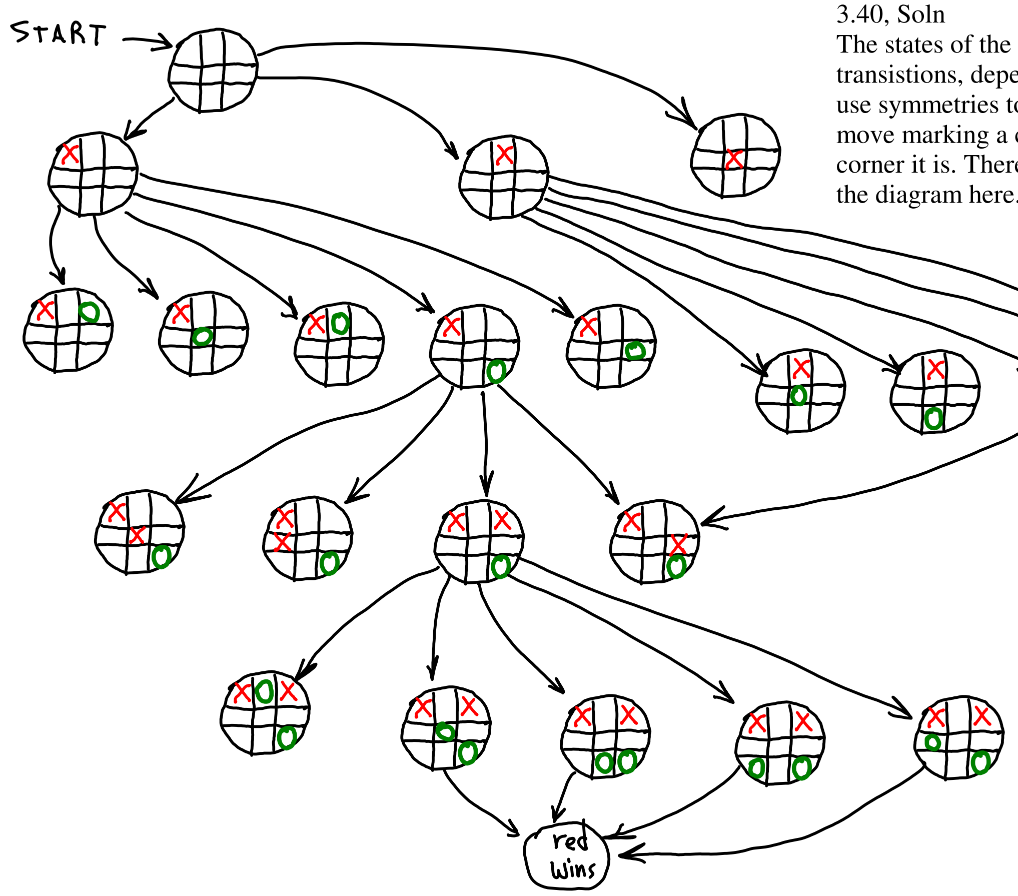
# Lec-5-HW-3-UTM-halting-SOLN

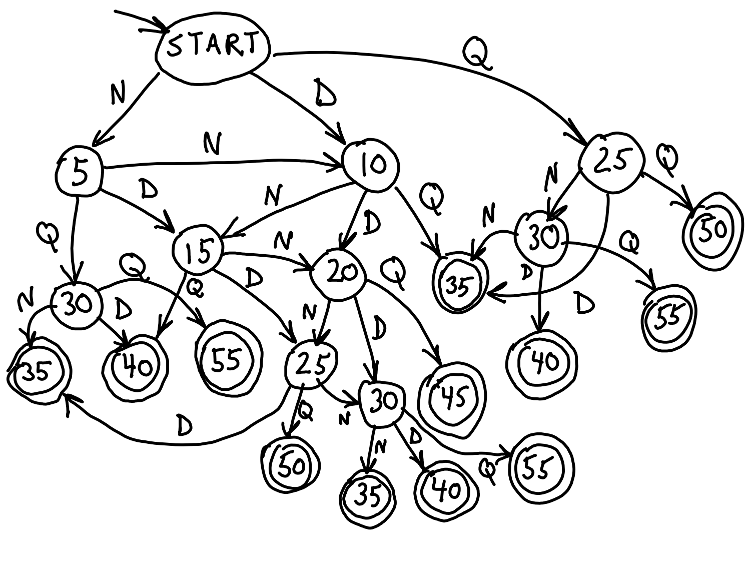
Reading:
TM Lecture notes and PP Chp. 3: 3.6(sequencial machines), 3.6.1(combination lock), 3.6.2(state), 3.6.3(FSM, clock), 3.6.4(danger light example).

Do the following problems from PP: 3.40 (tic-tac-toe FSM), 3.41 (soda machine FSM)



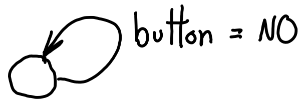
3.40, Soln
The states of the game are connected by state transistions, depending on each player's move. We use symmetries to simplify the diagram, e.g., a first move marking a corner is the same no matter what corner it is. There are too many states to complete the diagram here. A win is a final state.

Note: I've never tried to come up with a succint characterization of the distinct states, when all possible symmetries are taken into account. Extra credit problem!



3.41, Soln
Accepting states are double circled. Final/accepting states transition to START w/o input. You could take this to mean that all states self-loop until button = YES. In the diargram, circles with the same label are the same state.

Legend: "Q", quarter, "D", dime, "N", nickel.

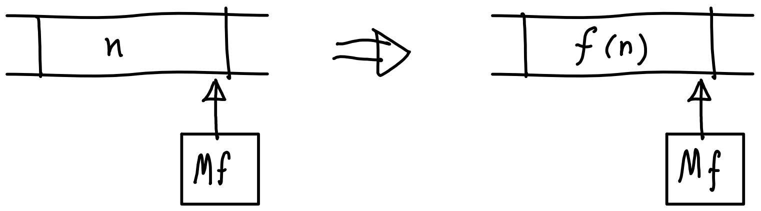


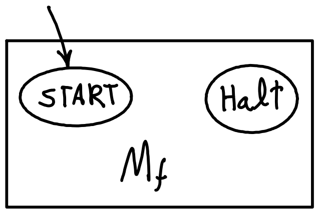
(PROBLEM) --------------------
A computer is in essence a Universal Turing Machine (UTM). We will know we have built a computer if what we build can do anything a UTM can do. What can a UTM do? NB-The question does not ask what TMs in general can or cannot do, the question asks what UTMs do.

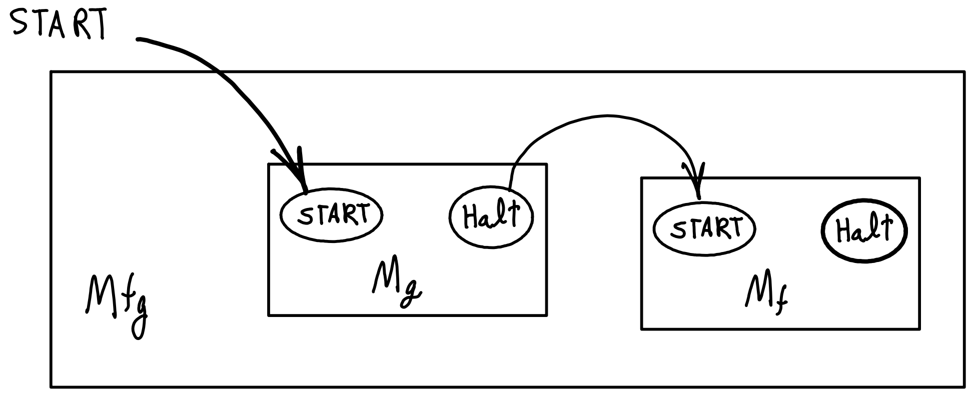
Soln:
A UTM can simulate any arbitrary TM. That's all we need.

(PROBLEM) --------------------
Consider a Turing Machine (TM) with symbol set, { 0, 1}. Recall that we refer to a TM's tape configuration as bounded if all 1s on the tape are within some finite distance from the initial head position. We saw in lecture that every bounded tape configuration can be viewed as an encoding of a binary number, and that a TM can be said to compute the integer function f if, for every n, starting that TM with a tape encoding n will result in its halting with the tape containing an encoding of f(n). Given TMs that compute f(n) and g(n), respectively, describe how to construct a TM that computes f(g (n)). You may use boxes to represent the two macines, indicating how to connect their halt and start states.

Soln
Suppose Mf and Mg are TMs that compute f(n) and g(n), respectively. Suppose they both erase their inputs, leaving the result in its place and halting in their original position. Mfg is composed of one copy of each. Mfg starts in the start state of Mg, which results in g(n) being written on the tape to the left of the R/W head. Mg's halt state is Mf's start state. Subsequently, f( g (n) ) will be written to the tape to the left of the RW head. Mfg's halt state is Mf's halt state.







(PROBLEM) --------------------
A simulating UTM can read descriptions of machines written, i.e., encoded, in a way appropriate for that UTM. We call that encoding method a language. Here we deem the language the UTM's ISA, Instruction Set Architecture. There can be many different UTMs with differing ISAs. Suppose we have two UTMs, UTM-1 and UTM-2, using the same symbol set, but with different ISAs, L1 and L2, respectively.

(a) Can a description of machine M written in L-1 be used as input to UTM-2? That is, if UTM-2 has this on its input tape,

... x desc-L1( M ) ...

where "desc-L1( M )" is a description of M encoded in L1, what would happen? Will M be simulated correctly? (Make a general statement about the consequences of running UTM-2 with that input, assuming the R/W head was appropriately positioned initially: for both UTM-1 and UTM-2 the R/W head should be on the first cell to the right of the input.)

(b) Suppose we have know the design of a translating TM, Tr, that can translate desc-L2(M) to desc-L1(M). We have a UTM-1, and we also desc-L2(M). We want to have UTM-1 simulate Tr, which will translate desc-L2(M) to desc-L1(M), and then have UTM-1 simulate M. What language would we use to describe Tr? Show the input tape to UTM-1. Assume we can restart UTM-1 after the translation is completed and Tr halts, having repositioned its R/W head so that it will simulate M. Tr expects its input to be to the left of its R/W head when it starts.

Soln
(a) A description of machine M written in L-1 would be essentially meaningless as input to UTM-2. UTM-2 would run, and do something, but it would not successfully simulate M. (That is, most likely it would produce gibberish output; although, it could by chance give the correct output.)

(b)
For UTM-1 to simulate the translator machine, Tr, UTM-1 would need desc-L1(Tr) on its input tape. Here is the input tape for UTM-1 to simulate the translator, Tr, and be prepared to simulate M,

x desc-L2(M) desc-L1(Tr)

After Tr had been simulated and translated the description of M, the tape might look like this,

x desc-L2(M) desc-L1(Tr) desc-L1(M)

[NB--Technical details we have ignored are whether M's input data, x, is properly encoded for UTM-1; where on the tape UTM-1 expects to find its input; where Tr expects to put its output; where UTM-1's output tape area is; whether parts of the tape get erased in the process; and how UTM-1's R/W head gets repositioned to begin simulating M.]

(PROBLEM) --------------------
If we can build a Finite State Machine (FSM), a read/write tape, and a read/write head, and connect them to work together as a TM, we can build any TM, including a UTM. If we can build a UTM, we can build a computer. Let's say we are building such a thing. At what point in the process should we make a decision about what our UTM/computer's ISA should be? NB-"Build" means design and actually implement in physical hardware.

Soln
We cannot build the simulator until we have decided how machines will be described. So, designing the ISA has to come first.

(PROBLEM)-------------------
Suppose we have a UTM, T_sim. T_sim's R/W head's correct initial position is on the first blank cell to the right of its input. Since T_sim is a TM, it can be described by some encoding scheme; for instance, by using unary encoding for states and symbols and laying out the rule table as we did in our prior assignment. For instance, if we have this on T_sim's tape initially,

... y desc(X) ...

with the R/W head positioned correctly, T_sim will simulate machine X reading y. Here, "desc()" means the description of a machine in an encoding suitable for T_sim, and "y" is the encoded tape for the simulated machine. Consider this input tape for T_sim,

... y desc(T_sim) ...

a) What are the consequences of running T_sim with this input?

b) Suppose y = [ x desc( M ) ], where M is a turing machine and x is the input for that machine, encoded appropriately as input to T_sim. Explain what results from running T_sim with input,

... [ x desc(M) ] desc( T_sim ) ...

NB--There is a double-layered encoding of input and tapes here. That is, y = [x desc(M)] is encoded as an input tape suitable for the machine described, desc(T_sim), to read. Within that encoding, x is encoded appropriately as M's input tape.

SOLN -------------------
(a) Any TM can be encoded appropriately for T_sim to simulate it; so, desc(T_sim) can also be used as input to T_sim. T_sim will simulate itself reading y as input. That is, the result will be the same as if this were the input to T_sim,

... y ...

except that the output will be in the form of an encoded version of the tape T_sim would have produced.

(b) If T_sim is given the input,

... x desc(M) desc(T_sim) ....

T_sim will simulate itself. The simulated T_sim will simulate M reading x as input. The output would be the same as an encoded version of running T_sim with this input,

... x desc(M) ...

which would simulate M reading x. That result would be the same as if we had run M with this input,

... x ...

That is, simply running M on x will not tell us whether M will ever halt when reading x: we would never know if we had waited long enough to determine that M was not going to halt. In the same way, simulating M reading x is no improvement, nor is simulating the simulation of M reading x.

(PROBLEM)-------------------
We proved in lecture that the function Halts(x,M), which determines whether M reading x halts, is incomputable. For the following function, describe whether it is computable or not and explain your reasoning.

HaltsBefore(x,M,s)

If M reading x would halt within s steps, this machine halts after printing "1"; otherwise it halts after printing "0."

The input tape for the TM, HaltsBefore, looks like this,

... xMs ...

(Note that the input is written "xMs". This is just a convention to indicate that the input tape does not have separators between sections of tape. Consequently, the total input looks like a single string. Also note that we use "M" as shorthand for "desc(M)". The technical details of how our TM could "know" that x ends and descr(M) begins is a bit complicated. Suffice it to say that we can encode a machine in a such a way that it is possible to detect the begin and end of the encoding, regardless of what is adjacent to it. This is called a "self-delimiting encoding".

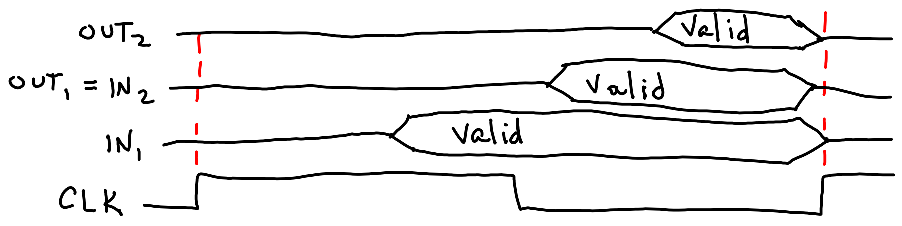
Soln
We will modify some UTM. We know UTMs exist (at least we've made a plausible argument that they should not be too difficult to design.) Suppose our UTM has the following cycle of actions:
1. match current state part of tape to current-state part of a rule in machine description.
If not matched, move to next rule; go to 1.
2. match current simulated input symbol with input symbol of rule.
If not matched, move to next rule; go to 1.
3. copy output symbol to simulated tape, adjusting simulated tape as needed.
4. simulate move, adjusting simulated tape as needed.
5. copy current state code to current-state area.
6. match current-state code to list of halting-state codes; if match, halt; else go to 1.

We modify this machine to have a counter area on its tape, and decrement the counter just before step 1:
0. decrement counter, if negative, print "0" in output area and halt.

We also modify step 6:
if match, print "1" in output area and halt.

This machine will always halt within s steps, where s was part of its input. So, it always halts and prints either "0" or "1". We have demonstrated the existence of a TM to compute the function HaltsBefore(). Of course, this "proof" was conditioned on assuming there is some UTM we can modify. The technical details of an actual UTM were shown by Alan Turing; so, we can assume at least one UTM exists. We argue that it is plausible to assume we could modify his UTM to have an equivalent machine to what we described above.

(Problem)---------------------
Consider two FSMs. The first, M1, sends its output to the second, M2, where it is used as input. M2's output likewise goes to M1. It there any problem with the timing of valid data in this arrangement? Illustrate with a timing diagram.



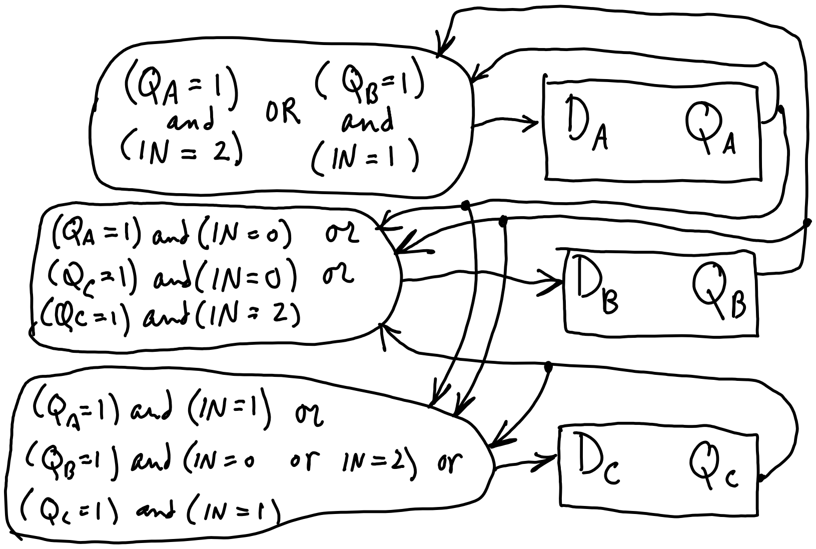
SOLN
Suppose M1 and M2 are Mealy machines. Then M1's input becomes valid sometime after the clock's rising edge occurs. Sometime after that, M1's output becomes valid. M2's output cannot be valid until sometime after that. Notice that all signals become invalid with state change, which occurs with the clock's rising edge. So, M2's output must be valid for a shorter time than M1's input. If M2's output is input for another machine, it may not be valid long enough to be useable.

(Problem)--------------
Here is the rule table for a FSM, M (a Moore machine, ignore output):

| current-state | input | next-state |
|---|---|---|
| A | 0 | B |
| A | 1 | C |
| A | 2 | A |
| B | 0 | C |
| B | 1 | A |
| B | 2 | C |
| C | 0 | B |
| C | 1 | C |
| C | 2 | B |

(1) Pick a 1-hot state encoding.
(2) Show the next state function, F, as a table, using that encoding.
(2) Implement M: show state elements, indicate the next state function as a circle.
Note: The next-state function will actually consist of several functions that implement control branching. Indicate each as a circle with a description of its branching logic.

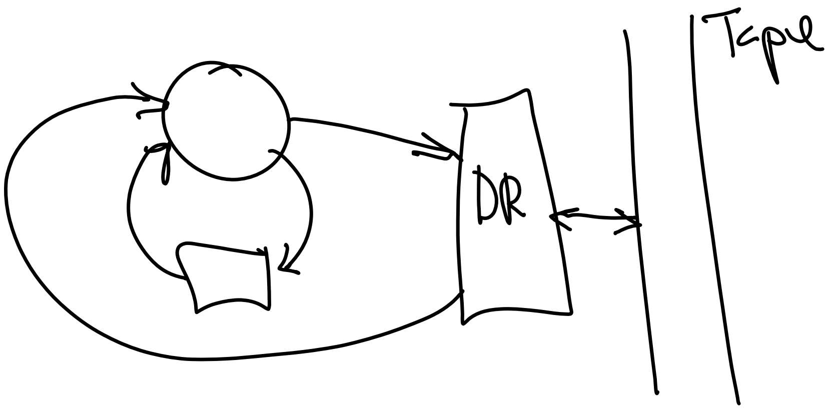
| $Q_C$ | $Q_B$ | $Q_A$ | IN | F: $D_C$ | $D_B$ | $D_A$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 2 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 2 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 2 | 0 | 1 | 0 |



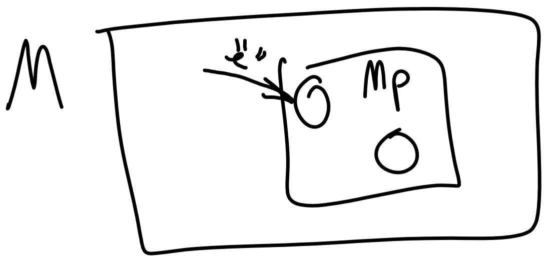
$(Q_A=1)$ and $(IN=2)$ OR $(Q_B=1)$ and $(IN=1)$
$D_A$ $Q_A$
$(Q_A=1)$ and $(IN=0)$ or $(Q_C=1)$ and $(IN=0)$ or $(Q_C=1)$ and $(IN=2)$
$D_B$ $Q_B$
$(Q_A=1)$ and $(IN=1)$ or $(Q_B=1)$ and $(IN=0$ or $IN=2)$ or $(Q_C=1)$ and $(IN=1)$
$D_C$ $Q_C$

(Problem)-------------------
Suppose we want a description language that works for TMs conceptualized in the controller/datapath model. In that model there is a controller FSM to describe, registers and RTL operations to describe, and functional data operations to describe. It was easy to devise a language for describing TMs in the Turing sense of description: it was just a rule table. Suggest a language approriate for composing descriptions in the controller/datapath model. In particular, suggest how to describe RTL operations and controller conditional branching.

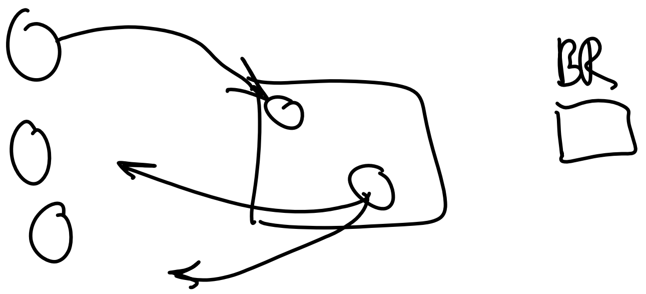
L

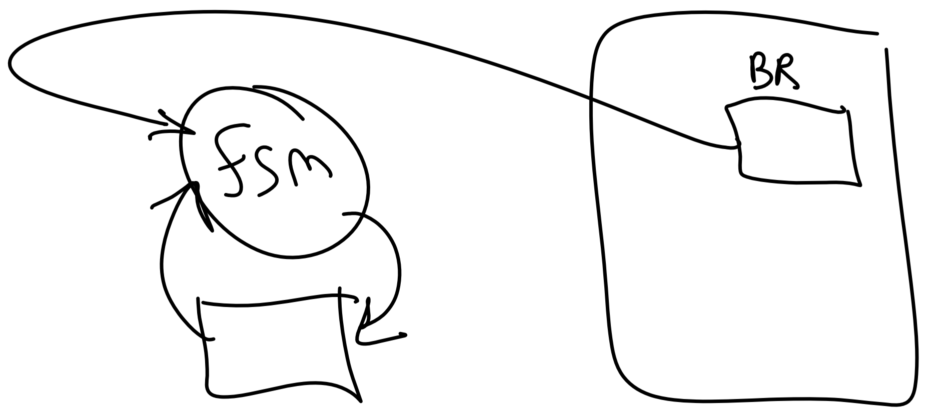


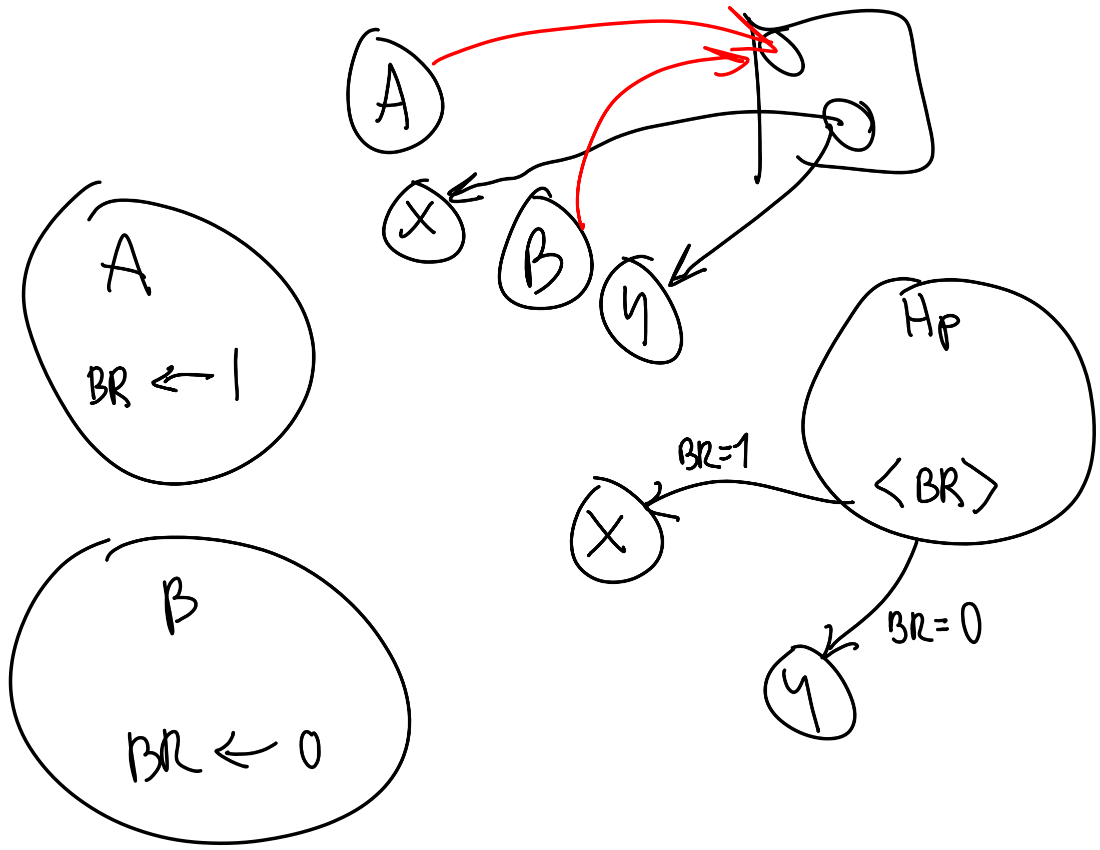
(Problem)------------------------
Suppose we have a TM, M, with a built-in even-parity machine as part of its FSM. Let's suppose its symbol set includes "e": whenever it sees "e" as input, it enters the start state of the parity machine. Suppose it also has a register that serves to record which state it should enter after exiting the parity machine. (We'll think of this in our controller/datapath model.) Using the controller/datapath model, and using RTL for controller states, show two different states, X and Y, of M that cause a transfer to the parity machine. Also show the return from that hardware sub-routine. It returns to state A, if started from state X, and to state B, if started from state Y. Show how M sets up the return, and how the return branching is done. You will need more states than just X, Y, A, and B.



$\Sigma_M = \{$"e",


BR


fsm
BR


A
X
B
y
A
BR ← 1
B
BR ← 0
Hp
&lt;BR&gt;
BR=1
X
BR=0
y