COMP20121 The Implementation and Power of Computer Languages

‘Power’ Part

http://www.cs.man.ac.uk/~petera/2121/index.html .

Peter Aczel

room: CS2.52, tel: 56155

email: petera@cs.man.ac.uk

School of Computer Science, University of Manchester

COMP20121, ‘power’ part: section 4

LECTURE NINE

Section 4: Computability

Overview

In this section we investigate the use of

Turing machines as computation devices.

Overview

In this section we investigate the use of Turing machines as computation devices.

- We can view a Turing machine as a program: It takes an input (the content of the tape at the start) and delivers an output (the content of the tape when it stops) or goes on forever.

Overview

- We consider variants of TMs.

Overview

- We consider variants of TMs.
- We compare TMs to other computation devices, in particular to ‘real’ computers.

Overview

- We consider variants of TMs.
- We compare TMs to other computation devices, in particular to ‘real’ computers.
- We ask whether there are any problems which cannot be solved (by any computing device).

Overview

- We consider variants of TMs.
- We compare TMs to other computation devices, in particular to ‘real’ computers.
- We ask whether there are any problems which cannot be solved (by any computing device).
- We investigate which kinds of problems are solvable
  and which aren’t.

Some history

Turing machines were created by Alan Turing in the 1930s.

Some history

Turing machines were created by Alan Turing in the 1930s. This is before the advent of most sophisticated computing devices!

Some history

Turing machines were created by Alan Turing in the 1930s. This is before the advent of most sophisticated computing devices!

Turing wanted to have a framework in which he had a tight control over what it meant to carry out a calculation. This also gave him a very definite meaning of the concept of an algorithm.

Some history

Having begun his academic career in Cambridge, Turing spent the war at Bletchley Park where he was instrumental in breaking the German Enigma code.
He later came to Manchester to help in building the ‘baby’, the first stored program computer.

Some history

Having begun his academic career in Cambridge, Turing spent the war at Bletchley Park where he was instrumental in breaking the German Enigma code.
He later came to Manchester to help in building the ‘baby’, the first stored program computer. Some personal differences led to him being employed by the Maths department instead.

Computations via Turing machines

Idea: We can view the content of the tape when the machine starts as the input for the machine.

Computations via Turing machines

Idea: We can view the content of the tape when the machine starts as the input for the machine. There are now two possibilities.

Computations via Turing machines

Idea: We can view the content of the tape when the machine starts as the input for the machine. There are now two possibilities.

- The machine stops.

Computations via Turing machines

Idea: We can view the content of the tape when the machine starts as the input for the machine. There are now two possibilities.

- The machine stops. In that case, we consider the content of the tape as the output of the machine for the given input.

Computations via Turing machines

- The machine does not stop.

Computations via Turing machines

- The machine does not stop. In that case, we consider the machine not to give an output for the given input.

Computations via Turing machines

- The machine does not stop. In that case, we consider the machine not to give an output for the given input.
- As before, we assume that the machine starts with the head pointing at the left-most symbol of the input.

Computations via Turing machines

- The machine does not stop. In that case, we consider the machine not to give an output for the given input.
- As before, we assume that the machine starts with the head pointing at the left-most symbol of the input.
- If we use Turing machines in this way, we typically do not identify accepting states (since they are here irrelevant).

A machine

Here is an example of a Turing machine which . . . does what?

A machine

Here is an example of a Turing machine which . . . does what?

δ	a	b	$\sqcup$
0	$(1, \sqcup, R)$	$(2, \sqcup, R)$	stop
1	$(1, a, R)$	$(1, b, R)$	$(3, \sqcup, R)$
2	$(2, a, R)$	$(2, b, R)$	$(4, \sqcup, R)$
3	$(3, a, R)$	$(3, b, R)$	$(5, a, L)$
4	$(4, a, R)$	$(4, b, R)$	$(6, b, L)$
5	$(5, a, L)$	$(5, b, L)$	$(7, \sqcup, L)$
6	$(6, a, L)$	$(6, b, L)$	$(8, \sqcup, L)$
7	$(7, a, L)$	$(7, b, L)$	$(0, a, R)$
8	$(8, a, L)$	$(8, b, L)$	$(0, b, R)$

The Copy machine

As we have seen in the case of the machine that recognizes all palindromes, it can be difficult to deduce the action of a machine from the transition table. It turns out that there are parallels between states 1 and 2, 3 and 4, 5 and 6, as well as 7 and 8.

The Copy machine

In state 0, the machine behaves as follows.

δ	a	b	$\sqcup$
0	$(1, \sqcup, R)$	$(2, \sqcup, R)$	stop

In other words, it remembers whether the current symbol is an a (by going to state 1) or a b (by going to state 2). It overwrites that symbol with a $\sqcup$.

The Copy machine

In state 1, the machine moves to the right until it finds a $\sqcup$.

δ	a	b	$\sqcup$
1	$(1, a, R)$	$(1, b, R)$	$(3, \sqcup, R)$

Upon doing so it has reached the end of the input string, and it moves to state 3.

The Copy machine

In state 3, the machine moves to the right until it finds another $\sqcup$.

δ	a	b	$\sqcup$
3	$(3, a, R)$	$(3, b, R)$	$(5, a, L)$

It has now reached the end of the second string on the tape (which is empty at the start) and it prints the remembered letter (namely a). It then moves one to the left and goes to state 5.

The Copy machine

In states 5 and 7 the machine moves to the left until it finds a $\sqcup$.

δ	a	b	$\sqcup$
5	$(5, a, L)$	$(5, b, L)$	$(7, \sqcup, L)$
7	$(7, a, L)$	$(7, b, L)$	$(0, a, R)$

The Copy machine

δ	a	b	$\sqcup$
5	$(5, a, L)$	$(5, b, L)$	$(7, \sqcup, L)$
7	$(7, a, L)$	$(7, b, L)$	$(0, a, R)$

In the first case, this is the $\sqcup$ between the two strings, in the second this is the position of the original symbol. The machine now restores the remembered symbol, in this case an a, and goes one to the right to the next symbol. It then starts over in state 0.

The Copy machine

If the symbol the machine found is a b, it goes through the same steps via states 2, 4, 6 and 8.

The Copy machine-II

In summary, we have found the following behaviour:

- Look at the current symbol, remember whether it's an a or a b (by going into state 1 or 2), overwrite it with a blank, move to the right. If it's a blank, stop.

The Copy machine-II

In summary, we have found the following behaviour:

- Look at the current symbol, remember whether it's an a or a b (by going into state 1 or 2), overwrite it with a blank, move to the right. If it's a blank, stop.
- Move to the right until a blank is found. This is the end of the input string.

The Copy machine-II

Move to the right until the next blank is found. This is the end of the second string. Write the remembered letter, move one to the left. (Keep remembering which letter it was that is being copied, again by using different states for different letters, so that it can be rewritten in its old place.)

The Copy machine-II

- Move to the left until a blank is found.

  This is the one between the two strings.

The Copy machine-II

- Move to the left until a blank is found. This is the one between the two strings.
- Move to the left until the next blank is found. That's where the letter copied was originally. Write the remembered letter, move one to the right. Start over at the first step.

The Copy machine-II

- Move to the left until a blank is found. This is the one between the two strings.
- Move to the left until the next blank is found. That's where the letter copied was originally. Write the remembered letter, move one to the right. Start over at the first step.
- In other words, the machine copies the input string.

Building Turing machines

Given a complex task, it may seem very difficult to build a Turing machine which fulfils it—certainly, it can be difficult to get this exactly right!

Building Turing machines

Given a complex task, it may seem very difficult to build a Turing machine which fulfils it—certainly, it can be difficult to get this exactly right!

However, it is possible to start with simple machines and build them up to more complicated ones.

Building Turing machines

- Assume we want to build a machine which moves to the end of the input string and then prints an a.

Building Turing machines

- Assume we want to build a machine which moves to the end of the input string and then prints an a.
- This machine can be built from one which goes to the end of the input string, and another machine which prints an a.

Building Turing machines

- Assume we want to build a machine which moves to the end of the input string and then prints an a.
- This machine can be built from one which goes to the end of the input string, and another machine which prints an a.
- Yes, this is a trivial example, but it does illustrate a point.

Building Turing machines-II

M_1

δ_1	a	b	$\sqcup$
0	$(0, a, R)$	$(0, b, R)$	stop

M_2

δ_1	a	b	$\sqcup$
0	$(1, a, R)$	$(1, a, R)$	$(1, a, R)$
1	stop	stop	stop

Building Turing machines-II

M_1

δ_1	a	b	$\sqcup$
0	$(0, a, R)$	$(0, b, R)$	stop

This machine finds the end of the input string.

M_2

δ_1	a	b	$\sqcup$
0	$(1, a, R)$	$(1, a, R)$	$(1, a, R)$
1	stop	stop	stop

This machine prints an a and moves one to the right.

Building Turing machines-II

M_1

δ_1	a	b	$\sqcup$
0	$(0, a, R)$	$(0, b, R)$	stop

This machine finds the end of the input string.

M_2

δ_1	a	b	$\sqcup$
0	$(1, a, R)$	$(1, a, R)$	$(1, a, R)$
1	stop	stop	stop

This machine prints an a and moves one to the right.

To get a machine which behaves first like M_1 and then like M_2:

Building Turing machines-II

M_1

δ_1	a	b	$\sqcup$
0	$(0, a, R)$	$(0, b, R)$	stop

This machine finds the end of the input string.

M_2

δ_1	a	b	$\sqcup$
0	$(1, a, R)$	$(1, a, R)$	$(1, a, R)$
1	stop	stop	stop

This machine prints an a and moves one to the right.

To get a machine which behaves first like M_1 and then like M_2:

- Rename the states of M_2 so as not to overlap with those of M_1.

Building Turing machines-II

M_1

δ_1	a	b	$\sqcup$
0	$(0, a, R)$	$(0, b, R)$	stop

This machine finds the end of the input string.

M_2

δ_1	a	b	$\sqcup$
1	$(2, a, R)$	$(2, a, R)$	$(2, a, R)$
2	stop	stop	stop

This machine prints an a and moves one to the right.

To get a machine which behaves first like M_1 and then like M_2:

- Rename the states of M_2 so as not to overlap with those of M_1.

Building Turing machines-II

M_1

δ_1	a	b	$\sqcup$
0	$(0, a, R)$	$(0, b, R)$	stop

This machine finds the end of the input string.

M_2

δ_1	a	b	$\sqcup$
1	$(2, a, R)$	$(2, a, R)$	$(2, a, R)$
2	stop	stop	stop

This machine prints an a and moves one to the right.

To get a machine which behaves first like M_1 and then like M_2:

- Rename the states of M_2 so as not to overlap with those of M_1.
- Replace all stop commands for M_1 by transitions to the start state of M_2, writing the current symbol and not moving the head.

Building Turing machines-II

M_1

δ_1	a	b	$\sqcup$
0	$(0, a, R)$	$(0, b, R)$	$(1, \sqcup, N)$

M_2

δ_1	a	b	$\sqcup$
1	$(2, a, R)$	$(2, a, R)$	$(2, a, R)$
2	stop	stop	stop

This machine finds the end of the input string.

This machine prints an a and moves one to the right.

To get a machine which behaves first like M_1 and then like M_2:

- Rename the states of M_2 so as not to overlap with those of M_1.
- Replace all stop commands for M_1 by transitions to the start state of M_2, writing the current symbol and not moving the head.

Building Turing machines-II

M_1

δ_1	a	b	$\sqcup$
0	$(0, a, R)$	$(0, b, R)$	$(1, \sqcup, N)$

M_2

δ_1	a	b	$\sqcup$
1	$(2, a, R)$	$(2, a, R)$	$(2, a, R)$
2	stop	stop	stop

This machine finds the end of the input string.

This machine prints an a and moves one to the right.

To get a machine which behaves first like M_1 and then like M_2:

- Rename the states of M_2 so as not to overlap with those of M_1.
- Replace all stop commands for M_1 by transitions to the start state of M_2, writing the current symbol and not moving the head.

This is a general principle which can be used to build complicated machines from simple ones.

Variants of Turing machines

What happens if we change our idea of the storage medium a Turing machine can access?

Variants of Turing machines

One-sided tapes

Instead of having a tape which is infinite on both sides, we could have one which has a definite start and is infinite on one side only (typically the right).

If the machine then wants to move to the left when it has reached the beginning of the tape we assume that no move occurs.

Variants of Turing machines

One-sided tapes

Instead of having a tape which is infinite on both sides, we could have one which has a definite start and is infinite on one side only (typically the right).

Clearly we can simulate a TM with a one-sided type using one with a two-sided tape.

Variants of Turing machines

One-sided tapes

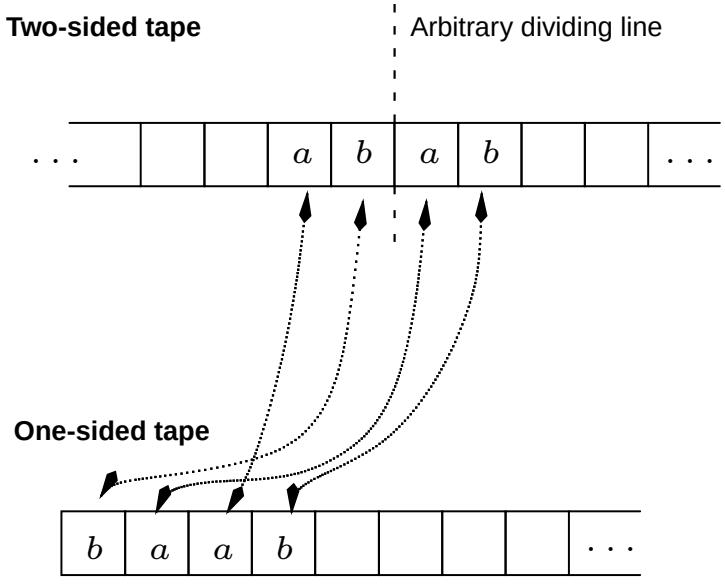
Instead of having a tape which is infinite on both sides, we could have one which has a definite start and is infinite on one side only (typically the right).

Conversely, if we want to simulate a TM with a two-sided tape by one which merely has a one-sided tape, we can split up the tape into different ‘zones’.

Variants of Turing machines

One-sided tapes

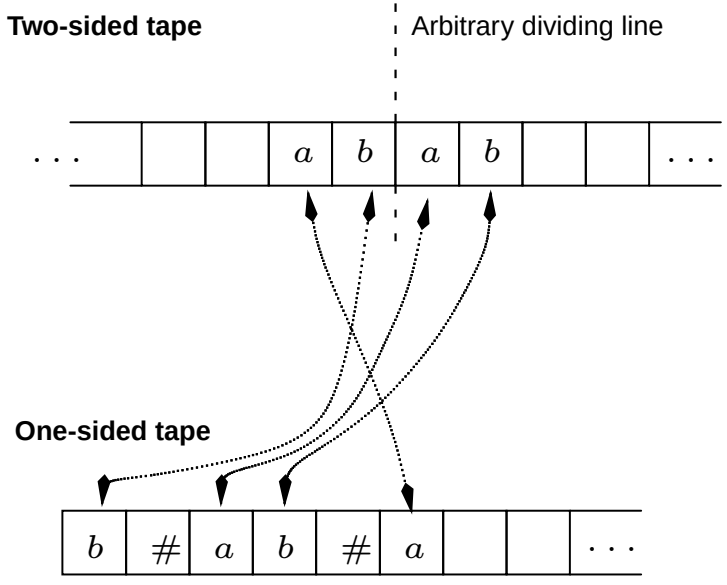
Either we can use ‘even’ and ‘odd’ slots for the ‘right’ and ‘left’ side of the tape (from an arbitrary cell):



Variants of Turing machines

One-sided tapes

Or we can use a special symbol, say $\#$, to separate the tape into regions which correspond to the left/right half of the tape:



Variants of Turing machines

One-sided tapes

We can now create a transition table which simulates the two-sided tape TM on a one-sided tape. This machine has to make a lot of transitions to move between the different regions!

Variants of TMs: Multiple Tapes

A machine might have more than one tape, with a head for each tape.

Variants of TMs: Multiple Tapes

A machine might have more than one tape, with a head for each tape. In that case, a transition table would have to

- make every transition dependent on all the currently-read symbols and
- have a write/move instruction for each head.

Variants of TMs: Multiple Tapes

Such a machine might look like this:

δ	$(a, \ )$	$(b, \ )$	$(\sqcup, \ )$
0	$(0, a, R, a, R)$	$(0, b, R, b, R)$	stop

Variants of TMs: Multiple Tapes

δ	$(a, \ )$	$(b, \ )$	$(\sqcup, \ )$
0	$(0, a, R, a, R)$	$(0, b, R, b, R)$	stop

Once again, we use an empty entry in a tuple to indicate that the transition in question does not depend on that particular symbol.

Variants of TMs: Multiple Tapes

δ	$(a, \ )$	$(b, \ )$	$(\sqcup, \ )$
0	$(0, a, R, a, R)$	$(0, b, R, b, R)$	stop

This particular machine has two tapes; it copies a string from the first to the second tape. Compare this with the 'copy' machine from earlier!

Variants of TMs: Multiple Tapes

Clearly we can simulate a machine with one tape by a machine which has several tapes.

Variants of TMs: Multiple Tapes

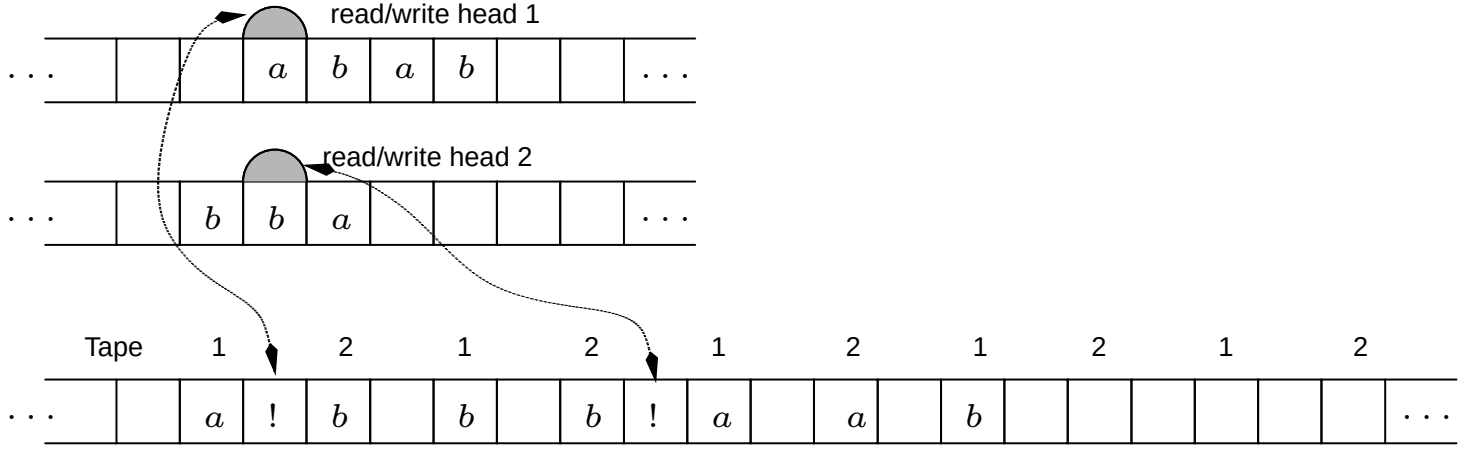
Conversely, we can simulate a machine with several, say two, tapes by one that has just one tape: We have to split up the tape to capture the content of both tapes. We also have to find a way of marking where the two heads are.

Variants of TMs: Multiple Tapes

We do the latter by using only every second cell to hold data; we use a marker such as ! in the non-data cells to indicate that the preceding cell is where one of the heads is located. Hence the tape consists of blocks of two cells.

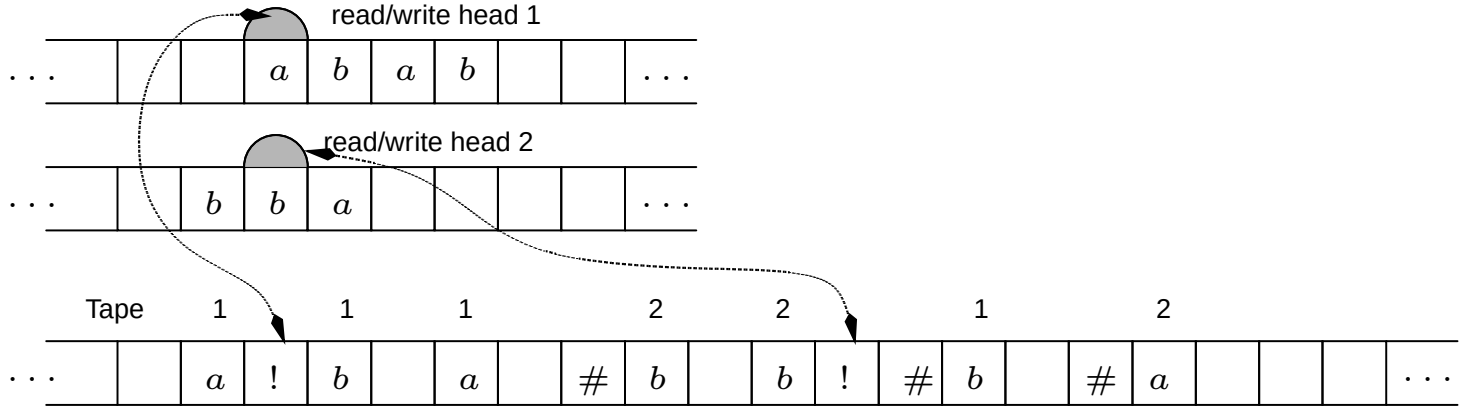
Variants of TMs: Multiple Tapes

Either we can use ‘even’ and ‘odd’ blocks for the first and second tape:



Variants of TMs: Multiple Tapes

Or we can use a special symbol, say $\#$, to separate the tape into regions which correspond to the first/second tape:



Variants of TMs: Multiple Tapes

We can now create a transition table which simulates the two-tape TM on one tape. This machine has to make a lot of transitions to move between the different regions!

Variants of Turing machines

Multi-dimensional tapes

A machine might have a multi-dimensional tape with one head, which can move in any direction (for a two-dimensional tape those would be left, right, up and down).

Variants of Turing machines

Multi-dimensional tapes

A machine might have a multi-dimensional tape with one head, which can move in any direction (for a two-dimensional tape those would be left, right, up and down).

Such a machine can be simulated by a TM with just one ‘ordinary’ tape.

Variants of Turing machines

Multi-dimensional tapes

A machine might have a multi-dimensional tape with one head, which can move in any direction (for a two-dimensional tape those would be left, right, up and down).

Such a machine can be simulated by a TM with just one ‘ordinary’ tape.

An exercise.

Variants of Turing machines

Non-deterministic TMs

Just as there are non-deterministic finite state automata or PDAs, there are non-deterministic TMs: They are given by transition tables which may have more than one entry for a given state and current symbol.

Variants of Turing machines

Non-deterministic TMs

Every non-deterministic TM can be simulated by a deterministic one.

We will not go into the details of the proof, but here is the principle: The several computations a machine might perform given some input form a finitely branching tree.

Variants of Turing machines

Non-deterministic TMs

One can build a deterministic machine which explores this tree in a breadth-first manner. The easiest way of doing this is to let the simulating machine use three tapes:

Variants of Turing machines

Non-deterministic TMs

- a tape to store the input,
- a tape to mirror computations, and
- a tape to keep track of the current position in the tree of non-deterministic computations.

Variants of Turing machines

Non-deterministic TMs

- a tape to store the input,
- a tape to mirror computations, and
- a tape to keep track of the current position in the tree of non-deterministic computations.

This is explained in detail in the text books.

Variants of Turing machines

In summary: There are plenty of variants of Turing machines using:

- several tapes,
- one-sided tapes,
- multi-dimensional tapes, or
- non-deterministic ones.

Variants of Turing machines

In summary: There are plenty of variants of Turing machines using:

- several tapes,
- one-sided tapes,
- multi-dimensional tapes, or
- non-deterministic ones.

All these machines are precisely as powerful as the ones we use as our basis: Deterministic machines which have one two-sided tape.

Variants of Turing machines

From now on we will feel free to describe such a variant in a proof. In particular, we will freely make use of several tapes where this simplifies the description of a machine.

The power of Turing machines

So how powerful are Turing machines compared with other computing devices? They are, after all, fairly simple, all in all.

The power of Turing machines

So how powerful are Turing machines compared with other computing devices? They are, after all, fairly simple, all in all.

Given a Turing machine, what does it take to simulate it using a normal computer?

The power of Turing machines

So how powerful are Turing machines compared with other computing devices? They are, after all, fairly simple, all in all.

Given a Turing machine, what does it take to simulate it using a normal computer?

Problem: A TM has a potentially infinite tape

The power of Turing machines

Problem: A TM has a potentially infinite tape (but at any given moment, only finitely many cells are used).

The power of Turing machines

Problem: A TM has a potentially infinite tape (but at any given moment, only finitely many cells are used).

Solution: Think of a computer to which we can add memory whenever it threatens to run out.

The power of Turing machines

Problem: A TM has a potentially infinite tape (but at any given moment, only finitely many cells are used).

Solution: Think of a computer to which we can add memory whenever it threatens to run out. Clearly we can write a TM-simulator in, say, Java or C.

The power of Turing machines-II

But given a computer, how could we possibly simulate it using a TM?

The power of Turing machines-II

But given a computer, how could we possibly simulate it using a TM? Remember CS100 or CS101. We may think of a computer as a von Neumann machine which has a limited set of instructions and registers to store things in.

The power of Turing machines-II

The instructions allow the machine

- to load the contents of a register,
- to store something to a register,
- to add the contents of two registers,
- to branch to an instruction based on the contents of a register.

The power of Turing machines-II

The instructions allow the machine

- to load the contents of a register,
- to store something to a register,
- to add the contents of two registers,
- to branch to an instruction based on the contents of a register.

But these kinds of instructions are simple enough that we can simulate them using a Turing machine!

Simulating a von Neumann machine

- We use a tape to store the contents of the registers.

Simulating a von Neumann machine

- We use a tape to store the contents of the registers.
- We could use a second tape to store information about where to find the contents of some given register.

Simulating a von Neumann machine

- We use a tape to store the contents of the registers.
- We could use a second tape to store information about where to find the contents of some given register.
- Or we might decide to reserve sufficiently many cells for each register, and have separate regions (which might be divided by #) for registers numbered from 0 to the highest available number.

Simulating a von Neumann machine

The instructions can then be translated into a big transition table (where each instruction will be coded by several lines of the table).

Simulating a von Neumann machine

- Loading a register corresponds to reading the contents,
- storing to a register corresponds to writing into the appropriate cells,
- addition can be encoded quite easily (see Exercise 28 (c) for adding 1),
- and branching statements just require the control flow to be directed towards the appropriate state.

Simulating a von Neumann machine

In this way we can simulate a computer using a Turing machine!