Homework #3

Sam Quinn

CS321

1/25/2016

1. a) The Turing machine simulator that I ended up using was <http://morphett.info/> the syntax was the most similar to that of what we learned in class.

b) The first simple problem that I examined was that of a binary palindrome. I knew for a fact that a Turing machine could solve a palindrome since I proved it in the past homework. The way that the Turing machine simulator determined if the binary string was a palindrome was the exact same as how I described it in the past assignment.

1. Read the leftmost symbol
2. find the rightmost symbol
3. Check if the rightmost matches the leftmost symbol
4. Delete the symbols that match and go back to the new leftmost symbol and start again.
5. If there is nothing left in the string then you have a palindrome.

|  |  |
| --- | --- |
| **n** | **Steps** |
| 4 | 16 |
| 5 | 23 |
| 10 | 67 |
| 46 | 1083 |

c) The second simple problem that I examined was binary addition. This took in two binary numbers separated by a space and adds them together.

1. Find the space between the numbers
2. Subtract one digit from x and add it to y
3. continue doing the above operations until x is zero

|  |  |
| --- | --- |
| **n + n** | **Steps** |
| 1111 + 0000 | 60 |
| 10101010 + 00110001110 | 334 |
| 1110101010100010 + 1 | 36 |
| 11 + 100101010101010101010 | 1025 |

d) The hard Turing machine problem that I looked at was determining if a binary number is prime or not. This was a cool process to watch with all of the substitute symbols that the machine had to add and even small numbers like 7 would take thousands of operations.

|  |  |
| --- | --- |
| **n** | **Steps** |
| 101 (5) | 1574 |
| 111 (7) | 2606 |
| 1100101 (101) | > 300,000 (Never did finish I left it running all night.) |

2. A Universal Turing machine would need at least two languages. One language for the UTM and the other for the nested TM. This is mandatory since you need instructions for the UTM to be able to handle the instructions of the simulated Turing machine. It is like how our modern day compilers work, we send them our english language and then they have a set of symbols and rules that translate our language into machine code for the actual hardware.

One way that you could setup a UTM is to setup a single tape with a single head. You would be able to store the UTM’s language in the program section of the tape. Store the nested Turing machine in the “M’s tape” section. “M’s tape” section could potentially continue for ever but that is ok as long as we know where the UTM’s language is stored. The last section that you would need to delegate is an area for the UTM to store M’s state, symbol M is reading and actions I labeled this area as “Work Area” below.

|  |  |  |
| --- | --- | --- |
| M’s tape | Work Area | Program |

One thing you should take note of is that the UTM’s language is a fixed size, the size of the language that M (the nested Turing machine) uses could be an arbitrary length. The setup of the UTM’s program is what we could call CODING while the output produced by running M as the DECODING process.

3. Let E be Euclidean geometry and expressed as a *formal system*

as a formal system there needs to be a set of axioms (assumed truths) and a set of rules (of inference). The axioms are dynamic just as lemmas are in proofs you can use previous rules to create new “truths”. New truths are what form the theorems.

You can use a Turing Machine as an **acceptor** for E as listed below:

Input: x

Output: Yes

No if the Turing Machine never halts

Based on the fact that there are certain conditions of E that could potentially never return a definite result (e.g. two parallel lines). There is most definitely an **acceptor** that could be ran with a turing machine.

With the same argument raised above there could not be a **recognizer** that a Turing machine could calculate. Because there could be geometries with arbitrary size where with infinite amount of time not be able to determine a definite answer.

4.

**One head (simulate 2 heads):**

A Turing machine with 2 heads could be simulated very easily by modifying the language that the Turing Machine uses to operate.

For example:

The two tapes could be simulated by stacking the symbols and directions on top of the other like shown above. The top items (, ) would be the instructions for one head while the bottom symbols (, ) would be for the second head.

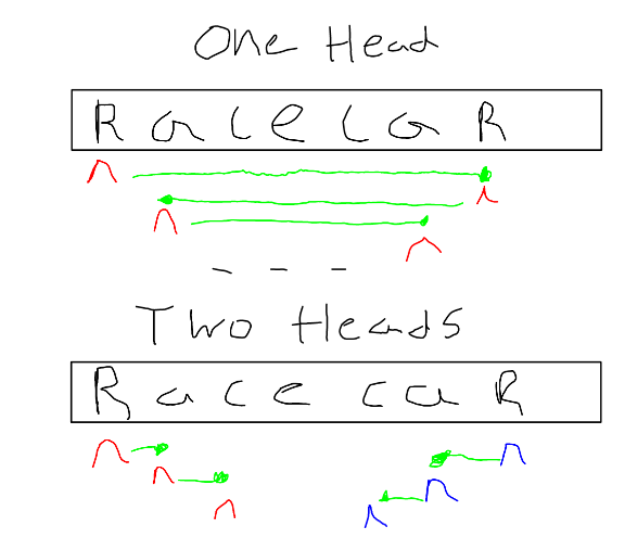
The logic behind a Turing machine that would simulate two read write head could get tricky. You would have to cover cases where the next actions for the read write head would be in opposite directions. You could have the UTM execute the next action that is closest to the current location but then you would risk “starving” the second heads actions. The best way is just to take the approach that just does the top one first every time and then executes the second action. This way neither of the heads would be starved.

5.

**Two heads:**

Having two Turing Machine heads running around on one tape could cause problems. What if the heads collide into each other? What would happen if both the heads try to write to the same block? Who would go first?

Let's assume that we have figured out all of the problems that 2 heads could create. There are plenty of problems that a Turing Machine with two heads could compute much faster than if there were only one head. Consider the palindrome problem. In a standard Turing Machine the head must continually traverse the length of the word to the other side to compare values.



g(n) = n f(n) = n/2

1. As you can see from above a Turing Machine with only one head could compute a palindrome in O(g(n)).
2. If a Turing Machine with 2 heads that share common memory you could compare the letters in O(f(n)) time, because there could be a head on both sides comparing values in parallel.
3. The Turing Machine algorithm to solve a palindrome is deterministic which means we can show that it has to touch every value which gives it O(n) with two heads you would solve the palindrome problem in O(n/2) since the each head would only touch n/2 letters. Therefore we can say for sure that O(n/2) < O(n)

But as we already know that a Turing Machine with only one head can also solve the palindrome problem. But will two head solve any problem that cannot be solved with only one head? No, no matter how many more heads you add to a Turing Machine it will never make problems that are not computable by Turing Machines anymore computable.

6) With Kleene’s theorem saying that any computable function can be computed by a program with at at most one **while loop.** Because a Universal Turing Machine can compute any computable problem and also be simulated with only one while loop this is an accurate statement.

A Turing Machine can be simulated with one while loop as follows:

**While(1) {**

case 0: // Going right

go right until desired block

case 1: // Going left

go left until desired block

case 3: // Same block

stay at the same block

case 4: // Halt

halt and return answer

**}**

While this is very rudimentary, it explains the basic premise behind how you could use Kleene’s Normal Form Theorem to prove that a Turing Machine could run with only one while loop. Furthermore, if a Turing Machine could be simulated with one while loop, and a Turing Machine can solve any computational function you can say that ony computational function could be solved with at most one while loop.