Turing Machine

Things that Think
Spring 2007

Brandon Leffert
Erich Hacker
Andrew Boehm
Turing Machine Construction Kit

For our construction kit project, our team chose to build a Turing machine. The original idea behind a Turing machine consists of four parts: a tape, a read/write head, a state table, and a state register.

**Tape**

The purpose of the tape in a Turing machine is memory storage. In a pure machine, the tape is a piece of paper that can be written on or erased from. For our purposes, we built a series of balsa planks. These planks could be either flipped up or down, which would be the equivalent of an off/on bit. We attached the planks to a balsa board with two racks to help them move back and forth by brass pipe that allowed the planks to rotate. The racks were in contact with two gears each. The gears were cut on the laser cutter out of and were powered by a single Lego motor. We used two balls held down with fishing wire to keep the board on the machine as it became unbalanced as it came to the edges. Our Turing machine had eight planks and hence 8 bits.

**Read Head**

The read head consisted of four break beam sensors. Two pairs of sensors were located on either side of the write head and one of each pair was located on either the front or
pack of the Turing machine. The sensors were equipped with acrylic ramps to ensure that the planks entered the sensors correctly. While we would have liked to place the break beam sensors in the same location as the write head, doing so would have made it impossible to flip the planks over. (The sensors cover the top and bottom of the planks.) Therefore, we took the position of the planks before moving the tape in the correct direction.

**Write Head**

The write head consisted of two servos, one on the front and one on the back of the machine. This was our first time working with servos, and found them to be very useful for our purposes. Since we could control the exact position they ended in, we did not have to worry about over-rotating and breaking the planks or under-rotating and being ineffective. The servos were equipped with a piece of wire with a small square of balsa wood on the tip to lift and flip the planks.

**State Table**

Our state table consisted of 6 parts. It consisted one a pair of each of the following for each state: whether to flip or not, whether to move left or right after flipping, and which state to change to.
after moving. The set of directions to use was dependent on the position of the plank on the write head. The state table existed as a set of arrays stored in the Super Cricket hardware. To program the state table, we created a Windows GUI using C# and the .NET framework. The GUI output a Cricket Logo file that could then be loaded into the Cricket and simply added values to the state table arrays. Our program allowed a maximum of 20 states.

**State Register**

The state register consisted simply as a byte stored in the Super Cricket.

**Super Crickets**

For this project, we used the new Super Crickets to control our sensors and hardware and to store our state table and state register. As opposed to the Handy Crickets, which had two motor controls and two sensor inputs, the Super Crickets have four motor controls, six sensor inputs, and eight servo controls. All of these technologies were available on the Handy Crickets through the built in bus. However, since we were using so many of the new capabilities, it was to our advantage to use the Super Crickets. In this project, we did not have to depend on IR data transfer as we did in the previous project because we were able to attach all of our hardware to one Cricket. The addition of the servo controls made it much easier to flip the planks over and gave us fine motor control. The Super Crickets also had a built in switch counter on sensors C-F.
This allowed us to count the total number of changes to the break beam sensors which allowed us to know the position of the planks to a certain extent.

**Problems and Solutions**

The largest problem we experienced was a lack of fine motor control when moving the tape back and forth. We were unable to make sure that the tape stopped with the planks in exactly the right positions. To help rectify this problem, we were forced to manually move the planks into the correct position after each movement. After each movement, the program would idle until we hit a switch that indicated that the planks were now in position. With finer motor control or smaller servos, we would be able to move the planks into place automatically.

**Expandability**

To greater expand the capacity of this construction kit, we need to add more memory and increase the length of the tape. To do this, we would need a way to connect planks similar to the size of prototype we built. From there, we would build a series of tables to keep the tape on track and balanced. These tables may also have motor controls that connect to the main Cricket to keep heavy tapes moving. With a long enough tape, we could really create an approximation of a Turing machine that could be used to teach students about the history of computation and its power.
Appendix: Cricket source

init

global [state lastpos]

array [flpup 20 flpdown 20 ;0 for don't flip, 1 for flip
        mvup 20 mvdown 20 ;0 for left, 1 for right
        stup 20 stdown 20] ; -1 to end

to init

;Begin state table code
aset flpup 0 1
aset flpdown 0 1
aset mvdown 0 0
aset mvup 0 0
aset stup 0 1
aset stdown 0 0
;end state table code

setstate 1 ;should first state be setable?
servo 1 255
servo 2 0

send moveleft ;Check to see if mvright first?

rightmachine ;lastpos up = 1, down = 0
send lastpos

runmachine

eend

to runmachine

ifelse (lastpos = 0)

[

if ((aget flpdown state - 1)) = 1)
ifelse ((aget mvdown (state - 1)) = 0)

setstate (aget stdown (state - 1))
]

[if ((aget flpup (state - 1)) = 1)
[flipitdown]
ifelse ((aget mvup (state - 1)) = 0)
[leftmachine][rightmachine]
setstate (aget stup (state - 1))
]
if ((state - 1) = -1)
[stop!]
send lastpos
runmachine
end

to leftmachine
send 20
ifelse (not switchc)
[setlastpos 1]
[
  ifelse (not switchd)
  [setlastpos 0]
  [send error]
]
send moveleft
end

to rightmachine
send 30
ifelse (not switche)
[setlastpos 1]
[ifelse (not switchf)
[setlastpos 0]
to moveright
  resett
  resetcountere
  resetcounterf
  if ((not(switche)) or (not(switchf)))
    [a, thatway
     a, on
     waituntil [((countere + counterf) = 2) or (timer > 3000)]
     ifelse (countere + counterf = 2)
    ]
    [a, off
     switchstuff
     output 1
    ]
    [a, off
     output error
    ]
  ]
  if (switche or switchf)
  [a, thisway
   a, on
   waituntil [((countere + counterf) = 1) or (timer > 3000)]
   ifelse (countere + counterf = 1)
  ][a, off
   switchstuff
   output 1
  ]
  [a, off
   output error
  ]
  output error
end

to moveleft
  resett
  resetcountere
  resetcounterf
  if ((not(switche)) or (not(switchf)))
    [a, thatway
     a, on
     waituntil [((countere + counterf) = 2) or (timer > 3000)]
     ifelse (countere + counterf = 2)
    ]
    [a, off
     switchstuff
     output 1
    ]
    [a, off
     output error
    ]
  ]
  if (switche or switchf)
  [a, thisway
   a, on
   waituntil [((countere + counterf) = 1) or (timer > 3000)]
   ifelse (countere + counterf = 1)
  ][a, off
   switchstuff
   output 1
  ]
  [a, off
   output error
  ]
  output error
end
resetcounterd
if ((not(switchc)) or (not(switchd)))
[
  a, thisway
  a, on
  wait until [((counterc + counterd) = 2) or (timer > 3000)]
if else (counterc + counterd = 2)
[
  a, off
  switchstuff
  output 1
]
[
  a, off
  output error
]
]
if (switchc or switchd)
[
  a, thisway
  a, on
  wait until [((counterc + counterd) = 1) or (timer > 3000)]
if else (counterc + counterd = 1)
[
  a, off
  switchstuff
  output 1
]
[
  a, off
  output error
]
]output error
end

to flipitdown
  servo 1 0
  wait 10
  servo 1 255
  wait 10
  off servo 1
end

to flipitup
  servo 2 255
wait 10
servo 2 0
wait 10
offservo 2
end
to switchstuff
  beep
  send state
  wait 100
end
to error
  note 119 30
  stop!
  output -1
end