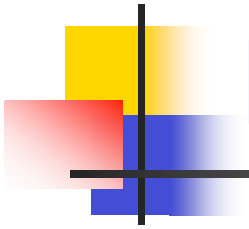


Example 1

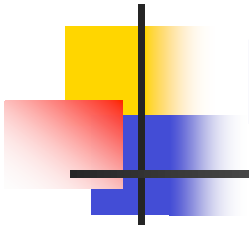
```
Stmt  ::= 1 if expr then Stmt else Stmt |
          2 while Expr do Stmt |
          3 begin Stmts end
Stmts ::= 4 Stmt ; Stmts | 5 ε
Expr  ::= 6 id
```

	if	then	else	while	do	begin	end	id	;	$
Stmt	1			2		3				
Stmts	4			4		4	5			
Expr								6		

empty = error

LL(1) Parsing Algorithm

```
push S$                                   /* S is start symbol */
while Stack not empty
    X := pop(Stack)
    a := peek at next input "token"       /* EOF => $ */
    if X is terminal or $
        If  X==a, read token a else abort;
    else look at PREDICT(X, a) /* X is nonterminal*/
        Empty        : abort
        rule X → α : push α
If not at end of input, Abort else Accept
```



Parser Example

- Following slides trace execution of the parser (slide 5) on a token string according to the grammar from slide 4 and the corresponding parse tree
- Snapshots show parser state at the top of the while loop and just before the “if” statement at each iteration, together with a summary of the action taken in the “if”
- Notice how the leaves on the right side of the parse tree correspond to the stack contents

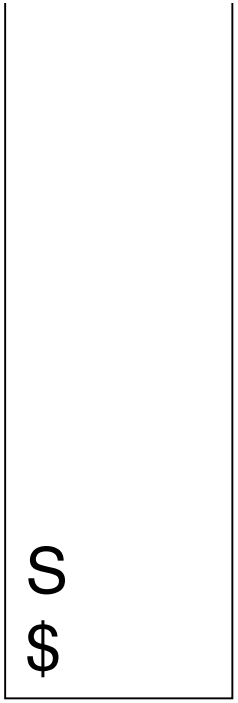
S

At top of loop

X:

a:

Stack:



while	id	do	begin	begin	end	;	end	$

Action: 2 `S ::= while E do S`

Mid loop

S

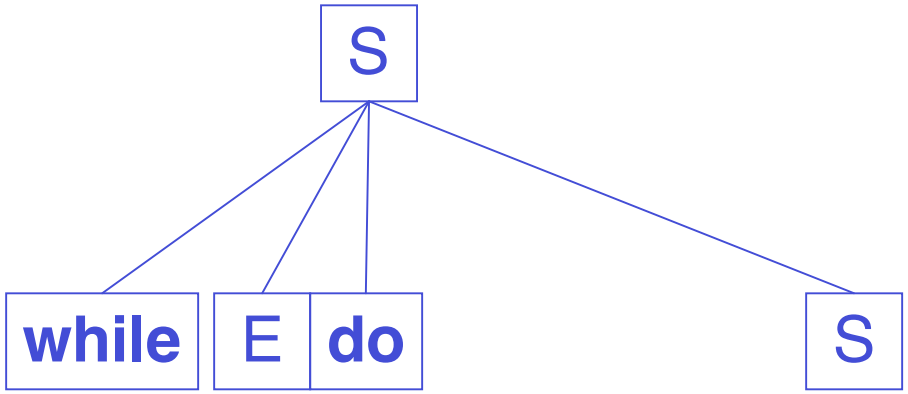
X: S

a: while

Stack:

$

while	id	do	begin	begin	end	;	end	$



At top of loop

X:

a:

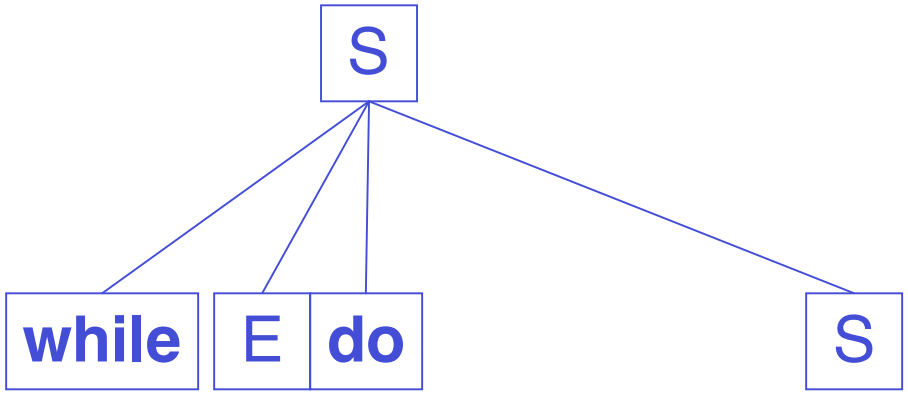
Stack:

while
E
do
S
$

while	id	do	begin	begin	end	;	end	$

Action: `Match`

Mid loop



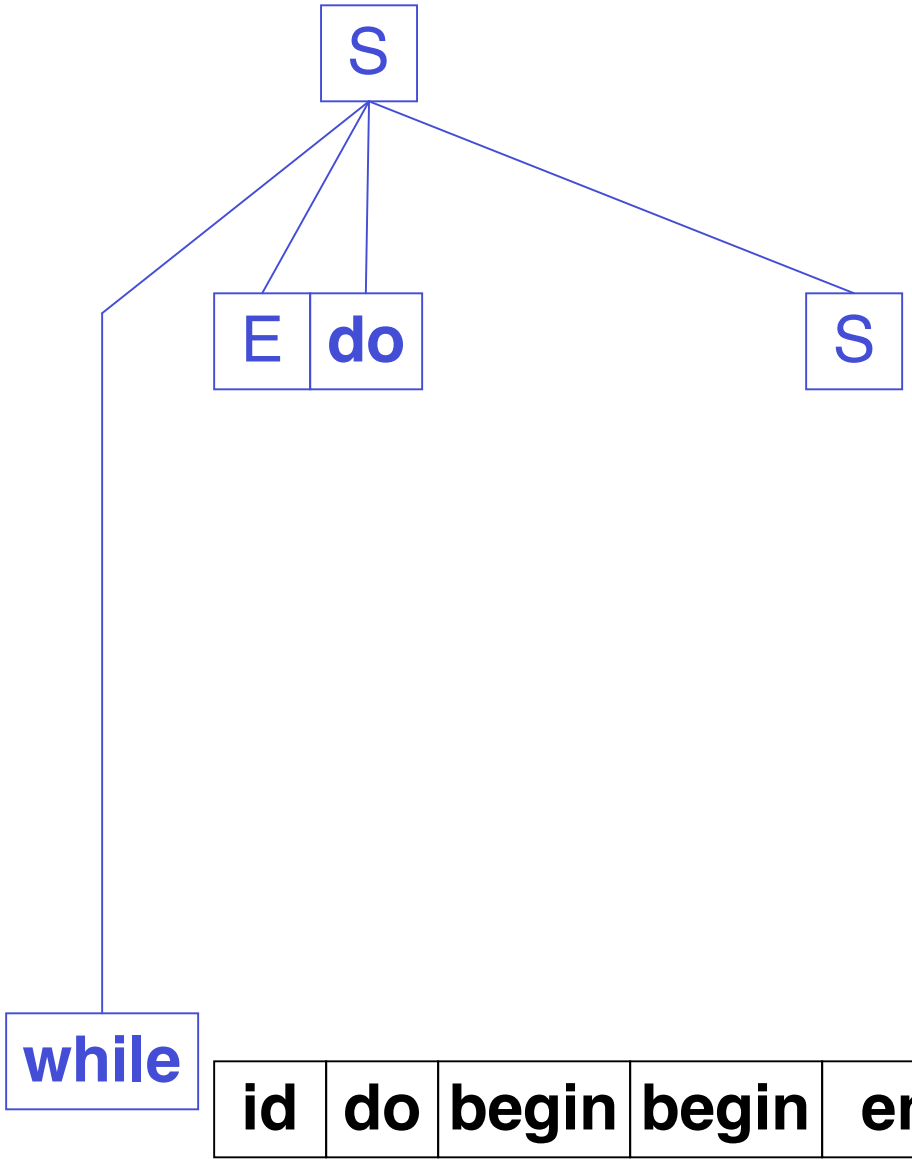
X: while

a: while

Stack:

E
do
S
$

while	id	do	begin	begin	end	;	end	$



At top of loop

X:

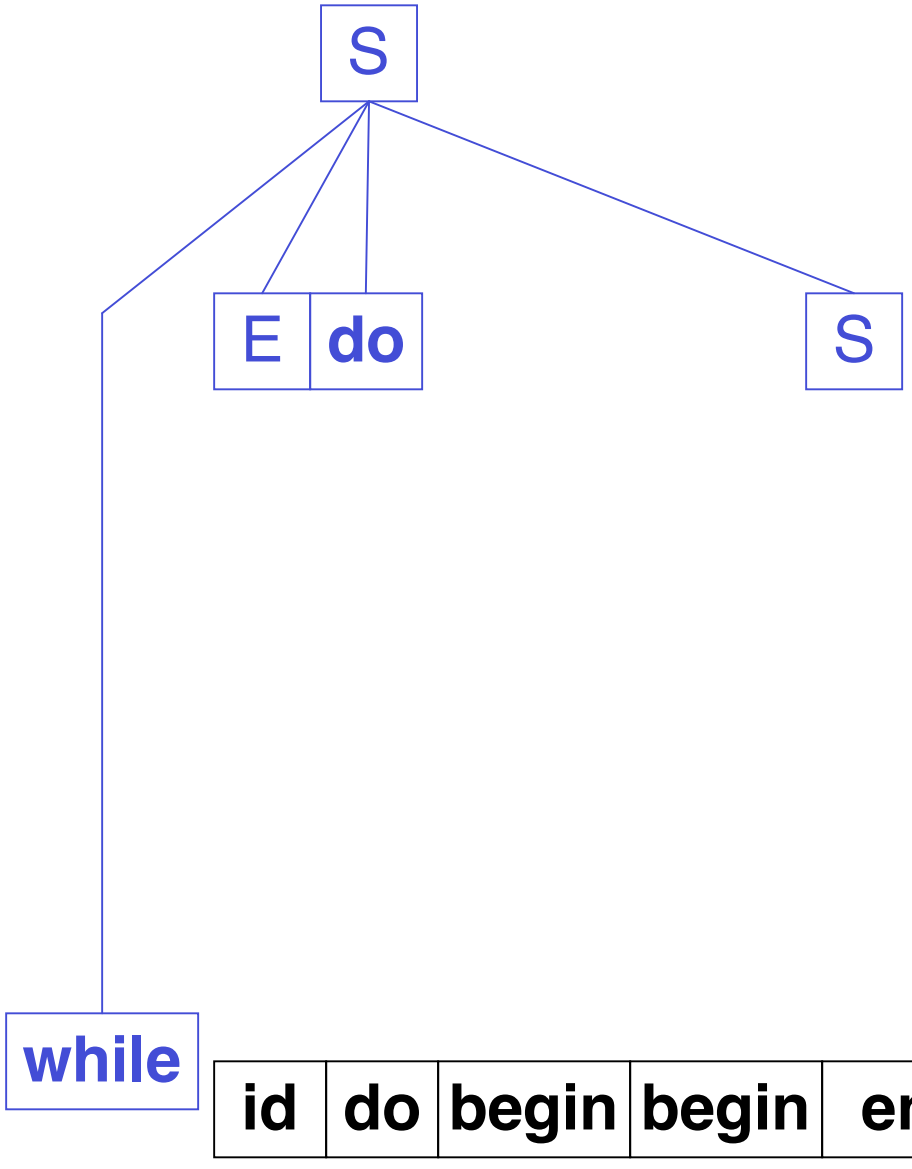
a:

Stack:

E
do
S
$

Action: 6 `E ::= id`

Mid loop



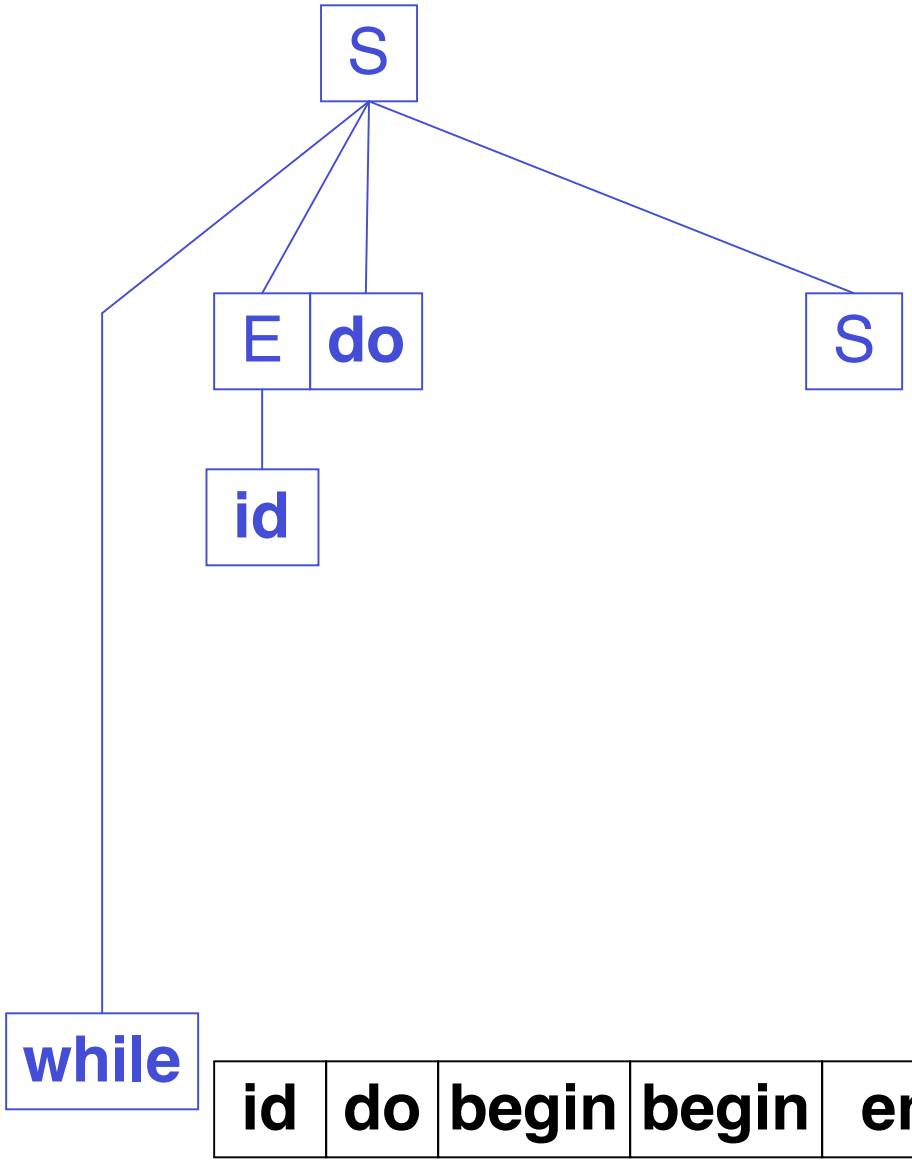
id	do	begin	begin	end	;	end	$

X: E

a: id

Stack:

do
S
$



id	do	begin	begin	end	;	end	$

At top of loop

X:

a:

Stack:

id
do
S
$

Action: Match

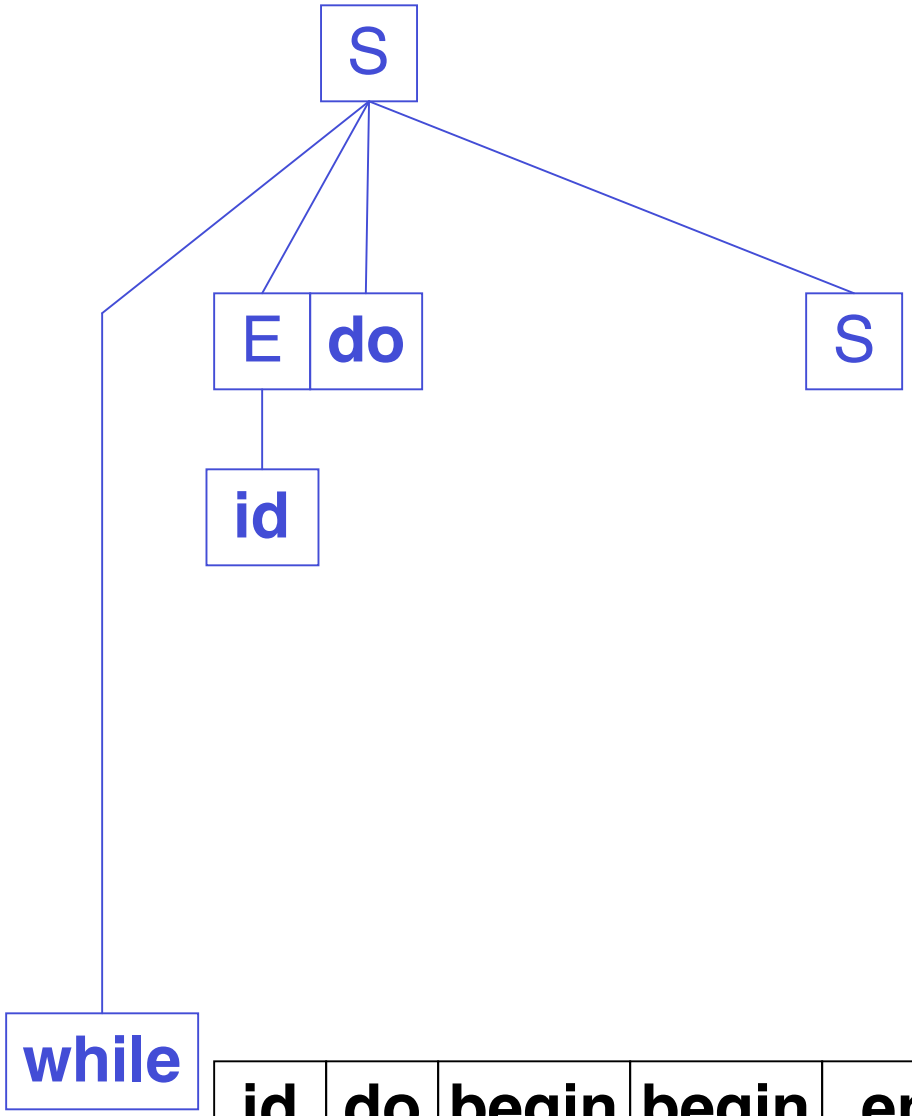
Mid loop

X: id

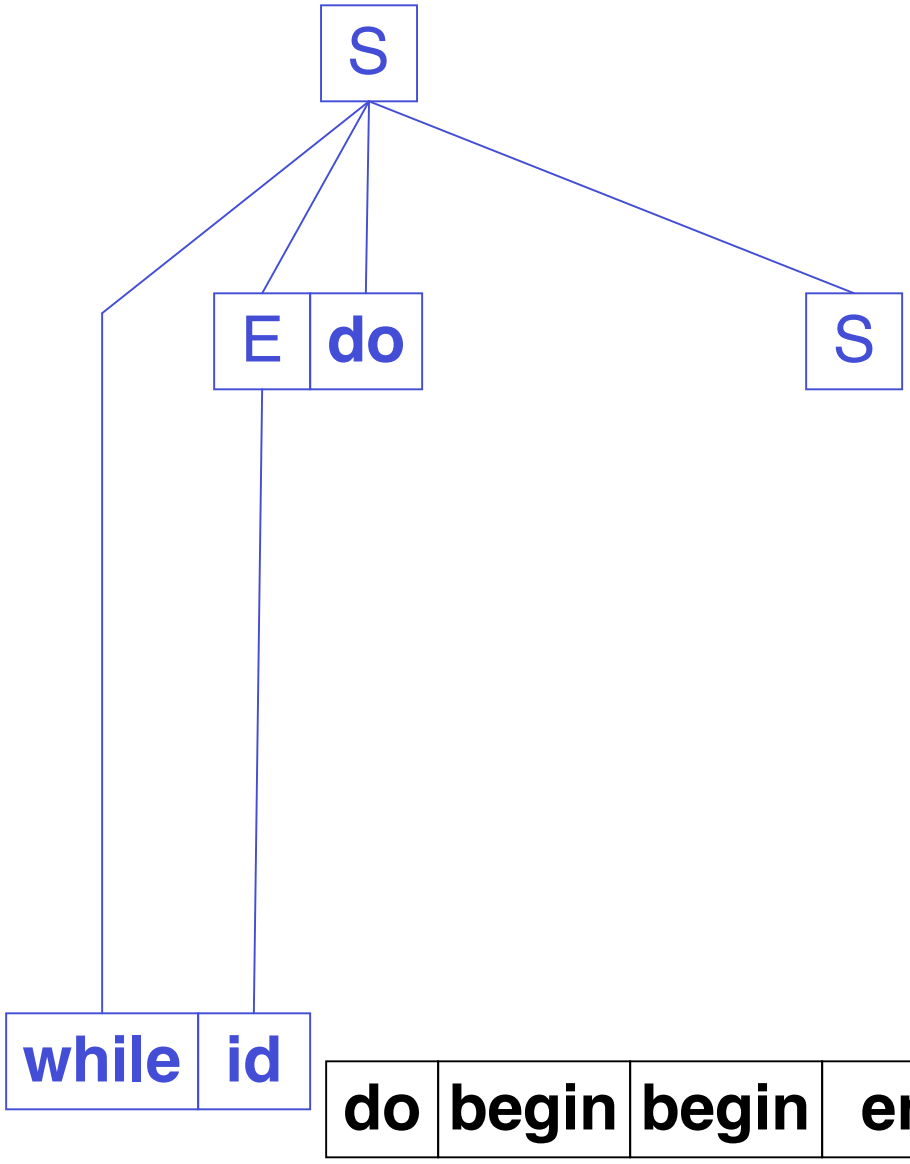
a: id

Stack:

do
S
$



id	do	begin	begin	end	;	end	$



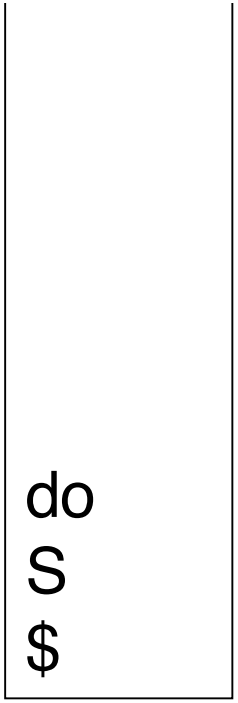
do	begin	begin	end	;	end	$

At top of loop

X:

a:

Stack:



Action: `Match`

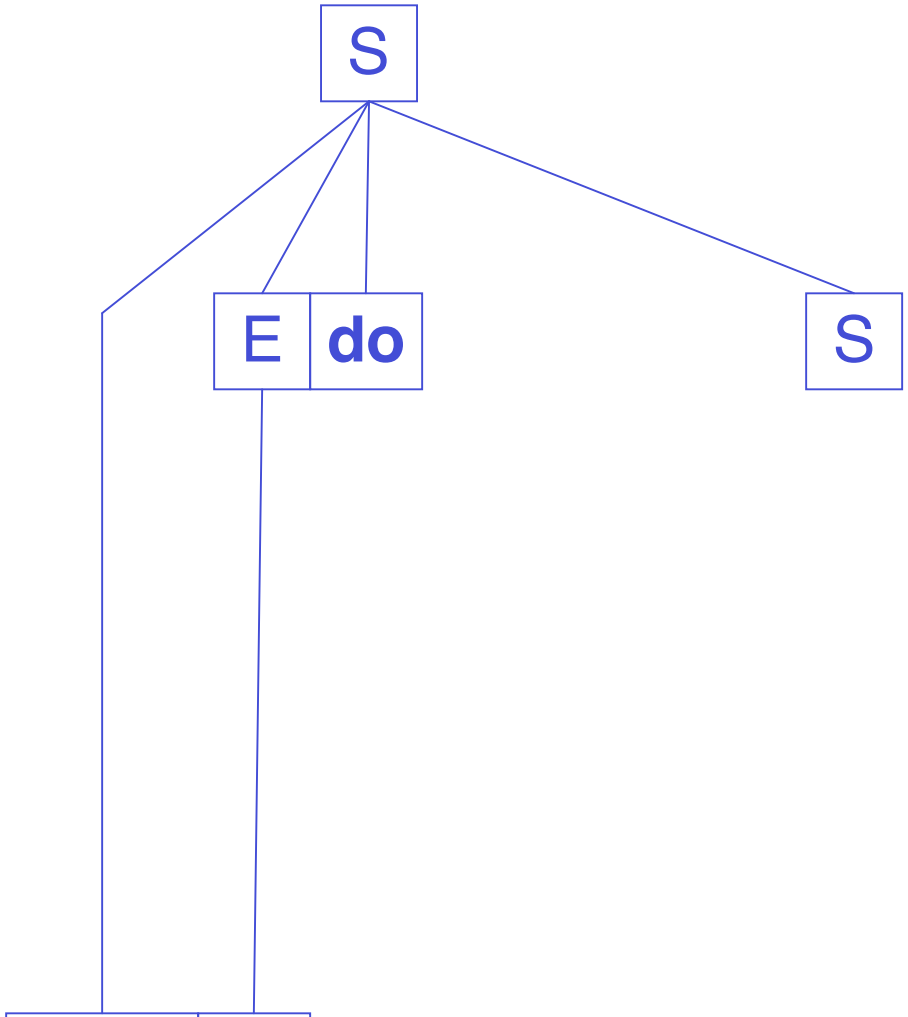
Mid loop

X: do

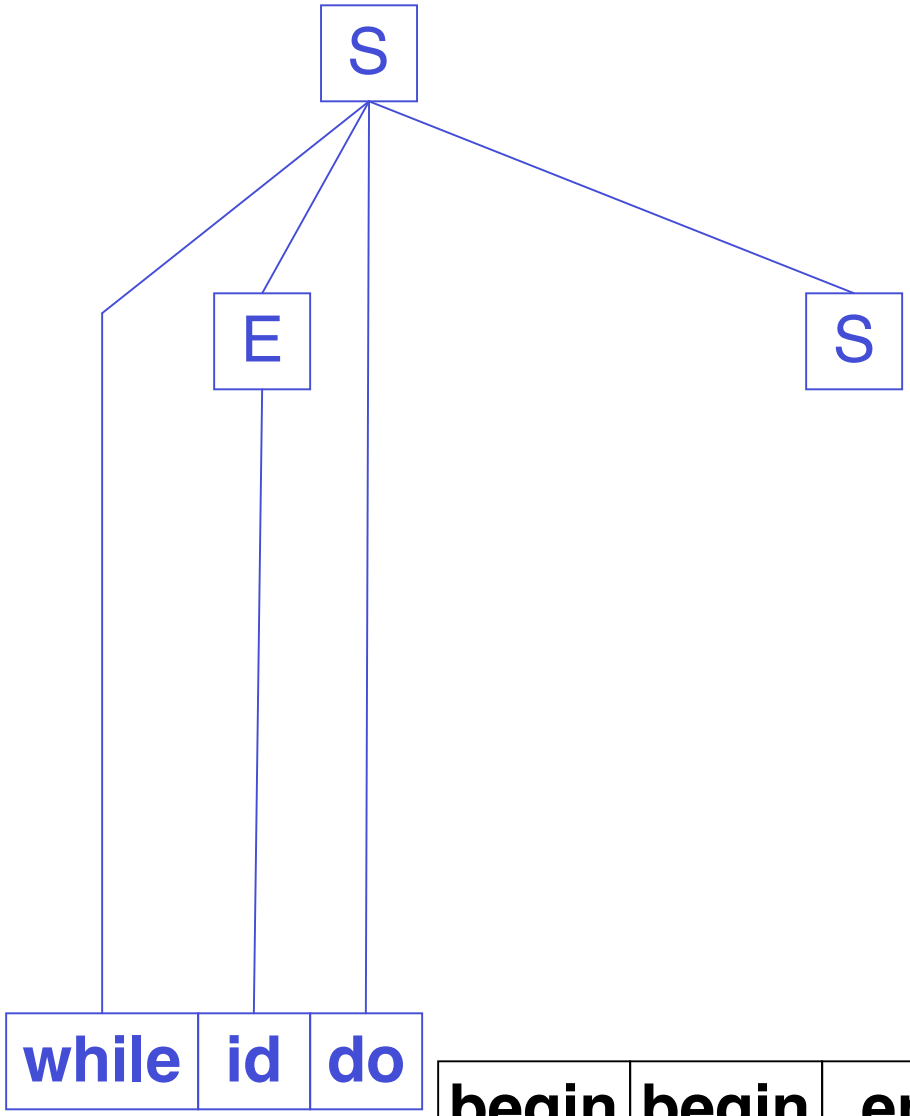
a: do

Stack:

S
$



do	begin	begin	end	;	end	$



begin	**begin**	**end**	**;**	**end**	**$**

At top of loop

X:

a:

Stack:

S
$

Action: 3 `S ::= begin Ss end`

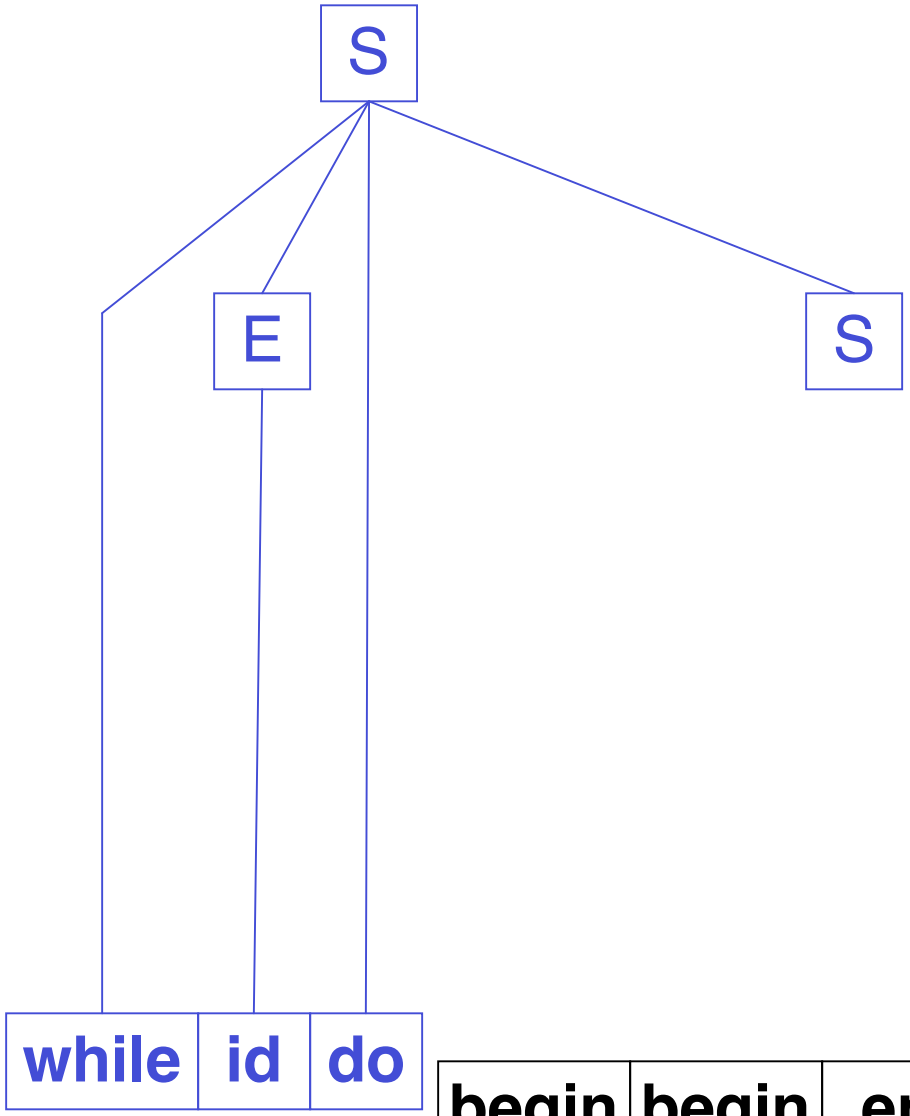
Mid loop

X: S

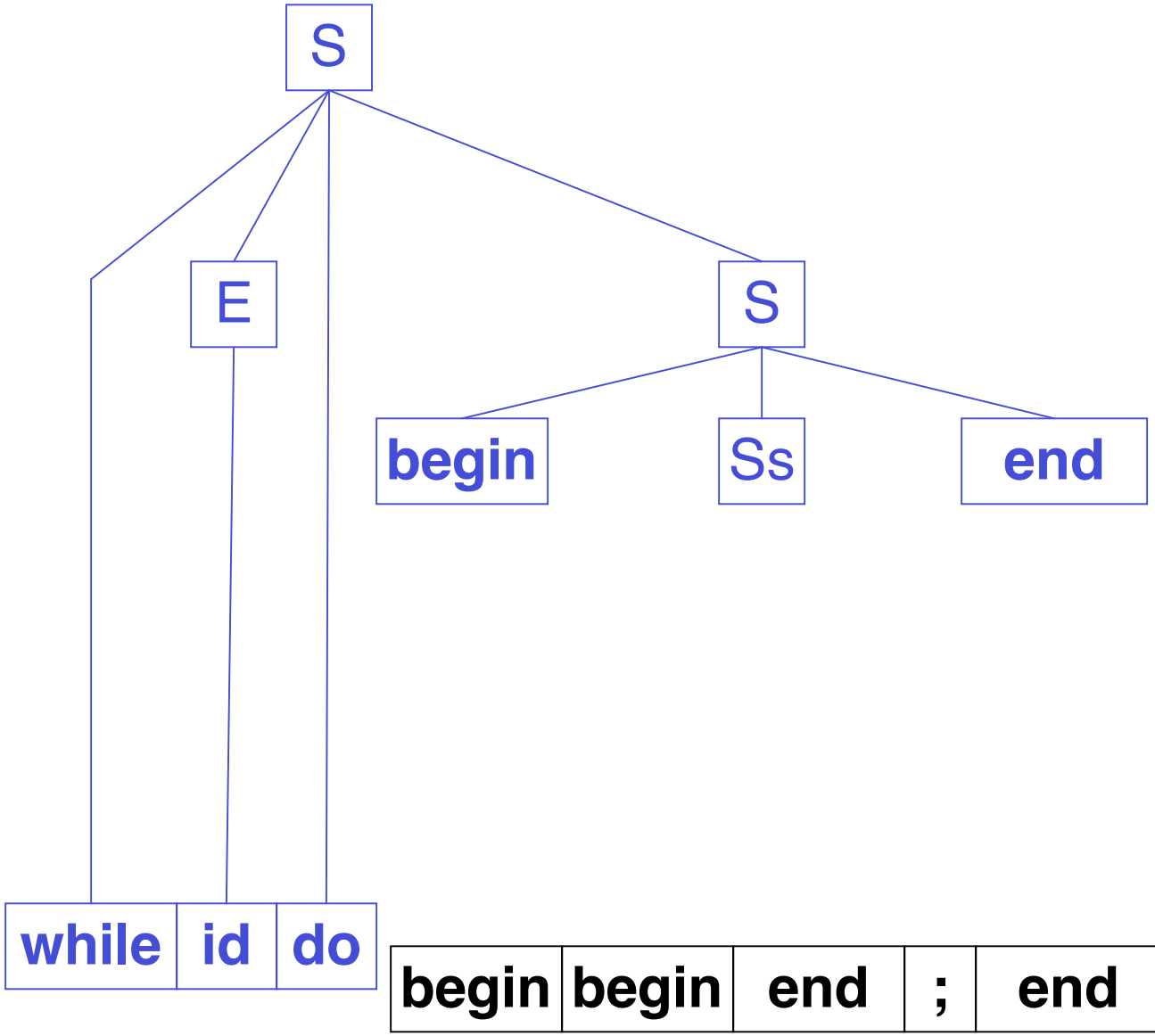
a: begin

Stack:

$



begin	begin	end	;	end	$



begin	begin	end	;	end	$

At top of loop

X:

a:

Stack:

begin
Ss
end
$

Action: `Match`

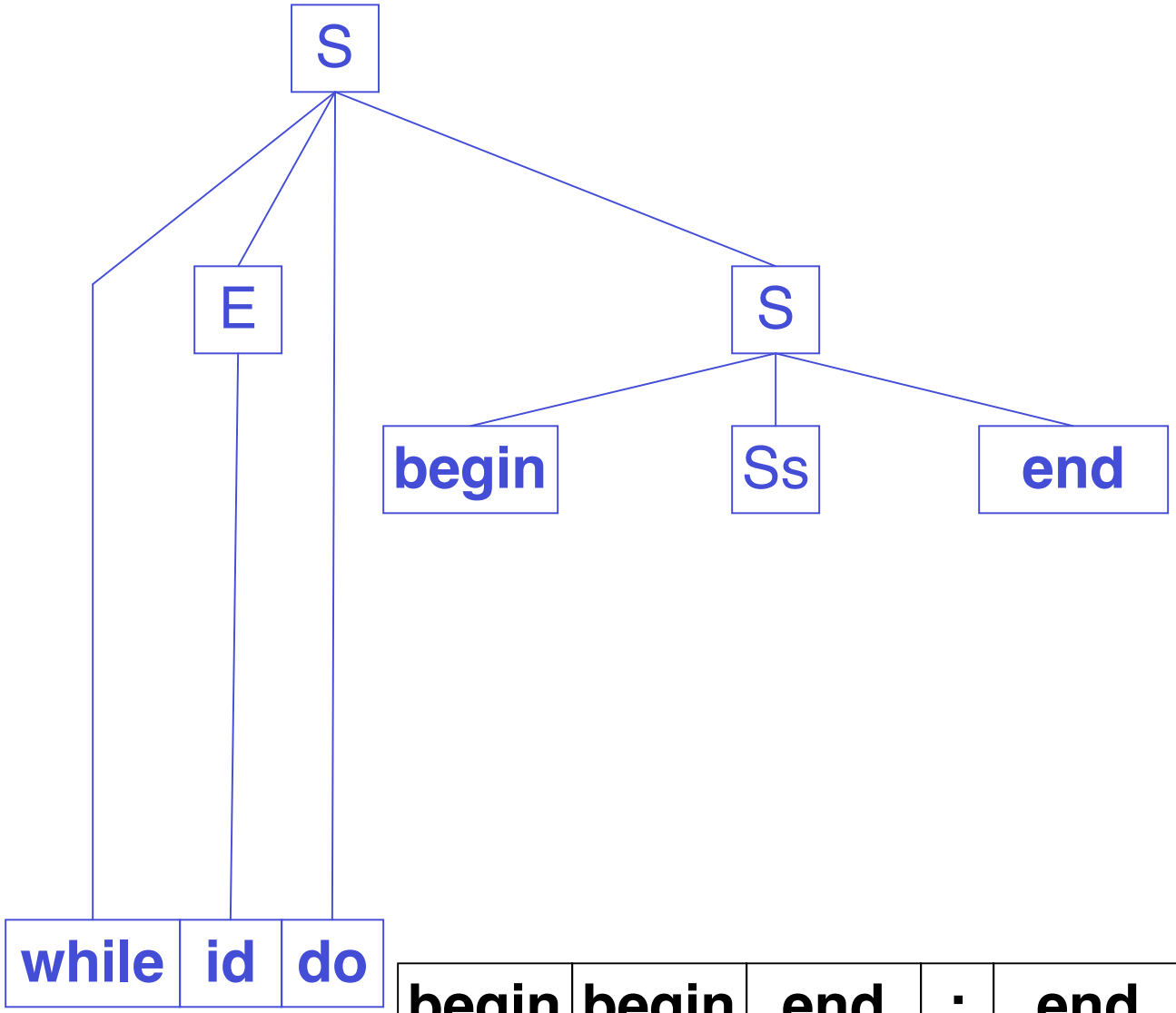
Mid loop

X: begin

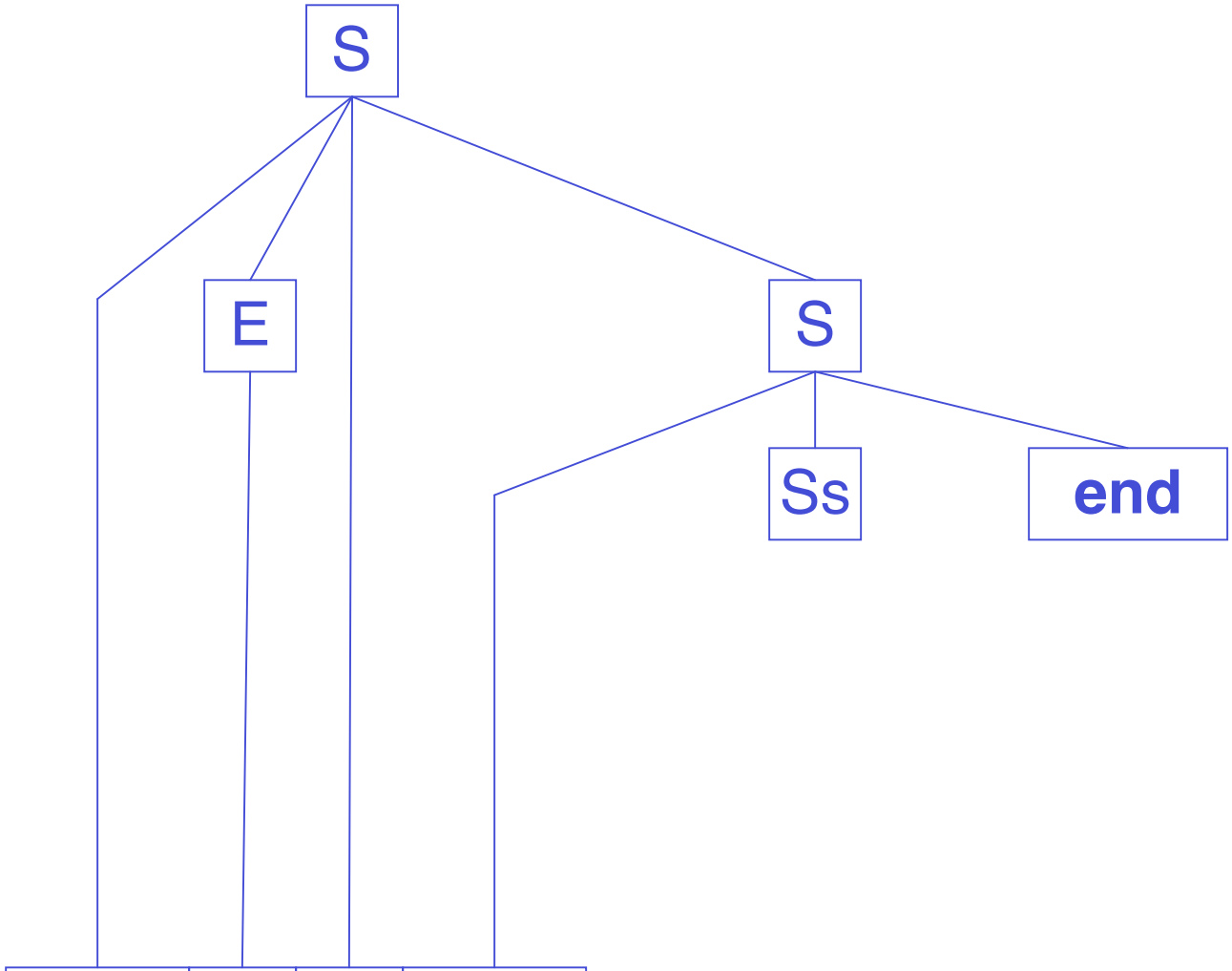
a: begin

Stack:

Ss
end
$



begin	begin	end	;	end	$



begin	end	;	end	$

At top of loop

X:

a:

Stack:

Ss
end
$

Action: 4 `Ss ::= S ; Ss`

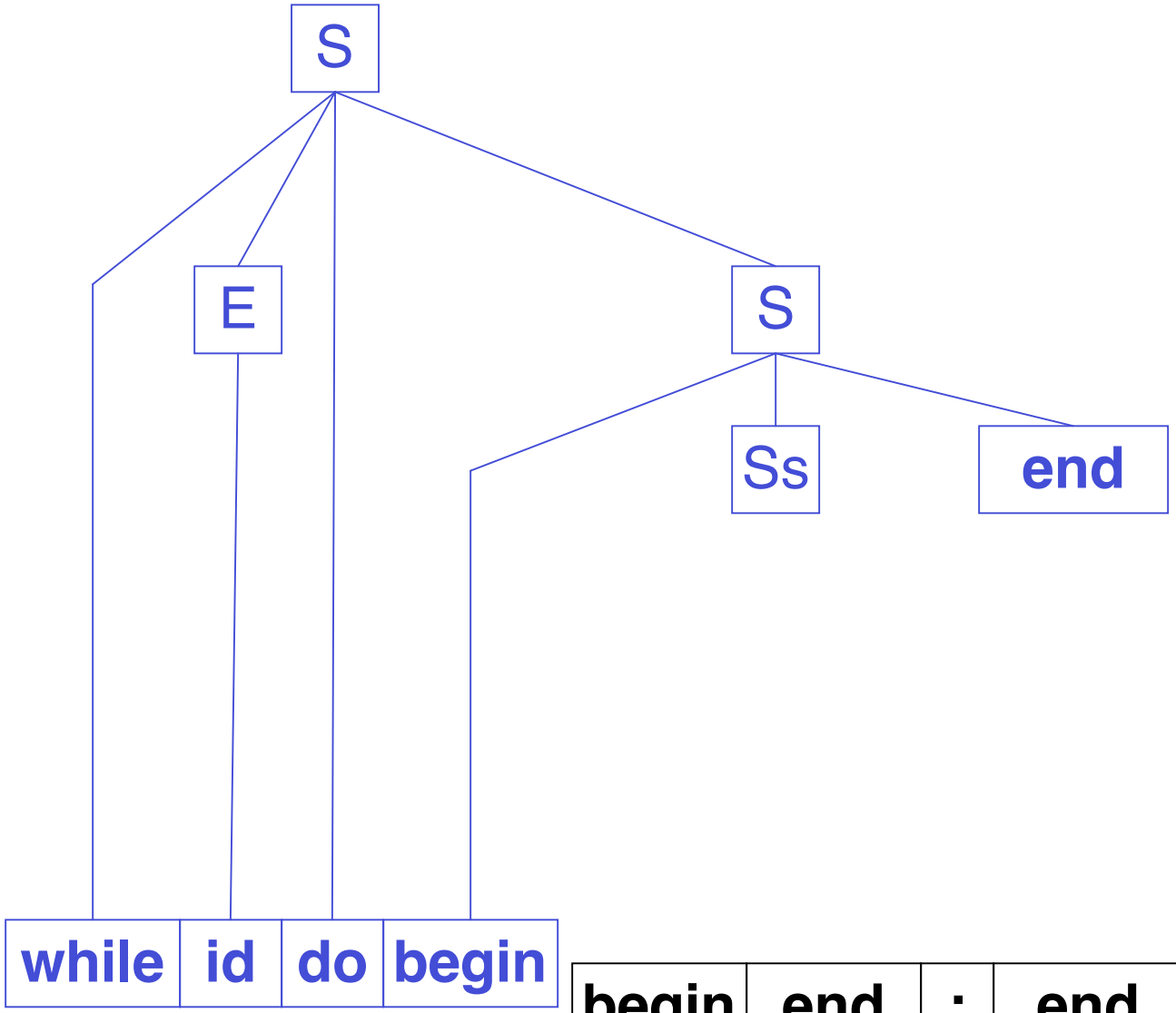
Mid loop

X: Ss

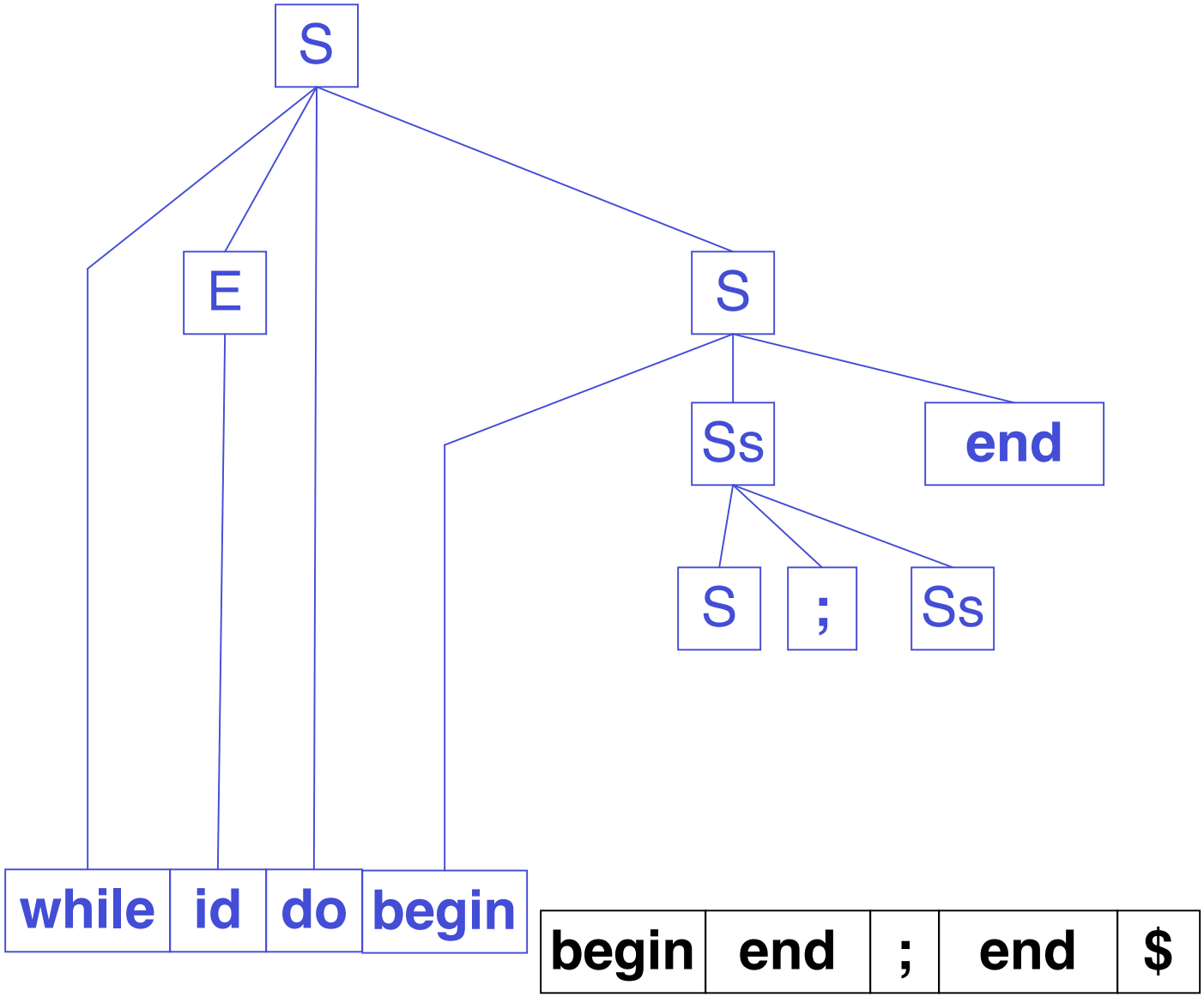
a: begin

Stack:

end
$



begin	end	;	end	$



begin	end	;	end	$

At top of loop

X:

a:

Stack:

S
;
Ss
end
$

Action: 3 `S ::= begin Ss end`

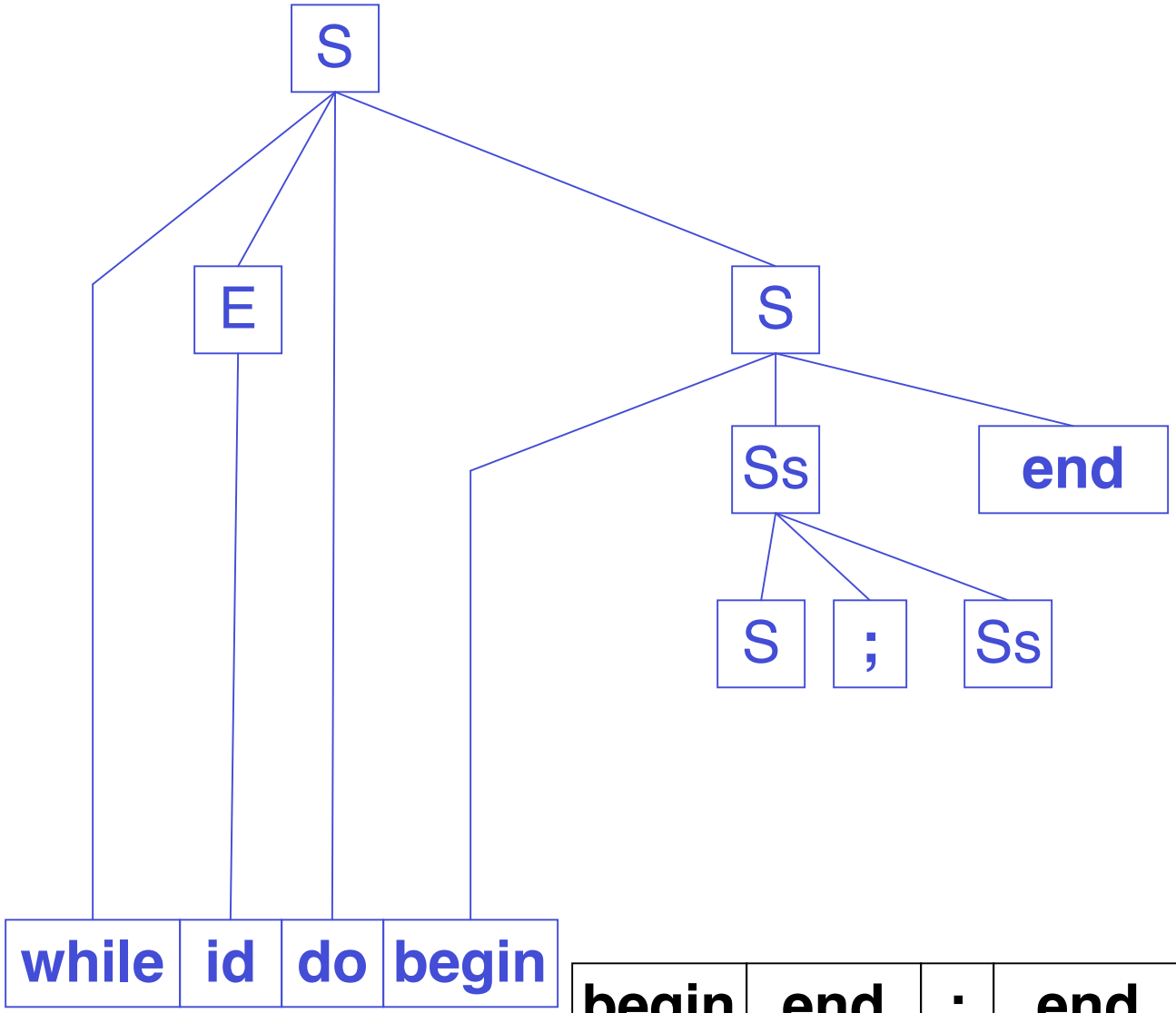
Mid loop

X: S

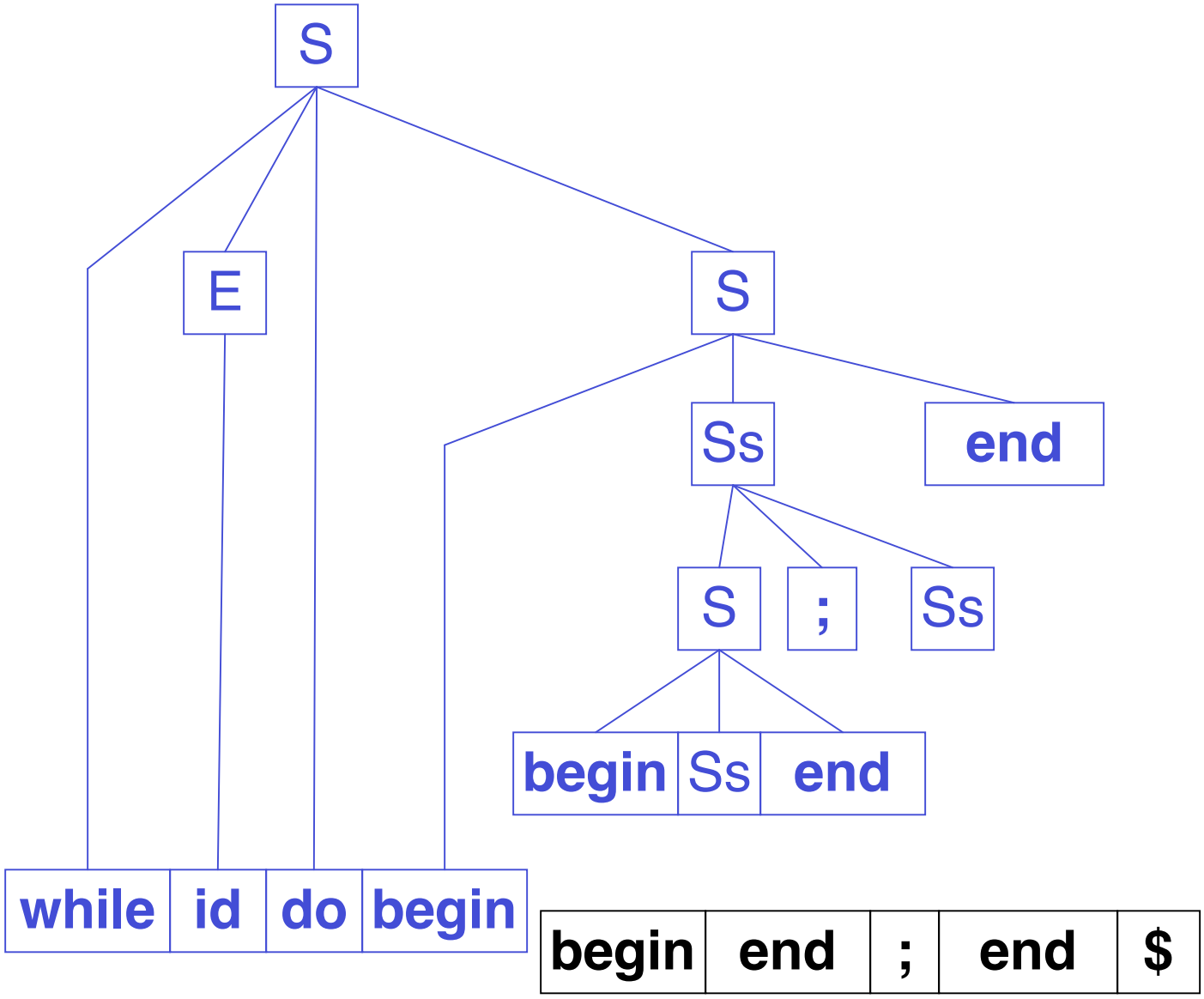
a: begin

Stack:

;
Ss
end
$



begin	end	;	end	$



Top of loop

X:

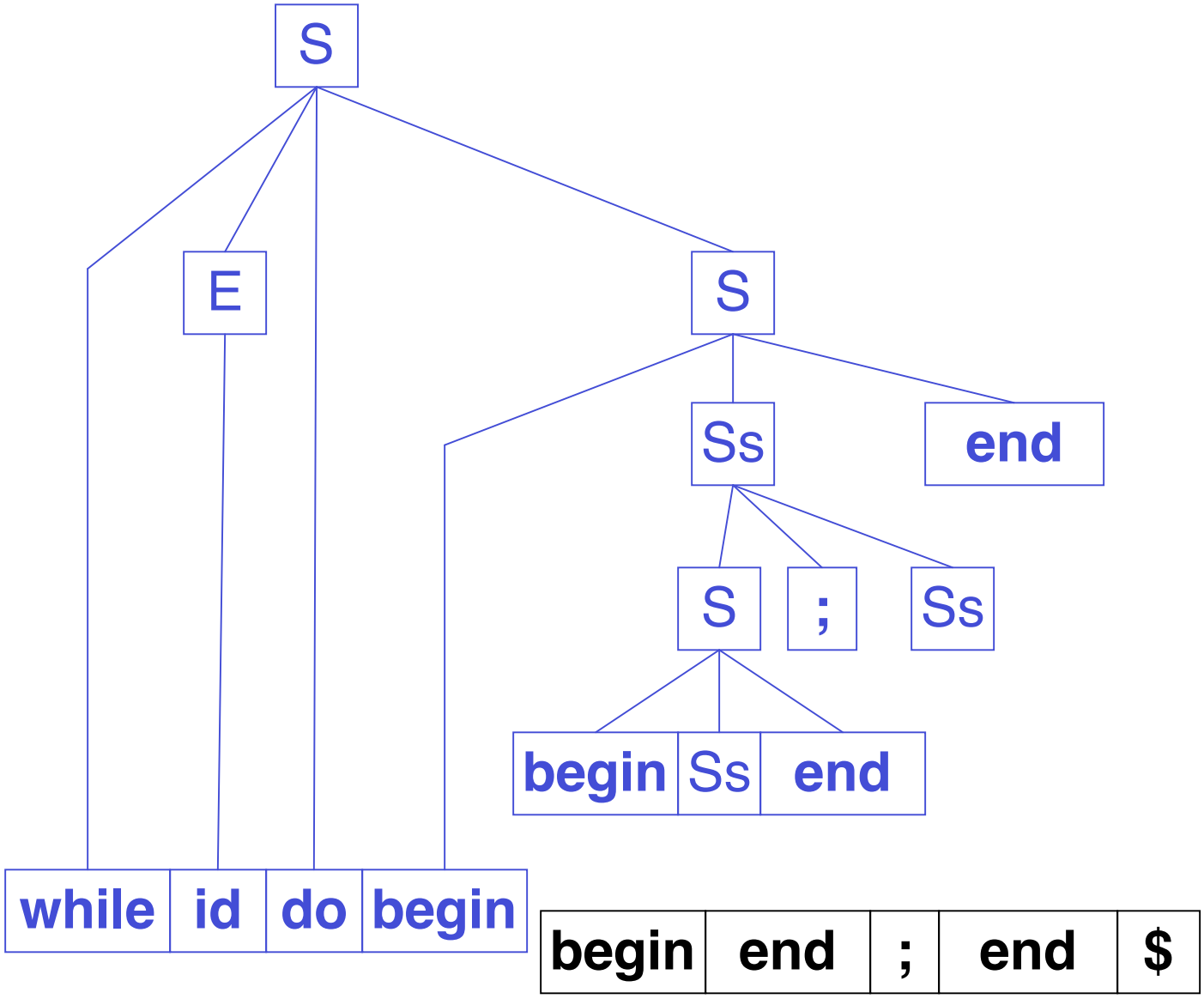
a:

Stack:

begin
Ss
end
;
Ss
end
$

Action: `Match`

Mid loop

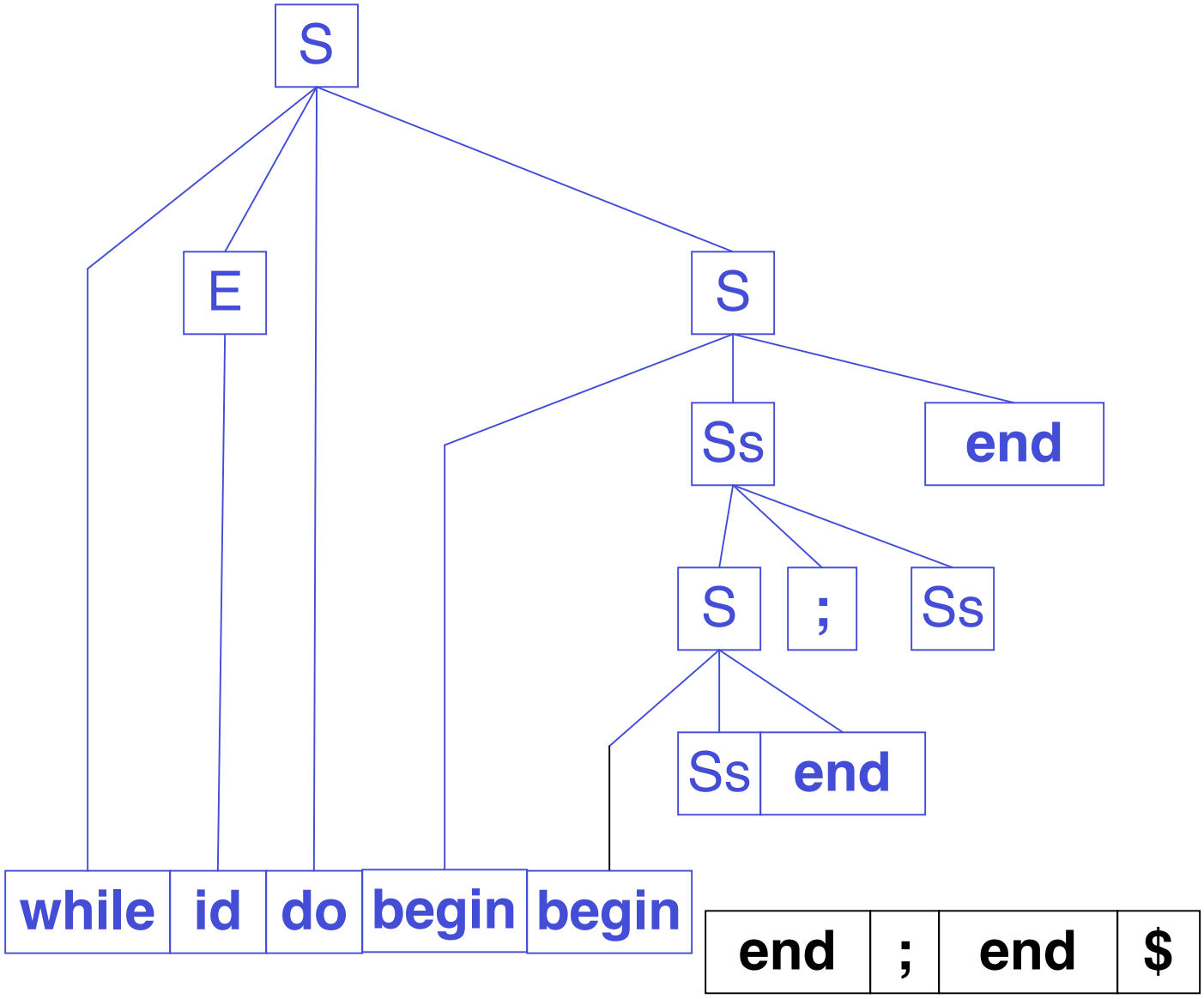


X: begin

a: begin

Stack:

Ss
end
;
Ss
end
$



Top of loop

X:

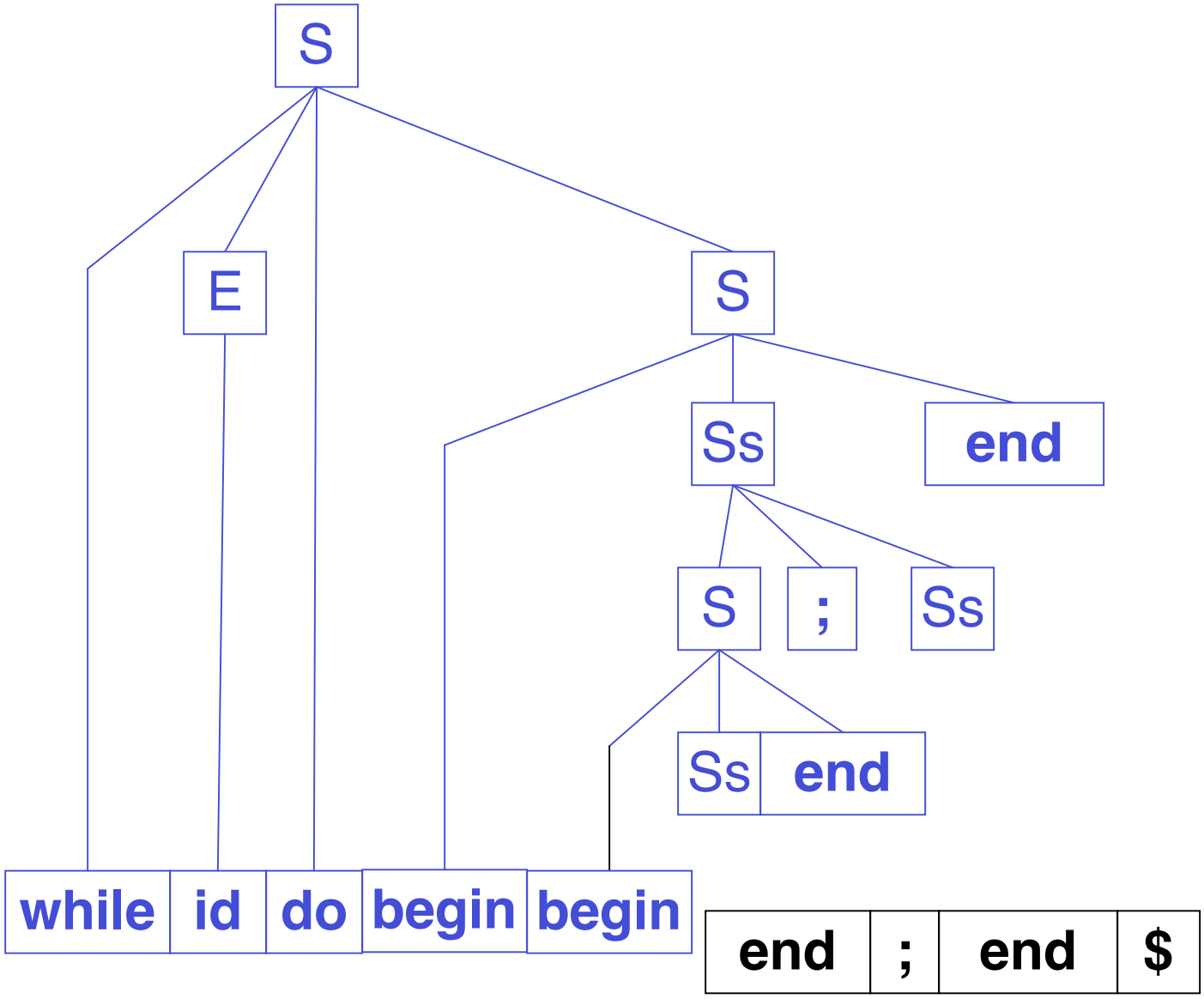
a:

Stack:

Ss
end
;
Ss
end
$

Action: 5 Ss ::= ε

Mid loop

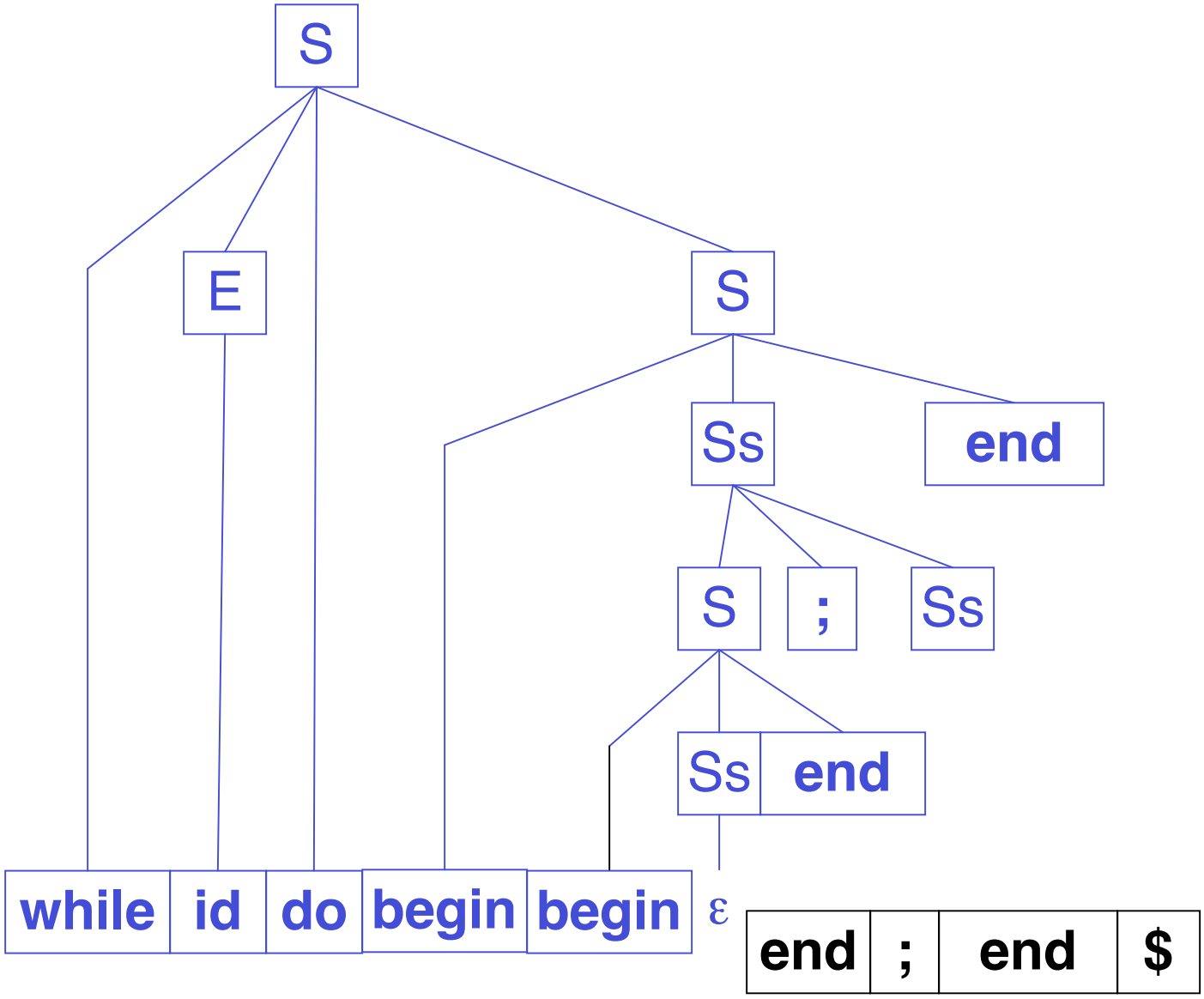


X: Ss

a: end

Stack:

end
;
Ss
end
$



Top of loop

X:

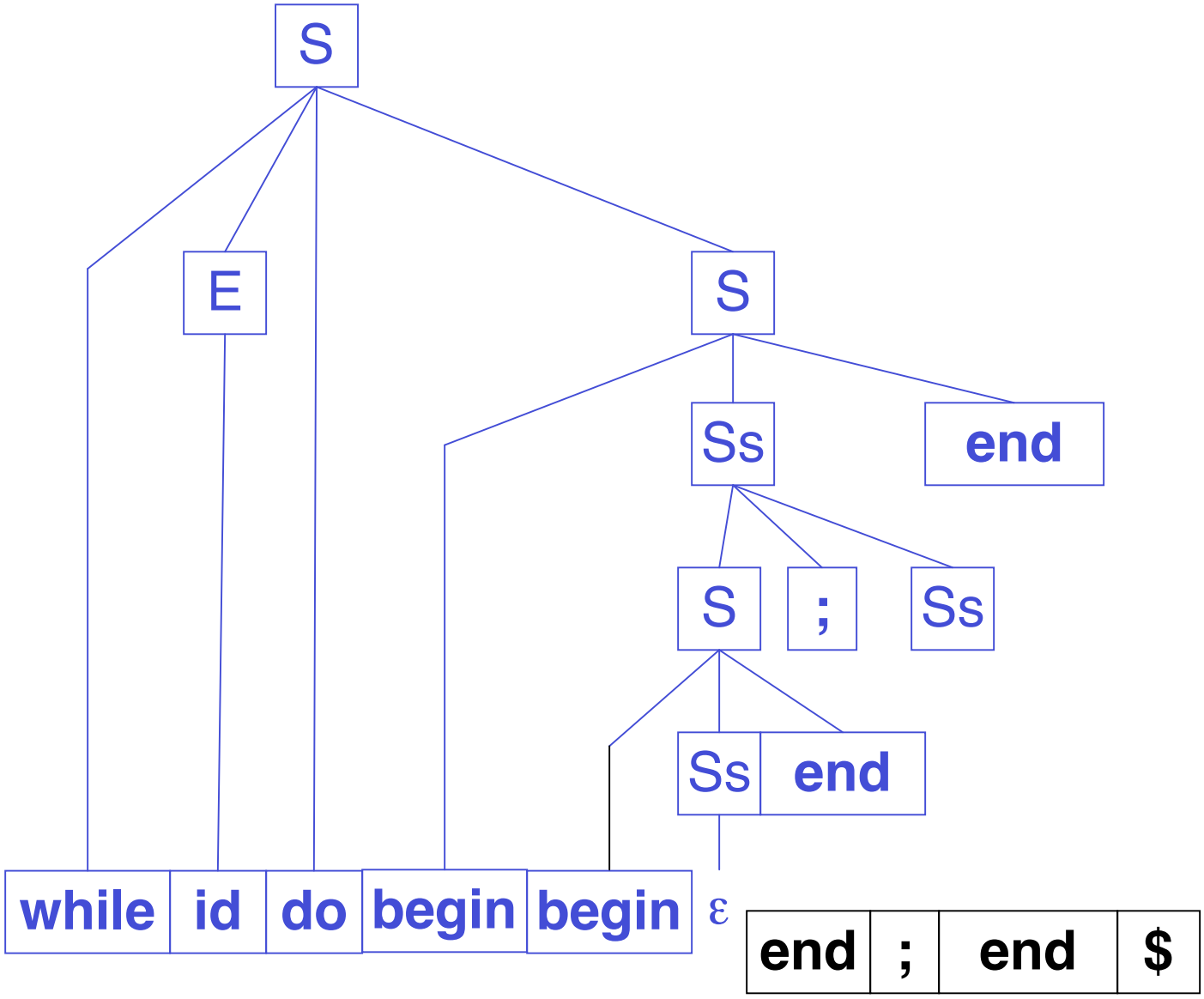
a:

Stack:

end
;
Ss
end
$

Action: `Match`

Mid loop

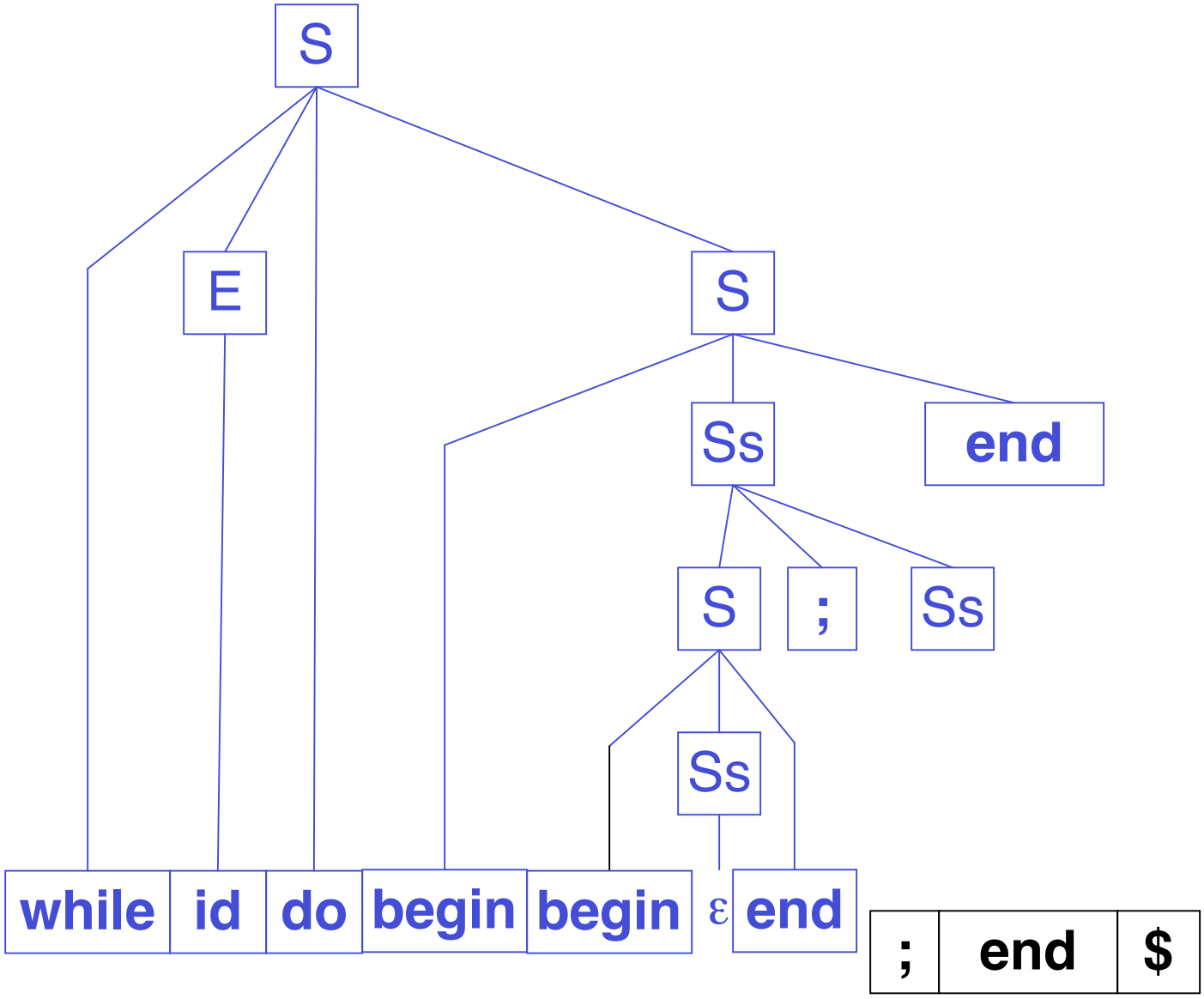


X: end

a: end

Stack:

;
Ss
end
$



Top of loop

X:

a:

Stack:

;
Ss
end
$

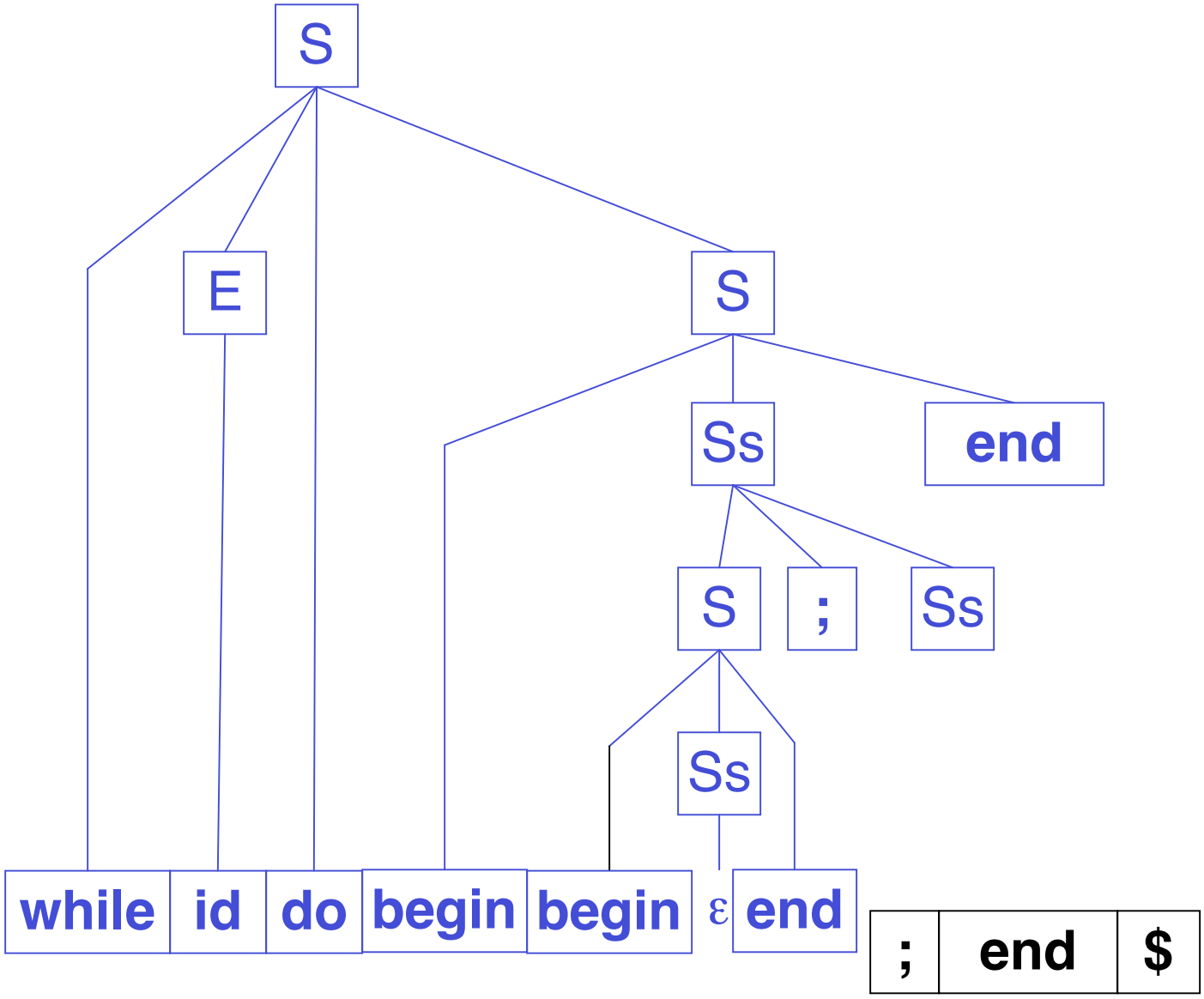
Action: `Match`

Mid loop

X: ;

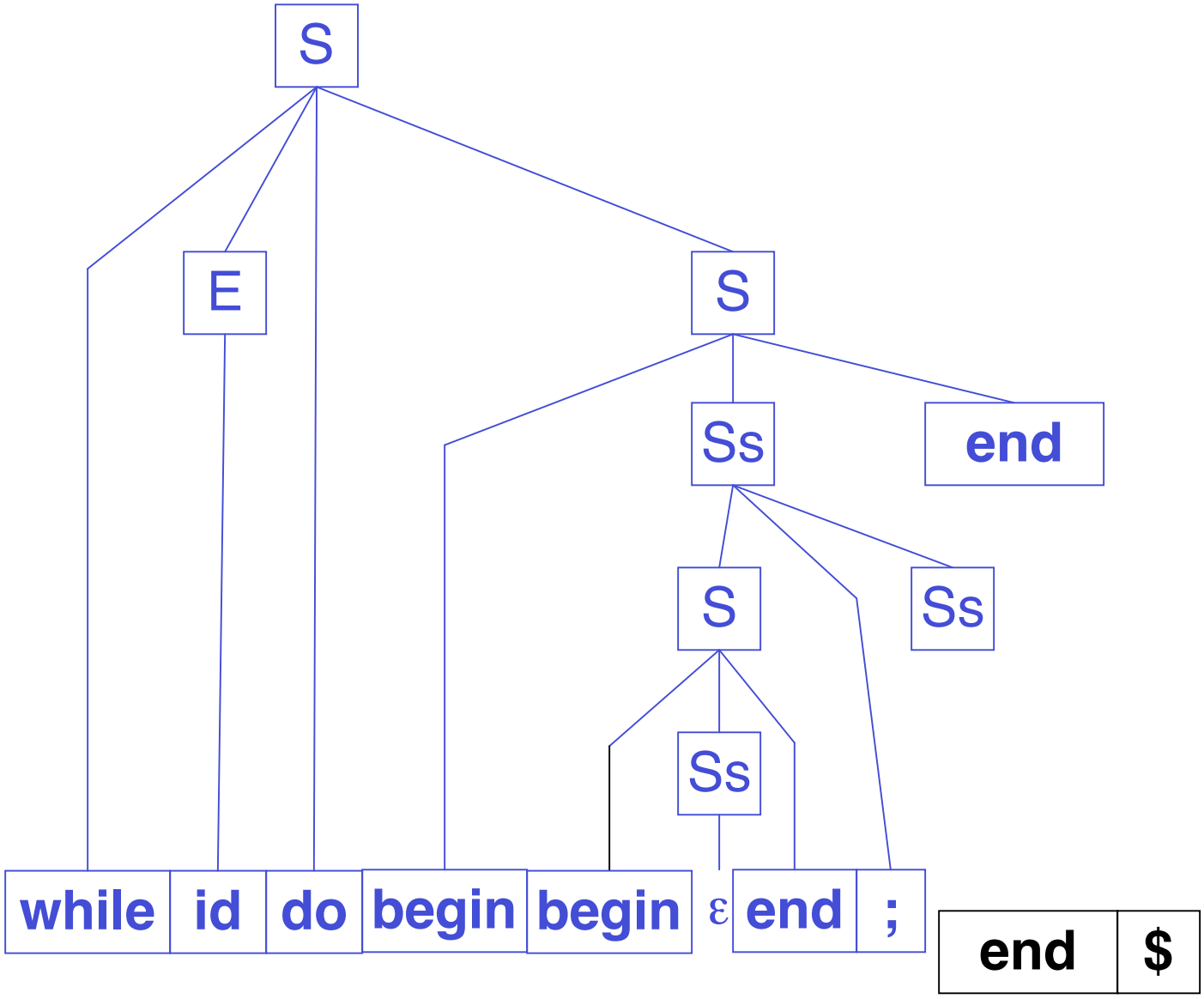
a: ;

Stack:



;	**end**	**$**

Ss
end
$



Top of loop

X:

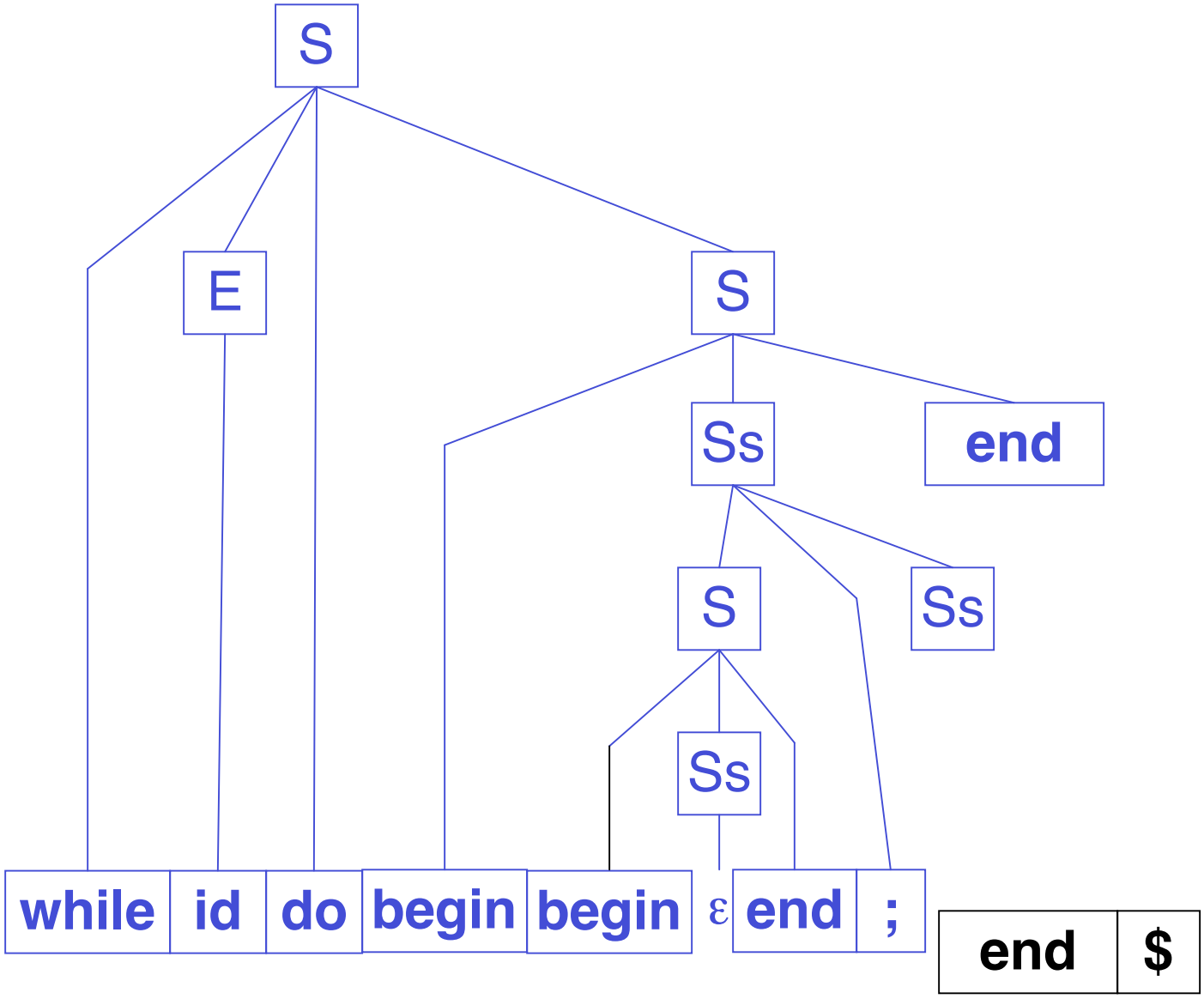
a:

Stack:

Ss
end
$

Action: 5 Ss ::= ε

Mid loop

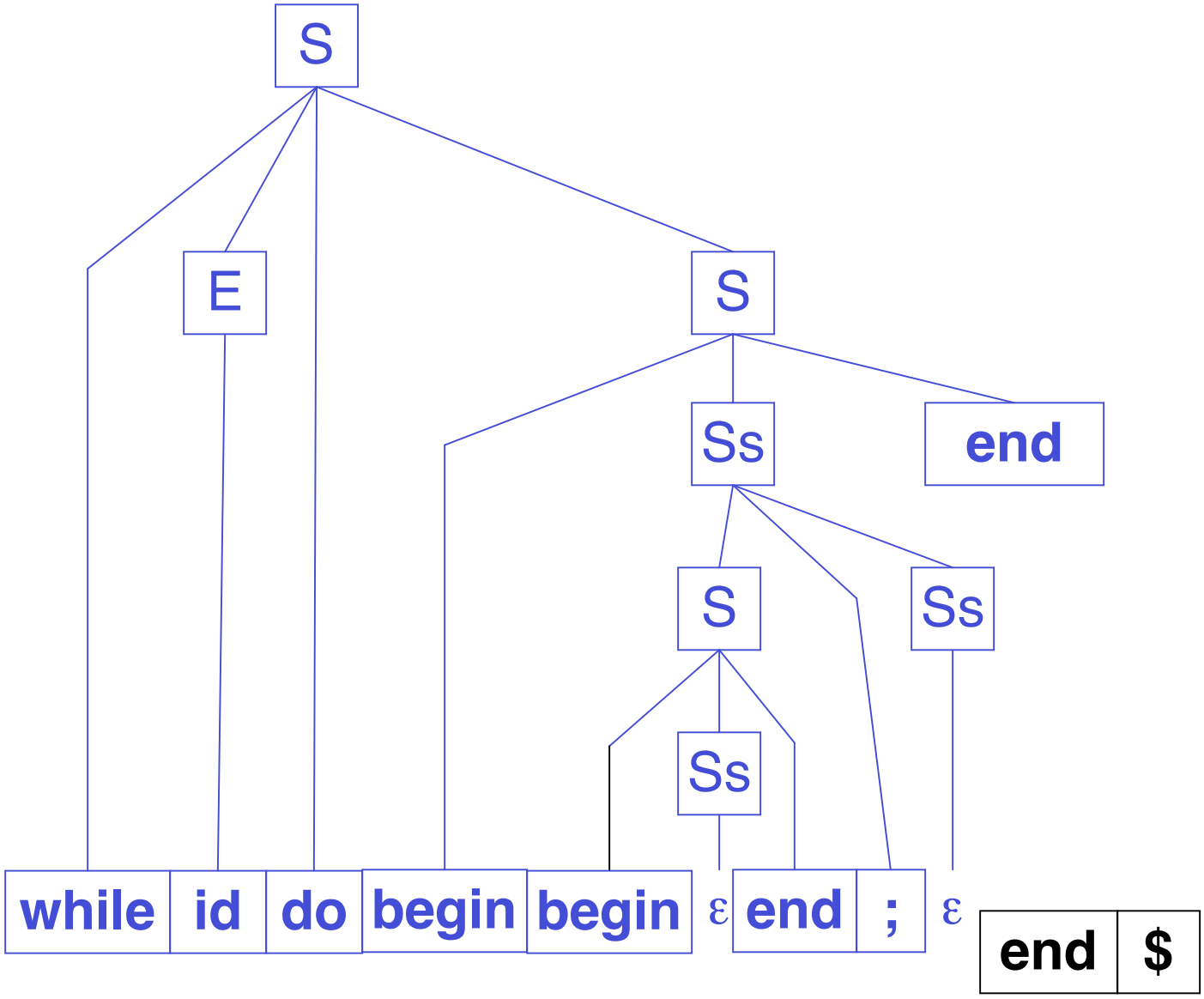


X: Ss

a: end

Stack:

end
$



Top of loop

X:

a:

Stack:

end
$

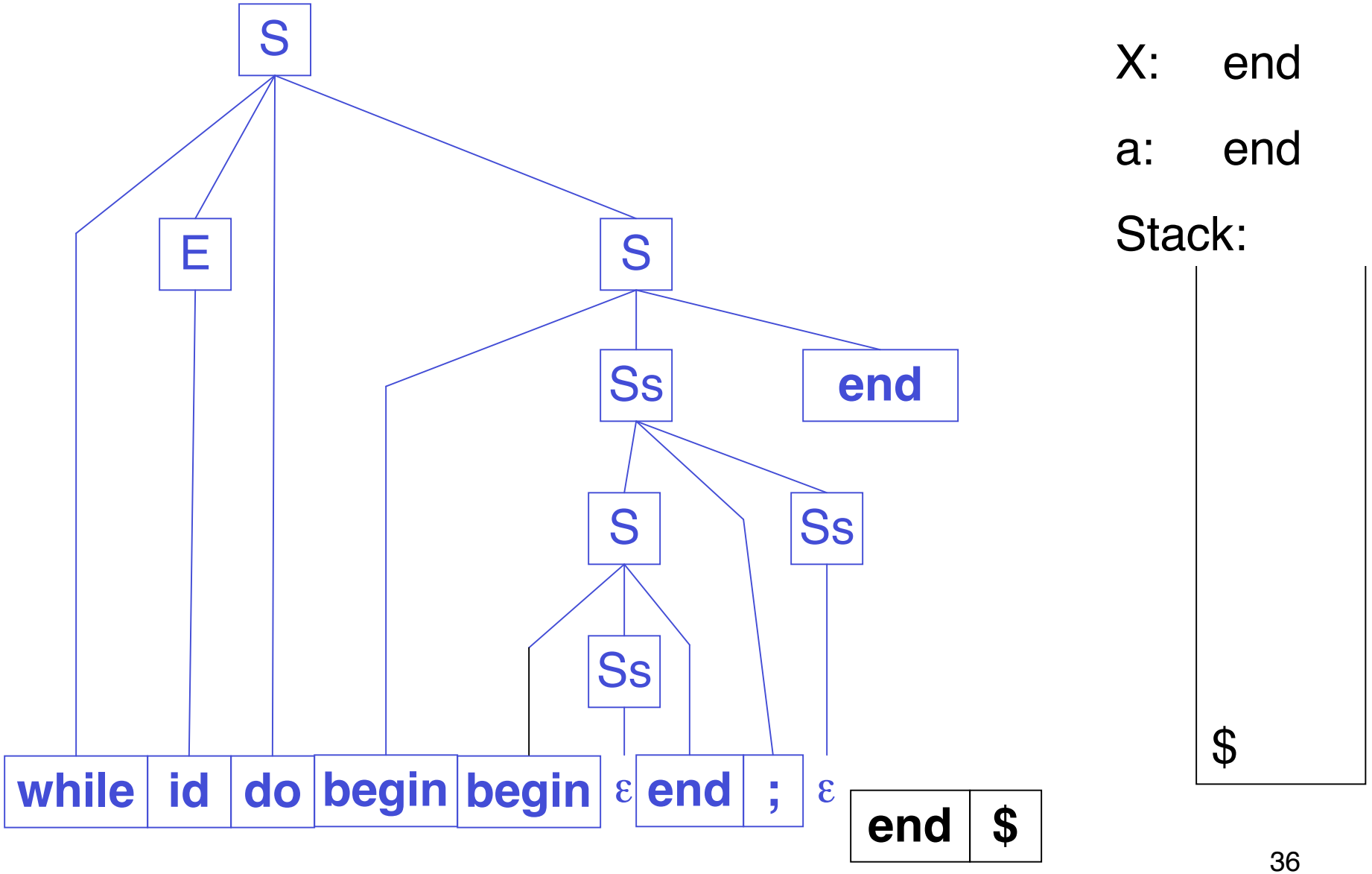
Action: `Match`

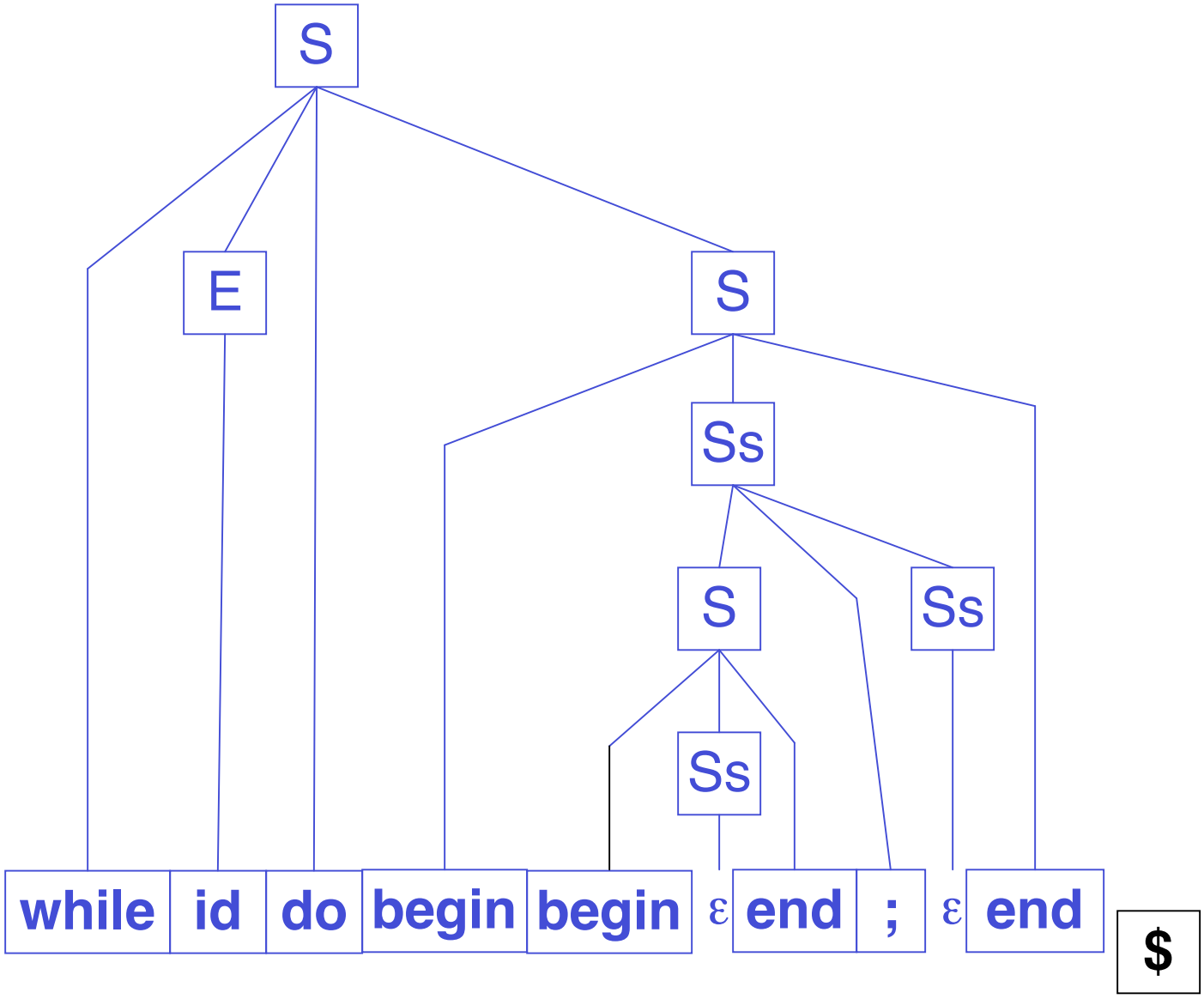
Mid loop

X: end

a: end

Stack:



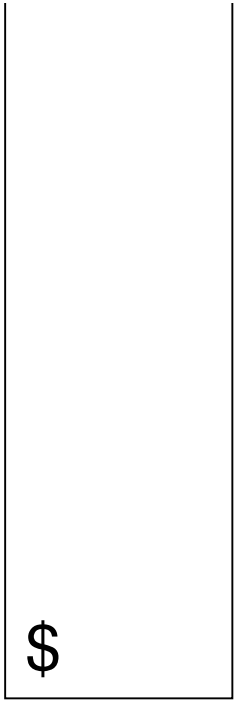


Top of loop

X:

a:

Stack:



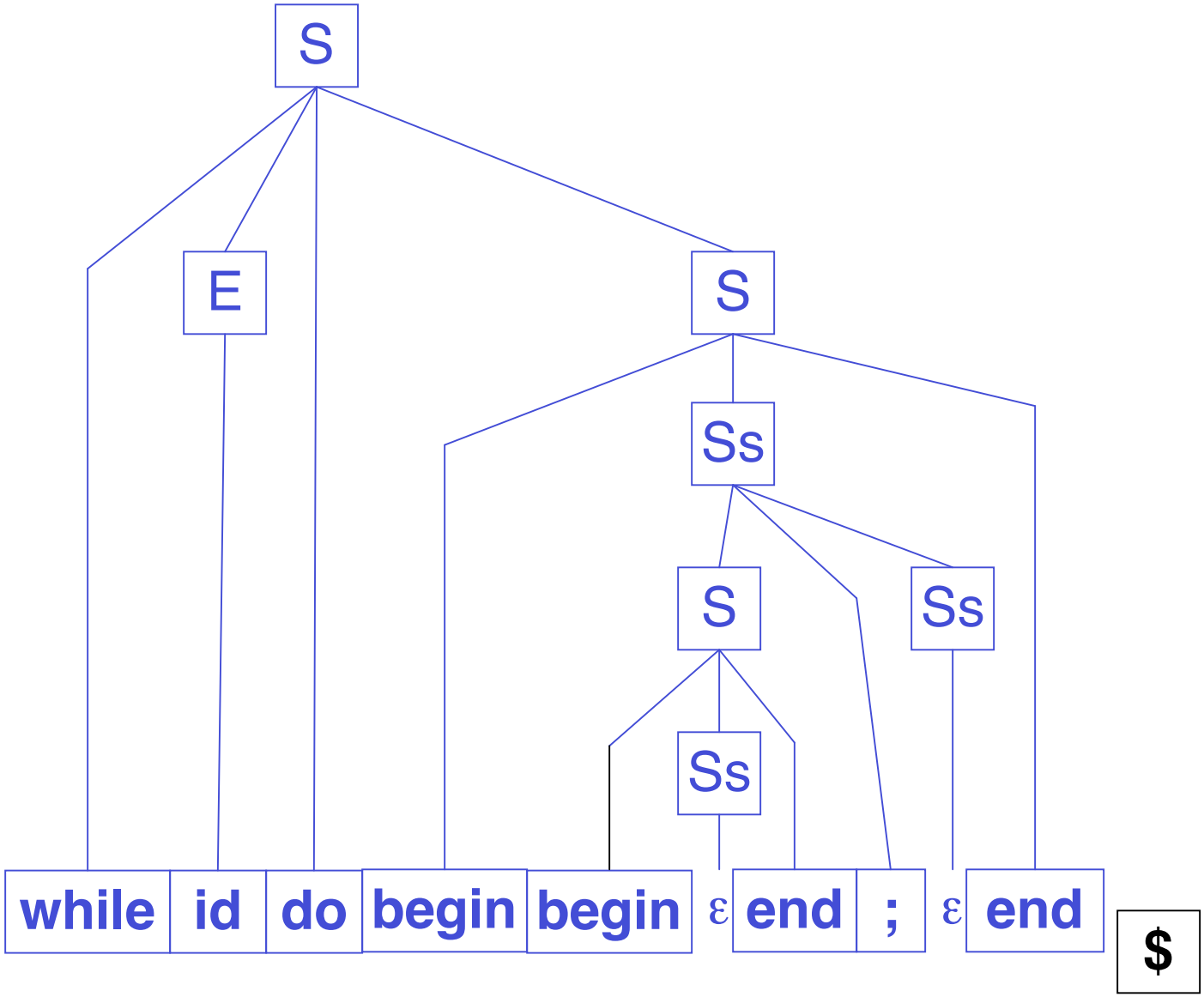
Action: `Match`

Mid loop

X: $

a: $

Stack:



Top of loop

X:

a:

Stack: