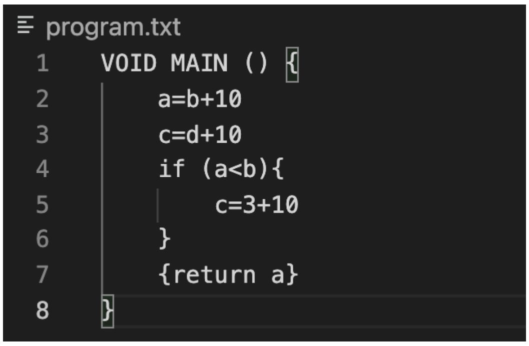
I. Practical

Grammars used in problem 1.

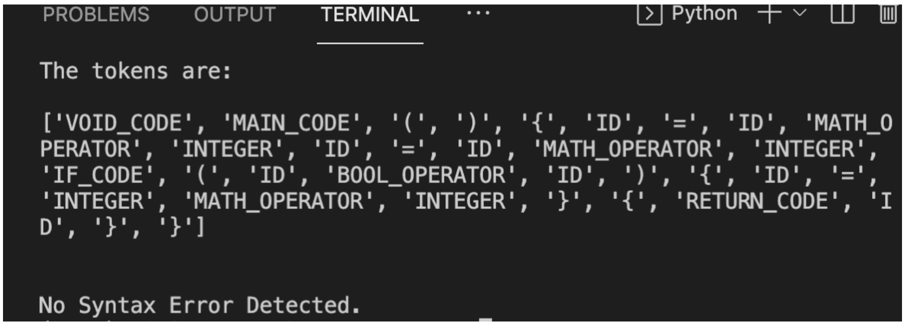
```
# <program> --> VOID MAIN "(" ")" <block>
# <block> --> "{" {<statement>} "}"
# <assign> --> id = < term>
# <term> --> <factor> { (*|/|%) <factor> }
# <factor> --> identifier | int | float
# <returnstmt> --> return <factor>
# <statement> --> <ifstmt>  <assign> |<return>; |<forstmt>| <foreachstmt>| <dowhilestmt>|
<whilestmt> | <switchstmt>
# <forstmt> --> for "(" <ID>”;” <boolstmt> “;”  <assign> ")" <block>
# <ifstmt> --> if (<boolstmt>) "{" <block> "}" else "{" <block> "}"
# <foreachstmt> --> for_each "(" <id>;<id>")" <block>
# <dowhilestmt> --> do <block> while "(" <boolstmt> ")"
# <whilestmt> --> while "(" <boolstmt> ")" <block>
# <switchstmt> --> switch "(" <id>")"  "{"  {<case_stmt>}  "}"
# <casestmt> --> case <factor>  “:” <block>
# <boolstmt> --> <factor> (>|<) <factor>
```

Testing Examples

Testing Example 1:

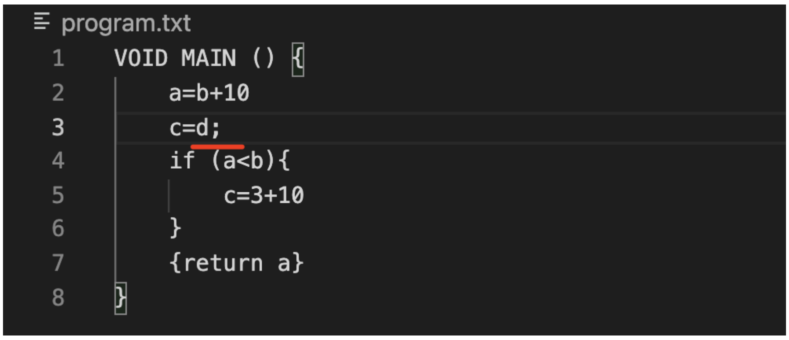


```
VOID MAIN () {
    a=b+10
    c=d+10
    if (a<b){
        c=3+10
    }
    {return a}
}
```

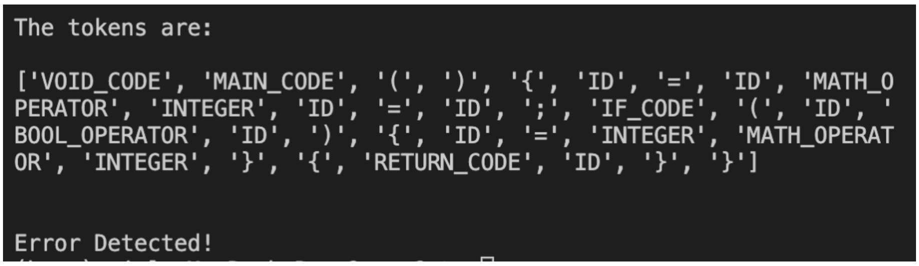


```
The tokens are:

['VOID_CODE', 'MAIN_CODE', '(', ')', '{', 'ID', '=', 'ID', 'MATH_O
PERATOR', 'INTEGER', 'ID', '=', 'ID', 'MATH_OPERATOR', 'INTEGER',
'IF_CODE', '(', 'ID', 'BOOL_OPERATOR', 'ID', ')', '{', 'ID', '=',
'INTEGER', 'MATH_OPERATOR', 'INTEGER', '}', '{', 'RETURN_CODE', 'I
D', '}', '}']


No Syntax Error Detected.
```

Testing Example 2:

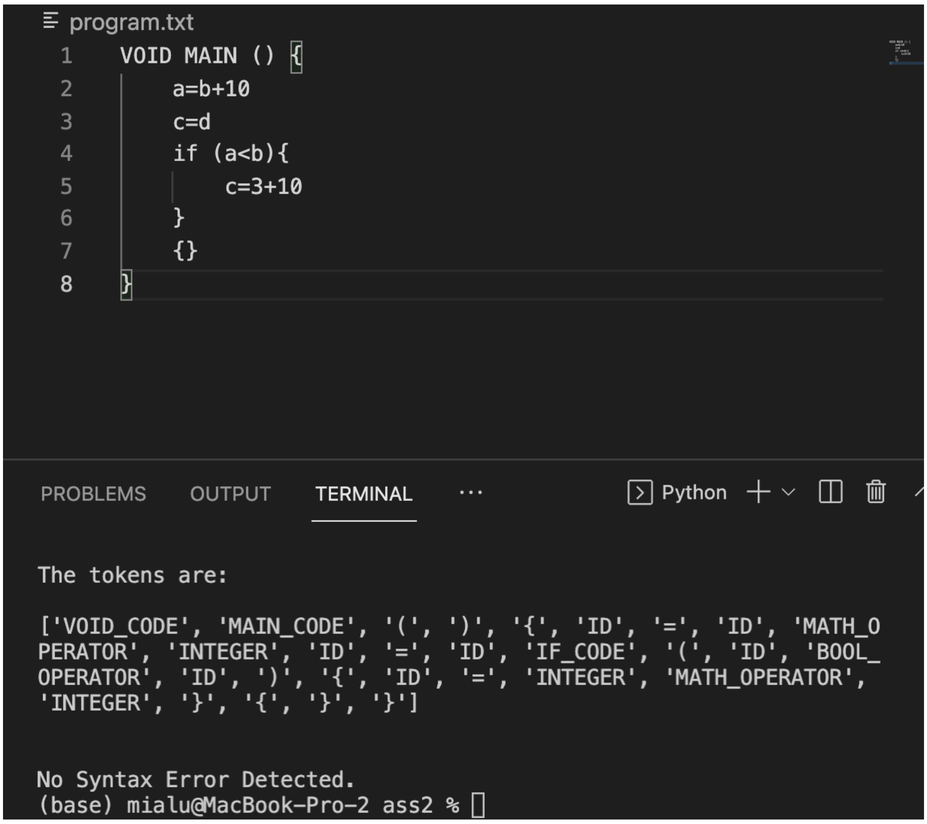


```
VOID MAIN () {
    a=b+10
    c=d;
    if (a<b){
        c=3+10
    }
    {return a}
}
```



```
The tokens are:

['VOID_CODE', 'MAIN_CODE', '(', ')', '{', 'ID', '=', 'ID', 'MATH_O
PERATOR', 'INTEGER', 'ID', '=', 'ID', ';', 'IF_CODE', '(', 'ID', '
BOOL_OPERATOR', 'ID', ')', '{', 'ID', '=', 'INTEGER', 'MATH_OPERAT
OR', 'INTEGER', '}', '{', 'RETURN_CODE', 'ID', '}', '}']


Error Detected!
```

Testing Example 3:



```
VOID MAIN () {
    a=b+10
    c=d
    if (a<b){
        c=3+10
    }
    {}
}
```

```
The tokens are:

['VOID_CODE', 'MAIN_CODE', '(', ')', '{', 'ID', '=', 'ID', 'MATH_O
PERATOR', 'INTEGER', 'ID', '=', 'ID', 'IF_CODE', '(', 'ID', 'BOOL_
OPERATOR', 'ID', ')', '{', 'ID', '=', 'INTEGER', 'MATH_OPERATOR',
'INTEGER', '}', '{', '}', '}']


No Syntax Error Detected.
(base) mialu@MacBook-Pro-2 ass2 %
```

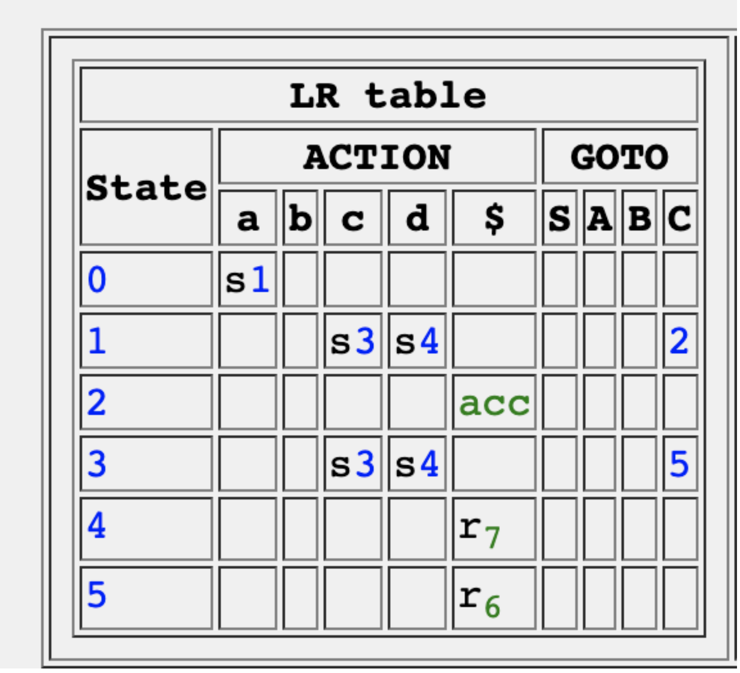
1.

YOUR CHOICE – Choose remaining 50 points

1. 25 points
   Create an LR Parsing table for the following grammar (10 Points) and show the steps to solve the following problems
   S -> a C | A C
   A -> a B b | A a
   B -> b B | c C
   C -> c C | d

   a. abbbccd
   b. accd
   c. acdbaacd
   d. acdbd
   e. abcdbad

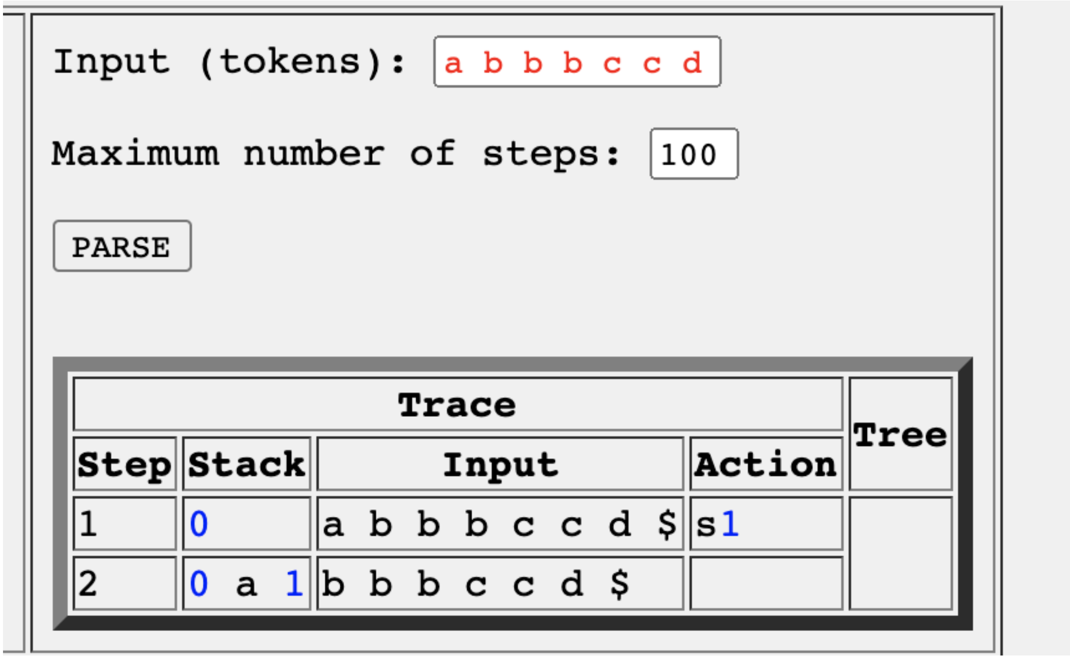
LR table



LR table

State	ACTION					GOTO			
	a	b	c	d	$	S	A	B	C
0	s1								
1			s3	s4					2
2					acc				
3			s3	s4					5
4					r_7				
5					r_6				

a

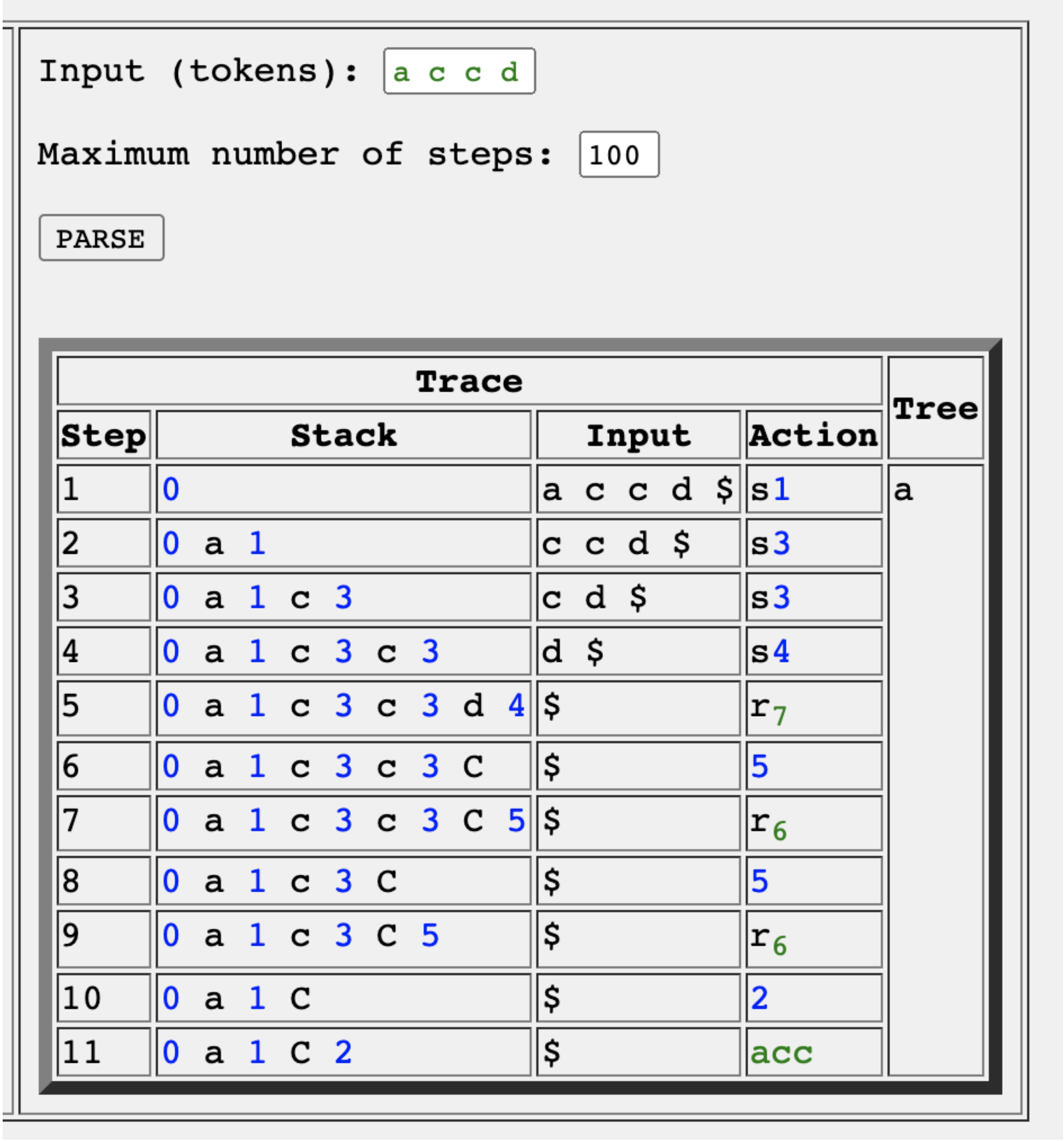


Input (tokens): a b b b c c d

Maximum number of steps: 100

Trace				Tree
Step	Stack	Input	Action	
1	0	a b b b c c d $	s1	
2	0 a 1	b b b c c d $		

b

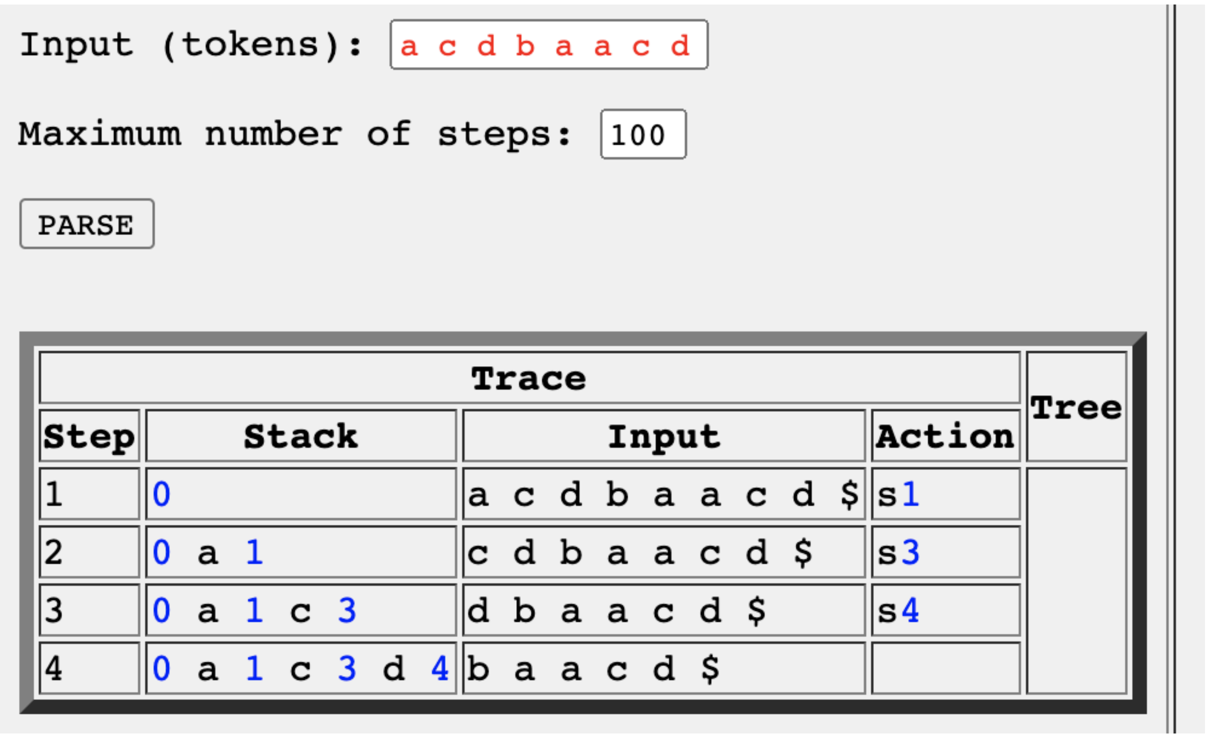


Input (tokens): a c c d

Maximum number of steps: 100

Trace				Tree
Step	Stack	Input	Action	
1	0	a c c d $	s1	a
2	0 a 1	c c d $	s3	
3	0 a 1 c 3	c d $	s3	
4	0 a 1 c 3 c 3	d $	s4	
5	0 a 1 c 3 c 3 d 4	$	r_7	
6	0 a 1 c 3 c 3 C	$	5	
7	0 a 1 c 3 c 3 C 5	$	r_6	
8	0 a 1 c 3 C	$	5	
9	0 a 1 c 3 C 5	$	r_6	
10	0 a 1 C	$	2	
11	0 a 1 C 2	$	acc	

c

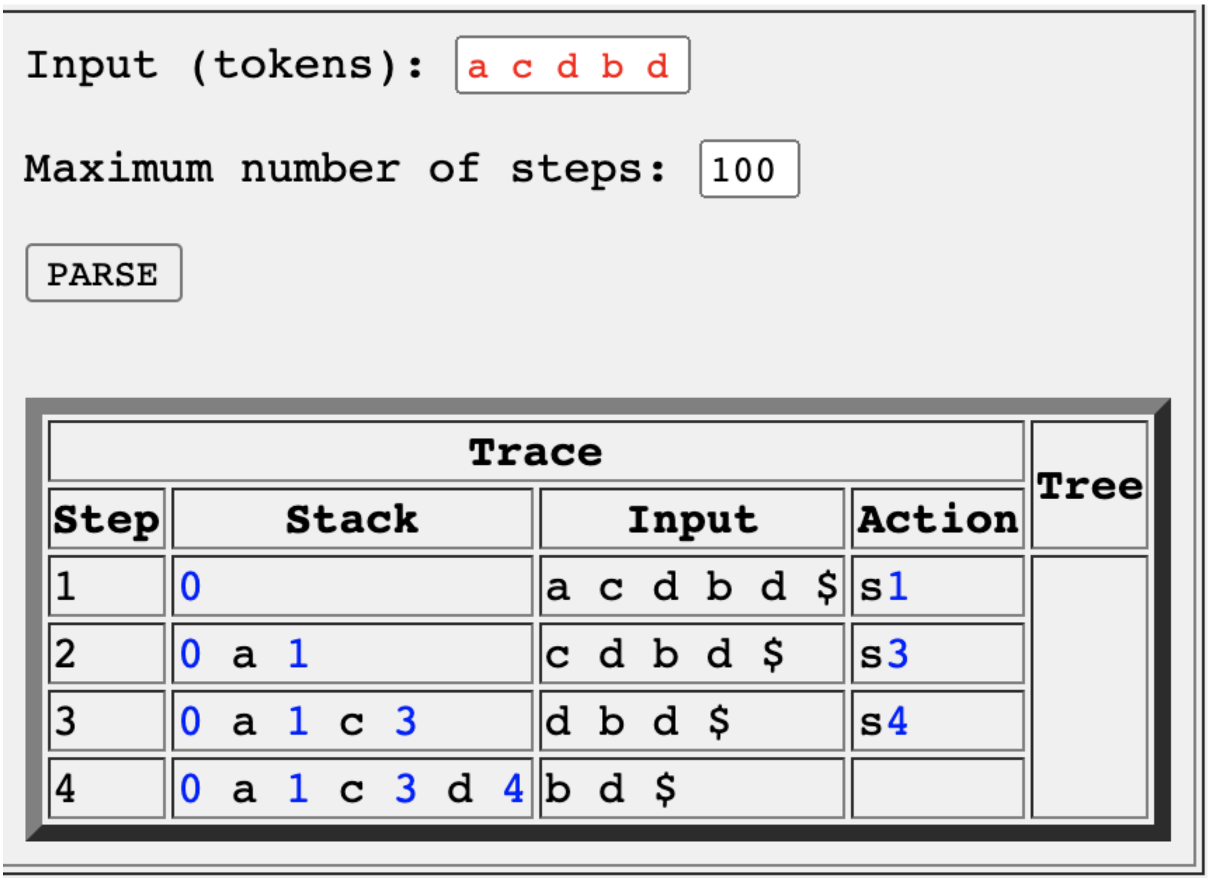


Input (tokens): a c d b a a c d

Maximum number of steps: 100

Trace				Tree
Step	Stack	Input	Action	
1	0	a c d b a a c d $	s1	
2	0 a 1	c d b a a c d $	s3	
3	0 a 1 c 3	d b a a c d $	s4	
4	0 a 1 c 3 d 4	b a a c d $		

d

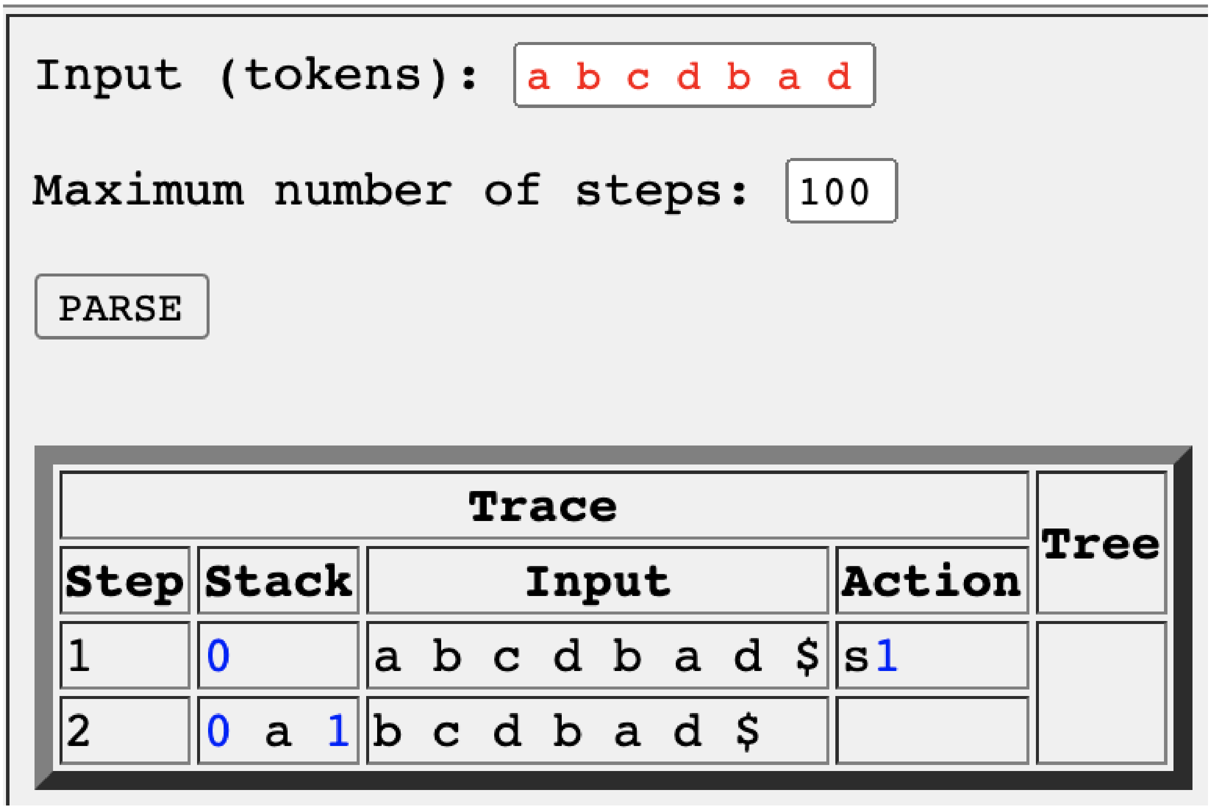


Input (tokens): a c d b d

Maximum number of steps: 100

Trace				Tree
Step	Stack	Input	Action	
1	0	a c d b d $	s1	
2	0 a 1	c d b d $	s3	
3	0 a 1 c 3	d b d $	s4	
4	0 a 1 c 3 d 4	b d $		

e



Input (tokens): a b c d b a d

Maximum number of steps: 100

Trace				Tree
Step	Stack	Input	Action	
1	0	a b c d b a d $	s1	
2	0 a 1	b c d b a d $		

3

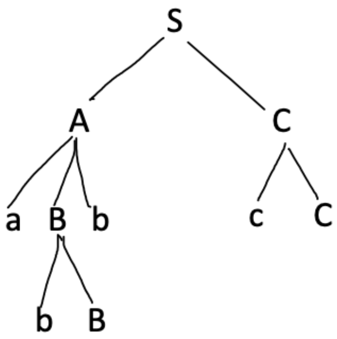
3. (25 points) Given the grammar from the previous problem if the string is in the language show the parse tree, right most derivation, handle, phrases and simple phrases for the following strings, or prove they are not in the language

   a. abBbcC
   b. accCd
   c. aCbaacd
   d. acdabd
   e. abCbad

a

S -> aC | AC
A -> aBb | Aa
B -> bB | cC
C -> cC | d

a. abBbcC



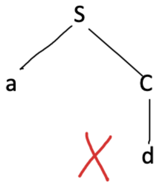
S	PHRASE	SIMPLE PHRASE	HANDLE
AC	abBbcC	cC	bB
AcC	cC	bB	
aBbcC	abBb		
abBbcC	bB		

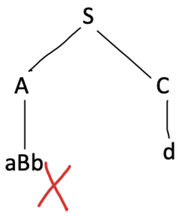
b

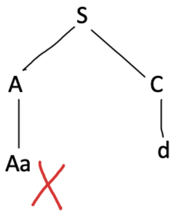
It is not in the language.

S -> aC | AC
A -> aBb | Aa
B -> bB | cC
C -> cC | d

b. accCd







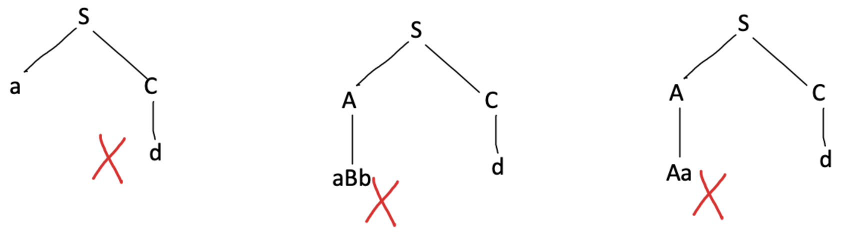
Above figures show the right most derivation.
fig 1: we can get to d. Then the next is ...ad, not ...Cd
fig 2: we can get to d. Then the next is ...bd, not ....Cd
fig 3: we can get to d. Then the next is … ad, not … Cd.

c

It is not in the language

S -> aC | AC
A -> aBb | Aa
B -> bB | cC
C -> cC | d

c. aCbaacd



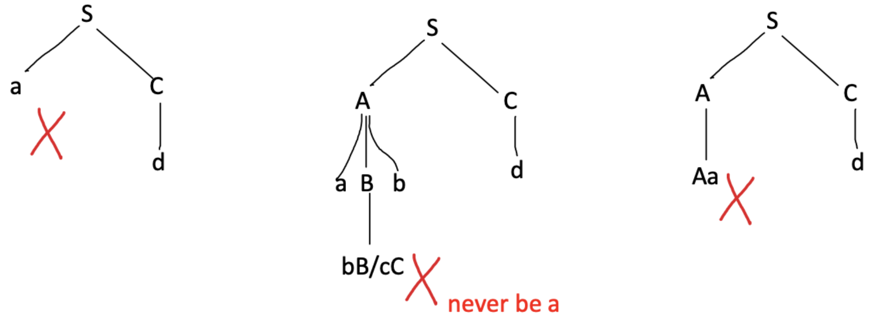
Above figures show the right most derivation.
fig 1: we can get to d. Then the next is ...ad, not ...cd
fig 2: we can get to d. Then the next is ...bd, not ....cd
fig 3: we can get to d. Then the next is … ad, not … cd.

d

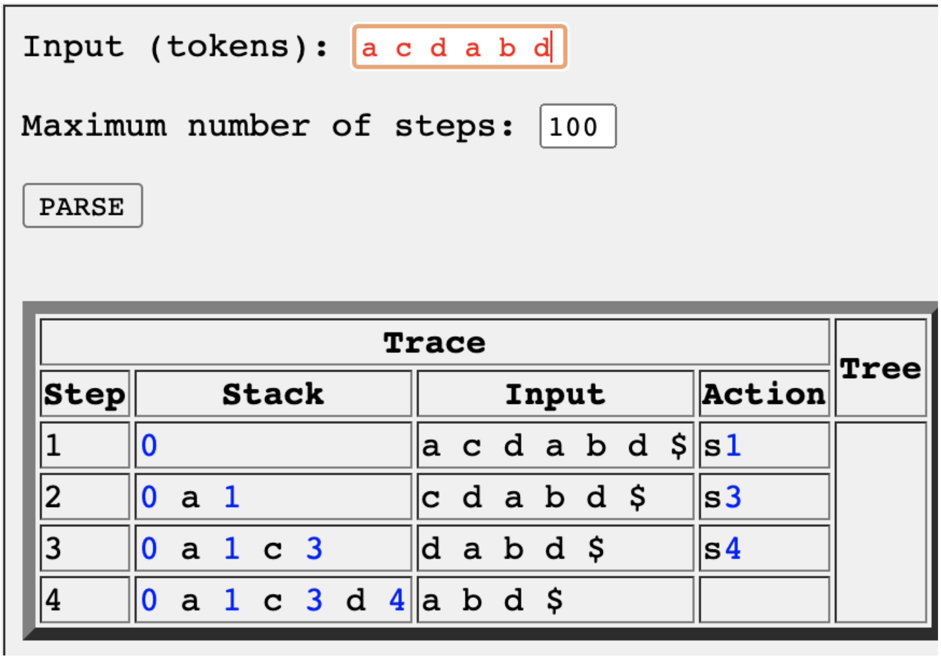
Not in the language

S -> aC | AC
A -> aBb | Aa
B -> bB | cC
C -> cC | d

d. acdabd



Above figures show the right most derivation.
fig 1: we can get to d. Then the next is ...ad, not ...bd
fig 2: we can get to bd. Then the next will never be ...abd
fig 3: we can get to d. Then the next is … ad, not … bd.



Input (tokens): a c d a b d

Maximum number of steps: 100

PARSE

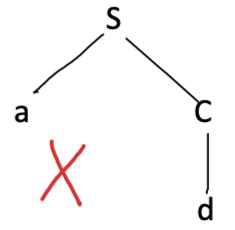
Trace				Tree
Step	Stack	Input	Action	
1	0	a c d a b d $	s1	
2	0 a 1	c d a b d $	s3	
3	0 a 1 c 3	d a b d $	s4	
4	0 a 1 c 3 d 4	a b d $		

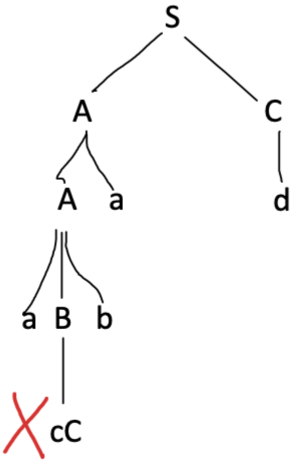
e

Not in the language.

S -> aC | AC
A -> aBb | Aa
B -> bB | cC
C -> cC | d

e. abCbad





Above figures show the right most derivation.
fig 1: we can get to ad. Then the string is terminated and cannot go further to ...bad.
fig 2: we can get to Cbad. Then the next is ...cCbad, not ... bCbad.