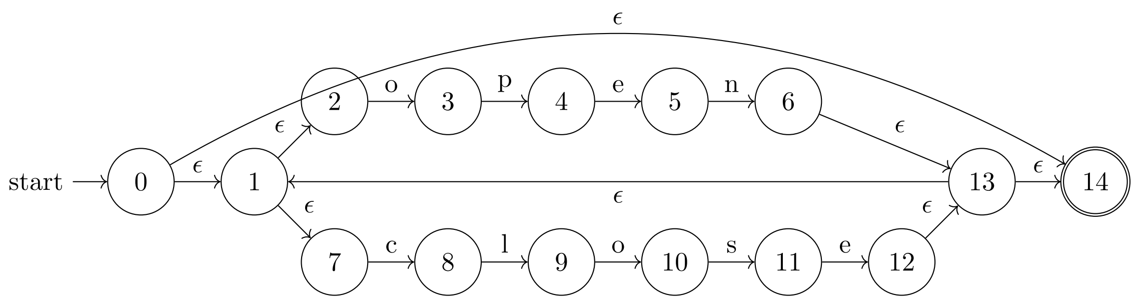
ECE 467 Midterm 1

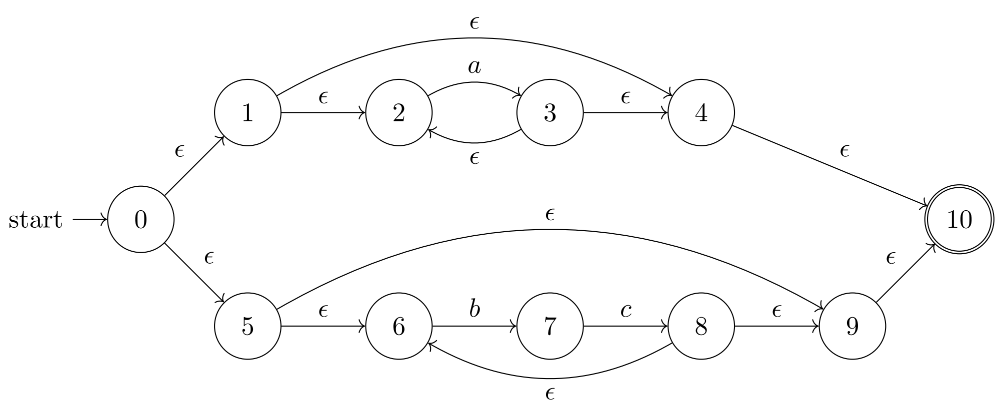
University of Toronto

2022 October 11

1. (3 points) Draw an NFA for **(open|close)*** (where the parentheses indicate the order of operations). Draw states as circles. Label the start state. Indicate final state(s) with a double-circle. See the next question for an example.

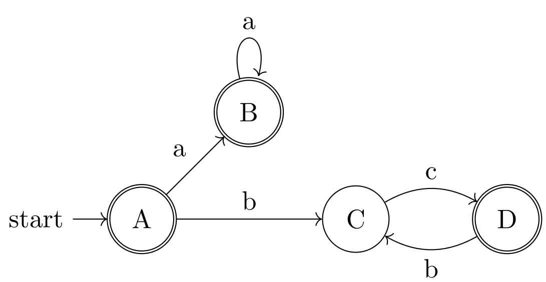


2. (4 points) Draw the DFA for the following NFA. Show your work.



$$
\begin{aligned}
\epsilon\text{-closure}(\{\,0\,\}) &= \{\,0,1,2,4,5,6,9,10\,\} = A \\
\epsilon\text{-closure}(\text{move}(A,a)) &= \epsilon\text{-closure}(\{\,3\,\}) = \{\,3,2,4,10\,\} = B \\
\epsilon\text{-closure}(\text{move}(A,b)) &= \epsilon\text{-closure}(\{\,7\,\}) = \{\,7\,\} = C \\
\epsilon\text{-closure}(\text{move}(A,c)) &= \epsilon\text{-closure}(\{\ \}) = \{\ \} \\
\epsilon\text{-closure}(\text{move}(B,a)) &= \epsilon\text{-closure}(\{\,3\,\}) = \{\,3,2,4,10\,\} = B \\
\epsilon\text{-closure}(\text{move}(B,b)) &= \epsilon\text{-closure}(\{\ \}) = \{\ \} \\
\epsilon\text{-closure}(\text{move}(B,c)) &= \epsilon\text{-closure}(\{\ \}) = \{\ \} \\
\epsilon\text{-closure}(\text{move}(C,a)) &= \epsilon\text{-closure}(\{\ \}) = \{\ \} \\
\epsilon\text{-closure}(\text{move}(C,b)) &= \epsilon\text{-closure}(\{\ \}) = \{\ \} \\
\epsilon\text{-closure}(\text{move}(C,c)) &= \epsilon\text{-closure}(\{\,8\,\}) = \{\,8,6,9,10\,\} = D \\
\epsilon\text{-closure}(\text{move}(D,a)) &= \epsilon\text{-closure}(\{\ \}) = \{\ \} \\
\epsilon\text{-closure}(\text{move}(D,b)) &= \epsilon\text{-closure}(\{\,7\,\}) = \{\,7\,\} = C \\
\epsilon\text{-closure}(\text{move}(D,c)) &= \epsilon\text{-closure}(\{\ \}) = \{\ \}\,.
\end{aligned}
$$

A, B, and D contain accepting NFA states so they are accepting DFA states.



3. (10 points) Consider the following grammar, where E is the start symbol.

$$
\begin{aligned}
&E \rightarrow \text{T C} \\
&T \rightarrow \text{lparen E rparen} \mid \text{id} \\
&C \rightarrow \text{C op T} \mid \epsilon
\end{aligned}
$$

(a) (3 points) Compute the NULLABLE, FIRST, and FOLLOW functions for all nonterminals in the grammar.

Nonterminal	NULLABLE	FIRST	FOLLOW
E	false	lparen, id	$, rparen
T	false	lparen, id	$, rparen, op
C	true	op	$, rparen, op

(b) (5 points) Draw the LR(0) states for the grammar (with labeled arrows between the states).

$$
\begin{aligned}
\text{closure}(\{\, E' \to \cdot E \,\}) =& E' \to \cdot E \\
& E \to \cdot TC \\
& T \to \cdot (E) \\
& T \to \cdot id && = S_0. \\
\text{goto}(S_0, E) =& E' \to E \cdot && = S_1. \\
\text{goto}(S_0, T) =& E \to T \cdot C && = S_2'. \\
\text{goto}(S_0, lparen) =& T \to (\cdot E) && = S_3'. \\
\text{goto}(S_0, id) =& T \to id \cdot && = S_4. \\
\text{closure}(S_2') =& E \to T \cdot C \\
=& C \to \cdot CopT \\
=& C \to \cdot && = S_2. \\
\text{closure}(S_3') =& T \to (\cdot E) \\
& E \to \cdot TC \\
=& T \to \cdot (E) \\
=& T \to \cdot id && = S_3. \\
\text{goto}(S_2, C) =& E \to TC \cdot \\
=& C \to C \cdot opT && = S_5. \\
\text{goto}(S_3, E) =& T \to (E \cdot) && = S_6. \\
\text{goto}(S_3, T) =& E \to T \cdot C && = S_2'. \\
\text{goto}(S_3, lparen) =& T \to (\cdot E) && = S_3'. \\
\text{goto}(S_3, id) =& T \to id \cdot && = S_4. \\
\text{goto}(S_5, op) =& C \to Cop \cdot T && = S_7'. \\
\text{closure}(S_6') =& T \to (\cdot E) \\
=& E \to \cdot TC \\
=& T \to \cdot (E) \\
=& T \to \cdot id && = S_6. \\
\text{closure}(S_7') =& C \to Cop \cdot T \\
=& T \to \cdot (E) \\
=& T \to \cdot id && = S_7. \\
\text{goto}(S_6, rparen) =& T \to (E) \cdot && = S_8. \\
\text{goto}(S_7, T) =& C \to CopT \cdot && = S_9. \\
\text{goto}(S_7, lparen) =& T \to (\cdot E) && = S_3'. \\
\text{goto}(S_7, id) =& T \to id \cdot && = S_4.
\end{aligned}
$$

(c) (2 points) Write the SLR parsing table for the grammar **based on your previous two parts** (you will not lose marks in this part if your previous computations were incorrect).

Note that FOLLOW(E') = FOLLOW(E).

Number the following productions:

0. $E' \to E$
1. $E \to TC$
2. $T \to (E)$
3. $T \to id$

4. $C \rightarrow CopT$
5. $C \rightarrow \epsilon$

State	lparen	rparen	id	op	\$	E'	E	T	C
0	s3		s4				1	2	
1					r0				
2		r5		r5	r5				5
3	s3		s4				6	2	
4		r3		r3	r3				
5		r1		s7	r1				
6		s8							
7	s3		s4					9	
8		r2		r2	r2				
9		r4		r4	r4				

4. (3 points) Based on the course lab project.

   (a) (1 point) You wish to add support for augmented assignment operators (e.g. `+=` and `-=`). The instructor says you don't need to introduce new tokens as augmented assignment operators consist of existing tokens, e.g. `+=` is a `PLUS` token followed by an `ASSIGN` token. What could go wrong?

   The lexer ignores whitespace, so the parser cannot distinguish between `+=` and `+ =`; in both cases, the lexer simply returns `PLUS` followed by `ASSIGN`.

   (b) (2 points) Suppose you have a sorted vector `v` of the offsets of all newline characters in the input buffer (starting from 0). Given an arbitrary offset `k`, you wish to use the binary search function `partition` in the standard library of your favourite language to find which line it is on. Suppose `partition` returns the *smallest* index `i` such that `v[i] >= k`. Given the vector of newline offsets `v`, an input offset `k`, write the code to compute the line and column of `k` using `partition`, assuming lines and columns both start at 1.

   The function `partition` gives us the index of the next newline after (or at) offset `k`. Since a newline character terminates a line, the index of the next newline is the index of the line a character belongs to in a file, counting from 0.

```
index = partition(v, k);
line = index + 1;
if index == 0 {
    // k is before the very first newline character in the file.
    column = k + 1;
} else {
    column = k - v[index - 1];
}
```

5. (8 points) Let a_k be the string of length $2k$ consisting of k left parentheses followed by k right parentheses. Consider the language $L = \{ a_k \mid k \in \mathbb{N} \}$ (i.e. all strings a_k for non-negative integers k). **Suppose** there exists a DFA that defines L; call it D.

   (a) (1 point) Define *reachable* of a state v of D to be true if there exists a string $s_v \in L$ such that the execution of the DFA is at state v at some point during the input of s_v, and false otherwise. Consider the subset V of *reachable* states of D. Argue that V is finite. *(Hint: the argument is very simple.)*

   A DFA has finitely many states. A subset of a finite set is finite.

   (b) (1 point) For each state $v \in V$, choose some string s_v such that the execution of D given s_v (at some point) reaches v. Consider the set S of all such chosen strings s_v (one string for each $v \in V$). What is the most you can say about the size of S?

There is at least one chosen string because there is at least one reachable state in the DFA (the start state). There are at most $|V|$ chosen strings in S, since we may have chosen duplicate strings and sets don't "remember" duplicates.

(c) (3 points) Argue that there exists a string **not in** L that D accepts. *(Hint: let n be the greatest integer such that $a_n \in S$; i.e. a_n is the longest string in S. Consider the execution of D given the string a_{n+1}.)*

D accepts a_{n+1} by assumption, so during the execution of D on a_{n+1}, D is always at some reachable state. Let u be the (reachable) state D is at after matching half of a_{n+1}; i.e. from the start state, transitioning given $n+1$ left parentheses.

Consider the earlier chosen string $s_u \in S$. At some point while matching s_u (upon consuming some prefix of s_u), D reaches state u. Therefore the "rest of" s_u (some suffix of s_u) takes D from state u to an accepting state. Define s'_u to be a suffix of s_u that takes D from state u to an accepting state. Note that any suffix of s_u contains at most n right parentheses, by definition/choice of n.

The string consisting of $n+1$ left parentheses concatenated with s'_u is accepted by D, so $n+1$ left parentheses takes D from the start state to state u, and then s'_u takes D from state u to an accepting state. However, this string is not in L, since it contains $n+1$ left parentheses and at most n right parentheses.

(d) (1 point) Give a definition for a regular language.

A regular language is a set of strings that can be defined by a regular expression.

(e) (2 points) Combine the previous parts to prove that L is not regular.

Suppose we have a DFA D that recognizes L. But by the previous parts, D accepts a string not in L. By contradiction, no such DFA can exist. By the equivalence of DFAs and regular expressions, L is not regular.