Turing machines (continued)

Ashley Montanaro
ashley@cs.bris.ac.uk

Department of Computer Science, University of Bristol
Bristol, UK

28 March 2014

Introduction

- We have seen an introduction to Turing machines and how they are defined.
- Today: More about what they can do, and some other models which are equivalent to the Turing machine.

Levels of formality

We can distinguish several different levels of formality when describing a Turing machine:

Levels of formality

We can distinguish several different levels of formality when describing a Turing machine:

- A formal description is the complete details of the states and transition function of the machine, specified via a transition diagram or a formal mathematical description.

Levels of formality

We can distinguish several different levels of formality when describing a Turing machine:

- A formal description is the complete details of the states and transition function of the machine, specified via a transition diagram or a formal mathematical description.
- An implementation description is a higher-level description where the operation of the machine is described in English text. However, enough details are provided to be able to derive the formal description with no additional ideas.

Levels of formality

We can distinguish several different levels of formality when describing a Turing machine:

- A formal description is the complete details of the states and transition function of the machine, specified via a transition diagram or a formal mathematical description.
- An implementation description is a higher-level description where the operation of the machine is described in English text. However, enough details are provided to be able to derive the formal description with no additional ideas.
- A high-level description describes the operation of the machine informally without mentioning the details of how it operates on the tape.

Levels of formality

We can distinguish several different levels of formality when describing a Turing machine:

- A formal description is the complete details of the states and transition function of the machine, specified via a transition diagram or a formal mathematical description.
- An implementation description is a higher-level description where the operation of the machine is described in English text. However, enough details are provided to be able to derive the formal description with no additional ideas.
- A high-level description describes the operation of the machine informally without mentioning the details of how it operates on the tape.

For the remainder of the unit, we will use implementation or high-level descriptions of Turing machines.

Levels of formality

We can distinguish several different levels of formality when describing a Turing machine:

- A formal description is the complete details of the states and transition function of the machine, specified via a transition diagram or a formal mathematical description.
- An implementation description is a higher-level description where the operation of the machine is described in English text. However, enough details are provided to be able to derive the formal description with no additional ideas.
- A high-level description describes the operation of the machine informally without mentioning the details of how it operates on the tape.

For the remainder of the unit, we will use implementation or high-level descriptions of Turing machines.

- Unless otherwise specified, if you are asked to describe a Turing machine, the implementation description is the right level to pick.

Encodings

We will often want to encode the objects on which a Turing machine operates.

- Given an object O, we write its representation as a string of symbols in some alphabet as $\langle O \rangle$.
- Each object has many different representations; we just pick a "reasonable" one.

Encodings

We will often want to encode the objects on which a Turing machine operates.

- Given an object O, we write its representation as a string of symbols in some alphabet as $\langle O \rangle$.
- Each object has many different representations; we just pick a "reasonable" one.

Some examples:

- $n \in \mathbb{N}$: we can write n as a binary string, e.g. $\langle 5 \rangle = \texttt{101}$.

Encodings

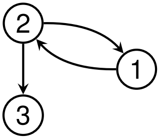
We will often want to encode the objects on which a Turing machine operates.

- Given an object O, we write its representation as a string of symbols in some alphabet as $\langle O \rangle$.
- Each object has many different representations; we just pick a "reasonable" one.

Some examples:

- $n \in \mathbb{N}$: we can write n as a binary string, e.g. $\langle 5 \rangle = \texttt{101}$.
- If G is a graph: we could write its adjacency matrix in binary, with commas between the rows, e.g. for

$G =$

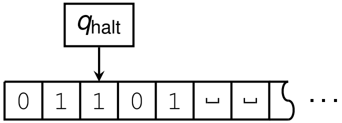


$\langle G \rangle = \texttt{010,101,000}$

Computing using a Turing machine

Turing machines can be used to solve more than decision problems.

- We can generalise the notion of Turing machines slightly by allowing the machine to output data.
- We can introduce a state q_{halt} such that, if the machine reaches state q_{halt}, it stops and outputs the contents of its tape.



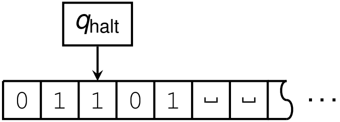
Ashley Montanaro
ashley@cs.bris.ac.uk

University of BRISTOL

Computing using a Turing machine

Turing machines can be used to solve more than decision problems.

- We can generalise the notion of Turing machines slightly by allowing the machine to output data.
- We can introduce a state q_{halt} such that, if the machine reaches state q_{halt}, it stops and outputs the contents of its tape.



This allows us to implement general algorithms on a Turing machine.

- A Turing machine can be seen as computing a mathematical function $f(x)$ of its input x.
- If a function can be computed by a Turing machine, we call it computable.

Ashley Montanaro
ashley@cs.bris.ac.uk

The Church-Turing thesis

We have seen that Turing machines are more powerful than PDAs. How much more powerful?

The Church-Turing thesis

We have seen that Turing machines are more powerful than PDAs. How much more powerful?

Church-Turing thesis

Everything which can be computed can be computed by a Turing machine.

The Church-Turing thesis

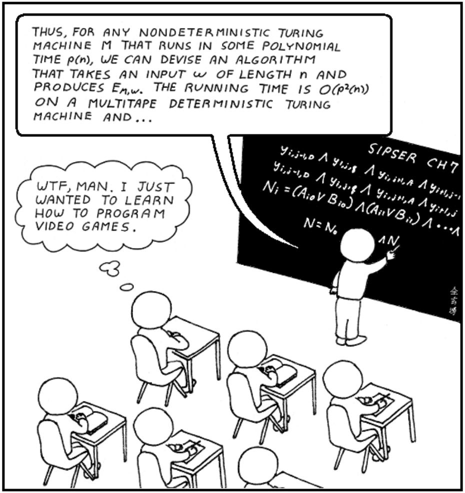
We have seen that Turing machines are more powerful than PDAs. How much more powerful?

Church-Turing thesis

Everything which can be computed can be computed by a Turing machine.

- This claim can be interpreted as saying that the Turing machine formalises our intuitive notion of what an algorithm is.
- NB: the Church-Turing thesis is unproven!
- It is a claim about physical reality: that anything we can compute in our physical universe can be computed by a Turing machine.

So why study Turing machines?



Pic: abstrusegoose.com

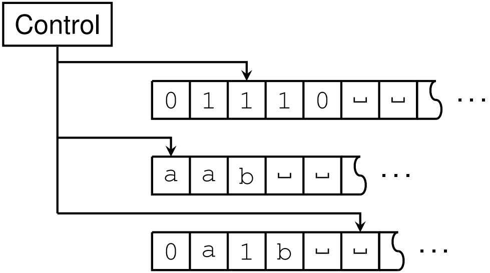
Because they give us a formal basis for understanding computation.

Evidence for the Church-Turing thesis

- One way in which we can strengthen our intuition that the C-T thesis is true: Consider computational models which are apparently stronger than TMs, and show that they can be simulated by TMs.

Evidence for the Church-Turing thesis

- One way in which we can strengthen our intuition that the C-T thesis is true: Consider computational models which are apparently stronger than TMs, and show that they can be simulated by TMs.
- The first of these models: the multitape Turing machine.



Multitape Turing machines

A k-tape Turing machine M is defined as follows.

- M has access to k tapes, each with its own head for reading and writing.
- The input is initially placed at the left end of tape 1; the other tapes are blank.
- The transition function is now of the form

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k,$$

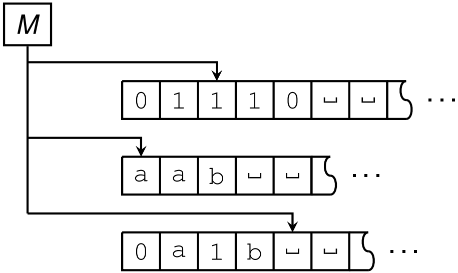
  so all k tapes can be written, and all k heads can be moved, simultaneously.
- Note that the state of M is shared across all of the tapes.

Multitape Turing machines

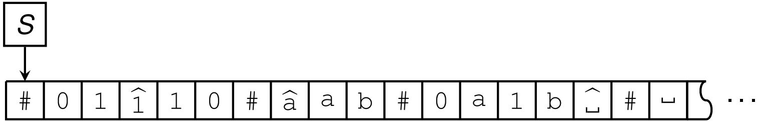
We show that a k-tape Turing machine can be simulated by a 1-tape Turing machine.

Multitape Turing machines

We show that a k-tape Turing machine can be simulated by a 1-tape Turing machine. The basic idea:



is mapped to



Simulating k tapes with 1 tape

Assume the input is a string $w_1 w_2 \dots w_n$.

1. To start with, S puts its tape into the format corresponding to a concatenation of all k tapes of M, i.e.:

$$\# \widehat{w_1} w_2 \dots w_n \# \hat{\sqcup} \# \hat{\sqcup} \# \dots \#$$

Simulating k tapes with 1 tape

Assume the input is a string $w_1 w_2 \dots w_n$.

1. To start with, S puts its tape into the format corresponding to a concatenation of all k tapes of M, i.e.:

$$\# \widehat{w_1} w_2 \dots w_n \# \widehat{\sqcup} \# \widehat{\sqcup} \# \dots \#$$

2. To simulate a move of M, S scans the tape from the first $\#$ to the $(k+1)$'st $\#$, to determine what the symbols are under the heads.

Simulating k tapes with 1 tape

Assume the input is a string $w_1 w_2 \dots w_n$.

1. To start with, S puts its tape into the format corresponding to a concatenation of all k tapes of M, i.e.:

$$\# \widehat{w_1} w_2 \dots w_n \# \widehat{\sqcup} \# \widehat{\sqcup} \# \dots \#$$

2. To simulate a move of M, S scans the tape from the first $\#$ to the $(k+1)$'st $\#$, to determine what the symbols are under the heads.
3. S then passes back through the tape to update it according to M's transition function operating on these symbols.

Simulating k tapes with 1 tape

Assume the input is a string $w_1 w_2 \dots w_n$.

1. To start with, S puts its tape into the format corresponding to a concatenation of all k tapes of M, i.e.:

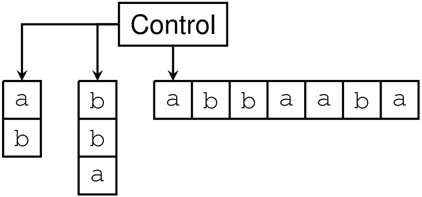
$$\#\widehat{w_1}w_2 \dots w_n \# \widehat{\sqcup} \# \widehat{\sqcup} \# \dots \#$$

2. To simulate a move of M, S scans the tape from the first $\#$ to the $(k+1)$'st $\#$, to determine what the symbols are under the heads.
3. S then passes back through the tape to update it according to M's transition function operating on these symbols.
4. At some point S may move one of the k virtual heads rightwards onto a $\#$. This implies that M has moved that physical head onto a blank square of the tape. In this situation, S shifts everything to the right of that cell one position to the right, and then writes a $\widehat{\sqcup}$ symbol onto that tape cell.

Other equivalent models

There are also machines which may seem weaker (or very different), but which turn out to be equivalent to Turing machines.

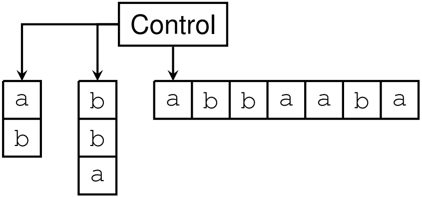
- First, a pushdown automaton with two stacks:



Other equivalent models

There are also machines which may seem weaker (or very different), but which turn out to be equivalent to Turing machines.

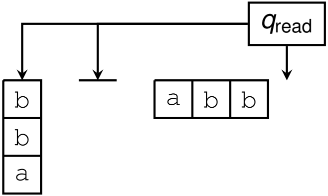
- First, a pushdown automaton with two stacks:



- The basic idea is that one stack represents the tape extending to the left, the other the tape extending to the right (including the cell under the head).
- The PDA begins by reading in the input into the second stack. Then it simulates the operation of a Turing machine using the stacks.

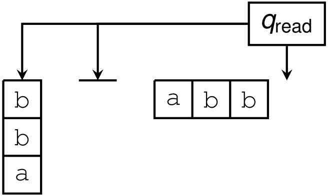
PDAs with two stacks

- The state of the PDA after it has first read in the input looks like

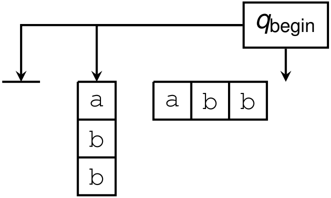


PDAs with two stacks

- The state of the PDA after it has first read in the input looks like



- The PDA repeatedly pops the top symbol from the first stack and pushes it onto the second stack. The resulting state is

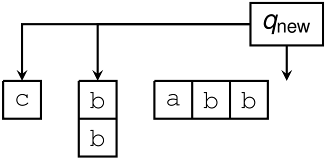


- This corresponds to the starting configuration of a Turing machine.

PDAs with two stacks

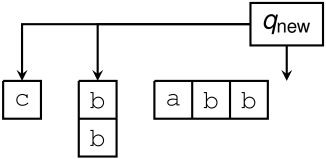
- The PDA can look at the top element of the second stack to determine which operations to perform.

- So if (for example) the current state of the Turing machine wants to replace the current symbol with c and move to the right, the new configuration of the PDA would be



PDAs with two stacks

- The PDA can look at the top element of the second stack to determine which operations to perform.
- So if (for example) the current state of the Turing machine wants to replace the current symbol with c and move to the right, the new configuration of the PDA would be



- The PDA then continues to simulate the Turing machine in this way until it accepts or rejects.

Counter machines

Consider the following, very simple model of a “real” computer:

- We have a small, fixed number of registers, each containing an integer (which can be arbitrarily large).
- We have an instruction set containing only the instructions
  - INC r: increment register r
  - DEC r: decrement register r (never going below 0)
  - JZ r, k: if $r = 0$, jump to instruction k, otherwise continue.

Counter machines

Consider the following, very simple model of a “real” computer:

- We have a small, fixed number of registers, each containing an integer (which can be arbitrarily large).
- We have an instruction set containing only the instructions
  - INC r: increment register r
  - DEC r: decrement register r (never going below 0)
  - JZ r, k: if $r = 0$, jump to instruction k, otherwise continue.
- If we have a register Z always set to 0, we can define GOTO k as shorthand for JZ Z, k.

Counter machines

Consider the following, very simple model of a "real" computer:

- We have a small, fixed number of registers, each containing an integer (which can be arbitrarily large).
- We have an instruction set containing only the instructions
  - INC r: increment register r
  - DEC r: decrement register r (never going below 0)
  - JZ r, k: if $r = 0$, jump to instruction k, otherwise continue.
- If we have a register Z always set to 0, we can define GOTO k as shorthand for JZ Z, k.

An example program fragment on a counter machine:

```
1. DEC B
2. JZ B, 4
3. GOTO 1
4. ...
```

Counter machines

Consider the following, very simple model of a “real” computer:

- We have a small, fixed number of registers, each containing an integer (which can be arbitrarily large).
- We have an instruction set containing only the instructions
  - INC r: increment register r
  - DEC r: decrement register r (never going below 0)
  - JZ r, k: if $r = 0$, jump to instruction k, otherwise continue.
- If we have a register Z always set to 0, we can define GOTO k as shorthand for JZ Z, k.

An example program fragment on a counter machine:

```
1. DEC B
2. JZ B, 4
3. GOTO 1
4. ...
```

The above code zeroes register B before continuing (call this “ZERO B”).

Counter machines

What does the following code do?

```
1. ZERO B
2. ZERO C
3. JZ A, 8
4. DEC A
5. INC B
6. INC C
7. GOTO 3
8. JZ C, 12
9. INC A
10. DEC C
11. GOTO 8
12. ...
```

Counter machines

What does the following code do?

```
1. ZERO B
2. ZERO C
3. JZ A, 8
4. DEC A
5. INC B
6. INC C
7. GOTO 3
8. JZ C, 12
9. INC A
10. DEC C
11. GOTO 8
12. ...
```

It copies register A to register B, using register C as temporary storage.

Counter machines can simulate Turing machines

Our simulation will be based around having three registers:

- H stores the tape symbol under the head;
- L stores the tape to the left of the head;
- R stores the tape to the right of the head (up to the first blank symbol).

Counter machines can simulate Turing machines

Our simulation will be based around having three registers:

- H stores the tape symbol under the head;
- L stores the tape to the left of the head;
- R stores the tape to the right of the head (up to the first blank symbol).

Assume the tape alphabet is $\{\mathtt{0}, \mathtt{1}, \sqcup\}$. We store multiple tape symbols in one register by encoding them as an integer written in binary.

- If the tape to the right of the head is $r_0 r_1 \dots r_{m-1}$ (going left to right), the integer in R is $\sum_{i=0}^{m-1} r_i 2^i$.

Counter machines can simulate Turing machines

Our simulation will be based around having three registers:

- H stores the tape symbol under the head;
- L stores the tape to the left of the head;
- R stores the tape to the right of the head (up to the first blank symbol).

Assume the tape alphabet is $\{0, 1, \sqcup\}$. We store multiple tape symbols in one register by encoding them as an integer written in binary.

- If the tape to the right of the head is $r_0 r_1 \dots r_{m-1}$ (going left to right), the integer in R is $\sum_{i=0}^{m-1} r_i 2^i$.
- Similarly, if the tape to the left of the head is $\ell_0 \ell_1 \dots \ell_{m-1}$ (going right to left), the integer in L is $\sum_{i=0}^{m-1} \ell_i 2^i$.

Counter machines can simulate Turing machines

Our simulation will be based around having three registers:

- H stores the tape symbol under the head;
- L stores the tape to the left of the head;
- R stores the tape to the right of the head (up to the first blank symbol).

Assume the tape alphabet is $\{0, 1, \sqcup\}$. We store multiple tape symbols in one register by encoding them as an integer written in binary.

- If the tape to the right of the head is $r_0 r_1 \dots r_{m-1}$ (going left to right), the integer in R is $\sum_{i=0}^{m-1} r_i 2^i$.
- Similarly, if the tape to the left of the head is $\ell_0 \ell_1 \dots \ell_{m-1}$ (going right to left), the integer in L is $\sum_{i=0}^{m-1} \ell_i 2^i$.

e.g.:

0	1	1	0	0	1	$\sqcup$	$\cdots$

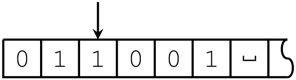
Counter machines can simulate Turing machines

Our simulation will be based around having three registers:

- H stores the tape symbol under the head;
- L stores the tape to the left of the head;
- R stores the tape to the right of the head (up to the first blank symbol).

Assume the tape alphabet is $\{\mathtt{0}, \mathtt{1}, \sqcup\}$. We store multiple tape symbols in one register by encoding them as an integer written in binary.

- If the tape to the right of the head is $r_0 r_1 \dots r_{m-1}$ (going left to right), the integer in R is $\sum_{i=0}^{m-1} r_i 2^i$.
- Similarly, if the tape to the left of the head is $\ell_0 \ell_1 \dots \ell_{m-1}$ (going right to left), the integer in L is $\sum_{i=0}^{m-1} \ell_i 2^i$.

e.g.: 

$\cdots$ H$=1$, L$=1$, R$=4$.

Ashley Montanaro
ashley@cs.bris.ac.uk

Counter machines can simulate Turing machines

- We start the simulation by having the register H contain the first symbol on the tape, and R contain the rest of the input.
- Then the simulation proceeds similarly to the two-stack PDA, where we think of L as containing the first stack and R the second stack.

Counter machines can simulate Turing machines

- We start the simulation by having the register H contain the first symbol on the tape, and R contain the rest of the input.
- Then the simulation proceeds similarly to the two-stack PDA, where we think of L as containing the first stack and R the second stack.

A sketch of how this works:

1. We have different code for each possible state of the Turing machine.
2. We can check whether the symbol under the head is 0 or 1 using JZ.
3. Imagine it is 0 and in the current state we should write a 1 and move right (the other cases are similar). Then:
   - We multiply L by 2 and then add 1 to L;
   - We update H to contain the lowest bit of R;
   - We divide R by 2 (delete the lowest bit).

The multiplication and division operations can be performed using a sequence of increments and decrements (exercise!).

Extremely small counter machines

- Using this basic idea, we can simulate a Turing machine using 5 registers (the H, L and R registers, the Z register and another "workspace" register).

Extremely small counter machines

- Using this basic idea, we can simulate a Turing machine using 5 registers (the H, L and R registers, the Z register and another "workspace" register).

- This can be reduced to 2 registers if a trick called Gödel encoding is used, where several integers $z_1, \dots, z_k$ are encoded as a product of prime numbers raised to corresponding powers:

$$z_1, z_2, \dots, z_k \quad \mapsto \quad 2^{z_1} 3^{z_2} 5^{z_3} \dots$$

Extremely small counter machines

- Using this basic idea, we can simulate a Turing machine using 5 registers (the H, L and R registers, the Z register and another "workspace" register).

- This can be reduced to 2 registers if a trick called Gödel encoding is used, where several integers $z_1, \dots, z_k$ are encoded as a product of prime numbers raised to corresponding powers:

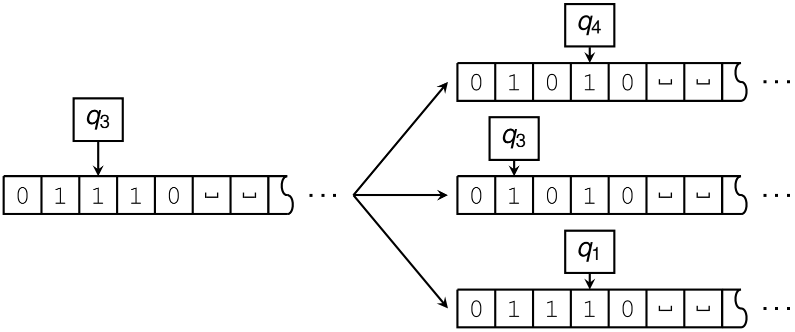
$$z_1, z_2, \dots, z_k \quad \mapsto \quad 2^{z_1} 3^{z_2} 5^{z_3} \dots$$

- Thus, assuming that the Church-Turing thesis is true, a computer with 2 registers and 3 instructions can compute everything which can be computed!

Nondeterministic Turing machines

Just as we can have nondeterministic finite and pushdown automata, we can have nondeterministic Turing machines (NDTMs).

- An NDTM can explore multiple computational paths simultaneously.
- It accepts if and only if at least one of the computational paths accepts.



Nondeterministic Turing machines

- NDTMs are formally described in the same way as TMs, except that their transition function is of the form

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

- The right-hand side of this expression specifies the set of new configurations reachable from any given configuration. We think of this as exploring a tree of possible computation paths simultaneously.
- An NDTM accepts if any of the computation paths leads to the accepting state q_{accept}.

Nondeterministic Turing machines

- NDTMs are formally described in the same way as TMs, except that their transition function is of the form

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

- The right-hand side of this expression specifies the set of new configurations reachable from any given configuration. We think of this as exploring a tree of possible computation paths simultaneously.
- An NDTM accepts if any of the computation paths leads to the accepting state q_{accept}.
- Unlike TMs, we do not believe we can really implement NDTMs physically.
- Nevertheless, they are a useful theoretical tool…

Nondeterministic Turing machines

- NDTMs are formally described in the same way as TMs, except that their transition function is of the form

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

- The right-hand side of this expression specifies the set of new configurations reachable from any given configuration. We think of this as exploring a tree of possible computation paths simultaneously.
- An NDTM accepts if any of the computation paths leads to the accepting state q_{accept}.
- Unlike TMs, we do not believe we can really implement NDTMs physically.
- Nevertheless, they are a useful theoretical tool…

Are NDTMs more powerful than TMs?

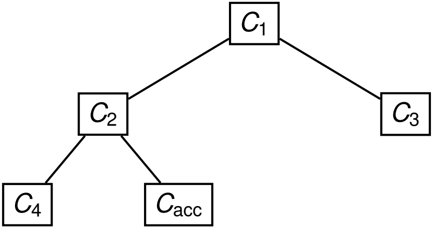
Nondeterministic Turing machines

Theorem

Every NDTM has an equivalent TM.

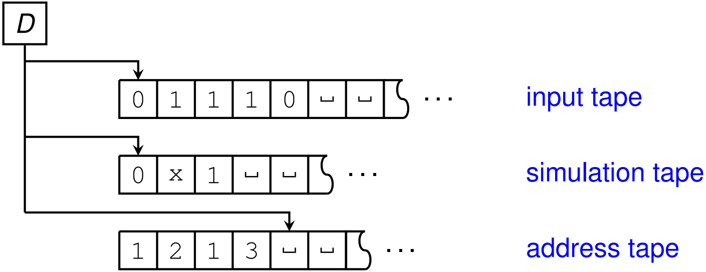
Proof idea

- Given an NDTM N, we form a deterministic multitape TM D by trying all possible branches of its computation tree.
- D accepts if any of these branches accept. Otherwise, D runs forever.
- We explore this tree using breadth-first search.



Ashley Montanaro
ashley@cs.bris.ac.uk

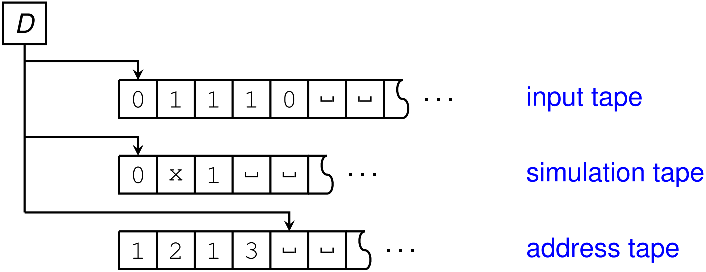
The simulation (sketch)



Our simulating TM will have three tapes:

- The input tape contains the original input to the NDTM N.
- The simulation tape contains the current contents of N's tape.
- The address tape describes a sequence of states which can be reached by N.

The simulation (sketch)



The simulation proceeds as follows:

1. Initialise the simulation and address tapes to be empty.
2. Copy the input tape to the simulation tape.
3. Simulate some steps of N on the simulation tape, according to the address tape. Accept if N accepts.
4. Replace the address tape with the next valid identifier of a branch of N's computation, and go to step (2).

The universal Turing machine

The Turing machine model is powerful enough to simulate itself.

- In other words, there exists a Turing machine U which can simulate the operation of any other Turing machine M, given a description of M as input.
- Intuitively, this is like having an interpreter for a programming language written in the language itself.
- We show this by sketching a 2-tape TM U which simulates the operation of any 1-tape TM M (as we have seen, U can then be converted to only use 1 tape).

The universal Turing machine

1. The first tape of U stores a description $\langle M \rangle$ of M (for example, M's transition function). The second tape will store M's configuration at any point in the computation.

The universal Turing machine

1. The first tape of U stores a description $\langle M \rangle$ of M (for example, M's transition function). The second tape will store M's configuration at any point in the computation.

2. At each step of the computation, U reads M's transition function to decide what step to perform next and updates the configuration on the second tape appropriately.

The universal Turing machine

1. The first tape of U stores a description $\langle M \rangle$ of M (for example, M's transition function). The second tape will store M's configuration at any point in the computation.

2. At each step of the computation, U reads M's transition function to decide what step to perform next and updates the configuration on the second tape appropriately.

3. If the simulation of M enters the accept or reject state, U accepts or rejects (respectively).

Summary and further reading

- The Church-Turing thesis states that everything which can be computed can be computed by a Turing machine.

Summary and further reading

- The Church-Turing thesis states that everything which can be computed can be computed by a Turing machine.
- Some evidence for this: Turing machines are equivalent to other models of computation such as multitape Turing machines, pushdown automata with two stacks and counter machines.

Summary and further reading

- The Church-Turing thesis states that everything which can be computed can be computed by a Turing machine.
- Some evidence for this: Turing machines are equivalent to other models of computation such as multitape Turing machines, pushdown automata with two stacks and counter machines.
- The nondeterministic Turing machine model is apparently more powerful but can also be simulated by a deterministic Turing machine.

Summary and further reading

- The Church-Turing thesis states that everything which can be computed can be computed by a Turing machine.
- Some evidence for this: Turing machines are equivalent to other models of computation such as multitape Turing machines, pushdown automata with two stacks and counter machines.
- The nondeterministic Turing machine model is apparently more powerful but can also be simulated by a deterministic Turing machine.
- There are universal Turing machines which can simulate any other Turing machine given as input.

Summary and further reading

- The Church-Turing thesis states that everything which can be computed can be computed by a Turing machine.
- Some evidence for this: Turing machines are equivalent to other models of computation such as multitape Turing machines, pushdown automata with two stacks and counter machines.
- The nondeterministic Turing machine model is apparently more powerful but can also be simulated by a deterministic Turing machine.
- There are universal Turing machines which can simulate any other Turing machine given as input.
- Further reading: Sipser §3.1-3.3, and *Computation: Finite and Infinite Machines* (Marvin L. Minsky, Prentice-Hall, 1967).