CS143 Midterm
Spring 2021

- Please read all instructions (including these) carefully.
- There are 5 questions on the exam, some with multiple parts. You have 95 minutes to both finish the exam and upload it. After 95 minutes, gradescope will close your exam and automatically submit it, so make sure submit what you have by then.
- The exam is open note. You may use laptops, phones, e-readers, and the internet, but you may not consult anyone.
- You must upload the answer to each question as an image file or a PDF. You may submit images of hand-written answers taken with your phone (but allow yourself time to send the image to your computer, so you can upload it to gradescope). It is your responsibility, however, to ensure they are legible. Computer typed answers as a PDF are also permitted.
- Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary. Partial solutions will be graded for partial credit.

Problem	Max points	Points
1	15	
2	15	
3	20	
4	20	
5	30	
TOTAL	100	

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

Type your name as a signature below:

1. **Sizes of NFAs and DFAs**

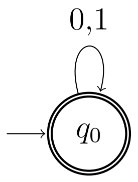
A regular expression can be converted to both an NFA and a DFA. (Remember that NFAs and DFAs must have a starting state and accepting states and that each state of a DFA must have an outgoing transition for every symbol in the alphabet.) Given the alphabet $\{0, 1\}$, provide

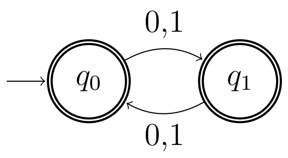
- a regular expression,
- a corresponding NFA, and
- a corresponding DFA

where:

(a) The NFA has fewer states than the DFA.
Answer:

$(0|1)*$

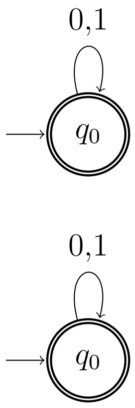




(b) The NFA has the same number of states as the DFA.

Answer:

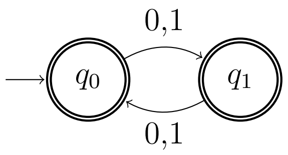
(0|1)∗

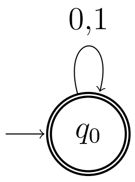


(c) The NFA has more states than the DFA.

Answer:

(0|1)∗





2. **CFG Transformations**

Left factor and remove immediate left recursion in the following grammar:

$$\begin{aligned} S &\rightarrow cA \mid cBe \mid cBd \mid Sc \mid SA \\ A &\rightarrow e \\ B &\rightarrow f \end{aligned}$$

Answer:

Left factored:

$$\begin{aligned} S &\rightarrow cX \mid SY \\ X &\rightarrow BZ \mid A \\ Y &\rightarrow c \mid A \\ Z &\rightarrow e \mid d \\ A &\rightarrow e \\ B &\rightarrow f \end{aligned}$$

With immediate recursion eliminated:

$$\begin{aligned} S &\rightarrow cXS' \\ S' &\rightarrow YS' \mid \epsilon \\ X &\rightarrow BZ \mid A \\ Y &\rightarrow c \mid A \\ Z &\rightarrow e \mid d \\ A &\rightarrow e \\ B &\rightarrow f \end{aligned}$$

3. **Syntax-Directed Translation**

Given the following grammar, add semantic actions that computes the number of sequences of successive a's. For example, both $acab$ and $aacabb$ have two sequences of a's each, while $abacabcb$ have three such sequences.

You must use the following attributes: `int val`, `bool left`, and `bool right`. Use bison syntax: `$i.val` refers to the `val` attribute of the ith symbol of the production and `$$.val` refers to the `val` attribute of the production's result. You may not use global variables.

$$\begin{array}{rcl} S & \rightarrow & TS \\ & \mid & \epsilon \\ T & \rightarrow & aTb \\ & \mid & bTc \\ & \mid & cTa \\ & \mid & \epsilon \end{array}$$

Answer:

```
S -> TS {
  if ($1.right && $2.left) {
    $$.val = $1.val + $2.val - 1;
  } else {
    $$.val = $1.val + $2.val;
  }
  $$.left = $1.left;
  $$.right = $2.right;
}
| epsilon {
  $$.val = 0;
  $$.left = false;
  $$.right = false;
}
T -> aTb {
  $$.val = $2.left ? $2.val : $2.val + 1;
  $$.left = true;
  $$.right = false;
}
| bTc {
  $$.val = $2.val;
  $$.left = false;
  $$.right = false;
}
| cTa {
  $$.val = $2.right ? $2.val : $2.val + 1;
  $$.left = false;
  $$.right = true;
}
| epsilon {
    $$.val = 0;
    $$.left = false;
    $$.right = false;
}
```

4. **Top-Down Parsers**

In this question we will explore top-down parsers and LL(k) grammars.

(a) Construct the LL(1) top-down parser table for the following grammar.

$$\begin{array}{rcl} S & \rightarrow & aSA \\ & | & \epsilon \\ A & \rightarrow & bS \\ & | & c \end{array}$$

Answer:

$$\text{First}(S) = \{a, \epsilon\}$$
$$\text{First}(A) = \{b, c\}$$
$$\text{Follow}(S) = \text{Follow}(A) = \{\$, b, c\}$$

(Since A never goes to epsilon, the Follow sets are not useful in filling the table.)

	a	b	c	\$
S	aSA	ϵ	ϵ	ϵ
A		bS	c	

(b) Show that the following grammar is not LL(1) by pointing out where a conflict occurs when constructing its LL(1) top-down parser table.

$$\begin{array}{rcl} S & \rightarrow & aSA \\ & \mid & \epsilon \\ A & \rightarrow & abS \\ & \mid & c \end{array}$$

Answer:

$$\begin{aligned} \text{First}(S) &= \{a, \epsilon\} \\ \text{First}(A) &= \{a, c\} \\ \text{Follow}(S) = \text{Follow}(A) &= \{\$, a, c\} \end{aligned}$$

Since $a \in \text{First}(aSA)$, the table must have an entry $T[S, a] = aSA$ corresponding to the production $S \rightarrow aSA$. However, as $\epsilon \in \text{First}(\epsilon)$ and $a \in \text{Follow}(S)$, there must also be an entry $T[S, a] = \epsilon$ corresponding to $S \rightarrow \epsilon$.

5. **Bottom-Up Parsers**

Consider the following grammar:

$$\begin{aligned} S &\rightarrow Ae \mid AB \mid dB \\ A &\rightarrow d \mid f \\ B &\rightarrow g \end{aligned}$$

where the terminals are $\{d, e, f, g\}$.

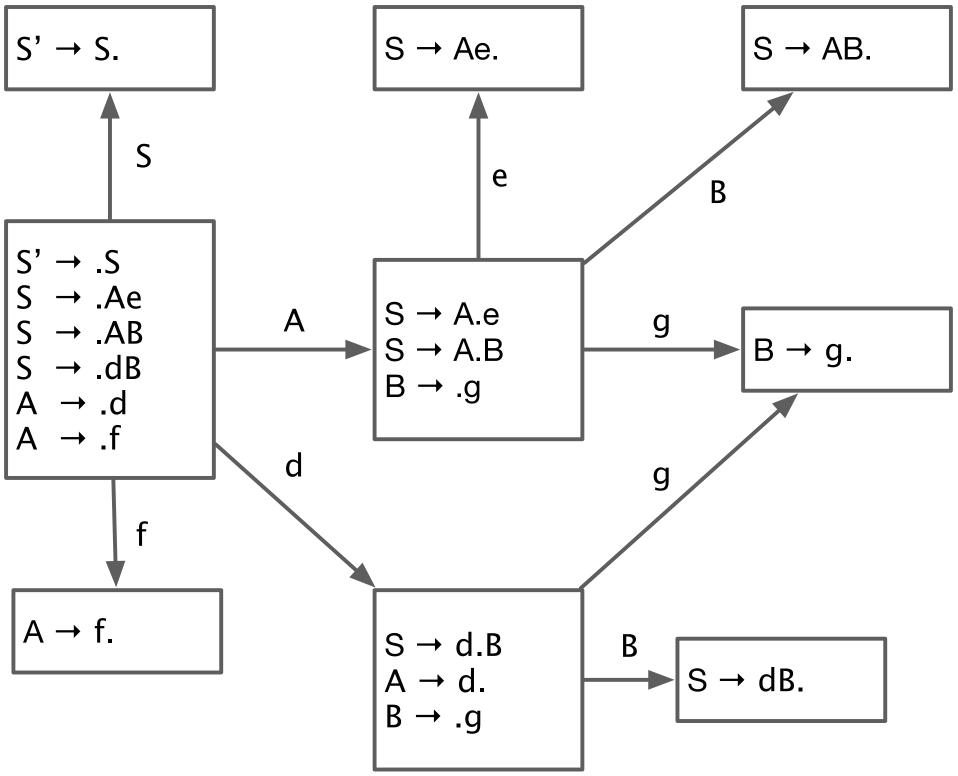
(a) List the first and follow sets of the non-terminals.
Answer:

$$\text{First}(S) = \{d, f\}$$
$$\text{First}(A) = \{d, f\}$$
$$\text{First}(B) = \{g\}$$

$$\text{Follow}(S) = \text{Follow}(B) = \{\$\}$$
$$\text{Follow}(A) = \{e, g\}$$

(b) Draw the LR(0) DFA. (In this question, you need only provide accepting states, including their items, and transitions between accepting states.)

Answer:



(c) Is the grammar SLR(1)? Justify your answer.

Answer:

No, there is a shift-reduce conflict in the state in the middle, between $S \to A.B$ and $B \to .g$, since $g \in \text{Follow}(A)$.