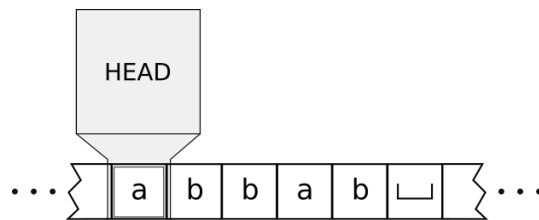# Converting Fancy Turing Machines to Simple Machines

1 Time vs. Space Efficiency of TMs

See also the video lecture recording: Time vs. Space Efficiency on Canvas.

Sometimes there's a trade-off between time efficiency and space efficiency of Turing machines. Consider the following TM configuration:
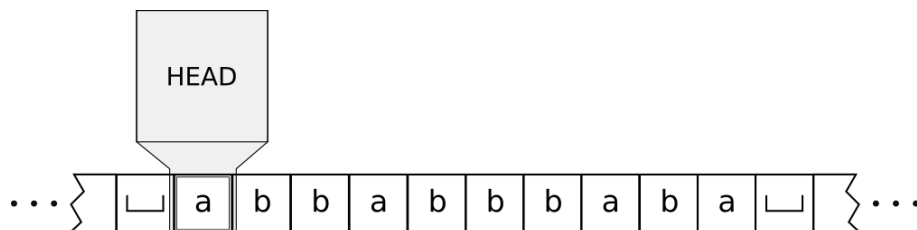

We know that the same configuration can also be represented as: ȧbbab

Question: How do we find the space efficiency (complexity) of the above TM?
Short Answer: We can count all the locations which either have a *non-blank character* at some point in time or the *head* at some point in time.

Suppose we start with an input block of size 10, and the head is to the left of the input block. The program runs for a 1000 steps. *What is the maximum space usage?*

The same configuration can also be represented as: ␣ȧbbabbbaba␣

We know that a TM needs to use two steps (Write x and move Right), on the average, to make use of a new tape cell. Therefore, we approximate the space usage of the above TM as ≈ 510 cells. However, the space usage of the same TM would be considered as ≈ 1000 cells, if it just kept on moving to the right in each step. The least space usage for the same TM would be 10 cells, if the TM does not move *outside* the input during the execution of the program.

In general, the worst case space efficiency is proportional to the execution time or less. So, if we have the time complexity as $O(n^2)$ for a given TM, then the space complexity is also within the same upper bound *i.e.* $O(n^2)$.

2 Turing Machines using Auxiliary Characters

See also the video lecture recording: Turing Machines using Auxiliary Characters on Canvas.
Suppose that, in addition to the input alphabet and the blank, we have a finite set of auxiliary characters. A program assumes that initially these do not appear, and must guarantee that finally they don't appear, but in the middle of execution they can be used. For example, suppose the input alphabet is $\{a, b, \llcorner\}$ and the auxiliary alphabet is $\{c, d\}$. This means that we can have instructions Write a, Write b, Write c, Write d and Write $\llcorner$, and each Read instruction has 5 possible outcomes (a, b, c, d and $\llcorner$). We can now write a more straightforward (but still quadratic time) program for copy-reverse problem, using c to indicate that character a is currently being copied, and d to indicate that character b is currently being copied (rather than using the blank, as previously). For example, here is a fancy TM (a TM including auxiliary characters) for the copy-reverse problem discussed earlier.
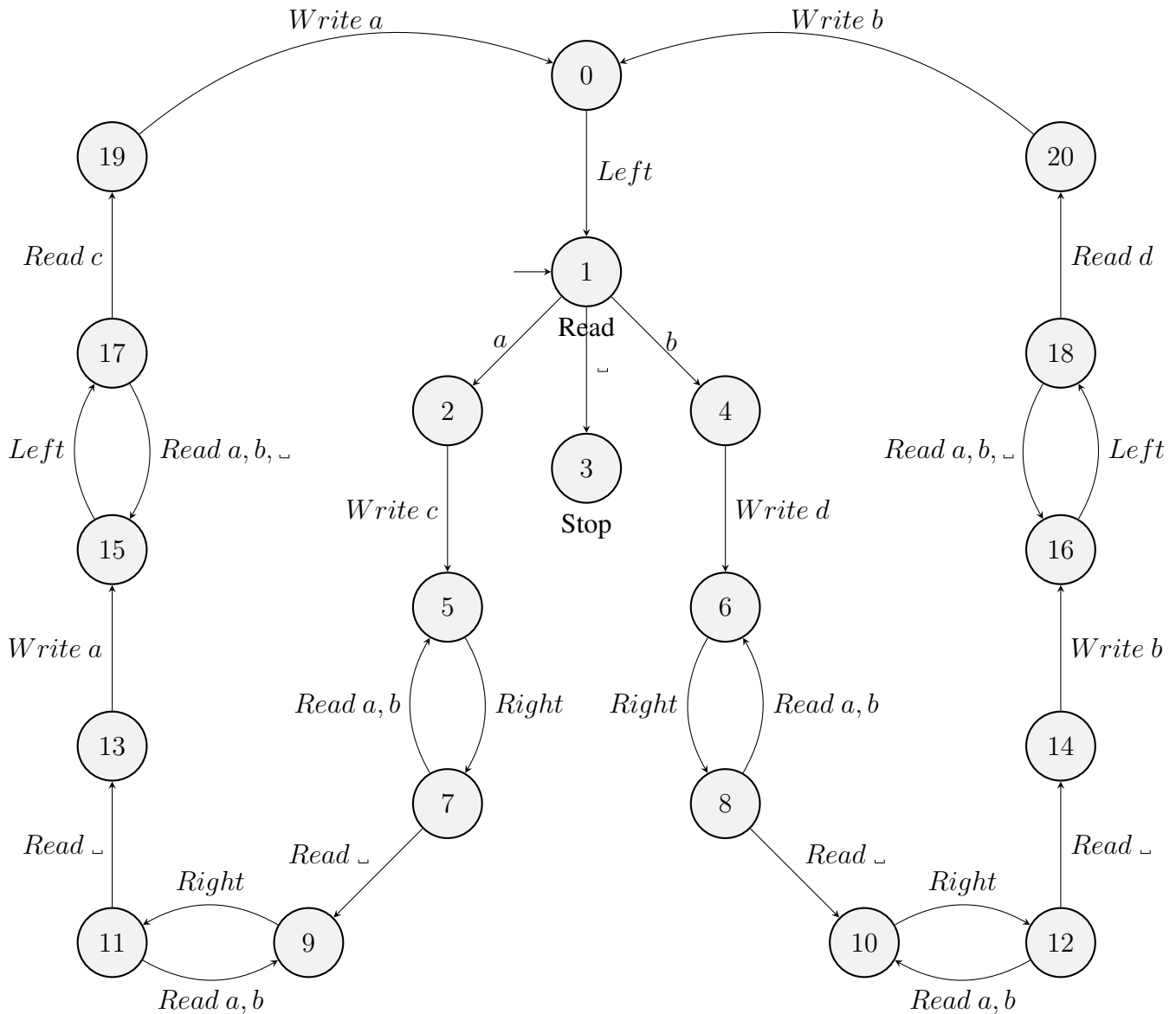
We're going to learn a general method for converting a fancy TM to an ordinary TM. This general method will involve the following steps:

1. Give a relation between the tape configurations of fancy TM and the tape configuration of the simple TM. In simple words, define a way to represent the tape configuration (i.e. tape contents plus head position) of the fancy TM as a configuration of the simple TM. This is a *creative step!*

2. Give a program to convert the initial configuration of the fancy TM into a corresponding configuration of the simple TM (The "Setting-up" program).

3. For each instruction of the fancy TM, show how to simulate it on a simple TM. In other words, show how to "perform" each step of the fancy TM on a simple TM (The "Simulating" programs).

4. Give a program to convert the result of simple TM to the fancy TM result (The "Finishing" program).

Typically all of the above programs are polynomial time.

Key Point: *A polynomial time program on a Fancy Turing machine can be converted into a polynomial time program on a Simple Turing machine.*

We will now show the details of the above steps in the following sections.

2.1 Defining Relation between Fancy & Simple Tape Configurations

See also the video lecture recording: Relation between Fancy & Simple Tape Configurations on Canvas.
Let's say we have a fancy TM using the input alphabet $\{a, b, \llcorner\}$, and auxiliary characters $\{c, d\}$. It means that we **assume** only a, b and ␣ at the start of TM, but we are **allowed** to use c and d in the middle of execution. However, we must **ensure** that only a, b, ␣ are present in the output of the fancy TM at the end. When defining the relationship between fancy and simple tapes, the key point will be to assume that the fancy and simple tape configurations are related at the start of each simulation step, we may violate this relationship in the middle of the step, but we must ensure it at the end.

In this step, we shall define a relation between the fancy and simple TMs' tapes configurations. Lets consider the following fancy tape configuration:

$$\text{ac\.bccda}$$

Note: *Obviously, the above configuration is not the initial configuration of the fancy TM, as the auxiliary symbols can only appear on the tape during the execution, but not at the start or at the end.*

In order to represent the fancy TM's tape configuration as a simple TM's tape configuration, we define the following relation (which is a *creative suggestion*):

Character on Fancy tape	Represented on Simple tape
a	aa
b	bb
c	ab
d	ba
␣	␣␣

The simple TM's head position will be the leftmost of the two characters representing the fancy TM's head position. For example, the fancy tape configuration:

$$\text{ac\.bccda} \tag{1}$$

is represented as the simple tape configuration:

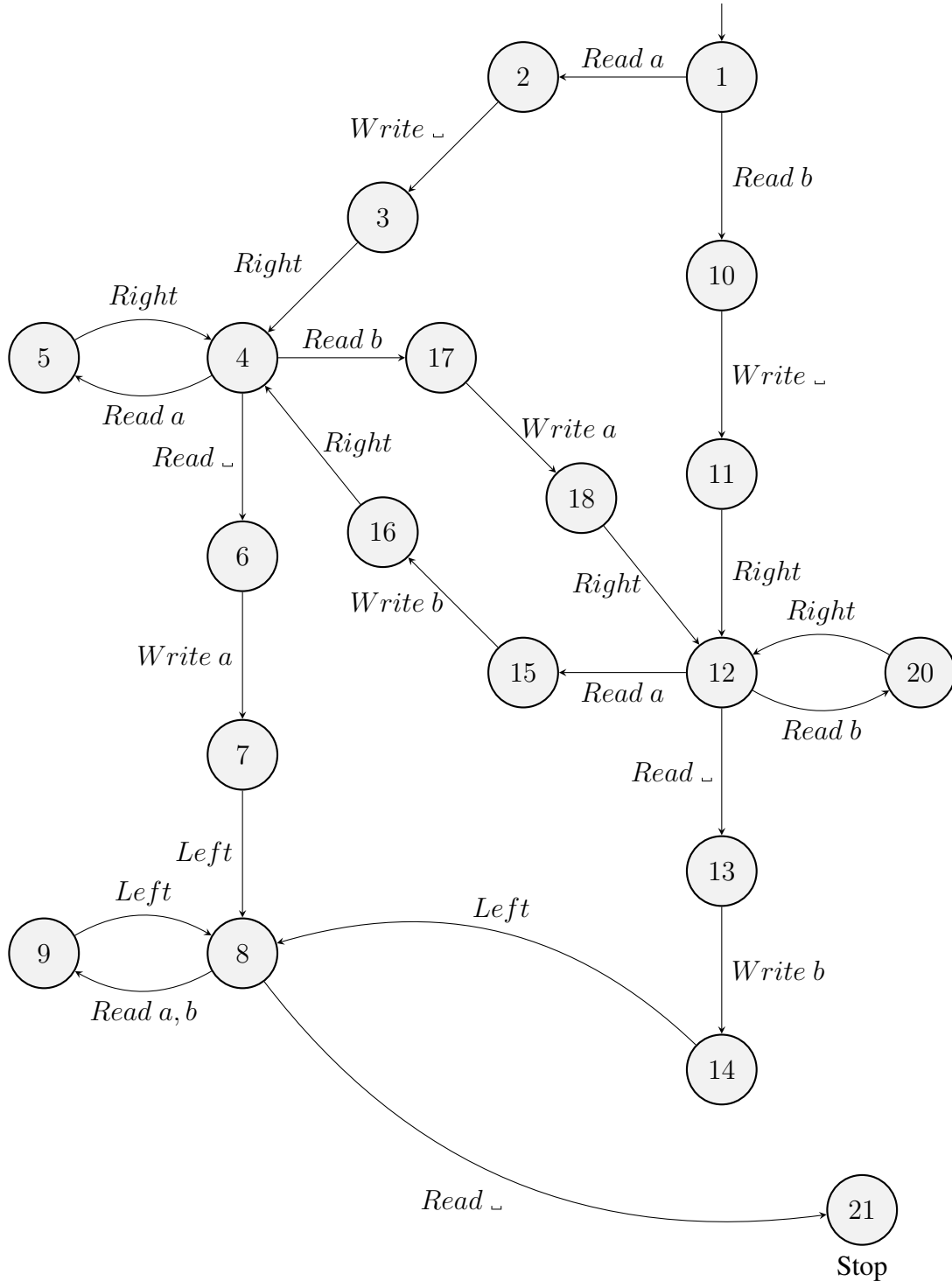$$\text{aaab\.bbababbaaa} \tag{2}$$

2.2 Converting Initial Configurations – The Setting-up Program

See also the video lecture recording: The Setting-up Program on Canvas.
In this step, we will setup the simple TM to represent the fancy TM *i.e.* we want a program that stretches a fancy *input* tape — recall that this will contain only a, b, ␣ — into a corresponding simple tape. For example, if the input is initially a\.bbab, the simple TM will start by *stretching* the given input to a\.abbbbaabb.

3

One way to write a stretching program is to add one character at a time, which would require $O(n)$ steps, and then repeat it for each of the characters in the input, requiring a total of $O(n^2)$ steps. As an example, consider the modified version of the TM seen during last week, which makes a space at the current head position e.g. given the input $\dot{a}bab$ it will result in the output $\sqcup abab$. We can further modify this TM to achieve the stretching program (left as an exercise for you).

```
        2  ←── Read a ──  1
         \                |
    Write ⌣                | Read b
           \               ↓
            3             10
             \            |
       Right   \       Write ⌣
    Right        \       |
  5 ⇄ 4 ── Read b ──→ 17  ↓
      |              \    11
 Read a  Read ⌣  Right  Write a  |
           |       16     \     Right
           ↓        \      18    Right
           6        Write b  \   |
           |         \   Right \  ↓
      Write a         15 ──────  12 ⇄ 20
           |      Read a        Read b
           ↓                     |
           7                  Read ⌣
           |                     ↓
        Left                    13
    Left                         |
   9 ⇄ 8 ──── Left ────      Write b
     Read a, b        \        ↓
           \           \       14
            \  Read ⌣   \
             \           → 21
                          Stop
```

You can easily see that the above TM takes $O(n)$ steps for creating a space; we will have a similar complexity for the modified version to duplicate a single character on the tape. The full stretching program will repeat the above steps for each of the characters on the tape, therefore, it will take $O(n^2)$ steps.

4

2.3   Performing Fancy TM Instructions – The Simulating Programs

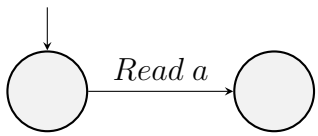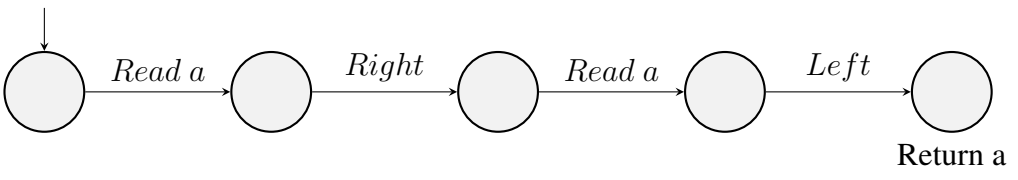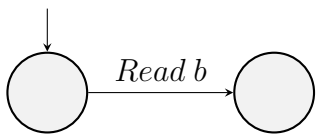See also the video lecture recording: The Simulating Programs on Canvas.
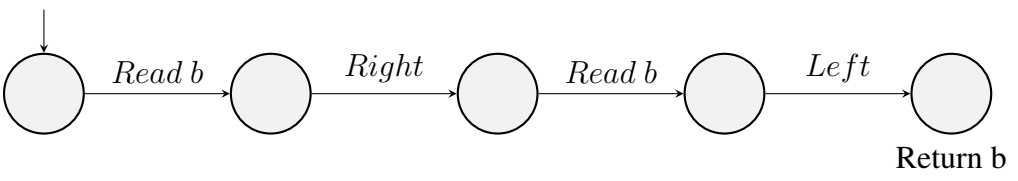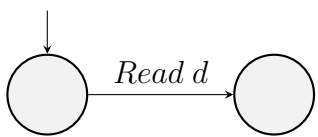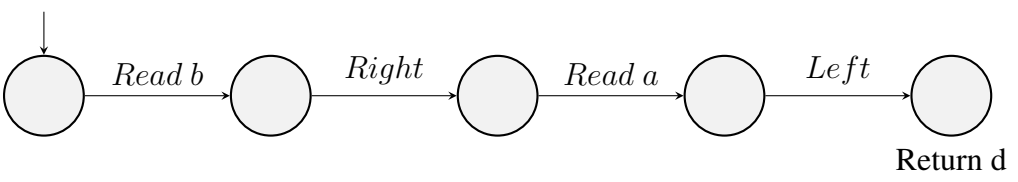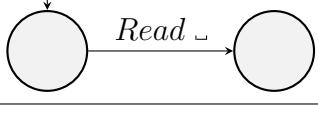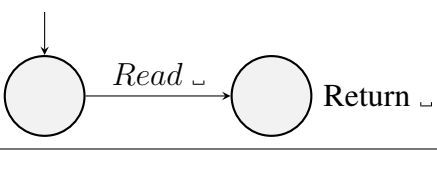In this step, we simulate each instruction of the fancy TM by a program of the simple TM. For example, we simulate the fancy Right, Left and Stop as the following simple programs:
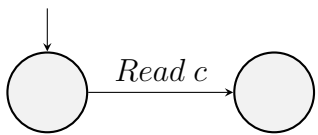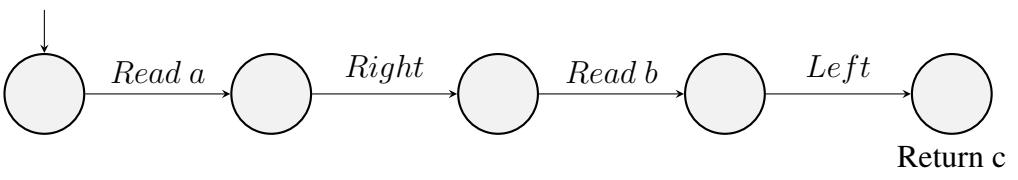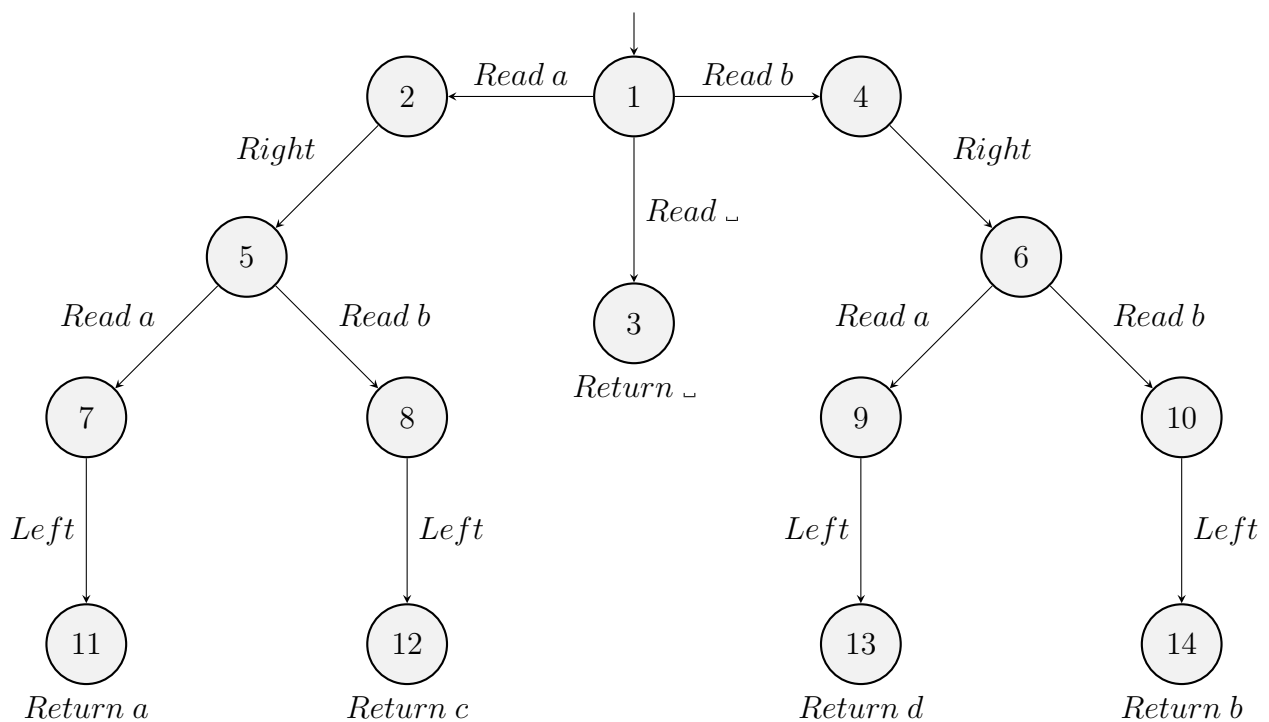
Fancy TM Steps	Simple Simulating TMs
○ —*Right*→ ○	○ —*Right*→ ○ —*Right*→ ○ Stop
○ —*Left*→ ○	○ —*Left*→ ○ —*Left*→ ○ Stop
○ Stop	○ Stop

Similarly, we can create the following simple TM programs for the Write operations.

Fancy TM Steps	Simple Simulating TMs
○ —*Write a*→ ○	○ —*Write a*→ ○ —*Right*→ ○ —*Write a*→ ○ —*Left*→ ○ Stop
○ —*Write b*→ ○	○ —*Write b*→ ○ —*Right*→ ○ —*Write b*→ ○ —*Left*→ ○ Stop
○ —*Write c*→ ○	○ —*Write a*→ ○ —*Right*→ ○ —*Write b*→ ○ —*Left*→ ○ Stop
○ —*Write d*→ ○	○ —*Write b*→ ○ —*Right*→ ○ —*Write a*→ ○ —*Left*→ ○ Stop
○ —*Write ␣*→ ○	○ —*Write ␣*→ ○ —*Right*→ ○ —*Write ␣*→ ○ —*Left*→ ○ Stop

It is important to note that the tape head on the simple TM is moved to the left, after writing the second character. Similarly, we can create the following simple TM programs for the Read operations.

Fancy TM Steps	**Simple Simulating TMs**
Read a	Read a → Right → Read a → Left → Return a
Read b	Read b → Right → Read b → Left → Return b
Read c	Read a → Right → Read b → Left → Return c
Read d	Read b → Right → Read a → Left → Return d
Read ␣	Read ␣ → Return ␣

We can combine the simple simulating TMs for the read operations as shown below:


All of these simple TM programs are constant times *e.g.* each Read/Write operation takes 4 steps, therefore, all of these have polynomial time complexity.

2.4 Converting Simple to Fancy TM's Output – The Finishing Program

See also the video lecture recording: The Finishing Program on Canvas.

In the last step, we want a program that *squashes* a simple tape back into a fancy one that serves as the output. For example, it would squash ȧabbaabbbb into ababb⌣. Similar to the *stretching* program, the squashing program will do it one character at a time and its complexity will be quadratic *i.e.* $O(n^2)$. Here is one possible program for squashing the output tape of a simple (simulating) TM back to the fancy tape.
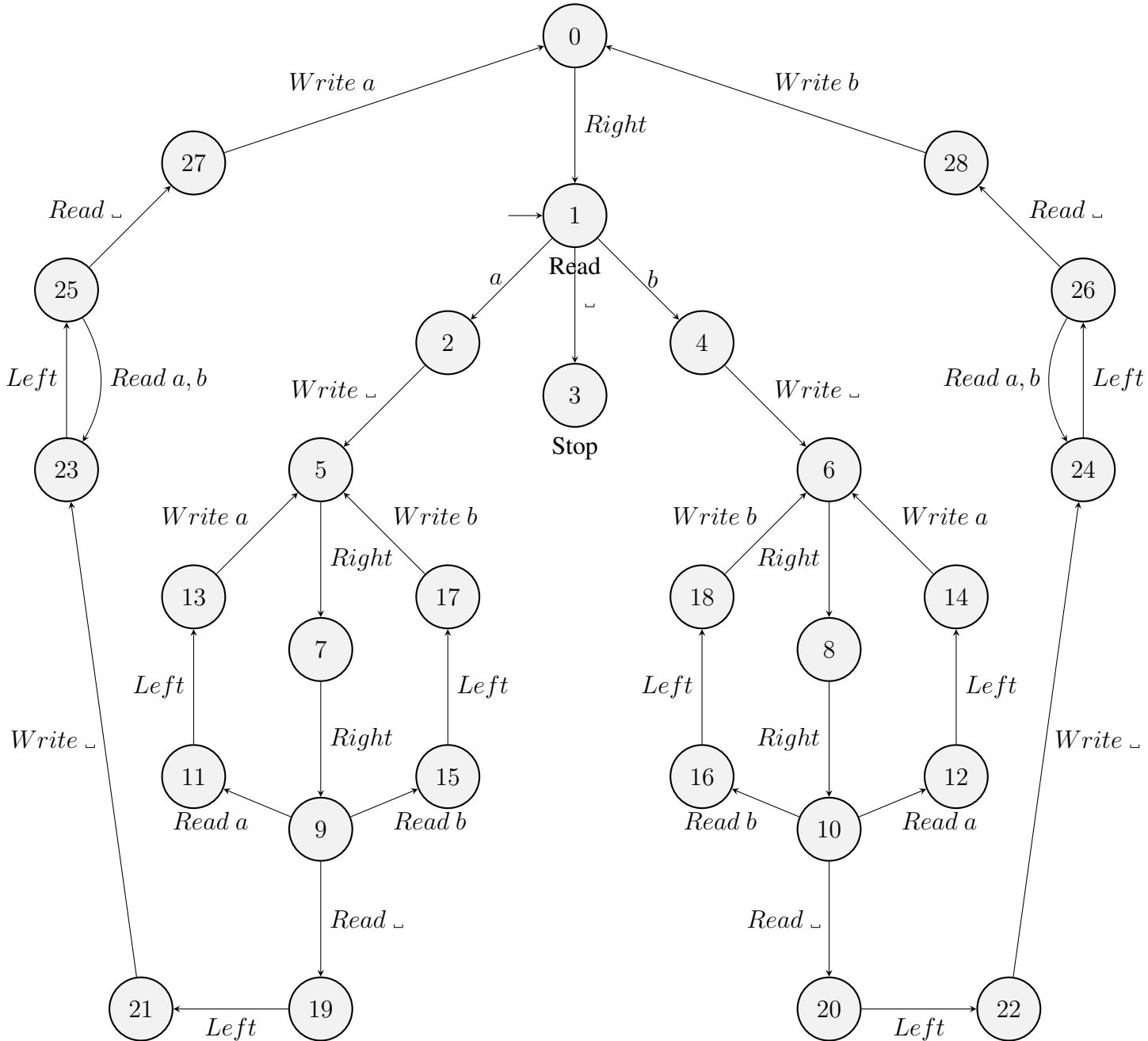
2.5 Fancy to Simple TM Conversion – Discussion

See also the video lecture recording: Fancy to Simple TM Conversion – Discussion on Canvas.

Given a fancy TM, the corresponding simple TM (expressed using macros) is as follows:
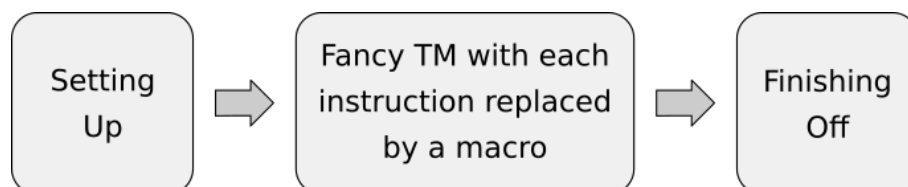
Suppose that the fancy program runs in polynomial time, then consider the following arguments:

- The stretching program is quadratic time, as discussed earlier.

- Each simulating instruction is constant time, and polynomially many of them happen.

- The squashing program is also quadratic time, and gets applied to the simple tape contents that's twice the size of the corresponding fancy tape contents.

Therefore, the simple TM runs in polynomial time as well.

2.5.1 Fancy to Simple TM Conversion – Example

Now lets consider how we can convert a fancy program into a simple program. Let's assume that we have a fancy TM that starts on the leftmost character, changes all a's to b's and ends-up on the cell to the left of input block. Lets assume that the program accomplishes this in two steps:

1. When going to the right, the fancy TM changes all a's to c's

2. When going to the left, the fancy TM changes all c's to b's.
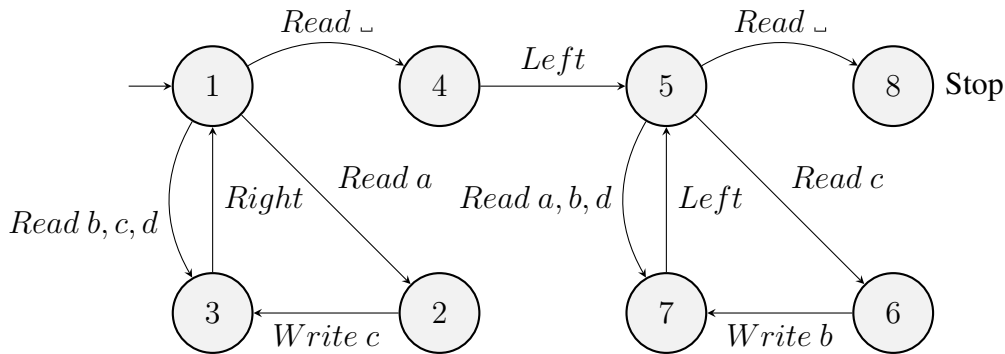
This becomes the following simple TM written using macros (that you can expand, if interested). Each macro name starts with m, in order to differentiate it from the fancy machine instruction. *e.g.* $mRead$ is a macro that replaces the $Read$ instruction of the fancy TM.
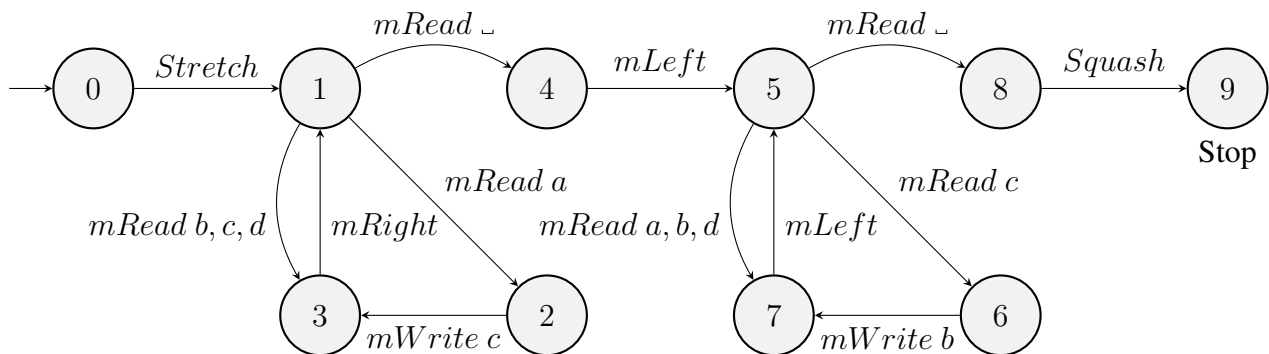

From the above discussion and examples, we can conclude the following:

1. We can convert a fancy TM using auxiliary characters into a simple TM, so allowing the extra characters does not give us any more power to express functions.

2. If the fancy TM is polynomial time (setting-up, simulating and finishing programs are all polynomial), then the resulting simple TM is also polynomial time.

3   Two-tape Turing machine

See also the video lecture recording: Two-tape Turing machines on Canvas.

A two-tape Turing Machine has a main tape and an auxiliary tape, with a head on each tape and each head is able to move independently. The available instructions are Write Main x, Write Aux x, Read Main, Read Aux, Left Main, Left Aux, Right Main, Right Aux, No-op and Return v. A program may assume that initially the auxiliary tape is blank. The cell positions on the main and auxiliary tapes can also be numbered as $\cdots, -1, 0, 1, 2, 3, \cdots$. We know that the copy-reverse problem can be solved in linear time using a two-tape machine. The following figure shows a model of two tape TM:
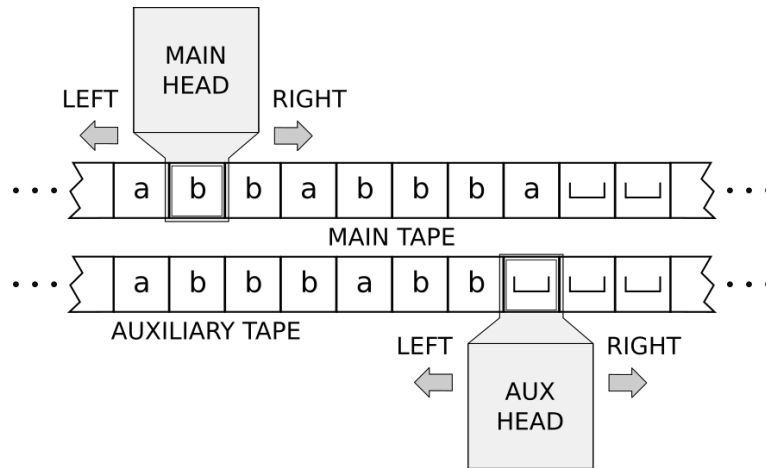
In this section, we will study how to convert a two-tape Turing machine for the alphabet $\{\mathtt{a}, \mathtt{b}, \text{␣}\}$ into a single tape machine using the following extended alphabet:

$$\{\mathtt{a}, \mathtt{b}, \text{␣}, \mathtt{L}, \mathtt{R}, \mathtt{H}\}.$$

We already know how to convert TMs using an extended alphabet (fancy programs) into TMs using the basic alphabet (simple programs). So the conversion process will look like:

Two-tape TM $\Rightarrow$ Single-tape Fancy TM $\Rightarrow$ Single-tape Simple TM

How to convert a two-tape configuration into a single-tape configuration? Let's suppose that each tape has position 0, position 1, position 2, etc. The idea is to take each position from the main tape and the corresponding position from the auxiliary tape and represent it on the single tape as a block of 4 characters, taken from the extended alphabet. For example, the following two-tape configuration:

$$\text{Main}\quad \mathtt{a\dot{b}aab}$$
$$\text{Aux}\quad \mathtt{baa\dot{b}bb}$$

corresponds to the following single-tape configuration:

$$\mathtt{\dot{L}␣a␣bHb␣a␣a␣a␣aHb␣b␣b␣␣␣bR}$$

which is represented with 6 blocks of four characters each. For each block of four characters, we record the following information:

- Is the main tape head here? $\mathtt{H}$ for yes and ␣ for no.

- What's the character on the main tape at this position?

- Is the auxiliary tape head here? $\mathtt{H}$ for yes and ␣ for no.

- What's the character on the auxiliary tape at this position?

For example, the block ␣aHb indicates that we have the character $\mathtt{a}$ on the main tape, and the character $\mathtt{b}$ on the auxiliary tape, with the auxiliary head on $\mathtt{b}$. We put an $\mathtt{L}$ marker on the left (with the fancy head located on it) and an $\mathtt{R}$ marker on the right, to delineate the whole configuration. Again, we can devise a Setting up program, simulating programs and a Finishing program; and all of them are polynomial time.

The Setting-up Program: We know that at the start, the two-tape TM will have input only on the main tape. For example, we can have ȧbabb as the input block, which is a simple configuration but as we are converting a two-tape to a single-tape TM, we can regard it as:

$$\text{Main} \quad \text{ȧbabb}$$
$$\text{Aux} \quad \dot{\text{⊔}}\text{⊔⊔⊔⊔}$$

Therefore, the corresponding configuration for the single-tape TM will be:

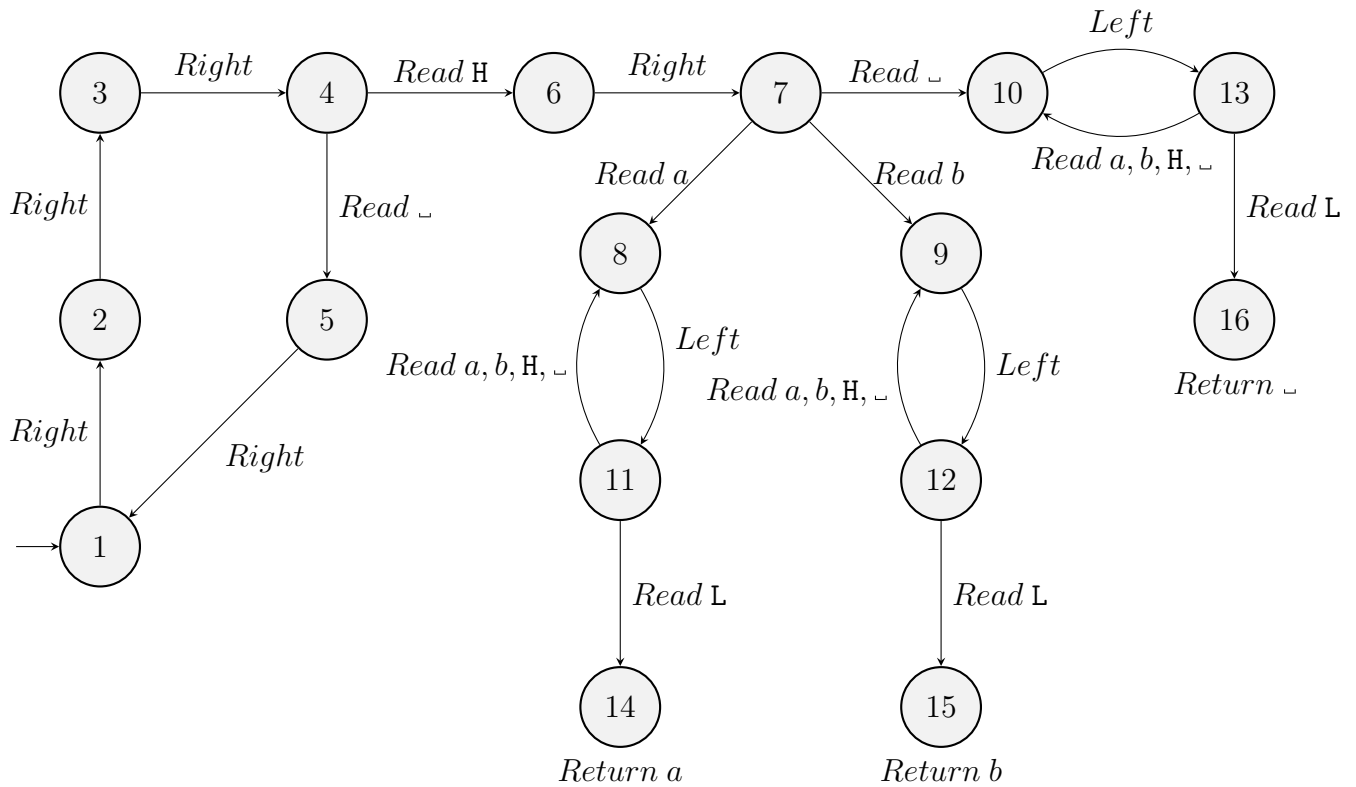$$\dot{\text{L}}\text{HaH⊔⊔b⊔⊔⊔a⊔⊔⊔b⊔⊔⊔b⊔⊔R}$$

The Simulating Programs: We can create simulating programs for each of the two-tape instructions and apply them to the single-tape input block. *Remember to move the single-tape fancy head back to the* L *position at the end of each simulating program.* For example, the following table shows a few of the two-tape instructions and how these will be simulated by the single-tape TM:

Two-tape TM Steps	Single-tape Simulating TM Input	Single-tape Simulating TM Output
Write Main b	L̇HaH⊔⊔b⊔⊔⊔a⊔⊔⊔b⊔⊔⊔b⊔⊔R	L̇HbH⊔⊔b⊔⊔⊔a⊔⊔⊔b⊔⊔⊔b⊔⊔R
Write Aux a	L̇Ha⊔⊔⊔b⊔⊔⊔aH⊔b⊔⊔⊔b⊔⊔R	L̇Ha⊔⊔⊔b⊔⊔⊔aHa⊔b⊔⊔⊔b⊔⊔R
Read Main	L̇⊔aH⊔⊔b⊔Ha⊔⊔⊔b⊔⊔⊔b⊔⊔R	L̇⊔aH⊔⊔b⊔⊔Ha⊔⊔⊔b⊔⊔b⊔⊔R / Return a
Read Aux	L̇Ha⊔⊔⊔b⊔⊔⊔aH⊔b⊔⊔⊔b⊔⊔R	L̇Ha⊔⊔⊔b⊔⊔⊔aH⊔b⊔⊔b⊔⊔R / Return ⊔
Right Main	L̇Ha⊔⊔⊔b⊔⊔⊔aH⊔b⊔⊔⊔b⊔⊔R	L̇⊔a⊔⊔Hb⊔⊔⊔aH⊔b⊔⊔⊔b⊔⊔R
Left Aux	L̇⊔a⊔⊔Hb⊔⊔⊔aH⊔b⊔⊔⊔b⊔⊔R	L̇⊔a⊔⊔HbH⊔a⊔⊔b⊔⊔⊔b⊔⊔R

For better understanding, a simulating program for the $Write\ Main\ b$ instruction is shown below:
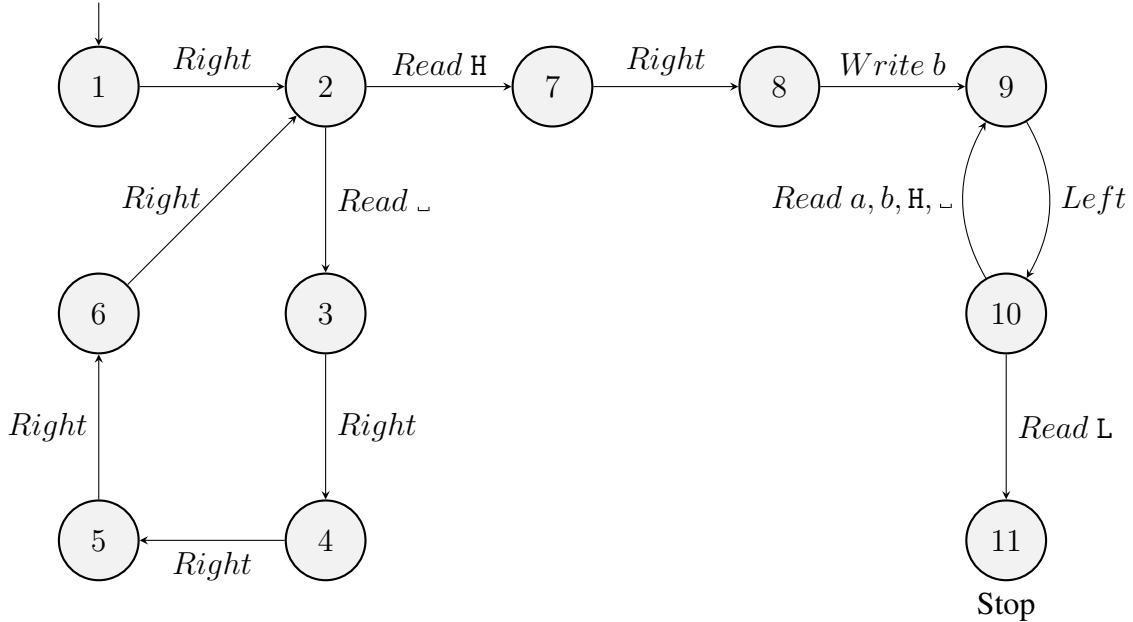
and here is another single-tape TM for simulating the $Read\ Aux$ instruction:

State diagram (simulating TM):

- 3 →(*Right*)→ 4 →(*Read* H)→ 6 →(*Right*)→ 7 →(*Read* ␣)→ 10
- 10 ↔ 13 : *Left* (10→13), *Read* $a, b,$ H$,$ ␣ (13→10)
- 13 →(*Read* L)→ 16 : *Return* ␣
- 1 →(*Right*)→ 2 →(*Right*)→ 3
- 4 →(*Read* ␣)→ 5
- 1 →(*Right*)→ 5
- 7 →(*Read a*)→ 8, 7 →(*Read b*)→ 9
- 8 ↔ 11 : *Read* $a, b,$ H$,$ ␣ (8→11), *Left* (11→8)
- 9 ↔ 12 : *Read* $a, b,$ H$,$ ␣ (9→12), *Left* (12→9)
- 11 →(*Read* L)→ 14 : *Return a*
- 12 →(*Read* L)→ 15 : *Return b*

Simulating programs for other instructions such as $Write\ Aux\ a$, $Read\ Main$, $Left\ Main$, $Right\ Aux$ *etc* are left as an exercise.

The Finishing Program: As we already know that the finishing program will do the opposite of setting-up *i.e.* convert the fancy single-tape output back to the two-tape configuration. For example, if the final configuration of the simulating TM looks like:

$$\dot{\mathtt{L}}\mathtt{Hb}␣␣\mathtt{b}␣␣\mathtt{a}␣␣\mathtt{a}␣␣\mathtt{b}␣␣\mathtt{bH}␣\mathtt{R}$$

then the finishing program will produce the following configuration:

$$\begin{array}{ll} \text{Main} & \dot{\mathtt{b}}\mathtt{baabb} \\ \text{Aux} & ␣␣␣␣␣\dot{␣} \end{array}$$

which is in fact equivalent to the output: $\dot{\mathtt{b}}\mathtt{baabb}$

Each of these processes *i.e.* setting-up, simulating and finishing programs are polynomial time. So again, when we have an auxiliary tape:

1. There is no change in the functions we can program using a two-tape machine.

2. There is no change to what we can do in polynomial time.

4 2-Dimensional Turing machines

See also the video lecture recording: 2-Dimensional Turing machines on Canvas.

A two-dimensional Turing machine (2D-TM) has a infinite sheet rather than a tape. The available instructions are Write x, Read, Right, Left, Up, Down, No-op and Return v. A program may assume that initially the sheet is blank except for one row, and must ensure that finally it is blank except for that row. The following figure shows a schematic represenation of a 2D-TM during execution:
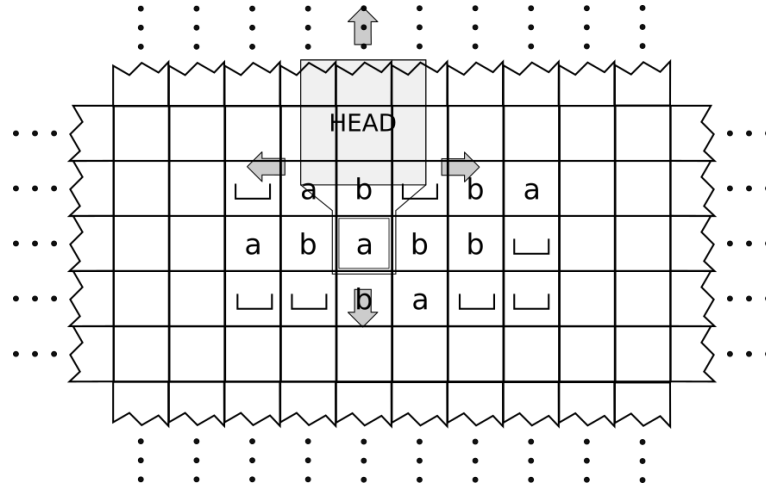
We can represent a 2D sheet with alphabet $\{a, b, \_\}$ as a 1D tape over:

$$\{a, b, \_, L, R, T, B\}.$$

where T stands for Top and B for Bottom.

Here is an example with sheet contents (same as shown in the figure above):

```
_ab_ba
abȧbb_
__ba__
```

It can be represented as

```
TL_ab_baRLabȧbb_RL__ba__RB
```

with each row given as a block of six characters is copied and enclosed in $L \ldots R$. Each row of the 2D sheet is represented with the same-length blocks on the 1D tape. The whole set of blocks are delimited by T and B, to indicate the Top and Bottom of 2D sheet. The head is in the position corresponding to the head position in the 2D machine.

Again we can devise a Setting up program, Simulating programs and a Finishing program. The setting-up program will setup the single tape configuration from the 2D sheet. The simulating programs will simulate each of the 2D-TM instructions such as Read, Write x, Right, Left, Up, Down etc. Read and Write instructions are straight forward, as these instructions operate at the current head position. The head movement instructions like Left and Right are easy to simulate, whereas the Up and Down instructions are more challenging, as we need to move the tape head between blocks on the 1D tape (details have been left to your imagination!). The finishing off program will take the 1D tape configuration and produce a 2D sheet output. Each of these programs run in polynomial time. So again, when we allow for a sheet instead of a tape:

1. It doesn't change the functions $\{a, b\}^* \to \{a, b\}^*$ that we can program using a 2D machine.

2. It doesn't change what we can do in polynomial time.

The reason why Polly only cares about whether something is polynomial time, is that this question doesn't depend on the particular machine model.

5 Summary

In this lecture series, we have studied:

- The general idea for finding the space complexity of a given Turing machine *i.e.* the worst case space efficiency is proportional to the execution time or less.

- How a fancy Turing machine using auxiliary characters can be converted into a simple Turing machine. This process involves the following four steps:

    1. Defining Relation between Fancy and Simple Tapes
    2. Converting Initial Configurations – The Setting-up Program
    3. Performing Fancy TM Instructions – The Simulating Programs and
    4. Converting Simple to Fancy TM's Output – The Finishing Program

- How a two-tape Turing machine can be converted into a single-tape fancy Turing machine. We can further convert this single-tape fancy TM to a single-tape simple TM using the process studied earlier.

- How a 2D Turing machine using a sheet can be converted into a single-tape Turing machine.