CS 1622 – Homework 3 Answers

Please submit a typewritten document. I'd prefer you draw your DFA on the computer, but if this is a challenge, you may hand draw them neatly on the paper by hand.

1.) For the grammar from homework 1, rewrite it to encode precedence and left factor. Not (!) should be highest precedence, ^ middle precedence, and v should be lowest.

E → E ^ E

E → E v E

E → ! E

E → true

E → false

We want "v" to be highest in the parse tree, so we add a new nonterminal for E to go to, etc.:

E → A v E

E → A

A → B ^ A

A → B

B → ! C

B → C

C→ True

C→ False

The productions E → A v E | A and A → B ^ A | B need to be left factored.

E → A X

X → v E | ε

A → B Y

Y → ^ A | ε

B → ! C | C

C → True | False

The grammar is now LL(1)

There is a second way to go with this. We used right recursion above to avoid any problematic left recursion. If we started with a left recursive grammar like:

E → E v A | A

A → A ^ B | B

B → ! C | C

C→ True | False

We'd have to do immediate left-recursion removal. We stated that for two productions:

N → N x | y

We could rewrite without left recursion as:

N → y N'

N' → x N' | ε

Following that pattern, we could have the grammar:

E → A X

X → v A X | ε

A → B Y

Y → ^ B Y | ε

B → ! C | C

C → True | False

Which is also LL(1).

2.) Construct an LL(1) parse table for your grammar from 1. Show the First and Follow sets you generated.

SYMBOL	FIRST SET	FOLLOW SET
E	!, True, False	$
X	v, ε	$
A	!, True, False	v, $
Y	^, ε	v, $
B	!, True, False	v, ^, $
C	True, False	v, ^, $

	v	^	!	true	False	$
E			E → A X	E → A X	E → A X	
X	X → v E					X → ε
A			A → B Y	A → B Y	A → B Y	
Y	Y → ε	Y → ^ A				Y → ε
B			B → ! C	B → C	B → C	
C				C → True	C → False	

3.) Show the LL(1) parsing action list for the input:

!false v false ^ false

Stack	Input	Action
E $	!false v false ^ false $	E → A X
A X $	!false v false ^ false $	A → B Y
B Y X $	!false v false ^ false $	B → ! C
! C Y X $	!false v false ^ false $	**terminal**
C Y X $	false v false ^ false $	C → false
false Y X $	false v false ^ false $	**terminal**
Y X $	v false ^ false $	Y → ε
X $	v false ^ false $	X → v E
v E $	v false ^ false $	**terminal**
E $	false ^ false $	E → A X
A X $	false ^ false $	A → B Y
B Y X $	false ^ false $	B → C
C Y X $	false ^ false $	C → false
false Y X $	false ^ false $	**terminal**
Y X $	^ false $	Y → ^ A
^ A X $	^ false $	**terminal**
A X $	false $	A → B Y
B Y X $	false $	B → C
C Y X $	false $	C → false
false Y X $	false $	**terminal**
Y X $	$	Y → ε
X $	$	X → ε
$	$	**Accept!**

4.) Based upon your grammar for 1), write a recursive descent parser. On the next page, I have provided a skeleton and minimal test suite for you to follow. You do not need to support whitespace and should use T for true and F for false. Fill in the skeleton with the implementation of a function per nonterminal in your grammar. Print out your code (it shouldn't be more than 2 pages) and submit it.

```
class RecursiveDescent {

	public static void main(String[] args) {
		String[] tests= {
			"T",
			"!F",
			"T^F",
			"T^T^",
			"!TvF",
			"!vvF",
			"!F",
			"!T^!T"
		};

		for (String test: tests) {

			if(E(new StringBuffer(test))) {
				System.out.println(test + ": Accept");
			}
			else {
				System.out.println(test + ": Reject");
			}
		}
	}

	//E → A X
	private static boolean E(StringBuffer s) {
		return A(s) && X(s) && s.length()==0;
	}

	//A → B Y
	private static boolean A(StringBuffer s) {
		return B(s) && Y(s);
	}

	//X → v E | ε
	private static boolean X(StringBuffer s) {
		if(s.length()>0 && s.charAt(0)=='v') {
			s.deleteCharAt(0);
			return E(s);
		}
		else {
			return true;
		}
	}
```

```
	//Y → ^ A | ε
	private static boolean Y(StringBuffer s) {
		if(s.length()>0 && s.charAt(0)=='^') {
			s.deleteCharAt(0);
			return A(s);
		}
		else {
			return true;
		}
	}

	//B → ! C | C
	private static boolean B(StringBuffer s) {
		if(s.length()>0 && s.charAt(0)=='!') {
			s.deleteCharAt(0);
			return C(s);
		}
		else {
			return C(s);
		}
	}

	//C → T | F
	private static boolean C(StringBuffer s) {
		if(s.length()>0 && (s.charAt(0)=='T' || s.charAt(0)=='F')) {
			s.deleteCharAt(0);
			return true;
		}
		return false;
	}

}
```