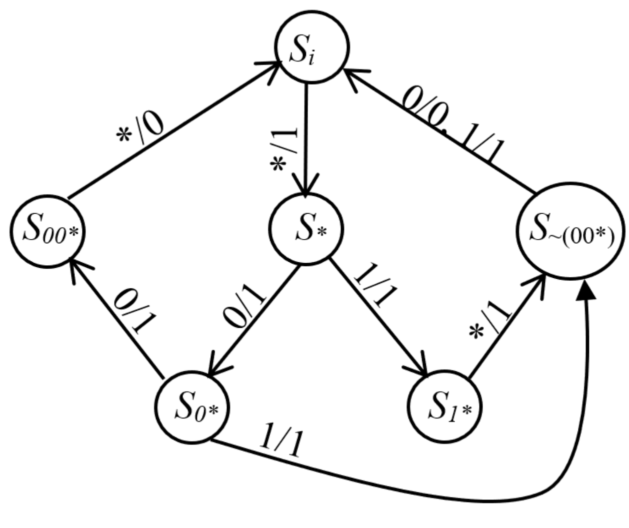
1. From last time
    a. Created minimized FSM for a BCD code checker
        i. BCD code checker outputs a 0 if the previous 4-bit sequence was between 0 – 9, and 1 otherwise
    b. Filled out state table
        i. Used minimized FSM to fill out new table
    c. Assigned binary codes to minimize amount of logic
        i. Try to assign states and their successors adjacent to each other (1 Hamming distance away)
        ii. Create K-map to help us assign codes



Minimized FSM State Diagram

Present State	**Next State**		**Output**	
	x = 0	x = 1	x = 0	x =1
i	*	*	1	1
*****	0*	1*	1	1
0*	00*	~(00*)	1	1
1*	~(00*)	~(00*)	1	1
00*	*i*	*i*	0	0
~(00*)	*i*	*i*	0	1

Minimized FSM State Table

C* \ *AB	**00**	**01**	**11**	**10**
0	*i*	*	0*	00*
1	~(00*)	1*		

Minimized FSM State Binary Code Assignments

    d. Make binary code table from the above
        i. Be careful when assigning values!
        ii. Table states don't line up exactly with table above

Present State	**Binary Code**	**Present State**			**Input**	**Next State**			**Output**
		A	***B***	***C***	***x***	***A'***	***B'***	***C'***	***z***
i	000	0	0	0	0	0	1	0	1
i	000	0	0	0	1	0	1	0	1
~(00*)	001	0	0	1	0	0	0	0	0
~(00*)	001	0	0	1	1	0	0	0	1
*****	010	0	1	0	0	1	1	0	1
*****	010	0	1	0	1	0	1	1	1
1*	011	0	1	1	0	0	0	1	1
1*	011	0	1	1	1	0	0	1	1
00*	100	1	0	0	0	0	0	0	0
00*	100	1	0	0	1	0	0	0	0
	101	1	0	1	0	d	d	d	d
	101	1	0	1	1	d	d	d	d
0*	110	1	1	0	0	1	0	0	1
0*	110	1	1	0	1	0	0	1	1
	111	1	1	1	0	d	d	d	d
	111	1	1	1	1	d	d	d	d

e. Create K-maps for each flip flop based on input and present state in table above
   i. Be careful when entering values from the binary code table!
   ii. Some states are missing and are don't cares, like 101
   iii. Can insert missing inputs into table and write don't cares there to make filling K-maps easier

A'

Cx \ AB	**00**	**01**	**11**	**10**
00	0	1	1	0
01	0	0	0	0
11	0	0	d	d
10	0	0	d	d

$A' = B\overline{Cx}$

B'

Cx \ AB	**00**	**01**	**11**	**10**
00	1	1	0	0
01	1	1	0	0
11	0	0	d	d
10	0	0	d	d

$B' = \overline{AC}$

C'

Cx \ AB	**00**	**01**	**11**	**10**
00	0	0	0	0
01	0	1	1	0
11	0	1	d	d
10	0	1	d	d

$C' = Bx + BC$

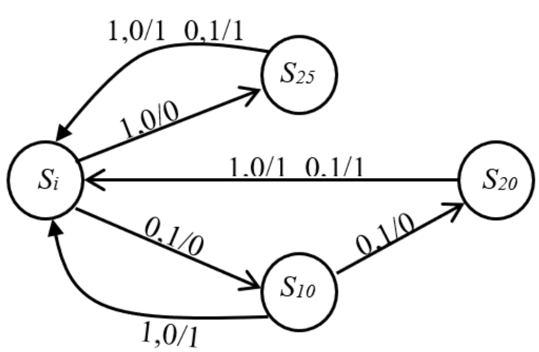
f. Use derivations from the K-maps to design initial combinational circuit
g. Create a K-Map based on flip-flops to determine the output combinational circuit
   i. Since this is a Mealy model, we also use the input
   ii. Don't cares are in same position as K-maps above

$z = B + \overline{AC} + \overline{A}x$

z

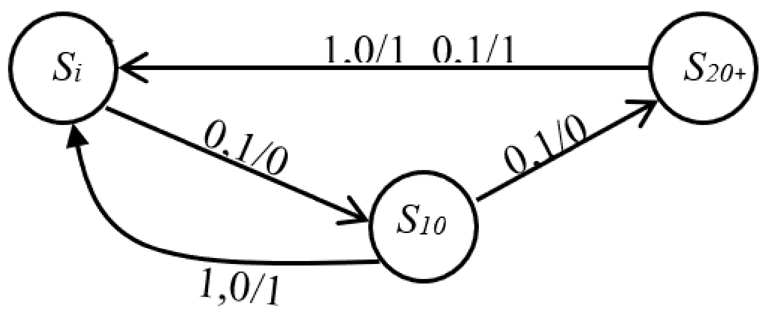
Cx \ AB	**00**	**01**	**11**	**10**
00	1	1	1	0
01	1	1	1	0
11	1	1	d	d
10	0	1	d	d

2. Vending machine example
   a. Design a vending machine that only takes dimes and quarters
      i. Merchandise is dispensed (z = 1) when the sum of the inputs >= 30 cents
      ii. Machine does not give change
   b. x_1 = quarter, x_2 = dime
      i. Assume that it is not possible to input both quarters and dimes simultaneously
      ii. Two input, single output
   c. Will use a Mealy model
      i. Provides for simpler logic in the end
   d. First, create state transition diagram
      i. Inputs are (x_1, x_2) for (quarter, dime)
      ii. Input (0, 0) is omitted, would just cause machine to stay in its current state



e. Next, minimize the number of states using the Partition Minimization Procedure
   i. P_1 = (i, 10, 20, 25)
   ii. P_2 = (i) (10) (20, 25)
      1. 20 and 25 have same k-successors (i for both) so they stay together

f. Draw new state transition diagram, with new state called 20+



g. Assign code words next
   i. ceil($\log_2 3$) = 2 flip flops
   ii. S_i starts in 00
   iii. No way to place all adjacent states 1 Hamming distance away
      1. Do the best we can, though

		A	
		0	**1**
B	**0**	*i*	20+
	1	10	

h. Next, create binary code table
   i. Don't have to create state transition table since this is simple enough
   ii. Will do the same as previous problem and add empty rows to the table for don't cares

Present State	**Binary Code**	**Present State**		**Inputs**		**Next State**		**Output**
		A	**B**	x_1	x_2	**A'**	**B'**	z
i	00	0	0	0	0	0	0	0
i	00	0	0	0	1	0	1	0
i	00	0	0	1	0	1	0	0
		0	0	1	1	d	d	d
10	01	0	1	0	0	0	1	0
10	01	0	1	0	1	1	0	0
10	01	0	1	1	0	0	0	1
		0	1	1	1	d	d	d
20+	10	1	0	0	0	1	0	0
20+	10	1	0	0	1	0	0	1
20+	10	1	0	1	0	0	0	1
		1	0	1	1	d	d	d
		1	1	0	0	d	d	d
		1	1	0	1	d	d	d
		1	1	1	0	d	d	d
		1	1	1	1	d	d	d

- i. Finally, create K-maps from table above
    - i. Be careful when entering values into K-map!
        1. For example, (A, B, x_1, x_2) = 0011 is missing since we can’t have (x_1, x_2) = (1, 1)
        2. Make those inputs don’t cares like normal
        3. We added extra rows to the binary code table
            - a. This way, we know exactly where the don’t cares go

A’		AB			
		00	**01**	**11**	**10**
x_1x_2	**00**	0	0	d	1
	01	0	1	d	0
	11	d	d	d	d
	10	1	0	d	0

A’ = $A\overline{x_1}\,\overline{x_2} + Bx_2 + \bar{A}\bar{B}x_1$

B’		AB			
		00	**01**	**11**	**10**
x_1x_2	**00**	0	1	d	0
	01	1	0	d	0
	11	d	d	d	d
	10	0	0	d	0

B’ = $B\overline{x_1}\,\overline{x_2} + \bar{A}\bar{B}x_2$

z		AB			
		00	**01**	**11**	**10**
x_1x_2	**00**	0	0	d	0
	01	0	0	d	1
	11	d	d	d	d
	10	0	1	d	1

z = $Ax_2 + Bx_1 + Ax_1$

3. Debugging an FSM
    - a. One good way of seeing if your FSM you drew was right is to give a stream of inputs into your FSM
        - i. For example, with the vending machine, give a dime, then a dime, then a quarter
        - ii. See what states you land at, and if those are what you expect
        - iii. Also see what outputs you get, and if those are what you expect as well