Chapter 07

Kleene's Theorem

Part 3: RE to FA

CS 4110 Notes

Proof of Kleene's Theorem: Part 3

- Need to show that **for any RE we can construct a FA** that recognizes the language it represents.
- The strategy is to
  - start with known FA for very simple RE.
  - A complex RE can be obtained by combining simple REs using **choice**, **sequence** and **iteration**.
  - Then we need to find methods to combine simple FA into a complicated one for **choice**, **sequence** and **iteration**.

a^* ab

[(ab+ba)(aa+bb)*(ab+ba)+aa+bb]*

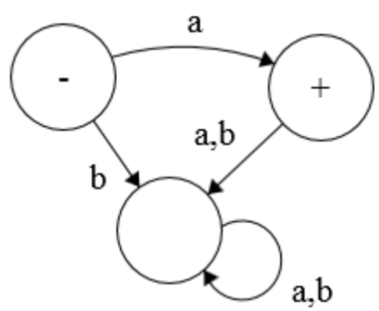
a+b

Choice/union **Sequence/product** **iteration**

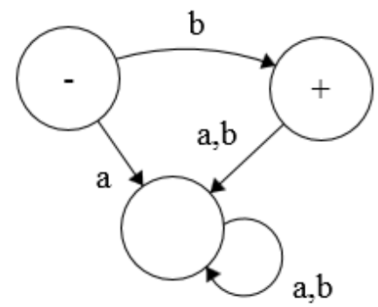
Proof by Recursive Definition and Parallel Construction

- Rule 1 (**base cases**) There is an FA that accepts a single character in Σ. There is an FA that accepts **Λ**

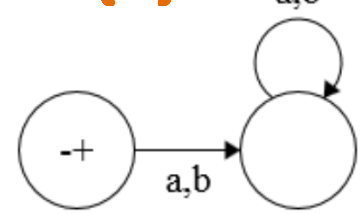
L={a}



L={b}



L={Λ}



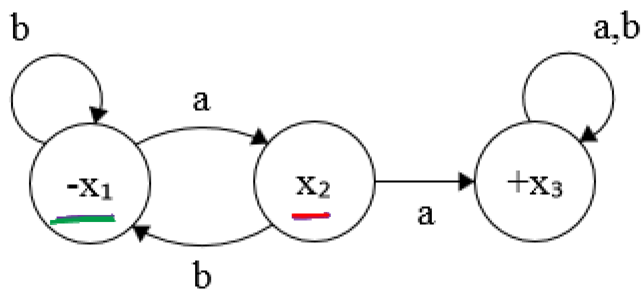
- Rule 2 (**choice/union**) If FA1 accepts RE1 and FA2 accepts RE2 we can construct FA3 that accepts RE1 + RE2 (page 109).
- Rule 3 (sequence/**product**) If FA1 accepts RE1 and FA2 accepts RE2 we can construct FA3 that accepts RE1RE2 (page 117).
- Rule 4 (**iteration**) If FA1 accepts RE1 we can construct FA2 that accepts (RE1)* (page 125).

Rule 2: Choice (Union of Two Machines)

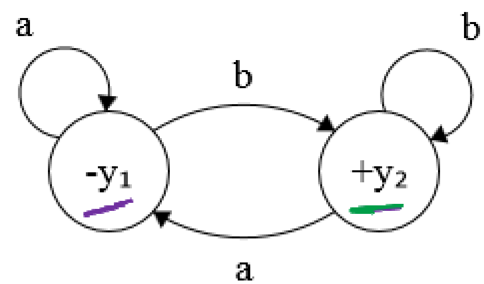
- If FA1 accepts RE1 and FA2 accepts RE2 we can construct FA3 that accepts **RE1 + RE2**
- Algorithm for constructing the union of two machines:
  - We imagine an input string being processed simultaneously on FA1 and FA2 and each z state represents "x_something or y_something"
  - We keep track of what state the input would take us to in FA1 and what state it would take us to in FA2 and we create enough **composite states** to represent the reachable two-state combinations.
  - This must be a finite number because the number of states in FA1 and the number of states in FA2 are both finite.
  - Each new state (z-state) corresponds to some **combination of x-states and y-states**

Example 07.03: RE → FA

a+b



L1={all strings containing **aa**}



L2={all strings ending in **b**}

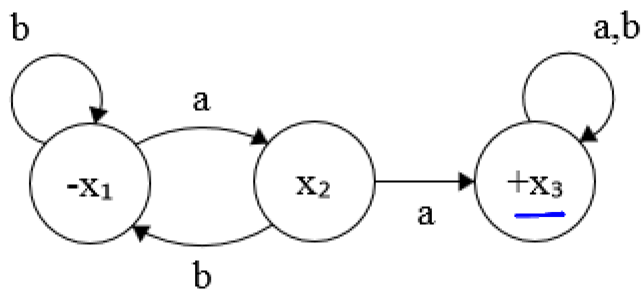
(1) Start State: **z1** = x1 or y1 (-)

(2) At state z1,
- ✓ if reading "a", FA1→x2 and FA2 →y1, so **z2** = x2 or y1
- ✓ if reading "b", FA1→x1 and FA2 →y2, so **z3** = x1 or y2 (**+**)

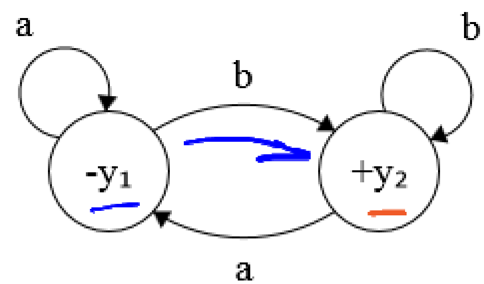
(3) At state z2,
- ✓ if reading "a", FA1→x3 and FA2 →y1, so **z4** = x3 or y1 (**+**)
- ✓ if reading "b", FA1→x1 and FA2 →y2, so still z3 (not a new state)

(4) At state z3,
- ✓ if reading "a", FA1→x2 and FA2 →y1, so z2 (not a new state)
- ✓ if reading "b", FA1→x1 and FA2 →y2, so z3 (not a new state)

Example 07.03: RE → FA



L1={all strings containing **aa**}

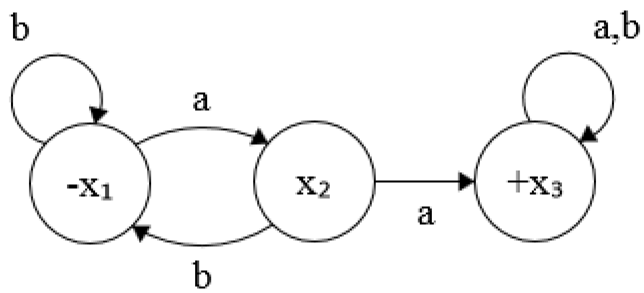


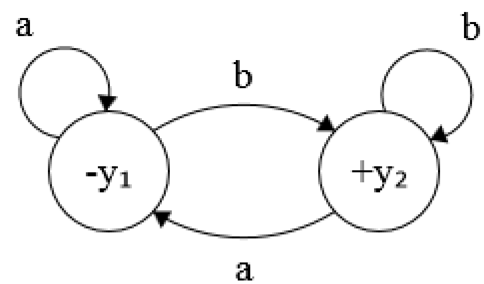
L2={all strings ending in **b**}

(1) At state z4 (x3 or y1),
- ✓ if reading "a", FA1→x3 and FA2 →y1, so z4 (not a new state)
- ✓ if reading "b", FA1→x3 and FA2 →y2, so **z5** = x3 or y2 (**+**)

(2) At state z5,
- ✓ if reading "a", FA1→x3 and FA2 →y1 so z4 (not a new state)
- ✓ if reading "b", FA1→x3 and FA2 →y2, so z5 (not a new state)

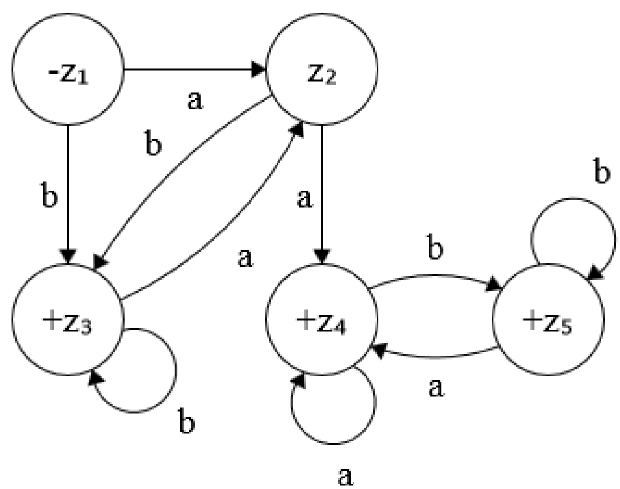
Example 07.03: RE → FA





L1={all strings containing **aa**}

L2={all strings ending in **b**}



L3= L1UL2 ={all strings containing **aa** or ending in **b**}

Thanks, see you next time!

CS4110 Notes