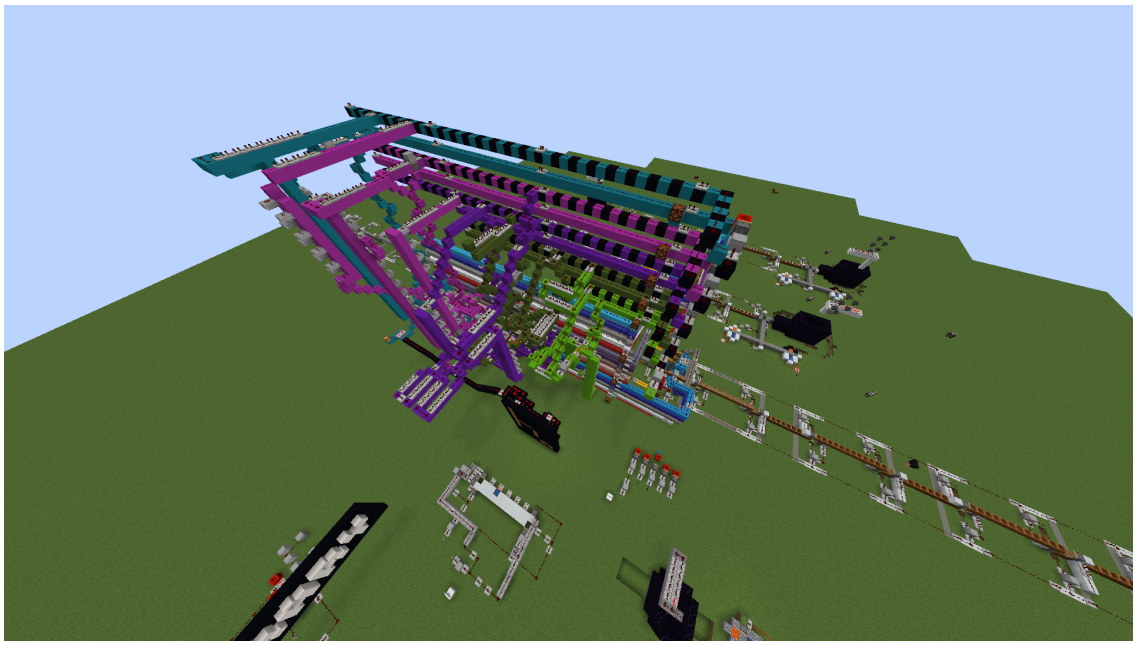
Beregnelighed og Logik
Hand-in 5: Minebyg

Laust Volsgaard Nielsen
Oscar Friberg Jacobsen

February 2th 2025

Contents



1 WE BULDED A DETERMINISTIC MULTI-TAPE TURING MACHINE IN WHAT??!!?!

1.1 Some Unfortunate Restrictions

As it is quite difficult to implement an infinite tape in Minecraft (Minebyg), our implementation is a linearly bounded turing machine. The TM can produce the product of two unary numbers $n, m \leq 30$ with a product $nm \leq 167$. The first bound on the respective input sizes is provided by the size constraints on the two auxiliary tapes for storing the inputs, the second constraint on the output is provided by the size constraint on the main tape which stores the result.

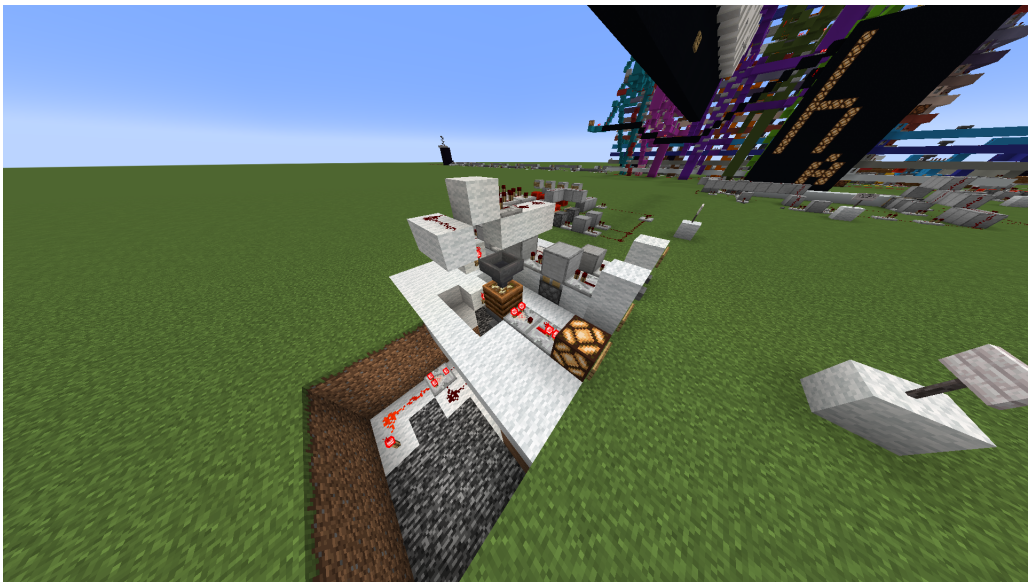
Increasing these bounds can be done by manually extending the tapes, although the length of the belts majorly challenges the computation needed, as the number of chunks needed to be loaded increases linearly with the size of the input (if chunks are loaded manually by commands) or even quadratically (if loading more chunks is done by increasing render distance). The bottleneck of loading chunks is further described in The User Guide.

For the sake of abstraction our supply of TNT and pumpkin pies is finite, but by implementing a TNT duplicator and a pumpkin farm, we could easily extend those ressources indefinitely. Making the TM implementation survival friendly.

1.2 Read/Write

The implementation in Minecraft (Minebyg) works by the fact that comparator in connection to a composter produces a signal if the composter is non-empty, thereby giving us the encoding of our alphabet:

$$1 := \text{Full Composter}$$
$$\Delta := \text{Empty Composter}$$



We can empty a composter only when it is full, by cutting off the signal to a hopper placed beneath it, and we can fill it by cutting off the signal to hopper placed above it for a duration long enough to dispense 7 pumpkin pies. Thereby providing our mechanisms for $1/\Delta$,Δ/Δ, $\Delta/1$, $1/1$.

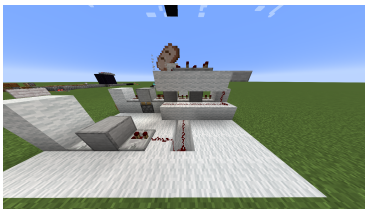


Figure 1: Half of signal extender, for pulse generation

Figure 2: Second half of the signal extender

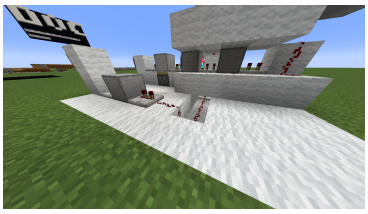
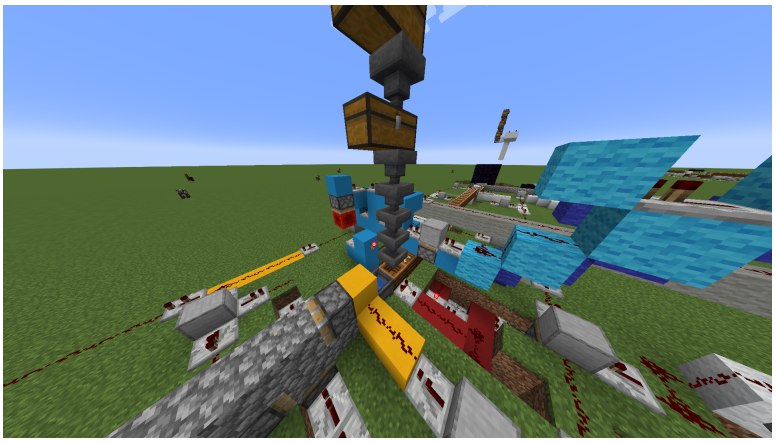


Figure 3: Line to the bottom hopper

Pumpkin pie is used as compost as it and cake are the only items to gurantee an increase in composter level, thereby allowing a simple timing mechanism to fill the composter. Cake is neither autocraftable due to milk nor is it stackable making pumpkin pie the better choice.

The implementation differs only slightly in wiring to accommodate the belt, along with the implementation of the special end of belt character, the cauldron. A cauldron is used mostly for visual distinction as it functions identically to a composter with a single pie in it.



The end of belt character is detected by the circuit which only activates with a weak signal which is still greater than zero, because the composter is only half full during the write of a 1 a noise eliminator is used which only lets through a sufficiently long signal. Although the end of tapes can be detected, halting the TM has not been implemented.

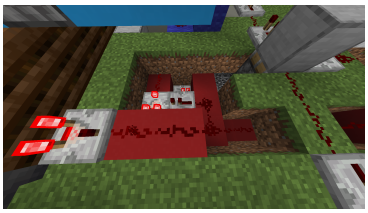


Figure 4: The result of a composter with a single pie in it

Figure 5: The result of a full composter

Figure 6: The noise eliminator

1.3 Belt

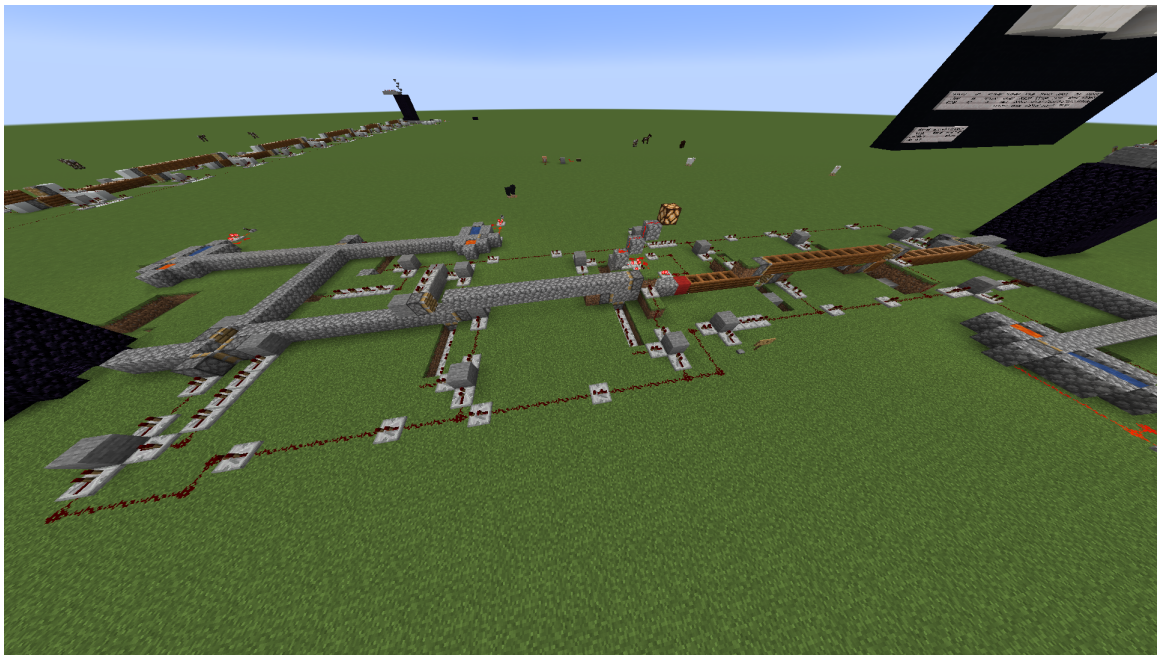
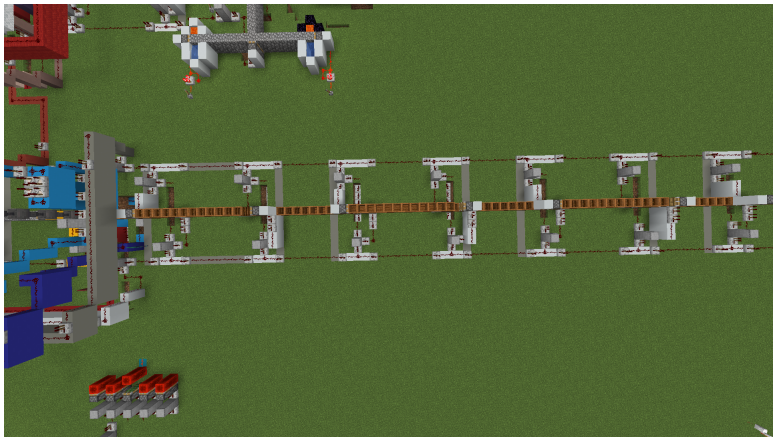


Figure 7: The original prototype.

The belt uses a completely modular design, which allows for increasing the length by simply repeating the same build, as seen in with the repeated build within the grey lines



To the immediate left of the reader/writer lies the special yellow section, which holds an extra block, such that this part can and always must be the first section to move a block in the desired direction. This is supported by the whole build of the belt, by the entire section to the right of the build maintains an airgap on the right side of the belt and vice versa with the left side.

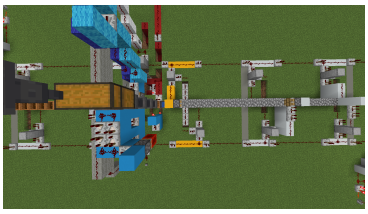


Figure 8: Overview of the yellow section

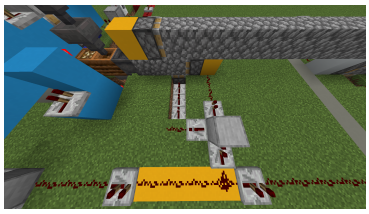


Figure 9: Timings for one direction

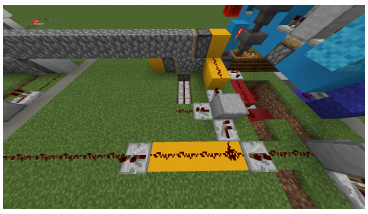
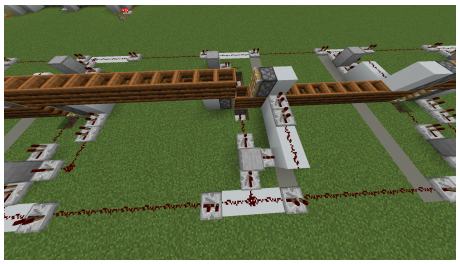


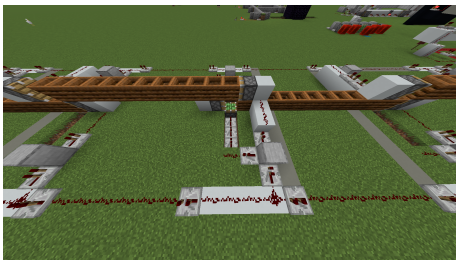
Figure 10: Timings for the other direction

Both directions for the yellow section have identical timings

the purpose of this is belt movement is propogated outwards so while finishing an entire left/right belt movement requires linear time in the length, the movement doesnt need to finish for any actions (including more movements in both directions) so the movement time is constant.

The two different sides of the grey sections have different timings, the both use the sticky piston in opposite ways using a 1 tick pulse





The timings are *rotationally* symmetric around the yellow belt and the grey belt is also rotationally symmetric While the belt could be implemented by having the belt wrap around, and it would likely perform better. We felt having the belt actually come to an end was more in the spirit of Turing so the belt uses cobblestone generators to generate blocks so the belt can still be pushed while destroying excess cobblestone in the other end of the belt using tnt.

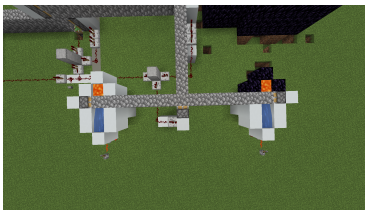


Figure 11: An elevated cobble generator for when the belt ends elevated

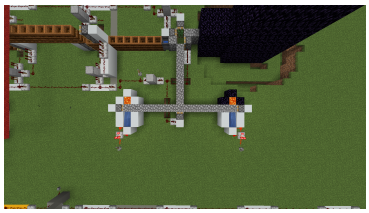


Figure 12: A cobble generator at ground level

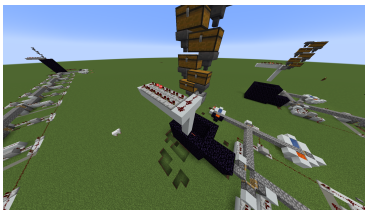


Figure 13: The tnt is timed so there is only one in the box at a time

The timing is different on the ends because the cobblestone is immediately pushed toward the tnt.

1.4 Main bus

Figure 14: The monster

All 5 of the io for each of the 3 belts (a total of 15) have been wired up to the main bus

For each of the belts there is a red wire in the bottom which denotes the state of the current block the tape head is looking at, the white wire denotes R movement, the Grey L movement, Blue denotes writing Δ and light blue denotes 1

Figure 15: The wires from the first belt

Figure 16: The wires from the second belt

Figure 17: The wires from the third belt

To denote the differene between the wires they use different textures while still using the same colors

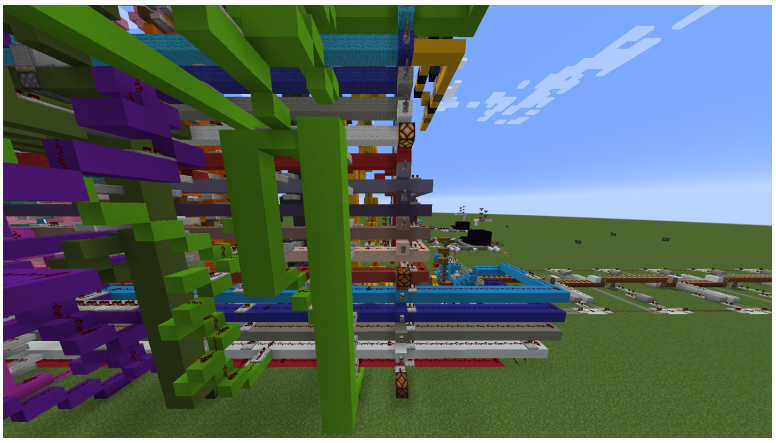


Figure 18: The belt wires all on top of each other

The state is represented by which one of the redstone blocks is in a position to power the wire

Each state has its own color, and a striped wire of the same color which when powered exits the state if already in it and enters the state if not.

Figure 19: The States

The TM was built first building the left side then making the right side and they only interact when the blue state on the right side causes it to enter the lime green state on the left side. As such it is an acceptable sacrifice when they cannot interact with each other.

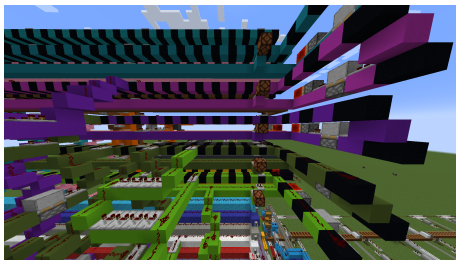


Figure 20: The left side state wires

Figure 21: The right side state wires

1.5 DFA for the belts

We were a little bit lazy so we hardwired the multiplication algorithm instead of making a UTM.

When reading the turing machine drawing you might notice some superflouos states, the reason for this is that we built the left half of the TM before reading the assignment. As such we also have the extra lime green state between state q_5 and q_6 such that the transition from q_5 to lime green doesnt change the belts and it is from lime green to q_6 the belts are moved to the right.

The states then go

- yellow: q_0
- Grey: q_1
- Orange: q_2
- Brown: q_3
- Pink wool: q_4
- Blue wool: q_5
- Lime green: $NULL$
- Dark green: q_6
- Purple: q_7
- Pink: q_8
- Blue: q_9
- Big h_a sign: h_a

Figure 22: h_a

Each state has a number of transitions associated with it, they all use the same color as the state, so differentiating between states is quite easy however differentiating between transitions is a bit more tricky as we will see later.

The theoretical implementation of the transitions is relatively straightforward as the decision to enter a transition can be thought of as a large $\wedge$ ie q_0 is

$$B_1 == \Delta \wedge B_2 == \Delta \wedge B_3 == \Delta \wedge q_0$$

and in practice it is trivial to make an OR gate and a NOT gate so we use the fact that

$$\neg(\neg q \vee \neg p) \vdash q \wedge p$$

Sadly though, the practical implementation is quite a bit more of a hassle

Whenever a transition is chosen. before doing anything else the transition has to exit the current state, as otherwise we risk the state taking another transition despite already having done so. Only then can there be read and or written, doing so requires 3 seconds of delay. After which the tapes can be ordered to move and for good measure an additional 3 seconds are waited before finally entering a state.

Furthermore if a wire has been activated it blocks all other attempts to activate it as a wire has to be turned off to activate it again, so all the signals everywhere are sent through tick generators.

The first example is the lime green state which has a single $\Delta\Delta\Delta/\Delta\Delta\Delta SSS$ transition.

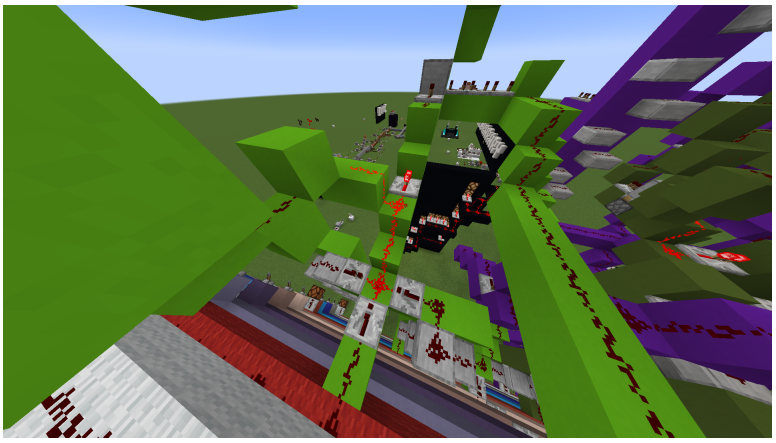


Figure 23: The logic of the lime green state

Since all of the belt wires have to evaluate to false (delta implicitly evaluating to false), there is no need to negate them, as double negation cancels out.

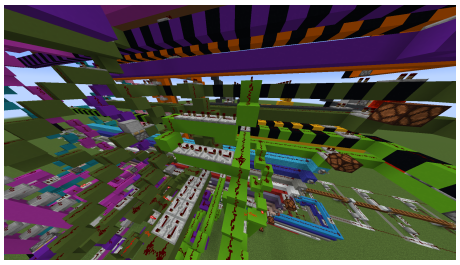


Figure 24: The delay before entering the next state

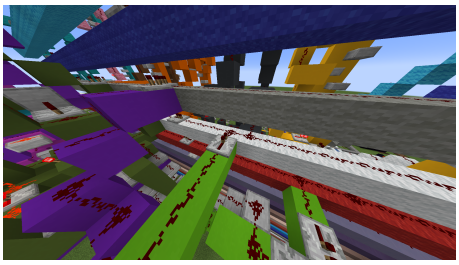
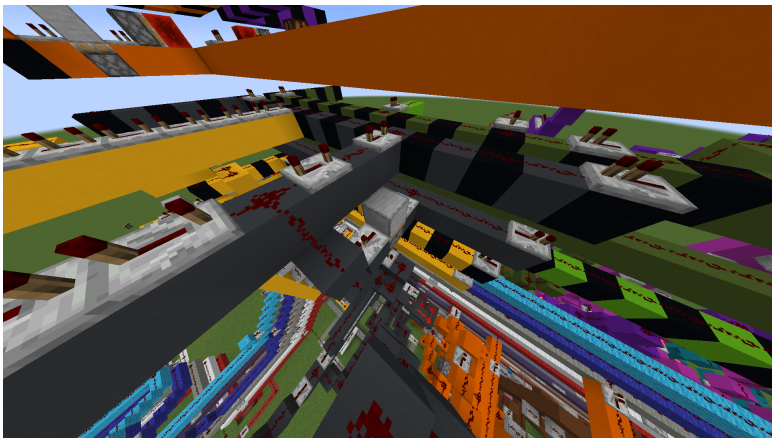


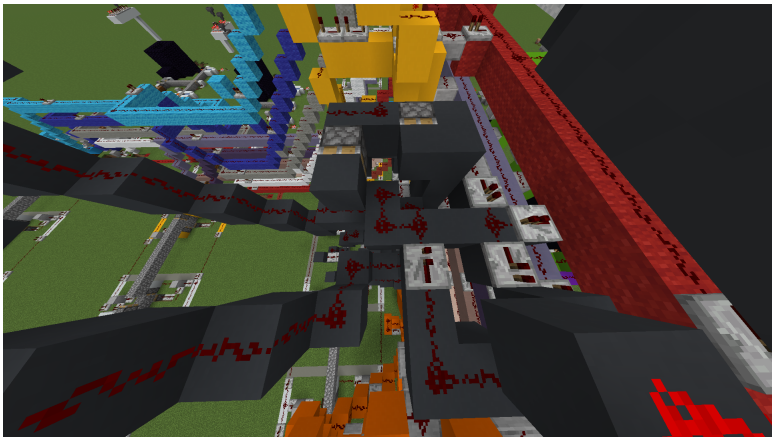
Figure 25: The transition moving the third belt

With the later transitions the state is automatically exited regardless of whether there is a valid transition out of the state. This just makes wiring easier.



And the reason differentiating between transitions being difficult is the fact that almost every state wherein one transition leads back to itself and the other leads to a different state, there is only a single $TRUE/FALSE$ evaluation deciding whether to do one transition or the other. As such the circuit pictured below does exactly that, blocking one or the other wire. while still maintaining all the other requirements.

This trick is very useful in making the transitions as the wiring work for each transition is cut in half.



The purple transition is a bit of a middle ground between the two approaches, still only exiting the state when it is certain that it will perform a transition without necessarily knowing which one it will perform, and using the trick to get two different transitions.

A final comment, the pink state $q - 8$ has two different loops which both do the same thing so it is implemented with a single loop which just does not care about the third belt.

1.6 User Guide

Before starting the world you must change the settings of the JVM minecraft runs in such that it has a minimum of 4GB ram.

When running the TM for the best consistency you should increase both render and simulation distance to at least 22 chuncks and increase you fov to quake pro, then stand at aproximately $38/-7/-6$ and try to see as much of the long belt as possible

As this is in fact is a TM you will manually have to set the unary inputs, as well as read them manually when the computation is done. **However the computation will of course only work after you pay your sacrifice at the bread alter.**

To set the imput, leave the first composter in the main tape blank, fill up the next n composters, leave the next blank, then fill up the next m composters, partially filled composters are UB. To read the output, simply read how long the output word is, i.e. the amount of filled composters, on the long belt this can be done by standing atop the last full composter and adding 4 to your z coordinate.