1 Lexical Analysis

1.1

Some words in the language described by $ab^*c^*|(ab)^*c$ are a, ab, abc, $abbc$, $ababc$.

1.2

The NFA obtained by applying Thompson's construction is shown on Figure 1.

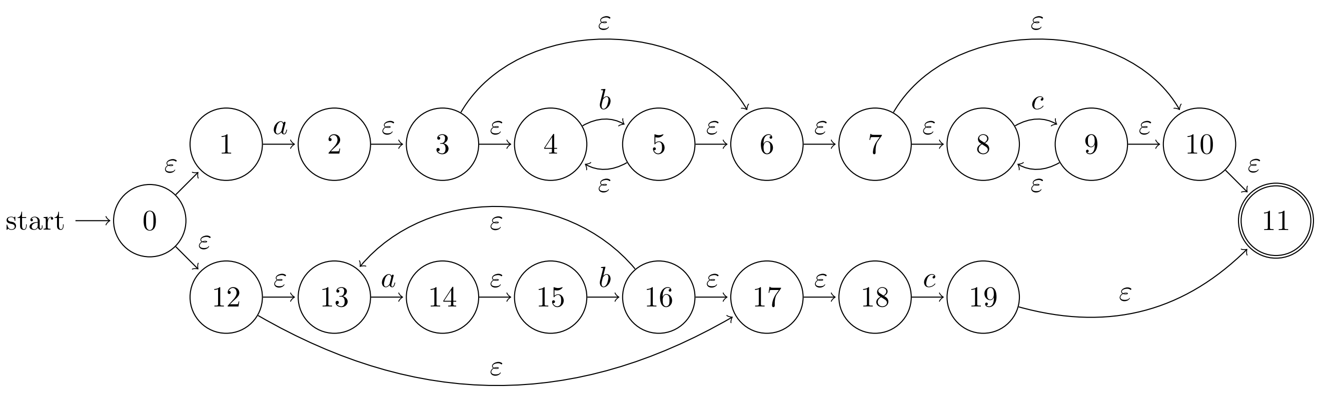

Figure 1: Thompson construction for the regular expression $ab^*c^*|(ab)^*c$.

1.3

The NFA obtained by the Thompson construction can be transformed into a DFA by computing meta-states in the new DFA, with a transition between meta-states indicating that the NFA states corresponding to the target meta-state are reachable by applying the transition to at least on of the NFA states in the origin meta-state. A meta-state is accepting if it contains an accepting state in the NFA. The resulting DFA is given in Figure 2.

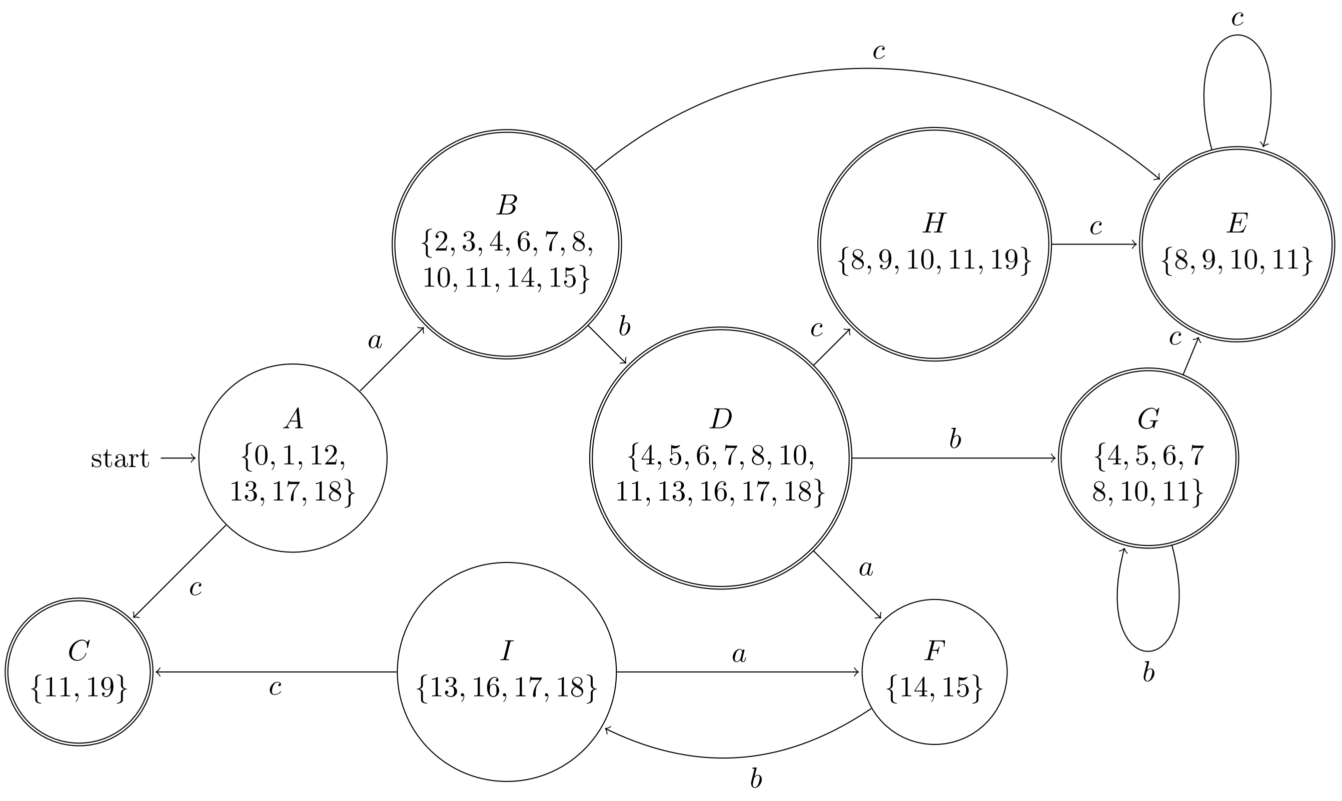

Figure 2: DFA corresponding to the regular expression $ab^*c^*|(ab)^*c$, obtained by computing ε-closures on the NFA obtained by Thompson's construction.

1.4

The minimal DFA obtained with Hopcroft's algorithm is shown on Figure 3. For each partition, the corresponding metastate in the previous DFA is shown.

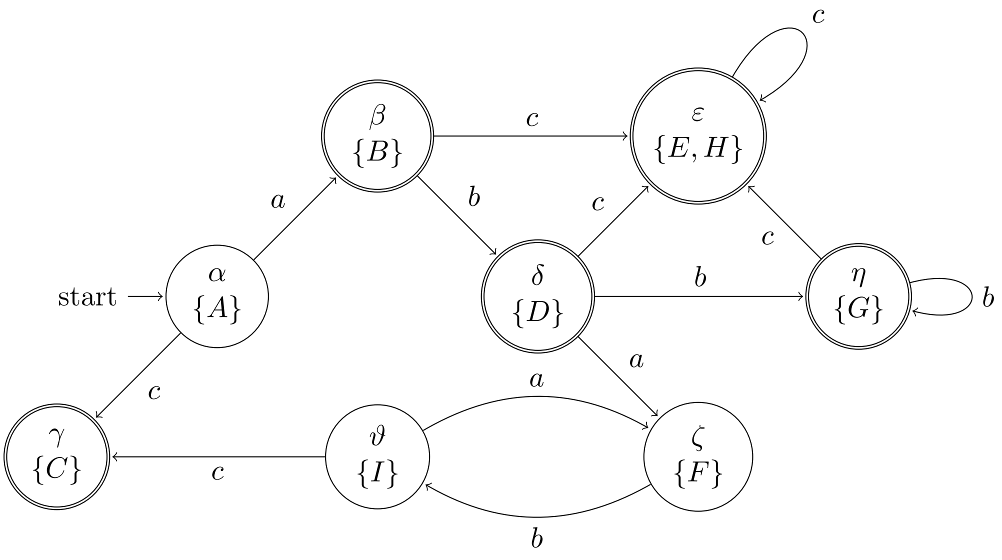

Figure 3: Minimal DFA for the regular expression $ab^*c^*|(ab)^*c$.

1.5

See code.

2 Parsing

2.1

The given grammar is left-recursive, hence it can not be LL(1). An equivalent LL(1) grammar is

E ::= TE'

E' ::= +TE' | −TE' | ε

T ::= FT'

T' ::= ×FT' | /FT' | ε

F ::= (E) | id

2.2

The following First-sets and Follow-sets were computed:

$$\begin{aligned}
\text{first(E)} &= \{(,\text{id}\},\\
\text{first(E')} &= \{+,-,\varepsilon\},\\
\text{first(T)} &= \{(,\text{id}\},\\
\text{first(T')} &= \{\times,/,\varepsilon\},\\
\text{first(F)} &= \{(,\text{id}\},\\
\text{follow(E)} &= \{\#,)\},\\
\text{follow(E')} &= \{\#,)\},\\
\text{follow(T)} &= \{(+,-,\#,)\},\\
\text{follow(T')} &= \{+,-,\#,)\},\\
\text{follow(F)} &= \{+,-,\times,/,\#,)\}.
\end{aligned}$$

The parsing table for the LL(1) grammar is given in Table 1.

	+	−	×	/	(	)	id	#
E					1		1	
E’	2	3				4		4
T					5		5	
T’	8	8	6	7		8		8
F					9		10	

Table 1: Parsing table for the LL(1) grammar given in Section 2.1.

2.3

Table 2 shows the steps in parsing $\text{id} + (\text{id} \times \text{id} - \text{id}/\text{id})$.

Stack	Input	Output
#E	id + (id × id − id/id)#	
#E’T	id + (id × id − id/id)#	1
#E’T’F	id + (id × id − id/id)#	5
#E’T’id	id + (id × id − id/id)#	10
#E’T’	+(id × id − id/id)#	
#E’	+(id × id − id/id)#	8
#E’T+	+(id × id − id/id)#	2
#E’T	(id × id − id/id)#	
#E’T’F	(id × id − id/id)#	5
#E’T’)E(	(id × id − id/id)#	9
#E’T’)E	id × id − id/id)#	
#E’T’)E’T	id × id − id/id)#	1
#E’T’)E’T’F	id × id − id/id)#	5
#E’T’)E’T’id	id × id − id/id)#	10
#E’T’)E’T’	×id − id/id)#	
#E’T’)E’T’F×	×id − id/id)#	6
#E’T’)E’T’F	id − id/id)#	
#E’T’)E’T’id	id − id/id)#	10
#E’T’)E’T’	−id/id)#	
#E’T’)E’	−id/id)#	8
#E’T’)E’T−	−id/id)#	3
#E’T’)E’T	id/id)#	
#E’T’)E’T’F	id/id)#	5
#E’T’)E’T’id	id/id)#	10
#E’T’)E’T’	/id)#	
#E’T’)E’T’F/	/id)#	7
#E’T’)E’T’F	id)#	
#E’T’)E’T’id	id)#	10
#E’T’)E’T’	)#	
#E’T’)E’	)#	8
#E’T’)	)#	4
#E’T’	#	
#E’	#	8
#	#	4

Table 2: Example of parsing $\text{id} + (\text{id} \times \text{id} - \text{id}/\text{id})$.

3 DFA

Yes, a DFA can accept an infinite language. One can prove that a DFA $\mathscr{D}$ with n states accepts an infinite language if and only if $\mathscr{D}$ accepts a string of length k, where $n \leq k < 2n$. One example is the language

$L(a^*)$, which has an infinite number of strings in it, but is accepted by the DFA in Figure 4. Unsurprisingly, the property above is verified for this DFA.

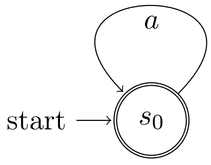

Figure 4: DFA which accepts the infinite language $L(a^*)$.

This is a theoretical result, however. In practice, detecting cycles which contain a start state and a final state is a more efficient way of determining whether a DFA accepts an infinite language. This can be done using depth-first search.

4 Language

This language is not regular.

Proof. Let L be the language of well-balanced parentheses. By contradiction, assume that L is regular. By the pumping lemma, there exists some constant p such that any word w of length at least p in L can be split into $w = xyz$ such that

- $|xy| \leq p$;
- $|y| > 0$;
- $xy^iz \in L$ for all $i \geq 0$.

Take w to be the balanced string consisting of p left parentheses followed by p right parentheses. The first condition of the pumping lemma then gives that y consists only of left parentheses, while the second guarantees y is nonempty. By the third condition, $xyyz$ should also be part of L. However, this is not the case, since by construction $xyyz$ contains more left parentheses than right parentheses. The initial assumption that L is regular must thus be false. □