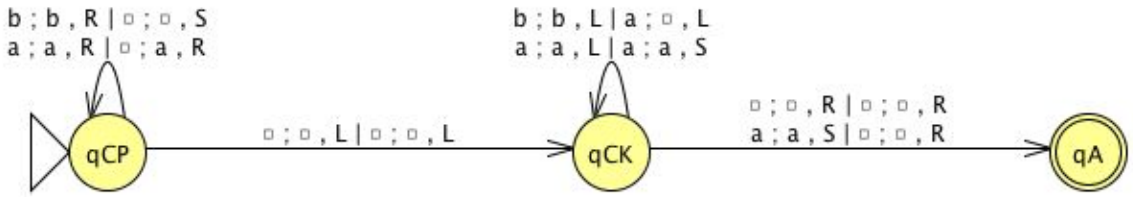
Chris Fenton (fenwil28)
CNC - Formal Languages
Winter Final

1. Construct a two-tape Turing machine with input alphabet {a, b} that accepts strings with the same number of a's and b's. The computation with input u should take no more than 2 ∗ length(u) + 3 transitions.



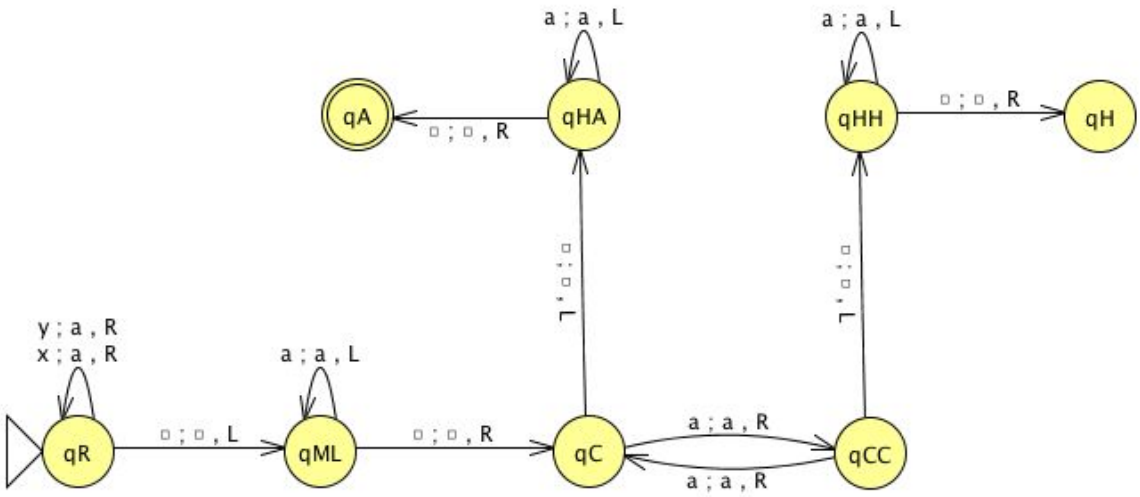
M = (Q, Σ, Γ, δ, q0, F)
q0 = qCopy
Q = {qCopy, qCheck, qAccept}
Γ = {a,b,B}
Σ = {a,b}
F = qAccept

δ(qCopy, a, B) = [qCopy; a, R; a, R]
δ(qCopy, b, B) = [qCopy; b, R; B, S]
δ(qCopy, B, B) = [qCheck; B, L; B, L]
δ(qCheck, a, a) = [qCheck; a, L; a, S]
δ(qCheck, b, a) = [qCheck; b, L; B, L]
δ(qCheck, a, B) = [qAccept; a, S; B, R]
δ(qCheck, B, B) = [qAccept; B, R; B, R]

Note: the little squares in the diagram should be blanks. My version of FLAP is either missing the right font or it uses the little box for blanks. I changed this manually in later diagrams (although the machine doesn't run in JFLAP when you do this).

2. Construct a Turing machine that reduces the language L to Q. In each case the alphabet of L is {x, y} and the alphabet of Q is {a, b}:

(a) L = (xy)* and Q = (aa)*

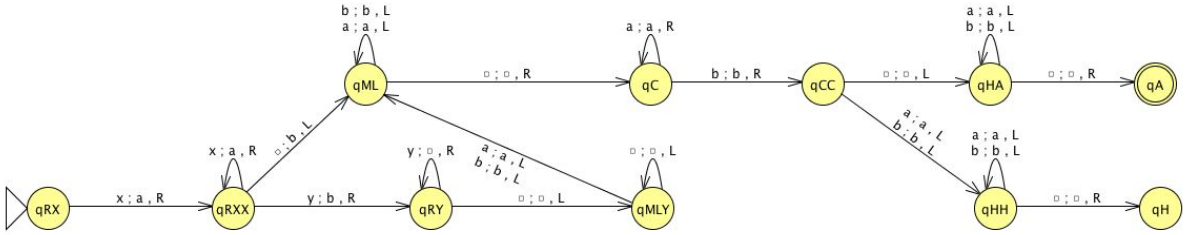


M = (Q, Σ, Γ, δ, q0, F)
q0 = qReplace
Q = {qReplace, qML, qCheck, qCheck2, qHA, qHH, qAccept, qHaltReject}
Γ = {x,y,a,B}
Σ = {x,y}
F = qAccept

δ(qReplace, x) = [qReplace; a, R]
δ(qReplace, y) = [qReplace; a, R]
δ(qReplace, B) = [qML; B, L]
δ(qML, a) = [qML; a, L]
δ(qML, B) = [qCheck; B, R]
δ(qCheck, a) = [qCheck2; a, R]
δ(qCheck, B) = [qHA; B, L]
δ(qCheck2, a) = [qCheck; a, R]
δ(qCheck2, B) = [qHH; B, L]
δ(qHA, a) = [qHA; a, L]
δ(qHA, B) = [qAccept; B, R]
δ(qHH, a) = [qHH; a, L]
δ(qHH, B) = [qHaltReject; B, R]

Note: I didn't fully grasp reduction when I did these. I thought about going back and fixing these so they just did the reduction (R machine) so they were ready to feed into the decider to say if they were in the language (these do the reduction, but maybe missing some edge cases and they use accepting/rejecting states when they should just halt). I don't have time to do this, but I did a more standard reduction in question 12.

(b) L = x^+y^* and Q = a^+b



M_q = (Q, Σ, Γ, δ, q0, F)
q0 = qRX
Q = {qRX, qRXX, qRY, qML, qMLY, qC, qCC, qHA, qHH, qA, qH}
Γ = {x,y,a,b,B}
Σ = {x,y}
F = qA

δ(qRX, x) = [qRXX; a, R]
δ(qRXX, x) = [qRXX; a, R]
δ(qRXX, y) = [qRY; b, R]
δ(qRXX, B) = [qML; b, L]
δ(qRY, y) = [qRY; B, R]
δ(qRY, B) = [qMLY; B, L]
δ(qMLY, B) = [qMLY; B, L]
δ(qMLY, a) = [qML; a, L]
δ(qMLY, b) = [qML; b, L]
δ(qML, a) = [qML; a, L]
δ(qML, b) = [qML; b, L]
δ(qML, B) = [qC; B, R]
δ(qC, a) = [qC; a, R]
δ(qC, b) = [qCC; b, R]
δ(qCC, B) = [qHA; B, L]
δ(qCC, b) = [qHH; b, L]
δ(qCC, a) = [qHH; a, L]
δ(qHA, b) = [qHA; b, L]
δ(qHA, a) = [qHA; a, L]
δ(qHA, B) = [qA; B, R]
δ(qHH, b) = [qHH; b, L]
δ(qHH, a) = [qHH; a, L]
δ(qHH, B) = [qH; B, R]

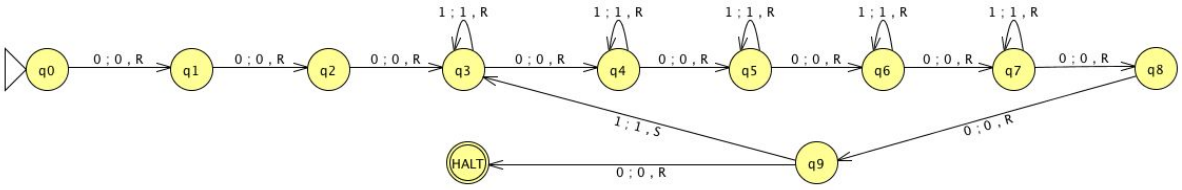
(c) $L = \{x^i y^j x^i \mid i \geq 0\}$ and $Q = a^i b^i \mid i \geq 0\}$

$M_q = (Q, \Sigma, \Gamma, \delta, q0, F)$
$q0 = qRX$
$Q = \{qRX, qRY, qDX, qMLL, qML, CP0, CPH, PA, PAA, PB, PBB, MH, CPH, CPR, qC, qCA, qCH, qHA, qHH, qHMH, qHMA, qA, qH\}$
$\Gamma = \{x,y,a,b,B,\_\}$
$\Sigma = \{x,y\}$
$F = qA$

$\delta(qRX, x) = [qRX; a, R]$
$\delta(qRX, y) = [qRY; b, R]$
$\delta(qRY, y) = [qRY; b, R]$
$\delta(qRY, x) = [qDX; \_, R]$
$\delta(qDX, x) = [qDX; \_, R]$
$\delta(qDX, \_) = [qMLL; \_, L]$
$\delta(qMLL, \_) = [qMLL; \_, L]$
$\delta(qMLL, b) = [qML; b, L]$
$\delta(qML, b) = [qML; b, L]$
$\delta(qML, a) = [qML; a, L]$
$\delta(qML, \_) = [CP0; \_, R]$
$\delta(CP0, a) = [PA; A, R]$
$\delta(CP0, b) = [PB; B, R]$
$\delta(CP0, \_) = [CPH; \_, L]$
$\delta(PA, a) = [PA; a, R]$
$\delta(PA, b) = [PA; b, R]$
$\delta(PA, \_) = [PAA; \_, R]$
$\delta(PAA, a) = [PAA; a, R]$
$\delta(PAA, b) = [PAA; b, R]$
$\delta(PAA, \_) = [MH; a, L]$
$\delta(PB, a) = [PB; a, R]$
$\delta(PB, b) = [PB; b, R]$
$\delta(PB, \_) = [PBB; \_, R]$
$\delta(PBB, a) = [PBB; a, R]$
$\delta(PBB, b) = [PBB; b, R]$
$\delta(PBB, \_) = [MH; b, L]$
$\delta(MH, a) = [MH; a, L]$
$\delta(MH, b) = [MH; b, L]$
$\delta(MH, \_) = [MH; \_, L]$
$\delta(MH, A) = [CP0; A, R]$
$\delta(MH, B) = [CP0; B, R]$
$\delta(CPH, A) = [CPH; A, L]$
$\delta(CPH, B) = [CPH; B, L]$
$\delta(CPH, \_) = [CPR; \_, R]$
$\delta(CPR, A) = [CPR; a, R]$
$\delta(CPR, B) = [CPR; b, R]$
$\delta(CPR, \_) = [qC; \_, R]$
$\delta(qC, a) = [qCA; x, R]$
$\delta(qC, x) = [qC; x, R]$
$\delta(qC, \_) = [qHA; \_, L]$
$\delta(qCA, a) = [qCA; a, R]$
$\delta(qCA, x) = [qCA; x, R]$
$\delta(qCA, b) = [qCH; x, L]$
$\delta(qCA, \_) = [qHH; \_, L]$
$\delta(qCH, x) = [qCH; x, L]$
$\delta(qCH, a) = [qCH; a, L]$
$\delta(qCH, b) = [qCH; b, L]$
$\delta(qCH, \_) = [qC; \_, R]$
$\delta(qHH, x) = [qHH; \_, L]$
$\delta(qHH, \_) = [qHMH; \_, L]$
$\delta(qHMH, b) = [aHMH; b, L]$
$\delta(qHMH, a) = [qHMH; a, L]$
$\delta(qHMH, \_) = [qH; \_, R]$
$\delta(qHA, x) = [qHA; \_, L]$
$\delta(qHA, \_) = [qHMA; \_, L]$
$\delta(qHMA, b) = [qHMA; b, L]$
$\delta(qHMA, a) = [qHMA; a, L]$
$\delta(qHMA, \_) = [qA; \_, R]$

3. Construct a Turing machine that decides whether a string over {0, 1}* is the encoding of a nondeterministic Turing machine. What would be required to change this to a machine that decides whether the input is the representation of a deterministic Turing machine.

Test u = 0001011101101110110011010101010011011011101101100111011010110100 0



M_u = (Q, Σ, Γ, δ, q0, F)
q0 = q0
Q = {q0 - q9, F}
Γ = {0,1,B}
Σ = {0,1}
F = HALT

δ(q0, 0) = [q1; 0, R]
δ(q1, 0) = [q2; 0, R]
δ(q2, 0) = [q3; 0, R]
δ(q3, 1) = [q3; 1, R]
δ(q3, 0) = [q4; 0, R]
δ(q4, 1) = [q4; 1, R]
δ(q4, 0) = [q5; 0, R]
δ(q5, 1) = [q5; 1, R]
δ(q5, 0) = [q6; 0, R]
δ(q6, 1) = [q6; 1, R]
δ(q6, 0) = [q7; 0, R]
δ(q7, 1) = [q7; 1, R]
δ(q7, 0) = [q8; 0, R]
δ(q8, 0) = [q9; 0, R]
δ(q9, 1) = [q3; 1, S]
δ(q9, 0) = [HALT; 0, R]

4. The universal machine introduced in Section 11.5 was designed to simulate the actions of Turing machines that accepts by halting. Consequently, the representation scheme R(M) did not encode accepting states.

(a) Extend the representation of R(M) of a Turing machine M to explicitly encode the accepting states of M.

We could encode the accepting states in a similar way that the transition rules are encoded. We could add these to the beginning of R(M) in an encoded version and read them to the tape or just pass them in as part of the input like we do with the input string w.

R(M) encoding with accepting states:
The prefix and postfix get padded with 1 more zero: “0000”
The the separator between transitions gets padded with 1 more zero: “000”
The separator between accepting states and transitions becomes: “00”
The separator between elements of a transition stay the same: “0”
The separator between elements of the accepting states also gets a single zero: “0”

R(M) becomes:

“Prefix” + “accepting states” + “transitions” + “postfix” where accepting states are encoded in the following fashion:

“q_{halt1}” + 0 + “q_{halt2}” + 0 + … + “q_{haltn}”

Example with accepting states $\{q_2, q_3\}$:

0001110111100...

The example R(M) from the book with making q_2 an accepting state:

000011100101110110111011000110101010100011011011101101100011101101011010000

(b) Design a universal machine U_f that accepts input of the form R(M)w if the machine M accepts input w by final state.

We use one more tape (tape 4). We then add 2 steps to the algorithm on page 356 between steps 3 and 4. This new steps would be:

1. Copy the accepting states from tape 1 to tape 4 (we could leave them on tape 1 or erase them)
2. Scan the accepting states on tape 4
    a. If the en(q_n) on tape two is contained in the halting states
        i. Replace en(q_n) with the halting state en(q_{h^*}) on tape 2 and halt
    b. If the en(q_n) on tape two isn’t contained in the halting states on the new tape
        i. Rewind the new tape
        ii. Move to step 4 in the algorithm

5. Explain the fundamental difference between the ”Halts on the n’th transition problem” from Example 11.5.2 and the ”Halting Problem” that makes the former decidable and the latter undecidable.

The key difference between the n’th transition problem and the halting problem is that the n’th transition problem is that it is specific to a particular machine M and input w; however, the halting problem is asking if there is an algorithm to test all possible machines and inputs for decidability. We can answer the decidability question for a particular machine and input but the more general question for any given machine and input.

6. Prove that there is no algorithm that determines whether an arbitrary Turing machine prints the symbol 1 on three consecutive transitions when run with a blank tape.

- Let’s call our arbitrary machine that prints the symbol 1 on three consecutive transitions when run with a blank tape M_a.
- The language that accepts M_a with an input w is the language A.
- Let’s make a machine D_{aw} that accepts the input $<M_a, w>$ and assume it halts w.
    - This machine D_{aw} would be a decider that would put ‘yes’ on the tape if $<M_a,w>$ is accepted or rejected by the language A and ‘no’ on the tape if there is a loop.
- We can successfully decide if the input is accepted or rejected, but we can’ decide there is a loop.

7. Use Rice’s theorem to show that the following properties of recursively enumerable languages are undecidable:

(a) L(M) = Σ*

Machine P returns true if Σ* is true for a given machine
Machine P returns false if Σ* is false for a given machine
Machine P_{true} returns true if if a given machine returns true
Machine P_{false} returns false if a given machine returns false
Where $P(P_{false})$ = false and $P(P_{true})$ = true
But you can build a machine that inverts the predicate:
P_{bad} returns true if Σ* is false and returns false if Σ* is true
So $P(P_{bad})$ would return true when it’s really false and
$P(P_{good})$ would return false when it’s really true

(b) L is infinite.

Machine P returns true if L is infinite is true for a given machine
Machine P returns false if L is infinite is false for a given machine
Machine P_{true} returns true if if a given machine returns true
Machine P_{false} returns false if a given machine returns false
Where $P(P_{false})$ = false and $P(P_{true})$ = true
But you can build a machine that inverts the predicate:
P_{bad} returns true if L is infinite is false and returns false if L is infinite is true
So $P(P_{bad})$ would return true when it’s really false and
$P(P_{good})$ would return false when it’s really true

8. Let g = id, h = $p_1^{(3)} + p_3^{(3)}$, and let f be defined from g and h by primitive recursion.

f(x,0) = g(x) = id(x)
f(x,y+1) = h(x, y, f(x,y)) = $(p_1^{(3)} + p_3^{(3)})$(x,y, f(x,y)) = x + f(x,y)

So
f(x,0) = id(x) //base case
f(x,y+1) = x + f(x,y) //recursive case

(a) Compute the values f(3, 0), f(3, 1), f(3, 2).

```
f(3,0) = 3

f(3,1) = 3 + f(3,0)
       = 3 + 3
       = 6

f(3,2) = 3 + f(3,1)
       = 3 + 6
       = 9
```

(b) Give a closed form (nonrecursive) definition of the function f.

f(x,y) = xy + x = x(y + 1)

9. Let g(x, y, z) be a primitive recursive function. Show that each of the following functions is primitive recursive.

(a) f(x, y) = g(x, y, x)

$f(x,0) = g(x,0,x)$
$f(x,y+1) = h(x,y,f(x,y)) = g(x,y,x)$ where $h = g(p_1^3, p_2^3, p_1^3)$

(b) f(x, y, z, 2) = g(x, y, x)

$f(0,y,z,2) = g(0,y,0)$
$f(x+1,y,z,2) = h(x,y,z,2,f(x,y,z,2))$ where $h = g(p_1^4, p_2^4, p_1^4)$

(c) f(x) = g(1, 2, x)

$f(0) = g(1,2,0)$
$f(x+1) = h(x,f(x))$ where $h = g(c_1^2, c_2^2, p_1^2)$

10. Show that gcd(x, y) = the greatest common divisor of x and y is primitive recursive

Building on the definitions of the division family of primitive recursive functions, the books provides a primitive recursive predicate function divides:

```
divides(x,y) = if x > 0, y > 0, and y is a divisor of x: 1
                                                 otherwise: 0
```

We can use the divides and the sg functions, we can compose a new gcd function:

```
gcd(x,y) = (sg(x) * sg(y)) * (x - g(x,y)) where
g(x,y) = uz[divides(x,x-z) * divides(y,x-z)]
```

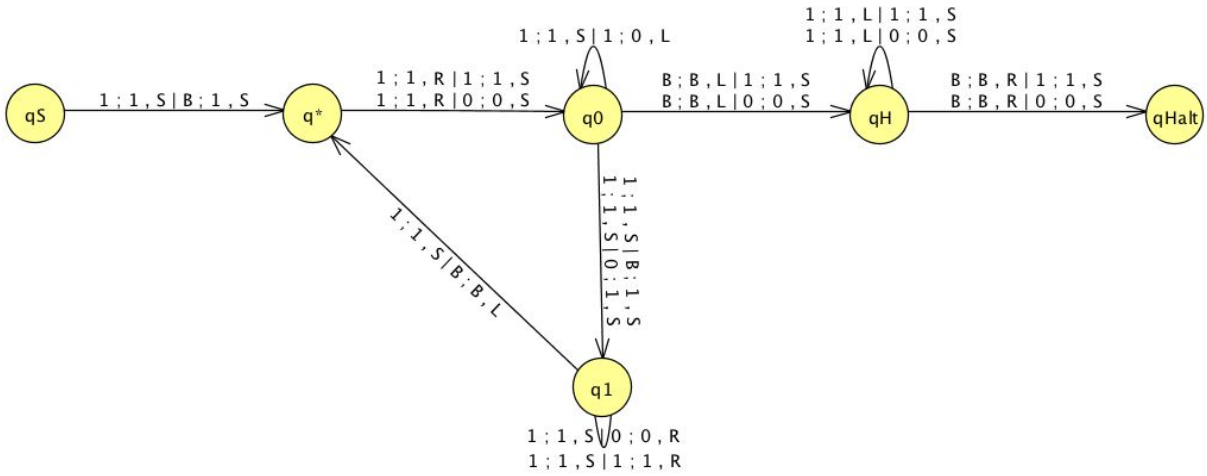
g(x,y) returns the value of z that satisfies the predicate divides(x,x-z) and divides(y,x-z). This is a number that divides both x and y. As long as x and y are not zero, sg(x) and sg(y) will both return 1, so gcd(x,y) will be 1 * (x - g(x,y))

I had to work this one out in haskell first:

```haskell
import Data.List
import Test.QuickCheck

gcd' x y = (sg x) * (sg y) * (x - (unwrap (g x y)))

g x y = findIndex p xs where
  p = (\x -> x == True)
  xs = [g'' x y z | z <- [0..x]]

-- helper functions
divides m n = m `rem` n == 0
p x y z = (divides x (x - z)) && (divides y (x - z))
sg x = if x == 0 then 0 else 1
unwrap (Just x) = x
unwrap Nothing = 1
g'' x y z
  | x == z = False
  | otherwise = p x y z

-- tests
zs = [(x,y) | x <- [1..10], y <- [1..10]]
buildList :: (Int -> Int -> Int) -> t -> [Int]
buildList f xs = map (\(x,y) -> f x y) zs
-- Narrow the type of stock gcd
gcd'' :: Int -> Int -> Int
gcd'' = gcd
test = (buildList gcd'' zs) == (buildList gcd' zs)
main = quickCheck test
```

11. Design a two-tape Turing machine that transforms unary numbers to binary numbers.

Note: I cheated a bit here and extended the tape on the left to make it easier to grow binary numbers to the left.



M = (Q, Σ, Γ, δ, q0, F)
q0 = qs
Q = {q2,q*,q0,q1,qH,F}
Γ = {0,1,B}
Σ = {1}
F = qHalt

δ(qs, 1, B) = [q*; 1, S; 1, S]
δ(q*, 1, 0) = [q0; 1, R; 0, S]
δ(q*, 1, 1) = [q0; 1, R; 1, S]
δ(q0, B, 0) = [qH; B, L; 0, S]
δ(q0, B, 1) = [qH; B, L; 1, S]
δ(q0, 1, 1) = [q0; 1, S; 0, L]
δ(q0, 1, 0) = [q1; 1, S; 1, S]
δ(q0, 1, B) = [q1; 1, S; 1, S]
δ(q1, 1, 0) = [q1; 1, S; 0, R]
δ(q1, 1, 1) = [q1; 1, S; 1, R]
δ(q1, 1, B) = [q*; 1, S; B, L]
δ(qH, 1, 0) = [qH; 1, L; 0, S]
δ(qH, 1, 1) = [qH; 1, L; 1, S]
δ(qH, B, 0) = [qHalt; B, R; 0, S]
δ(qH, B, 1) = [qHalt; B, R; 1, S]

(b) Time complexity is linear: $tc_M(7n)$

n	steps	diff	factor
1	5	5	5.0
2	12	7	6.0
3	17	5	5.7
4	26	9	6.5
5	31	5	6.2
6	38	7	6.3
7	43	5	6.1
8	54	11	6.8
9	59	5	6.6
10	66	7	6.6
11	77	11	7.0
12	80	3	6.7
13	85	5	6.5
14	92	7	6.6
15	97	5	6.5
16	110	13	6.9
17	115	5	6.8
18	122	7	6.8
19	127	5	6.7
20	136	9	6.8
40	276	-	6.9
80	556	-	7.0

12.

(a) Construct a deterministic machine that reduces the language L to Q in polynomial time.

i. L = {a^i $(bb)^i$ | i ≥ 0} and Q = {a^i b^i | i ≥ o}

R = (Q, Σ, Γ, δ, q0, F)
q0 = qS
Q = {qS,qH,qHH,q0,q1,q2,q3,q4,q5,F}
Γ = {a,b,x,$,B}
Σ = {a,b}
F = qHalt

δ(qS, a) = [q0; a, S]
δ(qS, b) = [qH; a, R]
δ(qH, a) = [qH; B, R]
δ(qH, b) = [qH; B, R]
δ(qH, B) = [qHH; B, L]
δ(qHH, B) = [qHH; B, L]
δ(qHH, a) = [qHalt; a, S]
δ(q0, a) = [q0; a, R]
δ(q0, b) = [q1; x, R]
δ(q0, B) = [q2; $, L]
δ(q1, b) = [q0; b, R]
δ(q1, B) = [q2; $, L]
δ(q2, a) = [q2; a, L]
δ(q2, b) = [q2; b, L]
δ(q2, x) = [q2; x, L]
δ(q2, $) = [q2; $, L]
δ(q2, B) = [q3; B, R]
δ(q3, a) = [q4; B, R]
δ(q3, b) = [q5; B, R]
δ(q3, $) = [qHalt; B, R]
δ(q3, x) = [q3; B, R]
δ(q4, B) = [q2; a, L]
δ(q4, $) = [q4; $, R]
δ(q4, x) = [q4; x, R]
δ(q4, a) = [q4; a, R]
δ(q4, b) = [q4; b, R]
δ(q5, B) = [q2; b, L]
δ(q5, $) = [q5; $, R]
δ(q5, x) = [q5; x, R]
δ(q5, a) = [q5; a, R]
δ(q5, b) = [q5; b, R]

(b) (Optional – this is a complexity problem.) Using the big oh notation, give the time complexity of the machine that computes the reduction:

tc_M ~= $O(n^3)$

n	steps	log
1	27	-
2	72	6.2
3	139	4.5
4	228	3.9
5	339	3.6
6	472	3.4
7	627	3.3
8	804	3.2
9	1003	3.1

Napkin math: This looks polynomial. You can see the leading coefficient is drowning out any constant and +n work being done as n increases. As n increases, it looks like the x in n^x gets closer and closer to 3. My guess is that $tc_M = O(n^3 + kn + k) = O(n^3)$.