Discussion7 Answer

1 Regular Expressions

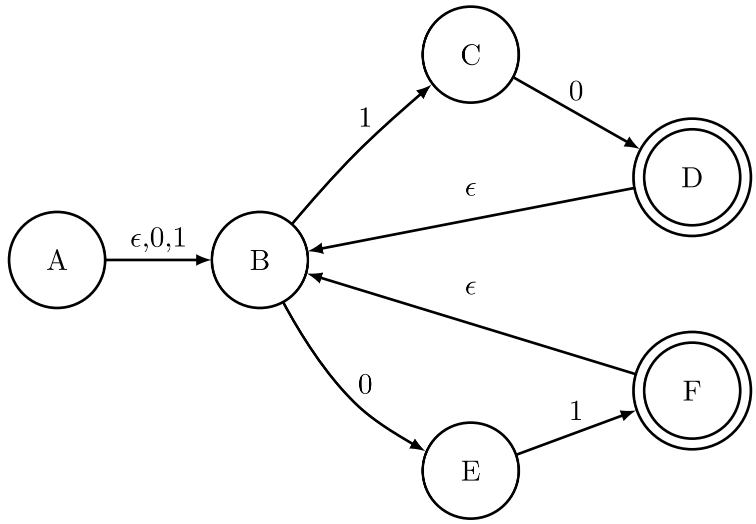
1. (a) Write a regex that matches binary strings divisible by 8.
   Answer: `[01]*000|0|00`

   (b) Provide a *regular* grammar for the regex from part (a).
   Answer:

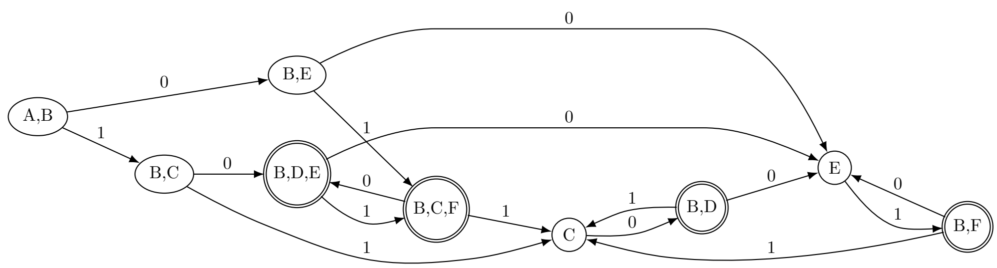
$$
\begin{array}{rcl}
B & : & \text{“0”} \,|\, \text{“1”} \\
P & : & \epsilon \\
  & | & B\ P \\
S & : & P\ \text{“000”} \\
  & | & \text{“0”} \\
  & | & \text{“00”}
\end{array}
$$

2 Finite State Automata

1. (a) Write the corresponding NFA for the regular expression `(0|1)?(10|01)+`;
   Answer:



(b) Convert the NFA from part (a) into a DFA.
Answer:



3 Grammar Rewriting and LL(k) Parsing

Consider the simple ambiguous grammar (which we've seen before):

$$\begin{array}{rcl} S & : & E \dashv \\ E & : & E + E \\ & | & E * E \\ & | & \texttt{ID} \end{array}$$

1. Show that the grammar is ambiguous with two different leftmost derivations of the string `a+b*c`;
   Answer:

   - `S` → `E` → `E * E` → `E + E * E` → `a + E * E` → `a + b * E` → `a + b * c`
   - `S` → `E` → `E + E` → `a + E` → `a + E * E` → `a + b * E` → `a + b * c`

2. Rewrite this grammar so that it preserves the standard order of operations, is LL(1), and is unambiguous. Draw the resulting tree for the string `a+b*c`;
Answer:

$$
\begin{array}{rcl}
S & : & E \dashv \\
E & : & T\,E' \\
E' & : & \epsilon \\
 & | & +\,E \\
T & : & F\,T' \\
T' & : & \epsilon \\
 & | & *\,T \\
F & : & \texttt{ID}
\end{array}
$$

3. Write down the equivalent unambiguous grammar that enforces both **left** associativity and correct precedence. Why can't this be achieved with an LL(1) grammar?
Answer: This can't be achieved with an LL(1) because it introduces left recursion.

$$
\begin{array}{rcl}
S & : & E \dashv \\
E & : & E + T \\
 & | & T \\
T & : & T * F \\
 & | & F \\
F & : & \texttt{ID}
\end{array}
$$

4 Earley's Algorithm

Consider the following grammar:

$$
\begin{array}{rcl}
P & : & E \dashv \\
E & : & ID \\
 & | & \lambda ID\,.\,E \\
 & | & E\,E
\end{array}
$$

1. Use it to parse the following expression with Earley's algorithm: λ`ID . ID ID` $\dashv$
Answer: We can construct two different parse trees.

		λ	ID	.	ID	ID
	0	1	2	3	4	5
a	P:$\bullet$E $\dashv$, 0	E:λ$\bullet$ID.E, 0	E:λID$\bullet$.E, 0	E:λID .$\bullet$E, 0	E:ID$\bullet$, 3	E:ID$\bullet$, 4
b	E:$\bullet$ID, 0			E:$\bullet$ID, 3	E:λID.$E_a \bullet$, 0	E:E $E_a \bullet$, 3
c	E:$\bullet$λ ID.E, 0			E:$\bullet$λ ID.E, 3	E:$E_a \bullet$E, 3	E:E $E_a \bullet$, 0
d	E:$\bullet$E E, 0			E:$\bullet$E E, 3	P:$E_b \bullet \dashv$, 0	E:$E_a \bullet$E, 4
e					E:$E_b \bullet$E, 0	E:λ ID.$E_b \bullet$, 0
f					E:$\bullet$ID, 4	E:$E_b \bullet$E, 3
g					E:$\bullet$λ ID.E, 4	P:$E_c \bullet \dashv$, 0
h					E:$\bullet$E E, 4	E:$E_c \bullet$E, 0
i						P:$E_e \bullet \dashv$, 0
j						E:$E_e \bullet$E, 0

		λ	ID	.	ID	ID
	0	1	2	3	4	5
a	P:$\bullet$E $\dashv$, 0	E:λ$\bullet$ID.E, 0	E:λID$\bullet$.E, 0	E:λID .$\bullet$E, 0	E:ID$\bullet$, 3	E:ID$\bullet$, 4
b	E:$\bullet$ID, 0			E:$\bullet$ID, 3	E:λID.$E_a \bullet$, 0	E:E $E_a \bullet$, 3
c	E:$\bullet$λ ID.E, 0			E:$\bullet$λ ID.E, 3	E:$E_a \bullet$E, 3	E:E $E_a \bullet$, 0
d	E:$\bullet$E E, 0			E:$\bullet$E E, 3	P:$E_b \bullet \dashv$, 0	E:$E_a \bullet$E, 4
e					E:$E_b \bullet$E, 0	E:λ ID.$E_b \bullet$, 0
f					E:$\bullet$ID, 4	E:$E_b \bullet$E, 3
g					E:$\bullet$λ ID.E, 4	P:$E_c \bullet \dashv$, 0
h					E:$\bullet$E E, 4	E:$E_c \bullet$E, 0
i						P:$E_e \bullet \dashv$, 0
j						E:$E_e \bullet$E, 0

2. Is the expression in the language? Is the grammar ambiguous?

   Answer: Yes. Yes.