Assignment#2 Key

1a. ProperPrefix(L) = { x | w is in L, y is not lambda and w = xy }

- Let **L** be a Regular language over the finite alphabet **Σ**. For each **a∈Σ**, define **f(a) = {a,a'}**, **g(a) = a'** and **h(a) = a**, **h(a') = λ**,
  f is a substitution, **g** and **h** are homomorphisms.
  ProperPrefix(L) = h(f(L) ∩ (Σ* g(Σ^+))
- Why this works:
  f(L) gets us every possible random priming of letters of strings in **L**.
  Σ* g(Σ^+) gets every word that ends with at least one letter primed and starts in a sequence (possibly null) of unprimed letters. Intersecting this with **f(L)** gets strings in **L** with non-null suffixes primed and the rest(the proper prefix) unprimed**.**
  Applying the homomorphism **h** erases all primed letters getting proper prefixes. This works as Regular Languages are closed under intersection, concatenation, *, $^+$, substitution, and homomorphism.
- Can also create an NFA from DFA for **L**, but that's too much work.

1a. ProperSuffix(L) = { x | w is in L, y is not lambda and w = xy or w = yx }

- Let **L** be a Regular language over the finite alphabet **Σ**. For each **a∈Σ**, define **f(a) = {a,a'}**, **g(a) = a'** and **h(a) = a**, **h(a') = λ**,
  f is a substitution, **g** and **h** are homomorphisms.
  ProperSuffix(L) = h(f(L) ∩ (g(Σ^+) Σ*))
- Why this works:
  f(L) gets us every possible random priming of letters of strings in **L**.
  g(Σ^+) Σ* gets every word that starts with at least one letter primed and ends in a sequence (possibly null) of unprimed letters. Intersecting this with **f(L)** gets strings in **L** with non-null prefixes primed and the restThe proper suffix) unprimed**.**
  Applying the homomorphism **h** erases all primed letters getting proper suffixes. This works as Regular Languages are closed under intersection, concatenation, *, $^+$, substitution, and homomorphism.
- Can also create an NFA from DFA for **L**, but that's too much work.

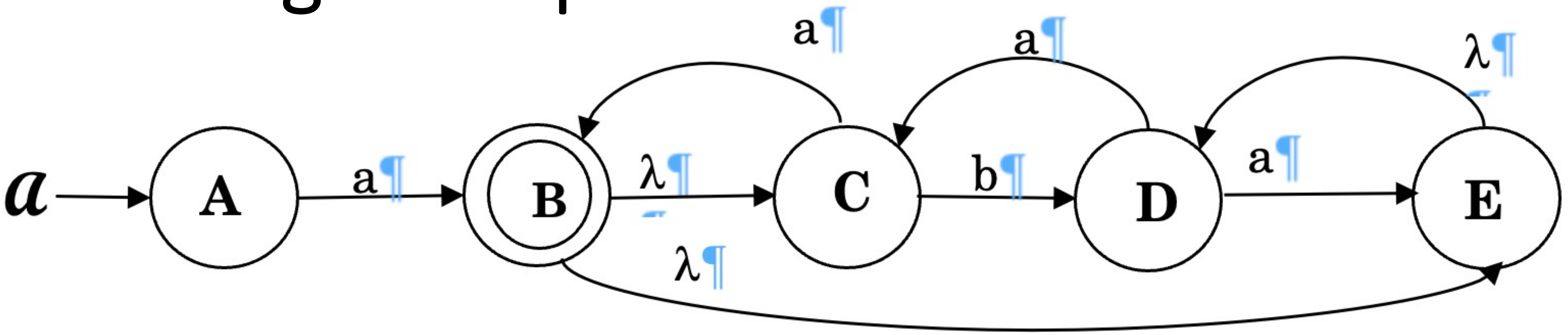
1a. ProperPreOrSuffix(L) = { x | w is in L, y is not lambda and w = yx }

- Let **L** be a Regular language over the finite alphabet **Σ**. For each **a∈Σ**, define **f(a) = {a,a'}**, **g(a) = a'** and **h(a) = a**, **h(a') = λ**,
  f is a substitution, **g** and **h** are homomorphisms.
  ProperPreOrSuffix(L) = h(f(L) ∩ (Σ* g(Σ⁺) ∪ (g(Σ⁺) Σ*))
- Why this works:
  Look back at ProperPrefix and ProperSuffix. This works as Regular Languages are closed union, intersection, concatenation, *, ⁺, substitution, and homomorphism..
- Can also create an NFA from DFA for **L**, but that's too much work.

1b. LastHalf(L) = { y | there exists a string x , |x| = |y| and xy is in L }

- Let **L** be a Regular language over the finite alphabet **Σ**. Assume **L** is recognized by the DFA
  A_1 = (Q, Σ, δ_1, q_1, F). Define the NFA
  A_2 = ((Q×Q×Q)∪{q_0}, Σ, δ_2, q_0, F'), where
  $\delta_2(q_0,\lambda)$ = union(q∈Q) {<q_1, q, q >} and
  δ_2(< s, t, u >, b) = union(a∈Σ) { < δ_1(s,a), δ_1(t,b), u> }, s,t,u ∈ Q
  F' = union(q∈Q) {<q, f, q>}, f∈F
- Why this works:
  The first part of a state **< s, t, u>** tracks **A_1** for all possible strings that are the same length as what **A_2** is reading in parallel**.** We guess it will end up in state **q** and so **u=q** to remember that guess.
  The second part of state **< s, t, u >** tracks **A_1** as if it has read a string that ended in state **q** (**u=q**).
- Thus, we start with a guess (**q**) as to what state **A_1** might end up in reading a string of length **x.** The guess is checked by requiring us to start up in state **q** in the mid part which reads **y**, where **|x|=|y|**.
- The final states check that our guess was correct, and that we could end in a final state of **A_1**, with using the guess when we started reading the second part.

2. Use Regular Equations to Solve for B



$A = \lambda$

$B = Aa + Ca \qquad = a + Ca = a + Ba^* (ba^+)^* a = a(a^* (ba^+)^* a)^*$

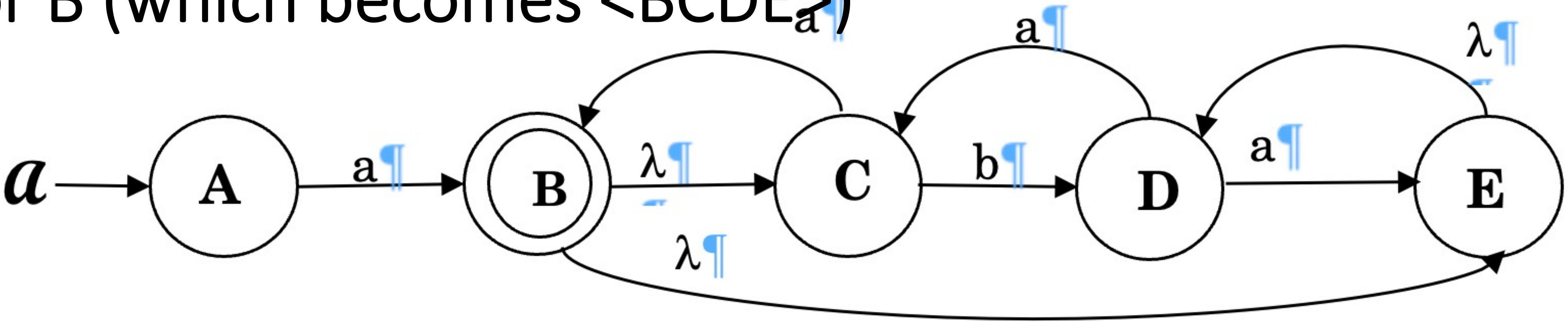
$C = B + Da \qquad = B + (Cb + B)\, a^+ = B + B\, a^+ + Cba^+ = (B + B\, a^+)\, (ba^+) = Ba^* (ba^+)^*$

$D = Cb + E \qquad = Cb + B + Da = (Cb + B)\, a^*$

$E = B + Da$

$L = B = a(a^* (ba^+)^* a)^*$

2. Use Lambda Closure and Regular Equations to Solve for B (which becomes <BCDE>)



A = λ
<BCDE> = Aa + <CDE>a + <BCDE>a = a + <BCDE> (a + b(ab)*a) = a(a + b(ab)*a)*
D = <BCDE>b + <CDE>b
<CDE> = Da = <BCDE>b + Dab = <BCDE>b(ab)*

L = <BCDE> = a(a + b(ab)* a)* = a(a* (ba^{+})* a)*
Proof of equivalent can be done by mutual inclusion.

3. $L = \{ ba^n ab^n \mid n > 0 \}$

a.) Use the **Pumping Lemma for Regular Languages** to show **L** **<u>is not</u>** Regular.

Assume **L** is Regular

Let **N>0** be value provided by PL

Choose **$ba^N ab^N$** as a string in **L**

PL splits **$ba^N ab^N$** into **xyz** such that **$|xy| \leq N$** and **$|y| > 0$**.

I have two cases:

y contains a **b**. This means the **b** is the starting character as **$|xy| \leq N$**

Let **i=0** then we erase the starting **b** and the resulting string is not in **L**.

y is strictly over **a's**. Set **i=0** and we get **$ba^{N-|y|} ab^N$** but then the starting **a**'s don't match the ending **b** in number and so the resulting string is not in **L**.

That two cases cover all possible cases, given the constraints, and so we get a contradiction for all possibilities and so **L** is not Regular based on the PL.

3. L = $\{ ba^n ab^n \mid n > 0 \}$

b.) Use the **Myhill-Nerode Theorem** to show **L <u>is not</u>** Regular.

Define the equivalence classes **$[ba^i]$, $i > 0$**

Clearly **ba^iab^i** is in **L**, but **ba^jab^i** is not in **L** when **$j \neq i$, $i, j>0$**

Thus, **$[a^i] \neq [a^j]$** when **$j \neq i$, $i, j>0$** and so the index of **R_L** is infinite.

By Myhill-Nerode, **L** is not Regular.