Highly modified Augmented Transition Networks)

This allows ambiguity/backtracking, not in the original design, and does not use any augmentations.

Terminal symbols are in capital letters,
they represent word classes such as noun and verb, or
specific words such as "the" and "hippopotamus"
Non-terminal symbols are in < and >,
they represent sub-grammars such as NounPhrase and Question
<S> is usually the starting point

Priorities:
things in parentheses (for grouping) or curly brackets for optional
postfix + (any number, at least one, of repetitions) and * (includes none at all) sequences
| for choices (one must be taken, no more than one may be taken)
::= is defined to be or may be replaced by

<S> ::= (A | B) B (<CDE> | <CD> | C) (D (<FG> | <F>) G | F (<GH> | <GI>) H)

or

$$\text{<S> ::=} \begin{pmatrix} A \\ or \\ B \end{pmatrix} \text{ B } \begin{pmatrix} < CDE > \\ or \\ < CD > \\ or \\ C \end{pmatrix} \left(\begin{pmatrix} D \begin{pmatrix} < FG > \\ or \\ < F > \end{pmatrix} G \end{pmatrix} \middle| \begin{pmatrix} or \\ F \begin{pmatrix} < GH > \\ or \\ < GI > \end{pmatrix} H \end{pmatrix} \right)$$

<CDE> ::= C (<DE> | <ED>)

<DE> ::= D E

<ED> ::= E D

<CD> ::= C D

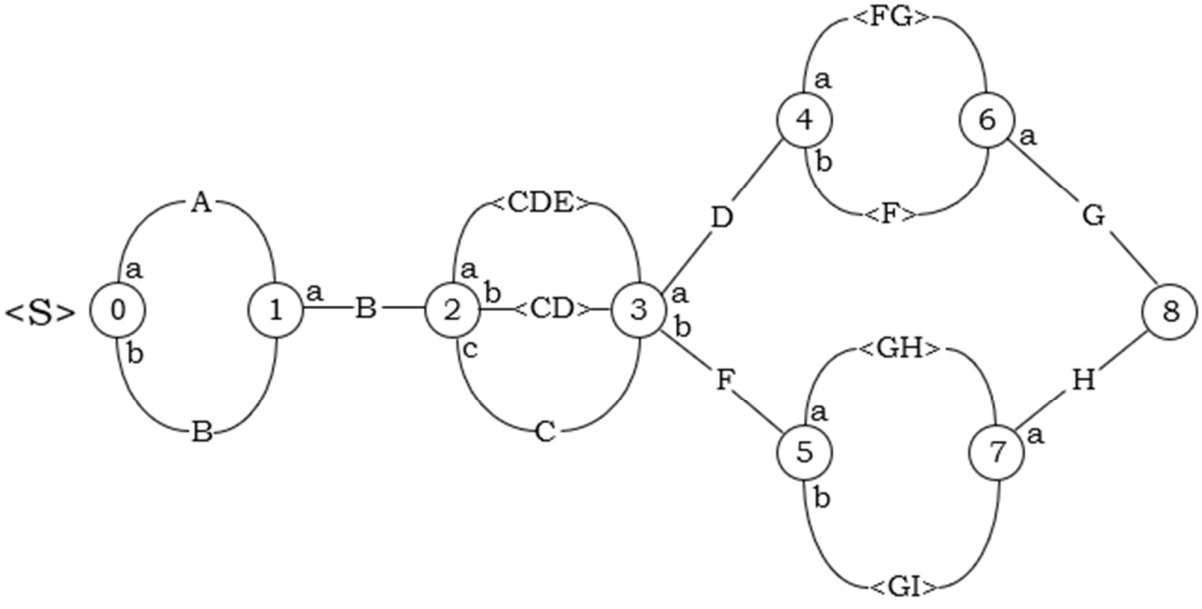
<GH> ::= G H

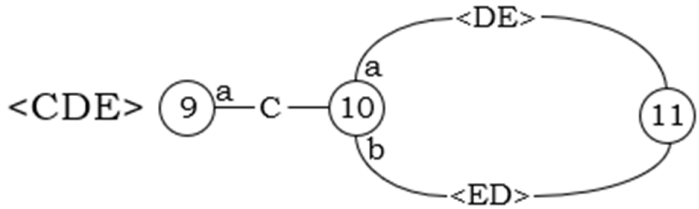
<GI> ::= G I

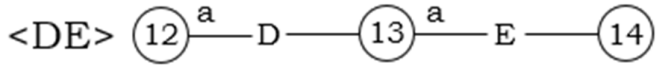
<FG> ::= F G

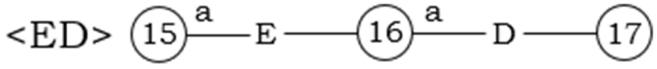
<F> ::= F

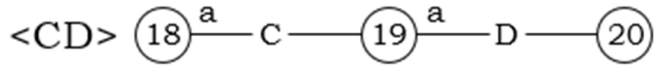
States have numbers,
Arcs from each state are labelled a, b, c, etc,
Direction: arc leading out has label, arc leading in has no label,

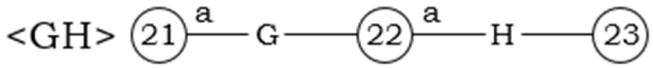


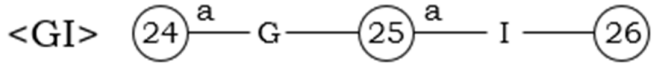


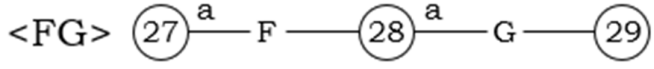


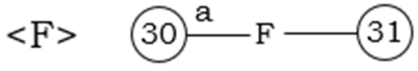












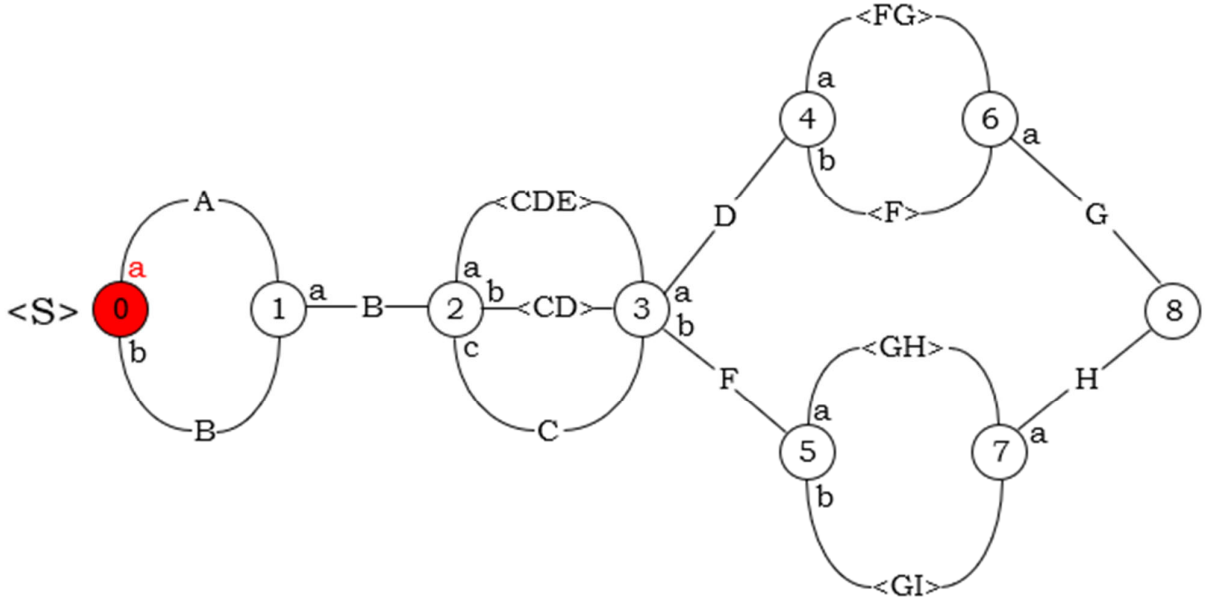
An easy representation is described at the end of this document.

Stack entries consist of:

- state we're in
- input not yet consumed, in parentheses
- arc to be taken
- set of arcs that still might be taken (after backtracking)
- parse "tree" object as it was before taking the arc.
- parse "tree" object as it is after taking the arc.

Input sentence is A B C D F G

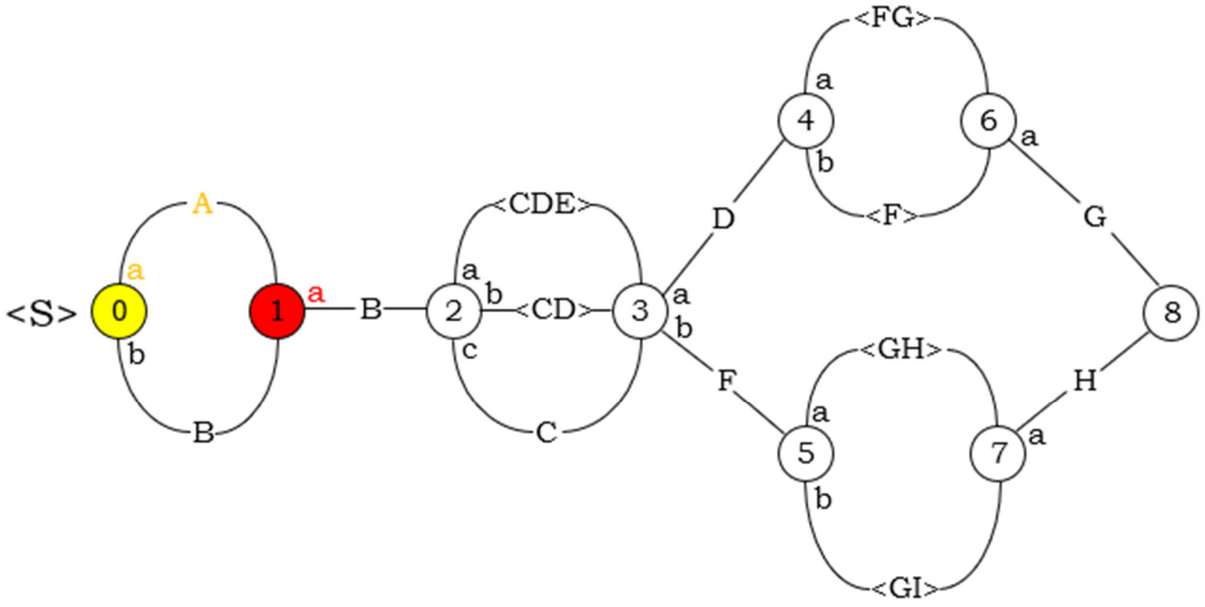
stack	metastack
empty	empty



stack	metastack
0 (ABCDFG) a {b} []	empty

stack
0 (ABCDFG) a {b} []

metastack
empty



stack
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]

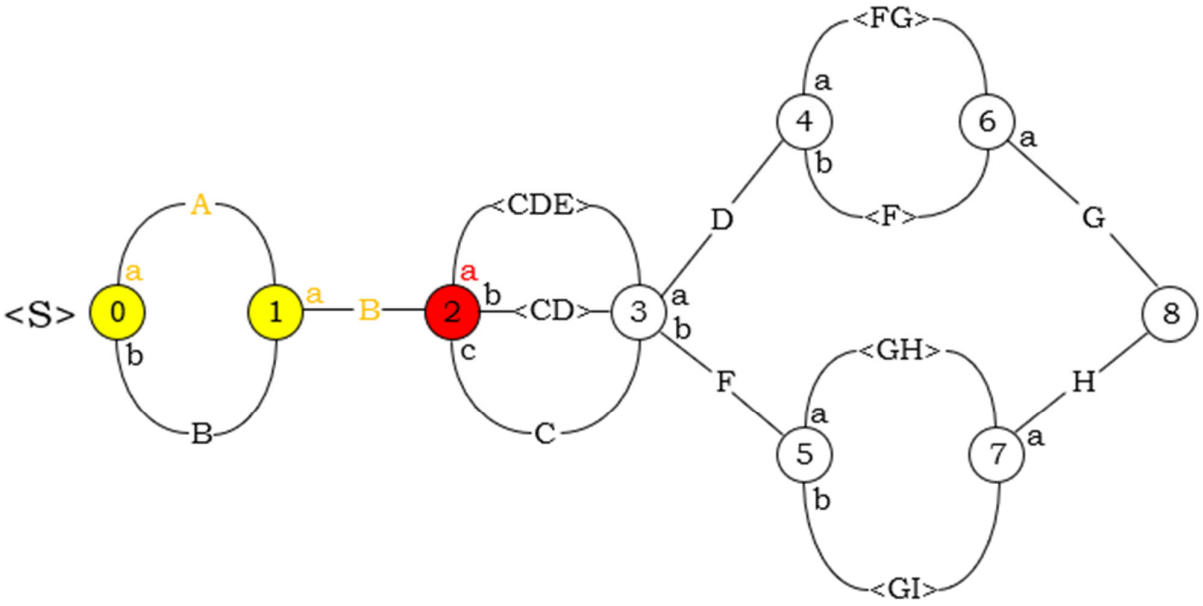
metastack
empty

stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
```

metastack

empty



stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) a {bc} [AB] []    .............
```

metastack

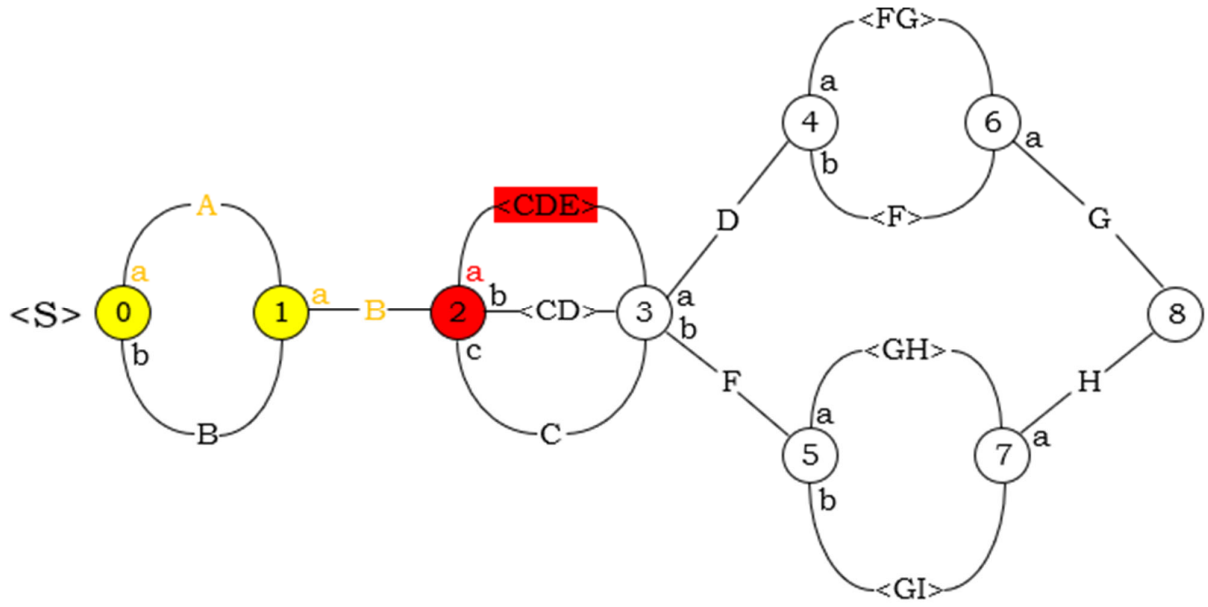
```
← call <CDE>
```

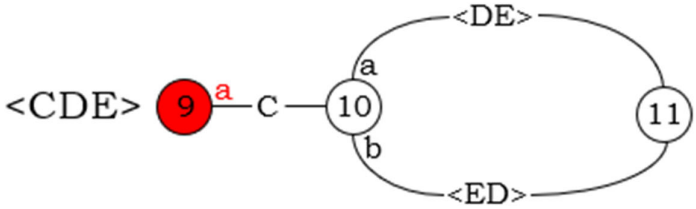
stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) a {bc} [AB] []
```

metastack

..... ← call <CDE>





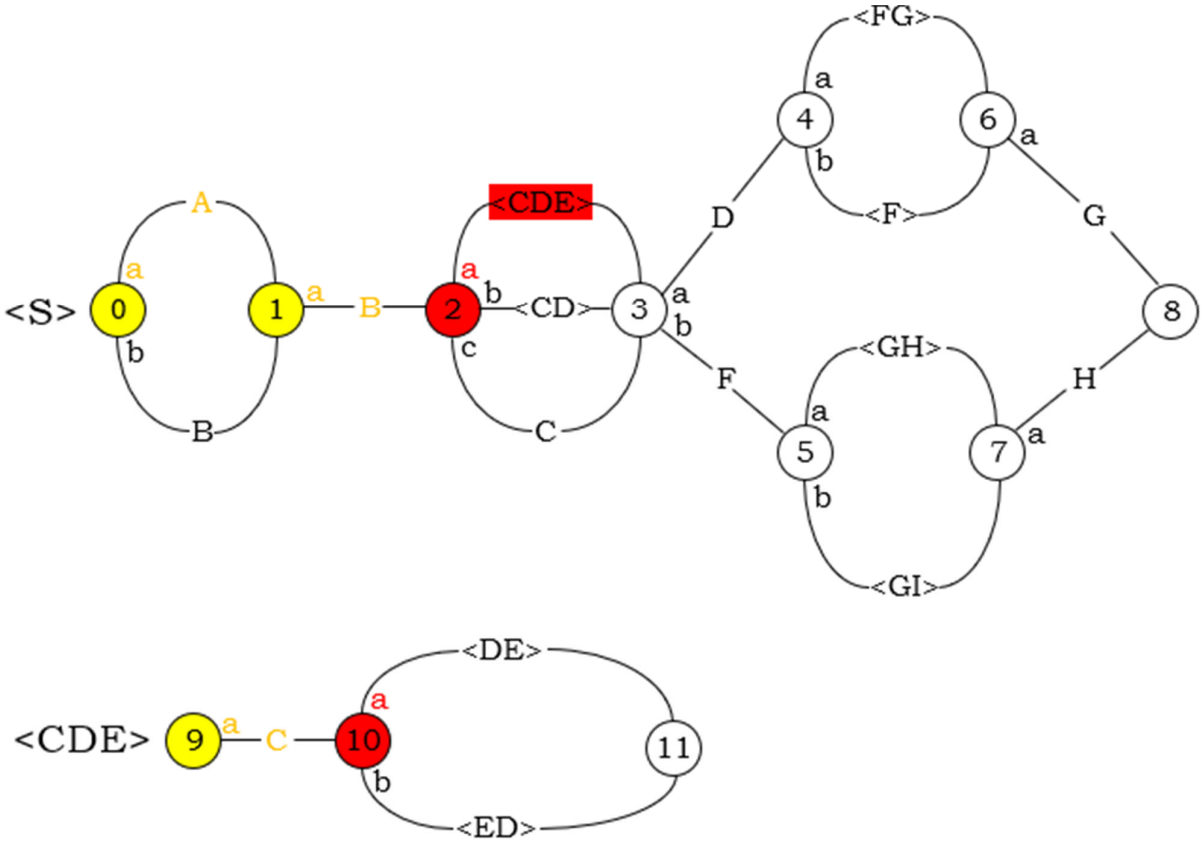
stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) a {bc} [AB] []
9 (CDFG) a {} [] [C]
```

metastack

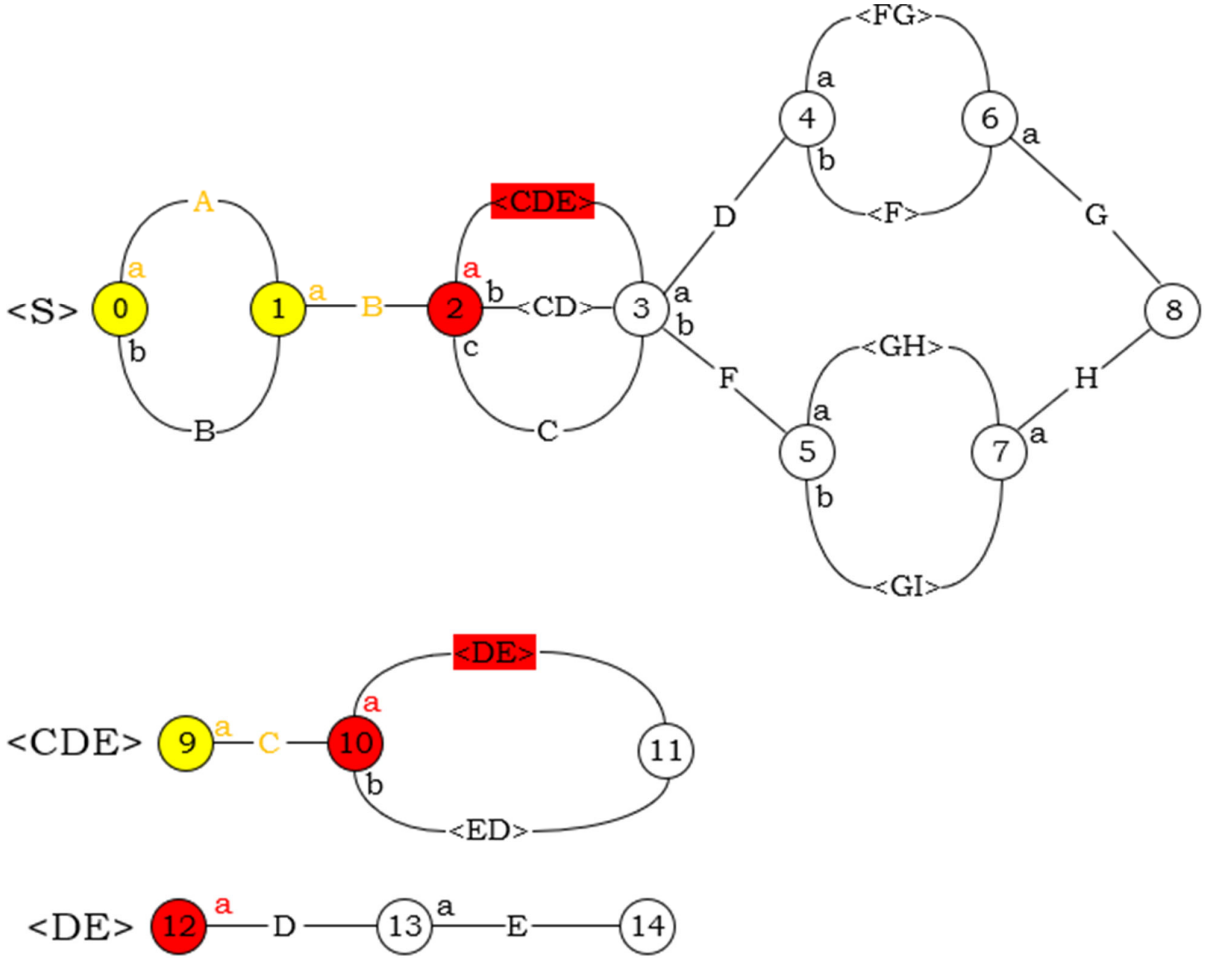
..... ← call <CDE>

stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C]		



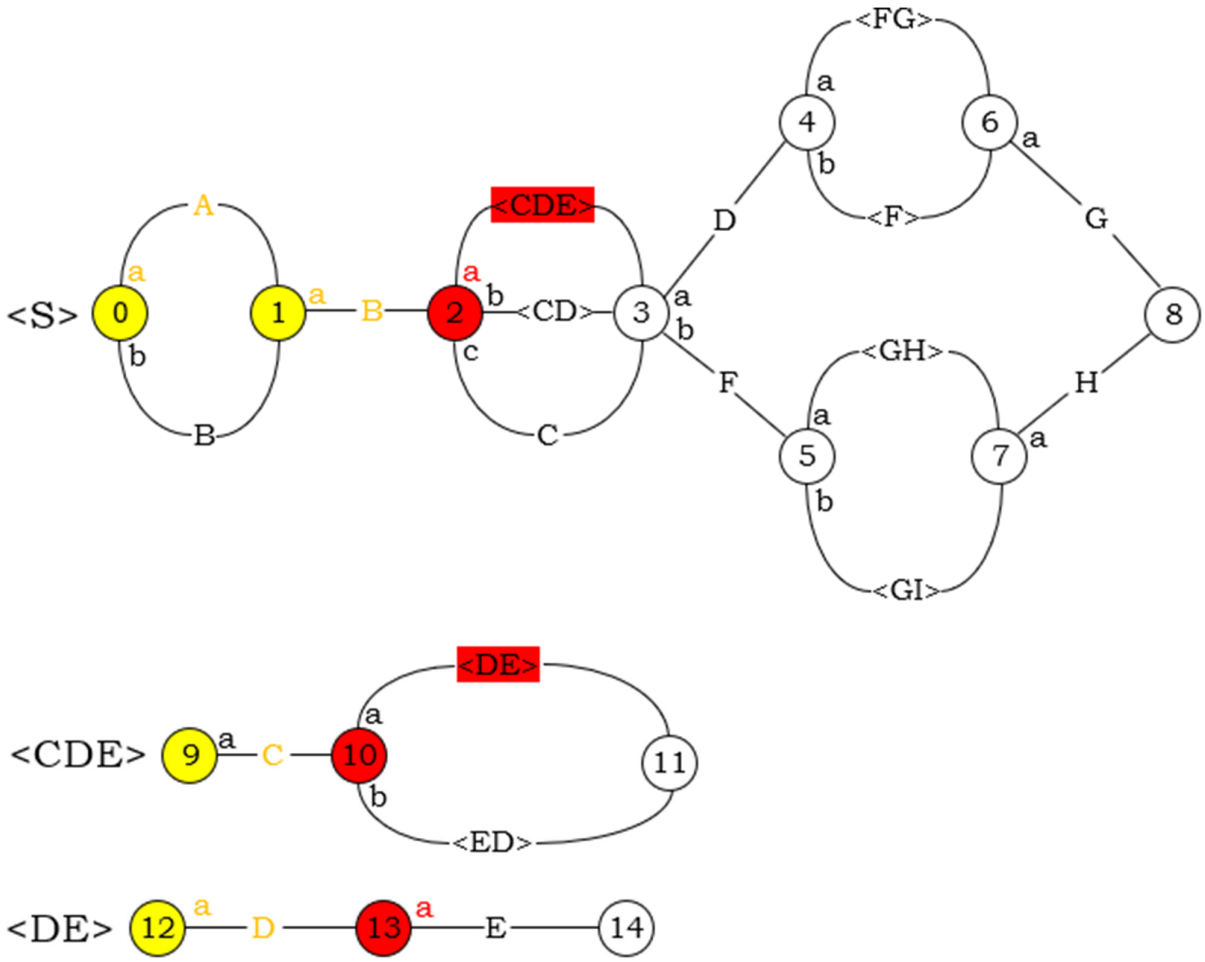
stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C]		
10 (DFG) a {b} [C] []	.....	← call <DE>

stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C]		
10 (DFG) a {b} [C] []	.....	← call <DE>



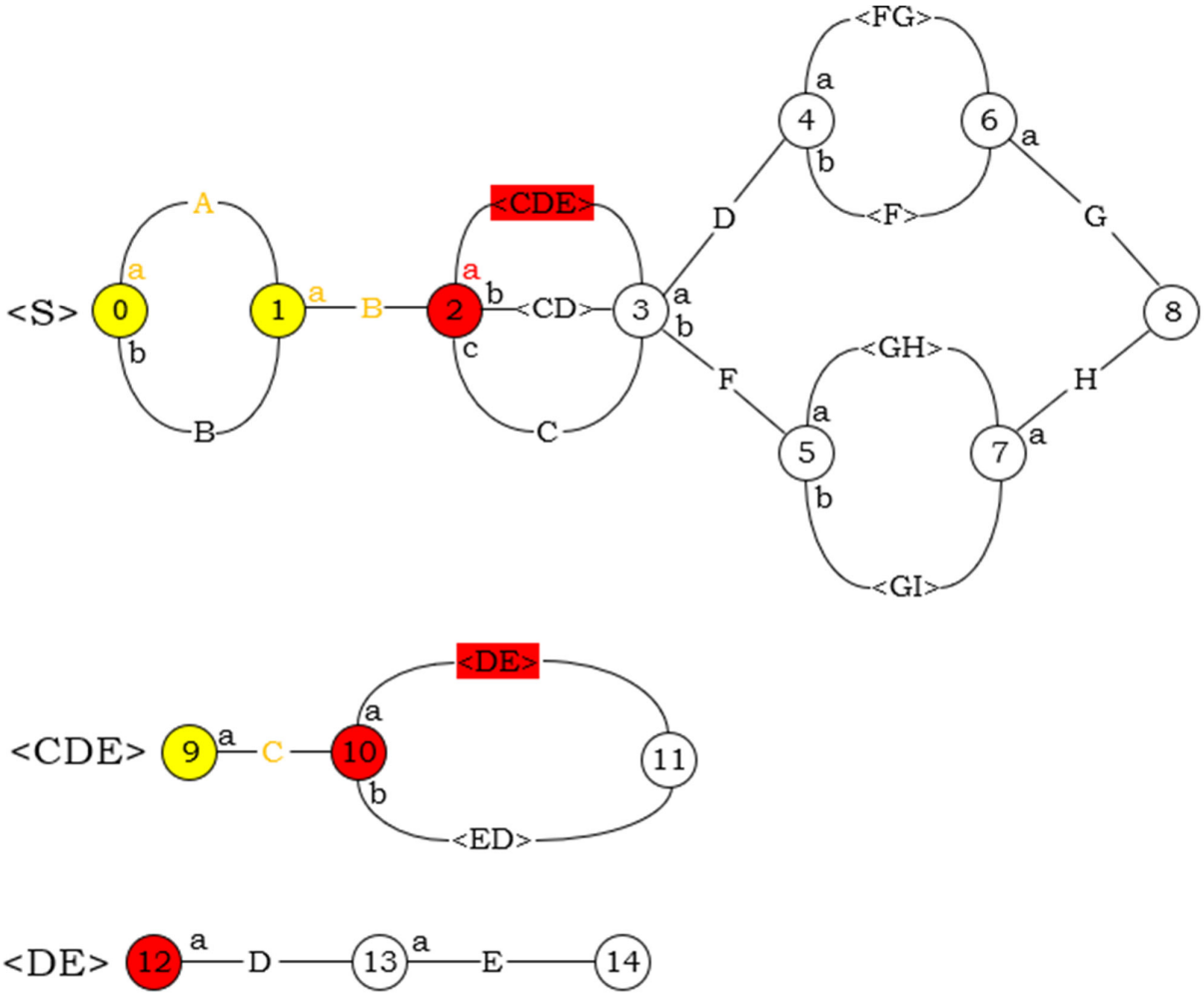
stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C]		
10 (DFG) a {b} [C] []	.....	← call <DE>
12 (FG) a {} [] [D]		

stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C]		
10 (DFG) a {b} [C] []	.....	← call <DE>
12 (FG) a {} [] [D]		



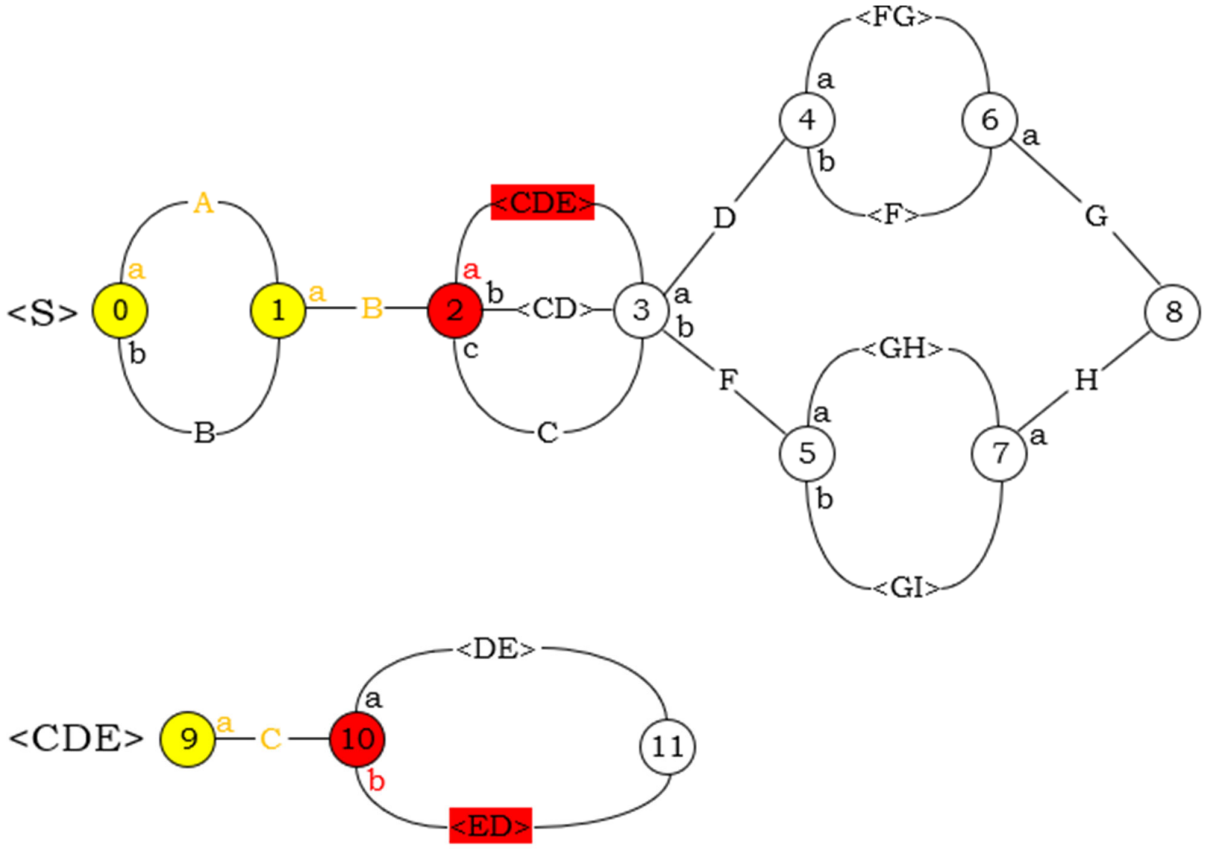
stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {} [AB] []	.....	← call <CDE>
9 (CDFG) a {bc} [] [C]		
10 (DFG) a {b} [C] []	.....	← call <DE>
12 (DFG) a {} [] [D]		
13 (FG) FAIL, E needed		

stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {} [AB] []	.....	← call <CDE>
9 (CDFG) a {bc} [] [C]		
10 (DFG) a {b} [C] []	.....	← call <DE>
12 (DFG) a {} [] [D]		
13 (FG) FAIL, E needed		



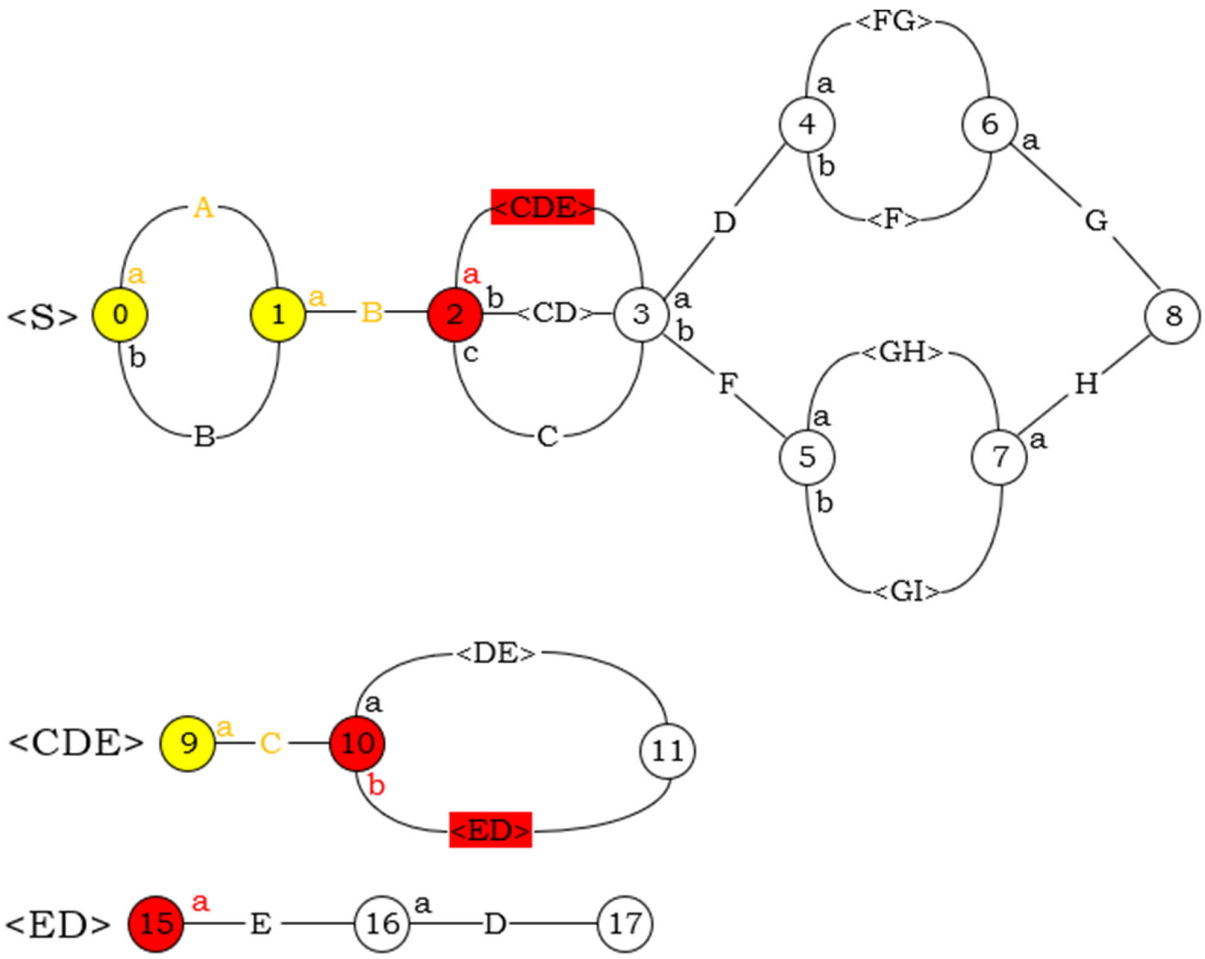
stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C]		
10 (DFG) a {b} [C] []	.....	← call <DE>
12 (DFG) a {} [] [D]	Fail, no options	

stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C]		
10 (DFG) a {b} [C] []	.....	← call <DE>
12 (DFG) a {} [] [D] Fail, no options		



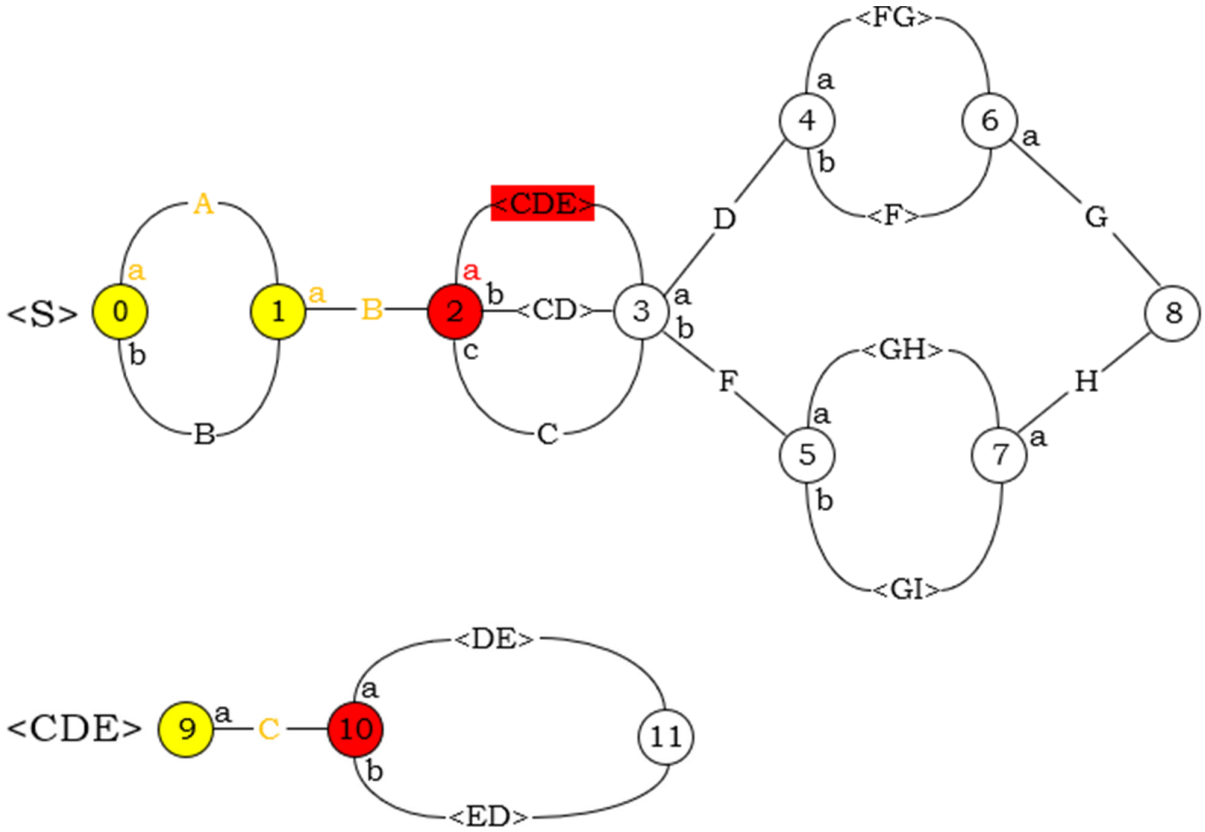
stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C]		
10 (DFG) a {b} [C] []	.....	← call <DE>
changes to		
10 (DFG) b {} [C] []	.....	← call <ED>

stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C]		
10 (DFG) b {} [C] []	.....	← call <ED>



stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C]		
10 (DFG) b {} [C] []	.....	← call <ED>
15 (DFG) Fail, E needed		

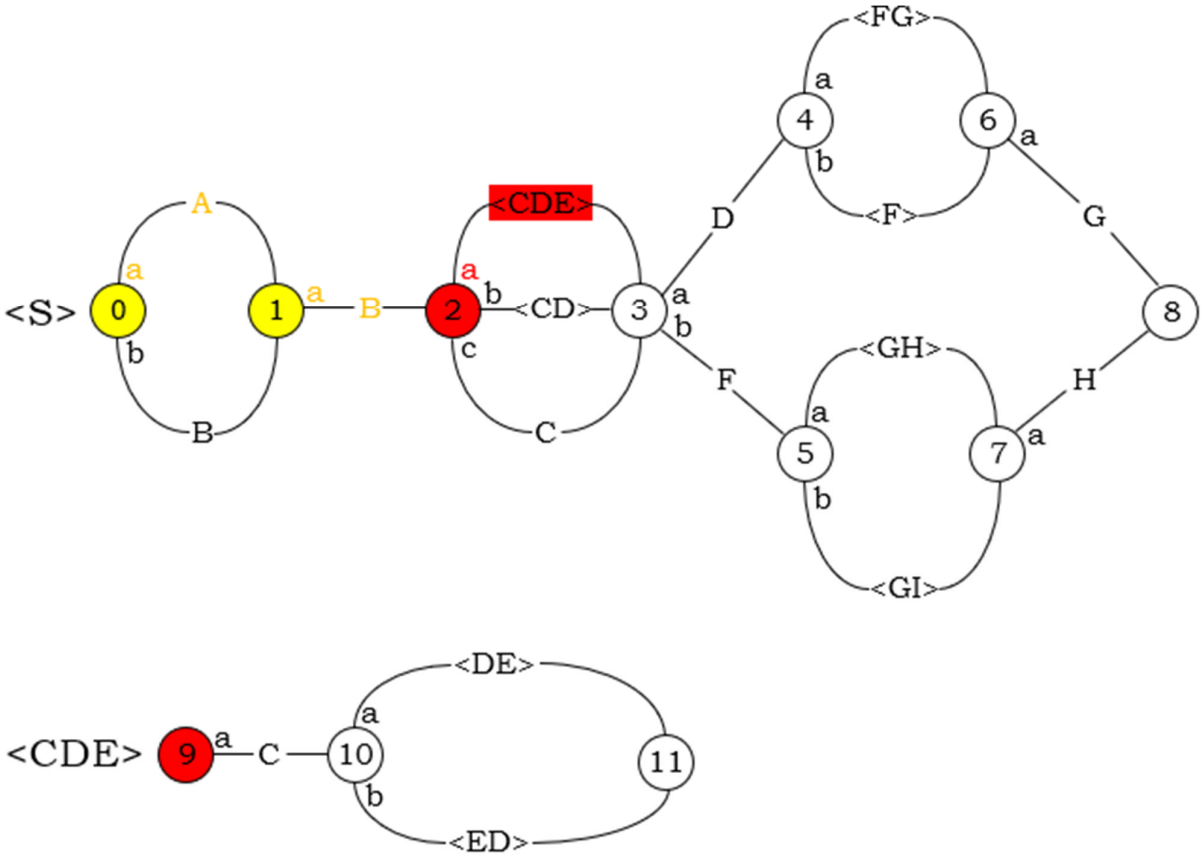
stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C]		
10 (DFG) b {} [C] []	.....	← call <ED>
15 (DFG) Fail, E needed		



*

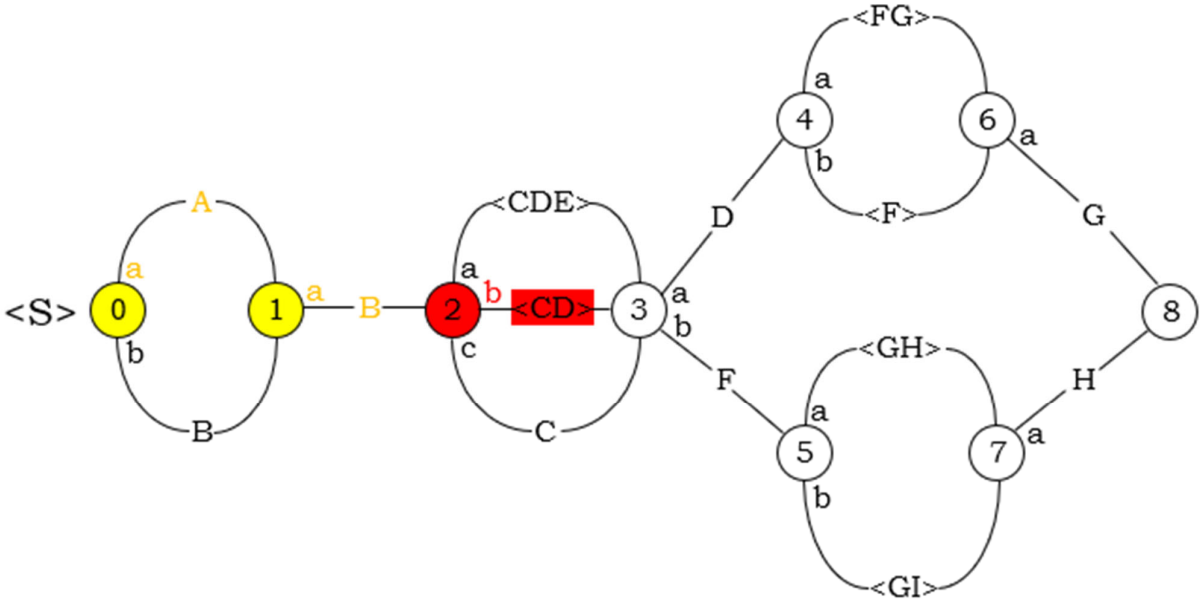
stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C]		
10 (DFG) b {} [C] [], FAIL, no options		← call <ED>

stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) a {bc} [AB] []	← call <CDE>
9 (CDFG) a {} [] [C]	
10 (DFG) b {} [C] [], FAIL, no options	← call <ED>



stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) a {bc} [AB] []	← call <CDE>
9 (CDFG) a {} [] [C] FAIL, no options	

stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
9 (CDFG) a {} [] [C] FAIL, no options		



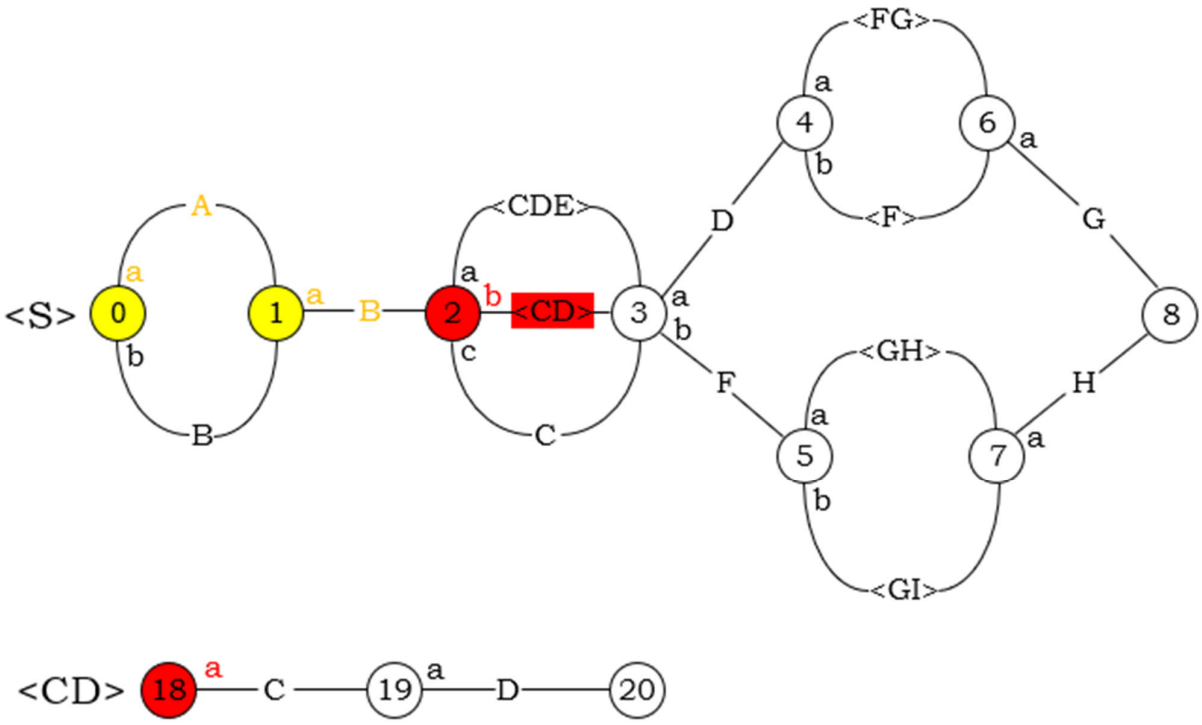
stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) a {bc} [AB] []	.....	← call <CDE>
changes to		
2 (CDFG) b {c} [AB] []	.....	← call <CD>

stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []
```

metastack

..... ← call <CD>



stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []
18 (CDFG) a {} [] [C]
```

metastack

..... ← call <CD>

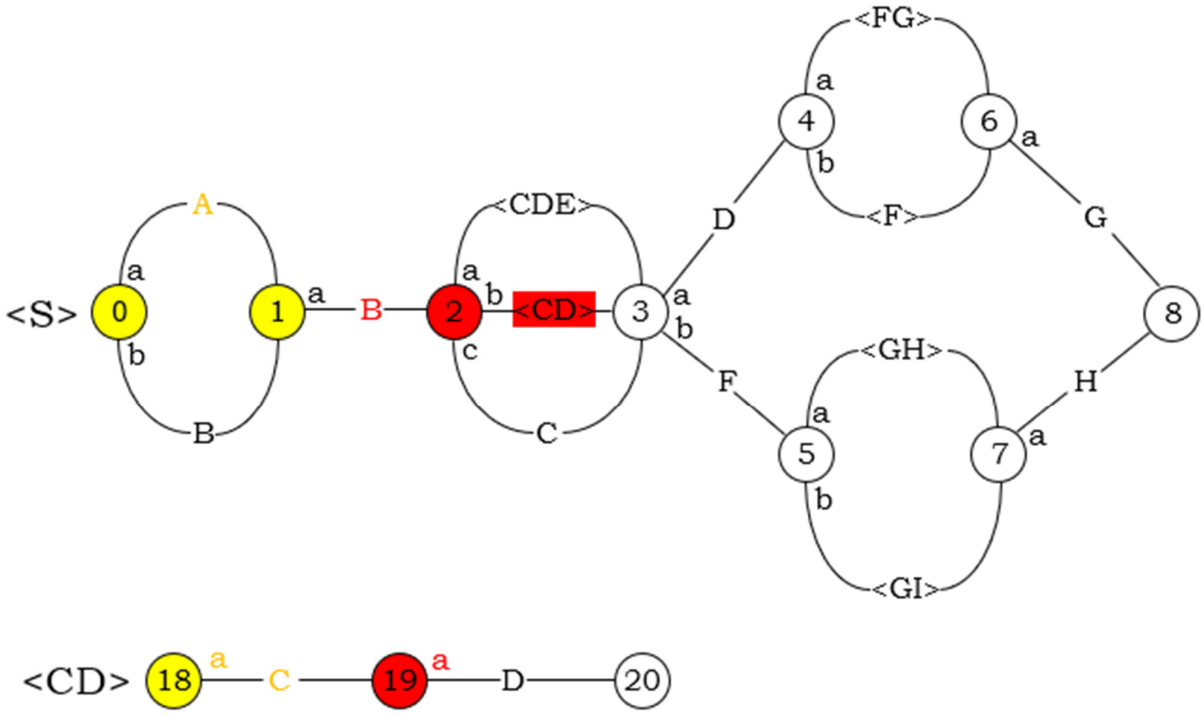
stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []
18 (CDFG) a {} [] [C]
```

.....

metastack

← call <CD>



stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]
```

.....

metastack

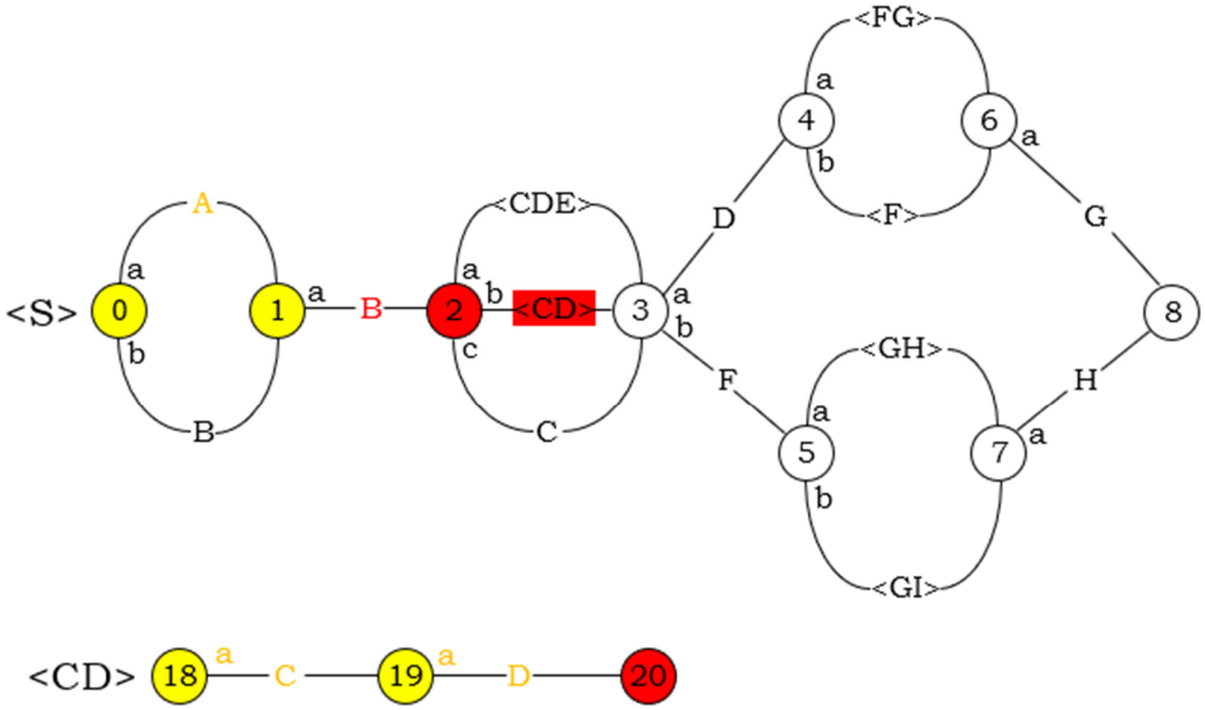
← call <CD>

stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]
```

..... ← call <CD>

metastack



stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]
20 (FG) Success
```

..... ← call <CD>

metastack

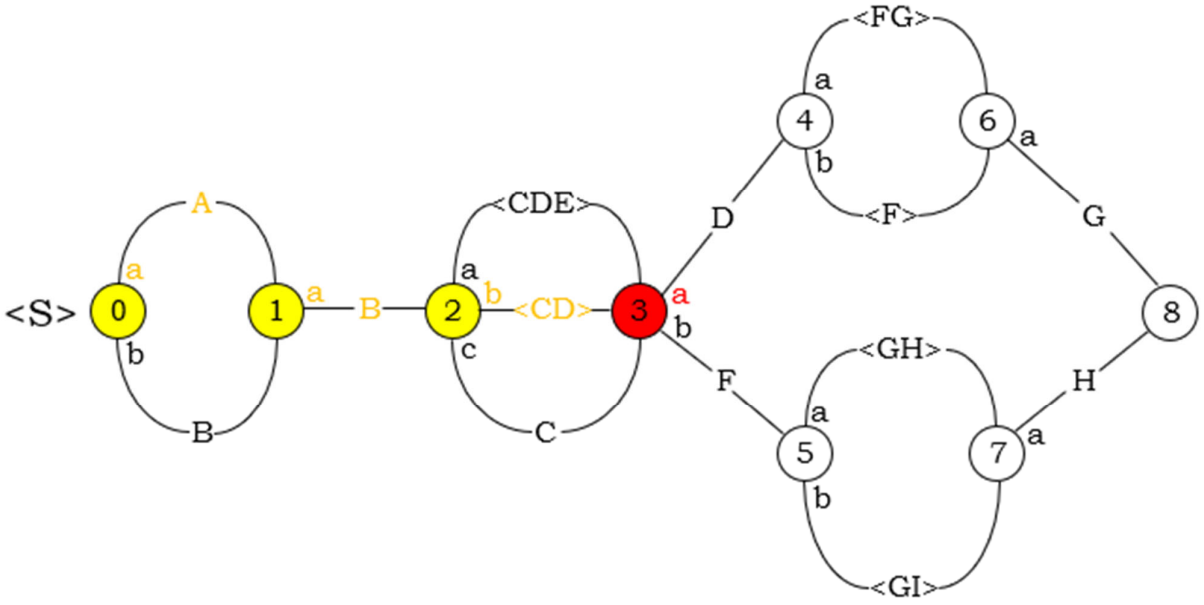
stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]
20 (FG) Success
```

metastack

```
.....
```

← call <CD>



stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]
20 (FG) Success
3 (FG) a {b} [AB[CD]] FAIL, D needed
```

metastack

```
.....
```

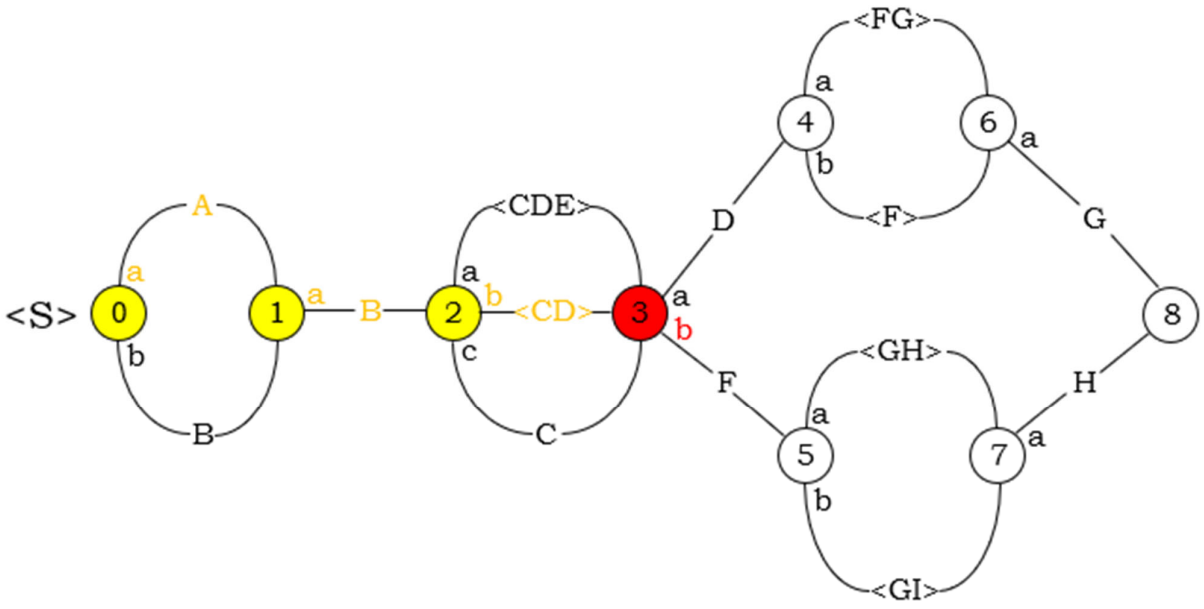
← call <CD>

stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]
20 (FG) Success
3 (FG) a {b} [AB[CD]] FAIL, D needed
```

metastack

```
..... ← call <CD>
```



stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]
20 (FG) Success
3 (FG) a {b} [AB[CD]]
```

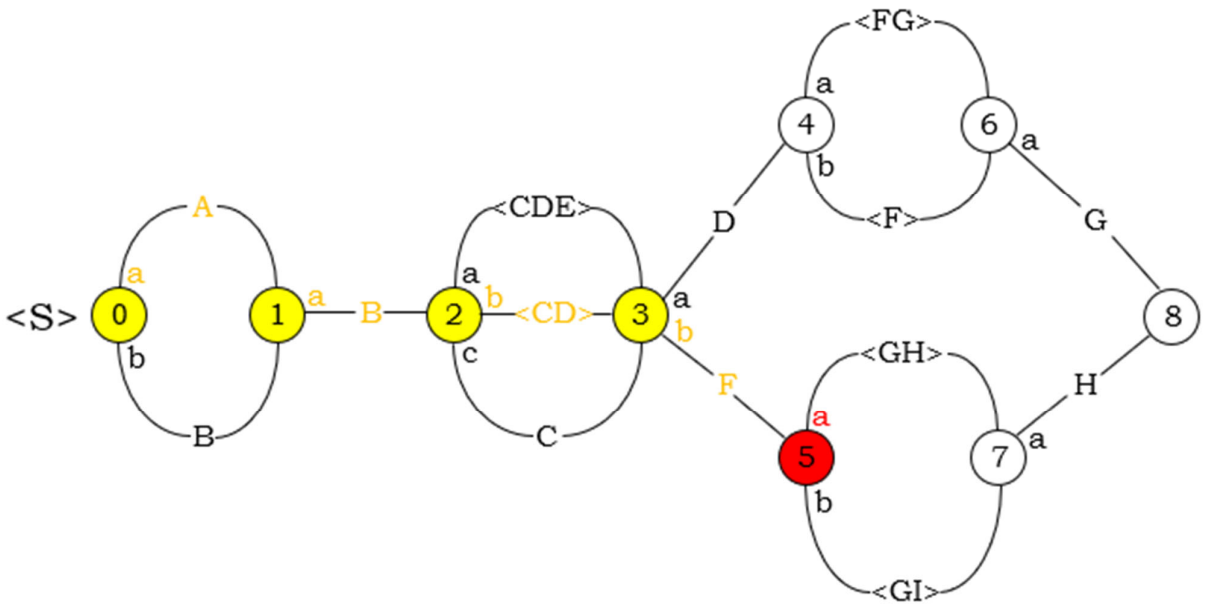
changes to

```
3 (FG) b {} [AB[CD]]
```

metastack

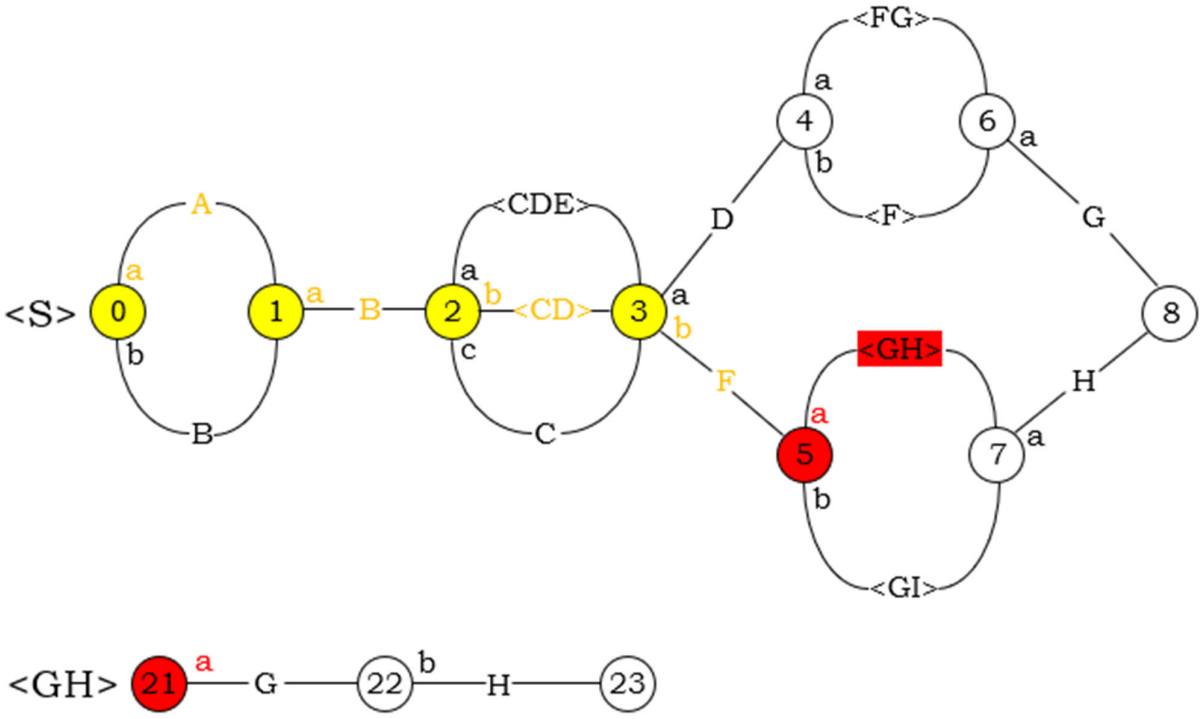
```
..... ← call <CD>
```

stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) b {c} [AB] []	.....	← call <CD>
18 (CDFG) a {} [] [C]		
19 (DFG) a {} [C] [CD]		
20 (FG) Success		
3 (FG) b {} [AB[CD]]		



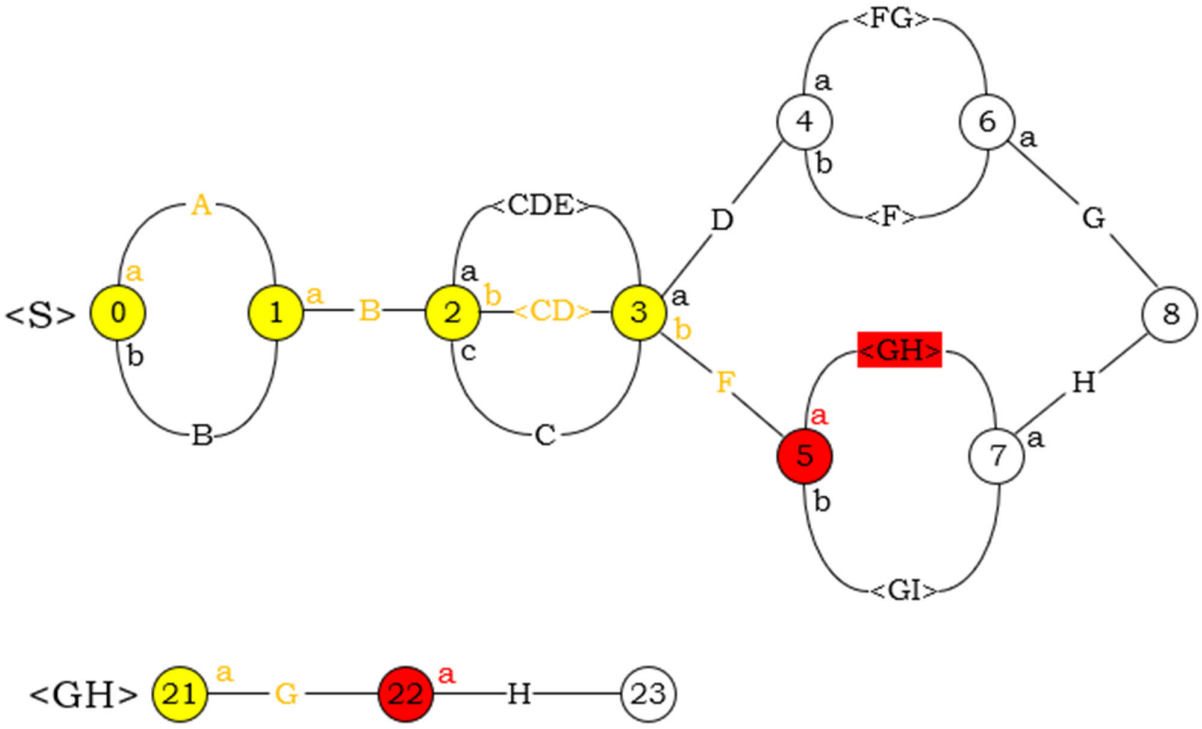
stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) b {c} [AB] []	.....	← call <CD>
18 (CDFG) a {} [] [C]		
19 (DFG) a {} [C] [CD]		
20 (FG) Success		
3 (FG) b {} [AB[CD]] [AB[CD]F]		
5 (G) a {b} [AB[CD]F] []	.....	← call <GH>

stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) a {b} [AB[CD]F] []	← call <GH>



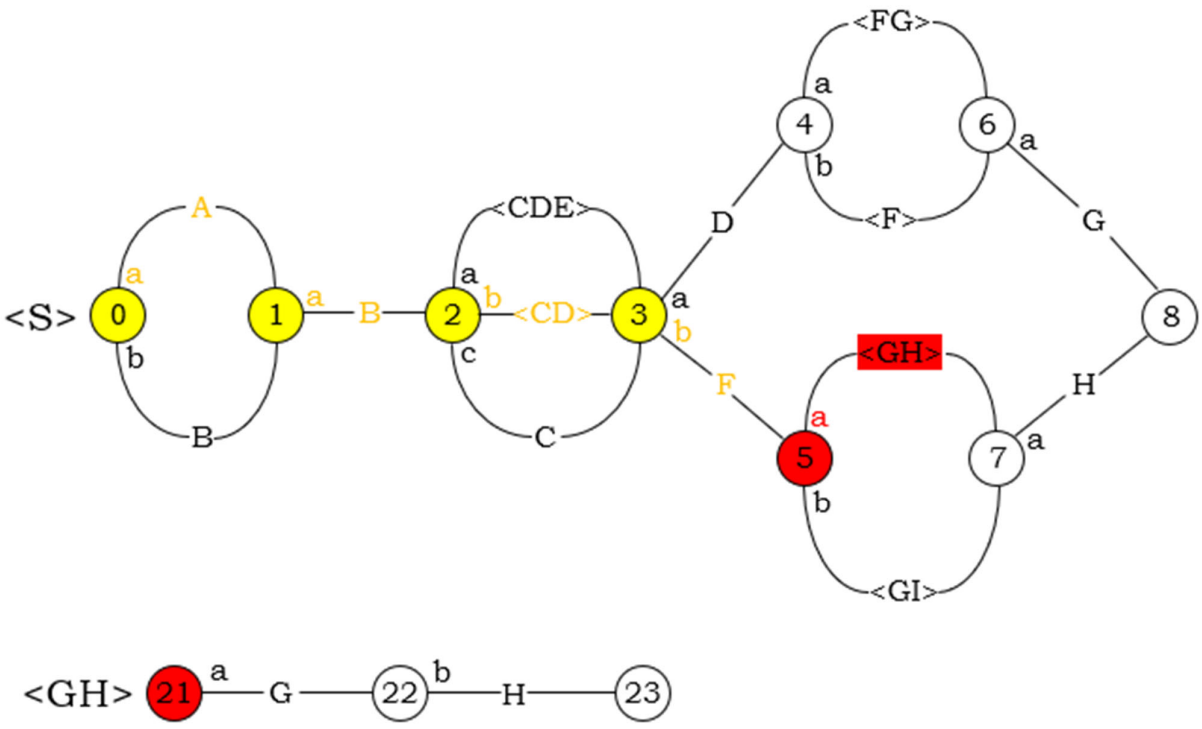
stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) a {b} [AB[CD]F] []	← call <GH>
21 (G) a {} [] [G]	

stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) a {b} [AB[CD]F] []	← call <GH>
21 (G) a {} [] [G]	



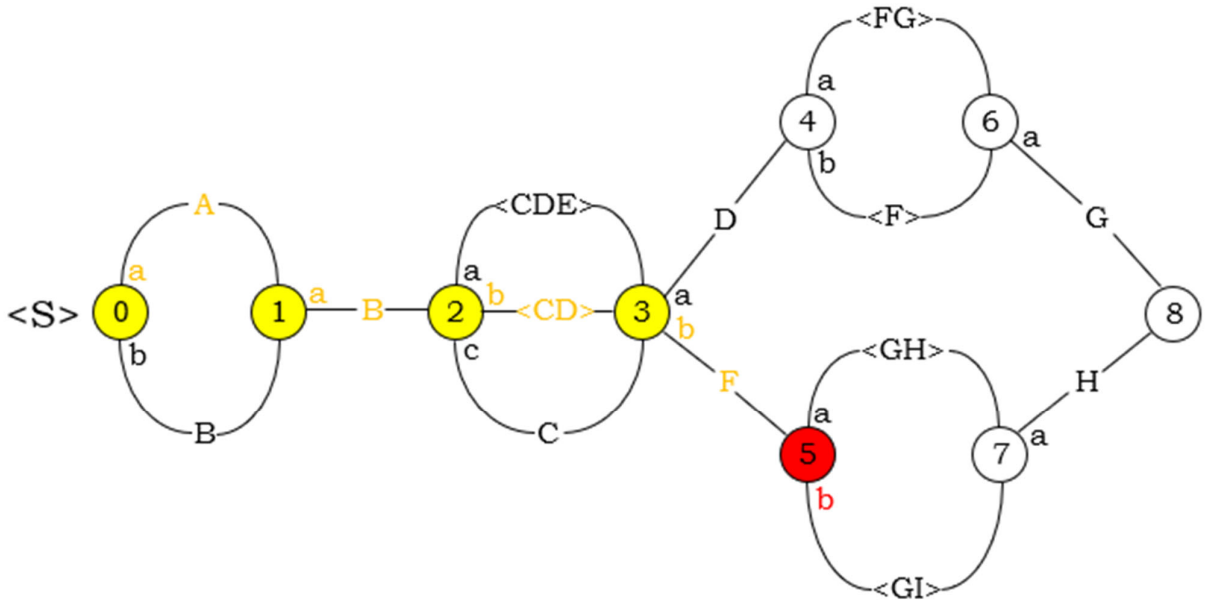
stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) a {b} [AB[CD]F] []	← call <GH>
21 (G) a {} [] [G]	
22 () FAIL, H needed	

stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) a {b} [AB[CD]F] []	← call <GH>
21 (G) a {} [] [G]	
22 () FAIL, H needed	



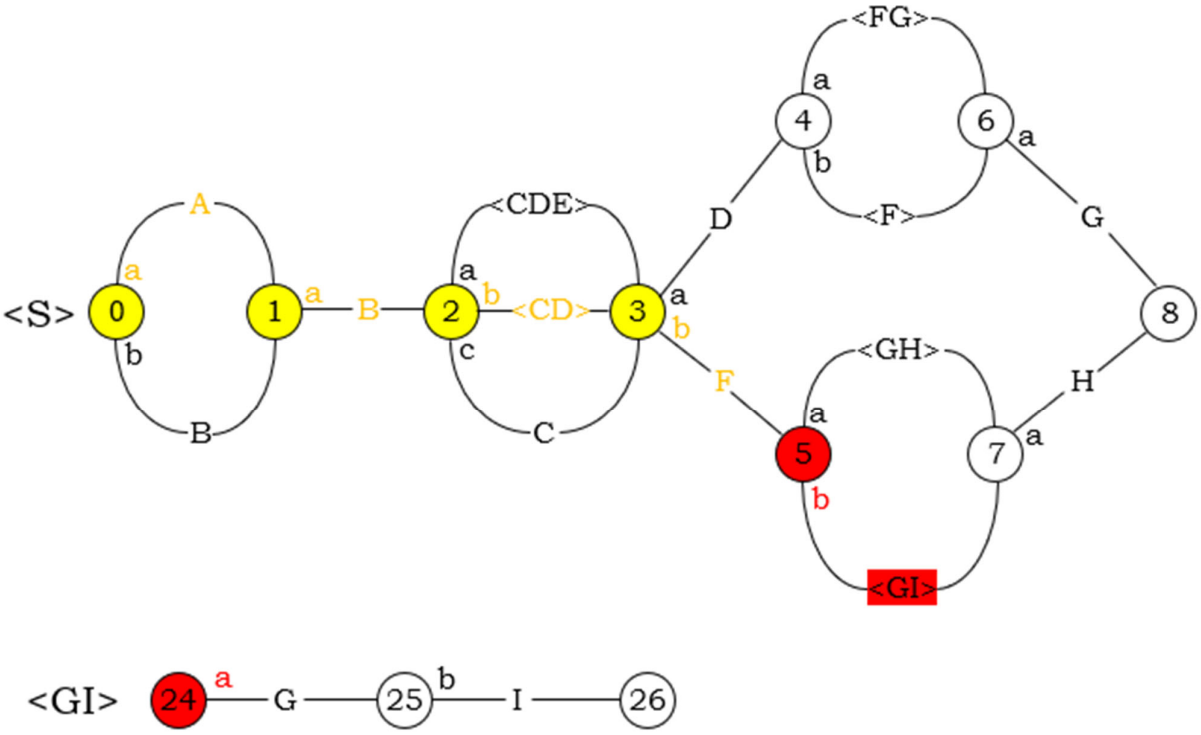
stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) a {b} [AB[CD]F] []	← call <GH>
21 (G) a {} [] [G] Fail, no options	

stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) a {b} [AB[CD]F] []	← call <GH>
21 (G) a {} [] [G] Fail, no options	



stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) a {b} [AB[CD]F] []	← call <GH>
changes to	
5 (G) b {} [AB[CD]F] []	
	← call <GI>

stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) b {} [AB[CD]F] []	← call <GI>



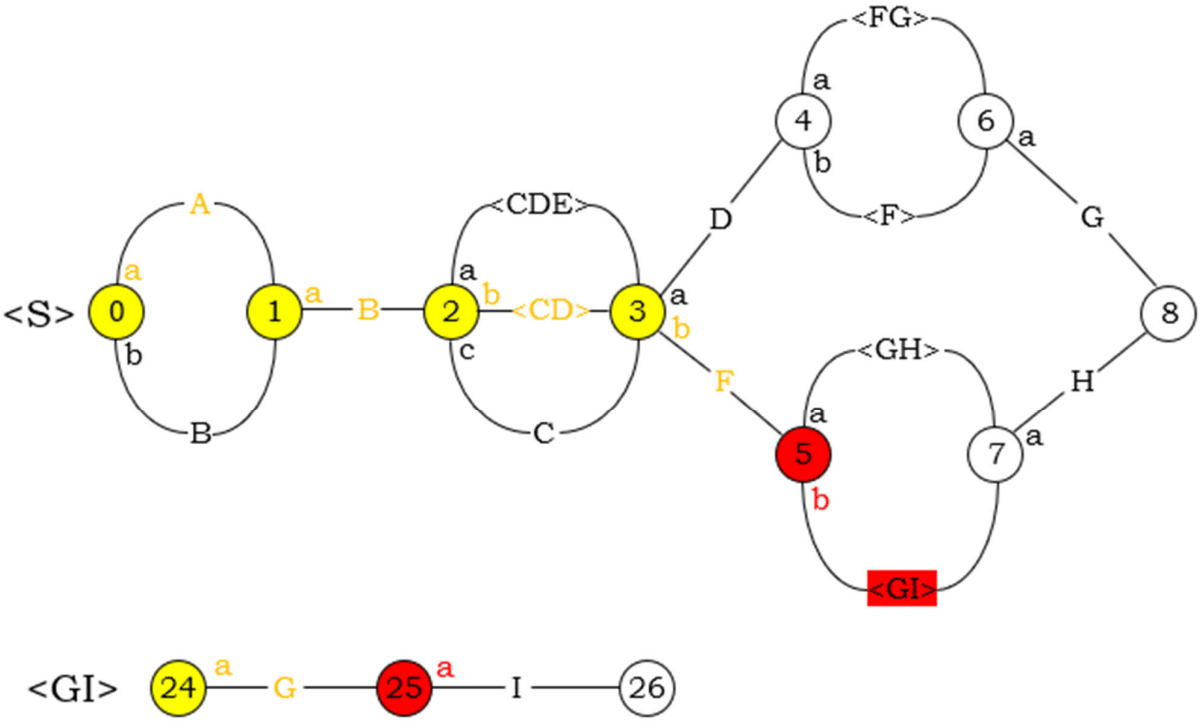
stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) b {} [AB[CD]F] []	← call <GI>
24 (G) a {} [] [G]	

stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]
20 (FG) Success
3 (FG) b {} [AB[CD]] [AB[CD]F]
5 (G) b {} [AB[CD]F] []
24 (G) a {} [] [G]
```

metastack

```
..... ← call <CD>
..... ← call <GI>
```



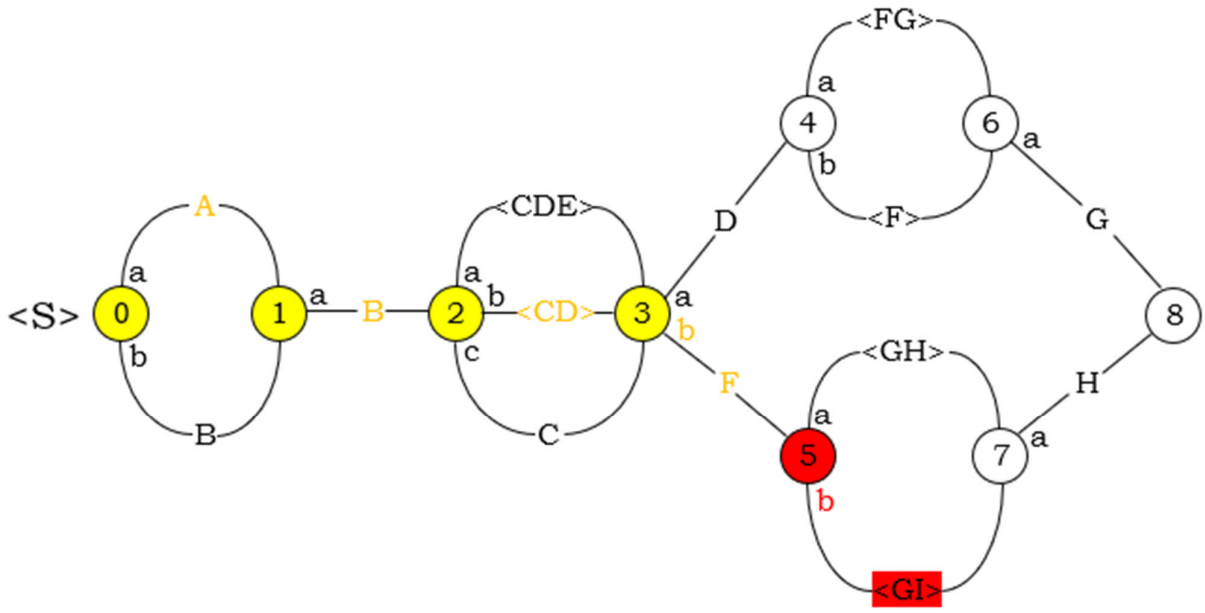
stack

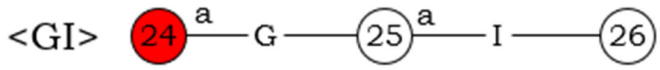
```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]
20 (FG) Success
3 (FG) b {} [AB[CD]] [AB[CD]F]
5 (G) b {} [AB[CD]F] []
24 (G) a {} [] [G]
25 () FAIL, I needed
```

metastack

```
..... ← call <CD>
..... ← call <GI>
```

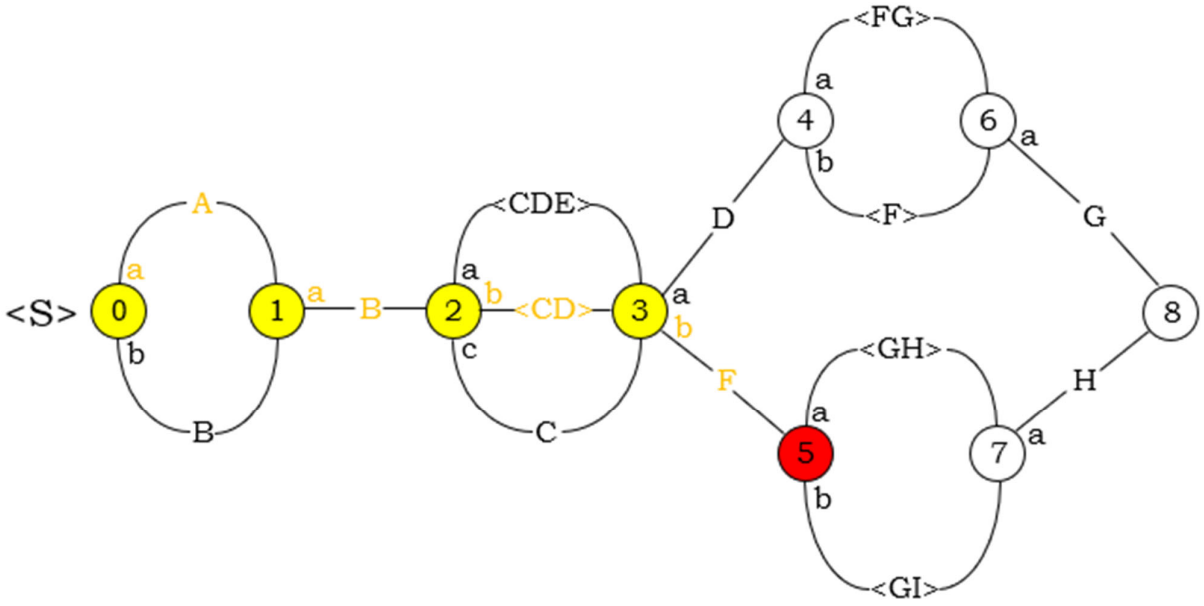
stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) b {} [AB[CD]F] []	← call <GI>
24 (G) a {} [] [G]	
25 () FAIL, I needed	





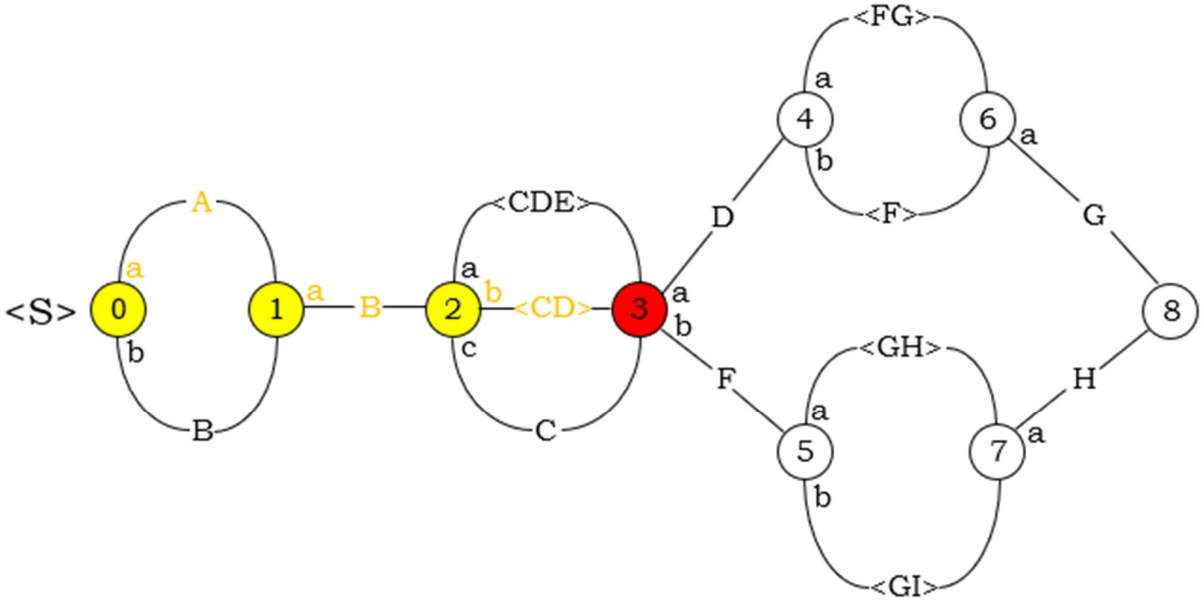
stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) b {} [AB[CD]F] []	← call <GI>
24 (G) a {} [] [G] Fail, no options	

stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) b {} [AB[CD]F] []	← call <GI>
24 (G) a {} [] [G] Fail, no options	



stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) b {} [AB[CD]F] [] FAIL, no options	← call <GI>

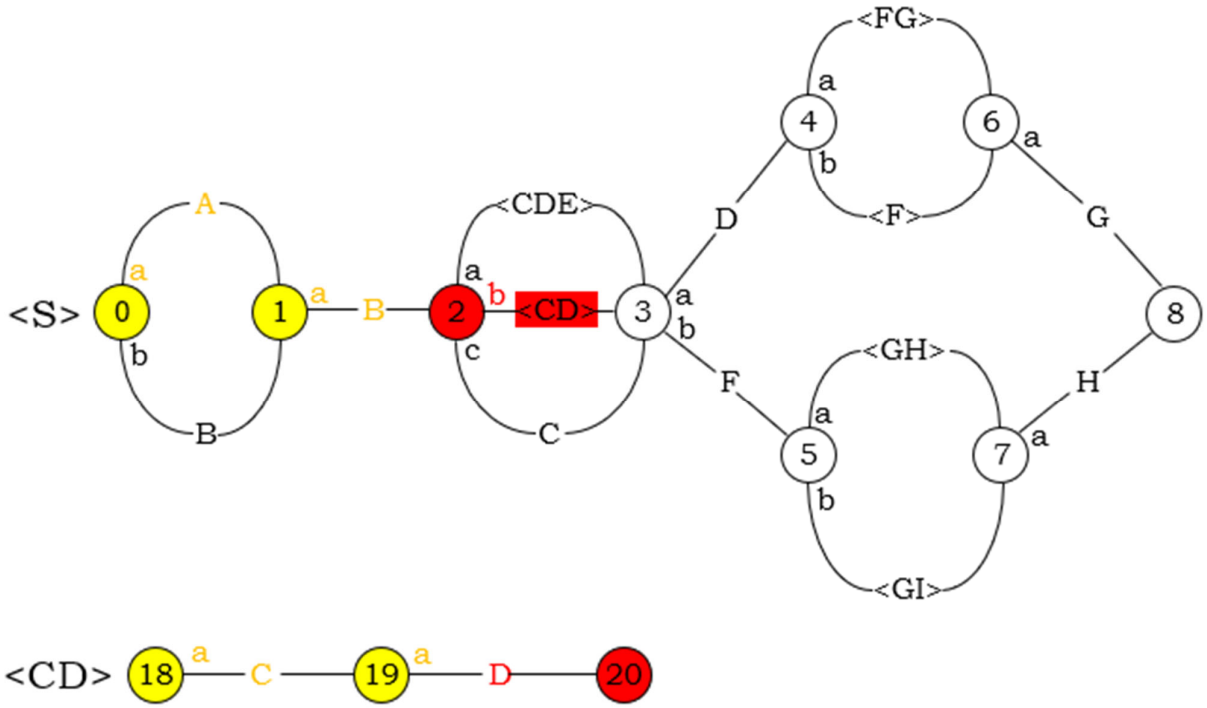
stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F]	
5 (G) b {} [AB[CD]F] [] FAIL, no options	← call <GI>



stack	metastack
0 (ABCDFG) a {b} [] [A]	
1 (BCDFG) a {} [A] [AB]	
2 (CDFG) b {c} [AB] []	← call <CD>
18 (CDFG) a {} [] [C]	
19 (DFG) a {} [C] [CD]	
20 (FG) Success	
3 (FG) b {} [AB[CD]] [AB[CD]F] FAIL, no options	

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []          .....          ← call <CD>
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]
20 (FG) Success
3 (FG) b {} [AB[CD]] [AB[CD]F] FAIL, no options
```

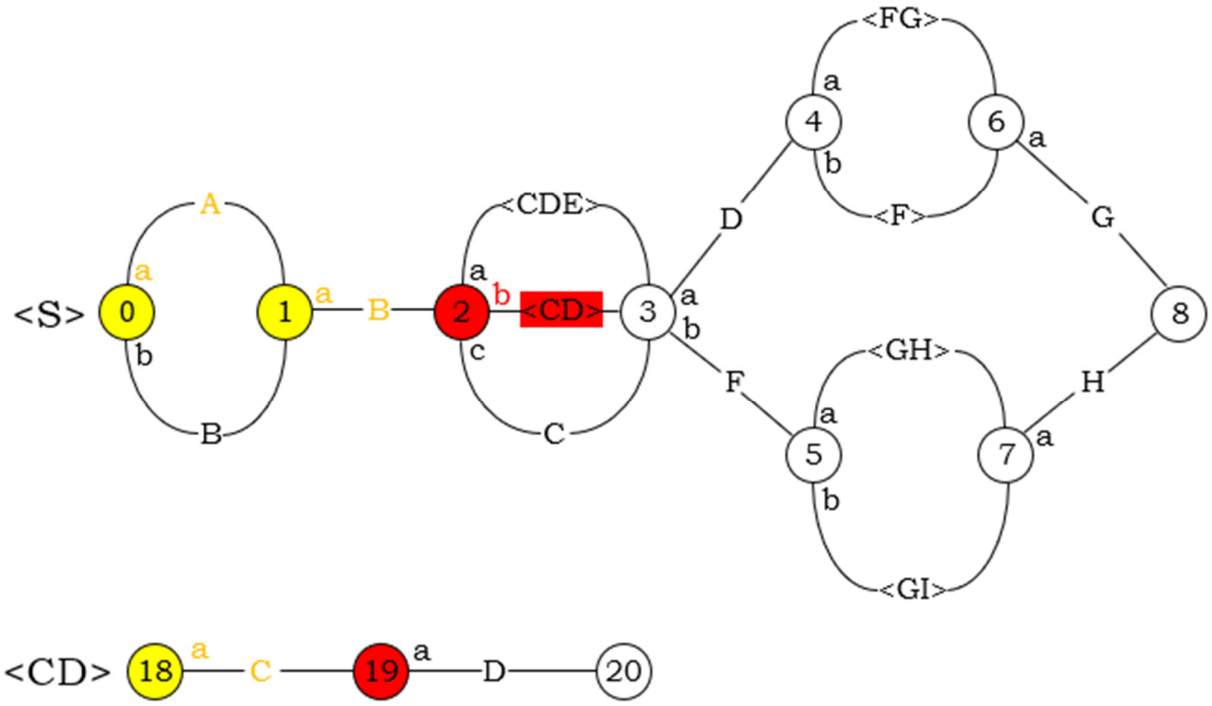


stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []          .....          ← call <CD>
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]
20 (FG) Success XXX failure continues
```

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []          .....          ← call <CD>
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]
20 (FG)  Success XXX failure continues
```

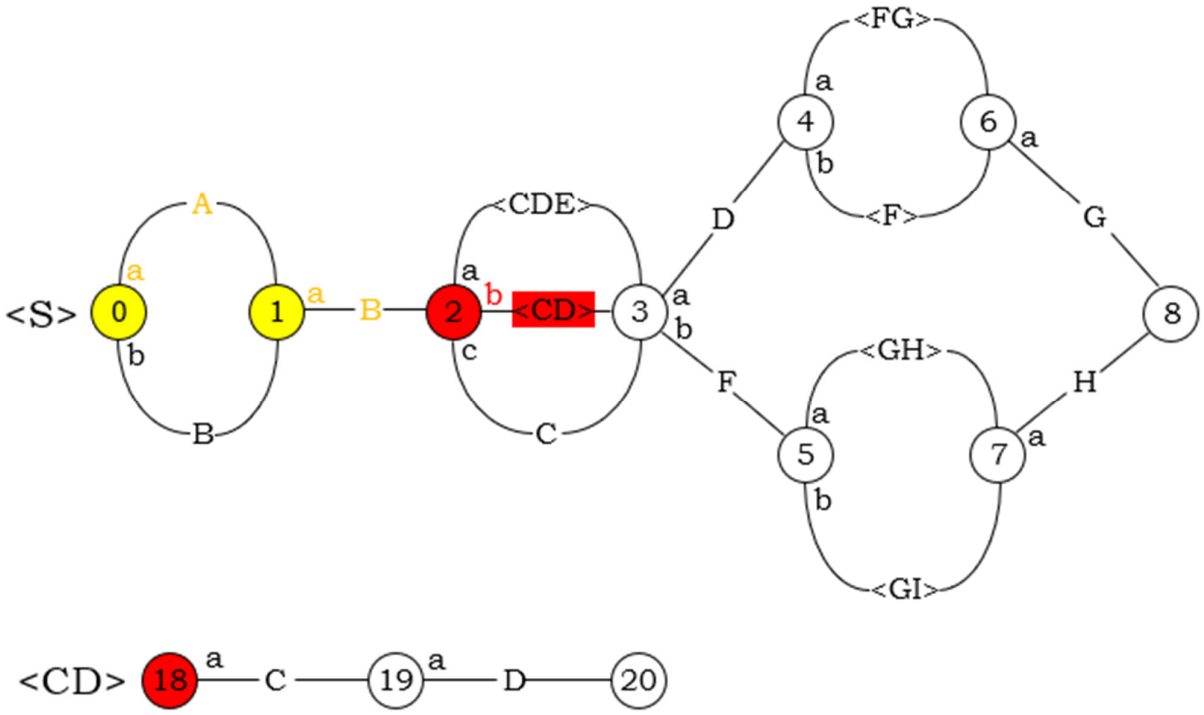


stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []          .....          ← call <CD>
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]  FAIL no options
```

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []          .....          ← call <CD>
18 (CDFG) a {} [] [C]
19 (DFG) a {} [C] [CD]  FAIL no options
```

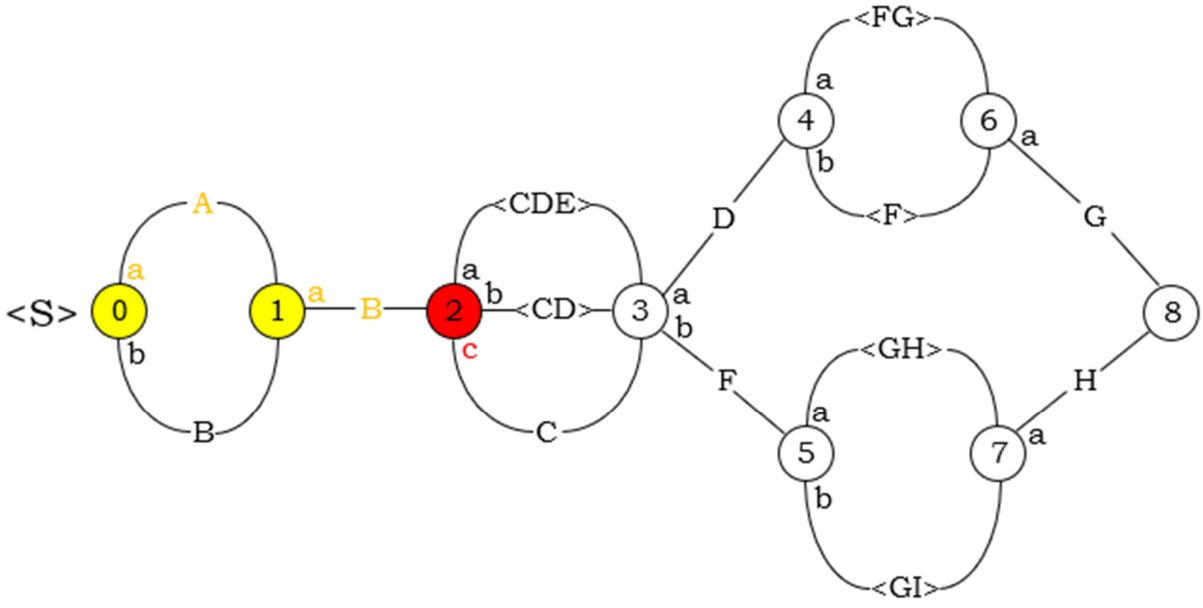


stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []          .....          ← call <CD>
18 (CDFG) a {} [] [C]  FAIL no options
```

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []          .....          ← call <CD>
18 (CDFG) a {} [] [C] FAIL no options
```



stack metastack

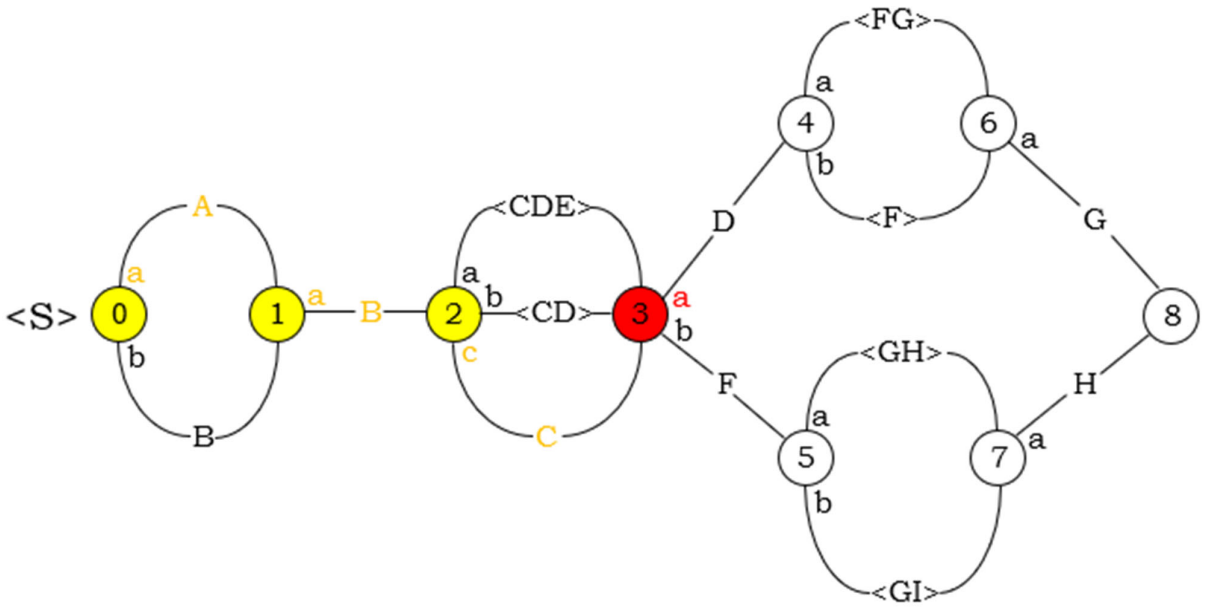
```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) b {c} [AB] []          .....          ← call <CD>
```

changes to

```
2 (CDFG) c {} [AB] [ABC]
```

stack
```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
```

metastack
empty



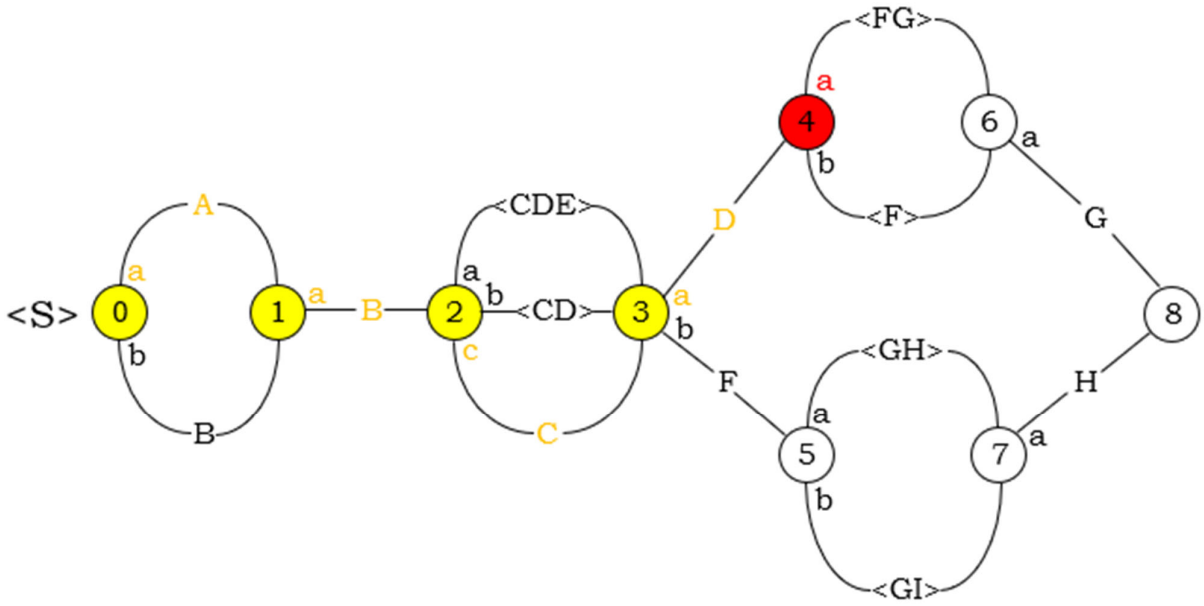
stack
```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
```

metastack
empty

stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
```

metastack
empty



stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []
```

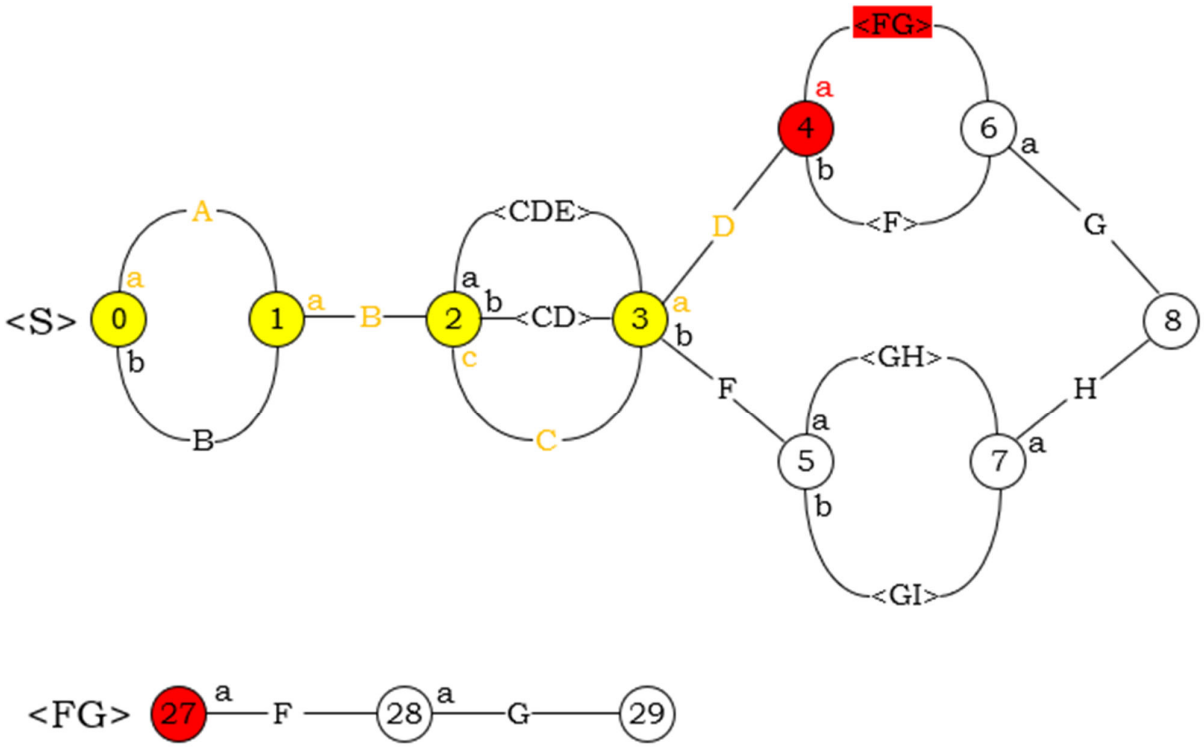
.....

metastack

← call <FG>

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []          .............   ← call <FG>
```



stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []          .............   ← call <FG>
27 (FG) a {} [] [F]
```

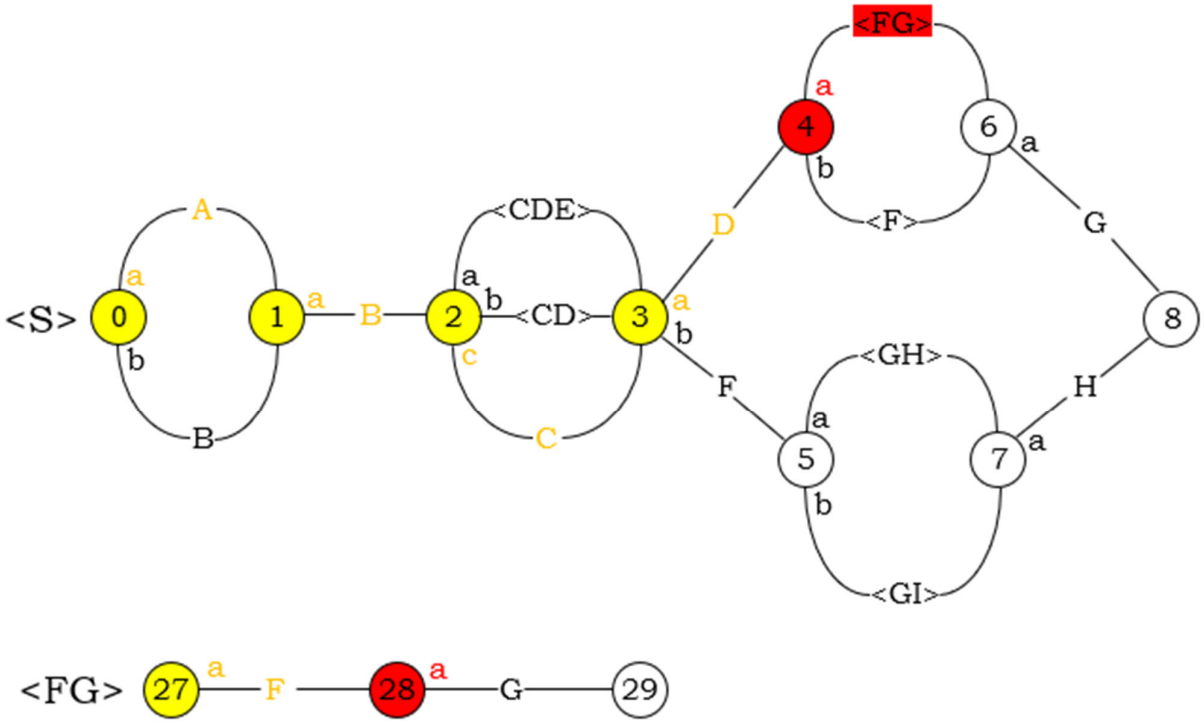
stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []
27 (FG) a {} [] [F]
```

metastack

```
.....
```

← call <FG>



stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []
27 (FG) a {} [] [F]
28 (G) a {} [F] [FG]
```

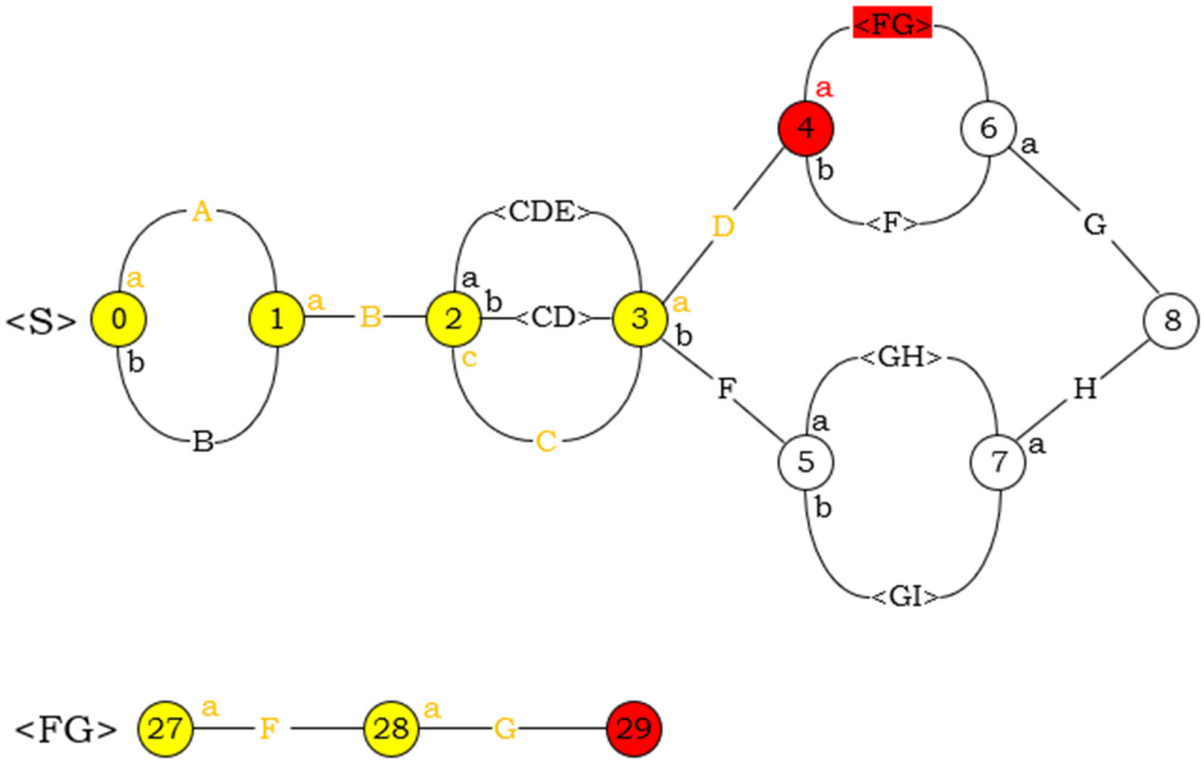
metastack

```
.....
```

← call <FG>

stack metastack

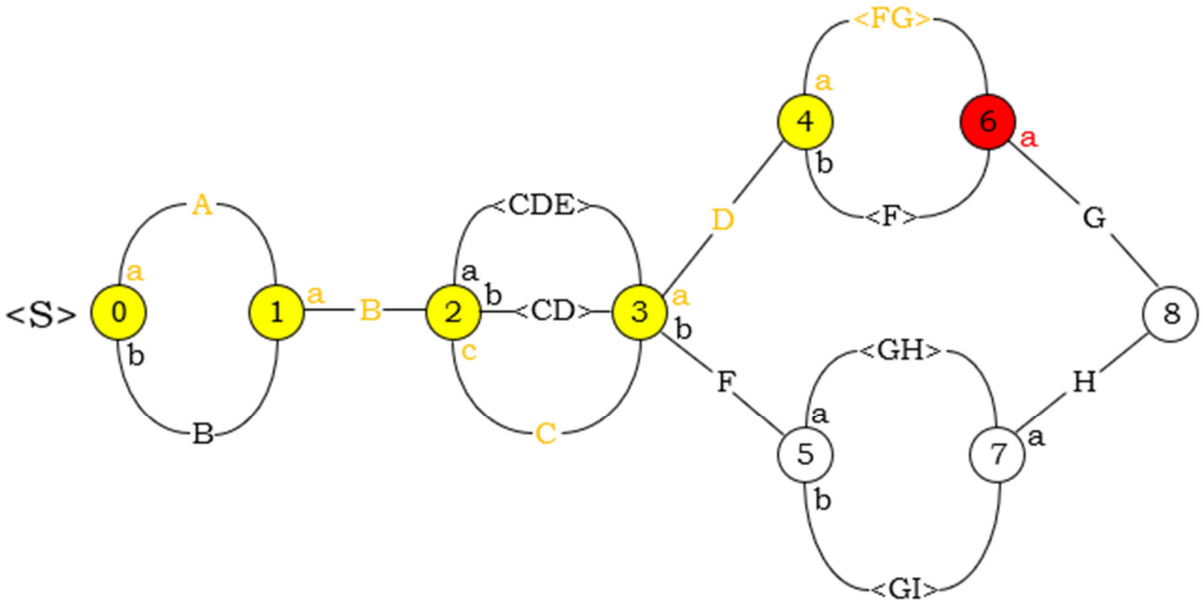
```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []          .............   ← call <FG>
27 (FG) a {} [] [F]
28 (G) a {} [F] [FG]
```



stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []          .............   ← call <FG>
27 (FG) a {} [] [F]
28 (G) a {} [F] [FG]
29 () Success
```

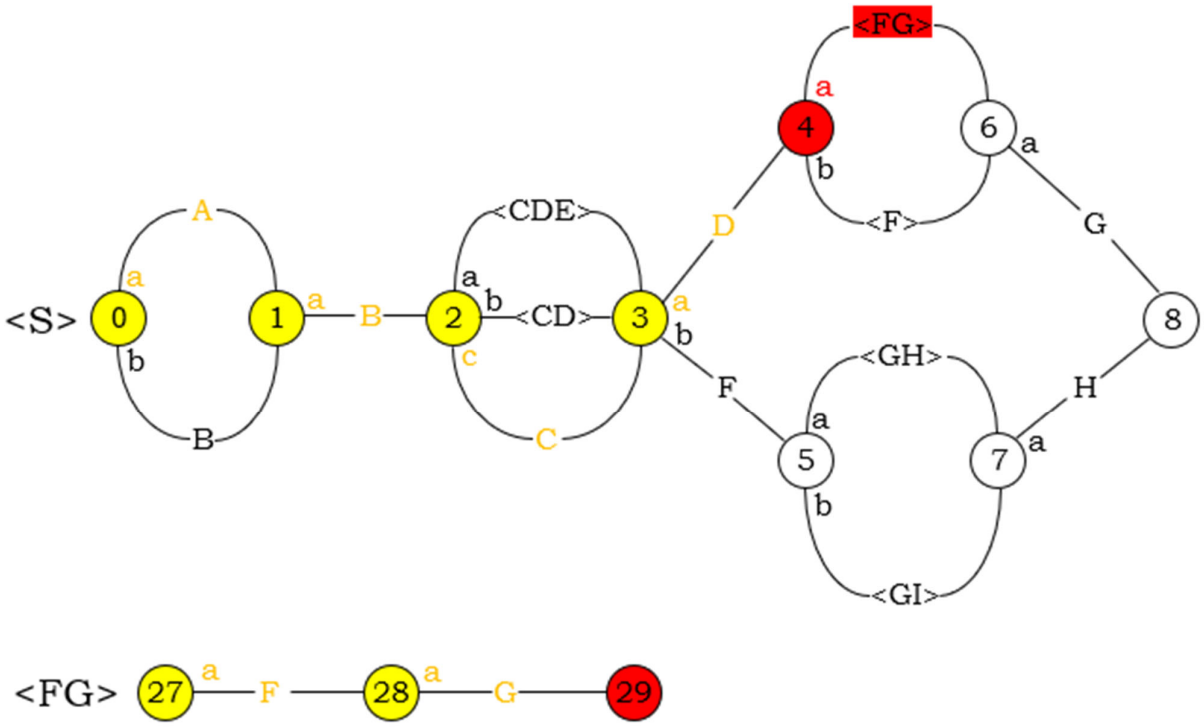
stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) c {} [AB] [ABC]		
3 (DFG) a {b} [ABC] [ABCD]		
4 (FG) a {b} [ABCD] []	.....	← call <FG>
27 (FG) a {} [] [F]		
28 (G) a {} [F] [FG]		
29 () Success		



stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) c {} [AB] [ABC]		
3 (DFG) a {b} [ABC] [ABCD]		
4 (FG) a {b} [ABCD] []	.....	←
27 (FG) a {} [] [F]		
28 (G) a {} [F] [FG]		
29 () Success		
6 () FAIL, G needed		

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []          .....          ← call <FG>
27 (FG) a {} [] [F]
28 (G) a {} [F] [FG]
29 () Success
6 () FAIL, G needed
```

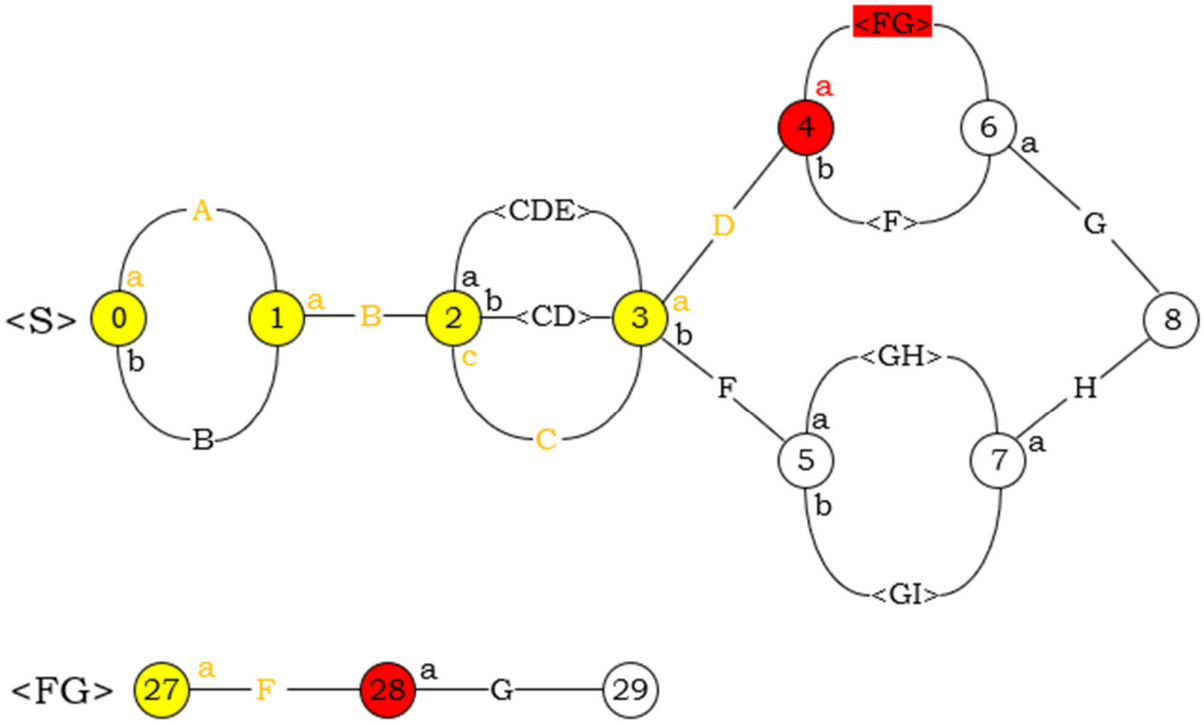


stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []          .....          ← call <FG>
27 (FG) a {} [] [F]
28 (G) a {} [F] [FG]
29 () Success XXX failure continues
```

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []          .............   ← call <FG>
27 (FG) a {} [] [F]
28 (G) a {} [F] [FG]
29 ()  Success XXX failure continues
```

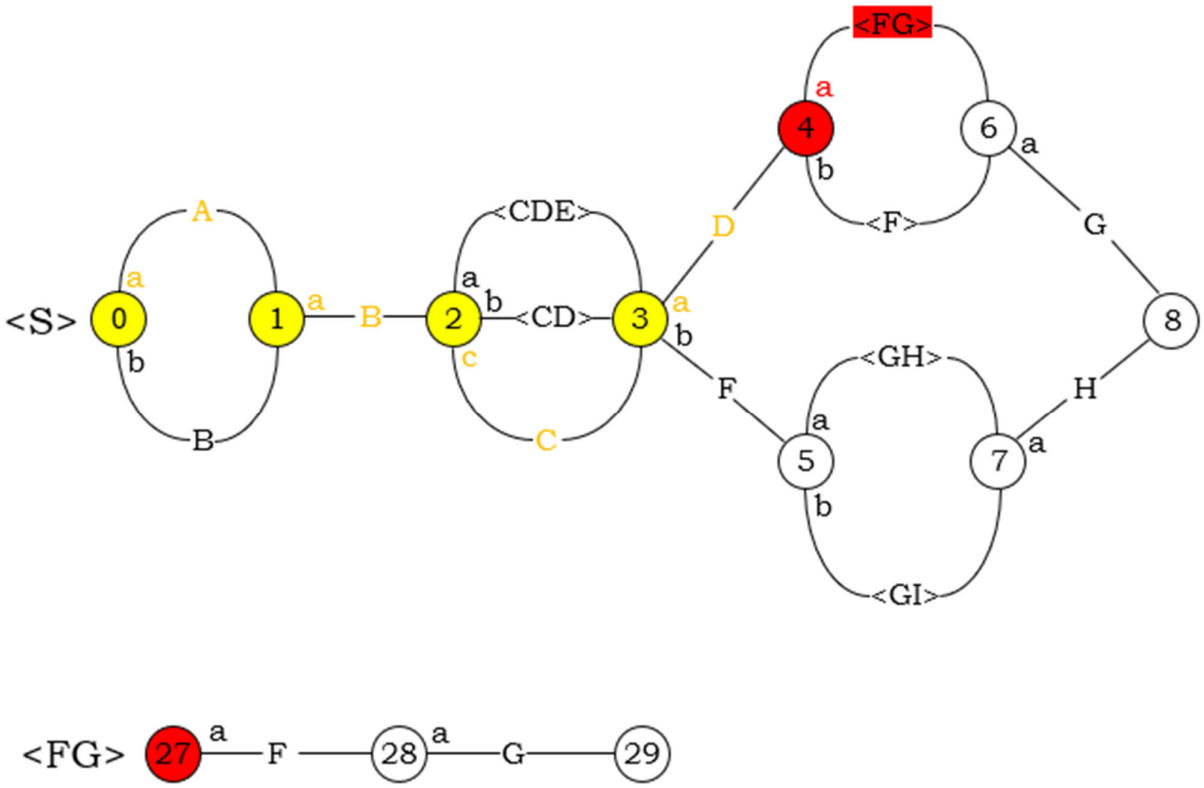


stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []          .............   ← call <FG>
27 (FG) a {} [] [F]
28 (G) a {} [F] [FG]  FAIL, no options
```

stack | metastack

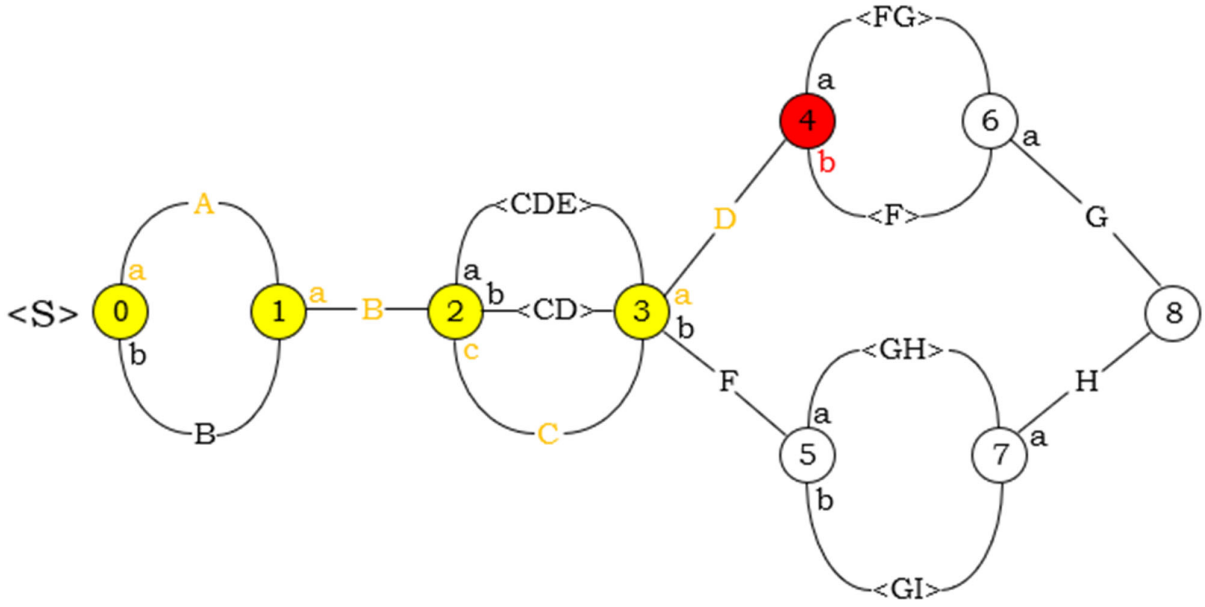
```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []          .....          ← call <FG>
27 (FG) a {} [] [F]
28 (G) a {} [F] [FG]  FAIL, no options
```



stack | metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) a {b} [ABCD] []          .....          ← call <FG>
27 (FG) a {} [] [F]  FAIL, no options
```

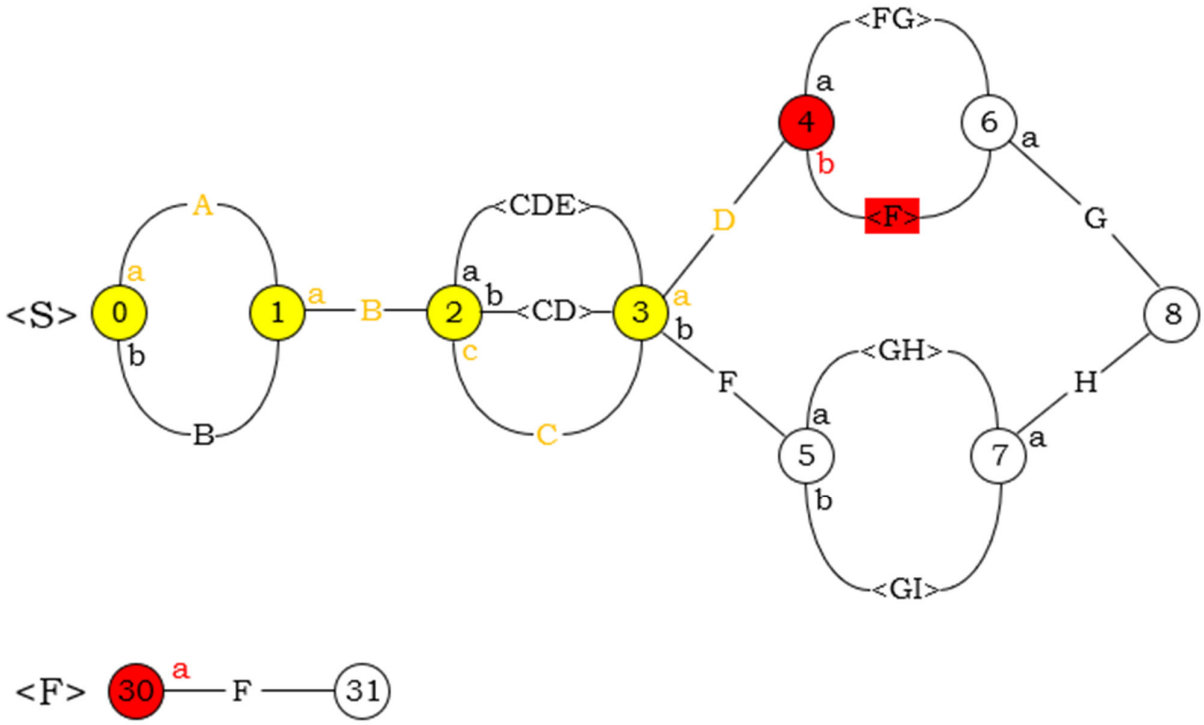
stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) c {} [AB] [ABC]		
3 (DFG) a {b} [ABC] [ABCD]		
4 (FG) a {b} [ABCD] []	.....	← call <FG>
27 (FG) a {} [] [F] FAIL, no options		



stack		metastack
0 (ABCDFG) a {b} [] [A]		
1 (BCDFG) a {} [A] [AB]		
2 (CDFG) c {} [AB] [ABC]		
3 (DFG) a {b} [ABC] [ABCD]		
4 (FG) a {b} [ABCD] []	.....	← call <FG>
changes to		
4 (FG) b {} [ABCD] []	.....	← call <F>

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .............   ← call <F>
```

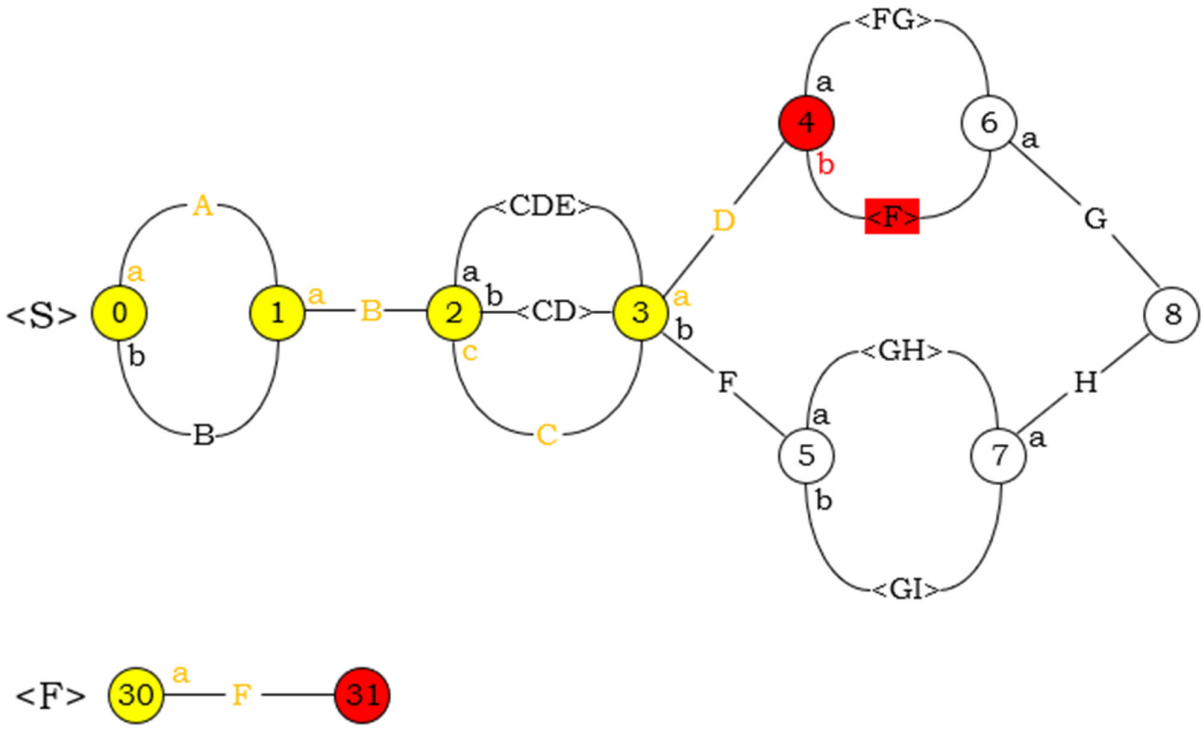


stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .............   ← call <F>
30 (FG) a {} [] [F]
```

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .....          ← call <F>
30 (FG) a {} [] [F]
```

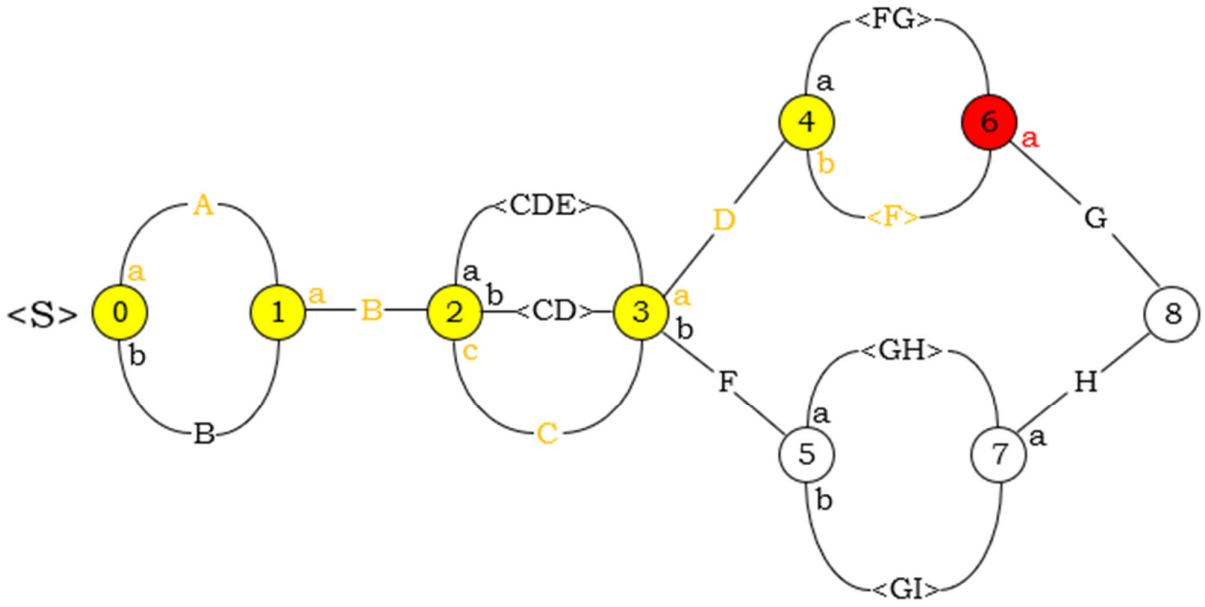


stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .....          ← call <F>
30 (FG) a {} [] [F]
31 (G) Success
```

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .....          ← call <F>
30 (FG) a {} [] [F]
31 (G) Success
```

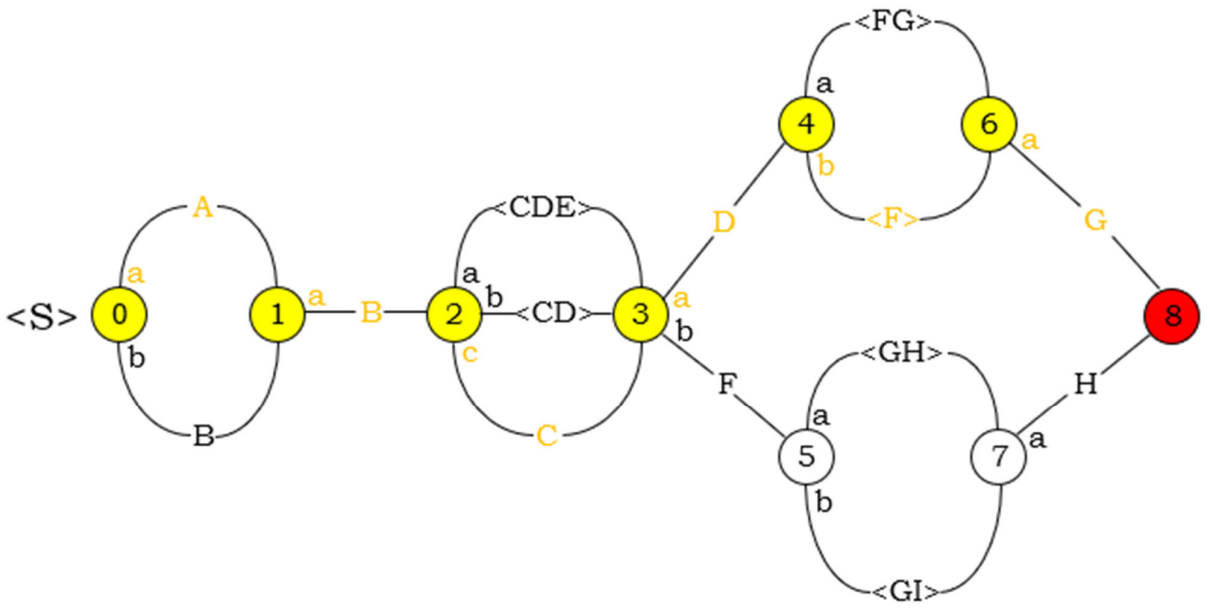


stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .....          ← call <F>
30 (FG) a {} [] [F]
31 (G) Success
6 (G) a {} [ABCD[F]] [ABCD[F]G]
```

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .............   ← call <F>
30 (FG) a {} [] [F]
31 (G) Success
6 (G) a {} [ABCD[F]] [ABCD[F]G]
```



stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .............   ← call <F>
30 (FG) a {} [] [F]
31 (G) Success
6 (G) a {} [ABCD[F]] [ABCD[F]G]
8 END
```

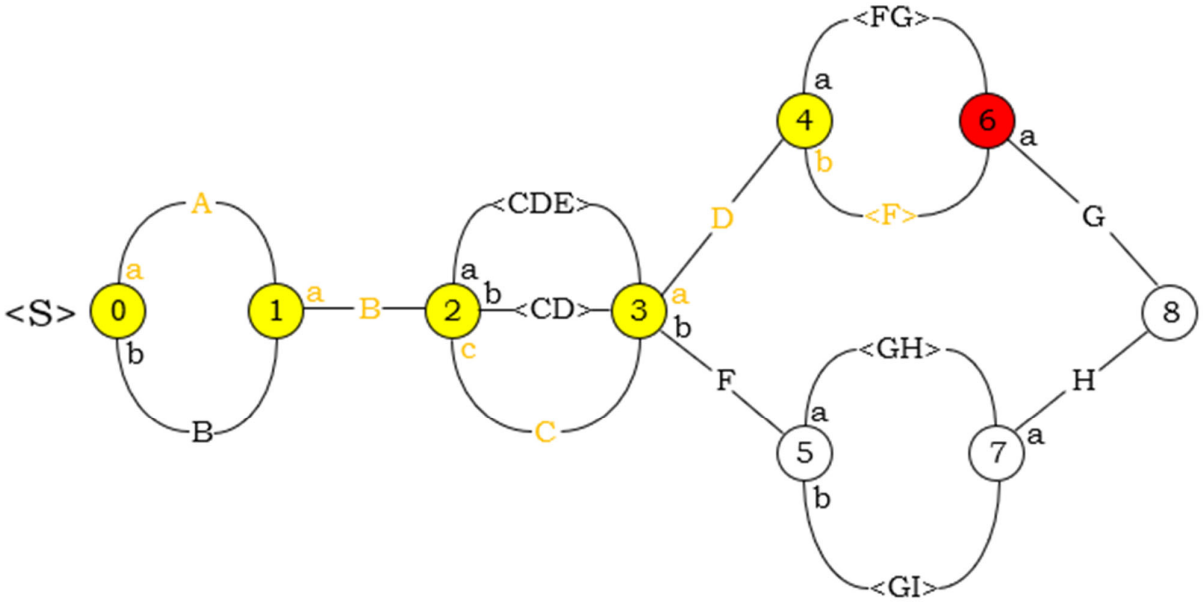
The solution is printed.

What if it is ambiguous and there is more than one valid parsing?

Pretend it was a failure and backtrack as normal

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .............   ← call <F>
30 (FG) a {} [] [F]
31 (G) Success
6 (G) a {} [ABCD[F]] [ABCD[F]G]
8 pretend to FAIL
*
```

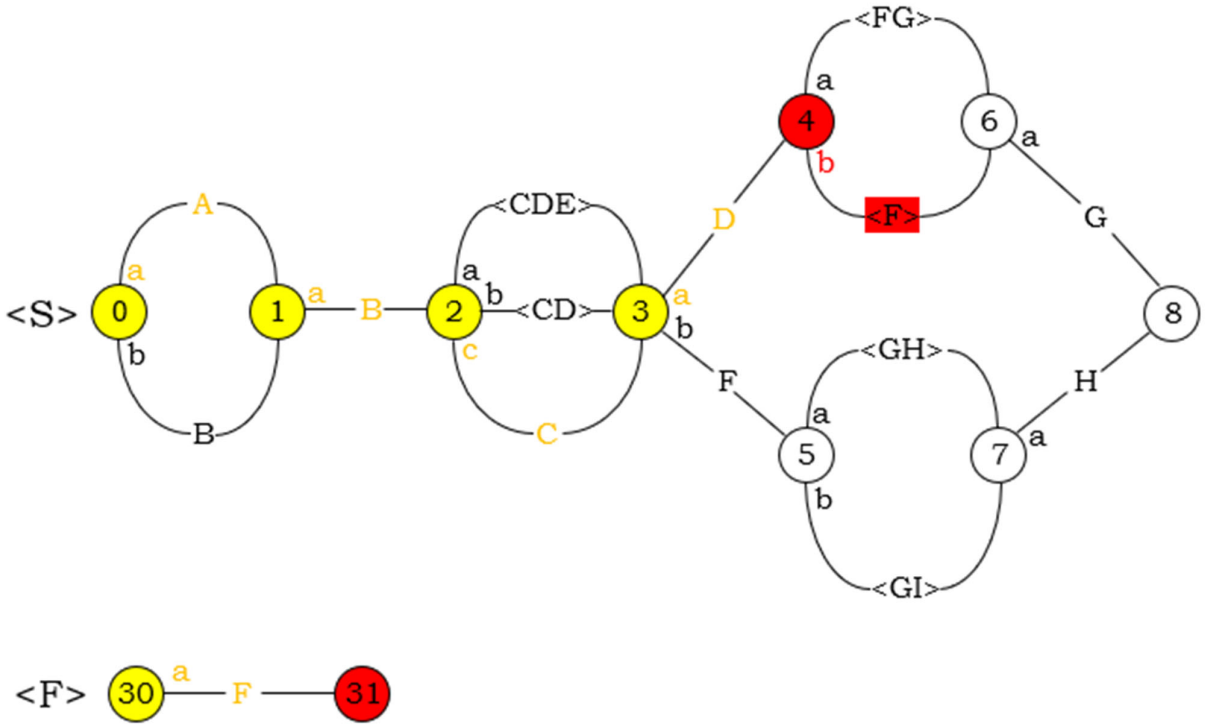


stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .............   ← call <F
30 (FG) a {} [] [F]
31 (G) Success
6 (G) a {} [ABCD[F]] [ABCD[F]G]  FAIL, no options
```

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .............   ← call <F>
30 (FG) a {} [] [F]
31 (G) Success
6 (G) a {} [ABCD[F]] [ABCD[F]G] FAIL, no options
```

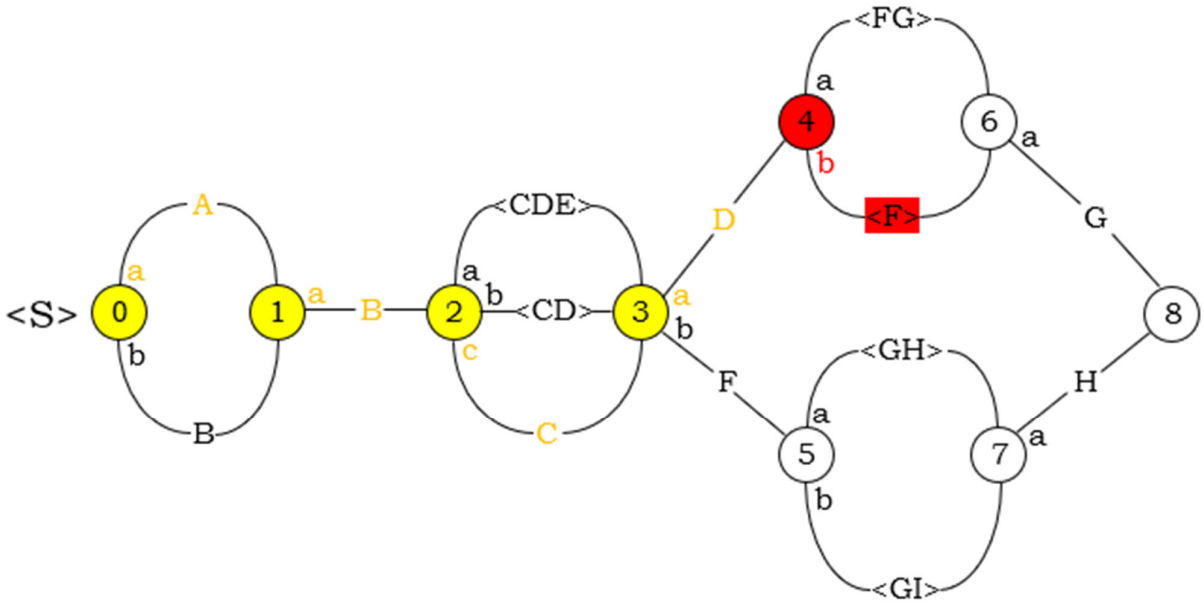


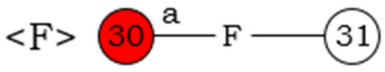
stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .............   ← call <F>
30 (FG) a {} [] [F]
31 (G) Success XXX failure continues
```

stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .....          ← call <F>
30 (FG) a {} [] [F]
31 (G) Success XXX failure continues
```



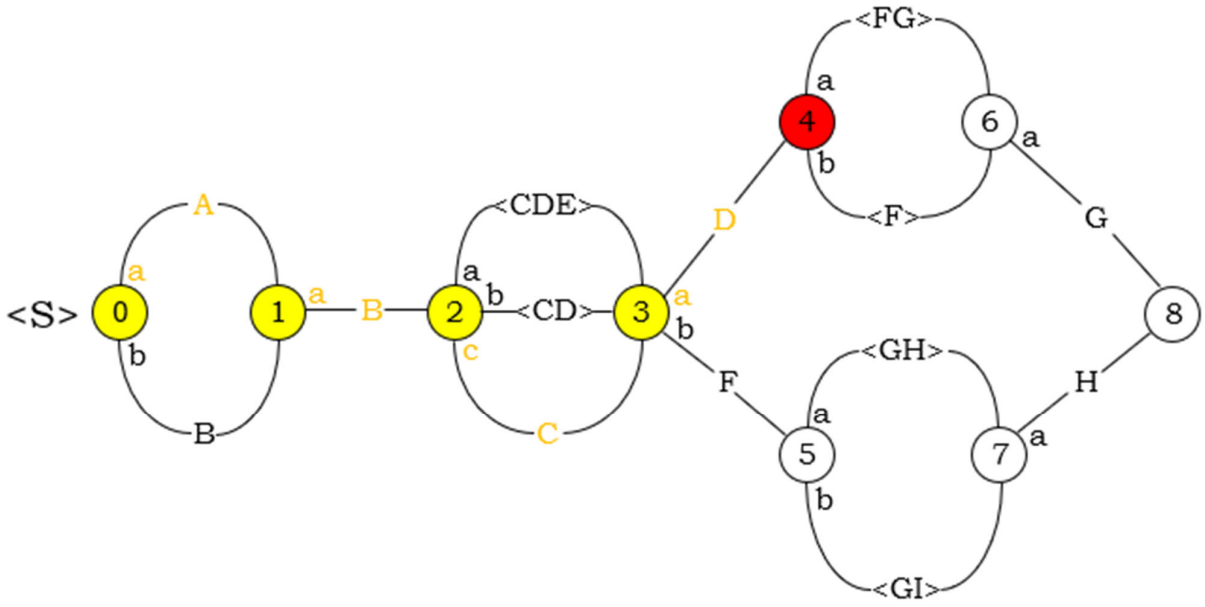


stack metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .....          ← call <F>
30 (FG) a {} [] [F] FAIL, no options
```

stack | metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []          .....          ← call <F>
30 (FG) a {} [] [F] FAIL, no options
```



stack | metastack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] [] FAIL, no options
```

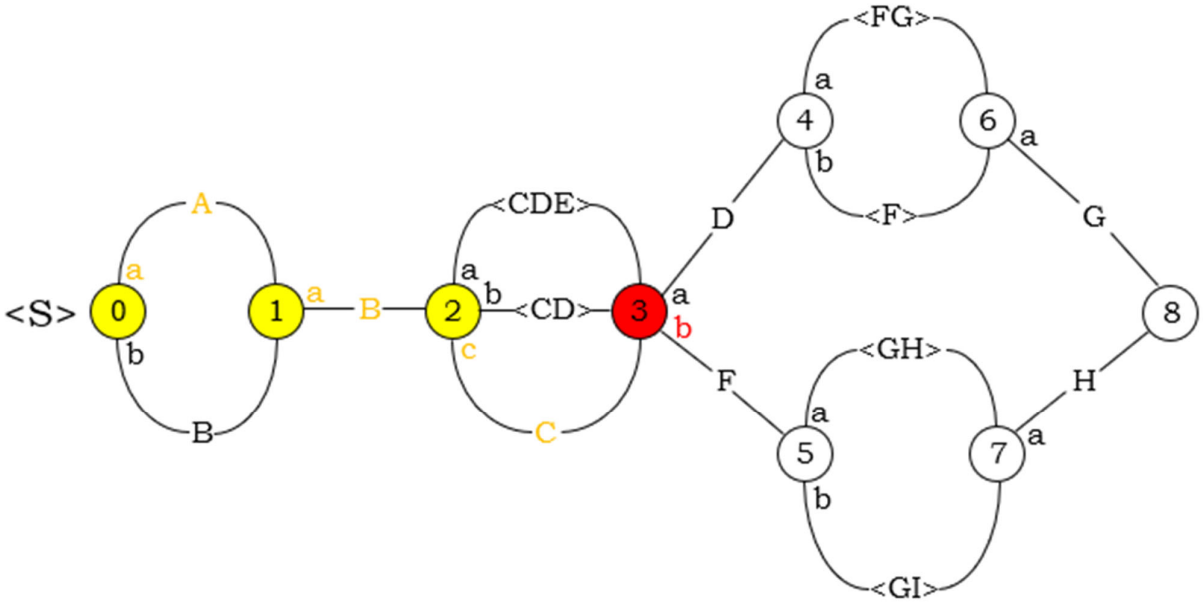
← call <F>

stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
4 (FG) b {} [ABCD] []  FAIL, no options
```

metastack

← call <F>



stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) a {b} [ABC] [ABCD]
```

changes to

```
3 (DFG) b {} [ABC] [ABCD]
```

metastack

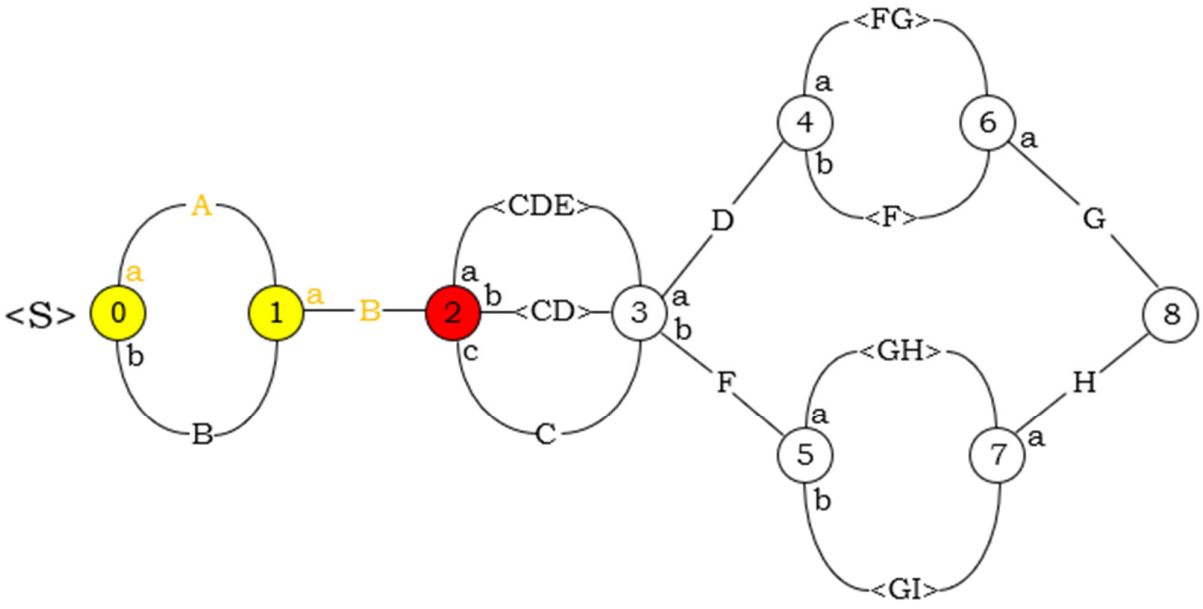
empty

This is where we see there could have been another valid parse, the F needed here could have been a D.

stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
3 (DFG) b {} [ABC] [ABCD]
```
FAIL, F needed

metastack
empty



stack

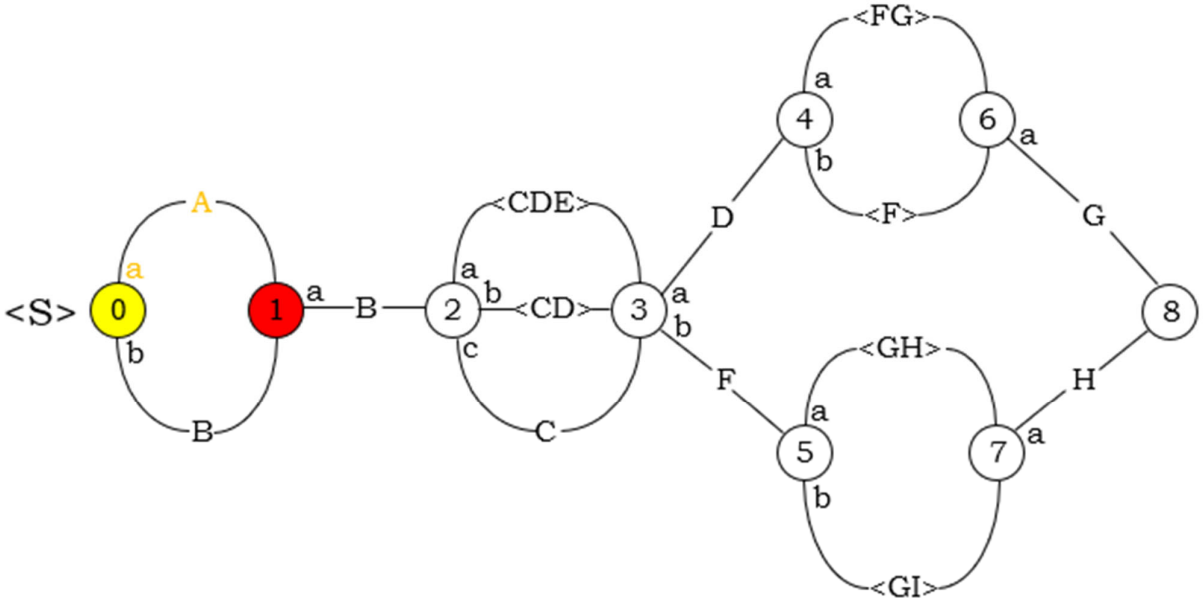
```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
```
FAIL, no options

metastack
empty

stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
2 (CDFG) c {} [AB] [ABC]
```
FAIL, no options

metastack
empty



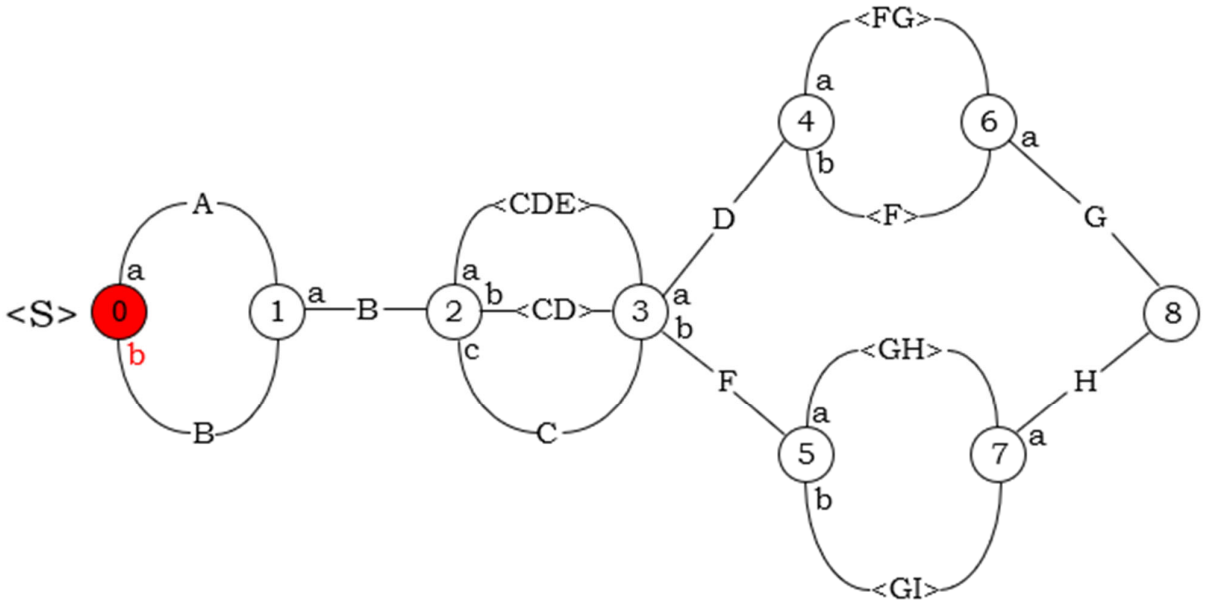
stack

```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
```
FAIL, no options

metastack
empty

stack
```
0 (ABCDFG) a {b} [] [A]
1 (BCDFG) a {} [A] [AB]
```
FAIL, no options

metastack
empty



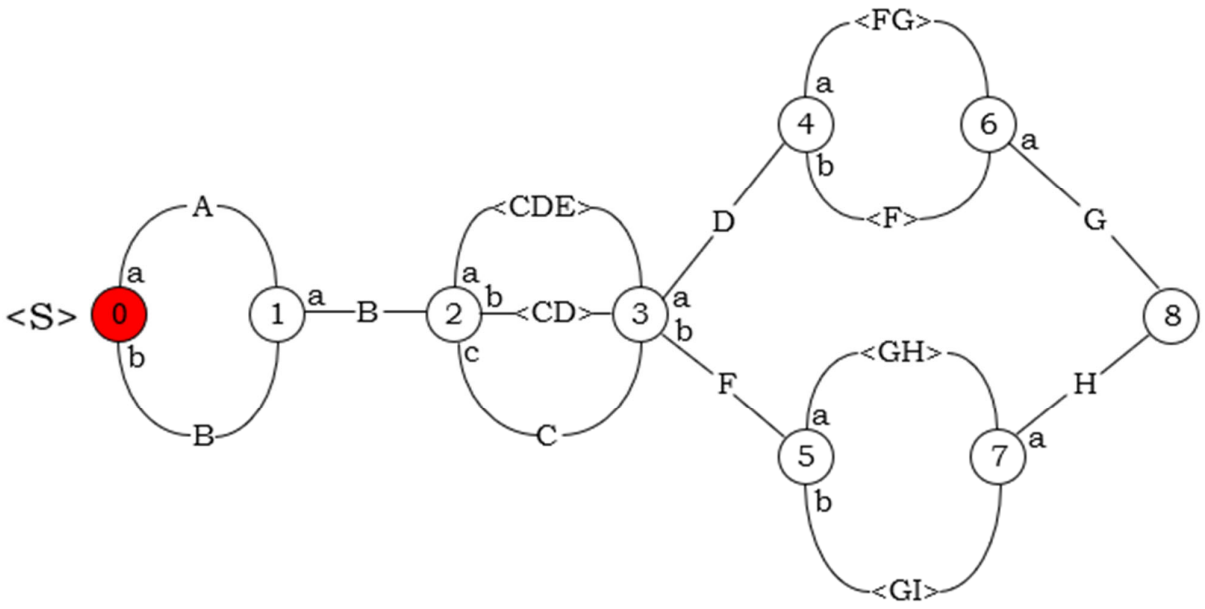
stack
```
0 (ABCDFG) a {b} [] [A]
```
changes to
```
0 (ABCDFG) b {} [] [A]
```

metastack
empty

There we had another chance for an extra success.

stack
0 (ABCDFG) b {} [] [A] FAIL, B needed

metastack
empty



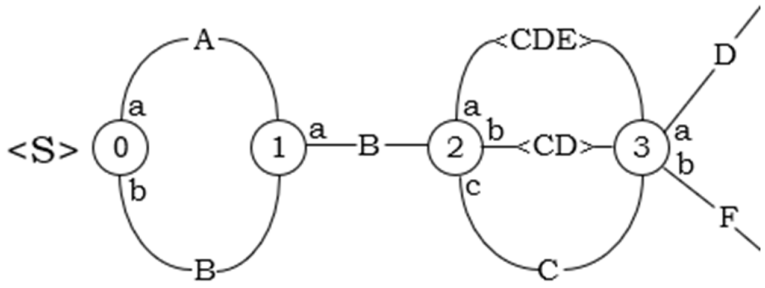
stack
empty

metastack
empty

No more options and the stack is empty.

Empty stack so there were no other valid parses after all.

An easy (but not so nice to look at) computer-friendly representation of the ATN for the first part of the example:



(in this, "state" refers to the numbered nodes in the ATN graph. The state that the algorithm requires would consist of all the things we saw on the stack, e.g. `3 (DFG) a {b} [ABC] [ABCD]`, plus a few more)

One possible transition (arc) per line:

current state
condition required if this transition is to happen
extra actions to take when this transition happens
new state reached after this transition (if there is one, successes don't have one)

Remember Python's `eval()` function. It will make the things that seem difficult or over complicated more or less vanish.

```
0 ("cat", "A") [("consume"), ("build")] 1
0 ("cat", "B") [("consume"), ("build")] 1
1 ("cat", "B") [("consume"), ("build")] 2
2 ("sub", "CDE") [("build", "CDE")] 3
2 ("sub", "CD") [("build", "CD")] 3
2 ("cat", "C") [("consume"), ("build")] 3
3 ("cat", "D") [("consume"), ("build")] ...
3 ("cat", "F") [("consume"), ("build")] ...
```

Sensible changes to the syntax for conditions and actions are acceptable.

conditions:

`("cat", X)`	category:	the next input word must be of type X, e.g. noun, verb.
`("word", X)`	category:	next input must be a specific word, X would be either a string or a set.
`("sub", X)`	subgraph:	the subgraph starting at state X must be "called" the transition only occurs if it is successful
`("succeed")`	successful return from subgraph; return the built parse object	
`("fail")`	abandon subgraph	
`("or", X, Y, ...)`	any one or more of conditions X, Y, ... must be true	
`("and", X, Y, ...)`	all of conditions X, Y, ... must be true	
all sorts of others		

actions:

`("consume")`	the input word is taken from the input stream.
`("build", X)`	whatever we just received (a word for "cat", and "word", or a whole parse object for "sub") is appended to the currently being build parse object. X is optional; without it just append the word, with it

append (X, the word)

Notes:

For "sub" the currently being built parse object is safe on the stack, the called subgraph starts with a new empty build object.

For "succeed" the word "return" does not imply a return from a function, it just means that the built parse object should be treated as a word for the original transition's "build" action.

To save typing you may set all sorts of defaults, such as "word" naturally performs "consume" and "build" actions; ("cat", X) naturally performs "consume" and ("build", X) actions; ("sub", X) naturally performs a ("build", x) action, etc. All you have to specify for the default actions to happen is ("dflt")

The "augmentations":

To help understand inputs properly, the states (on the stack) also need a simple memory and simple actions and conditions to make use of it. Like the parse object, each "sub" starts with a new memory for the subgraph.

A very basic example of why: in parts of sentences, some things such as number are supposed to agree. "I is tired" is wrong and being able to tell that can help.

`("note", X, Y)` memory[X] is set to Y

`("known", X)` is X one of memory's keys?

`("is", X, Y)` is X in memory's keys and memory[X] == Y?

`("isin", X, Y)` is X in memory's keys and memory[X] in set Y?

`("dict", X, Y)` obtain extra information from the word list.
words will be annotated with more detailed information,
e.g. for verbs and nouns

```
cat noun number singular
cats noun number plural
eats verb number singular
```

so `("dict", "cats", "number")` produces `"plural"`

`("word")` is the actual input word being considered

`("cat")` is the category (noun, verb, etc) of the input word.