Today's lecture: Regular Languages

I. Examples of Regular Languages
II. Closure Properties
III. More Examples

Reading:

Today: Regular Sets – Read: *Kozen* lecture 4 (or *Sipser* 1.1. as before).

September 14: Non-Determinism – Read: *Kozen* lecture 5 or *Sipser* 1.2.

September 16: Equivalence of DFAs and NFAs – Read: *Kozen* lecture 6 (or *Sipser* 1.2. as before).

September 19: Regular Expressions – Read: *Kozen* lectures 7 & 8 or *Sipser* 1.3.

September 21: Equivalence of Regular expressions and Finite Automata Read: *Kozen* lecture 9 (or *Sipser* 1.3. as before).

September 23: Nonregular Languages – Read: *Kozen* lecture 12 or *Sipser* 1.4.

September 26: More examples of the pumping Lemma Read: *Kozen* lecture 13 or *Sipser* 1.4.

September 28: Applications of finite automata

September 30: More applications

October 26: Midterm: In class.

I. Examples of Regular Languages

A. Examples that we've already seen:

1. Strings in $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$ such that every '$\mathtt{a}$' is followed by a '$\mathtt{b}$' without an intervening '$\mathtt{c}$' (DQ Sept. 9).
2. Strings in $\{\mathtt{0}, \mathtt{1}\}^*$ with an odd number of '$\mathtt{1}$'s (Sept. 9 lecture).
3. Strings in $\{\mathtt{a}, \mathtt{b}\}^*$ that have at least three '$\mathtt{a}$'s (Kozen, lecture 3).
4. Strings in $\{\mathtt{a}, \mathtt{b}\}^*$ that have three consecutive '$\mathtt{a}$'s (Kozen, lecture 3).
5. Strings in $\{\mathtt{0}, \mathtt{1}\}^*$ that end with a '$\mathtt{1}$' (Sipser, example 1.2).
6. Strings in $\{\mathtt{0}, \mathtt{1}\}^*$ that don't end with a '$\mathtt{1}$' (Sipser, example 1.3).
7. Strings in $\{\mathtt{a}, \mathtt{b}\}^*$ that start and end with the same symbol.
8. A simplistic spelling checker (Sept. 9 lecture).
9. Strings in $\{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{reset}\}^*$ such that the sum of the input symbols (when interpreted in the obvious way as integers) since the last '$\mathtt{reset}$' (or the beginning of the input string if the input has no '$\mathtt{reset}$' symbols) is a multiple of three (Sipser example 1.5).
10. Strings in $\{\mathtt{0}, \mathtt{1}, \ldots k-1, \mathtt{reset}\}^*$ such that the sum of the input symbols as calculated in the previous example is a multiple of k.

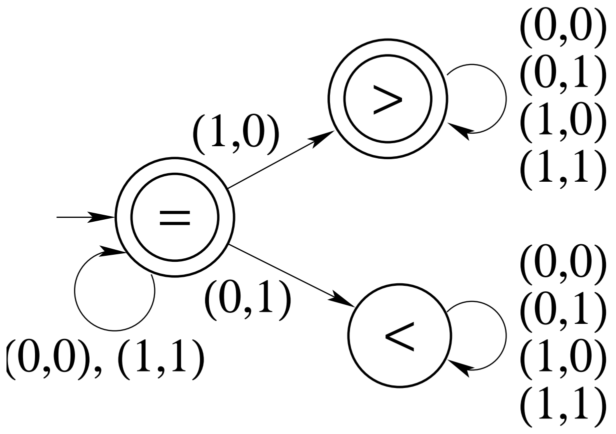

Figure 1: A finite automaton for comparing two numbers

11. Strings in $\{\texttt{0},\ \texttt{1}\}^*$ that when interpreted as binary numbers (most significant bit first) are a multiple of three.

B. One more example: a language that inputs pairs of binary numbers and accepts if the first number is greater than or equal to the second. The number are input with the most significant bit first.

1. $\Sigma = (\{\texttt{0}, \texttt{1}\}, \{\texttt{0}, \texttt{1}\})$.

a. In other words, each input is a pair of two bits. The left element of the pair is the bit for the "first" number being input, and the right element is the bit for the "second" number being input.

b. From now on, I'll write $\mathbb{B}$ as an abbreviation for $\{\texttt{0},\ \texttt{1}\}$.

2. Acceptance condition:
I first define a function $\mathsf{val} : \Sigma^* \rightarrow \mathbb{Z} \times \mathbb{Z}$ as shown below (where $\mathbb{Z}$ denotes the integers):

$$\begin{array}{rcl} \mathsf{val}(\epsilon) & = & (0,0) \\ \mathsf{val}(s \cdot (\mathsf{a},\mathsf{b})) & = & (2 * \mathsf{first}(\mathsf{val}(s)) + \mathsf{a}, 2 * \mathsf{second}(\mathsf{val}(s)) + b) \end{array}$$

A string, s, is in the language iff $\mathsf{first}(\mathsf{val}(s)) \geq \mathsf{second}(\mathsf{val}(s))$.

3. A finite automaton that recognizes this language

a. Design

i.. $\mathsf{val}(\epsilon) = (0,0)$. Thus, the empty string is accepting. This means that the initial state is accepting.

ii.. Now, consider two binary numbers represented by strings s_1 and s_2 in $\mathbb{B}^*$. Assume that s_1 and s_2 have the same length. Let x_1, y_1, x_2, y_2 be strings such that and $s_1 = x_1y_1$; $s_2 = x_2y_2$; and x_1 and x_2 have the same length (thus y_1 and y_2 have the same length). Note that if the binary number represented by x_1 is greater than the binary number represented by x_2, then $s_1 > s_2$ regardless of the details of y_1 and y_2. Conversely, if $x_1 < x_2$, then $s_1 < s_2$. Strings y_1 and y_2 only matter if $x_1 = x_2$.

iii.. Based on these observations, our automaton keeps track of whether the first number is less-than, equal-to, or greater-than the second based on the prefix that it has read so far. The machine starts in the state where the two numbers are equal (the empty string encodes 0 for both). It moves to the "less-than" or "greater-than" state as soon as it sees a bit where the two numbers differ. Once it has done so, it remains in that state for the rest of the string.

iv.. The machine accepts if the first number is greater-than-or-equal to the second. Thus, the states corresponding to "greater-than" and "equal" are accepting states, and the state corresponding to "less-than" is a non-accepting state.

b. Figure 1 shows the finite automaton based on this design.

c. Here's the formal definition of the machine:

i.. $\Sigma = \mathbb{B} \times \mathbb{B}$

ii.. $Q = \{\mathsf{LT}, \mathsf{EQ}, \mathsf{GT}\}$

iii..
$$\begin{array}{rcll} \delta(\mathsf{LT}, c) & = & \mathsf{LT}, & \text{for all } c \text{ in } \Sigma \\ \delta(\mathsf{EQ}, (0,0)) & = & \mathsf{EQ} & \\ \delta(\mathsf{EQ}, (0,1)) & = & \mathsf{LT} & \\ \delta(\mathsf{EQ}, (1,0)) & = & \mathsf{GT} & \\ \delta(\mathsf{EQ}, (1,1)) & = & \mathsf{EQ} & \\ \delta(\mathsf{GT}, c) & = & \mathsf{GT}, & \text{for all } c \text{ in } \Sigma \end{array}$$

iv.. $q_0 = \mathsf{EQ}$

v.. $F = \{\mathsf{EQ}, \mathsf{GT}\}$

4. Correctness proof: by induction on the length of the input string.

a. Induction hypothesis:
Define $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ in the usual way. The induction hypothesis is:

$$\begin{array}{rcll} \hat{\delta}(q_0, s) & = & \mathsf{LT}, & \text{iff } \mathsf{first}(\mathsf{val}(s)) < \mathsf{second}(\mathsf{val}(s)) \\ \hat{\delta}(q_0, s) & = & \mathsf{EQ}, & \text{iff } \mathsf{first}(\mathsf{val}(s)) = \mathsf{second}(\mathsf{val}(s)) \\ \hat{\delta}(q_0, s) & = & \mathsf{GT}, & \text{iff } \mathsf{first}(\mathsf{val}(s)) > \mathsf{second}(\mathsf{val}(s)) \end{array}$$

b. Base step: $s = \epsilon$
From the definition of $\mathsf{val}(s)$, $\mathsf{val}(\epsilon) = (0, 0)$. Thus, $\mathsf{first}(\mathsf{val}(\epsilon)) = 0$, and $\mathsf{second}(\mathsf{val}(\epsilon)) = 0$. By the definition of $del\hat{t}a$, $\hat{\delta}(q_0, \epsilon) = q_0$, and by the definition of the automaton, $q_0 = \mathsf{EQ}$. Thus, the induction hypothesis is satisfied for the base case.

c. Induction step: $s = x \cdot \mathsf{c}$
We consider three cases depending on $del\hat{t}a(x)$.
case $del\hat{t}a(q_0, x) = \mathsf{LT}$:

$$\begin{array}{rrcll} 1. & \mathsf{first}(\mathsf{val}(x)) & < & \mathsf{second}(\mathsf{val}(x)), & \text{Ind. and case hypotheses} \\ 2. & \mathsf{first}(\mathsf{val}(x)) & \leq & \mathsf{second}(\mathsf{val}(x)) - 1, & \mathsf{val}(s) \in \mathbb{Z} \times \mathbb{Z} \\ 3. & 2 * \mathsf{first}(\mathsf{val}(x)) + 1 & \leq & 2 * \mathsf{second}(\mathsf{val}(x)) - 1, & \text{mult. both sides by 2 and add 1} \\ 4. & 2 * \mathsf{first}(\mathsf{val}(x)) + 1 & < & 2 * \mathsf{second}(\mathsf{val}(x)), & \text{add one more to the right side} \\ 5. & 2 * \mathsf{first}(\mathsf{val}(x)) + \mathsf{first} c & \leq & 2 * \mathsf{first}(\mathsf{val}(x)) + 1, & c \leq 1 \\ 6. & \mathsf{first}(\mathsf{val}(s)) & = & 2 * \mathsf{first}(\mathsf{val}(x)) + c, & \text{def. } \mathsf{val} \\ 7. & \mathsf{first}(\mathsf{val}(s)) & < & 2 * \mathsf{second}(\mathsf{val}(x)), & \text{steps 4, 5, 6} \\ 8. & 2 * \mathsf{second}(\mathsf{val}(x)) & \leq & \mathsf{second}(\mathsf{val}(s)), & \text{analogous to steps 5, 6, 7} \\ 9. & \mathsf{first}(\mathsf{val}(s)) & < & \mathsf{second}(\mathsf{val}(s)), & \text{steps 7, 8} \\ 10. & \hat{\delta}(q_0, s) & = & \delta(\hat{\delta}(q_0, x), c), & \text{def. } \hat{\delta} \\ 11. & \hat{\delta}(q_0, s) & = & \delta(\mathsf{LT}, c), & \text{case hypothesis} \\ 12. & \hat{\delta}(q_0, s) & = & \mathsf{LT}\text{def. } \delta & \\ 13. & \text{The induction hypothesis is maintained} & & & \text{steps 9, 12} \end{array}$$

d. $\square$

5. Can you design an automaton that accepts pairs of binary numbers and accepts if the first is greater-than-or-equal-to the second when the numbers are input with their least significant bits first?

II. Translating finite automata into sequential circuits

Reminder: I'll never ask you to do this on homework or tests. This is just to aid intuition for the mathematical constructions that come next.

A. Consider the language, B_1 over $\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}^*$, where every '$\mathsf{a}$' is followed by a '$\mathsf{b}$' without an intervening 'c'.

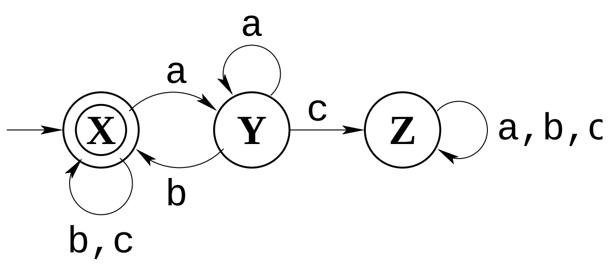

Figure 2: The state transition diagram for a finite automaton that recognizes the language described in II.A

1. Figure 2 shows the state transition diagram for a finite automaton that recognizes this language.

2. The formal definition of the automaton is the 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

$$\begin{aligned} Q &= \{\mathbf{X}, \mathbf{Y}, \mathbf{Z}\} \\ \Sigma &= \{\mathsf{a}, \mathbf{b}, \mathbf{c}\} \\ \delta \in Q \times \Sigma \to \Sigma \\ q_0 = \mathbf{X} \\ F = \{\mathbf{X}\} \end{aligned}$$

with

$$\begin{array}{lll} \delta(\mathbf{X}, \mathsf{a}) = \mathbf{Y}, & \delta(\mathbf{X}, \mathsf{b}) = \mathbf{X}, & \delta(\mathbf{X}, \mathsf{c}) = \mathbf{X}, \\ \delta(\mathbf{Y}, \mathsf{a}) = \mathbf{Y}, & \delta(\mathbf{Y}, \mathsf{b}) = \mathbf{X}, & \delta(\mathbf{Y}, \mathsf{c}) = \mathbf{Z}, \\ \delta(\mathbf{Z}, \mathsf{a}) = \mathbf{Z}, & \delta(\mathbf{Z}, \mathsf{b}) = \mathbf{Z}, & \delta(\mathbf{Z}, \mathsf{c}) = \mathbf{Z} \end{array}$$

3. We represent states with two binary bits, and arbitrarily choose to use the mapping:

$$\begin{array}{lcl} \mathbf{X} & \leftrightarrow & (0,0) \\ \mathbf{Y} & \leftrightarrow & (0,1) \\ \mathbf{Z} & \leftrightarrow & (1,0) \end{array}$$

We will write $Q_{hw} = \{(0,0),\ (0,1),\ (1,0)\}$ to represent the set of states used by the hardware.

4. Likewise, we represent each input symbol with two bits, and arbitrarily choose the mapping

$$\begin{array}{lcl} \mathsf{a} & \leftrightarrow & (0,0) \\ \mathsf{b} & \leftrightarrow & (0,1) \\ \mathsf{c} & \leftrightarrow & (1,0) \end{array}$$

We will write $\Sigma_{hw} = \{(0,0),\ (0,1),\ (1,0)\}$ to represent the set of input symbols used by the hardware.

5. Applying these mappings to δ we get

$$\begin{array}{lcll} \delta_{hw}((0,0),\ (0,0)) & = & (0,1), & \delta(\mathbf{X}, \mathsf{a}) = \mathbf{Y} \\ \delta_{hw}((0,0),\ (0,1)) & = & (0,0), & \delta(\mathbf{X}, \mathsf{b}) = \mathbf{X} \\ \delta_{hw}((0,0),\ (1,0)) & = & (0,0), & \delta(\mathbf{X}, \mathsf{c}) = \mathbf{X} \\ \\ \delta_{hw}((0,1),\ (0,0)) & = & (0,1), & \delta(\mathbf{Y}, \mathsf{a}) = \mathbf{Y} \\ \delta_{hw}((0,1),\ (0,1)) & = & (0,0), & \delta(\mathbf{Y}, \mathsf{b}) = \mathbf{X} \\ \delta_{hw}((0,1),\ (1,0)) & = & (1,0), & \delta(\mathbf{Y}, \mathsf{c}) = \mathbf{Z} \\ \\ \delta_{hw}((1,0),\ (0,0)) & = & (1,0), & \delta(\mathbf{Z}, \mathsf{a}) = \mathbf{Z} \\ \delta_{hw}((1,0),\ (0,1)) & = & (1,0), & \delta(\mathbf{Z}, \mathsf{b}) = \mathbf{Z} \\ \delta_{hw}((1,0),\ (1,0)) & = & (1,0), & \delta(\mathbf{Z}, \mathsf{c}) = \mathbf{Z} \end{array}$$

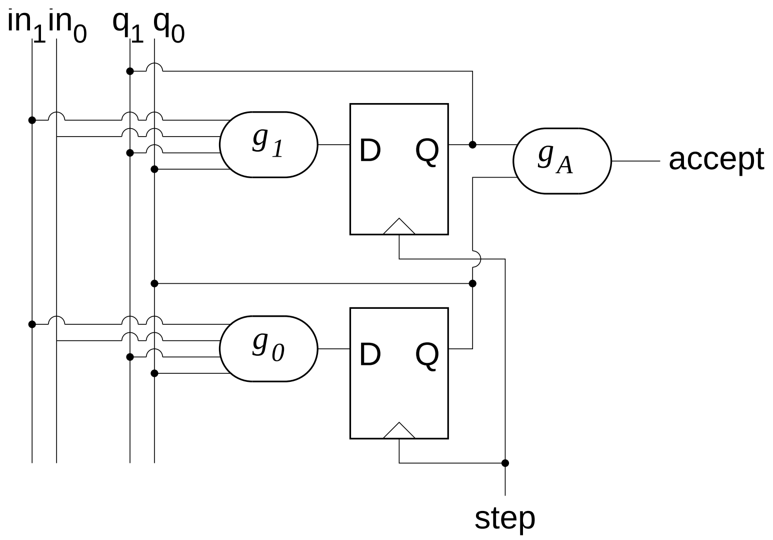

Figure 3: A template for implementing a sequential circuit for the finite automaton from figure 2

6. Figure 3 shows a template for implementing this automaton as a sequential circuit. The two flip-flops hold the state of the circuit. The circuit advances to the next state on the rising edge of the step signal. Now, we just need to implement the logic for g_0, g_1, and g_A.

a. g_0 calculates the right bit of the two-bit state-tuple. From the definition of δ_{hw} above we get:

$$\begin{aligned} g_0 &= (\mathsf{in} = \mathsf{a}) \wedge (\mathsf{q} = \mathbf{X}) \vee (\mathsf{in} = \mathsf{a}) \wedge (\mathsf{q} = \mathbf{Y}) \\ &= (\mathsf{in} = \mathsf{a}) \wedge ((\mathsf{q} = \mathbf{X}) \vee (\mathsf{q} = \mathbf{Y})) \\ &= (\mathsf{in}_1 = 0) \wedge (\mathsf{in}_0 = 0) \wedge ((\mathsf{q}_1 = 0) \wedge (\mathsf{q}_0 = 0) \vee (\mathsf{q}_1 = 0) \wedge (\mathsf{q}_0 = 1)) \\ &= \neg \mathsf{in}_1 \wedge \neg \mathsf{in}_0 \wedge \neg \mathsf{q}_1 \end{aligned}$$

b. Likewise,

$$\begin{aligned} g_1 &= (\mathsf{in} = \mathsf{c}) \wedge (\mathsf{q} = \mathbf{Y}) \vee (\mathsf{q} = \mathbf{Z}) \\ &= (\mathsf{in}_1 \wedge \mathsf{q}_0) \vee \mathsf{q}_1 \end{aligned}$$

Here, we took advantage of the fact that an input of $(1,1)$ is forbidden and that a state of $(1,1)$ is unreachable when simplifying the formula.

c. The only accepting state is **X**. Thus,

$$g_A = \neg \mathsf{q}_1 \wedge \neg \mathsf{q}_0$$

7. Figure 4 shows the circuit obtained by translating the Boolean formulas above directly into logic circuits.

8. Note that this circuit actually has four states and four input symbols. The fourth state is $(1,1)$ and the fourth input is $(1,1)$. It is straightforward to show that with the hardware shown there are no transitions into state $(1,1)$, and that state $(1,1)$ transitions to state $(1,0)$ (i.e. **Z**) regardless of what symbol is input. Similarly, it is straightforward to show that the transitions for input symbol $(1,1)$ are identical to those for input symbol $(1,0)$ (i.e. texttc).

B. Consider the language, B_2 over $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}^*$, such that $w \in B_2$ iff the number of 'a's in w is divisible by 4.

1. Figure 5 shows the state transition diagram for an automaton that recognizes language B_2.

2. The automaton from figure 5 uses the same alphabet, Σ, as the automaton from II.A. We define $Q = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}\}$, $q_0 = \mathbf{0}$, and $F = \mathbf{0}$. We define δ in the same fashion as we did in section II.A.

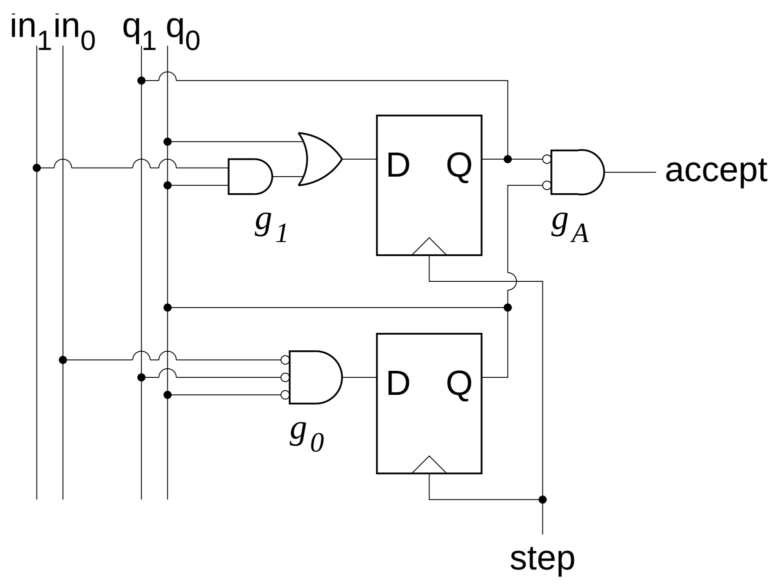

Figure 4: A sequential circuit that implements the finite automaton from figure 2

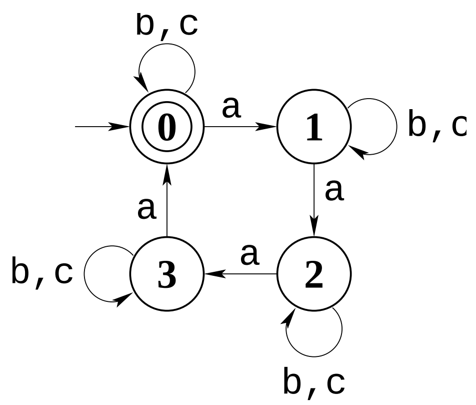

Figure 5: The state transition diagram for a finite automaton that that recognizes the language described in II.B

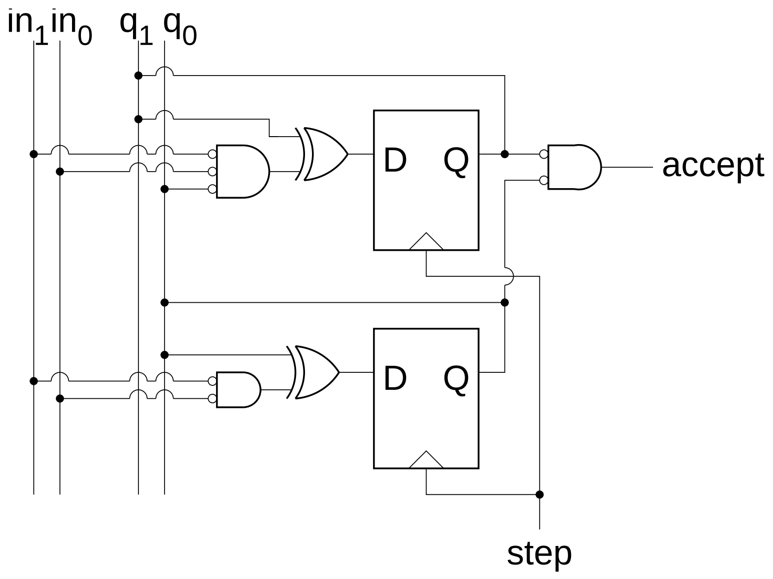

Figure 6: A sequential circuit that implements the finite automaton from figure 5

3. To make a hardware implementation, we implement input symbols with two bits using the same encoding as before – this will simplify our presentation of product machines shortly. We implement states with two bits using the mapping:

$$\mathbf{0} \leftrightarrow (0,0), \quad \mathbf{1} \leftrightarrow (0,1), \quad \mathbf{2} \leftrightarrow (1,0), \quad \mathbf{3} \leftrightarrow (1,1)$$

4. The process of translating of δ to a logic circuit is analogous to that described earlier. The acceptance condition just happens to be the same. Figure 6 shows the final circuit.

III. Closure Properties for Regular Languages

A. Regular Languages are closed under intersection:
If B_1 and B_2 are regular languages, then $B_1 \cap B_2$ is a regular language as well.

1. Intuition from the hardware perspective

a. Use the hardware from both automata

b. Connect the input to both machines

c. Connect the clock to both machines

d. The combined machine accepts if both machines accept

e. Note that if the first machine has m input bits and n_1 state bits, and the second machine has m inputs (the same input alphabet) and n_2 state bits, then the combined machine has m input bits and $n_1 + n_2$ state bits. Furthermore, n_1 state bits encode up to 2^{n_1} states, and n_2 state bits encode up to 2^{n_2} states. Thus, the combined machine has up to $2^{n_1+n_2}$ states. This is the product of the number of states for the two individual machines. This makes sense. For each of the n_1 states that machine M_1 is in, M_2 can potentially be in any of its n_2 states.

2. The product machine construction, let:

$$\begin{array}{rcl} M_1 & = & (Q_1, \Sigma, \delta_1, q_{0,1}, F_1) \\ M_2 & = & (Q_2, \Sigma, \delta_2, q_{0,2}, F_2) \end{array}$$

We'll now define $M_{1\cap 2} = (Q_{1\cap 2}, \Sigma, \delta_{1\cap 2}, q_{0,1\cap 2}, F_{1\cap 2})$ as the machine that recognizes the intersection of the languages recognized by M_1 and M_2.

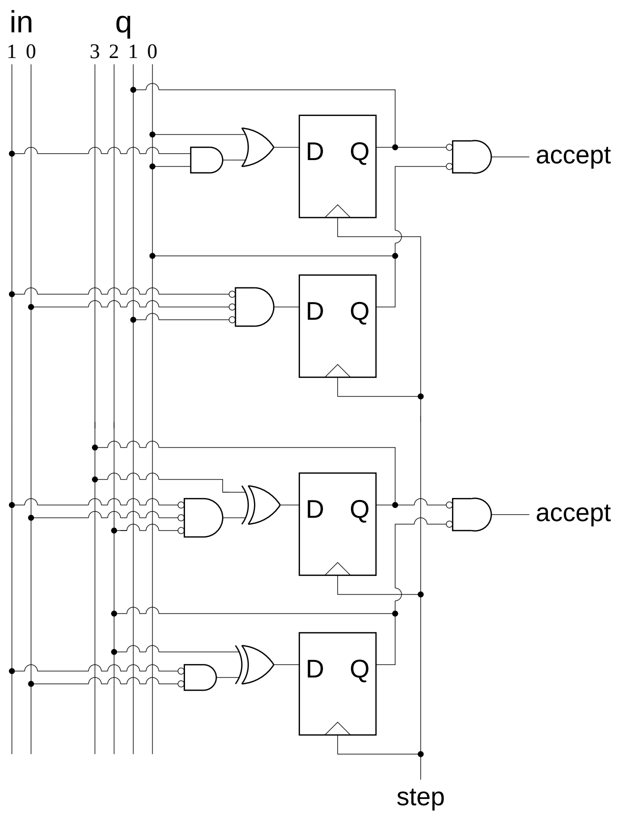

Figure 7: A sequential circuit that recognizes the intersection of the languages for the machines from figures 2 and 5

a. $Q_{1\cap 2} = Q_1 \times Q_2$
Intuitively, each state is a tuple (q_1, q_2) where q_1 corresponds to the state of machine M_1 and q_2 corresponds to the state of machine M_2. This also corresponds to the hardware construction of having the flip-flops for both machines and thus potentially having all combinations of pairs of their states.

b. Σ is unchanged. The product machine reads inputs from the same alphabet as the two original machines. This corresponds to connecting the input wires to both machines in figure 7.

c. $\delta_{1\cap 2}((q_1, q_2), \mathsf{c}) = (\delta_1(q_1, \mathsf{c}), \delta_2(q_2, \mathsf{c}))$ This says that the component of the product machine's state that corresponds to machine M_1 changes according to the state transition function of machine M_1, and likewise for the component corresponding to machine M_2. In the hardware, we see this in that the next state logic for each machine is used to update the corresponding flip-flops. Furthermore, the step signal is connected to the clock inputs of the flip-flops for both machines. This means that the two halves of the product machine take steps at the same time in response to the same input symbol.

d. $q_{0,1\cap 2} = (q_{0,1}, q_{0,2})$ – the product machine starts in the state that corresponds to the initial state of the two component machines.

e. $F = F_1 \times F_2$ – the product machine accepts if its state corresponds to states in the component machines where both component machines are accepting. This corresponds to the AND gate that produces the accept signal in figure 7.

3. Proof of closure under intersection
By induction on the length of the input string, w.

a. Define $\hat{\delta}_{1\cap 2}$ in the natural way:

$$\begin{array}{rcl} \hat{\delta}_{1\cap 2}(q, \epsilon) & = & q \\ \hat{\delta}_{1\cap 2}(q, w \cdot \mathsf{c}) & = & \delta_{1\cap 2}(\hat{\delta}_{1\cap 2}(q, w), c) \end{array}$$

b. Induction hypothesis:

$$\hat{\delta}_{1\cap 2}((q_1, q_2), s) \quad = \quad (\hat{\delta}_1(q_1, s), \hat{\delta}_2(q_2, s))$$

c. Base case: $w = \epsilon$

$$\begin{array}{rcl} \hat{\delta}_{1\cap 2}((q_1, q_2), \epsilon) & = & (q_1, q_2) \\ & = & (\hat{\delta}_1(q_1 \epsilon), \hat{\delta}_1(q_1 \epsilon)) \end{array}$$

d. Induction step: assume for w, prove for $w \cdot \mathsf{c}$

$$\begin{array}{rcl} \hat{\delta}_{1\cap 2}((q_1, q_2), w \cdot \mathsf{c}) & = & \delta_{1\cap 2}(\hat{\delta}_{1\cap 2}((q_1, q_2), w), \mathsf{c}), \text{def. } \hat{\delta}_{1\cap 2} \\ & = & \delta_{1\cap 2}((\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w)), \mathsf{c}), \text{induction hypothesis} \\ & = & (\delta_1(\hat{\delta}_1(q_1, w), \mathsf{c}), \delta_2(\hat{\delta}_2(q_2, w), \mathsf{c})), \text{def. } \delta_{1\cap 2} \\ & = & (\hat{\delta}_1(q_1, w \cdot \mathsf{c}), \hat{\delta}_2(q_2, w \cdot \mathsf{c})), \text{def. } \hat{\delta}_1, \text{ and } \hat{\delta}_2 \end{array}$$

4. Example, the machine that accepts the intersection of B_1 and B_2 as defined in sections II.A and II.B is $M = (Q_{1\cap 2}, \Sigma, \delta_{1\cap 2}, q_{0,1\cap 2}, F_{1\cap 2})$ with

$$\begin{array}{rcl} Q_{1\cap 2} & = & \{(\mathbf{X},\mathbf{0}), (\mathbf{X},\mathbf{1}), (\mathbf{X},\mathbf{2}), (\mathbf{X},\mathbf{3}), (\mathbf{Y},\mathbf{0}), (\mathbf{Y},\mathbf{1}), (\mathbf{Y},\mathbf{2}), (\mathbf{Y},\mathbf{3}), (\mathbf{Z},\mathbf{0}), (\mathbf{Z},\mathbf{1}), (\mathbf{Z},\mathbf{2}), (\mathbf{Z},\mathbf{3})\} \\ \Sigma & = & \{\mathsf{a}, \mathsf{b}, \mathsf{c}\} \\ q_{0,1\cap 2} & = & (\mathbf{X},\mathbf{0}) \\ F_{1\cap 2} & = & \{(\mathbf{X},\mathbf{0})\} \end{array}$$

Rather than writing out the 24 cases for $\delta_{1\cap 2}$, I've drawn the state transition diagram in figure 8.

B. Regular Languages are closed under complement:
If B is a regular language, then $\sim B$ is a regular language as well.

1. Hardware intuition: just add an inverter.

2. The formal construction

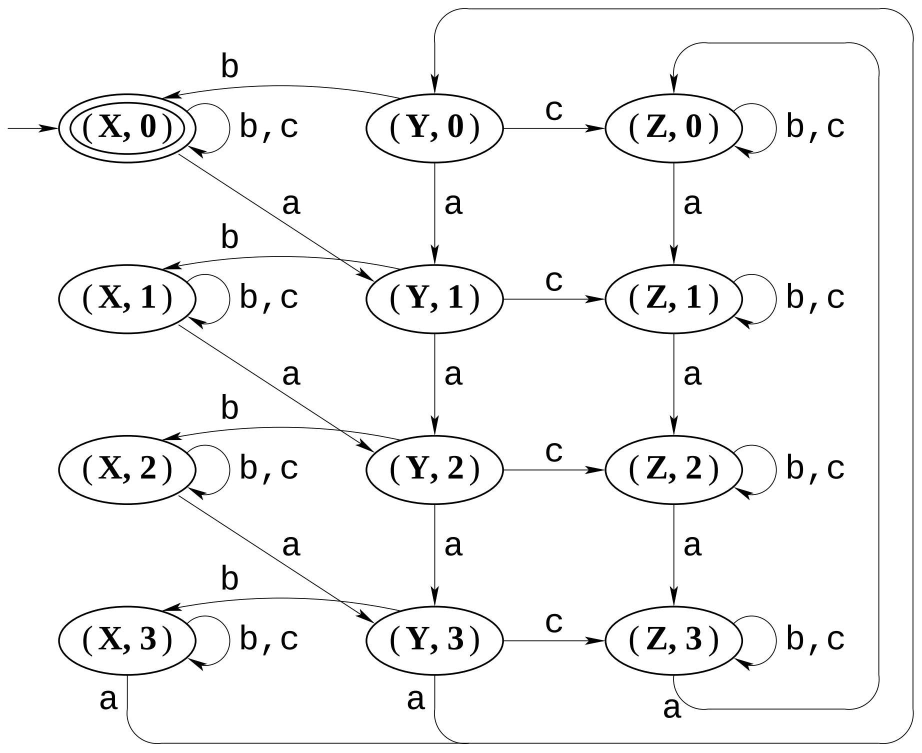

Figure 8: The state transition diagram for a machine that recognizes the intersection of the languages for the machines from figures 2 and 5

C. Regular Languages are closed under union:
If B_1 and B_2 are regular languages, then $B_1 \cup B_2$ is a regular language as well.

1. Intuition from the hardware perspective: just add an OR gate.

2. The formal construction: De Morgan's law given closure under intersection and complement.

D. Regular Languages are closed under arbitrary Boolean operations:
Regular languages are closed under intersection and complement. That provides a "NAND" function, which we have already shown to be universal for implementing Boolean functions.