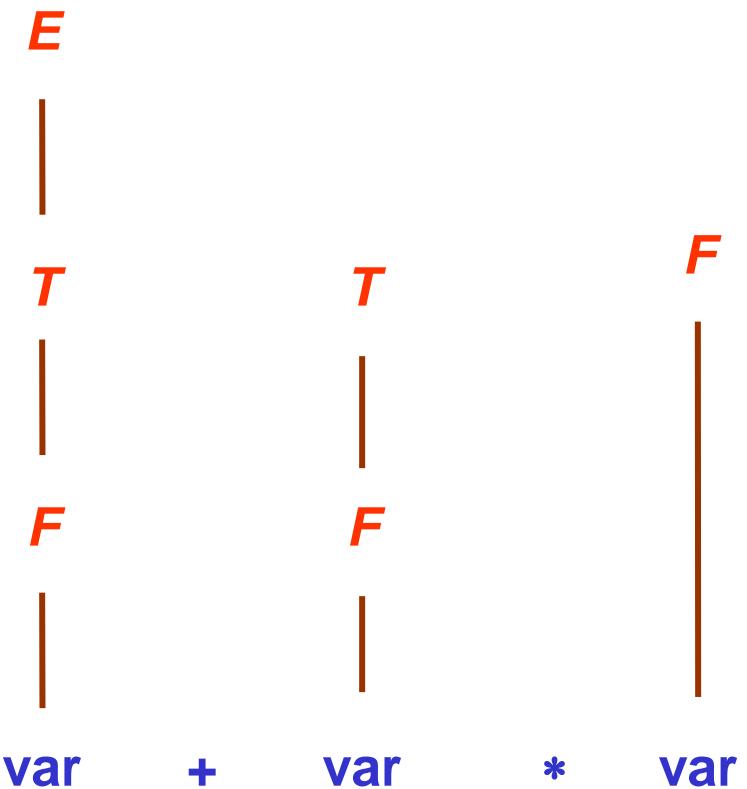
- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

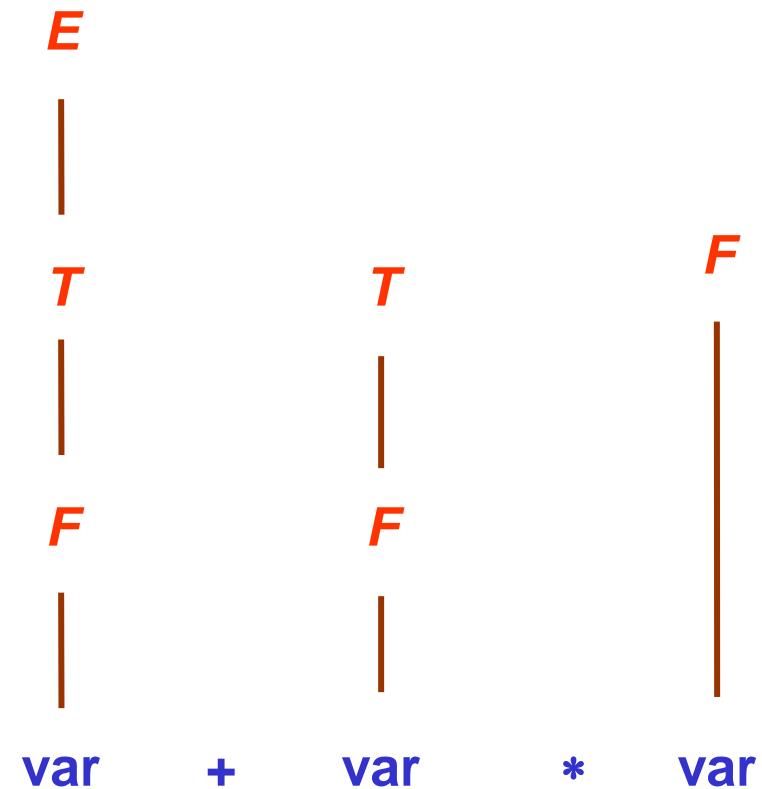
$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

$\Leftarrow E + T * F$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

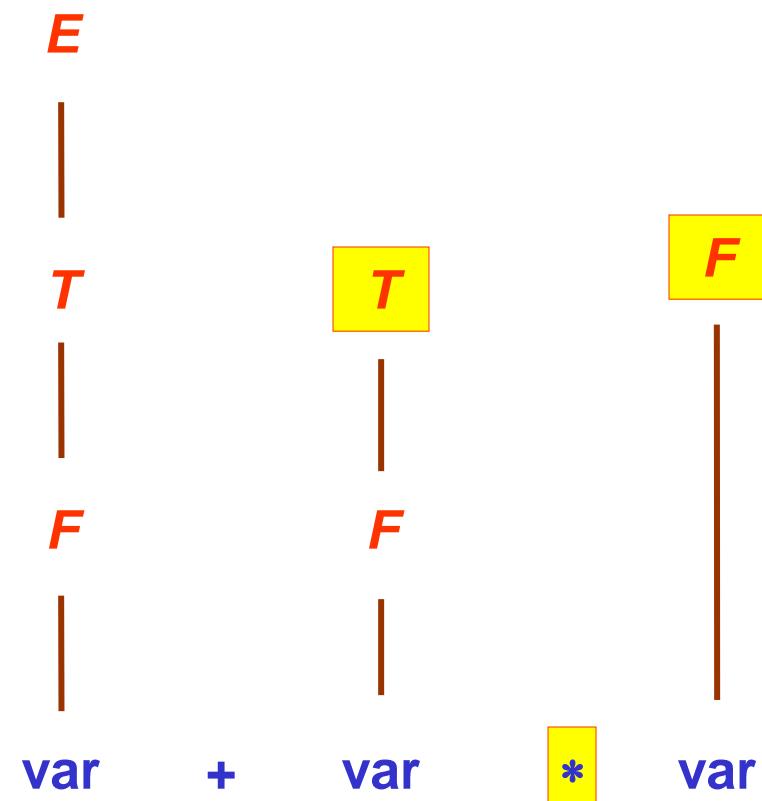
$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

$\Leftarrow E + T * F$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

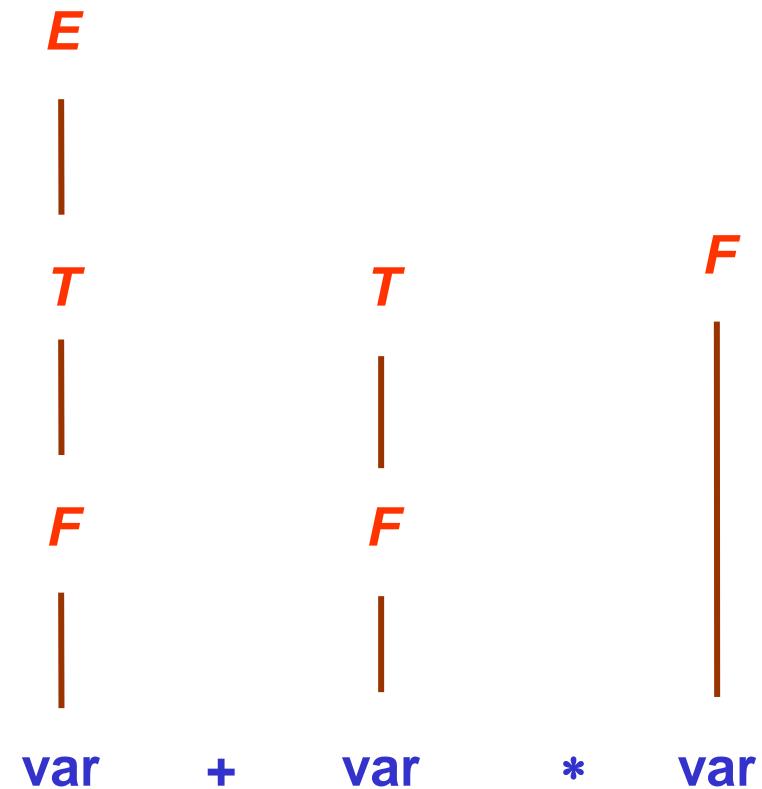
$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

$\Leftarrow E + T * F$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

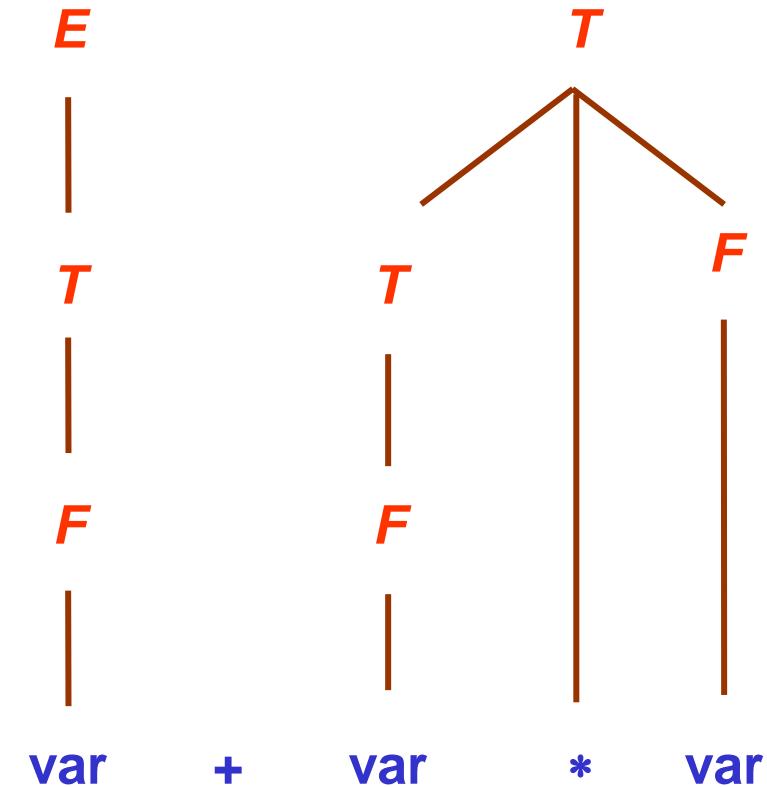
$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

$\Leftarrow E + T * F$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

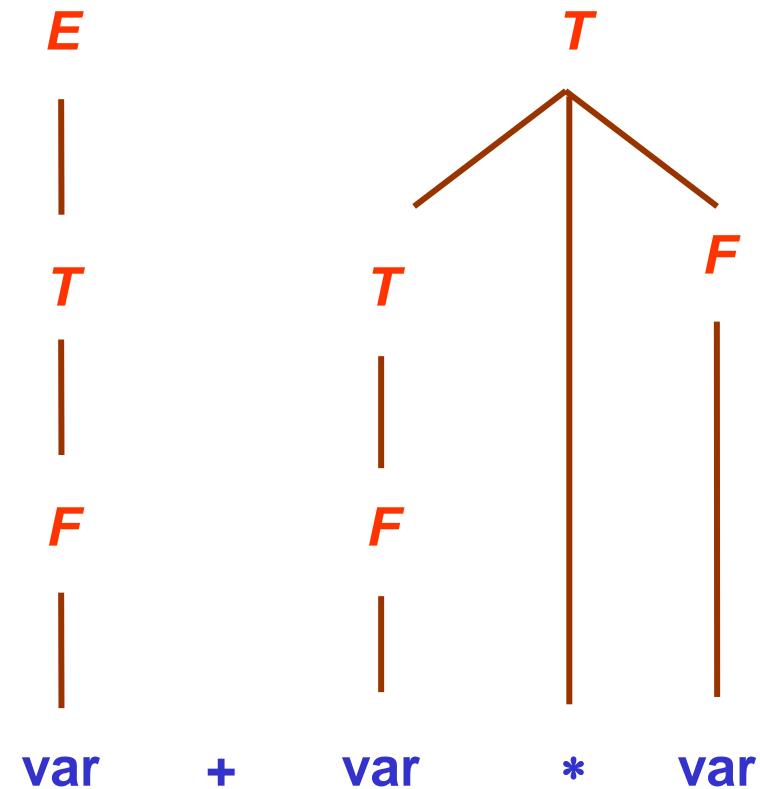
$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

$\Leftarrow E + T * F$

$\Leftarrow E + T$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

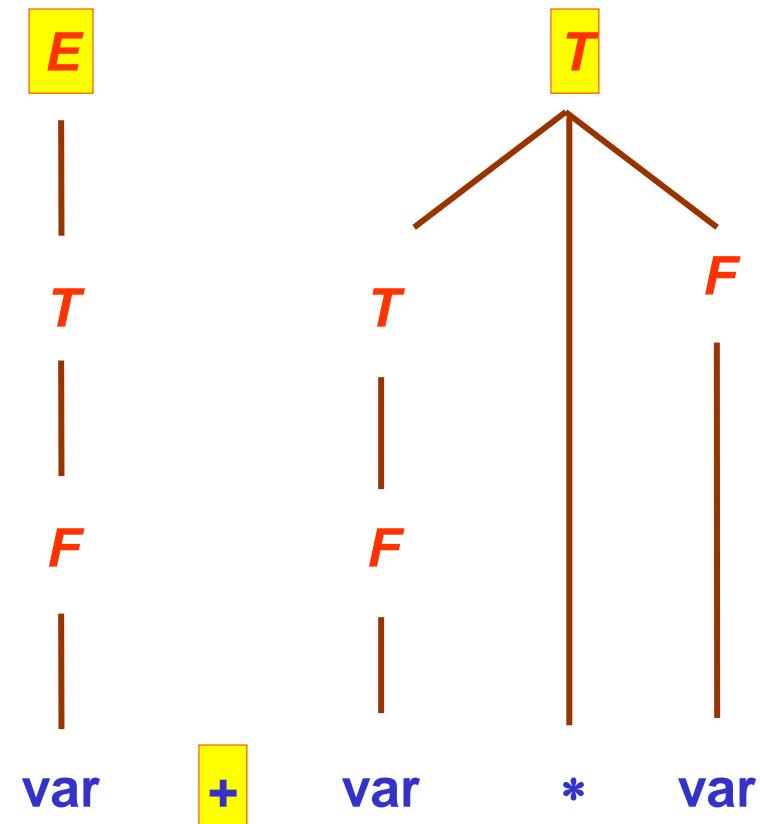
$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

$\Leftarrow E + T * F$

$\Leftarrow E + T$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

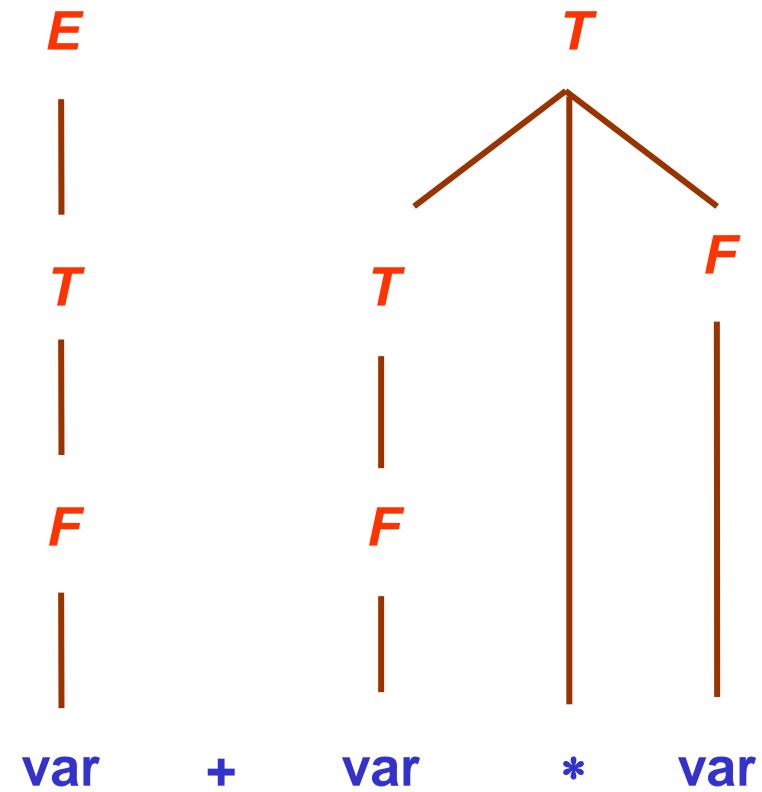
$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

$\Leftarrow E + T * F$

$\Leftarrow E + T$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

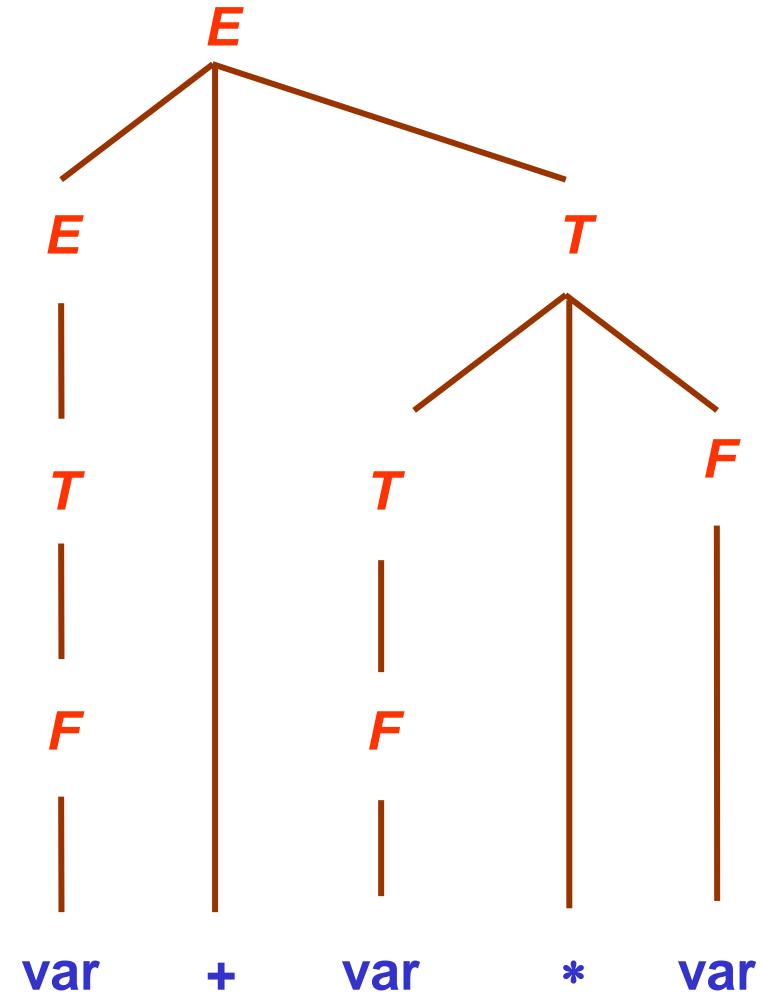
$\Leftarrow T + \text{var} * \text{var}$

$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

$\Leftarrow E + T * F$

$\Leftarrow E + T$



Bottom – up string parsing

- 1) $E \rightarrow E + T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow T * F$
- 4) $T \rightarrow F$
- 5) $F \rightarrow (E)$
- 6) $F \rightarrow \text{var}$

$\text{var} + \text{var} * \text{var} \Leftarrow F + \text{var} * \text{var}$

$\Leftarrow T + \text{var} * \text{var}$

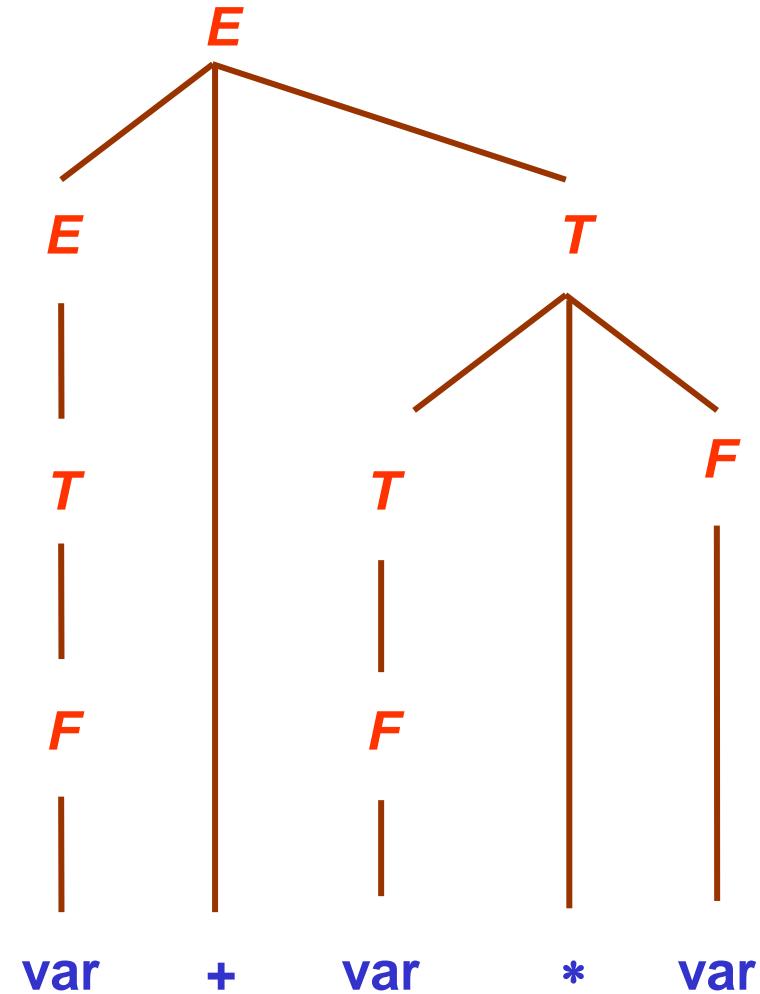
$\Leftarrow E + \text{var} * \text{var}$

$\Leftarrow E + T * \text{var}$

$\Leftarrow E + T * F$

$\Leftarrow E + T$

$\Leftarrow E$



Bottom – up string parsing

Bottom – up string parsing

- $LR(k)$ grammar

Bottom – up string parsing

- $LR(k)$ grammar
- $LR(k)$ grammar

Bottom – up string parsing

- $LR(k)$ grammar
- $LR(k)$ grammar
 - L
 - The string is parsed from left to right (**Left-to-right scanning**)

Bottom – up string parsing

- $LR(k)$ grammar
- $LR(k)$ grammar
 - L
 - The string is parsed from left to right (**Left-to-right scanning**)
 - R
 - The reversed algorithm of string generation by replacing the rightmost nonterminal symbol is applied (**Rightmost derivation**)

Bottom – up string parsing

- $LR(k)$ grammar
- $LR(k)$ grammar
 - L
 - The string is parsed from left to right (**Left-to-right scanning**)
 - R
 - The reversed algorithm of string generation by replacing the rightmost nonterminal symbol is applied (**Rightmost derivation**)
 - k
 - At most k symbols are read ahead in order to bring the decision which reduction needs to be applied
 - $k = 0, 1, 2, 3, 4, \dots$

Bottom – up string parsing

- $LR(k)$ grammar
- $LR(k)$ grammar
 - L
 - The string is parsed from left to right (**Left-to-right scanning**)
 - R
 - The reversed algorithm of string generation by replacing the rightmost nonterminal symbol is applied (**Rightmost derivation**)
 - k
 - At most k symbols are read ahead in order to bring the decision which reduction needs to be applied
 - $k = 0, 1, 2, 3, 4, \dots$
- $LR(0), LR(1)$ – syntax parsing for programming languages

LR parsing

LR parsing

- *LR Parser*

LR parsing

- ***LR Parser***
 - The most general parsing algorithm

LR parsing

- ***LR Parser***
 - The most general parsing algorithm
 - Difficult than the Recursive Descent Parsing

LR parsing

- ***LR* Parser**
 - The most general parsing algorithm
 - Difficult than the Recursive Descent Parsing
 - Different *LR* Parsers – different Parsing tables

LR parsing

- ***LR Parser***
 - The most general parsing algorithm
 - Difficult than the Recursive Descent Parsing
 - Different *LR* Parsers – different Parsing tables
 - *LR Parser* is built using special generator

LR parsing

- ***LR* Parser**
 - The most general parsing algorithm
 - Difficult than the Recursive Descent Parsing
 - Different *LR* Parsers – different Parsing tables
 - *LR* Parser is built using special generator
 - Parsing table is built based on the production set of the Context Free Grammar

LR parsing

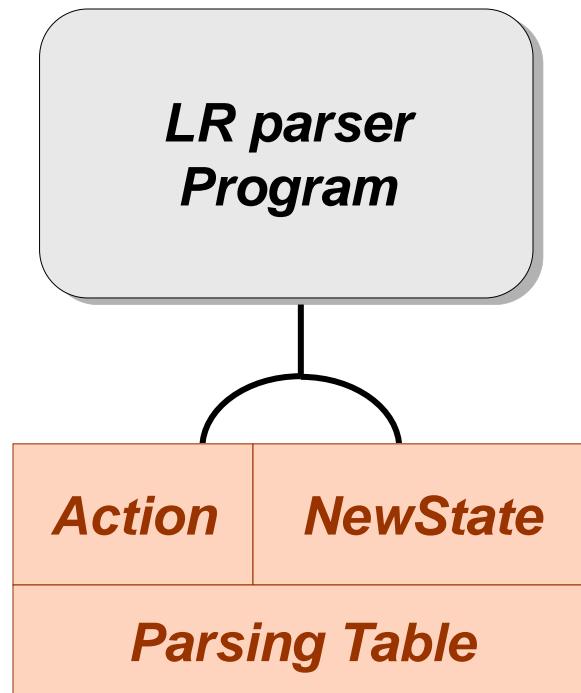
- **LR Parser**
 - The most general parsing algorithm
 - Difficult than the Recursive Descent Parsing
 - Different *LR* Parsers – different Parsing tables
 - *LR* Parser is built using special generator
 - Parsing table is built based on the production set of the Context Free Grammar
 - There are different generator classes (simple and more advanced)

LR parsing

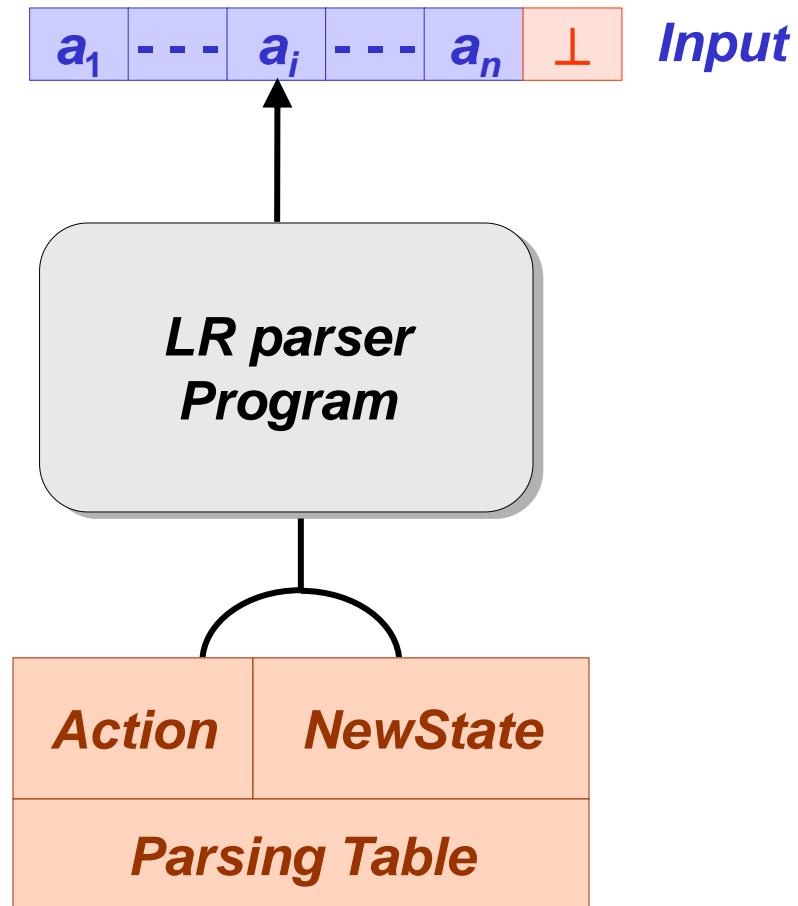
LR parsing

*LR parser
Program*

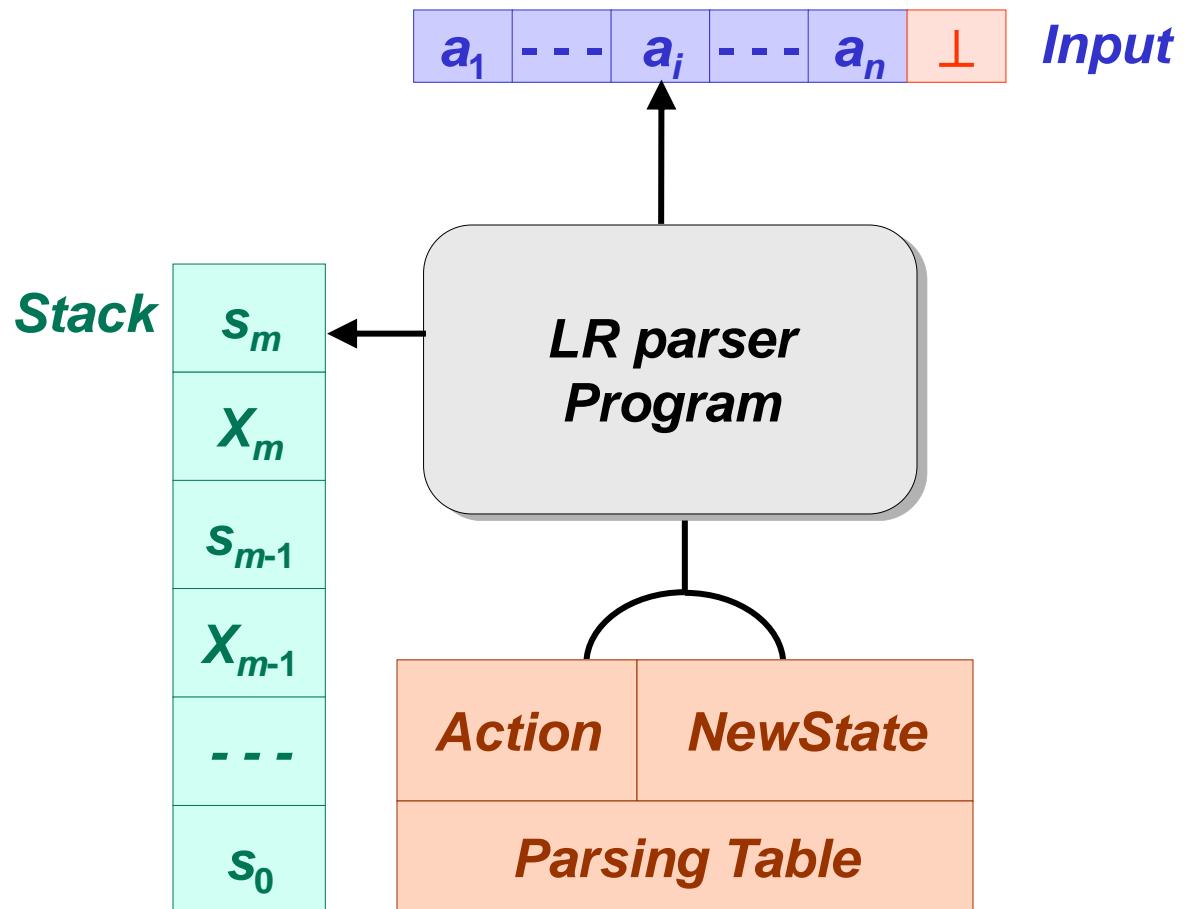
LR parsing



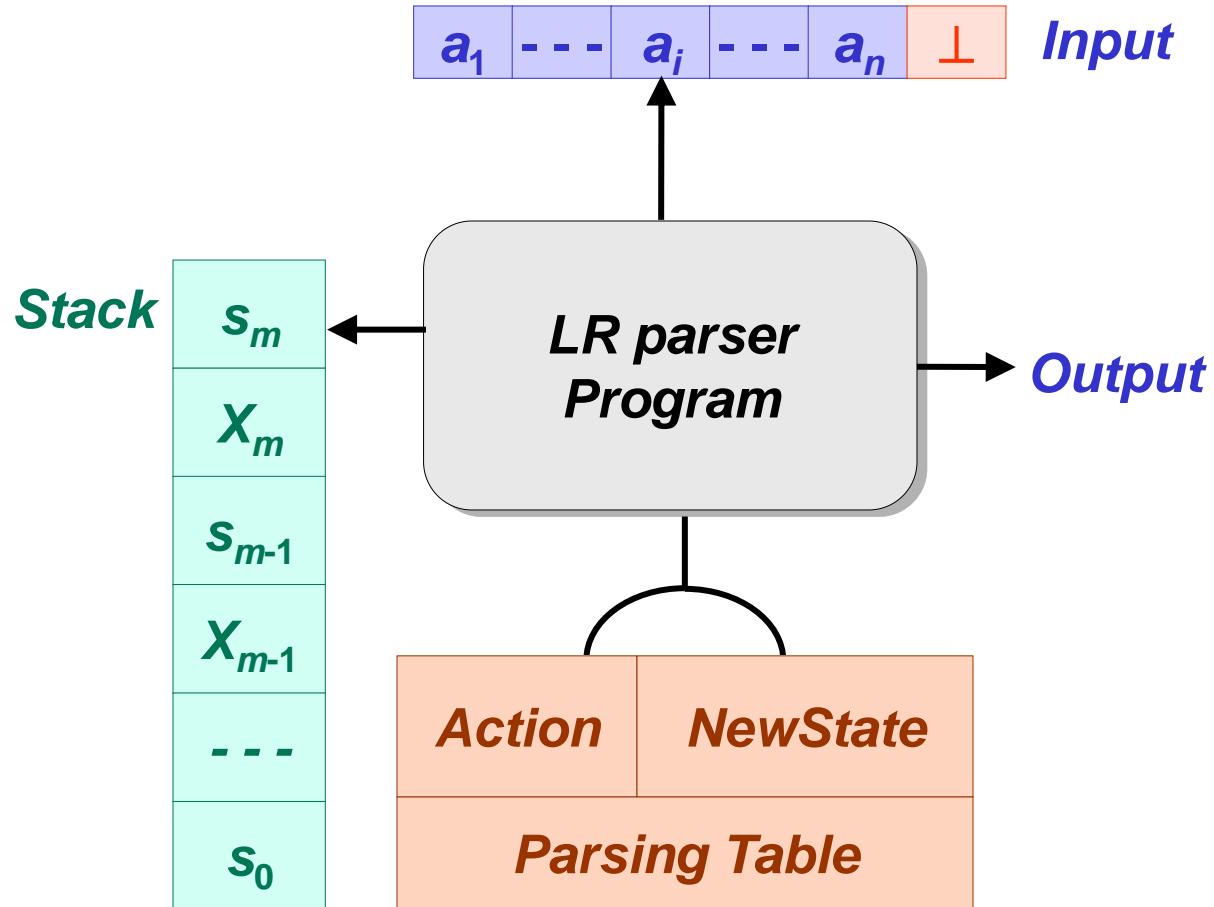
LR parsing



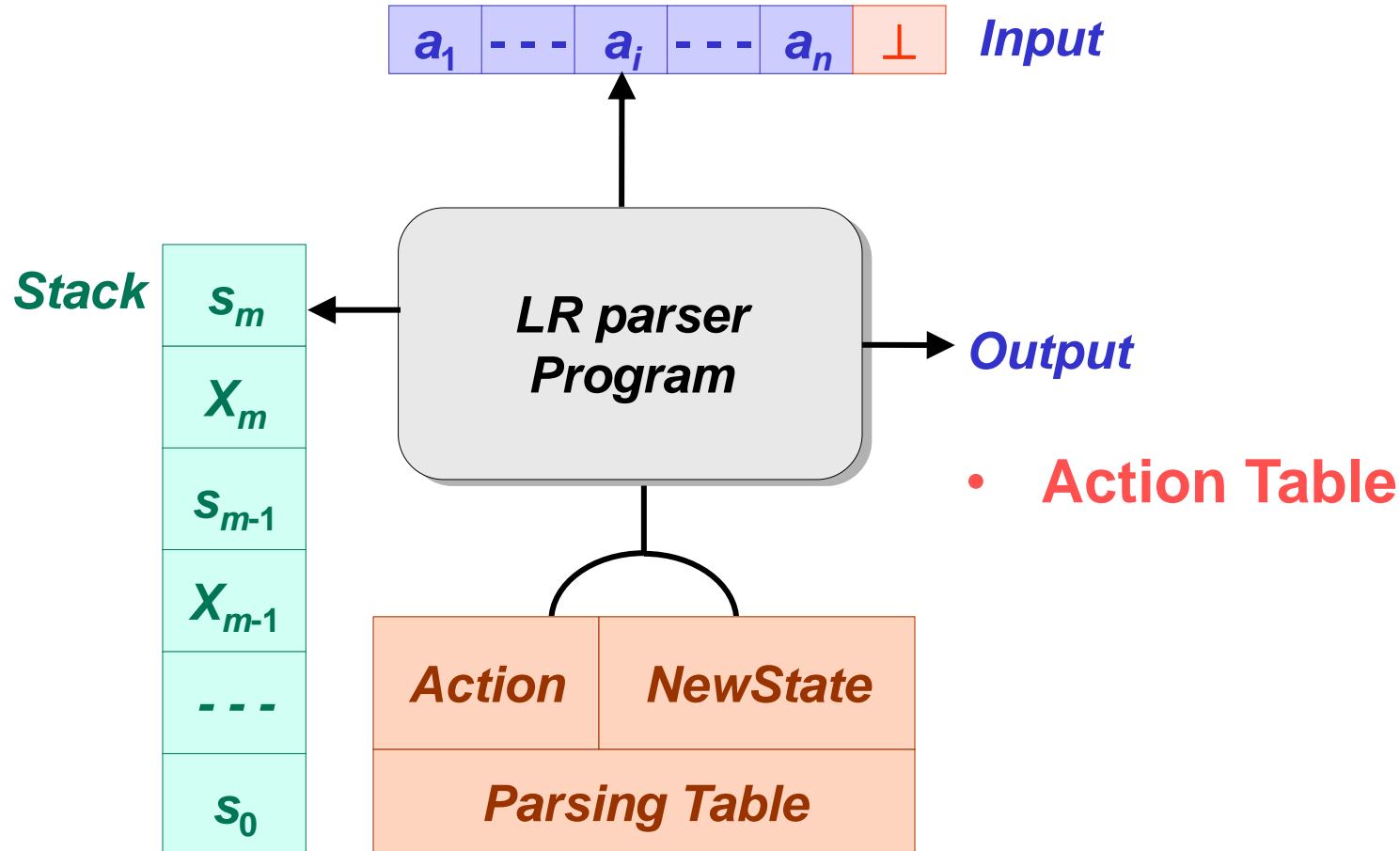
LR parsing



LR parsing

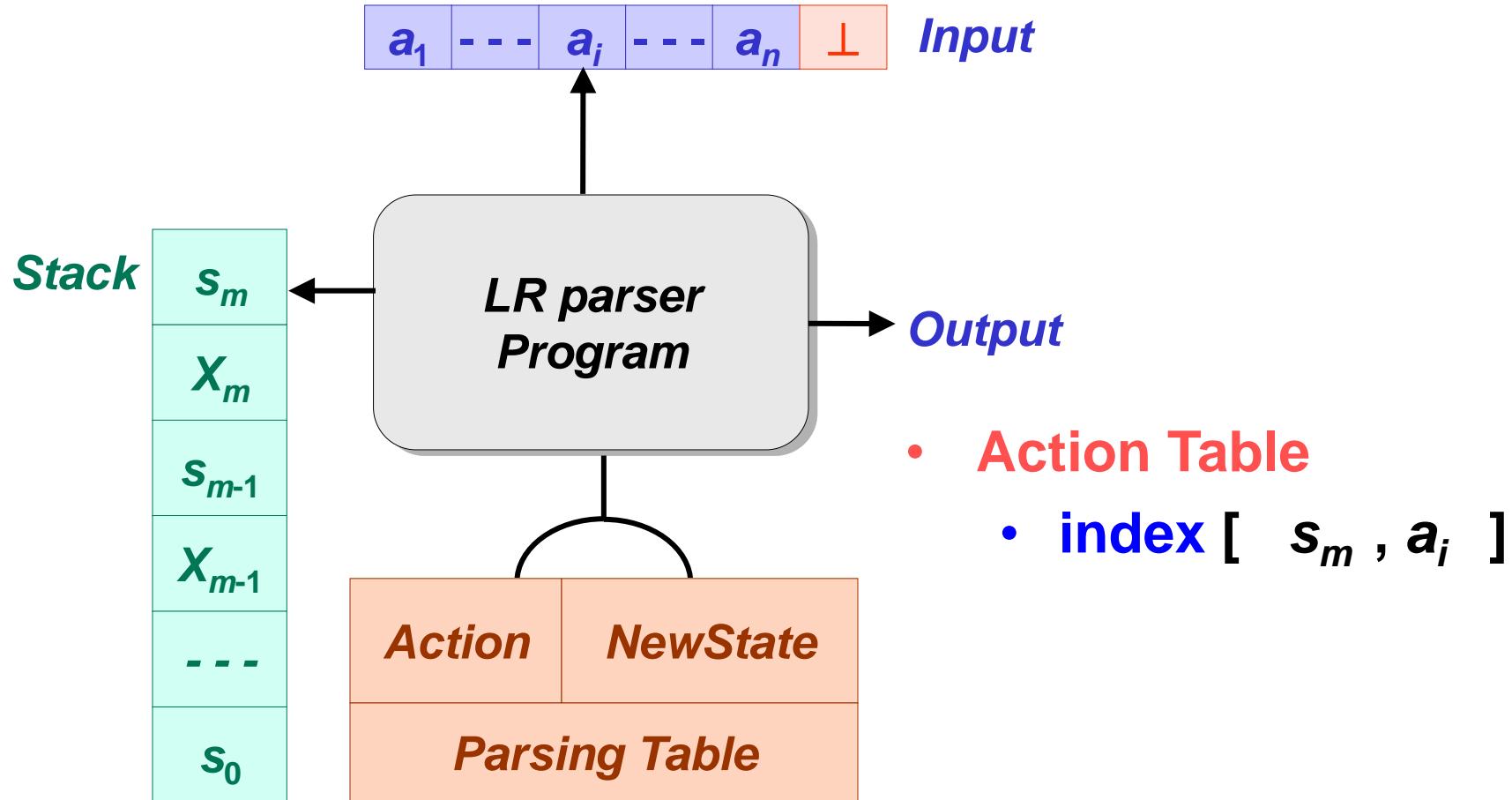


LR parsing

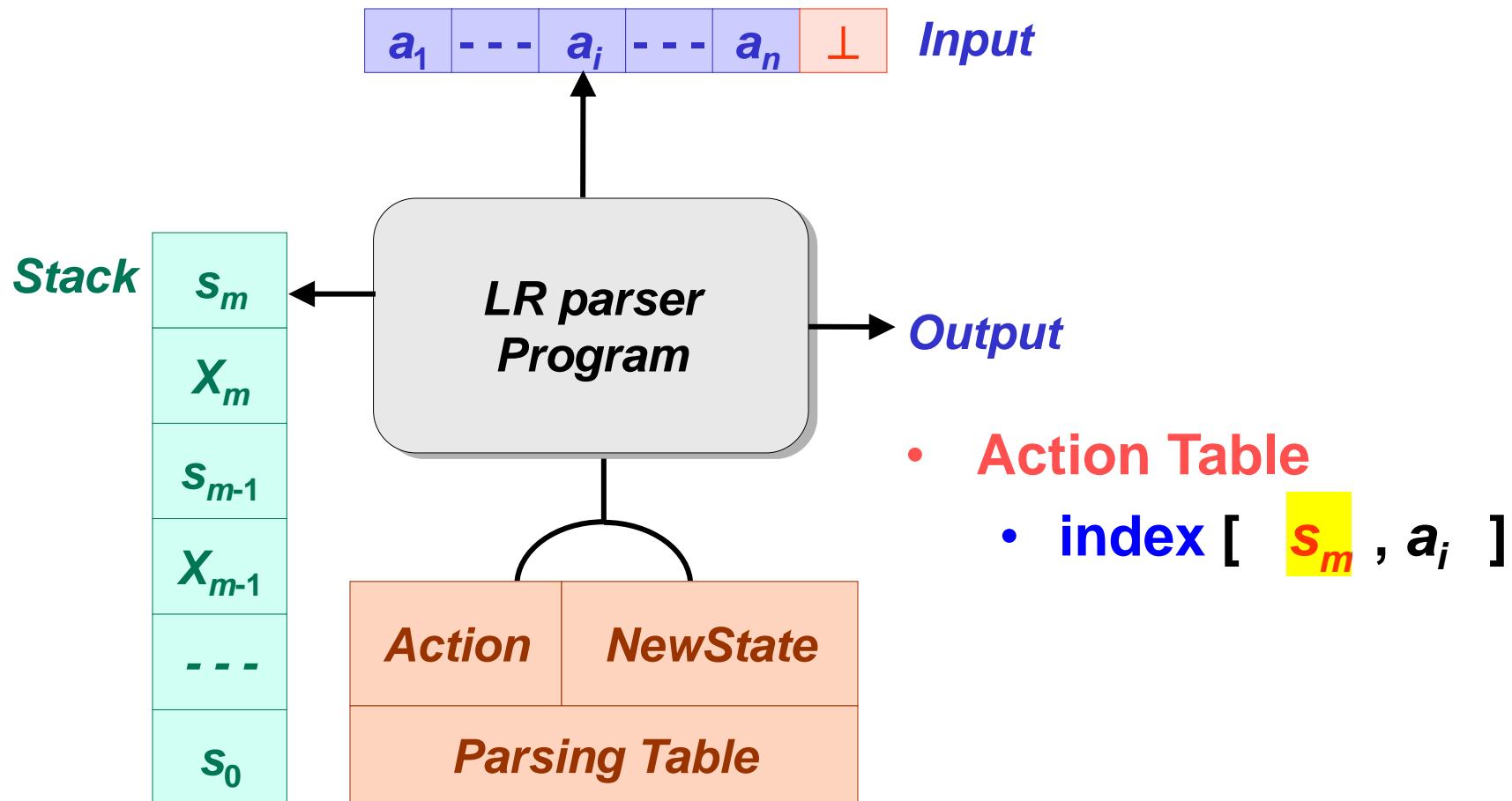


- Action Table

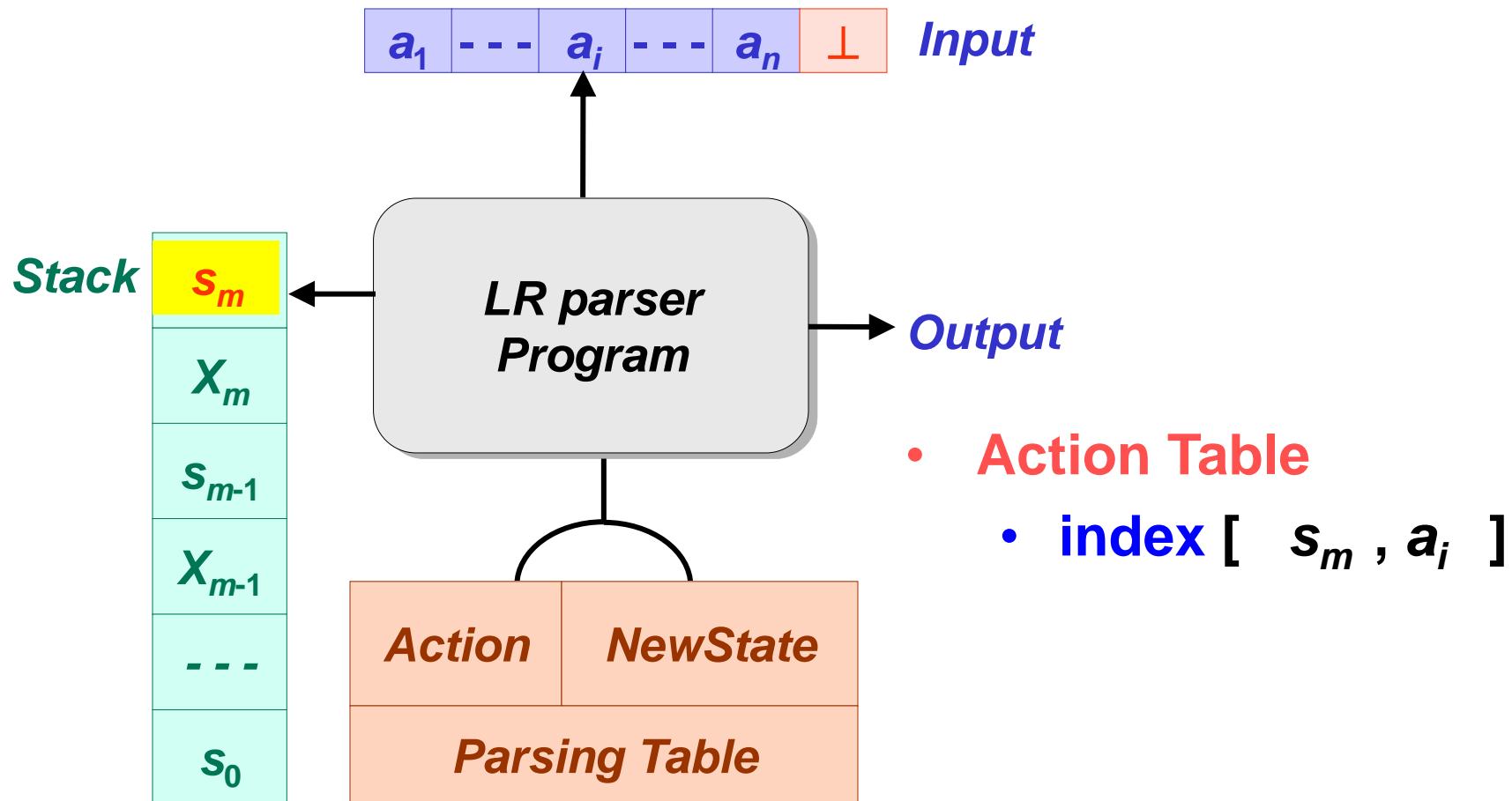
LR parsing



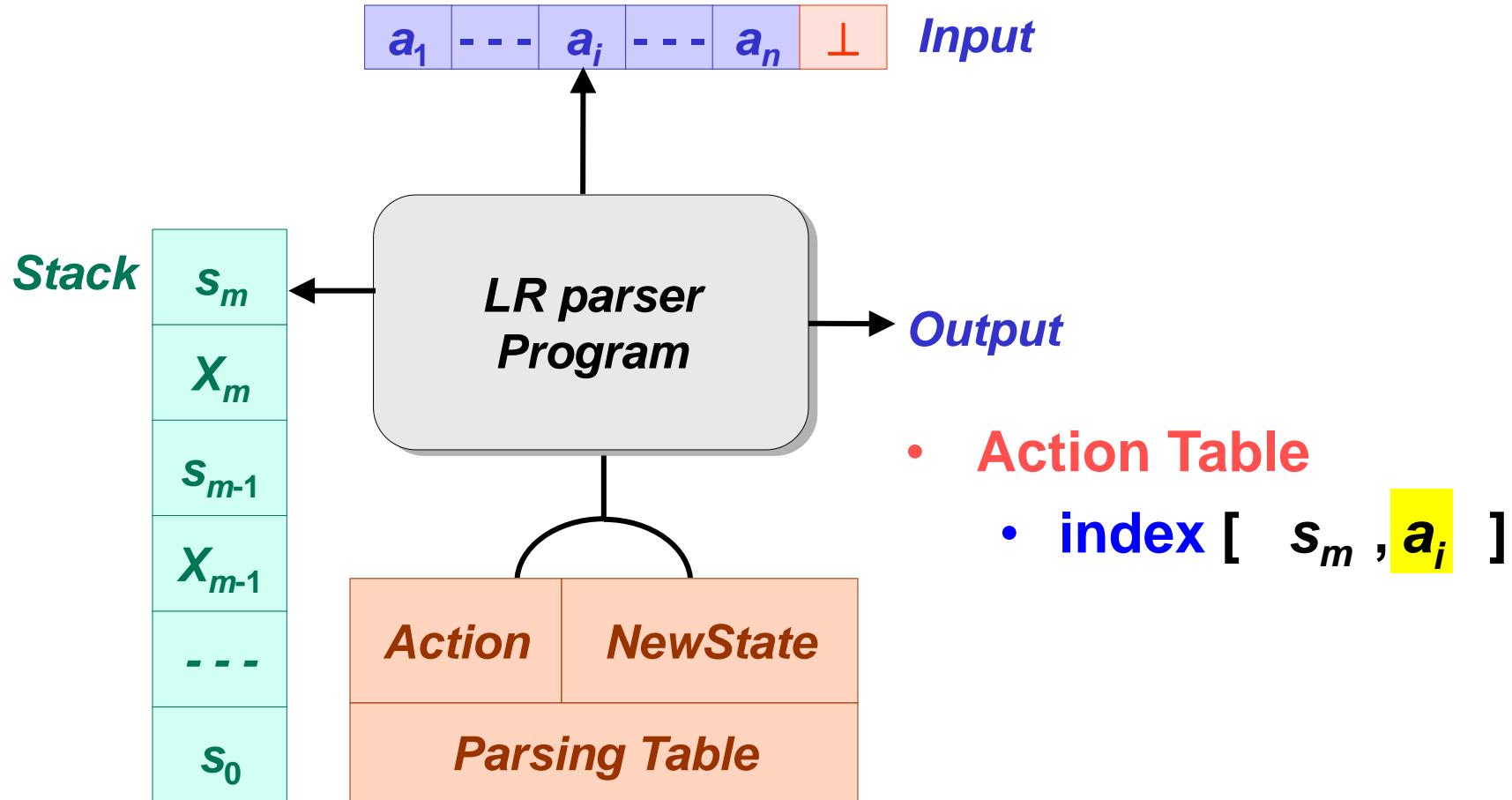
LR parsing



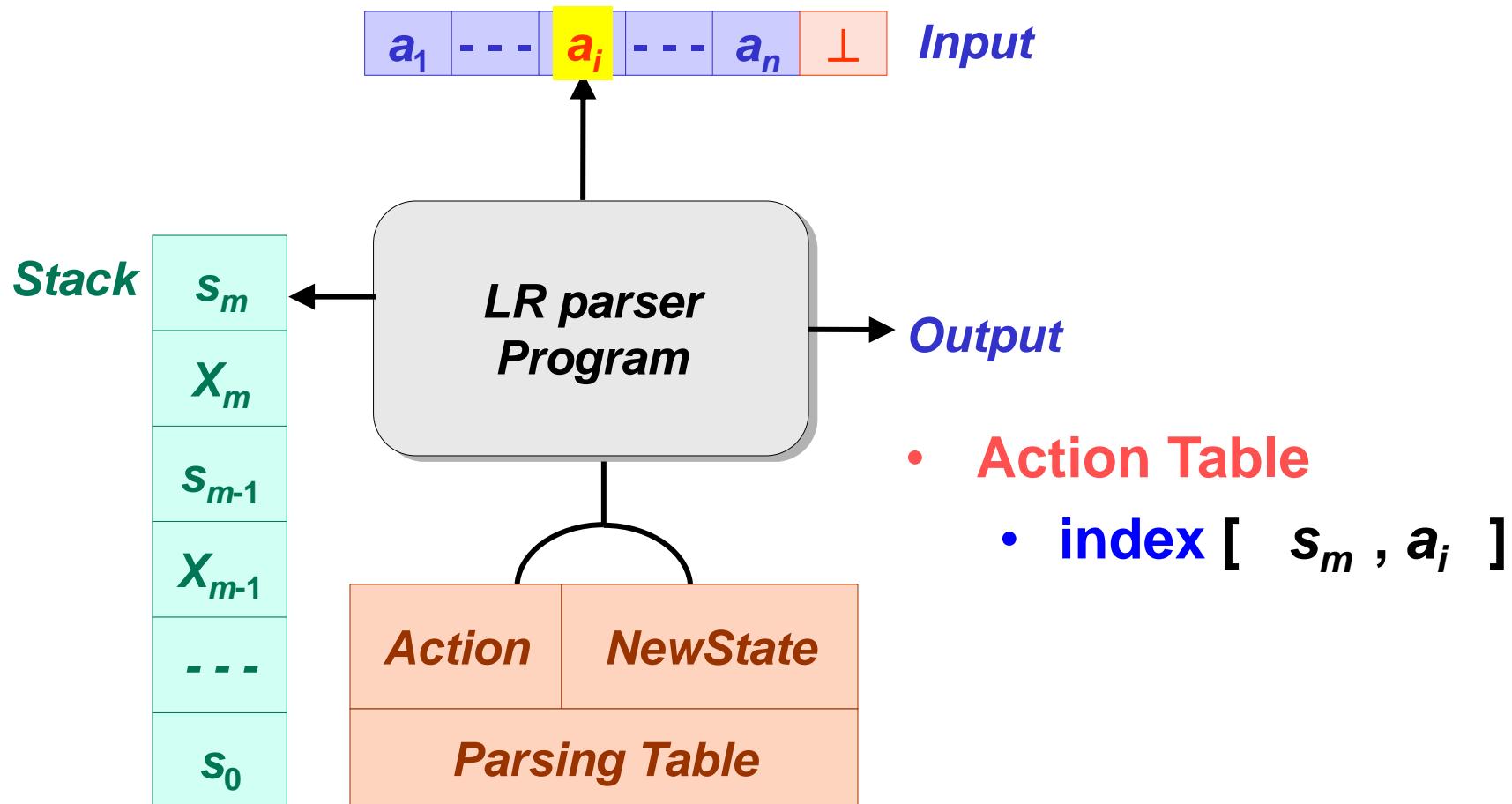
LR parsing



LR parsing

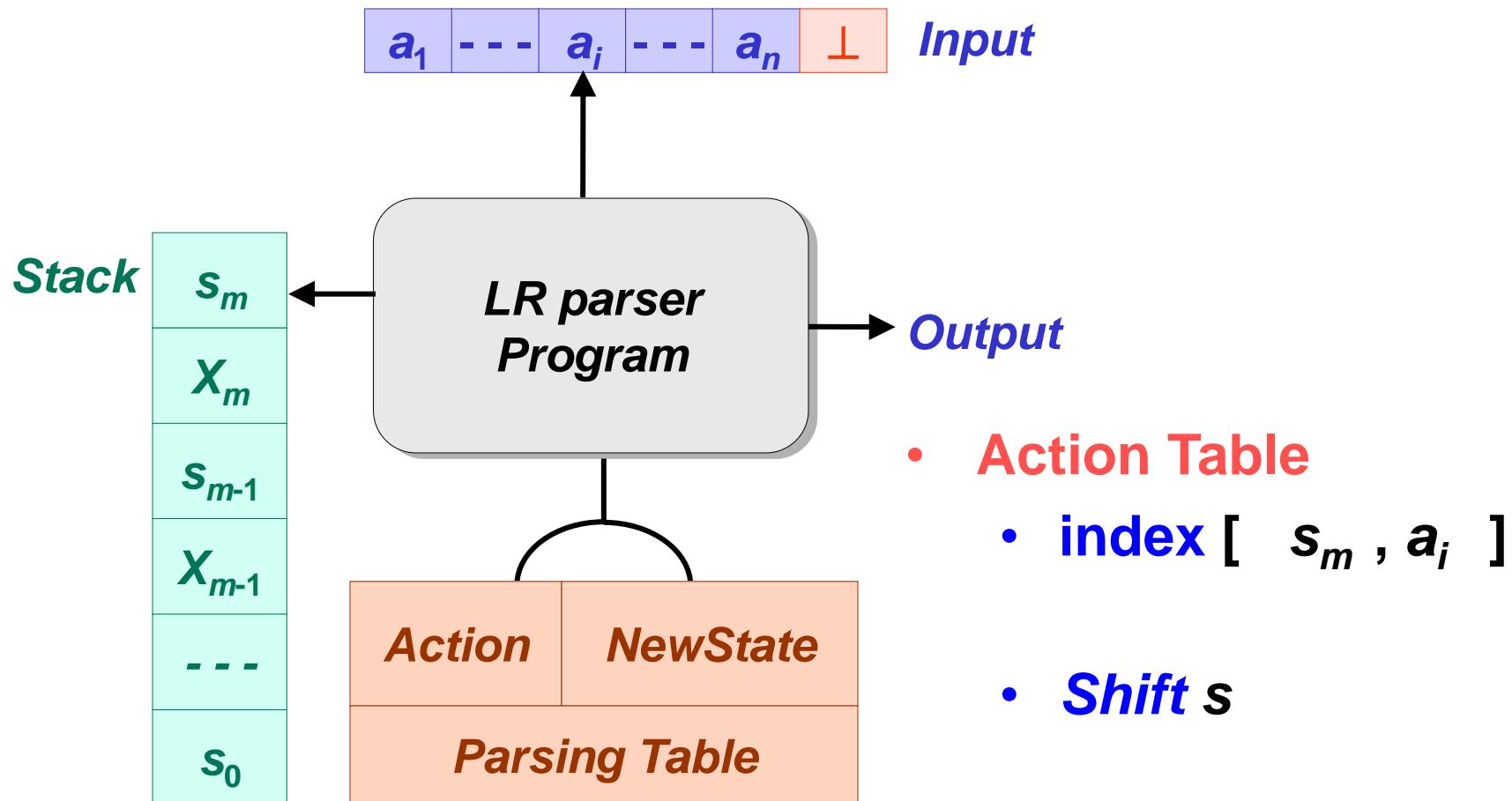


LR parsing

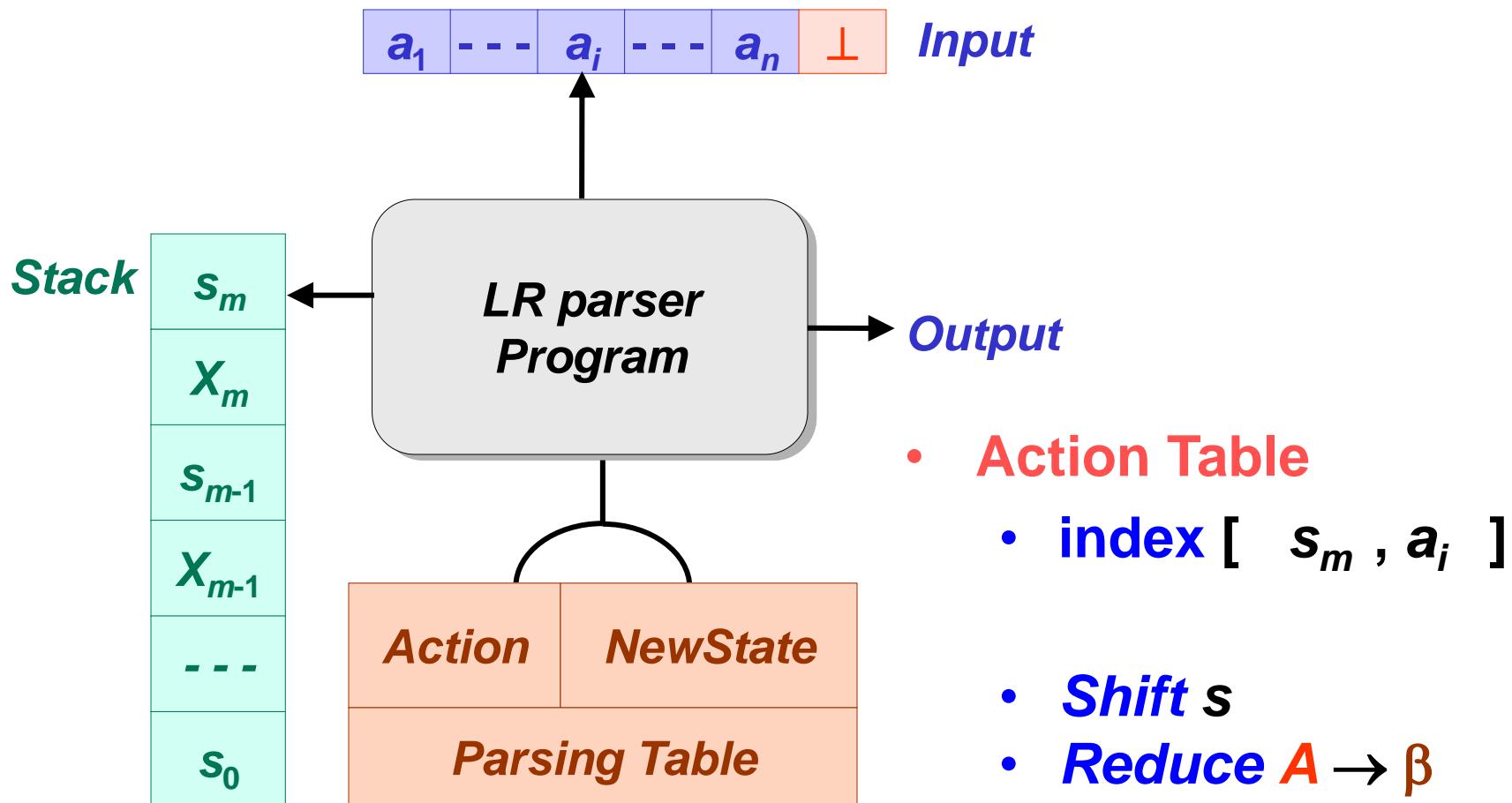


- **Action Table**
 - **index** [s_m, a_i]

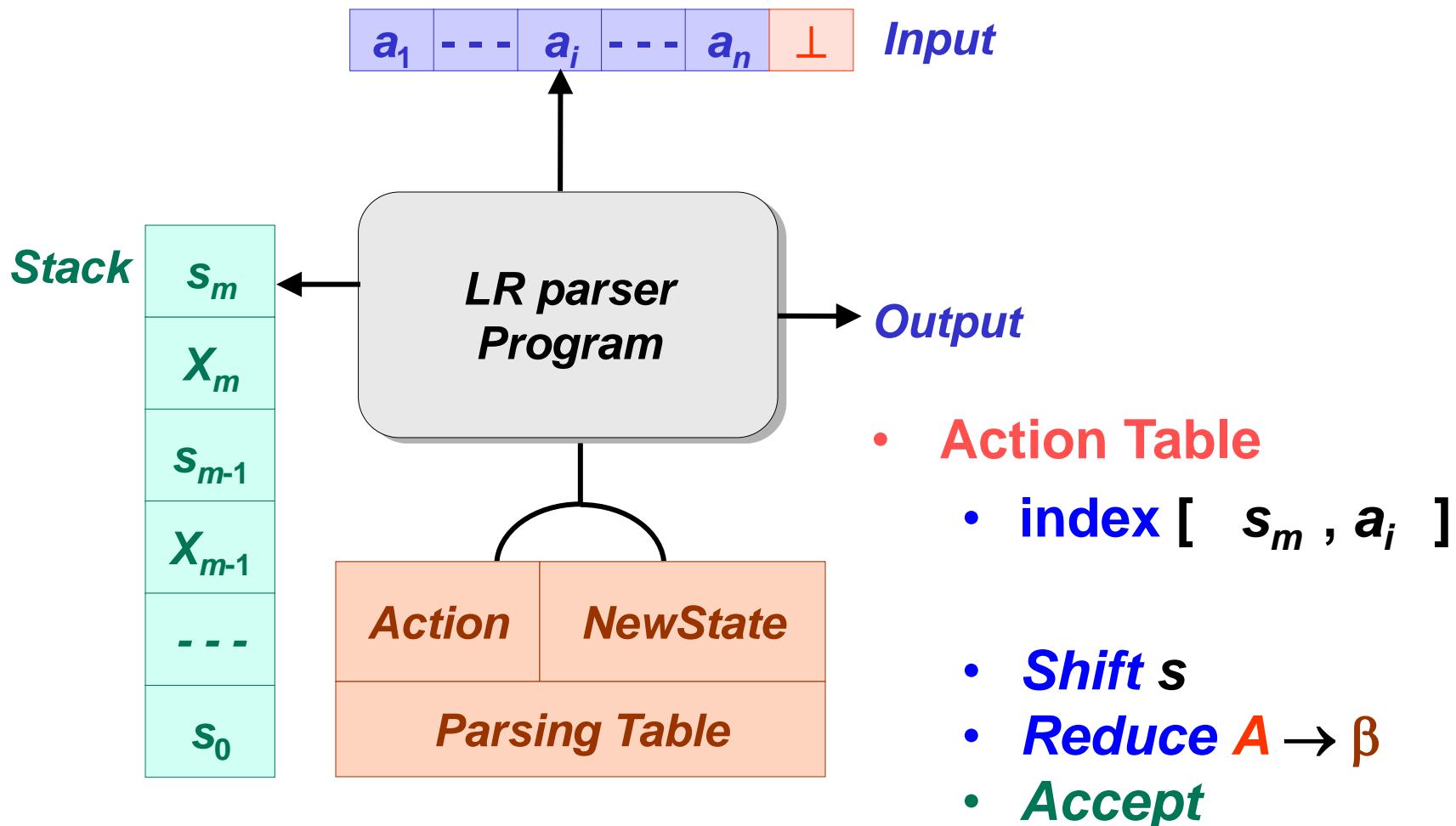
LR parsing



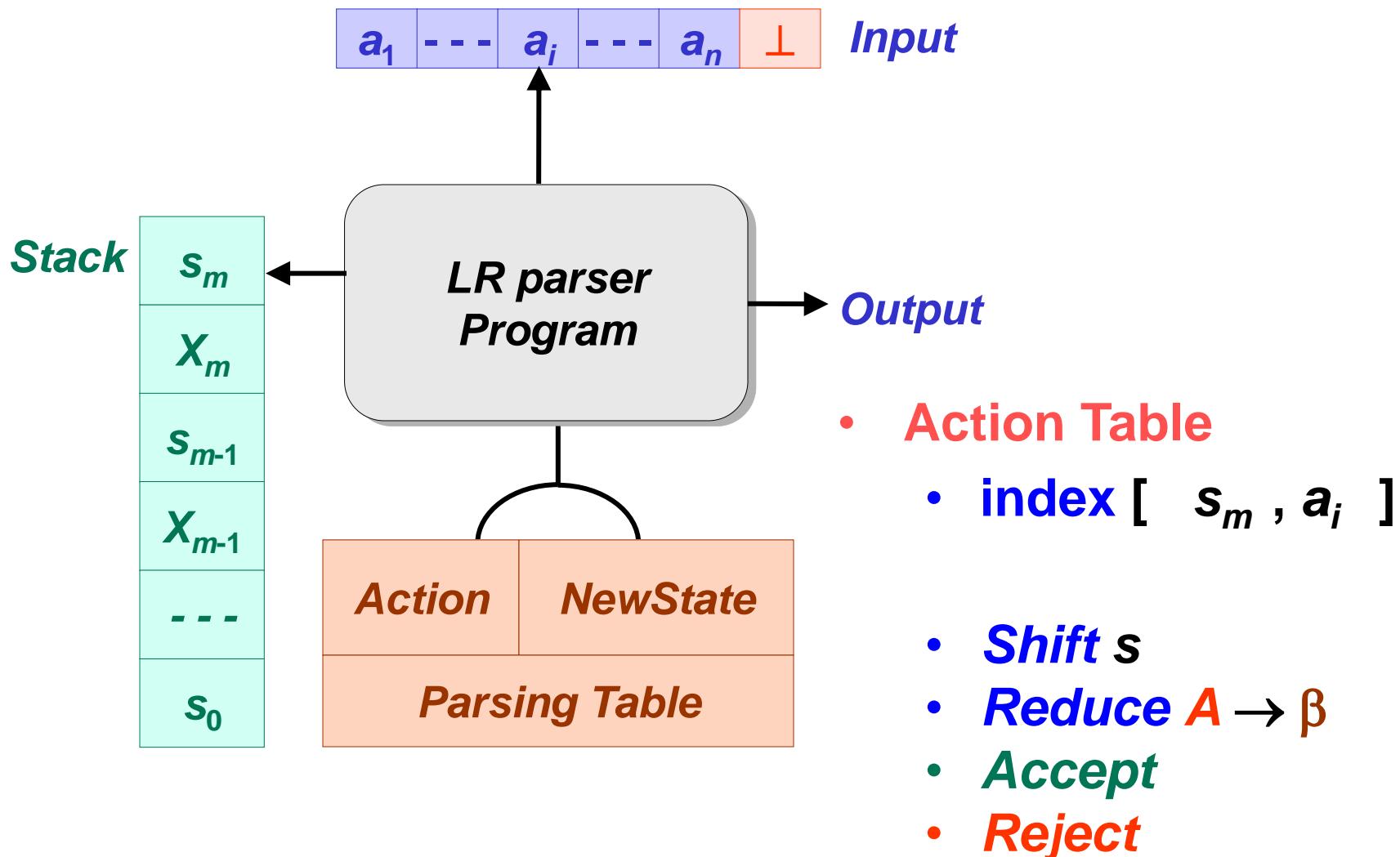
LR parsing



LR parsing

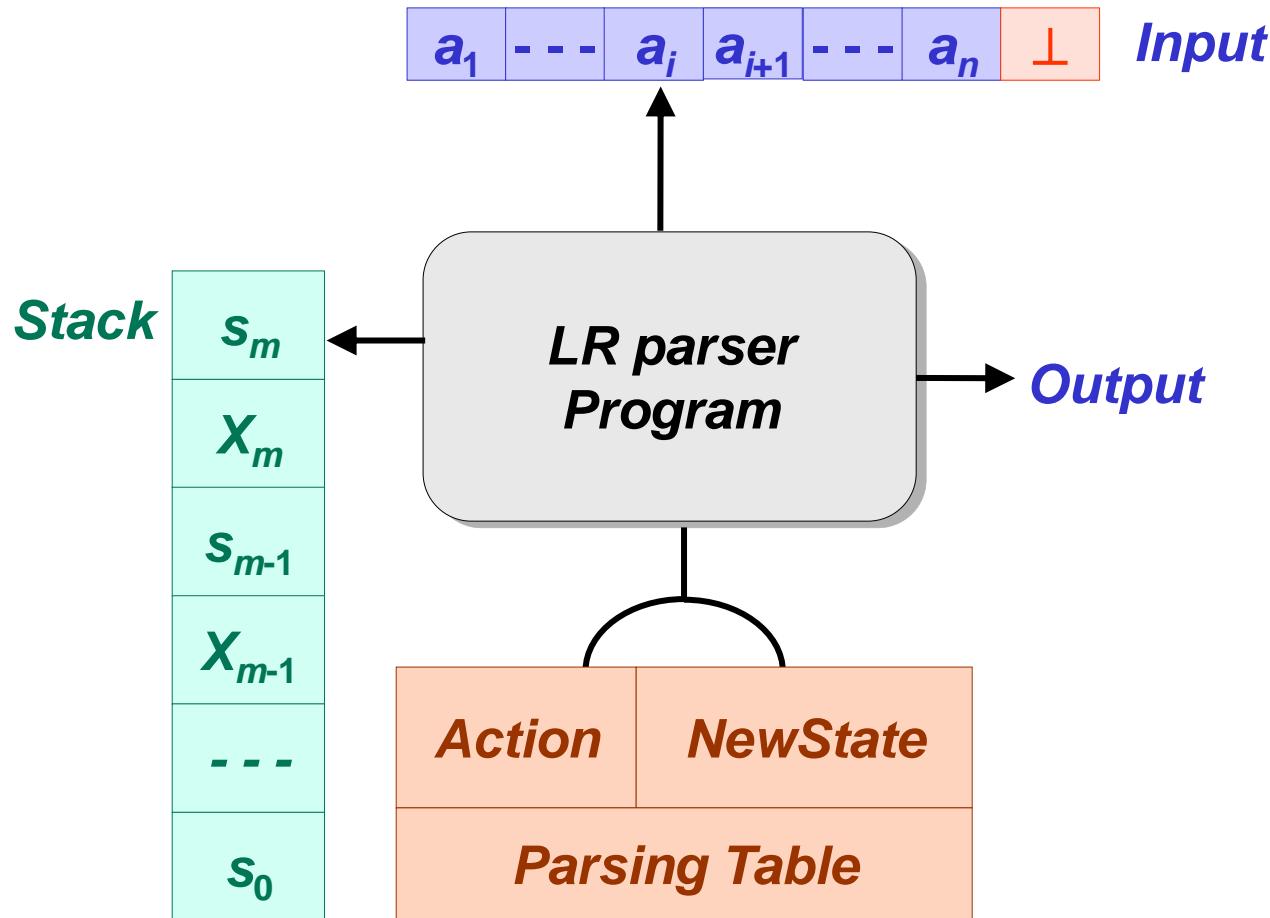


LR parsing

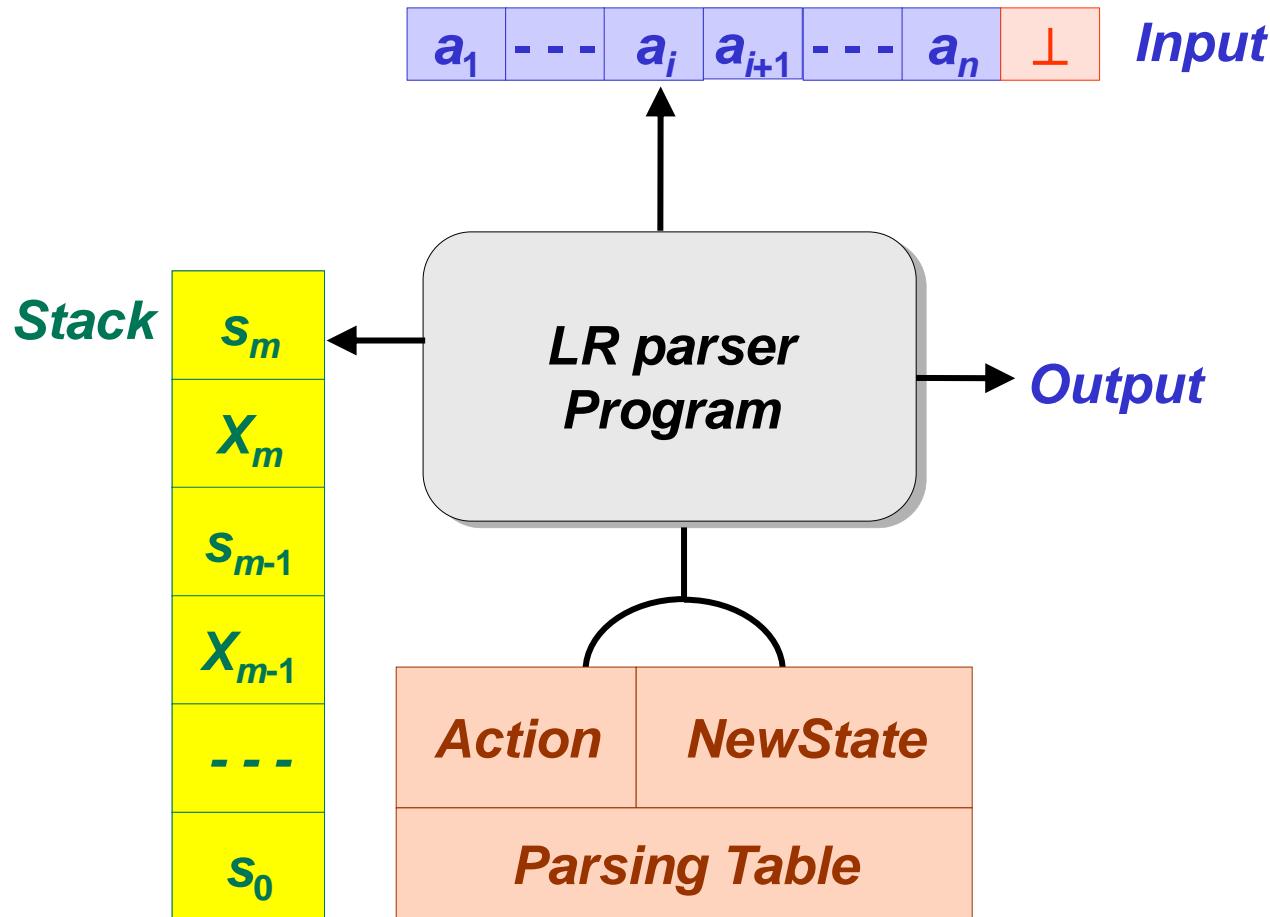


LR parsing

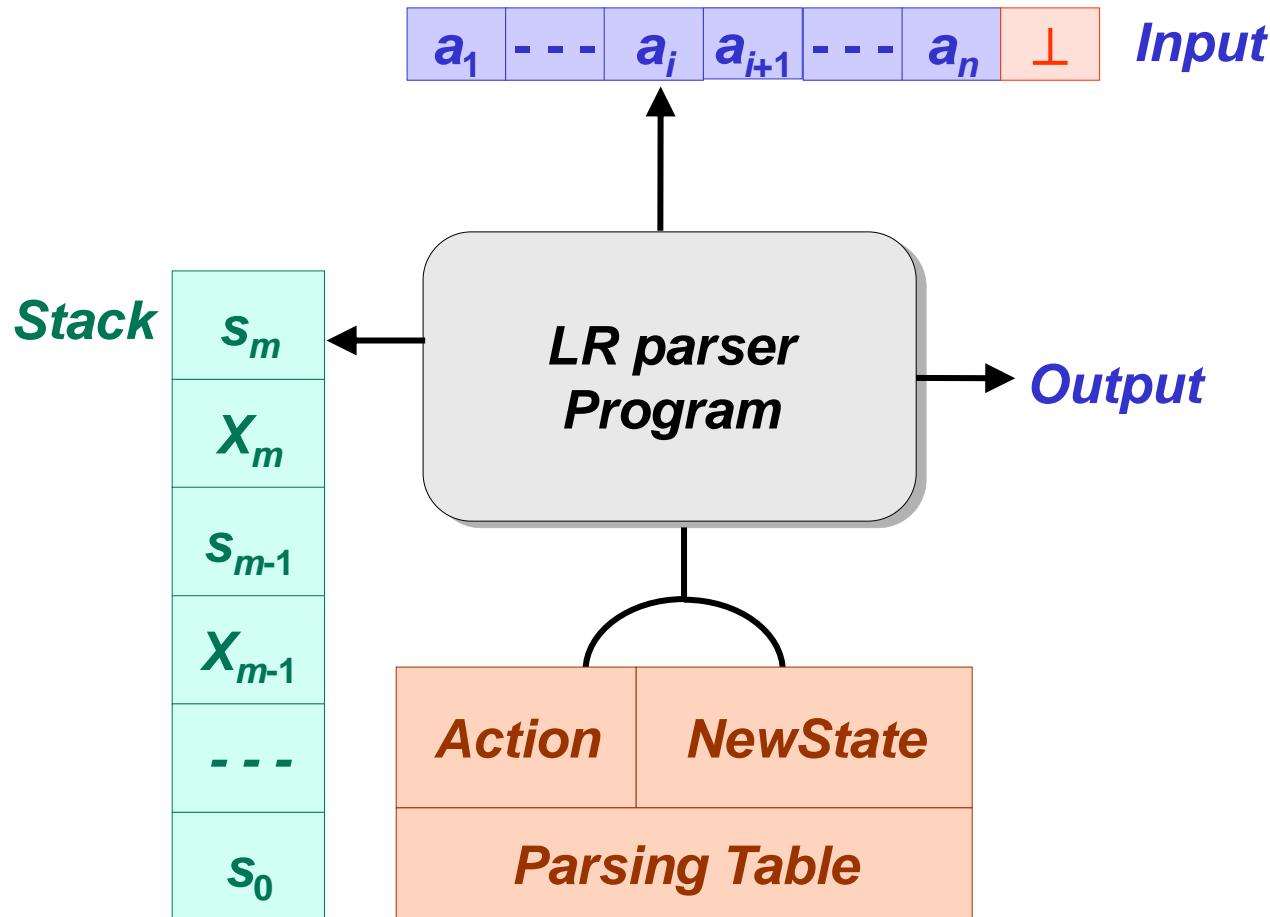
LR parsing



LR parsing

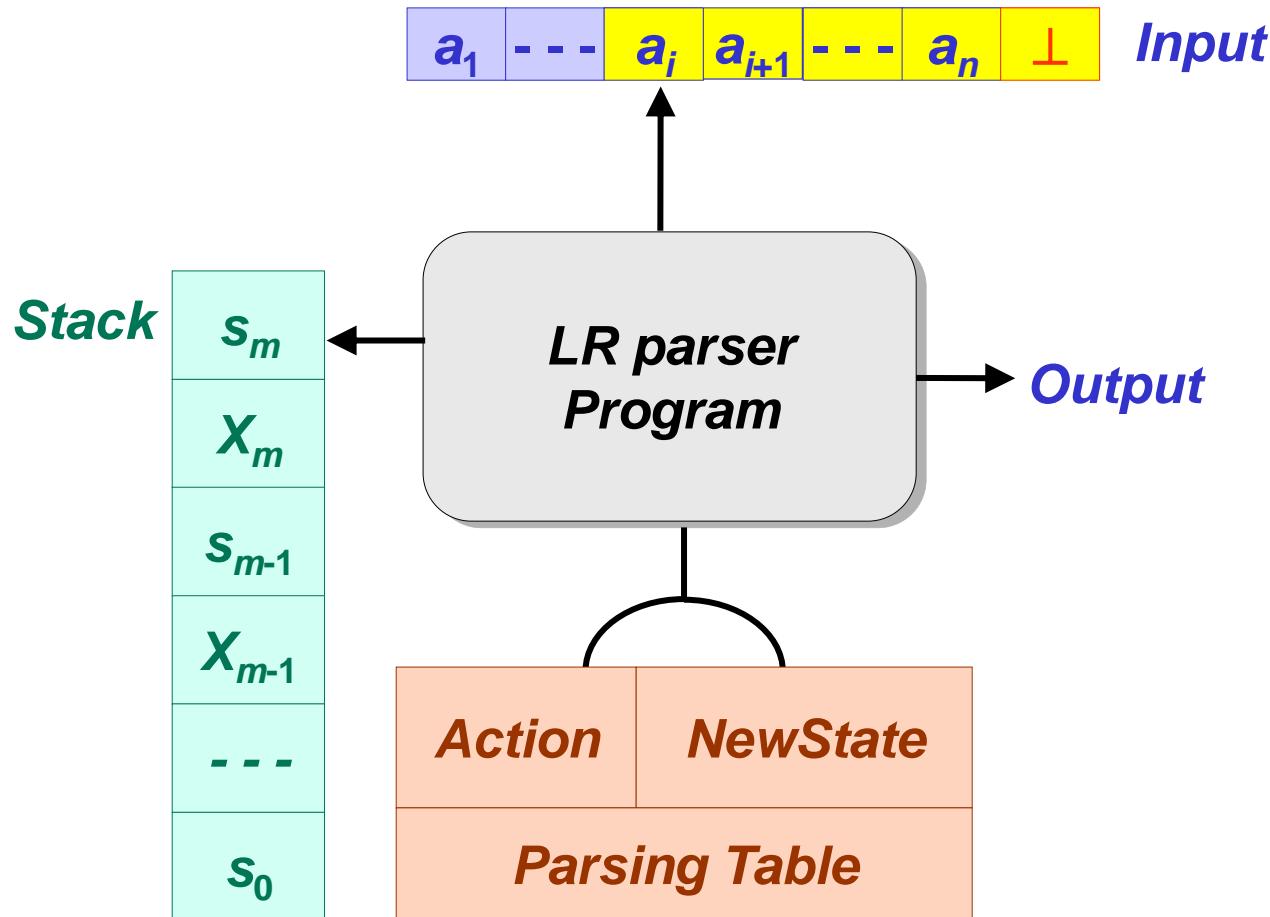


LR parsing



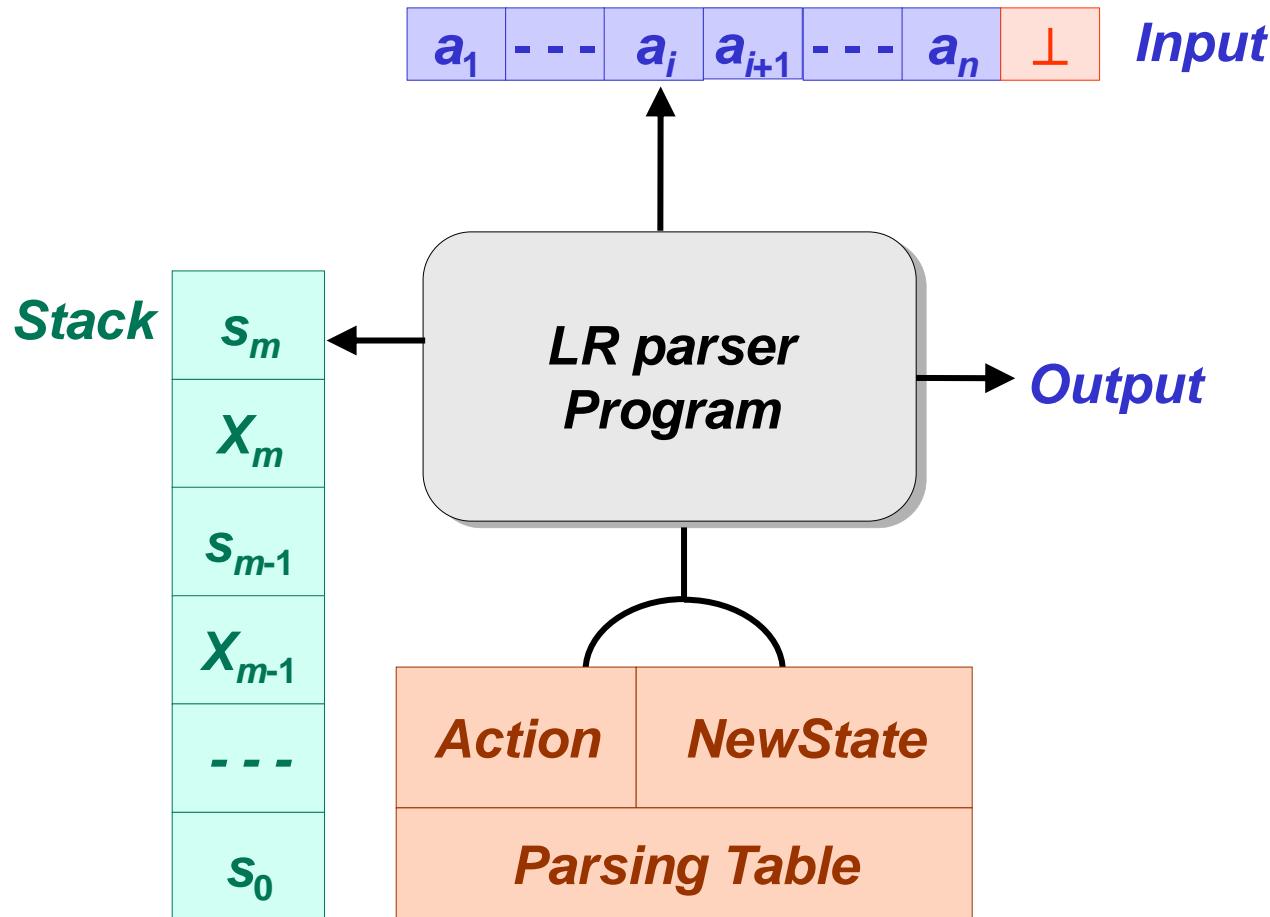
($s_0 \ X_1 \ s_1 \ X_2 \ s_2 \dots \ X_m \ s_m$,

LR parsing



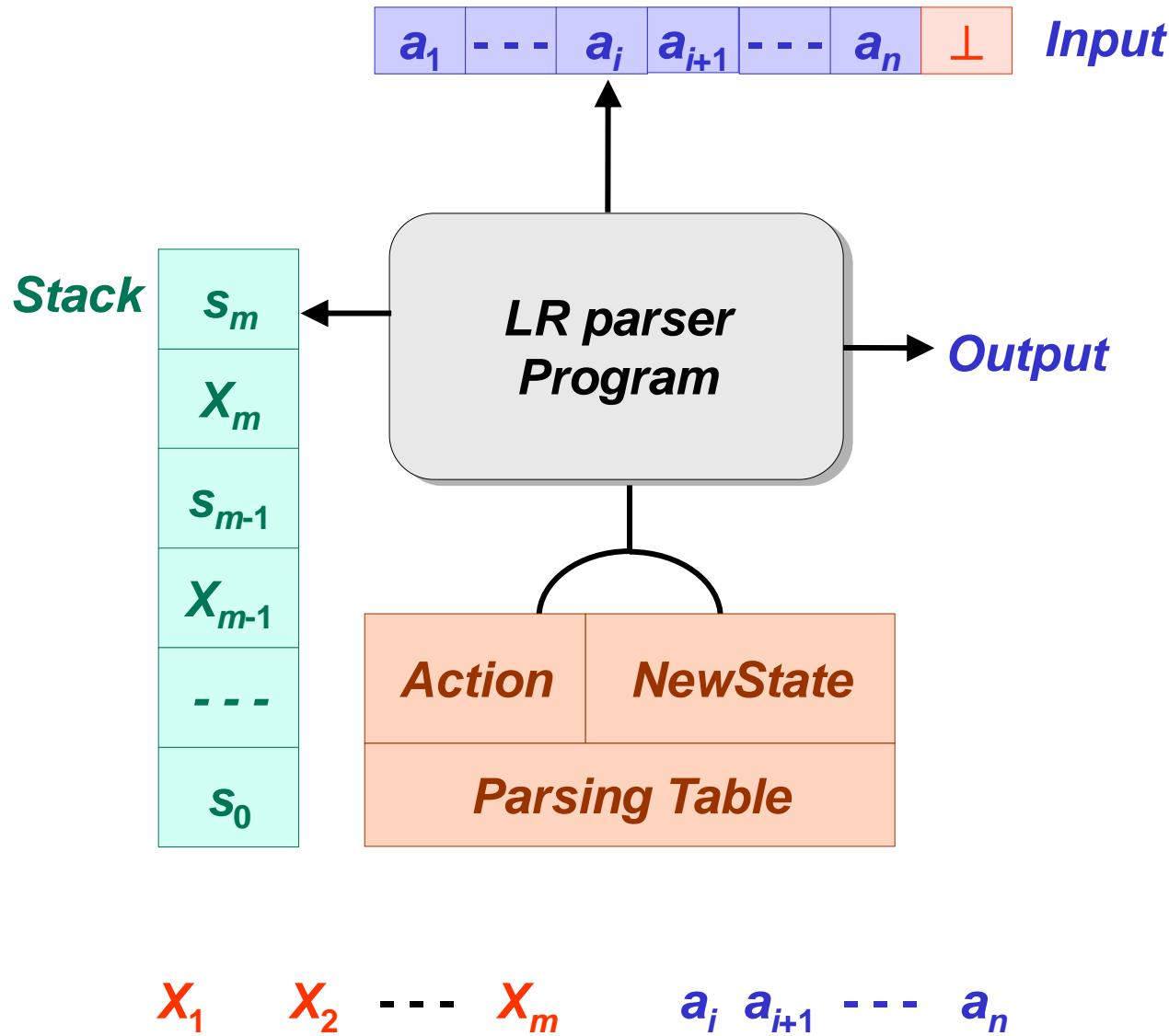
($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m$,

LR parsing

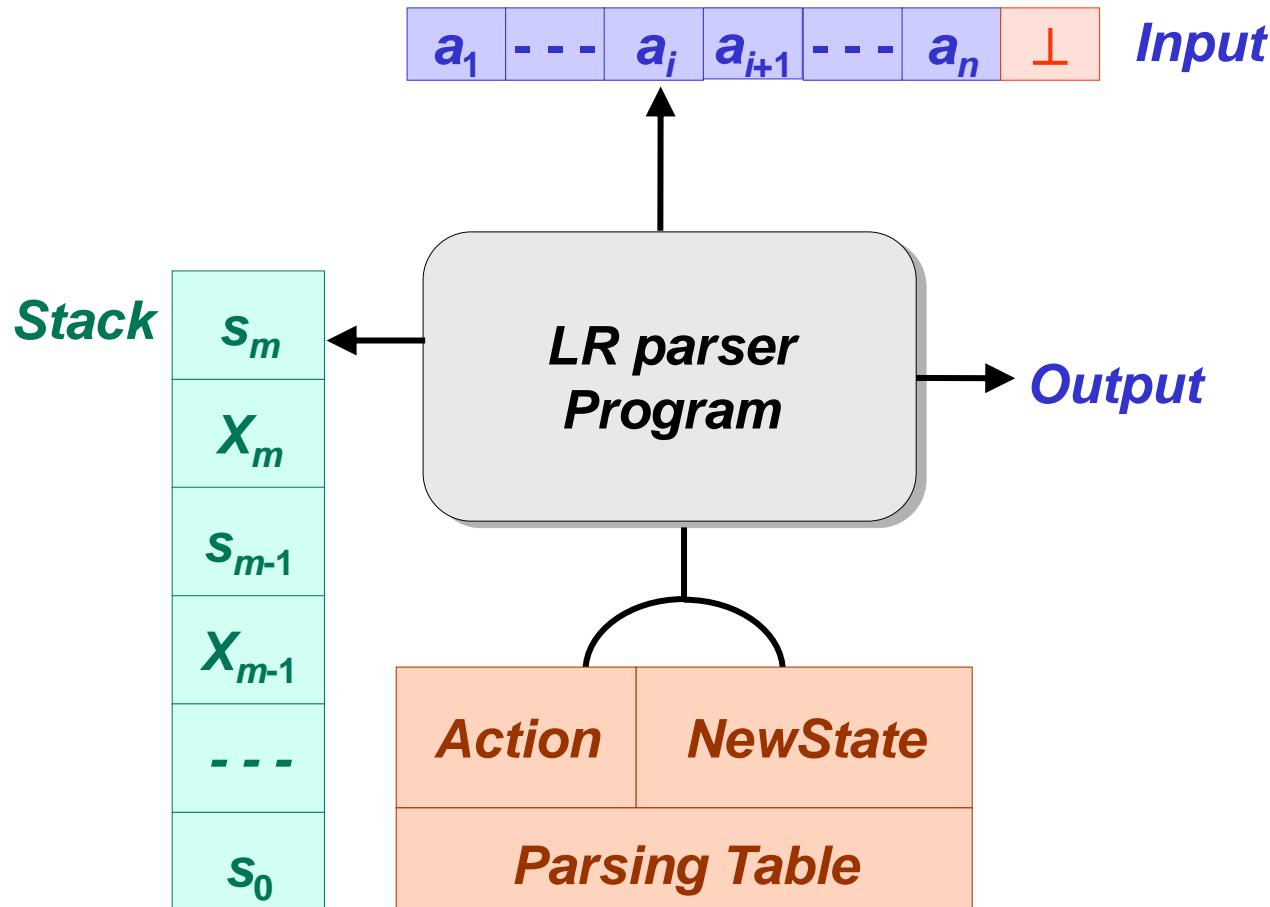


($s_0 \ X_1 \ s_1 \ X_2 \ s_2 \dots \ X_m \ s_m$, $a_i \ a_{i+1} \dots \ a_n \perp$)

LR parsing

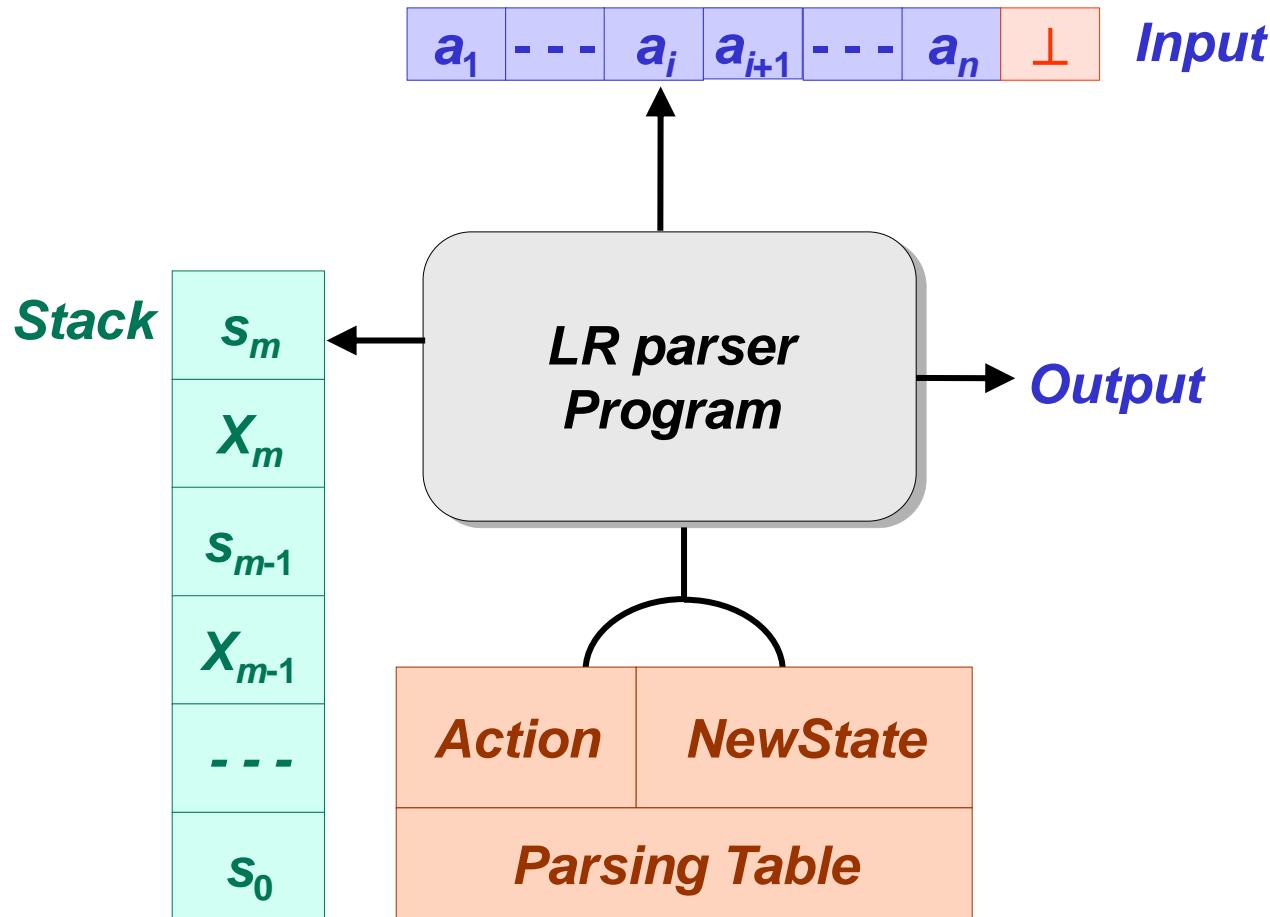


LR parsing



$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m , a_i a_{i+1} \dots a_n \perp)$

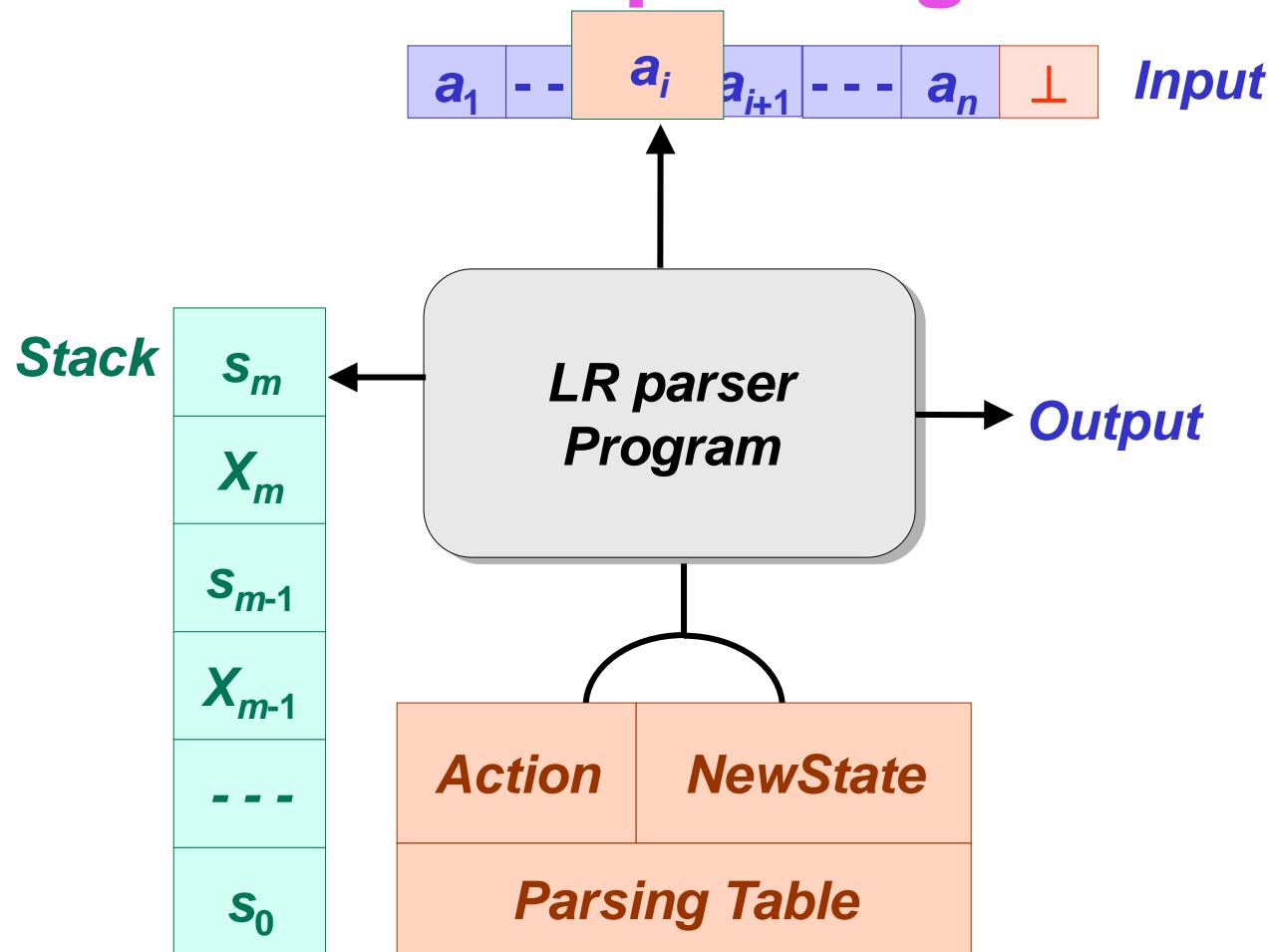
LR parsing



($s_0 \ X_1 \ s_1 \ X_2 \ s_2 \ \dots \ X_m \ s_m$, $a_i \ a_{i+1} \ \dots \ a_n \perp$)

Action [s_m, a_i] = **Shift** s

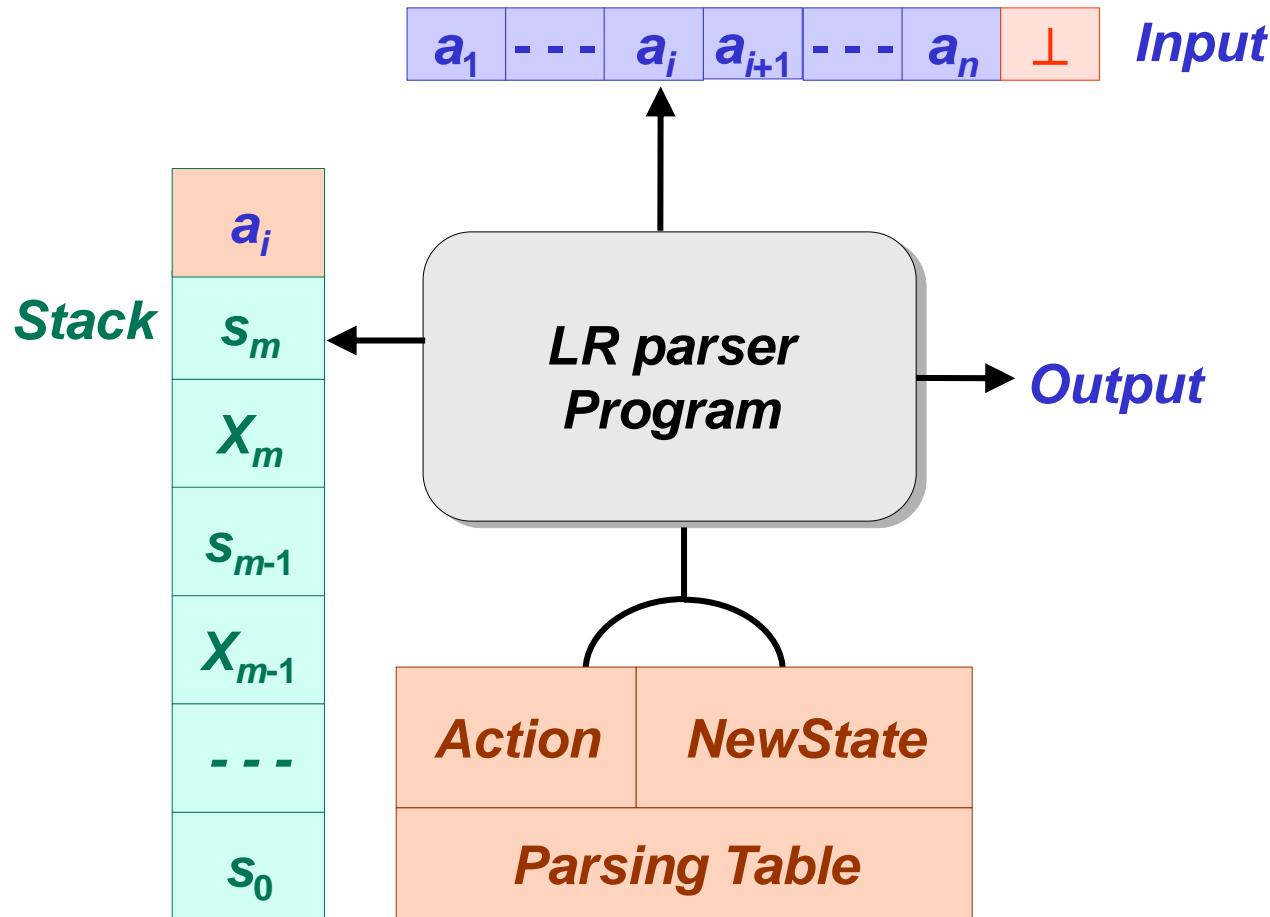
LR parsing



$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp)$

Action [s_m, a_i] = **Shift** s

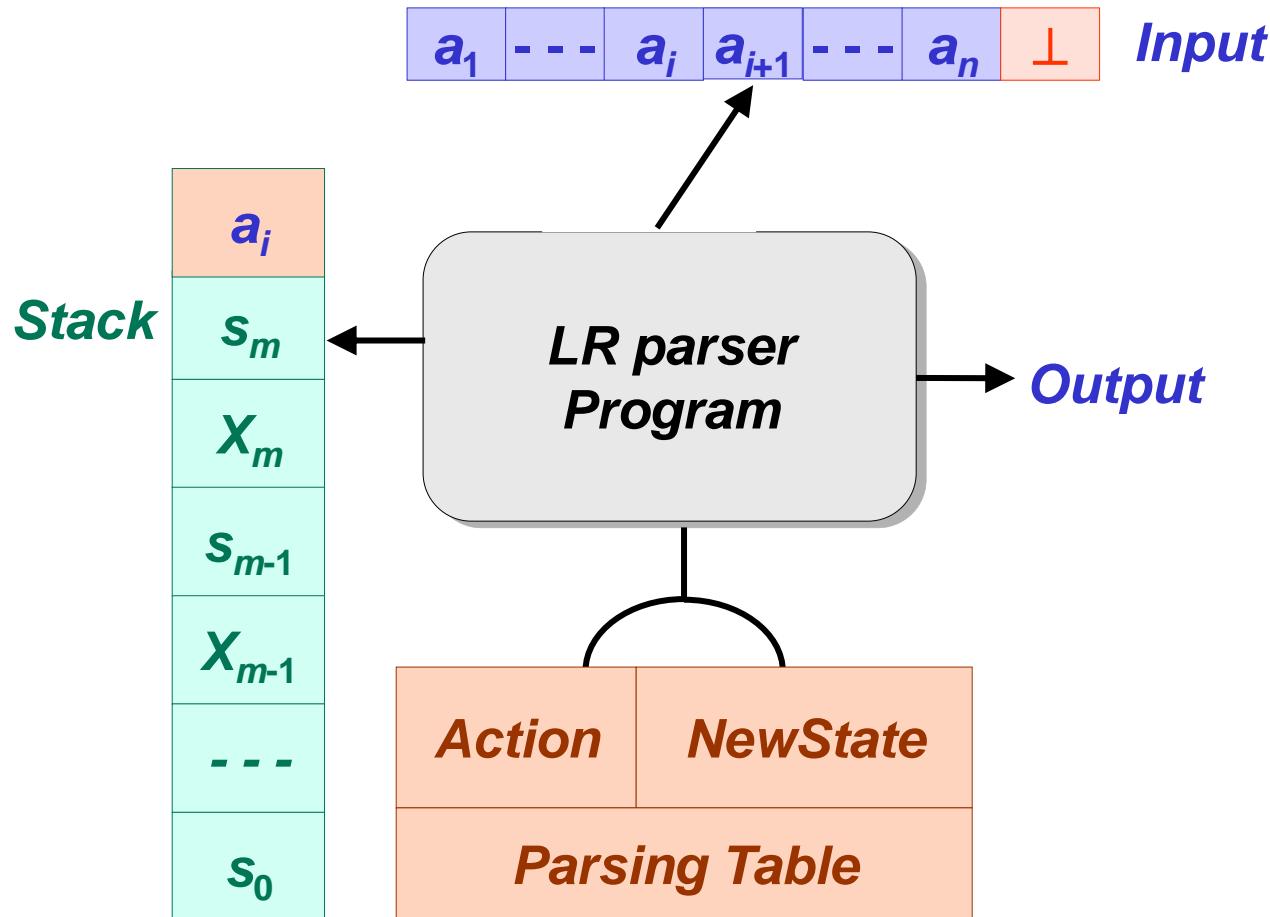
LR parsing



($s_0 \ X_1 \ s_1 \ X_2 \ s_2 \dots \ X_m \ s_m$, $a_i \ a_{i+1} \dots \ a_n \perp$)

Action [s_m, a_i] = **Shift** s

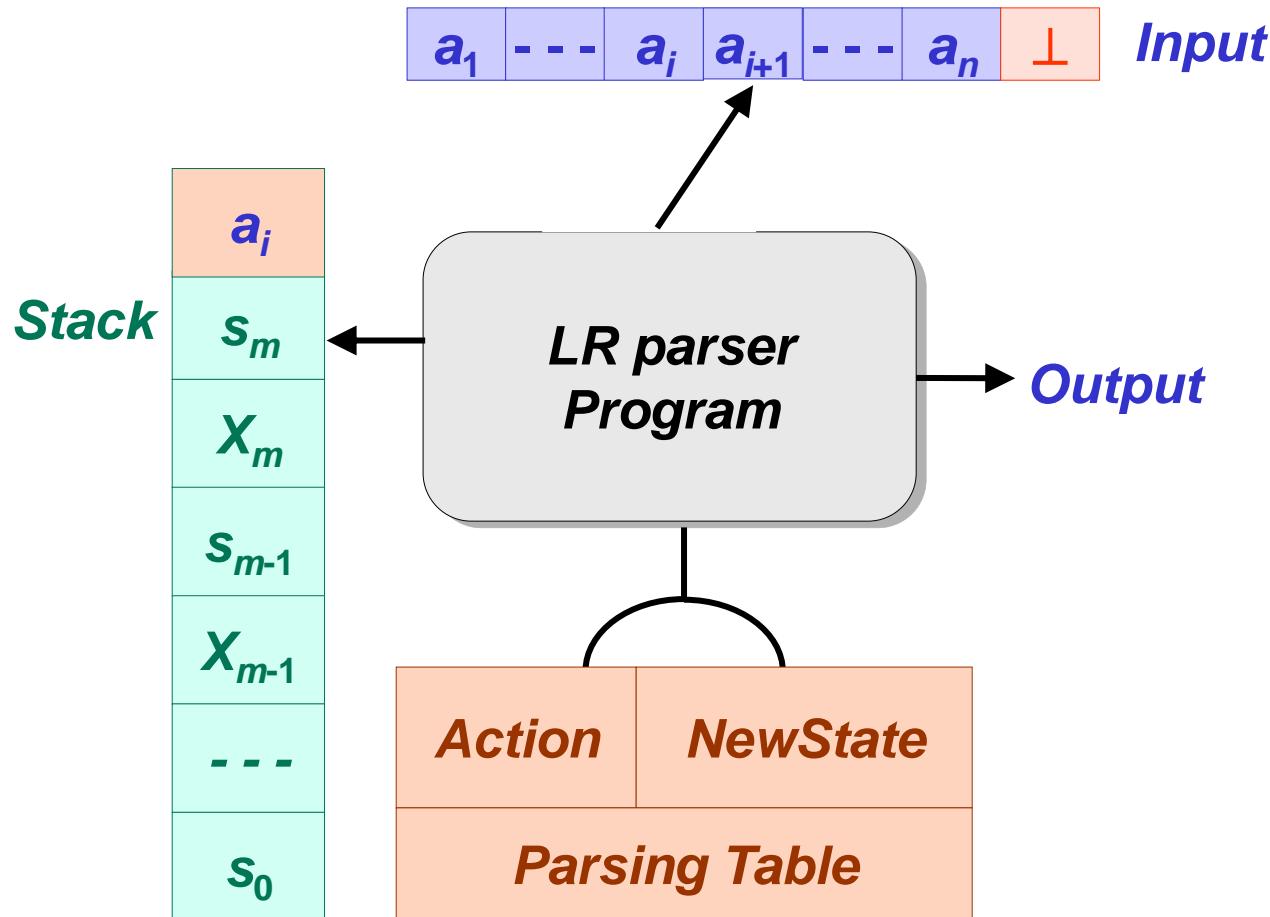
LR parsing



($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m$, $a_i a_{i+1} \dots a_n \perp$)

Action [s_m, a_i] = **Shift** s

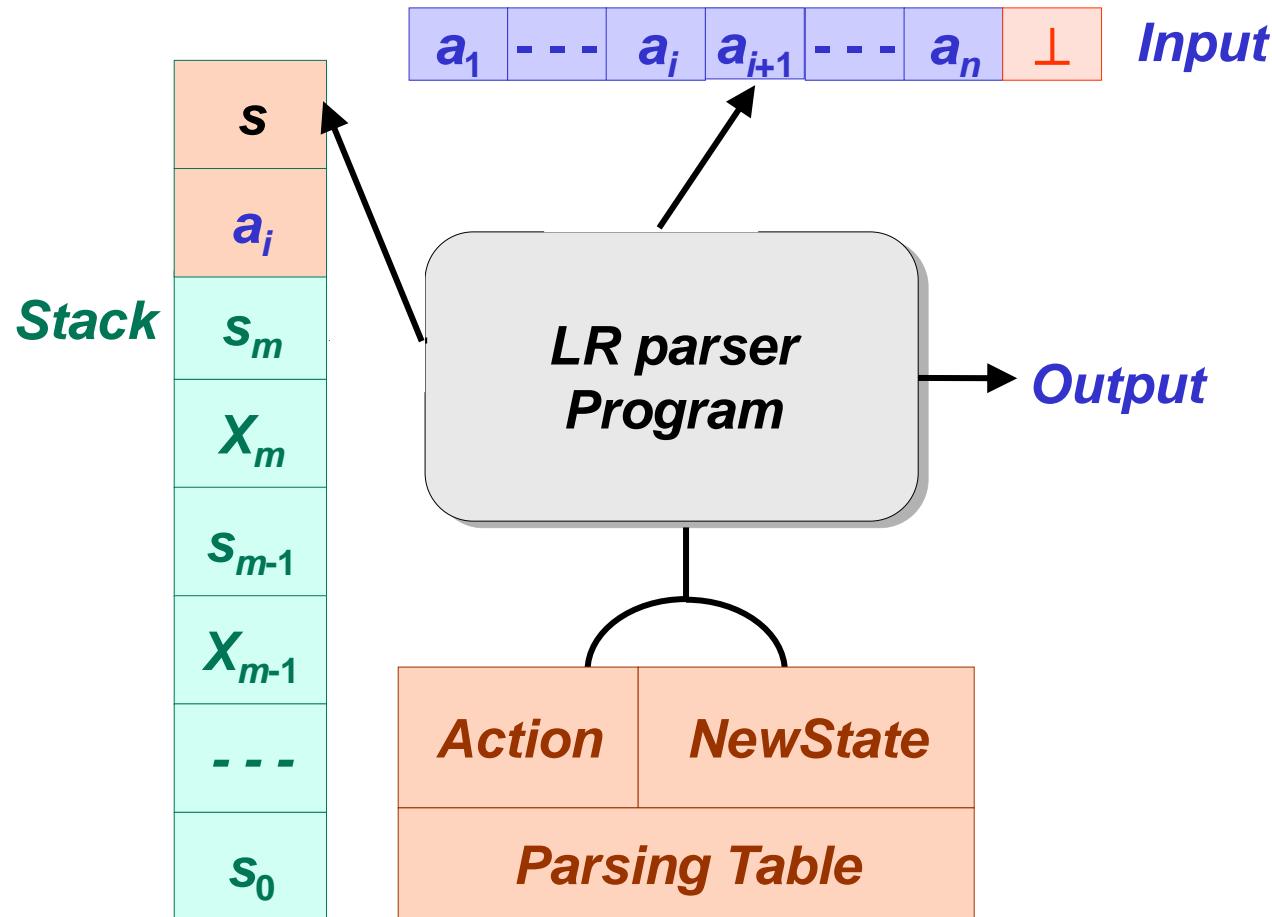
LR parsing



($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m$, $a_i a_{i+1} \dots a_n \perp$)

Action [s_m, a_i] = **Shift** s s

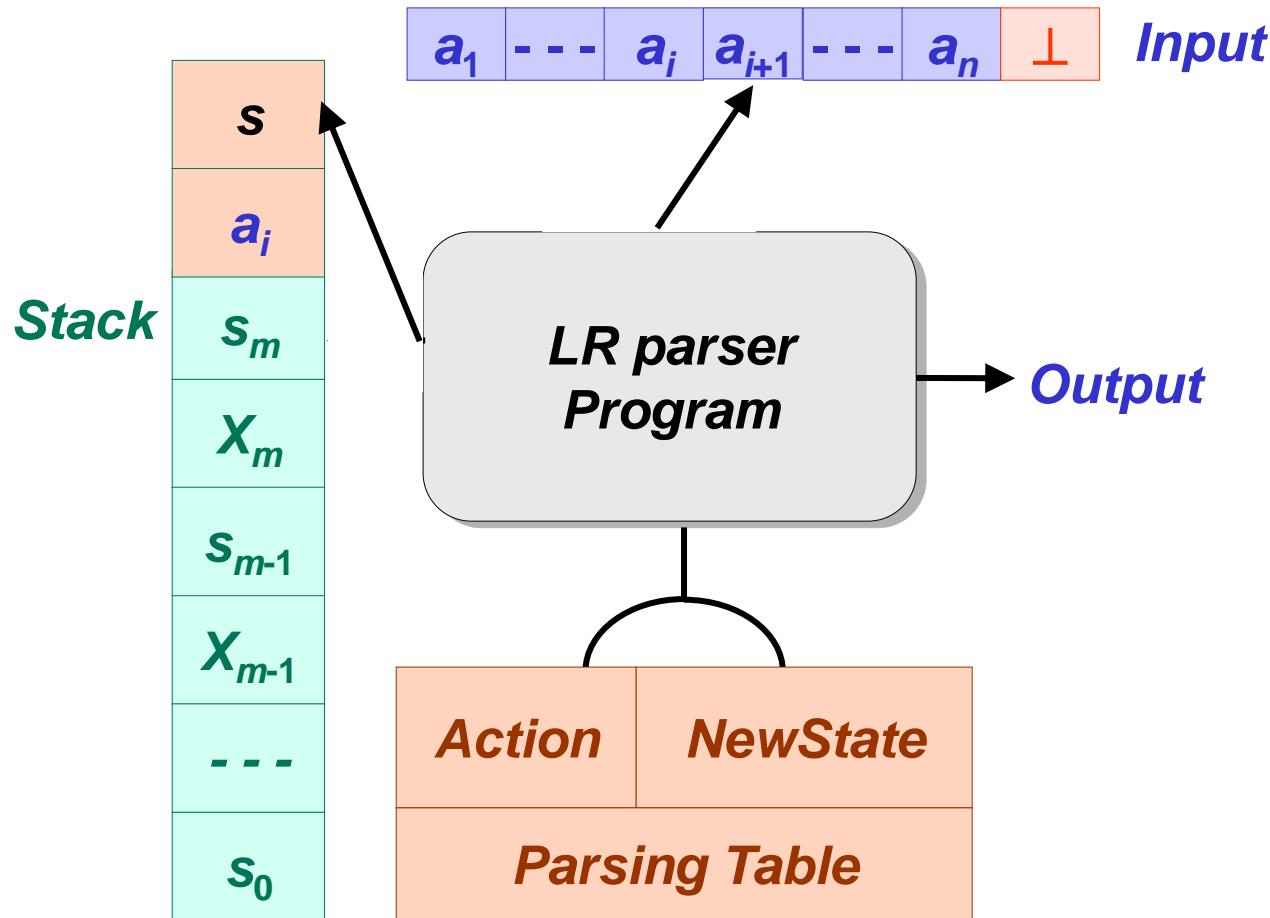
LR parsing



$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m , a_i a_{i+1} \dots a_n \perp)$

Action [s_m, a_i] = **Shift** s

LR parsing

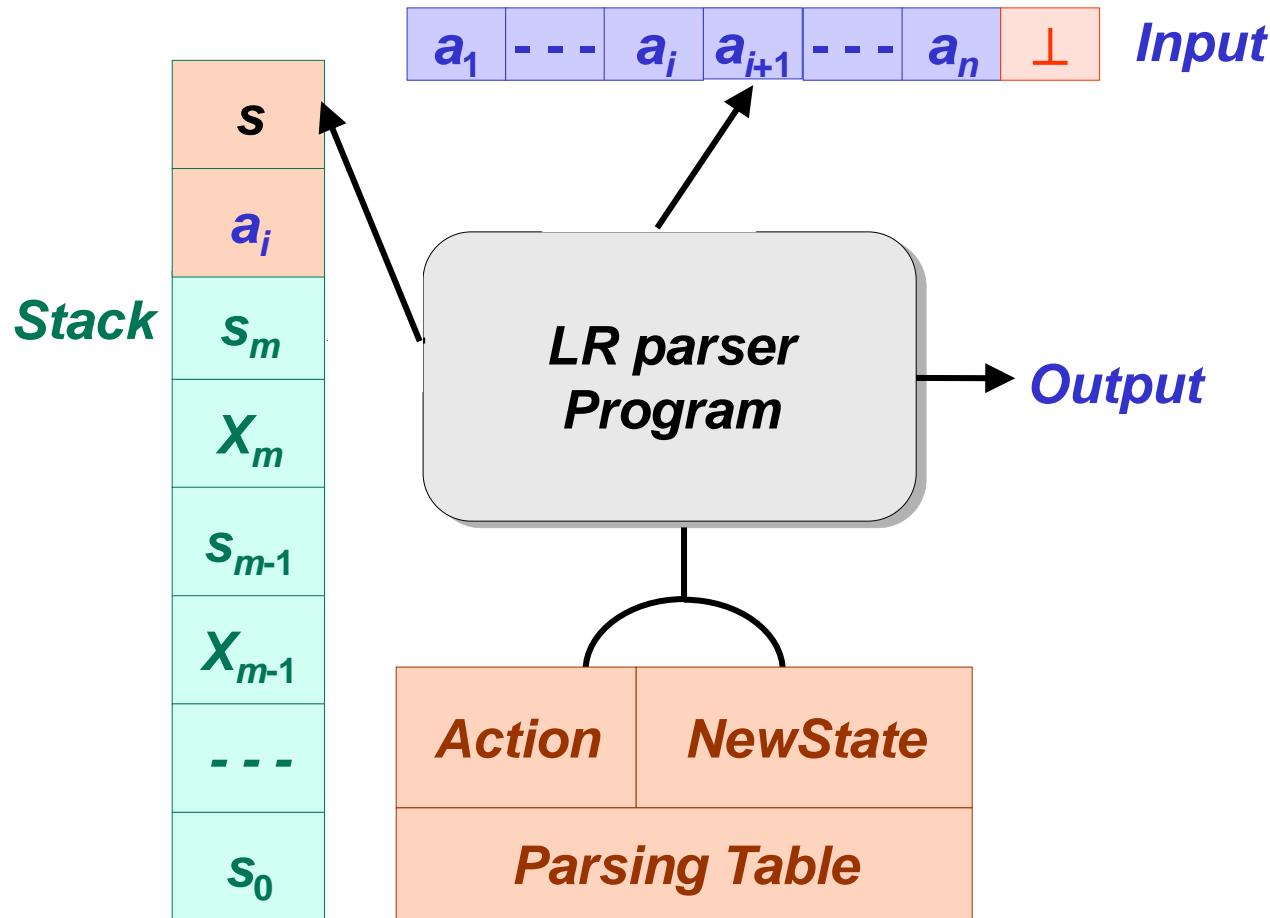


($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m$, $a_i a_{i+1} \dots a_n \perp$)

Action [s_m, a_i] = **Shift** s

($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m$

LR parsing

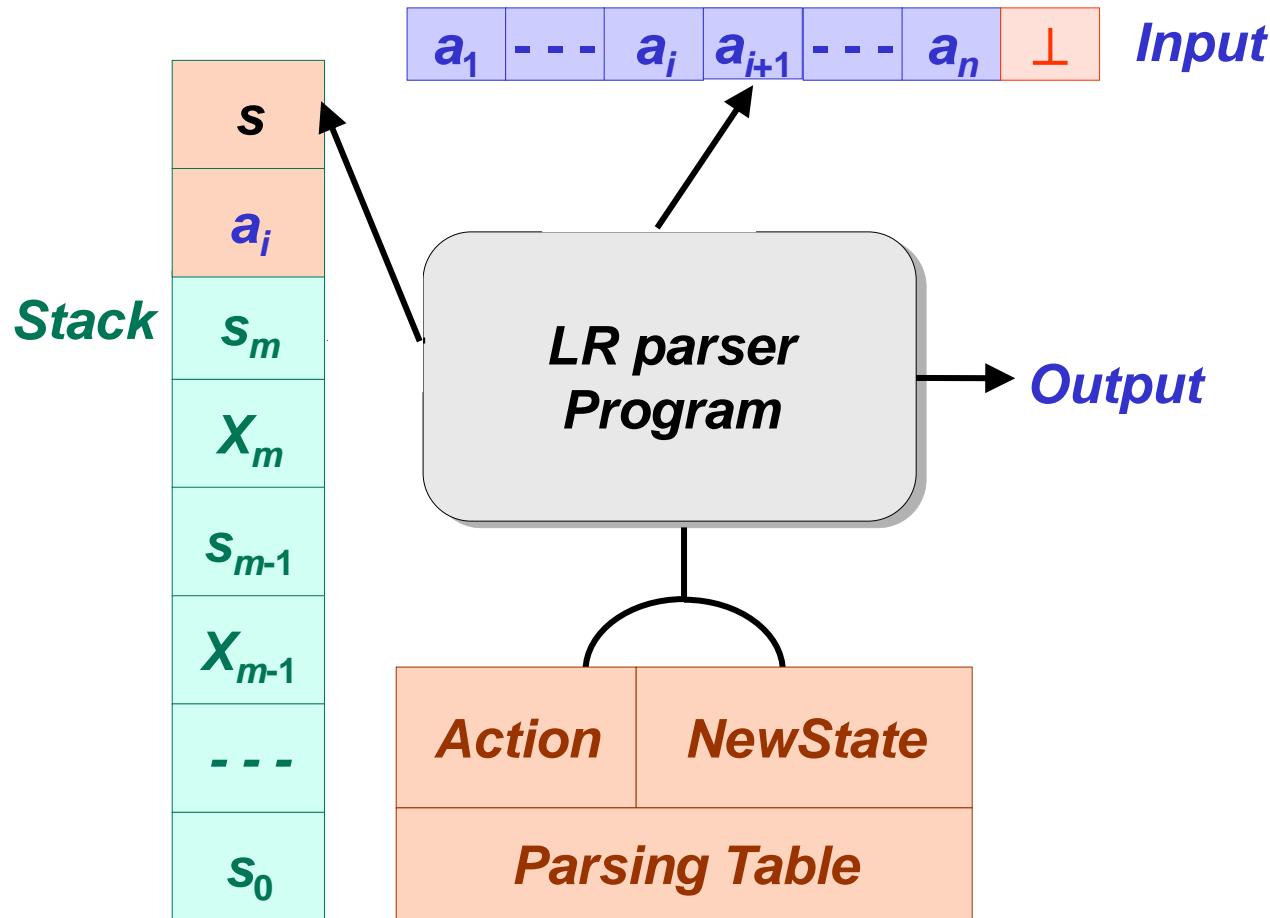


$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m , a_i a_{i+1} \dots a_n \perp)$

Action [s_m, a_i] = **Shift** s

$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m a_i , s ,$

LR parsing

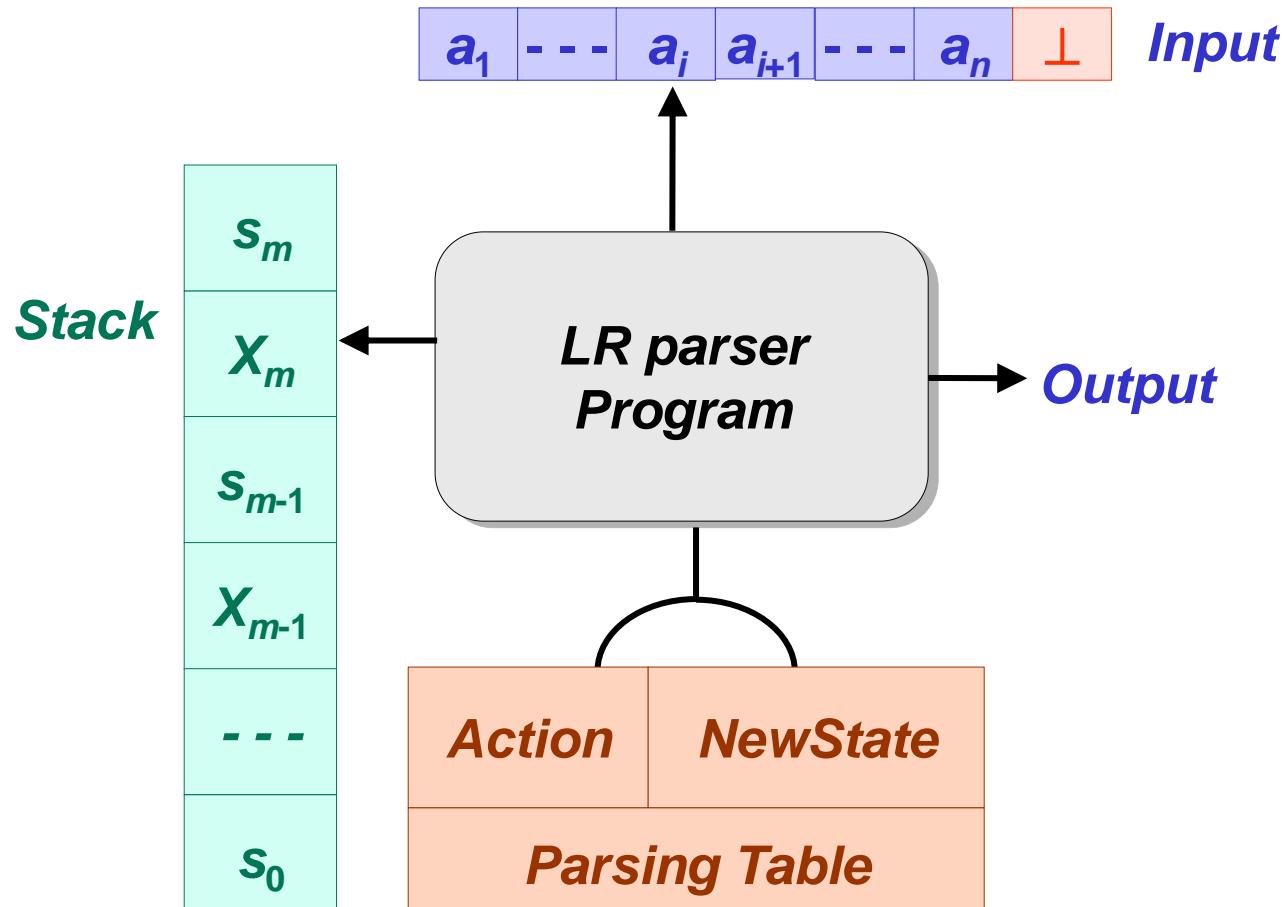


$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp)$

Action [s_m, a_i] = **Shift** s

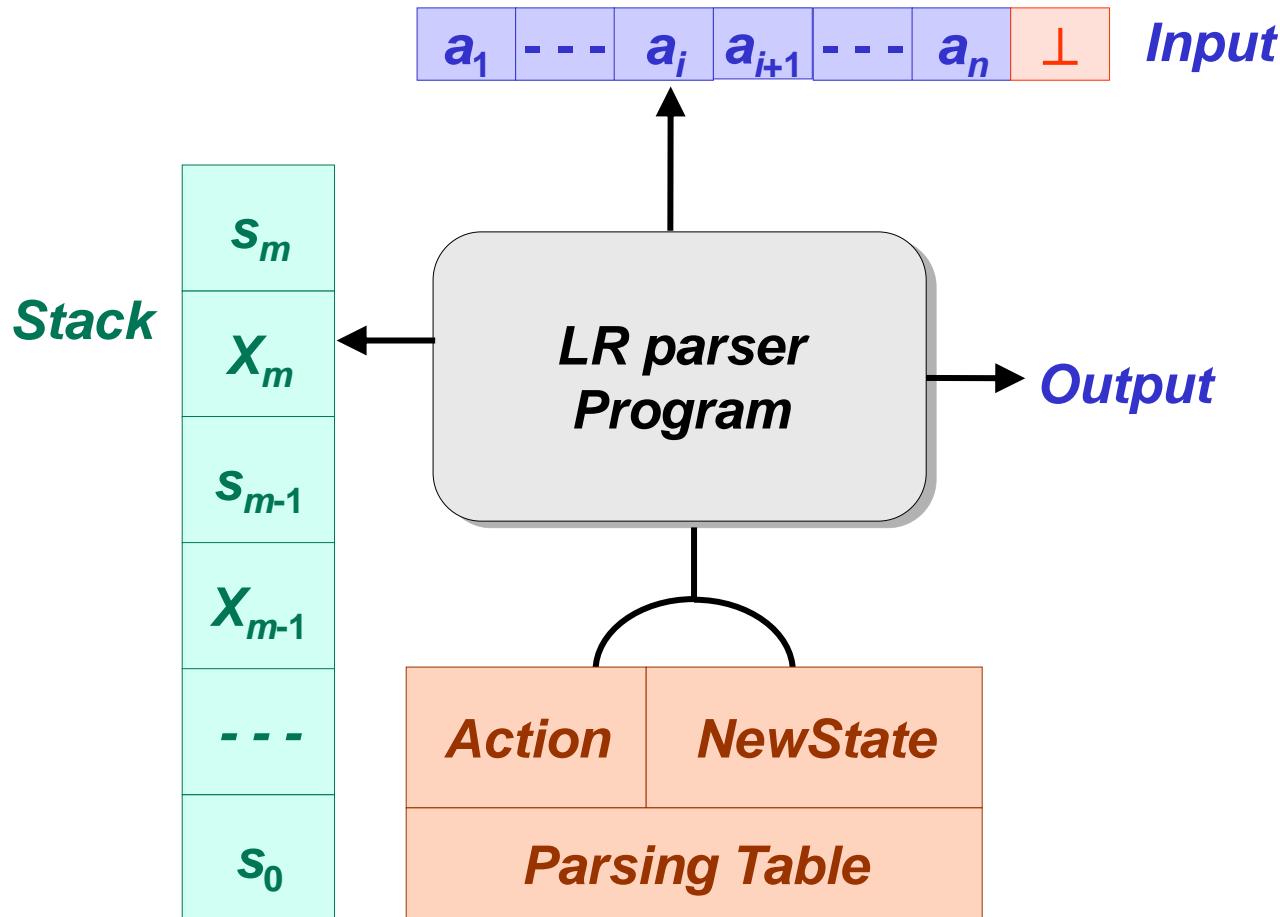
$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m a_i s, a_{i+1} \dots a_n \perp)$

LR parsing



$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m , a_i a_{i+1} \dots a_n \perp)$

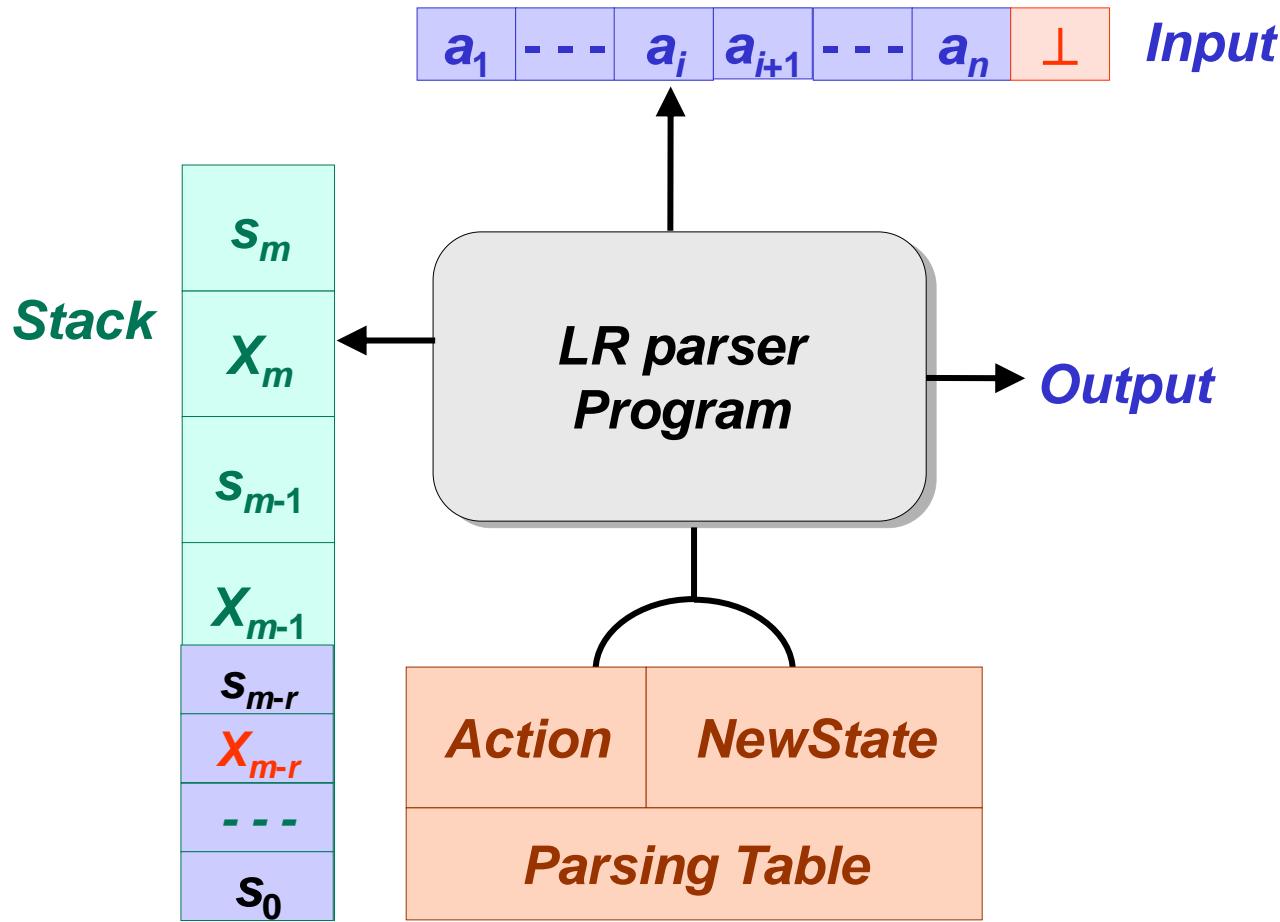
LR parsing



$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp)$

Action [s_m, a_i] = **Reduce** $A \rightarrow \beta$

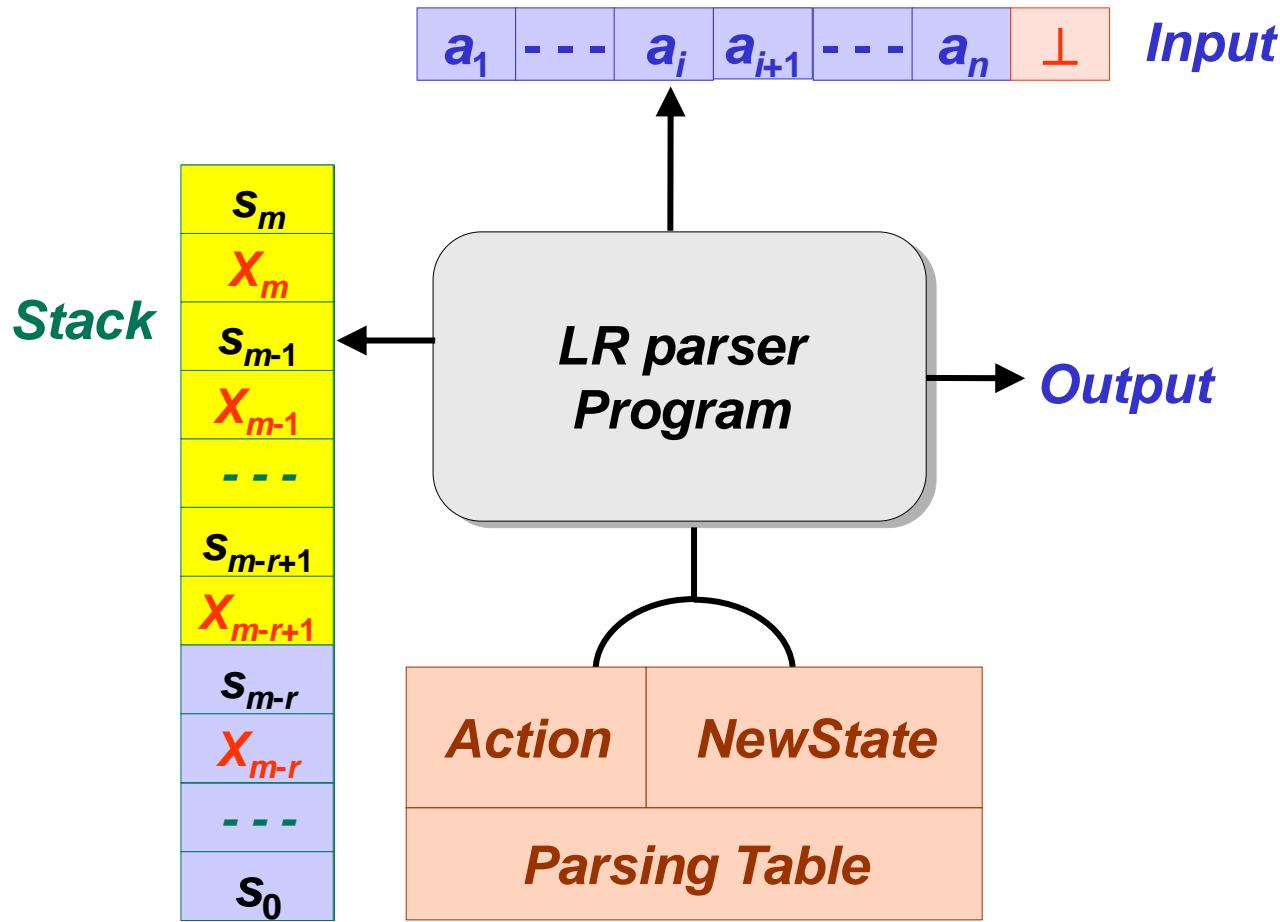
LR parsing



($s_0 \ X_1 \ s_1 \ X_2 \ s_2 \dots \ X_m \ s_m$, $a_i \ a_{i+1} \dots \ a_n \perp$)

Action [s_m, a_i] = **Reduce** $A \rightarrow \beta$

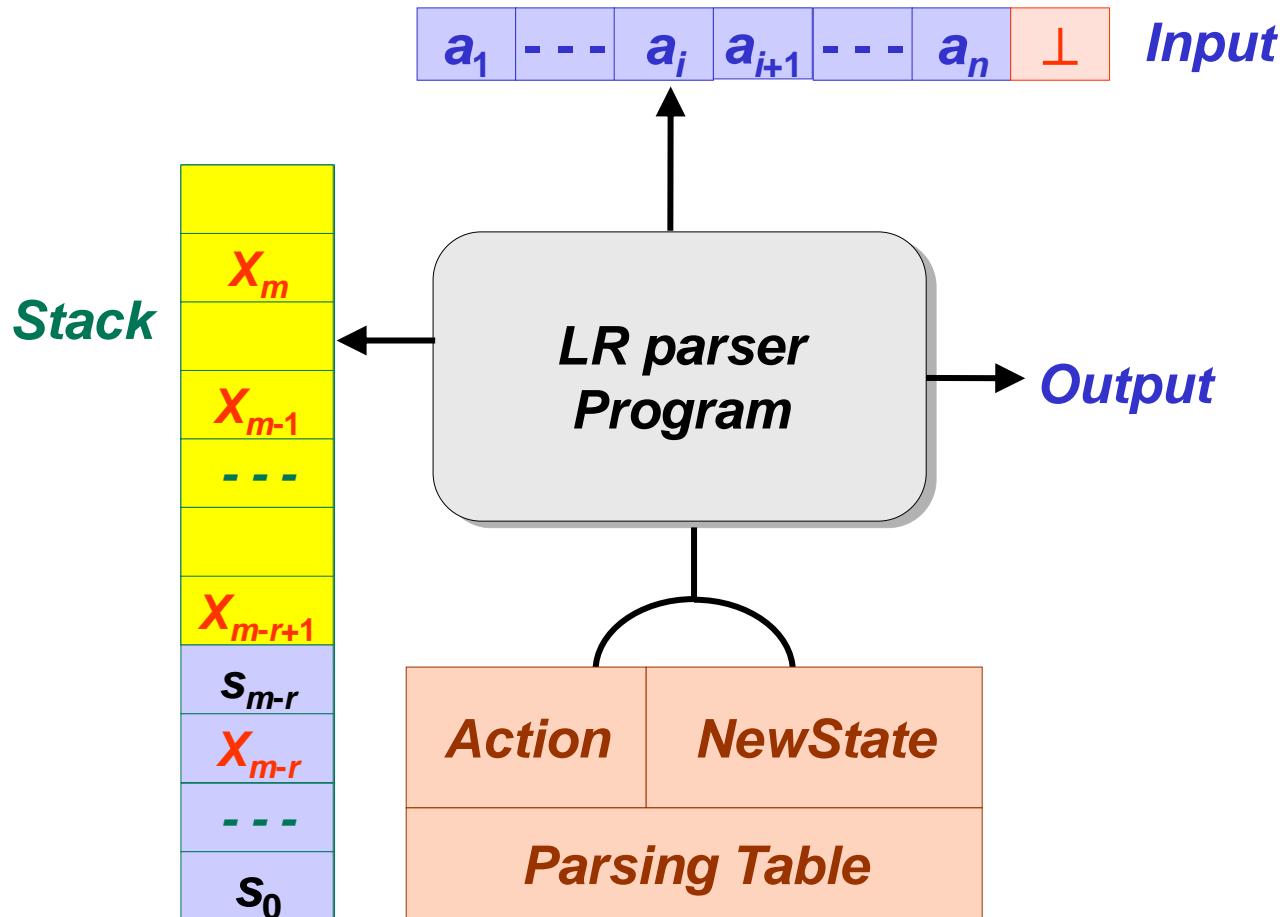
LR parsing



($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp$)

Action [s_m, a_i] = **Recuce** $A \rightarrow \beta$

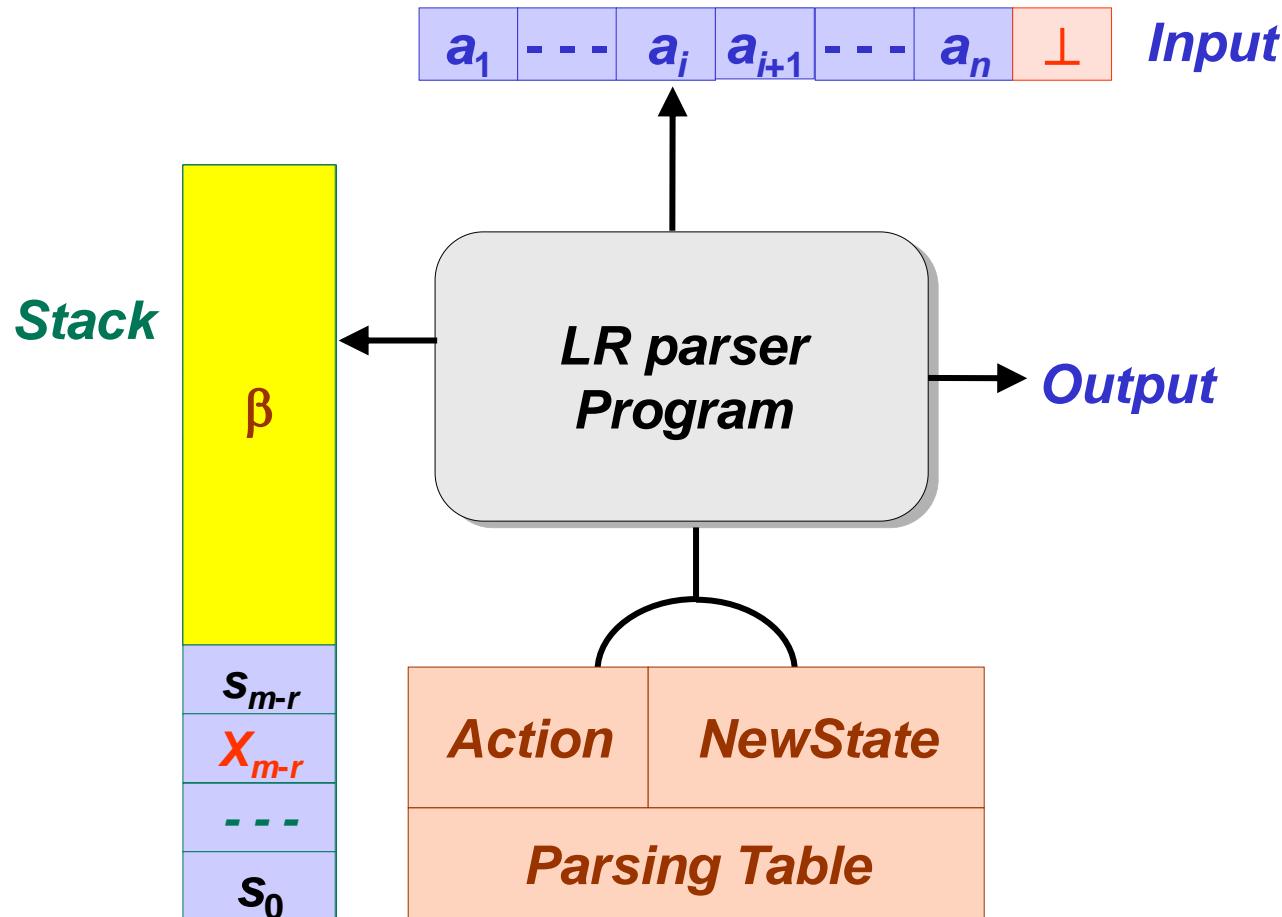
LR parsing



$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp)$

Action [s_m, a_i] = **Recuce** $A \rightarrow \beta$

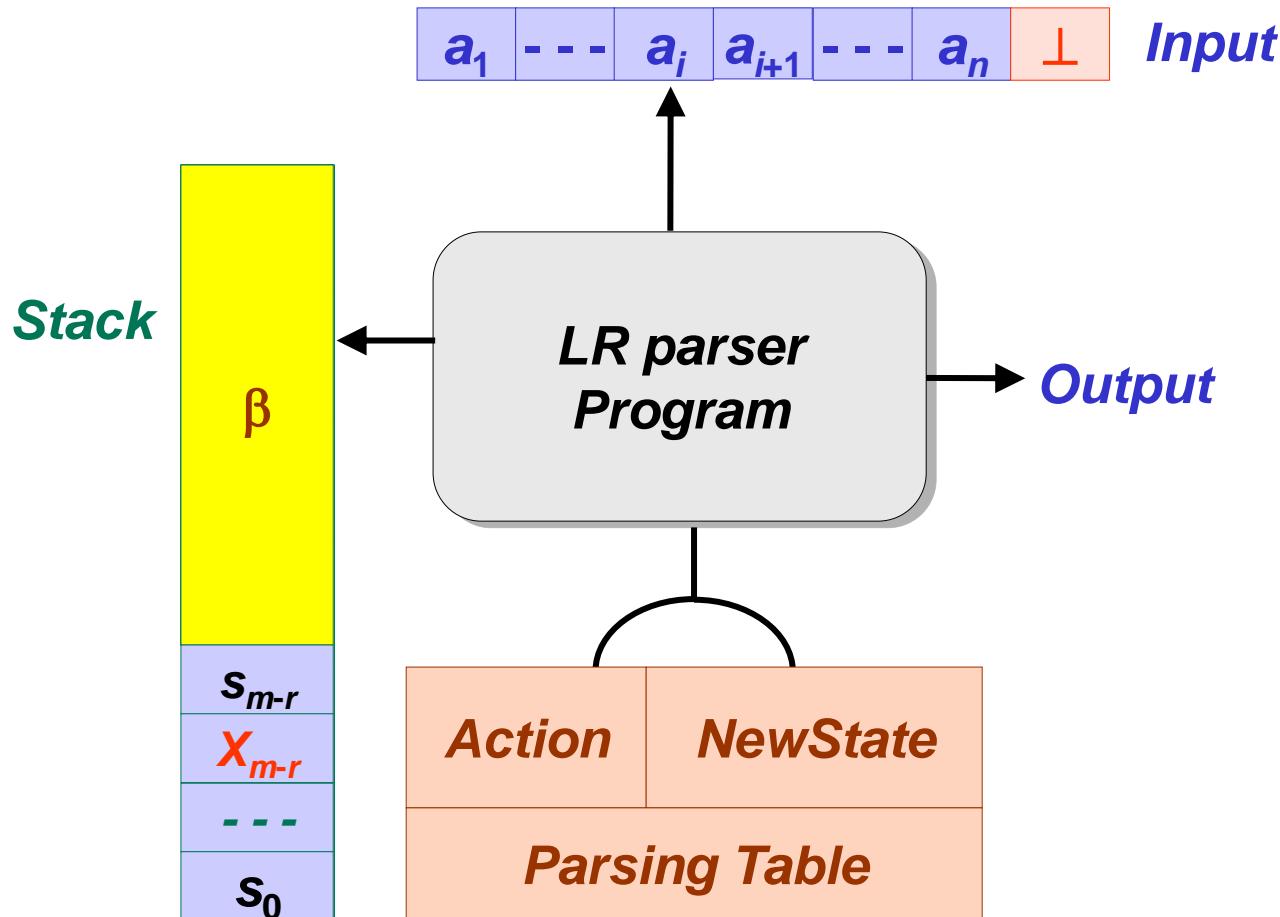
LR parsing



$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp)$

Action [s_m, a_i] = **Reduce** $A \rightarrow \beta$

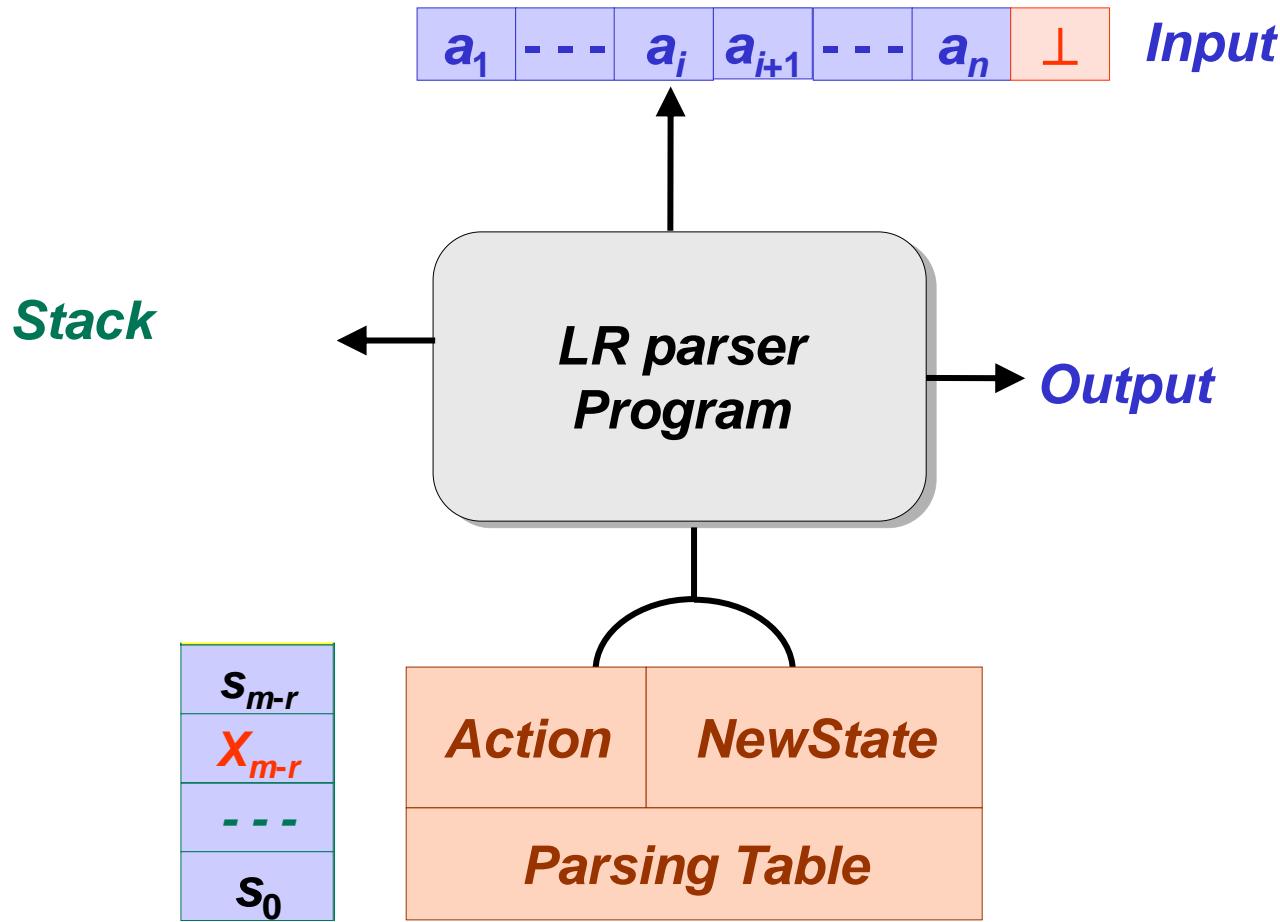
LR parsing



$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m , a_i a_{i+1} \dots a_n \perp)$

Action [s_m, a_i] = **Recuce** $A \rightarrow \beta$

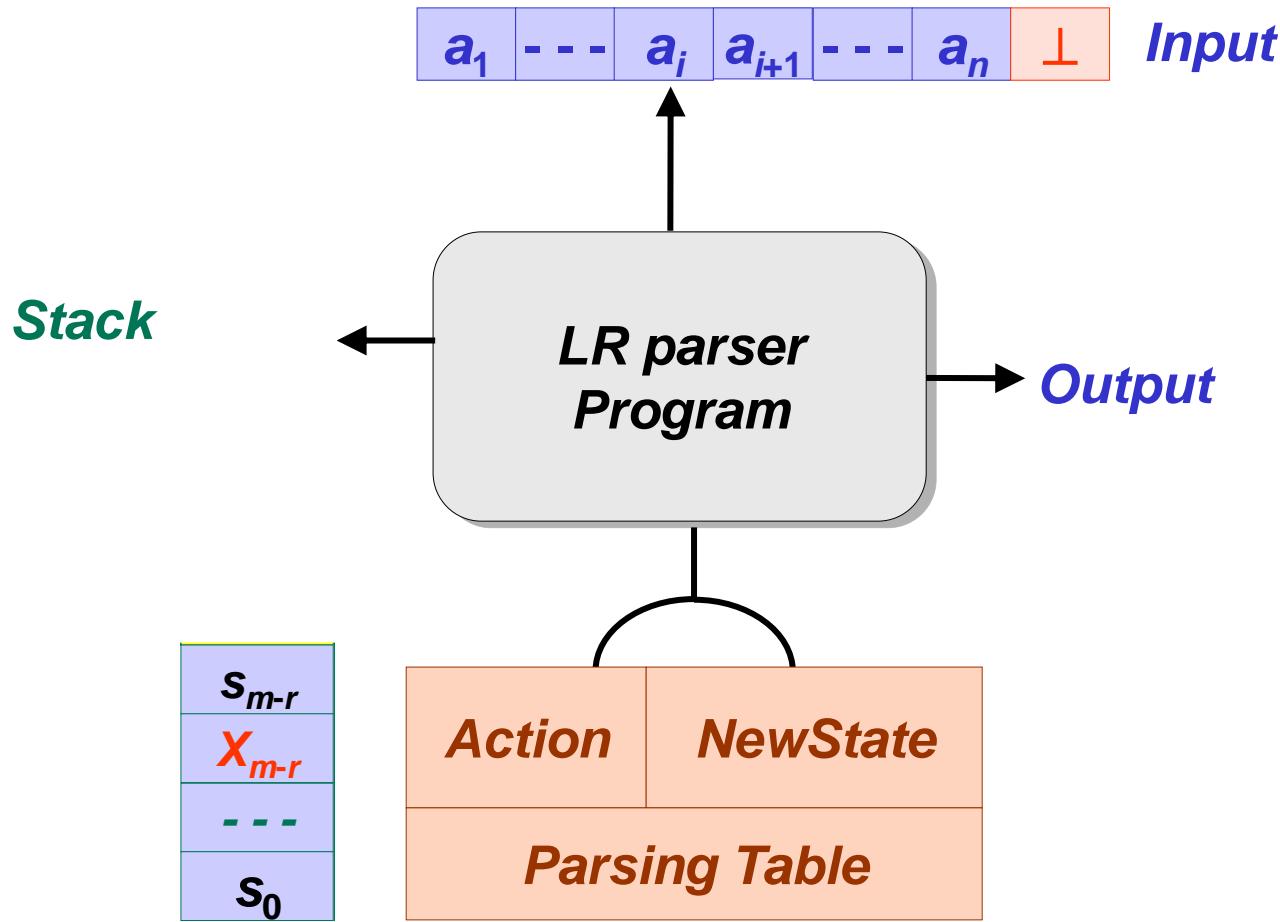
LR parsing



($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp$)

Action [s_m, a_i] = **Reduce** $A \rightarrow \beta$

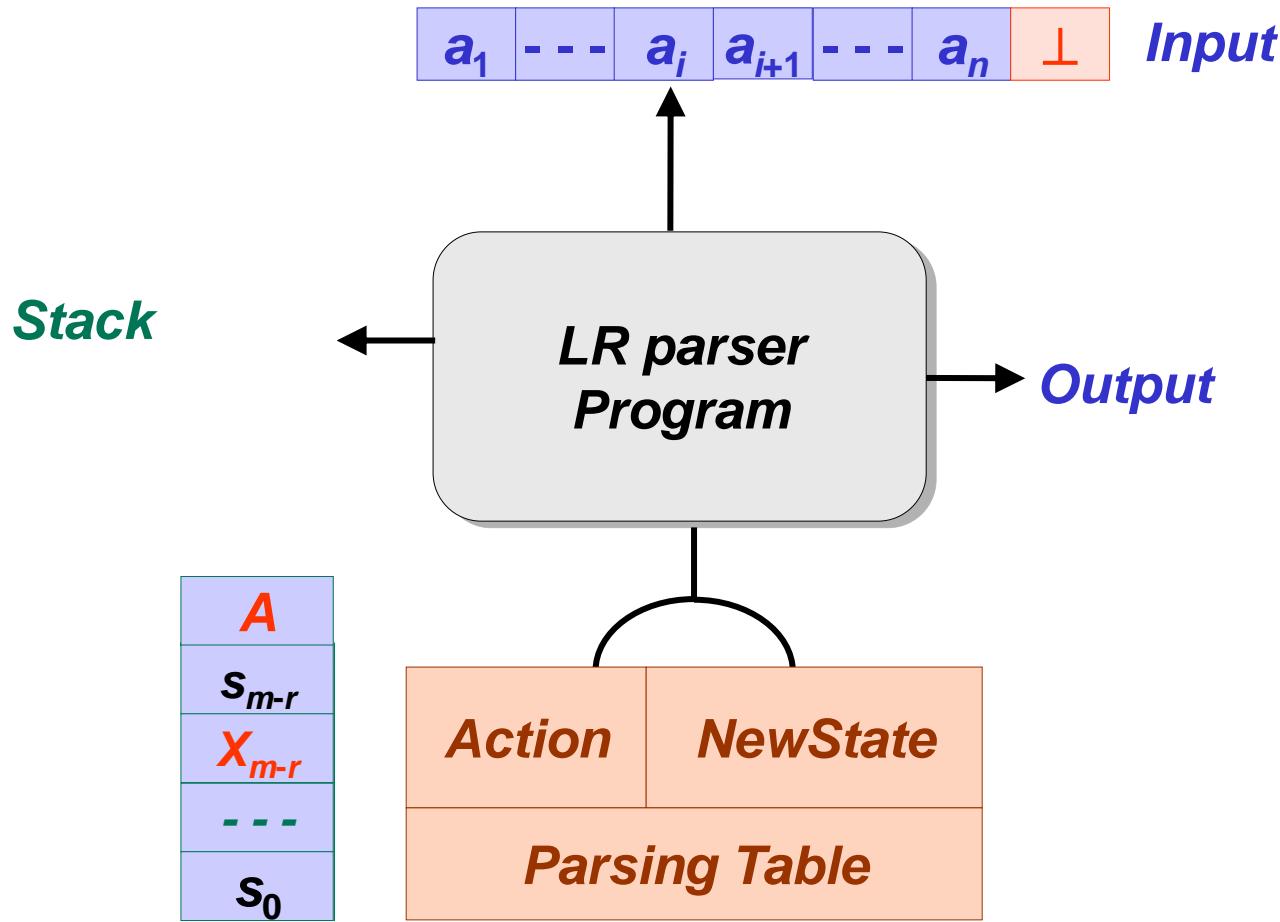
LR parsing



($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp$)

Action [s_m, a_i] = Reduce $A \rightarrow \beta$

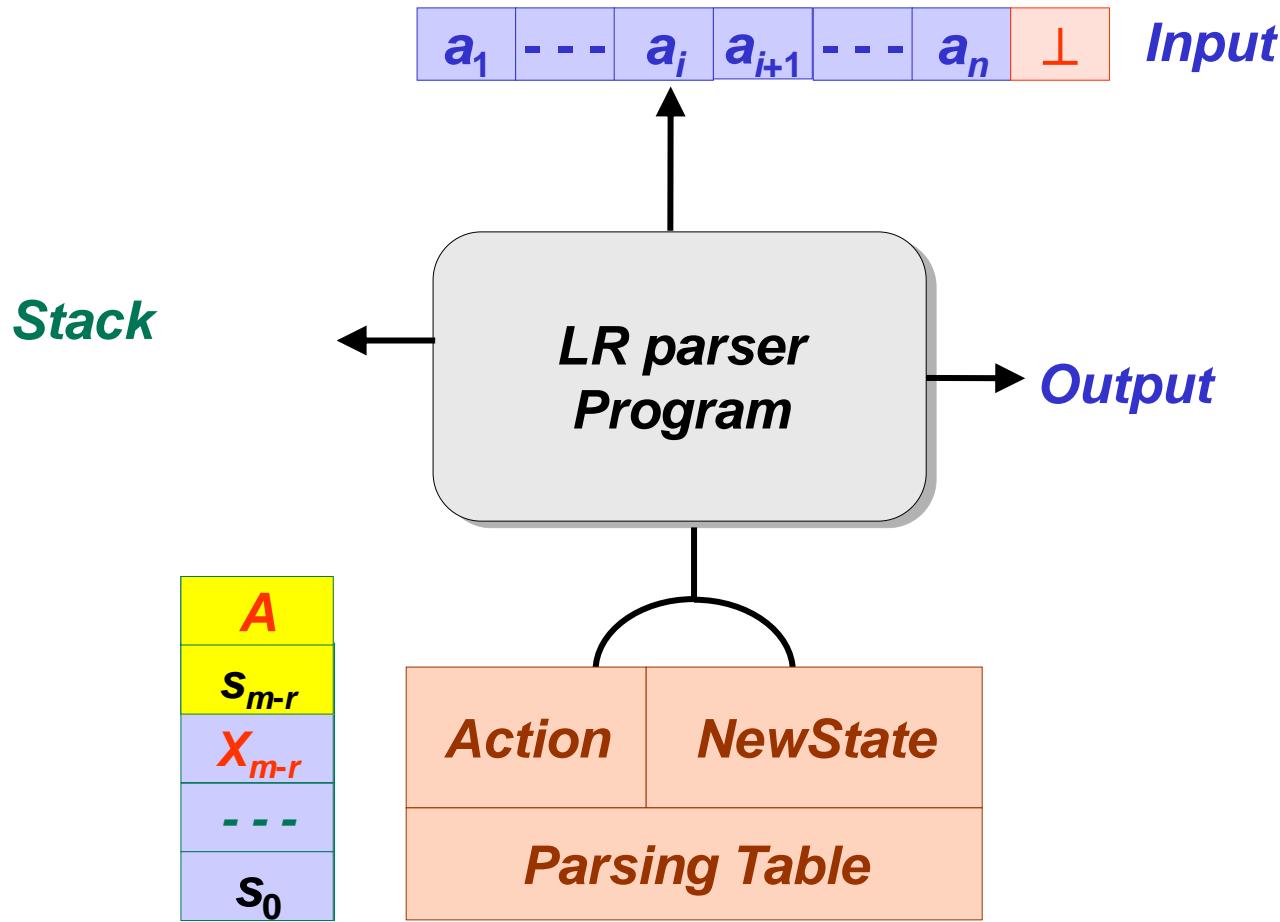
LR parsing



$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp)$

Action [s_m, a_i] = **Recuce** $A \rightarrow \beta$

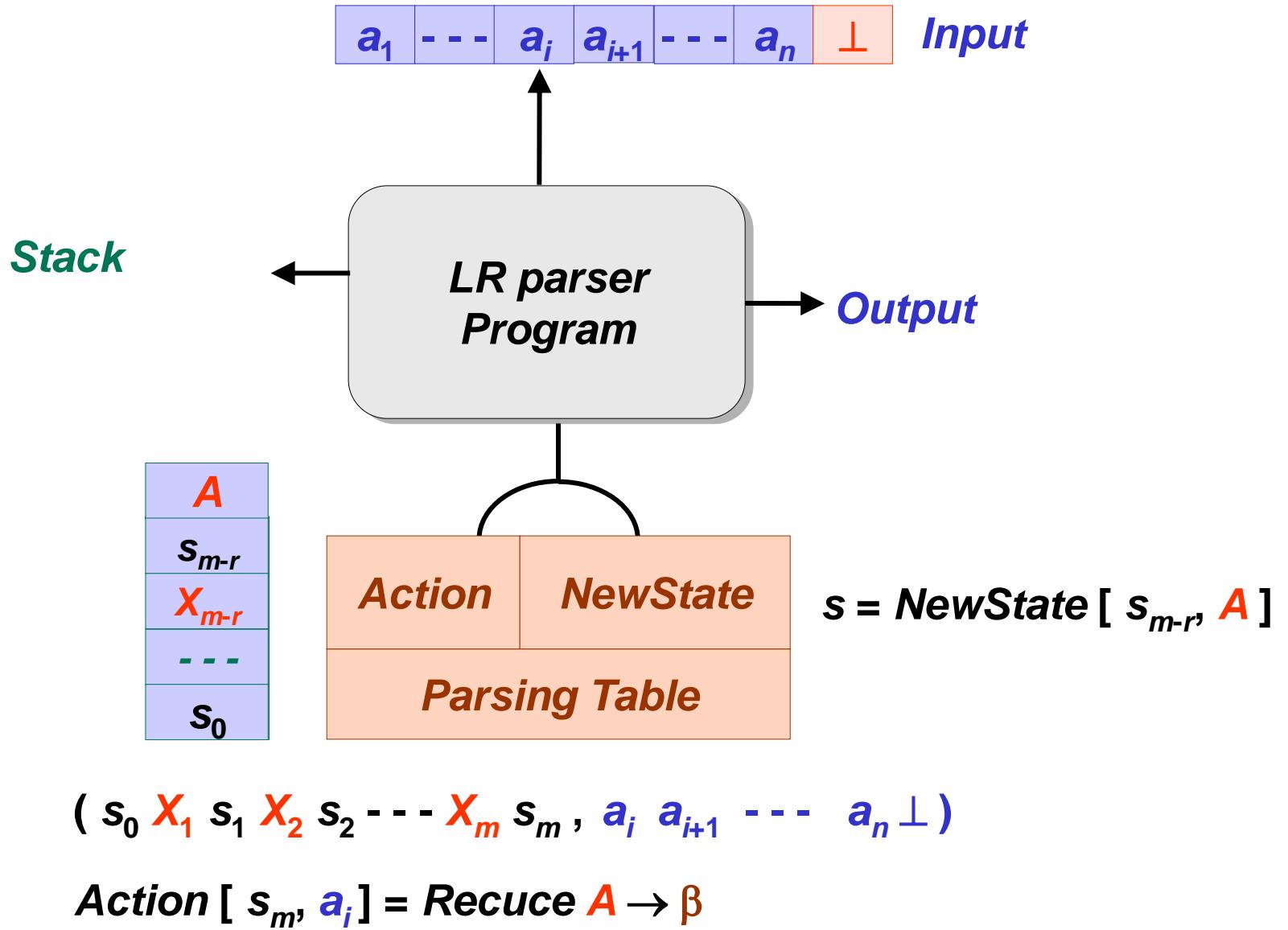
LR parsing



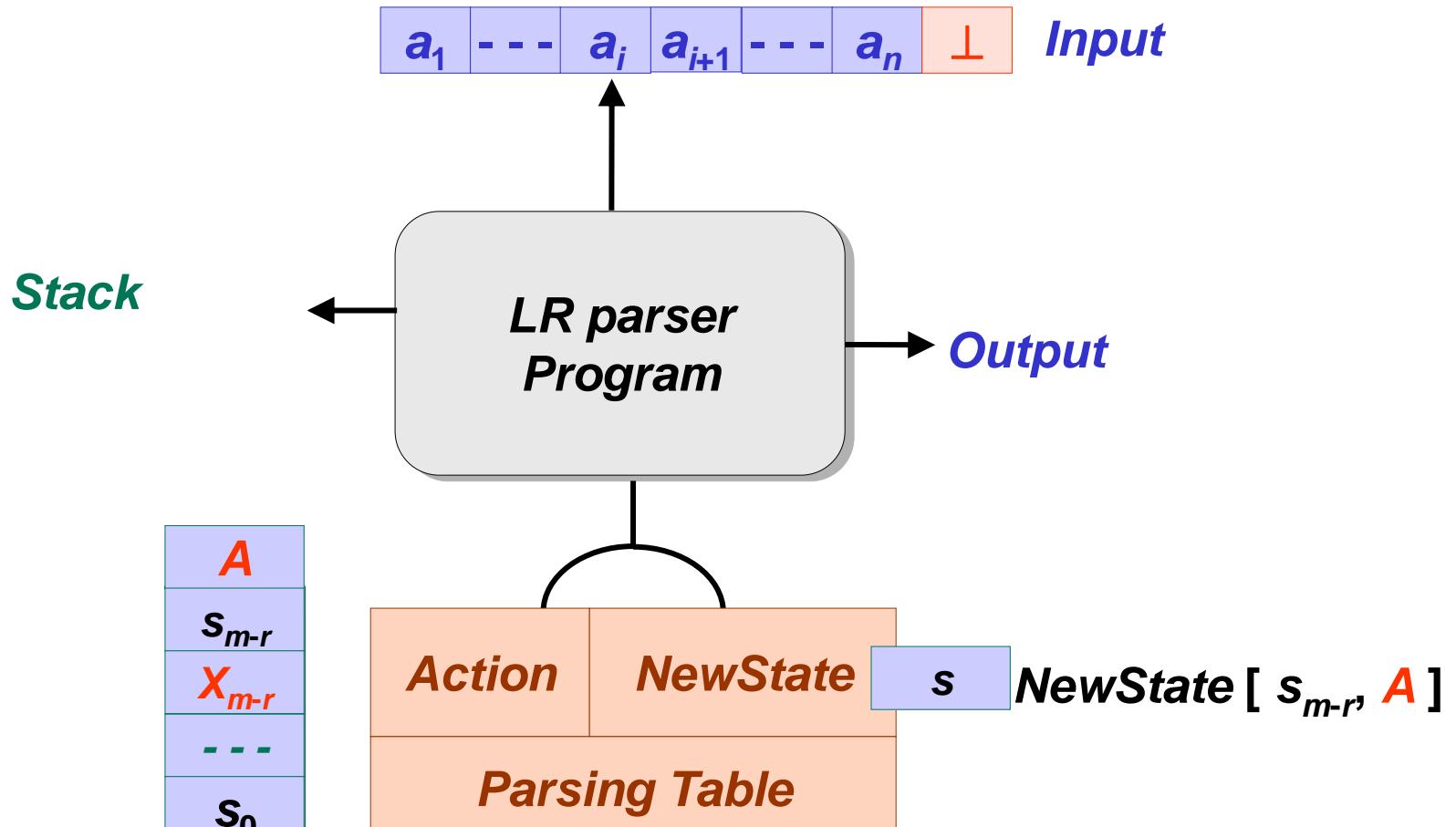
($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp$)

Action [s_m, a_i] = **Reduce** $A \rightarrow \beta$

LR parsing



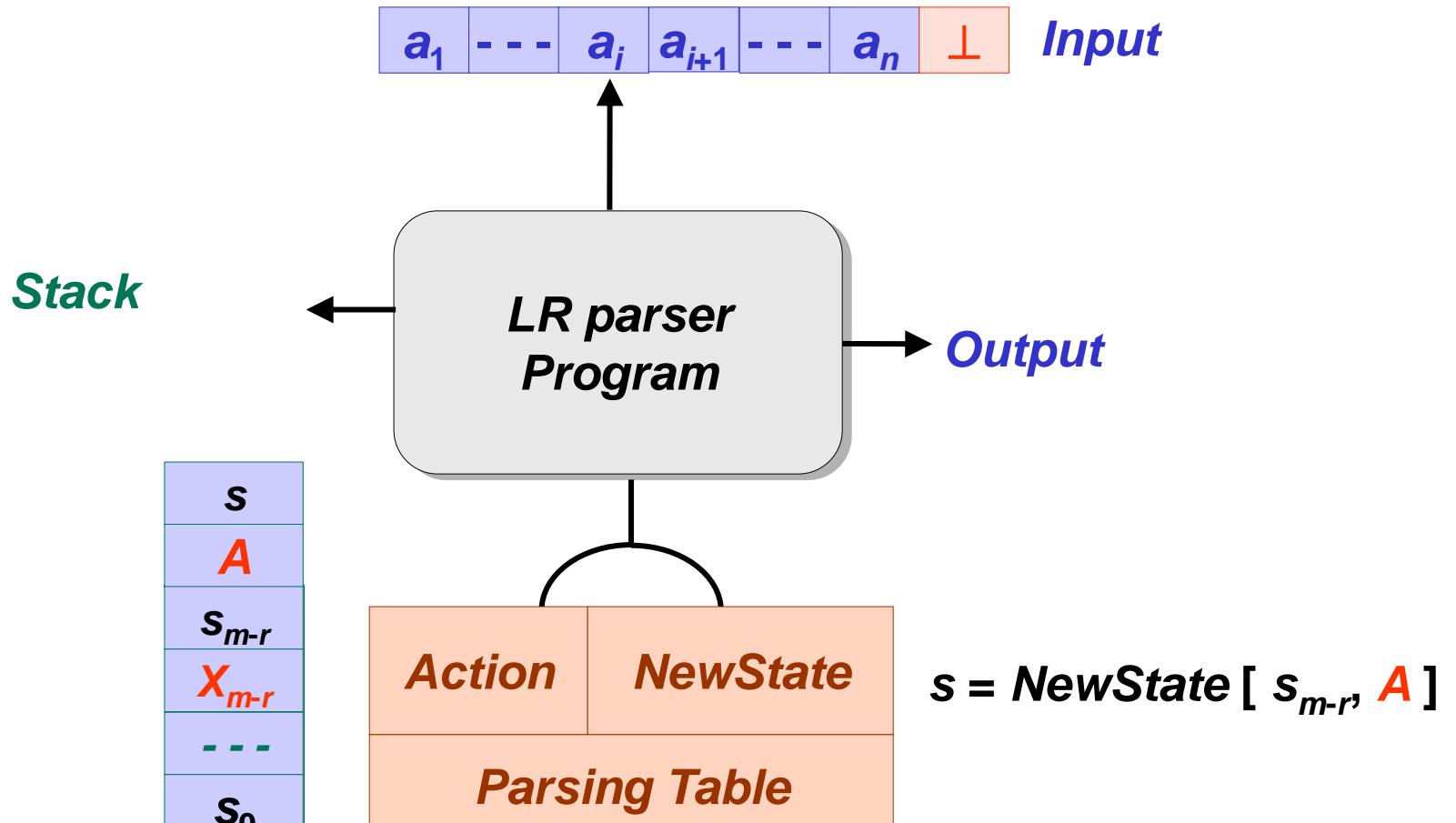
LR parsing



($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp$)

Action [s_m, a_i] = **Reduce** $A \rightarrow \beta$

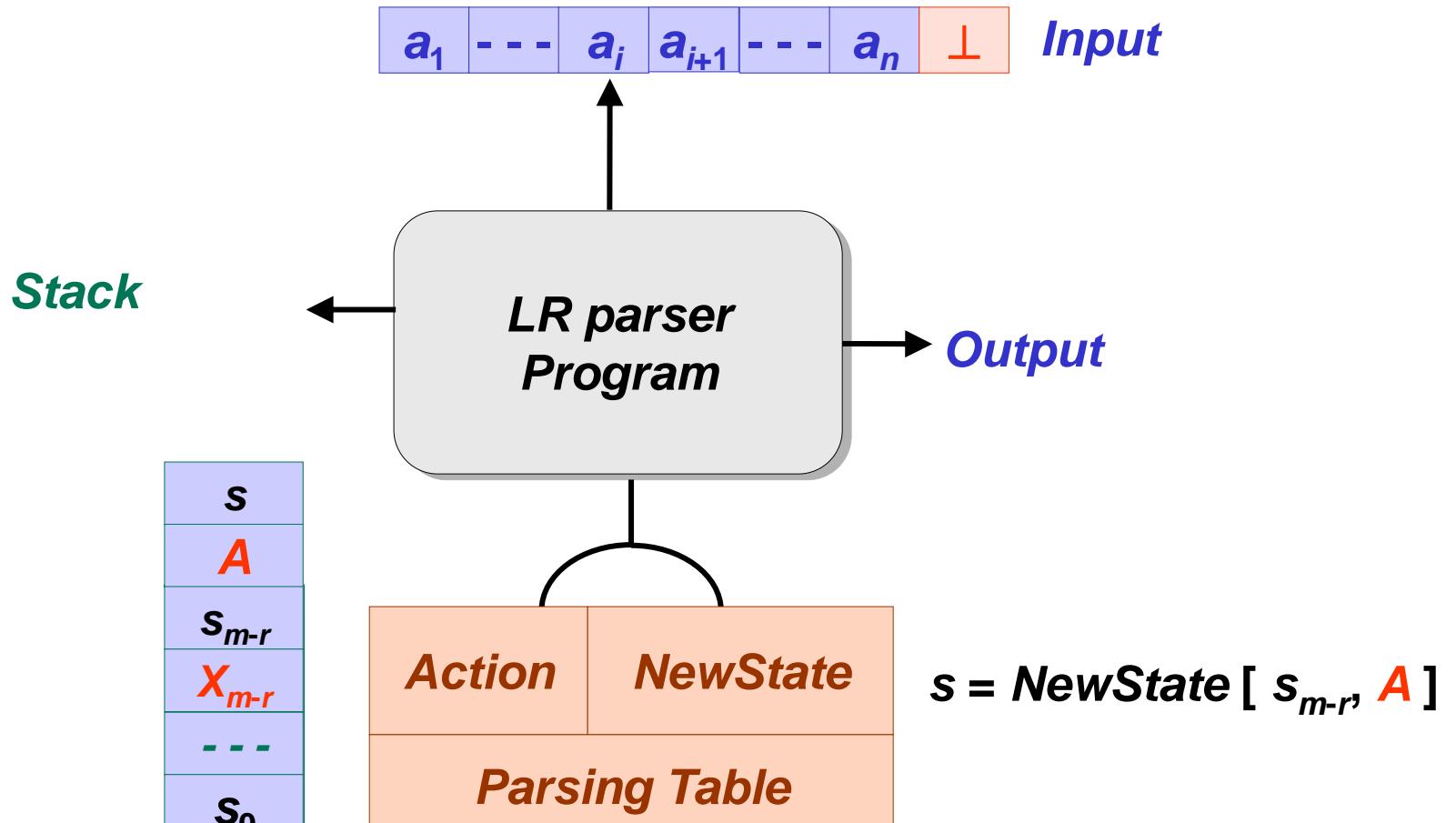
LR parsing



($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp$)

Action [s_m, a_i] = **Reduce** $A \rightarrow \beta$

LR parsing

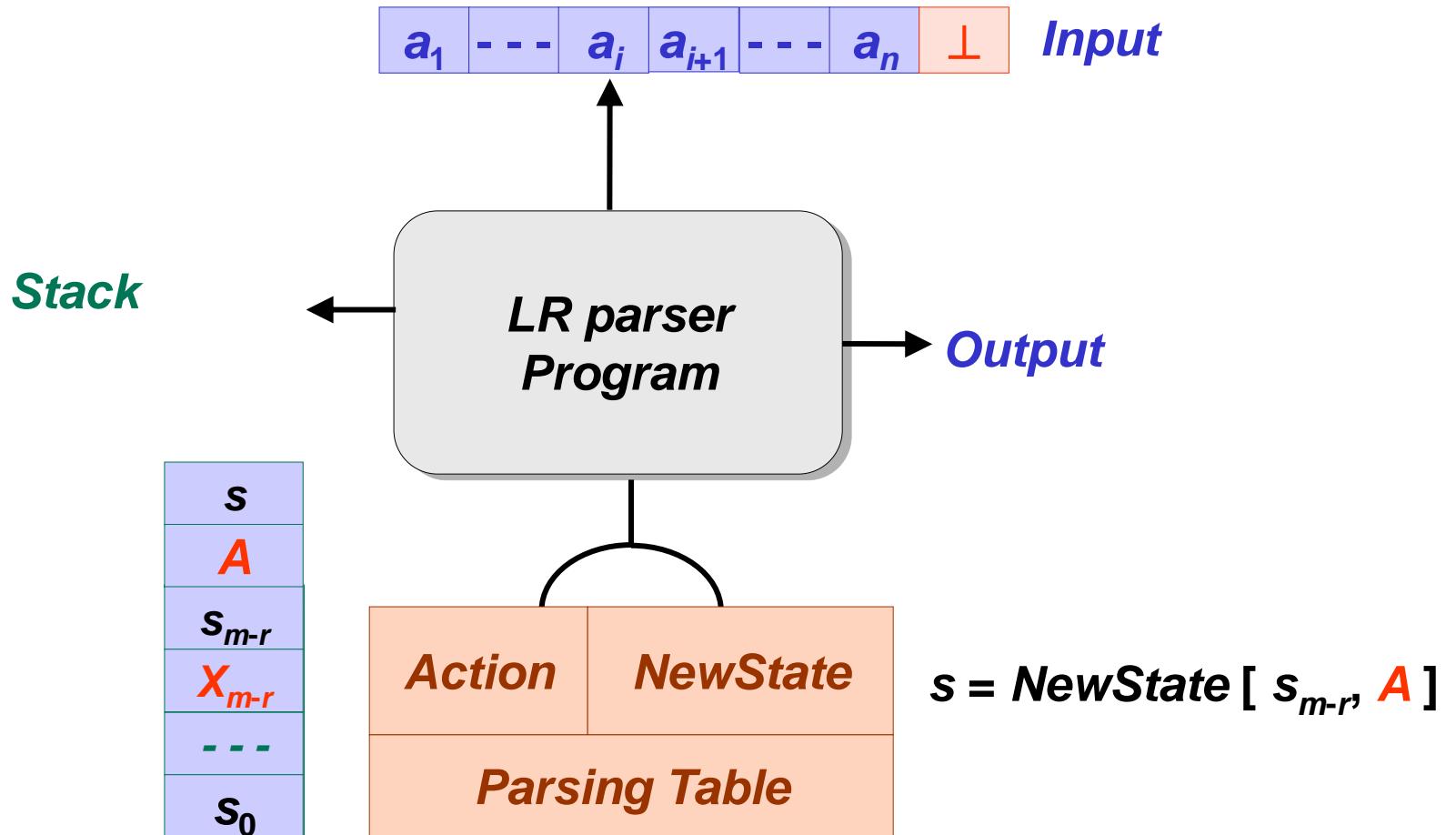


($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp$)

Action [s_m, a_i] = **Reduce** $A \rightarrow \beta$

($s_0 X_1 s_1 \dots X_{m-r} s_{m-r} \dots X_m s_m, a_i a_{i+1} \dots a_n \perp$)

LR parsing

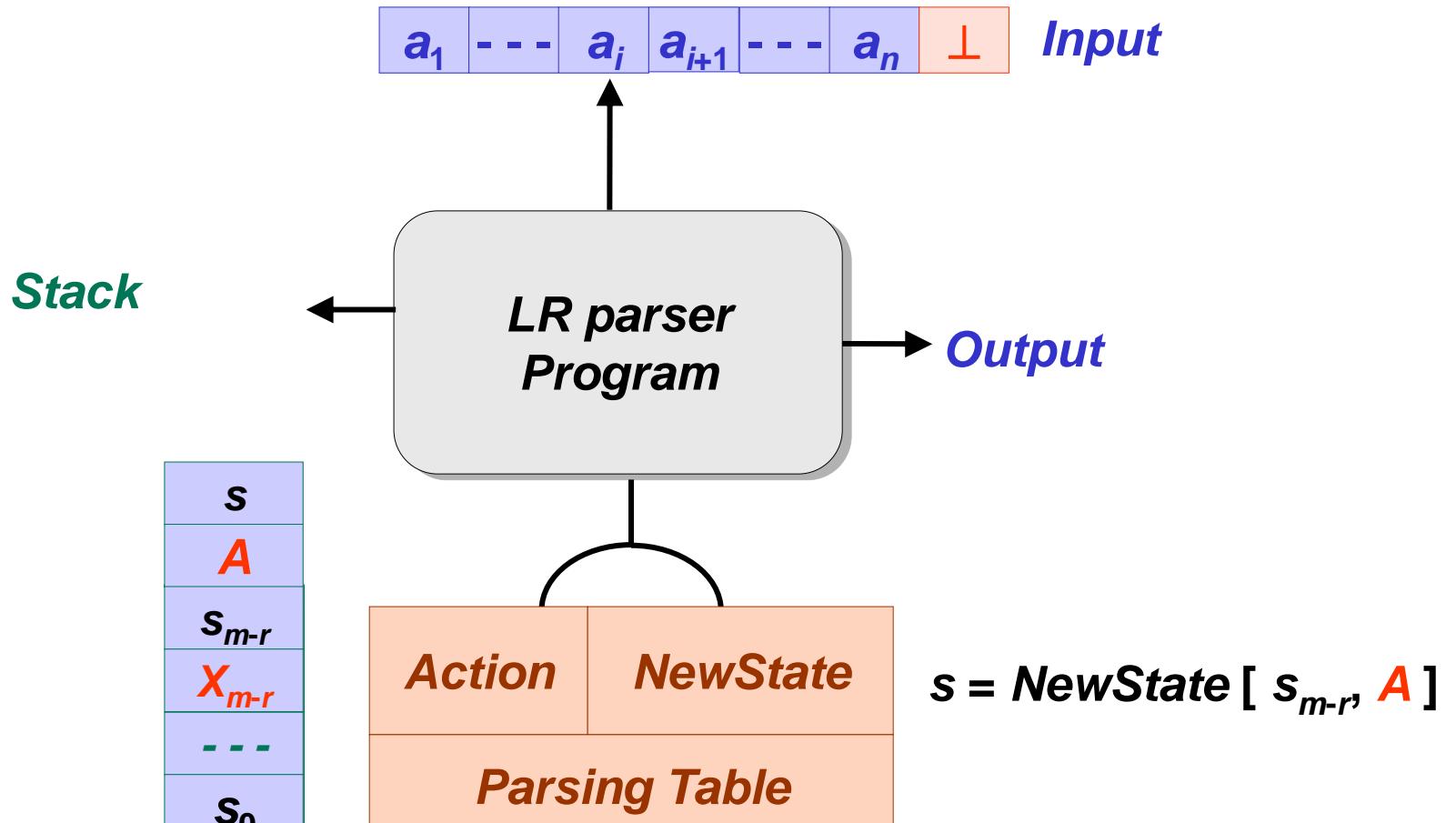


($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp$)

Action [s_m, a_i] = Reduce $A \rightarrow \beta$

($s_0 X_1 s_1 \dots X_{m-r} s_{m-r}, a_i a_{i+1} \dots a_n \perp$)

LR parsing



($s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp$)

Action [s_m, a_i] = **Reduce** $A \rightarrow \beta$

($s_0 X_1 s_1 \dots X_{m-r} s_{m-r} A s, a_i a_{i+1} \dots a_n \perp$)

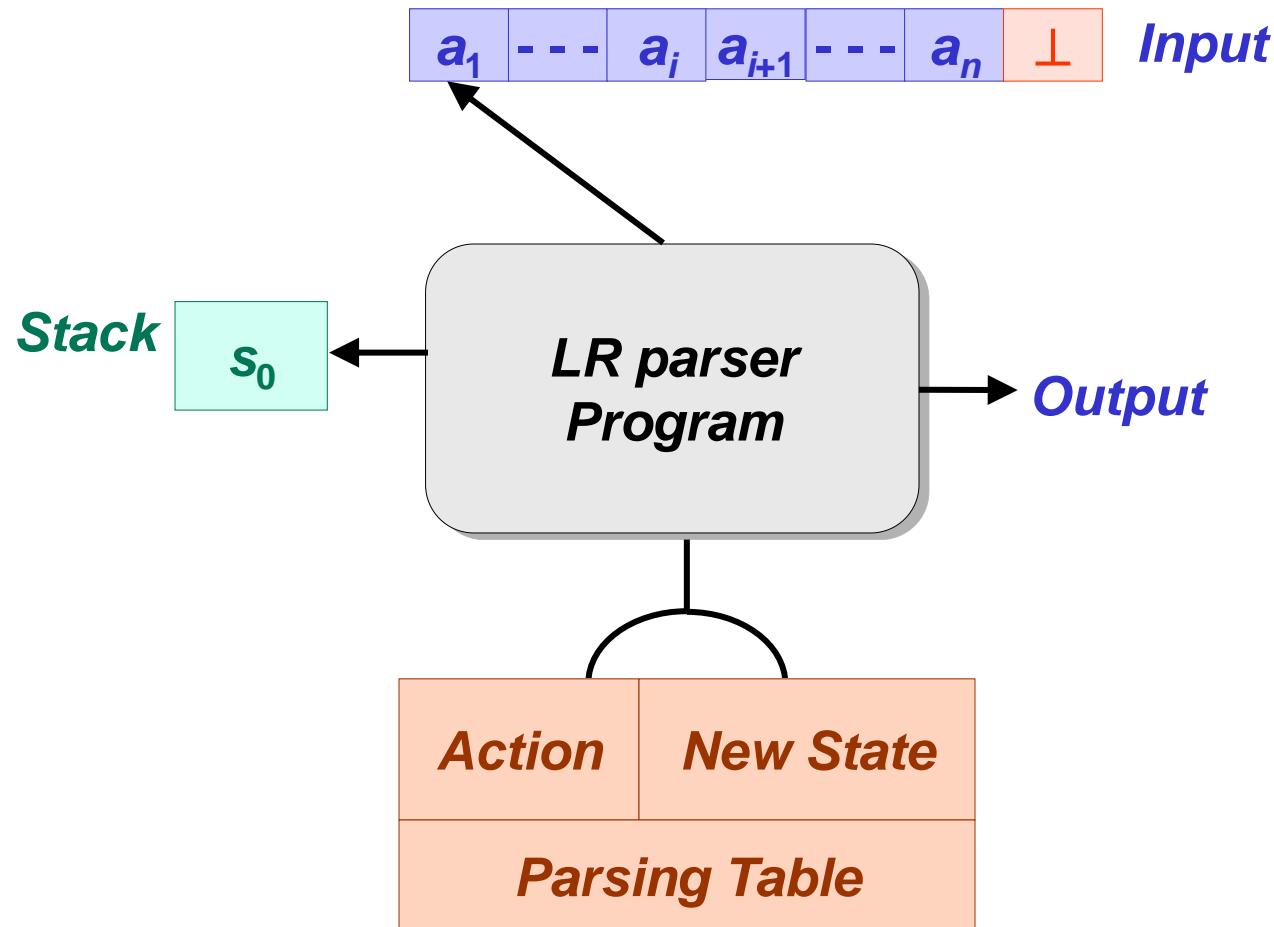
LR parsing

LR parsing

- Initial configuration

LR parsing

- Initial configuration



LR parsing



LR parsing

LRParserProgram()



LR parsing

```
LRParserProgram()  
{
```



LR parsing

```
LRParserProgram()  
{
```

Set the input handle on the leftmost symbol in string $w\perp$;



LR parsing

```
LRParserProgram()  
{
```

Set the input handle on the leftmost symbol in string $w\perp$;

```
while( 1 )
```



LR parsing

```
LRParserProgram()
```

```
{
```

Set the input handle on the leftmost symbol in string $w\perp$;

```
while( 1 )
```

```
{
```



LR parsing

```
LRParserProgram()
```

```
{
```

Set the input handle on the leftmost symbol in string $w \perp$;

```
while( 1 )
```

```
{
```

// Let s is a state on the top of the stack



LR parsing

```
LRParserProgram()
```

```
{
```

Set the input handle on the leftmost symbol in string $w \perp$;

```
while( 1 )
```

```
{
```

// Let s is a state on the top of the stack

// Let POINTER points on the current symbol in string $w \perp$



LR parsing

```
LRParserProgram()
```

```
{
```

Set the input handle on the leftmost symbol in string $w \perp$;

```
while( 1 )
```

```
{
```

// Let s is a state on the top of the stack

// Let POINTER points on the current symbol in string $w \perp$

```
case( Action[ s, a ] )
```



LR parsing

```
LRParserProgram()
```

```
{
```

Set the input handle on the leftmost symbol in string $w\perp$;

```
while( 1 )
```

```
{
```

// Let s is a state on the top of the stack

// Let POINTER points on the current symbol in string $w\perp$

```
case( Action[ s, a ] )
```

```
{
```



LR parsing

```
LRParserProgram()
```

```
{
```

Set the input handle on the leftmost symbol in string $w\perp$;

```
while( 1 )
```

```
{
```

// Let s is a state on the top of the stack

// Let POINTER points on the current symbol in string $w\perp$

```
case( Action[ s, a ] )
```

```
{
```

Shift s' :



LR parsing

```
LRParserProgram()
```

```
{
```

Set the input handle on the leftmost symbol in string $w\perp$;

```
while( 1 )
```

```
{
```

// Let s is a state on the top of the stack

// Let POINTER points on the current symbol in string $w\perp$

```
case( Action[ s, a ] )
```

```
{
```

Shift s' :

Reduce $A \rightarrow \beta$:



LR parsing

```
LRParserProgram()
```

```
{
```

Set the input handle on the leftmost symbol in string $w\perp$;

```
while( 1 )
```

```
{
```

// Let s is a state on the top of the stack

// Let POINTER points on the current symbol in string $w\perp$

```
case( Action[ s, a ] )
```

```
{
```

Shift s' :

Reduce $A \rightarrow \beta$:

Accept:



LR parsing

```
LRParserProgram()
```

```
{
```

Set the input handle on the leftmost symbol in string $w\perp$;

```
while( 1 )
```

```
{
```

// Let s is a state on the top of the stack

// Let POINTER points on the current symbol in string $w\perp$

```
case( Action[ s, a ] )
```

```
{
```

Shift s' :

Reduce $A \rightarrow \beta$:

Accept:

Other cases:



LR parsing

```
LRParserProgram()
```

```
{
```

Set the input handle on the leftmost symbol in string $w\perp$;

```
while( 1 )
```

```
{
```

// Let s is a state on the top of the stack

// Let POINTER points on the current symbol in string $w\perp$

```
case( Action[ s, a ] )
```

```
{
```

Shift s' :

Reduce $A \rightarrow \beta$:

Accept:

Other cases:

```
}
```



LR parsing

```
LRParserProgram()
```

```
{
```

Set the input handle on the leftmost symbol in string $w\perp$;

```
while( 1 )
```

```
{
```

// Let s is a state on the top of the stack

// Let POINTER points on the current symbol in string $w\perp$

```
case( Action[ s, a ] )
```

```
{
```

Shift s' :

Reduce $A \rightarrow \beta$:

Accept:

Other cases:

```
}
```

```
}
```



LR parsing

```
LRParserProgram()
```

```
{
```

Set the input handle on the leftmost symbol in string $w \perp$;

```
while( 1 )
```

```
{
```

// Let s is a state on the top of the stack

// Let POINTER points on the current symbol in string $w \perp$

```
case( Action[ s, a ] )
```

```
{
```

Shift s' :



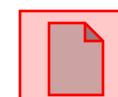
Reduce $A \rightarrow \beta$:



Accept:



Other cases:



```
}
```

```
}
```



LR parsing

Action[s, a] = Shift s' :

Push a on the stack;

Push s' on the stack;

*Move **POINTER** to the next symbol in $w\perp$;*



LR parsiranje

Action[s, a] = Reduce $A \rightarrow \beta$:

Pop $2^|\beta|$ symbols of the stack;*

// Let s' is a state on the top of the stack

// after removing $2^|\beta|$ symbols*

Push A on the stack;

Push NewState[s' , A] on the stack;

Write (“Reduction $A \rightarrow \beta$ applied”);



LR parsiranje

Action[s, a] = Accept:

Write (“ String w is in language $L(G)$ ”);

Stop the program;



LR parsiranje

Other cases:

Write (“*String w is not in language $L(G)$* ”);

Stop the program;



1) $E \rightarrow E + T$
2) $E \rightarrow T$

3) $T \rightarrow T * F$
4) $T \rightarrow F$

5) $F \rightarrow (E)$
6) $F \rightarrow \text{var}$

1) $E \rightarrow E + T$
 2) $E \rightarrow T$

3) $T \rightarrow T * F$
 4) $T \rightarrow F$

5) $F \rightarrow (E)$
 6) $F \rightarrow \text{var}$

State	Action							NewState		
	var	+	*	()	\perp	E	T	F	
0	s5			s4			1	2	3	
1		s6				Acc.				
2		r2	s7		r2	r2				
3		r4	r4		r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

1) $E \rightarrow E + T$
 2) $E \rightarrow T$

3) $T \rightarrow T * F$
 4) $T \rightarrow F$

5) $F \rightarrow (E)$
 6) $F \rightarrow \text{var}$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	E	T	F	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

0

Input

Action

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0

var+var*var \perp

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0

var+var*var \perp

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0

var + var * var \perp

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2		r2	s7		r2	r2				
3		r4	r4		r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0

var+var*var \perp

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0	var+var*var \perp	shift 5
---	---------------------	---------

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

0 var 5

Input

var+var*var \perp

Action

shift 5

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 var 5

+var*var \perp

shift 5

1) $E \rightarrow E + T$
2) $E \rightarrow T$

3) $T \rightarrow T * F$
4) $T \rightarrow F$

5) $F \rightarrow (E)$
6) $F \rightarrow \text{var}$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	E	T	F	
0	s5				s4			1	2	3
1			s6				Acc.			
2		r2	s7			r2	r2			
3		r4	r4			r4	r4			
4	s5				s4			8	2	3
5		r6	r6			r6	r6			
6	s5				s4				9	3
7	s5				s4					10
8			s6			s11				
9		r1	s7			r1	r1			
10		r3	r3			r3	r3			
11		r5	r5			r5	r5			

Stack

Input

Action

0 var 5

+var*var \perp

shift 5

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 var 5

+var*var \perp

shift 5

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 var 5

+var*var \perp

reduction $F \rightarrow \text{var}$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0

+var*var \perp

reduction $F \rightarrow \text{var}$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0	+var*var \perp	reduction $F \rightarrow \text{var}$
---	------------------	--------------------------------------

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *F*

+var*var \perp

reduction F → var

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 ***F***

+var*var \perp

reduction F → var

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *F*

+var*var \perp

reduction *F* \rightarrow var

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5				s4		1	2	3	
1			s6			Acc.				
2		r2	s7			r2	r2			
3		r4	r4			r4	r4			
4	s5				s4		8	2	3	
5		r6	r6			r6	r6			
6	s5				s4			9	3	
7	s5				s4				10	
8		s6				s11				
9		r1	s7			r1	r1			
10		r3	r3			r3	r3			
11		r5	r5			r5	r5			

Stack

Input

Action

0 *F*

+var*var \perp

reduction *F* \rightarrow var

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

0 *F* 3

Input

+var*var \perp

Action

reduction *F* \rightarrow var

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

0 *F* 3

Input

+var*var \perp

Action

reduction $T \rightarrow F$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

0 *T* 2

Input

+var*var \perp

Action

reduction $T \rightarrow F$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

0 *T* 2

Input

+var*var \perp

Action

reduction $E \rightarrow T$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1

+var*var \perp

reduction *E* \rightarrow *T*

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1

+var*var \perp

shift 6

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

0 *E* 1 + 6

Input

+var*var \perp

Action

shift 6

1) $E \rightarrow E + T$
 2) $E \rightarrow T$

3) $T \rightarrow T * F$
 4) $T \rightarrow F$

5) $F \rightarrow (E)$
 6) $F \rightarrow \text{var}$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	E	T	F	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 E 1 + 6

var*var \perp

shift 6

1) $E \rightarrow E + T$
 2) $E \rightarrow T$

3) $T \rightarrow T * F$
 4) $T \rightarrow F$

5) $F \rightarrow (E)$
 6) $F \rightarrow \text{var}$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	E	T	F	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 E 1 + 6

var*var \perp

shift 5

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 var 5

var*var \perp

shift 5

1) $E \rightarrow E + T$
2) $E \rightarrow T$

3) $T \rightarrow T * F$
4) $T \rightarrow F$

5) $F \rightarrow (E)$
6) $F \rightarrow \text{var}$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	E	T	F	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 E 1 + 6 var 5

*var \perp

shift 5

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 var 5

*var \perp

reduction *F* \rightarrow var

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *F* 3

*var \perp

reduction *F* \rightarrow var

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *F* 3

*var \perp

reduction $T \rightarrow F$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *T* 9

*var \perp

reduction $T \rightarrow F$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *T* 9

*var \perp

shift 7

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *T* 9 * 7

*var \perp

shift 7

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *T* 9 * 7

var \perp

shift 7

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *T* 9 * 7

var \perp

shift 5

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *T* 9 * 7 var 5

var \perp

shift 5

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *T* 9 * 7 var 5

\perp

shift 5

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *T* 9 * 7 var 5

\perp

reduction $F \rightarrow \text{var}$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *T* 9 * 7 *F* 10

\perp

reduction F → var

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *T* 9 * 7 *F* 10

\perp

reduction $T \rightarrow T * F$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *T* 9

\perp

reduction $T \rightarrow T * F$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1 + 6 *T* 9

\perp

reduction $E \rightarrow E + T$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1

\perp

reduction $E \rightarrow E + T$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 ***E*** 1

\perp

reduction $E \rightarrow E + T$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1

\perp

reduction $E \rightarrow E + T$

$$\begin{array}{l} 1) E \rightarrow E + T \\ 2) E \rightarrow T \end{array}$$

$$\begin{array}{l} 3) T \rightarrow T * F \\ 4) T \rightarrow F \end{array}$$

$$\begin{array}{l} 5) F \rightarrow (E) \\ 6) F \rightarrow \text{var} \end{array}$$

State

	<i>Action</i>							<i>NewState</i>		
	var	+	*	()	\perp	<i>E</i>	<i>T</i>	<i>F</i>	
0	s5			s4			1	2	3	
1		s6				Acc.				
2	r2	s7			r2	r2				
3	r4	r4			r4	r4				
4	s5			s4			8	2	3	
5		r6	r6		r6	r6				
6	s5			s4				9	3	
7	s5			s4					10	
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Stack

Input

Action

0 *E* 1

\perp

reduction $E \rightarrow E + T$