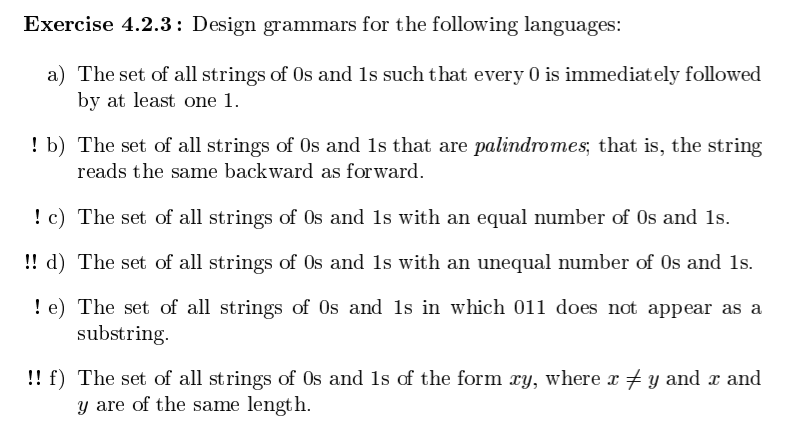
4.2.3,



a, The set of all strings of 0’s and 1’s such that every 0 is immediately followed by at least one 1.

grammar in the same format as those given for the labs.

Assuming that the null string is in the language.

$

1

0

$

S -> A

S -> 01A

A -> S

A -> 1A

A -> e

e = epsilon

b, The set of all strings of 0s and 1s that are palindromes, that is, strings that read the same backwards and forwards.

Assuming that the null string is in the language.

$

1

0

$

S-> 0

S-> 1

S-> e

S -> 1S1

S -> 0S0

c, The set of all strings of 0s and 1s with an equal number of 0s and 1s.

Assuming that the null string is in the language.

$

1

0

$

S -> 1S0

S -> 0S1

S -> SS

S -> e

Best way to do this is to construct PDA, construct grammar from PDA, and then simplify.

d, The set of all strings with 0s and 1s with unequal numbers of 0s and 1s.

Construct 2 DFAs, each producing either more 1s than 0s or more 0s than ones, then combine into one DFA, then convert to grammar.

We already have c, so we can just forcibly add more 1s or more 0s to get the grammars for this.

Grammar for more 1s than 0s:

$

0

1

$

S -> A1S

S -> A1A

A -> 1A0

A -> 0A1

A -> AA

A -> e

Note that A is just the starting state of the same number as 0s and 1s grammar. S guarantees we have at least one more 1 than 0.

Similarly, to construct more 0s than 1s.

$

0

1

$

S -> A0S

S -> A0A

A -> 1A0

A -> 0A1

A -> AA

A -> e

Now we simply add a new start state that goes to either S0 or S1, and convert the starting states from before to S0 and S1.

Grammar:

$

0

1

$

S -> [S0]

S -> [S1]

[S1] -> A1[S1]

[S1] -> A1A

[S0] -> A0[S0]

[S0] -> A0A

A -> 1A0

A -> 0A1

A -> AA

A -> e

Main issue here is that I still can’t quite figure out how to construct c from a PDA, since that seems a bit complicated. Building on c to construct d is simple, though.

e, The set of all strings of 0s and 1s in which 011 does not appear as a substring.

First, construct DFA to accept this string.

{0:{1:0, 0:A},

A:{1:B, 0:A},

B:{0:1} #1 crashes this, so no move on 1

}

Convert DFA to grammar.

$

1

0

$

S -> 1S

S -> 0A

A -> 0A

A -> 1B

B -> 0A

f, The set of all strings of 0s and 1s in the form xy, where x=/=y and x and y are of the same length.

This isn’t a CFG. (This is the inverse of constructing the string xx, which also isn’t a CFG. If a CFG can handle xx it can handle xy.)

To handle xx the program would need to remember the entire string, but a single stack can only remember information in reverse order, to turn xx^reverse into xx needs another stack, which a CFG doesn’t have access to, because it’s equivalent to a push down automata, which only has a single stack.

Since xx can’t be handled, neither can xy where |x| = |y| and x != y.