Løsningforslag for obligatorisk innlevering 1 – INF2820

February 27, 2017

Oppgave 1

a)

$$LM = \{abbba, aba, abbbbba, abcbba, abca, cbba, ca\}$$

b)

$$ML = \{bbaab, bbaabbb, bbaabc, bbac, aab, aabbb, aabc, ac\}$$

c)

$$\{ac, ab, aaabbb\} \subseteq M^*L$$

d)

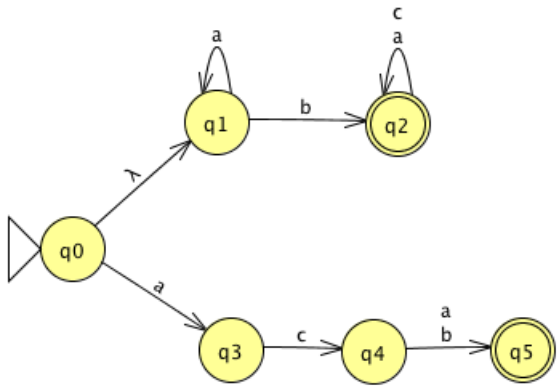
$$\{\epsilon, ac\} \subseteq (ML)^*$$

e)

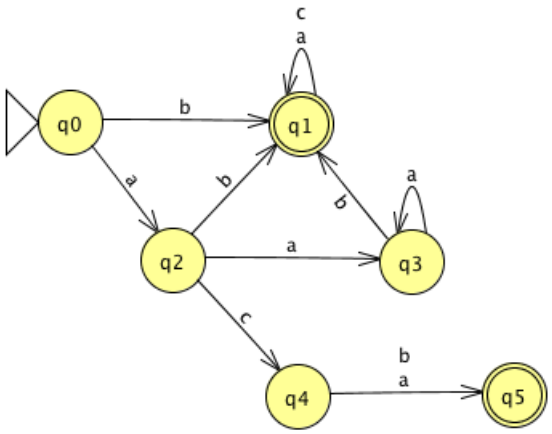
$$LN = \emptyset$$
$$LP = L$$

Oppgave 2

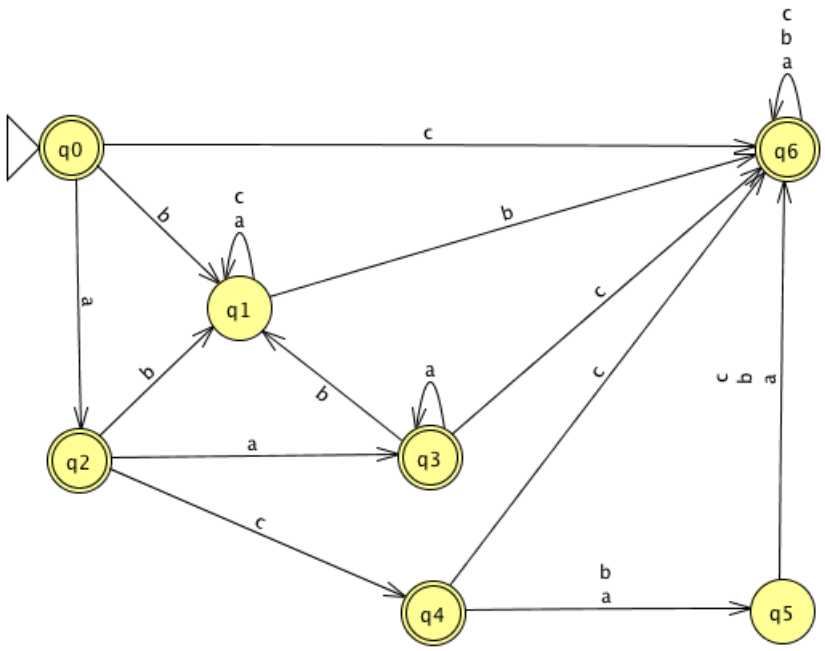
a)



b)



c)



Merk at vi har lagt til en trap-state før vi finner komplementet. Vi tegner vanligvis ikke denne tilstanden fordi det kompliserer diagrammet unødig, men antar heller at den er tilstede.

d)

$$\{abc, acb, bac, aaab, aaba, abaa, baaa\} \subseteq L1$$

Oppgave 3

a)

$$(a+b+c)^*bbb(a+b+c)^*$$

b)

$$(a+c)^*((bb+b)(a+c)(a+c)^*)^*(bb+b+\epsilon)$$

c)

$$((a+c)^*(b(a+c)^*b(a+c)^*b(a+c)^*)^*) + ((c+b)^*(a(c+b)^*a(c+b)^*)^*)$$

Fordi 0 er delelig på et hvert tall så skal også strenger hvor antall a'er eller b'er er 0 også være med.

Oppgave 4

```
import re

def beskriver(regeksp, ord):
    reg = "^("+regeksp+")$"
    if re.search(reg, ord):
        return True
    else:
        return False

def test(regex, correct, incorrect):
    print("Testing '{}'".format(regex))
    for word in correct:
        if not beskriver(regex, word):
            print("Should have been recognized: '{}'".format(word))

    for word in incorrect:
        if beskriver(regex, word):
            print("Should not have been recognized: '{}'".format(word))


# 4a
regex_a = "(a|b|c)*bbb(a|b|c)*"
correct_a = ("bbb", "cbabbbacba")
incorrect_a = ("b", "bbacabbca")

test(regex_a, correct_a, incorrect_a)

# 4b
regex_b = "(a|c)*((bb|b)(a|c)(a|c)*)*(bb|b|)"
correct_b = ("abba", "abcabcbb")
incorrect_b = ("bbb", "babcabbcbbb")

test(regex_b, correct_b, incorrect_b)

# 4c
regex_c = "((a|c)*(b(a|c)*b(a|c)*b(a|c)*)*)|((c|b)*(a(c|b)*a(c|b)*)*)"
correct_c = ("", "c", "a", "b", "aabccbbaabbcbacabcbcbccaabbb")
incorrect_c = ("abaab", "aabba")

test(regex_c, correct_c, incorrect_c)
```

Oppgave 5

a)

```
>>> pyt_raw = open("Python_INF2820_v2017.txt").read()
```

b)

```
>>> pyt_words1 = pyt_raw.split()
>>> pyt_words2 = nltk.word_tokenize(pyt_raw)
```

Når vi bruker split() så blir tegnsetting hengende fast i ordene og og vi får strenger som "INF2820,". Derimot når vi bruker tokenisering så blir tegnsetting skilt ut i egen tokens og vi får da "INF2820" og ",". Noen ganger skiller tokeniseringen for mye, f.eks. vil "'http://www.uio.no"' bli tokenisert til ["http", ":", "//www.uio.no"].

c)

```
>>> pyt_low = [w.lower() for w in pyt_words2]
```

d)

```
>>> forekomster = [w for w in pyt_low if re.search("[æøå]", w)]
>>> print(len(forekomster))
60
```

e)

```
>>> print(len(set(forekomster)))
23
```

f)

```
with open("norske.txt", "w") as f:
    for w in set(forekomster):
        f.write("{}\n".format(w))
```

Oppgave 6

a)

Her tar vi forbehold om at vi kun er ute etter stier som starter med "/" og at det hver "/" må være adskilt med i det minste ett symbol. For enkelthets skyld antar vi også at en sti ikke kan inneholde mellomrom.

```
  path_regex = "(/[^/ ]+)+/?"
```

b)

```
def replace_tokens(tokens):
    path_regex = "^(/[^/ \n]+)+/?$"
    num_regex = "^\d+(\.\d+)?$"

    replaced = []

    for token in tokens:
```

```
        if re.search(path_regex, token):
            replaced.append("<path>")
        elif re.search(num_regex, token):
            replaced.append("<num>")
        else:
            replaced.append(token)

    return replaced
```

c)

```
def replace_string(text):
    path_regex = "(/[^/ \n]+)+/?"
    num_regex = "\d+(\.\d+)?"

    text = re.sub(path_regex, "<path>", text)
    text = re.sub(num_regex, "<num>", text)

    return text
```