CS 320
Computer Language Processing
Exercises: Weeks 1 and 2

February 28, 2025

1 Languages and Automata

Exercise 1 Consider the following languages defined by regular expressions:

1. $\{a, ab\}^*$
2. $\{aa\}^* \cup \{aaa\}^*$
3. a^+b^+

and the following languages defined in set-builder notation:

A. $\{w \mid \forall i.0 \leq i \leq |w| \wedge w_{(i)} = b \implies (i > 0 \wedge w_{(i-1)} = a)\}$

B. $\{w \mid \forall i.0 \leq i < |w| - 1 \implies w_{(i)} = b \implies w_{(i+1)} = a\}$

C. $\{w \mid \exists i.0 < i < |w| \wedge w_{(i)} = b \wedge w_{(i-1)} = a\}$

D. $\{w \mid (|w| = 0 \mod 2 \vee |w| = 0 \mod 3) \wedge \forall i.0 \leq i < |w| \implies w_{(i)} = a\}$

E. $\{w \mid \forall i.0 \leq i < |w| - 1 \wedge w_{(i)} = a \implies w_{(i+1)} = b\}$

F. $\{w \mid \exists i.0 < i < |w| - 1 \wedge (\forall y.0 \leq y \leq i \implies w_{(y)} = a) \wedge (\forall y.i < y < |w| \implies w_{(y)} = b)\}$

For each pair (e.g. 1-A), check whether the two languages are equal, providing a proof if they are, and a counterexample word that is in one but not the other if unequal.

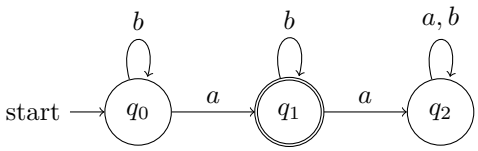
Exercise 2 For each the following languages, construct an NFA $\mathcal{A}$ that recognizes them, i.e. $L(\mathcal{A}) = L_i$:

1. L_1: binary strings divisible by 3
2. L_2: binary strings divisible by 4
3. L_3: $\{(w_1 \oplus w_2) \mid w_1 \in L_1 \wedge w_2 \in L_2 \wedge |w_1| = |w_2|\}$

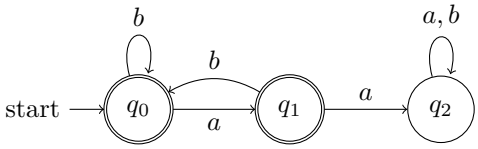
where $\oplus$ is the bitwise-xor operation on binary strings.

Exercise 3 Give a verbal and a set-notational description of the language accepted by each of the following automata. You can assume that the alphabet is $\Sigma = \{a, b\}$.

1. $\mathcal{A}_1$



2. $\mathcal{A}_2$



2 Lexing

Consider a simple arithmetic language that allows you to compute one arithmetic expression, construct conditionals, and let-bind expressions. An example program is:

```
let x = 3 in
let y = ite (x > 0) (x * x) 0 in
  (2 * x) + y
```

The lexer for this language must recognize the following tokens:

`keyword` :	`let` | `in` | `ite`
`op` :	`+` | `-` | `*` | `/`
`comp` :	`>` | `<` | `==` | `<=` | `>=`
`equal` :	`=`
`lparen` :	`(`
`rparen` :	`)`
`id` :	$letter \cdot (letter \mid digit)^*$
`number` :	$digit^+$
`skip` :	`whitespace`

For simplicity, *letter* is a shorthand for the set of all English lowercase letters $\{a - z\}$ and *digit* is a shorthand for the set of all decimal digits $\{0 - 9\}$.

Exercise 4 For each of the tokens above, construct an NFA that recognizes strings matching its regular expression.

A lexer is constructed by combining the NFAs for each of the tokens in parallel, assuming maximum munch. The resulting token is the first NFA in the token order that accepts a prefix of the string. Thus, tokens listed first have higher priority. We then continue lexing the remaining string. You may assume that the lexer drops any `skip` tokens.

Exercise 5 For each of the following strings, write down the sequence of tokens that would be produced by the constructed lexer, if it succeeds.

1. `let x = 5 in x + 3`
2. `let5x2`
3. `xin`
4. `==>`
5. `<===><==`

Exercise 6 Construct a string that would be lexed differently if we ran the NFAs in parallel and instead of using token priority, simply picked the longest match.