Math 29: Modeling Register Machines

April 8th, 2022

1 Modeling Register Machines

Given a register machine, we want to build a Turing machine which represents the same computation. The main difficulty is that Turing machines are restricted to representing natural numbers with some specific representation, whereas register machines are relatively agnostic. In a register machine, we can always add or subtract one very simply, and numbers are automatically kept separate in different registers. We do not need to worry about how we will represent numbers, or how we will access them. With Turing machines, we need to describe what the input string looks like, and we need to worry about having enough space on the tape to add bits when addition requires it without overwriting other data.

For example, one might attempt to represent the natural numbers in the registers using binary expansions like we have before, and separate each one with a blank cell to demarcate where numbers stop and end. However, the problem arises when we try to apply addition nodes. Suppose, for example, on the tape we have $01 * 1 * 1001101$. If we want to add 1 to the second number, in binary this becomes 10. We would now need to shift bits along the tape to make space for 10 while maintaining the organization of our tape.

Instead, we shall use **unary** coding to represent natural numbers. Given a natural number k, its unary representation is a string of k 1's. We will separate numbers on the tape with a single 0, allowing us to use blanks to signal the start or end of the tape. So, for example, if we want to model the run of a register machine with inputs 3 in R_0, 2 in R_1, 0 in R_2, and 1 in R_3, then we would input 111011001 into our Turing machine. Since only finitely many registers, say k, can be accessed by a given register machine program, we can ensure that there are at least k zeroes in the input to ensure that we do not need to create more register space. (We could do this, but we will avoid making our lives harder.)

1.1 Register Access

Next we describe how to access registers. The fact that the tape is connected and we cannot access indices arbitrarily means that we need to build a module

to move the head into position to operate on a specific register. We start with the specific $k = 2$, then describe how to generalize this process to arbitrary numbers: Consider the following:

- $\langle q_0, 1, R, q_1 \rangle$
- $\langle q_1, 1, 1, q_0 \rangle$
- $\langle q_1, 0, R, q_2 \rangle$
- $\langle q_2, 1, R, q_3 \rangle$
- $\langle q_3, 1, 1, q_2 \rangle$
- $\langle q_3, 0, R, q_4 \rangle$
- $\langle q_4, 1, R, q_5 \rangle$
- $\langle q_5, 1, 1, q_4 \rangle$
- $\langle q_5, 0, L, q_6 \rangle$
- $\langle q_5, *, L, q_6 \rangle$

To begin, state q_0 moves the process right. After each shift, we enter state q_1 and check the current value. If it is a 1, we do nothing and return to state q_0. If it is a 0, then this denotes the separation between the first number and second number. Since we want the third number (the number corresponding to R_2), we move right into the next number. q_2 and q_3 repeat this process, but for the second number. The state is keeping track of how many numbers we've read so far. q_4 and q_5 similarly read through the third number. This time, since this is the number we are interested in, we move left on either a 0 (if there are more registers) or a $*$ (if this is the last register we use) to position the head at the end of register R_2. State q_6 will start the next module based on what instruction our original register machine wants to apply to R_2.

We can generalize this to R_k by creating $2(k+1)$-many registers, where states q_{2i} and states q_{2i+1} handle traversing the i-th register for each i. However, we are not quite finished: once the instruction has been applied, we need to move the head back to the starting point of our string before we can continue, as the above module only works properly if we start at the beginning. To do this, we simply apply the following module:

- $\langle q_0, 1, L, q_0 \rangle$
- $\langle q_0, 0, L, q_0 \rangle$
- $\langle q_0, *, R, q_1 \rangle$

This simple module moves us left until we encounter blank space, then moves right once to reposition at the start of the string. Notice that the state names must be unique for each node on our register machine, so this will add an extra state for each node.

1.2 Addition

We are now ready to describe how to program an addition node. Once we have used the above module to move into position, we want to append a 1 to the current number. However, this will eliminate the 0 separating the numbers, so we need to shift everything else down one cell on the tape. We can do so as follows:

- $\langle q_0, 1, R, q_1 \rangle$
- $\langle q_0, 0, R, q_1 \rangle$
- $\langle q_0, *, R, q_1 \rangle$
- $\langle q_1, 0, 1, q_2 \rangle$
- $\langle q_1, *, 1, q_4 \rangle$
- $\langle q_1, 1, R, q_2 \rangle$
- $\langle q_2, 0, R, q_2 \rangle$
- $\langle q_2, *, 0, q_4 \rangle$
- $\langle q_2, 1, 0, q_3 \rangle$
- $\langle q_3, 1, R, q_3 \rangle$
- $\langle q_3, *, 1, q_4 \rangle$
- $\langle q_3, 0, 1, q_2 \rangle$

First, notice that the symbol at the current location could be any of the three: If we are dealing with R_0 and it contains a 0, it will be $*$. If it is not R_0 but it does contain 0, it will be 0. Otherwise, it will be 1. In all cases, we need to move right and add a 1 using state q_1. Due to the assumptions on our input tape and the use of the above module, we know that the symbol after moving right once will be either a 0 or a $*$, so in both cases we add the necessary 1. In the former case, we then move right and enter state q_2 to shift down the rest of the tape. In the latter case, we are finished and can use the above module to reset the head position.

States q_2 and q_3 will handle shifting the rest of the tape down. In state q_2, we replace the next open space (1 or $*$) with a 0 if we see one. If we overwrite a 1, we enter state q_3 to denote that we now need to write a 1 when we next see an open space. Regardless of how we get there, once we reach state q_4, we can reset the head position to prepare to simulate the next node.

1.3 Subtraction

Finally, we need to describe how to simulate a subtraction node. As with the addition node, we use the register module to position ourselves at the end of the appropriate register. Notice that if we see a 0 or $*$, then the register we are looking to subtract from is empty. So we immediately apply the reset module to simulate the case when the register machine follows the empty node. It is straightforward to modify the repositioning module to overwrite a 1 with a $*$ if we are already at the end of the string.

Therefore, the following module handles the case where we see a 1 and need to subtract, but there are some symbols that need to be shifted down.

- $\langle q_0, 1, 0, q_1 \rangle$
- $\langle q_1, 0, R, q_2 \rangle$
- $\langle q_1, 1, R, q_2 \rangle$
- $\langle q_1, *, L, q_6 \rangle$
- $\langle q_2, 0, R, q_3 \rangle$
- $\langle q_2, 1, R, q_3 \rangle$
- $\langle q_2, *, L, q_6 \rangle$
- $\langle q_3, 1, L, q_4 \rangle$

- $\langle q_3, 0, L, q_5 \rangle$
- $\langle q_4, 0, 1, q_1 \rangle$
- $\langle q_4, 1, 1, q_1 \rangle$
- $\langle q_5, 0, 0, q_1 \rangle$
- $\langle q_5, 1, 0, q_1 \rangle$
- $\langle q_6, 0, *, q_7 \rangle$
- $\langle q_6, 1, *, q_7 \rangle$

First, we overwrite the 1 with a 0, as that is guaranteed to be the next symbol. From here, we now move the head right twice with states q_1 and q_2 to check the next symbol that needs to be moved up the tape. If at any point we run into a blank while moving right, then we have moved up every symbol and need to simply erase the last one, which is handled by q_6. If we do not see a blank, then we move left one space with state q_3 and copy the next symbol into the previous cell with states q_4 and q_5 depending on if it is a 1 or a 0.

Finally, we can apply the reset module to complete the subtraction.

1.4 Wrap-up

Thus, we are able to fully simulate register machines through Turing machines. Given a register machine, each node represents finitely many instructions and states as described above. The connections between nods will be handled by placing the appropriate starting states at the end of the reset modules.

This is a good illustration of the power of Turing machines: while they seem very simple on the surface, with even basic numerical operations taking a fair amount of work to code, they are actually quite powerful.

2 Modeling Turing Machines

The converse is true: given a Turing machine, we can simulate its operation using a register machine. We will not give a complete proof of this, but you will explore the basic ideas on homework 2. We will use a single register to contain the entire tape, another register to contain the location of the head, and then a separate register for each state.

Lemma 1. *Describe in words or diagrams how you would use the value of one register in ternary to represent the input tape.*

Proof. Homework 2 Question 6. □

Lemma 2. *Describe in words or diagrams how you would use a register to simulate a state, its potential actions, and the state changes.*

Proof. Homework 2 Question 7. □