course: CSC 135 - Computing Theory and Programming Languages

instructor: Ted Krovetz

related notes: 2022-04-12 2022-04-07-CSC135-01-LEC-parsing

Pushdown Automata Parsing

W15.2 | Tuesday, April 12, 2022 | 10:27 AM

PDA Parsing Example01

$S \rightarrow aS$
$S \rightarrow T$
$T \rightarrow bT$
$T \rightarrow R$
$R \rightarrow cR$
$R \rightarrow \lambda$

Step01: Find which are Nullable?

All are Nullable: S, T, R

Step02: What is our set constraints?

Grammar		
$S \rightarrow aS$	$a \in first(S)$	
$S \rightarrow T$	$First(T) \leq First(S)$	$First(S) \leq First(T)$
$T \rightarrow bT$	$b \in first(T)$	
$T \rightarrow R$	$First(R) \leq First(T)$	$First(T) \leq First(R)$
$R \rightarrow cR$	$c \in first(R)$	
$R \rightarrow \lambda$		
$R \rightarrow \$\$$	$\$ \in Follows(S)$	

S → aS → aT → abT → abbT → abbR → abbc

- `abbR` → $R \rightarrow \lambda$ → abbc →

Step03: Build out our sets

	FIRST	Follow
S	a, b, c	$\$$
T	b, c	$\$$
R	c	$\$$

STEP04: Build out our prediction table

Grammar	FIRST RHS	IF RHS Nullable FOLLOWS LHS	PREDICTORS
$S \rightarrow aS$	a	$-$	a
$S \rightarrow T$	b, c	$\$$	$b, c, \$$
$T \rightarrow bT$	b	$-$	b
$T \rightarrow R$	c	$\$$	$c, \$$
$R \rightarrow cR$	c	$-$	c
$R \rightarrow \lambda$	$-$	$\$$	$\$$

STEP05: Is this grammar suitable for LL(1) parsing?

Check for three properties - if all are "NO" then yes the grammar is suitable for LL(1) parsing - Need prediction table

1. Is ambiguous grammar? - **has to be "NO"**
    1. Two left most ___
    2. Does it have any left recursion? - Is it left recursive? - **has to be "NO"**
        1. $A \rightarrow fw$
    3. Are there conflicting predictors for any non-terminal? - **has to be "NO"**

STEP 06: now write the parser...

Code writing the parser - Pushdown Automata Parsing

```
def parse(input):
    toks = scanner(input)
    stack = ['S']
    while len(stack) > 0:
        top = stack.pop()                # Always pop top of stack
```

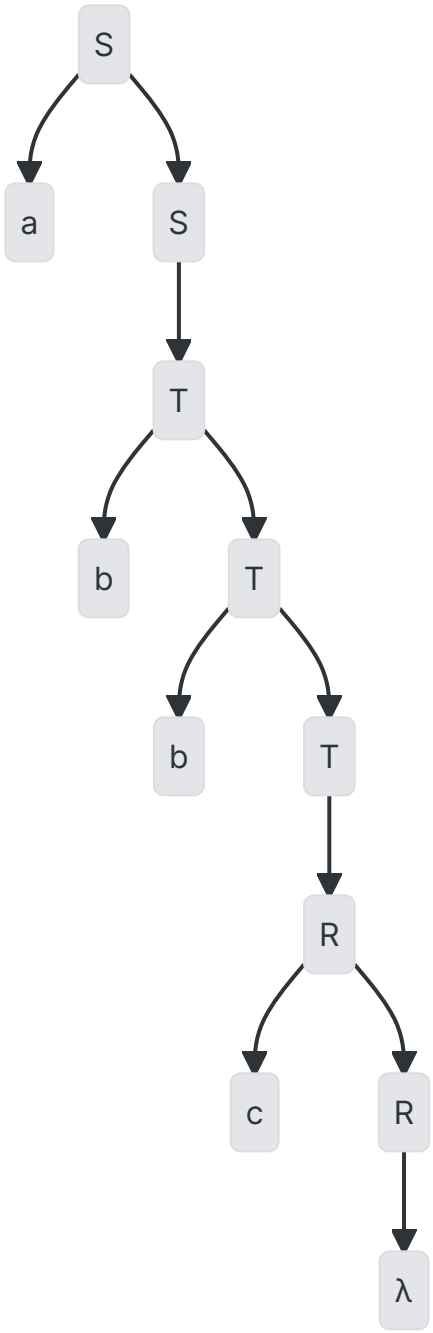
```
            tok = toks.next()                # None indicates token stream
    empty
            if top in ('a', 'b', 'c'):       # Try input/stack match
                toks.match(top)
            elif top == 'S' and tok == 'a':
                stack.append('S')
                stack.append('a')
            elif top == 'S' and (tok == None or tok in ('b', 'c')):
                stack.append('T')
            elif top == 'T' and (tok == None or tok in ('b'):
                stack.append('R')            # production to follow here
            elif top == 'R' and tok == 'c':
                stack.append('R')
                stack.append('c')            # production to follow here
            elif top == 'R' and tok == None:
                pass # Push nothing
            else:
                raise Exception              # Unrecognized top/tok
    combination
        if toks.next() != None:
            raise Exception
```

Recursive Decent Parsing

If you have a production

Production	
$A \rightarrow bCD$	in: b

b
C
D
STACK
where b, C, D is our to-do list



CODE: Recursive Decent Parsing

```
def parseT(toks):
        toks = toks.next()
        if tok == 'b'
                toks.match(toks)
                parseS(toks)
        elif tok == None or tok == 'c':
                parseR(toks)
        else:
                rise Exception
```

```
def parseS(toks):
        tok = toks.next()
        if tok == 'a'
                toks.match(toks)
                parseS(toks)
        elif tok == None or tok in ('b', 'c'):
                parseT(toks)
        else:
                rise Exception


def recursive_parse(input):
        toks = scanner(input)
        parseS(toks)
        if toks.next() != None:
                raise Exception


try:
        recursive_parse("aabbcc")
except:
        print("Reject")
else:
        print("Accept")
```