Lecture 23: Turing Machines

Examples

MATH230

Te Kura Pāngarau | School of Mathematics and Statistics
Te Whare Wānanga o Waitaha | University of Canterbury

Outline

Turing Machine Format

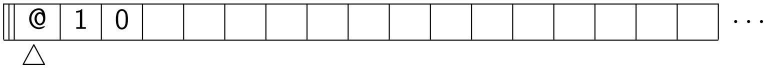
All machine instructions for tutorial and assignment questions must be submitted using the following format:

1. Written to a .txt file with a helpful name
2. Comments are lines beginning with #
3. Instructions are to be written in standard form
4. Instructions end with a semicolon and a newline
5. The final line must either be a comment or a blank line

This is so your instructions can be read by the script available on learn. This will help you visualise what your instructions do on particular inputs, keep record of your work, and more easily adjust your instructions. It will also help me check your instructions.

Example

Write a Turing machine to NAND two bits.

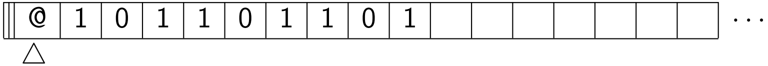


Write the output bit to the right of the home square.

C	R	P	M	U
q_0	@	@	R	

Example

Write a Turing machine to check if a binary input is a palindrome



Print 1 next to home if the input is a palindrome, otherwise print 0.

When we are trying to write instructions to do a task like this, we should always start with a high-level idea.

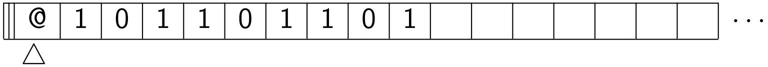
Once we have that it's a matter of breaking the high-level idea into specific subroutines. One thing you always need to think about is how you will detect whether the task is complete?

Example

How will we know whether the number is a palindrome?

Example

Write a Turing machine to check if a binary input is a palindrome

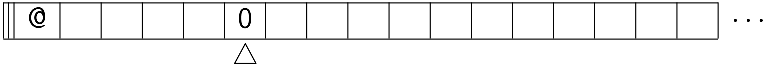


Let's write some instructions to get the back-and-forth checking.

C	R	P	M	U
q_0	@	@	R	

Example

Write a Turing machine to check if a binary input is a palindrome



Which instructions do we need to move to in order to output the appropriate bit?

C	R	P	M	U

E-Squares and F-Squares

In his 1936 paper Turing offers an example machine which prints the sequence

$$001011011101111011111\ldots$$

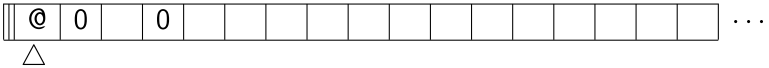
This machine never halts and prints increasingly longer sequences of 1s separated by 0s.

In this example he illustrates the use of E-squares and F-squares.

> E-squares are to *only* hold symbols for working
>
> F-squares are to *only* hold content symbols

E-Squares and *F*-Squares

Turing explains that the output can be printed on every second square so that the alternating squares can be used for markings to keep track of where one is in the calculation.

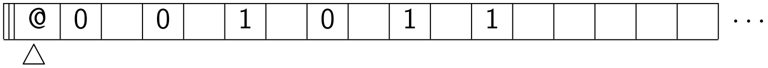


Turing suggests the following steps to obtain the desired output.

1. Find the next blank *F*-square and print a 1.
2. Search left for 1s in previous block. Print x next to them.
3. Use each x as a marker to write a 1 to the end of the tape.
4. Once all x symbols gone, write a 0 at the end of the tape.
5. Restart this loop from the first instruction.

E-Squares and *F*-Squares

Use *x* markers to help keep track of what needs to be copied.



Turing suggests the following steps to obtain the desired output.

1. Find the next blank *F*-square and print a 1.
2. Search left for 1s in previous block. Print x next to them.
3. Use each x as a marker to write a 1 to the end of the tape.
4. Once all x symbols gone, write a 0 at the end of the tape.
5. Restart this loop from the first instruction.

Abbreviated Tables

Turing uses shortcuts when describing many of his machine instructions.

> *There are certain types of process used by nearly all machines... These processes include copying down of symbols, comparing sequences, erasing all symbols of a given form ... we can abbreviate the tables for $\mathfrak{m}$-configurations considerably by the use of "skeleton tables".*

This style of thinking is similar to that of high-level languages. Turing explains that one can always substitute the precise commands into (i.e. compile?) the skeleton table to get back the complete set of $\mathfrak{m}$-configurations.

This is yet another reason why Turing's paper is so influential.

Examples of Abbreviations

By way of example, we note that Turing used the following when describing the capabilities of Turing machines.

> Look left until symbol found, then go into particular state.
>
> He uses if then, else to describe conditions for updating states.
>
> Look left erasing all symbols of a particular kind.
>
> Find the end and print a particular symbol.

When thinking about, and explaining, Turing machines in a high-level language you should feel free to use such language.

Your first description of a machine does not have to be in terms of specific quintuples of state instructions.

Computational Equivalence

We now have three models of computation

Primitive Recursion

General μ-recursion

Turing machines

One then begins to wonder how they relate to one another. Let us say two models M_1 and M_2 of computation are equivalent if a function $f : \mathbb{N}^k \to \mathbb{N}^m$ can be computed using M_1 if and only if it can be computed using M_2.

If a model of computation is computationally equivalent to a Turing machine, then we say that model of computation is **Turing complete**.

μ-Recursion = Turing

Theorem: a function on the natural numbers is computable by a Turing machine if and only if there is a μ-recursive function for computing the output of the function.

Variations on a Theme

Each of the choices made in the definition can be changed for a machine of *apparently* different capabilities:

- More characters in the alphabet.
- Two-way infinite tapes.
- Multiple tapes.
- Multiple independent scanning heads.
- Multiple tapes and multiple scanning heads.
- Multiple dimensions.
- Infinite time Turing machines.

Each of these apparently more powerful ideas of a computing machine are computationally equivalent to having a single one-way infinite tape with a single scanning head.

Oracle Machines

In the coming lectures we will see that Turing's definition sufficed to show (solutions to) some problems are not computable by a Turing machine.

One reaction to this is to try and get an understanding of the *relative* difficulty of particular computations. If we grant the use of an *oracle* that solves a particular problem, then what other problems can such an oracle machine solve?

These machines are, by definition, more "computationally" powerful than Turing's original definition.

Thus showing a hierarchy of computational methods.

Relativity

This strategy is also apparent in the work of mathematicians and logicians. Gödel clarified the issue of independence phenomena of first-order axiomatic theories. Showing that some statements are not provable from the axioms and the method of deduction.

If α, β are independent to axioms Σ, then one might wonder how they relate to one another. Does assuming one give the other?

Specifically one might try to determine

$\Sigma \cup \{\alpha\} \vdash \beta$?

$\Sigma \cup \{\beta\} \vdash \alpha$?

What other Γ will yield $\Sigma \cup \Gamma \vdash \alpha$?

These sorts of questions yield hierarchies of computability and independence respectively. Active topics of research in computer science and mathematics to this day. Of particular interest to mathematicians is the hierarchy of set theories.

Further Reading

Here are some recommended reading to follow up on the lecture content.

- The Annotated Turing *Charles Petzold*
- SEP: Turing Machines