Joel Christiansen
Tuesday 4:35

(1)

(a)

Let L1 = {0^n1^n2^n | n>= 0}
L2 = {0^i1^j2^k | i,j,k >=0}
L3 = {0^i1^i2^j | i,j >=0}
L4 = {0^i1^j2^j | i,j >=0}
Then L1_comp = L2_comp union L3_comp union L4_comp
L2, L3, L4 are DCFLs
L1 is not a context free language.
L2_comp, L3_comp, L4_comp are DCFLs
L1_comp is not a DCFL
So DCFLs are not closed under union.
by DeMorgan's law, DCFLs are not closed under intersections

(b) Let L be described by the PDA (Q1, Σ, Γ1, δ1, q01, F1)
R be described by the DFA (Q2, Σ, δ2, q02, F2)
Then LxR is (Q1xQ2, Σ, Γ, δ3, (q01, q02), F1xF2)
where δ3 = {
(qa1, qb2), α, β -> , γ, (qc1, qd2) where α in Σ
β, γ in Γ
(qa1, qb1) in Q1xQ2
(qc2, qd2) in Q1xQ2
}
Then LxR is a pda, and LxR is a PDA, so L(LxR) is a CFL

(2) M = <Q, Σ, Γ, δ, q_0, q_a, q_r>

Q = {q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_a, q_r}
Σ = {1,0}*
Γ = {1,0,#,_, x}
δ = {
(q_0, 1, q_1, x, R) #start states
(q_0, 0, q_4, x, R)
(q_0, _, q_r, _, R) # input doesn't match description
(q_0, x, q_r, x, R) # ^^^
(q_0, #, q_6, #, R) # time to check if x and y are empty

(q_1, 1, q_1, 1, R) #scanning to the first of x or y in order to check for 1
(q_1, 0, q_1, 0, R)
(q_1, #, q_2, #, R) #found x or y
(q_1, _, q_1, _, R)
(q_1, x, q_r, x, R) # shouldn't ever happen, including for completeness

(q_2, _, q_2, _, R) # searching for a 1 in x or y
(q_2, 0, q_r, 0, R) #rejects if it finds a 0
(q_2, 1, q_3, x, L) #found 1, goes to scan back state
(q_2, #, q_2, x, R) #x is empty, moves to y
(q_2, x, q_r, x, R) # shouldn't ever happen, including for completeness

Joel Christiansen
Tuesday 4:35

```
(q_3, 1, q_3, 1, L)       # scanning back to the next thing to check in w
(q_3, 0, q_3, 0, L)
(q_3, _, q_3, _, L)
(q_3, x, q_0, x, R)
(q_3, #, q_3, #, L)

(q_4, 1, q_4, 1, R)       #scanning to the first of x or y in order to check for 0
(q_4, 0, q_4, 0, R)
(q_4, #, q_5, #, R)         #found x or y
(q_4, _, q_4, _, R)
(q_4, x, q_r, x, R)          # shouldn't ever happen, including for completeness

(q_5, _, q_5, _, R)       # searching for a 0 in x or y
(q_5, 0, q_3, 0, R)          #rejects if it finds a 0
(q_5, 1, q_r, x, L)         #found 0, goes to scan back state
(q_5, #, q_5, x, R)          #x is empty, moves to y
(q_5, x, q_r, x, R)           # shouldn't ever happen, including for completeness

(q_6, #, q_6, #, R)       #checking if x and y are done
(q_6, x, q_6, x, R)
(q_6, 1, q_r, x, R)
(q_6, 0, q_r, x, R)
(q_6, _, q_a, _, R)
}
```

(3)

1) Shift everything to the right, put a # in front.
2) for i > 0:
shift everything to the right 1, generate a string w of length i in front of the first #. use (2) to determine if w = x+y. If it does, accept. else, generate the next string of length i. if there are no more strings of length i, move to i+1.

(4)

Start with "a#b#c#w" on the input string, where a=1, b=1, c=0. If a = w, accept. Otherwise, copy c to b, copy a to c, use (3) to generate a = b+c, and check if a = w again. Repeat until a > w, at which point reject.