MAT309 Tutorial 9

Clement Yung

7 Nov 2025

Today:

1. Turing machine construction for Diff(n, m).
2. Exp(n, m) is primitive recursive.

Turing machine construction

In the past couple of weeks you've learnt to construct TMs for various basic functions, including those from the recent online quiz.

We'll discuss today how to construct the TM for the function Diff(n, m):

$$\text{Diff}(n, m) = \begin{cases} n - m, & \text{if } n > m, \\ 0, & \text{otherwise.} \end{cases}$$

Attempt 1. If the TM has several cases to consider (e.g. in this case $n > m$ or $n \leq m$), then a good place to start would be to construct a TM that works for a specific case. For instance, let's try to construct a TM that works whenever $n > m$. Let's also not worry if the TM halts. An example would be that:

$$\underline{0}111110111000\ldots$$

$$\downarrow$$

$$\underline{0}111000000000\ldots$$

The idea is actually quite similar to the online quiz. We keep jumping between the 1's in each block, removing them one at a time. So let's try to do that.

M	0	1
1	0R2	
2	1R3	1R2
3	0R3	0L4
4	0L4	0R3

Idea.

1. State 1: We move the scanner to the first block of 1's.
2. State 2: We traverse through the 1's. Once we hit the end of the 1's (i.e. the first 0), we move the scanner to the right (i.e. the first 1 in the second block), and move on to state 3.
3. State 3: We change the 1 to a 0, then we start traversing through the 0's leftward. We denote that as state 4.
4. State 4: Once we hit the first 1 from the right, we change the 1 to a 0, then start traversing through the 0's rightward. We put that back into state 3.

Issue. This TM doesn't work if $n \leq m$. Remember that we need to get $01000\ldots$ if $n < m$.

We have two problems to solve:

1. We need to ensure that the leftmost 1 is not converted to 0.
2. We need to delete all remaining 1's on the right.

Attempt 2. We have to check that whether a 1 on the left block that we're trying to change is the final 1. That is, the next cell on the left is a 0.

M	0	1
1	0R2	
2	1R3	1R2
3	1R3	0L4
4	0L4	1L5
5	0R7	1R6
6		0R3
7		1R8
8	0R8	0R8

Idea.

1. State 4: When we hit a 1, instead of changing it to 0 and start moving right, we go one step further to the left to check if it's the final 1 remaining (i.e. state 5).
2. State 5: If it's not the final 1, go to state 6. If it's the final 1, go to state 7. Either way, move to scanner back to the 1 we're concerned with.
3. State 6: It's not the final 1, so change it to 0 and go back to state 3 (i.e. start traversing the 0s).
4. State 7: It's the final 1, so we keep it at 1, and change all remaining numbers to 0 (state 8).
5. State 8: Keep going right and change everything to 0!

Issue. This TM doesn't halt.

Attempt 3. Let's make this TM halt. We achieve this by placing a special marker at the end of the tape.

M	0	1	2
1	0R2		
2	0R9	1R2	
3	0R3	0L4	0L11
4	0L4	1L5	
5	0R7	1R6	
6		0R3	
7		1R8	
8	0R8	0R8	0L11
9	2L10	1R9	
10	1R3	1L10	
11	0L11	1L12	
12		1L12	

Idea.

1. State 2: Before we start erasing the 1s, we head to state 9.
2. State 9: We head to the back of the second block of 1s. Once we reach the end (hit a 0), we place a 2, and head to state 10.
3. State 10: We head back to the 0 between the two blocks of 1s, head to state 3 and start erasing the 1s.
4. State 11 & 12: We move the scanner back to the leftmost cell.
5. State 3 & 8: When the scanner goes rightwards, if it sees a 2 then it changes the 2 to a 0 and halts the TM.

Issue. None! Yay$\sim$

Primitive recursive functions

Let's recall how primitive recursive functions are constructed:

Definition

The *initial functions* are:

1. (Zero) O, the 1-place function s.t. $O(n) = 0$ for all $n \in \mathbb{N}$.
2. (Successor) S, the 1-place function s.t. $S(n) = n + 1$ for all $n \in \mathbb{N}$.
3. (Projection) π_i^k where $1 \leq i \leq k$, the k-place function s.t. $\pi_i^k(n_1, \ldots, n_k) = n_i$ for all $(n_1, \ldots, n_k) \in \mathbb{N}^k$.

Definition

Let $k \geq 1$, g is a k-place function, h is a $(k+2)$-place function. A $(k+1)$-place function f is obtained from g and h by *primitive recursion* if f is defined as follows:

1. $f(n_1, \ldots, n_k, 0) = g(n_1, \ldots, n_k)$.
2. $f(n_1, \ldots, n_k, m+1) = h(n_1, \ldots, n_k, m, f(n_1, \ldots, n_k, m))$.

Consider the exponential function $\mathrm{Exp}(x, n) := x^n$.

Fact

Exp *is a 2-place primitive recursive function.*

The rest of the slides will be used to prove this fact.

Recall from the notes that the following two functions are primitive recursive:

$$\mathrm{Mult}(n, m) := n \cdot m,$$

$$\mathrm{Pred}(n) := \begin{cases} n - 1, & \text{if } n > 0, \\ 0, & \text{if } n = 0. \end{cases}$$

We shall use these two functions to define Exp.

We need to find a 1-place function g and a 3-place function h such that Exp is obtained from g and h via primitive recursion. We have:

$$\mathrm{Exp}(n, 0) = 1 = (S \circ O)(n),$$

i.e. we take g to be the function $S \circ O$. And:

$$\begin{aligned} \mathrm{Exp}(n, m+1) &= \mathrm{Mult}(n, \mathrm{Exp}(n, m)) \\ &= (\mathrm{Mult} \circ (\pi^3_1, \pi^3_3))(n, m, \mathrm{Exp}(n, m)), \end{aligned}$$

i.e. we take f to be the function $\mathrm{Mult} \circ (\pi^3_1, \pi^3_3)$.

Note that Exp is not obtained from O and Mult via primitive recursion, as Mult is not a 3-place function.