



Theory of Computation

Week 10

Much of the material on this slides comes from the recommended textbook by Elaine Rich

Announcement

❑ Midterm 2

- Date: 25/05/2020
- Time: 9:00~10:00 (AEST)
- Venue: Online Exam (Blackboard based)
- Syllabus: Week 6 to Week 9

Detailed content

Weekly program

- ✓ Week 1 – Background knowledge revision: logic, sets, proof techniques
- ✓ Week 2 – Languages and strings. Hierarchies. Computation. Closure properties
- ✓ Week 3 – Finite State Machines: non-determinism vs. determinism
- ✓ Week 4 – Regular languages: expressions and grammars
- ✓ Week 5 – Non regular languages: pumping lemma. Closure
- ✓ Week 6 – Context-free languages: grammars and parse trees
- ✓ Week 7 – Pushdown automata
- ✓ Week 8 – Non context-free languages: pumping lemma and decidability. Closure
- ✓ Week 9 – Decidable languages: Turing Machines

Week 10 – Church-Turing thesis and the unsolvability of the Halting Problem

- ☐ Week 11 – Decidable, semi-decidable and undecidable languages (and proofs)
- ☐ Week 12 – Revision of the hierarchy. Safety-critical systems
- ☐ Week 13 – Extra revision (if needed)

Week 10 Videos

You already know

- ☐ How to encode a TM
- ☐ What is Universal Turing Machine



Videos to watch before lecture



Additional videos to watch for this week

Week 10 Lecture

Church-Turing Thesis

- ❑ Simulating a “Real” Computer
- ❑ Alternate TM Definitions
 - ❑ One-Way VS Two-way infinite Tape
 - ❑ Stack VS Tape
- ❑ Universal Turing Machine
- ❑ Entscheidungsproblem
- ❑ Church-Turing Thesis

Simulating a Real Computer

- An unbounded number of memory cells addressed by the integers starting at 0.
- An instruction set composed of basic operations including load, store, add, subtract, jump, conditional jump, and halt. Here's a simple example program:

```
R      10
MIR    10
CJUMP  1001
A      10111
ST     10111
```

- A program counter.
- An address register.
- An accumulator.
- A small fixed number of special purpose registers.
- An input file.
- An output file.

Simulating a Real Computer

Theorem: A random-access, stored program computer can be simulated by a Turing Machine. If the computer requires n steps to perform some operation, the Turing Machine simulation will require $\mathcal{O}(n^6)$ steps.

Proof: By construction.

simcomputer will use 7 tapes:

- Tape 1: the computer's memory.
- Tape 2: the program counter.
- Tape 3: the address register.
- Tape 4: the accumulator.
- Tape 5: the op code of the current instruction.
- Tape 6: the input file.
- Tape 7: the output file, initially blank.

1: Memory
2: Program Counter
3: Address Register
4: Accumulator
5: Current Instruction
6: Input File
7: Output File

Representing Memory

Memory will be organized as a series of (address, value) pairs, separated by delimiters:

$\#0, val_0 \#1, val_1 \#10, val_2 \#11, val_3 \#100, val_4 \# \dots \#$

Instructions: four bit opcode followed by address. So our example program could look like:

$\#0, 000110010 \#1, 11111001 \#10, 001110011 \#11, 001010111 \# \dots$

Must delimit words because no bound on their length:

- Addresses may get longer as the simulated program uses more words of its memory.
- Numeric values may increase as old values are added to produce new ones.

Simcomputer

simcomputer(program) =

1. Move the input string to tape 6.
2. Initialize the program counter (tape 2) to 0.
3. Loop:
 - 3.1 Starting at the left of the nonblank portion of tape 1, scan right looking for an index that matches the contents of tape 2 (the program counter).

/ Decode the current instruction and increment the program counter.*

3.2 Copy the operation code to tape 5.

3.3 Copy the address to tape 3.

3.4 Add 1 to the value on tape 2.

.....#10,001110011#,.....

/ Retrieve the operand.*

3.5 Starting at the left again, scan right looking for the address stored on tape 3.

/ Execute the instruction.*

3.6 If the operation is Load, copy the operand to tape 4.

3.7 If the operation is Add, add the operand to the value on tape 4.

3.8 If the operation is Jump, copy the value on tape 3 to tape 2

3.9 And so forth for the other operations.

1: Memory
2: Program Counter
3: Address Register
4: Accumulator
5: Current Instruction
6: Input File
7: Output File

How Long Does it Take?

To simulate a program running n steps, *simcomputer*:

- Executes the outer loop of step 3 n times.
- Step 3.1 may take t steps. (t is the length of tape 1.)
- Step 3.2 takes a constant number of steps.
- Step 3.3 may take ' a ' steps if ' a ' is the number of bits required to store the longest address used on tape 1.
- Step 3.4 may also take ' a ' steps.
- Step 3.5 may have to scan all of tape 1, so it may take t steps.
- The number of steps required to execute the instruction varies:
 - Addition takes v steps if v is the length of the longer operand.
 - Load takes v steps if v is the length of the value to be loaded.
 - Store generally takes v steps if v is the length of the value to be stored. But if the value to be stored is longer than the value that is already stored at that location, *simcomputer* must shift the remainder of Tape 1 one square to the right. So executing a Store instruction could take t steps.

How Long Does it Take?

How long is the tape after n steps?

$$k + n^2$$

Assume that $n \geq k$:

$$\mathcal{O}(n^2)$$

Total time for n steps: $\mathcal{O}(n^3)$

To do it on a one-tape, standard Turing Machine is: $\mathcal{O}(n^6)$

Turing Machine Definitions

12





An alternative definition of a Turing machine:

\diamond is a wall. The TM cannot move to the left past \diamond .

So δ is constrained as follows:

(a) if the input symbol is \diamond , the action is \rightarrow , and

(b) \diamond can never be written.

\diamond		a	b	b	a			
------------	---	---	---	---	---	---	---	---

Does This Difference Matter?

13

Remember the goal:

Define a device that is:

- powerful enough to describe all computable things,
- simple enough that we can reason formally about it

Both definitions are simple enough to work with, although details may make specific arguments easier or harder.

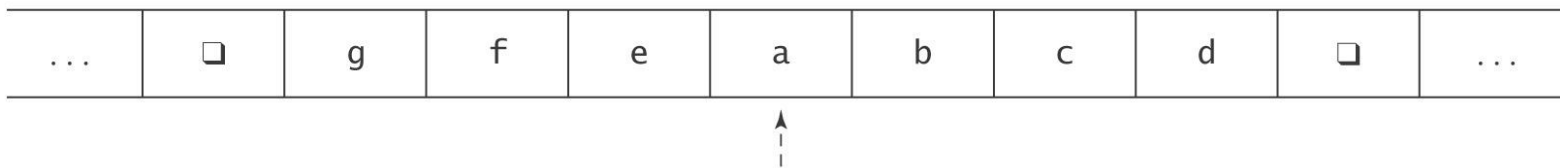
But, do they differ in their power?

Answer: No.

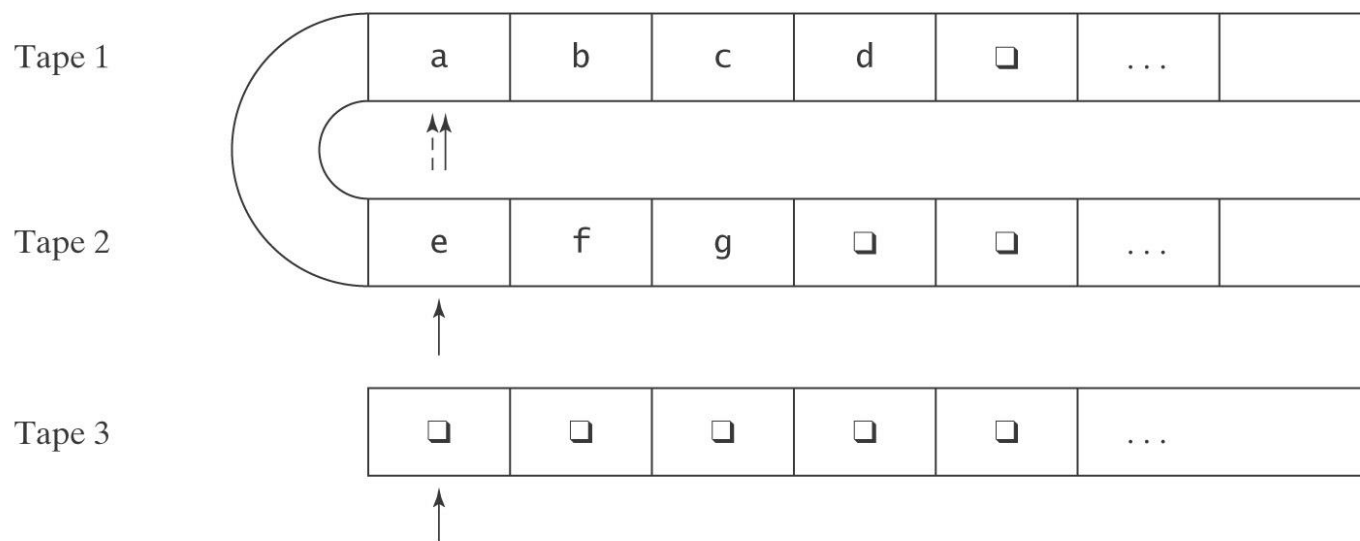
The Simulation

14

The two-way tape:



The simulation:



Stacks vs. a Tape

15

- Did we lose anything by giving up the PDA's stack in favor of the TM's tape?
- Could we have gotten the power of a TM's tape just with stacks?

Simulating a PDA

The components of a PDA:

- Finite state controller
- Input stream
- Stack

Tape 1
(Input)

...	a	a	a	b	b	□	...
-----	---	---	---	---	---	---	-----



Tape 2
Corresponding to

...	#	b	a	□	□	□	...
-----	---	---	---	---	---	---	-----



a
b

Running the Simulation

17

M will operate as follows:

1. Initialization: Write $\#$ under the read head of Tape 2. Set the simulated state S_{sim} to the start state of P .
2. Simulation: Let the character under the read head of Tape 1 be c .

At each step of P do:

2.1 If $c = \square$, halt and accept or reject.

2.2 Nondeterministically choose a transition:

$((S_{sim}, c, pop), (q_2, push))$, or

$((S_{sim}, \varepsilon, pop), (q_2, push))$.

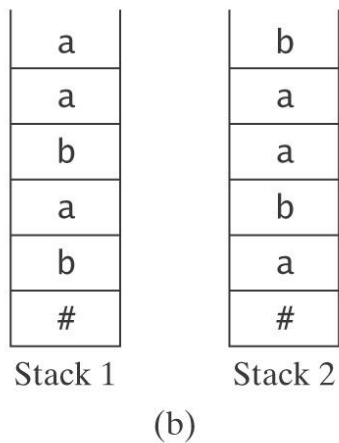
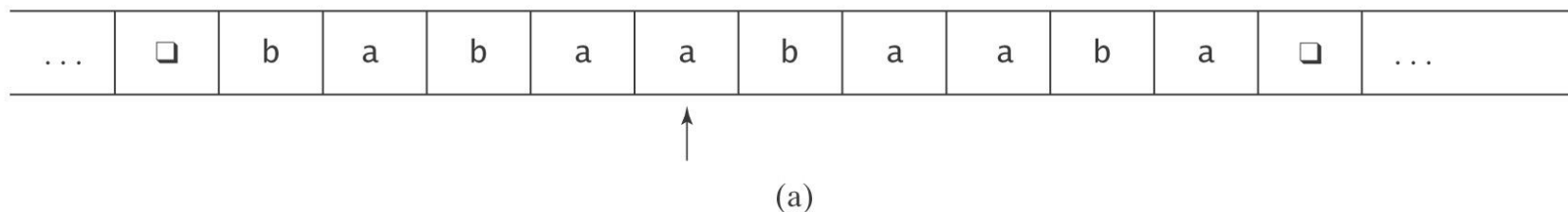
2.3 Scan left on Tape 2 $|pop|$ squares, checking that Tape 2 matches pop . If not, terminate this path. If it does, replace pop with $push$.

2.4 If we are not following an ε -transition, move the read head of Tape 1 one square to the right and set c to the character on that square.

2.5 Set S_{sim} to q_2 and repeat.

Simulating a Turing Machine with a PDA with Two Stacks

18



THE UNIVERSAL TURING MACHINE

19

Problem: All our machines so far are hardwired.

Question: Can we build a programmable TM that accepts as input:

program input string

executes the program, and outputs:

output string

THE UNIVERSAL TURING MACHINE



20

Yes, it's called the *Universal Turing Machine*.

To define the Universal Turing Machine U we need to:

1. Define an encoding operation for TMs.
2. Describe the operation of U given input $\langle M, w \rangle$, the encoding of:
 - a TM M , and
 - an input string w .

ENCODING A TURING MACHINE M



21

We need to describe $M = (K, \Sigma, \Gamma, \delta, s, H)$ as a string:

- The states
- The tape alphabet
- The transitions



ENCODING THE STATES

- Let i be $\lceil \log_2(|K|) \rceil$.
- Number the states from 0 to $|K|-1$ in binary:
 - Number s , the start state, 0.
 - Number the others in any order.
- If t' is the binary number assigned to state t , then:
 - If t is the halting state y , assign it the string $\underline{y}t'$.
 - If t is the halting state n , assign it the string $\underline{n}t'$.
 - If t is any other state, assign it the string $\underline{q}t'$.

ENCODING THE STATES

Example



23

Suppose M has 9 states.

$i = 4$

$s = q_10000,$

Remaining states (where y is 3 and n is 4):

$q_10001, q_10010, y_10011, n_10100,$
 $q_10101, q_10110, q_10111, q_11000$



ENCODING THE TAPE ALPHABET

The tape alphabet:

$ay : y \in \{0, 1\}^+,$

$|y| = j,$ and

j is the smallest integer such that $2^j \geq |\Gamma|.$

Example: $\Sigma = \{\square, a, b, c\}.$ $j = 2.$

$\square =$ a00

a = a01

b = a10

c = a11

ENCODING THE TRANSITIONS



25

The transitions: (state, input, state, output, move)

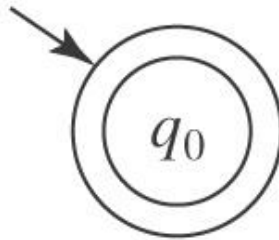
Example: (q000,a000,q110,a000,→)

Specify s as q000.

Specify H .

A SPECIAL CASE

We will treat this as a special case:



(q_0)

AN ENCODING EXAMPLE

27

Consider $M = (\{s, q, h\}, \{a, b, c\}, \{\square, a, b, c\}, \delta, s, \{h\})$:

state	symbol	δ
s	\square	$(q, \square, \rightarrow)$
s	a	(s, b, \rightarrow)
s	b	(q, a, \leftarrow)
s	c	(q, b, \leftarrow)
q	\square	(s, a, \rightarrow)
q	a	(q, b, \rightarrow)
q	b	(q, b, \leftarrow)
q	c	(h, a, \leftarrow)

state/symbol	representation
s	q00
q	q01
h	h10
\square	a00
a	a01
b	a10
c	a11

$\langle M \rangle = (q00, a00, q01, a00, \rightarrow), (q00, a01, q00, a10, \rightarrow),$
 $(q00, a10, q01, a01, \leftarrow), (q00, a11, q01, a10, \leftarrow),$
 $(q01, a00, q00, a01, \rightarrow), (q01, a01, q01, a10, \rightarrow),$
 $(q01, a10, q01, a11, \leftarrow), (q01, a11, h11, a01, \leftarrow)$

ENUMERATING TURING MACHINES

28

Now that we have an encoding scheme for Turing Machines, it is possible to create an enumeration of them

Theorem: There exists an infinite lexicographic enumeration of:

- (a) All syntactically valid TMs.
- (b) All syntactically valid TMs with specific input alphabet Σ .
- (c) All syntactically valid TMs with specific input alphabet Σ and specific tape alphabet Γ .

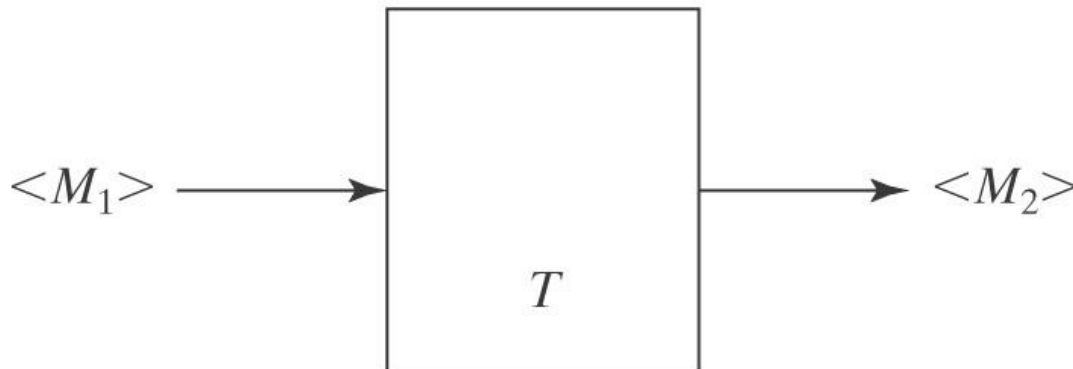
Proof:

$$\Sigma = \{ (,), a, q, y, n, 0, 1, \text{comma}, \rightarrow, \leftarrow \}$$

ADVANTAGES OF ENCODING

29

- Our motivation for defining $\langle M \rangle$ was that we would be able to input a definition of M to the UNIVERSAL TM, which will then execute M .
- But we can also create a TM T which takes the description of another as input and transforms it into another which can perform different (possibly related) tasks.
- We can talk about operations on programs (TMs)!

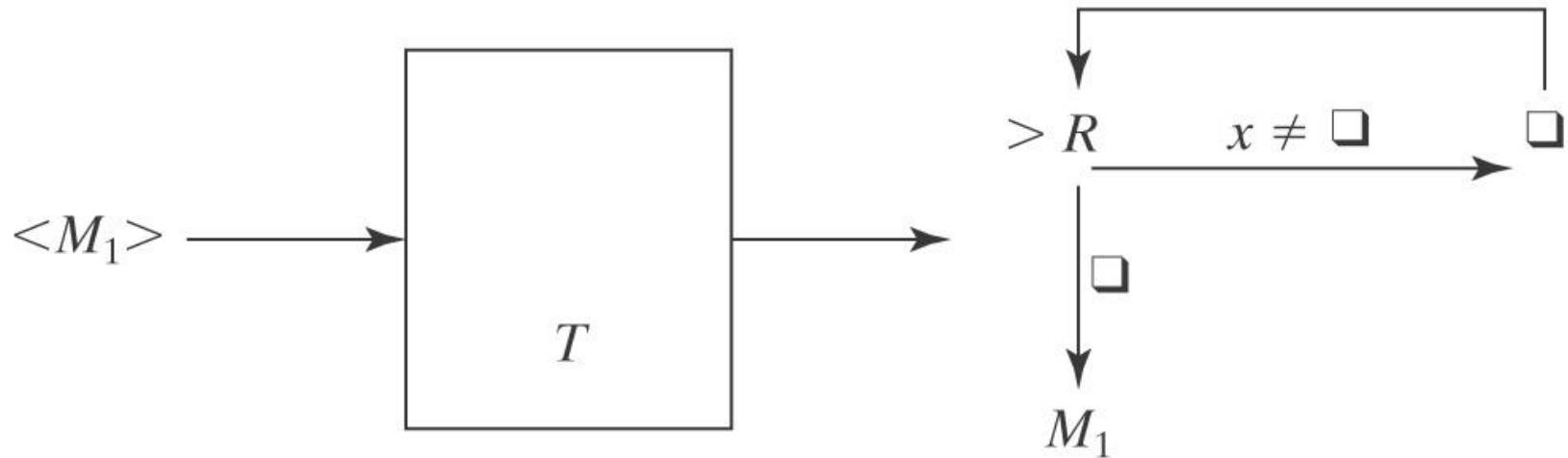


EXAMPLE OF A TRANSFORMING TM T

30

Input: a TM M_1 that reads its input tape and performs some operation P on it.

Output: a TM M_2 that performs P on an empty input tape.



ENCODING MULTIPLE INPUTS

31

- Sometimes we wish to define a TM that operates on more than one object.

For example the Universal TM will accept a machine M and a string w and simulate the execution of M on w

- We will use the notation

$$\langle x_1, x_2, \dots, x_n \rangle$$

to mean a single string that encodes the sequence of individual values:

$$x_1, x_2, \dots, x_n.$$

THE SPECIFICATION OF THE UNIVERSAL TM



32

On input $\langle M, w \rangle$, U must:

- Halt iff M halts on w .
- If M is a deciding or semideciding machine, then:
 - If M accepts, accept.
 - If M rejects, reject.
- If M computes a function, then $U(\langle M, w \rangle)$ must equal $M(w)$.



HOW U WORKS

U will use 3 tapes:

- Tape 1: M 's tape.
- Tape 2: $\langle M \rangle$, the “program” that U is running.
- Tape 3: will contain the encoding of M 's state at any stage in the simulation (the program counter)



HOW U WORKS

As any TM, the Universal TM starts with the input on tape 1 and the other two blank

□	$\langle M$	-----	-----	$M,$	w -----	-----	$w \rangle$	□	□
	1	0	0	0	0	0	0		
	□	□	□	□	□	□	□		
	1	0	0	0	0	0	0		
	□	□	□	□	□	□	□		
	1	0	0	0	0	0	0		

Note: we encode the pointer under the tape with a '1'

HOW U WORKS

Initialization of U :

1. Copy $\langle M \rangle$ onto tape 2.
2. Look at $\langle M \rangle$, figure out what i is, and write the encoding of state s on tape 3.

After initialization:

□	□	□	□	□	$\langle w \text{-----} w \rangle$			□	□
	0	0	0	0	1	0	0		
	$\langle M \text{-----} M \rangle$				□	□	□		
	1	0	0	0	0	0	0		
	q	0	0	0	□	□	□		
	1	□	□	□	□	□	□		

HOW U WORKS

□	□	□	□	□	$\langle w \text{---} \text{---} w \rangle$			□	□
	0	0	0	0	1	0	0		
	$\langle M \text{---} \text{---} M \rangle$				□	□	□		
	1	0	0	0	0	0	0		
	q	0	0	0	□	□	□		
	1	□	□	□	□	□	□		

Simulate the steps of M :

1. Until M would halt do:

- 1.1 Scan tape 2 for a quintuple that matches the current state, input pair.
- 1.2 Perform the associated action, by changing tapes 1 and 3. If necessary, extend the tape.
- 1.3 If no matching quintuple found, halt.

2. Report the same result M would report.

Recall from our first lecture

DECISION PROBLEMS

37

A ***decision problem*** is simply a problem for which the answer is yes or no (True or False). A ***decision procedure*** answers a decision problem.

The language recognition problem:
Given a language L and a string w , is w in L ?



Our focus

TURING MACHINES “DECIDE”

38

M **decides** a language $L \subseteq \Sigma^*$ iff:

For any string $w \in \Sigma^*$ it is true that:

- if $w \in L$ then M accepts w , and
- if $w \notin L$ then M rejects w .

A language L is **decidable** iff there is a Turing machine M that decides it. In this case, we will say that L is in **D** .

TURING MACHINES “SEMI-DECIDE”

39

Let Σ_M be the input alphabet to a TM M . Let $L \subseteq \Sigma_M^*$.

M **semidecides** L iff, for any string $w \in \Sigma_M^*$:

- $w \in L \rightarrow M$ accepts w
- $w \notin L \rightarrow M$ does not accept w .

M may either:
reject or
fail to halt.

A language L is **semidecidable** iff there is a Turing machine that semidecides it. We define the set **SD** to be the set of all semidecidable languages.

TURING MACHINES “COMPUTE”

40

Let $M = (K, \Sigma, \Gamma, \delta, s, \{h\})$. Its initial configuration is $(s, \sqcup w)$.

Define $M(w) = z$ iff $(s, \sqcup w) \vdash_M^* (h, \sqcup z)$.

Let $\Sigma' \subseteq \Sigma$ be M 's output alphabet.

Let f be any function from Σ^* to Σ'^* .

M **computes** f iff, for all $w \in \Sigma^*$:

- If w is an input on which f is defined: $M(w) = f(w)$.
- Otherwise $M(w)$ does not halt.

A function f is **recursive** or **computable** iff there is a Turing machine M that computes it and that always halts.

Are We Done?

FSM \Rightarrow PDA \Rightarrow Turing machine

Is this the end of the line?

There are still problems we cannot solve:

- There is a countably infinite number of Turing machines since we can lexicographically enumerate all the strings that correspond to syntactically legal Turing machines.
- There is an uncountably infinite number of languages over any nonempty alphabet.
- So there are more languages than there are Turing machines.

Are We Done?

42

Could we find something more powerful than a Turing Machine to recognise those languages?

Or more formally:

“Is there any computational algorithm that cannot be implemented by some Turing machine? And if so, can we find a more powerful model in which we could implement that algorithm?”

Are We Done?

In the last lecture we proved the following:

Theorem: If a nondeterministic TM M decides or semidecides a language, or computes a function, then there is a standard TM M' semideciding or deciding the same language or computing the same function.

So, it is not non-deterministic Turing Machines...

WHAT CAN ALGORITHMS DO?

44

1. Can we make all true statements theorems?

Is it possible to axiomatise all of the mathematical structure of interest in such a way that every true statement becomes a theorem?

2. Can we decide whether a statement is a theorem?

Does there exist an algorithm to decide, a given set of axioms, whether a given statement is theorem?

WHAT CAN ALGORITHMS DO?

45

1. Can we make all true statements theorems?

Can all facts be axiomatised?

2. Can we decide whether a statement is a theorem?

Can theoremhood be decided?

GÖDEL'S INCOMPLETENESS THEOREM

46

Kurt Gödel showed, in the proof of his Incompleteness Theorem [Gödel 1931], that the answer to question 1 is no.

- No consistent system of axioms whose theorems can be listed by an "effective procedure" (essentially, a computer program) is capable of proving all facts about the natural numbers.
 - For any such system, there will always be statements about the natural numbers that are true, but that are unprovable within the system.

The second incompleteness theorem shows that if such a system is also capable of proving certain basic facts about the natural numbers, then one particular arithmetic truth the system cannot prove is the consistency of the system itself.

THE ENTSCHEIDUNGSPROBLEM

[Hilbert and Ackermann 1928]

47

- Does there exist an algorithm to decide, given an arbitrary sentence w in first order logic, whether w is valid?
- Given a set of axioms A and a sentence w , does there exist an algorithm to decide whether w is entailed by A ?
- Given a set of axioms, A , and a sentence, w , does there exist an algorithm to decide whether w can be proved from A ?

THE ENTSCHEIDUNGSPROBLEM

48

To answer the question, in any of these forms, requires formalizing the definition of an algorithm:

- Turing: Turing machines.
- Church: lambda calculus.

Turing proved that Turing machines and the lambda calculus are equivalent.

CHURCH'S THESIS (CHURCH-TURING THESIS)

49

All formalisms powerful enough to describe everything we think of as a computational algorithm are equivalent.

This isn't a formal statement, so we can't prove it. But many different computational models have been proposed and they all turn out to be equivalent.

THE CHURCH-TURING THESIS

50

Examples of equivalent formalisms:

- Modern computers (with unbounded memory)
- Lambda calculus
- Partial recursive functions
- Tag systems (FSM plus FIFO queue)
- Unrestricted grammars:

$$aSa \rightarrow B$$

- Post production systems
- Markov algorithms
- Conway's Game of Life
- One dimensional cellular automata
- DNA-based computing
- Lindenmayer systems

❑ Automata, Computability and Complexity. Theory and Applications

- By Elaine Rich

❑ Chapter 17:

- Page : 393-407.

❑ Chapter 18:

- Page : 411-414.