COMP2022
Models of Computation

Turing Machines are Robust

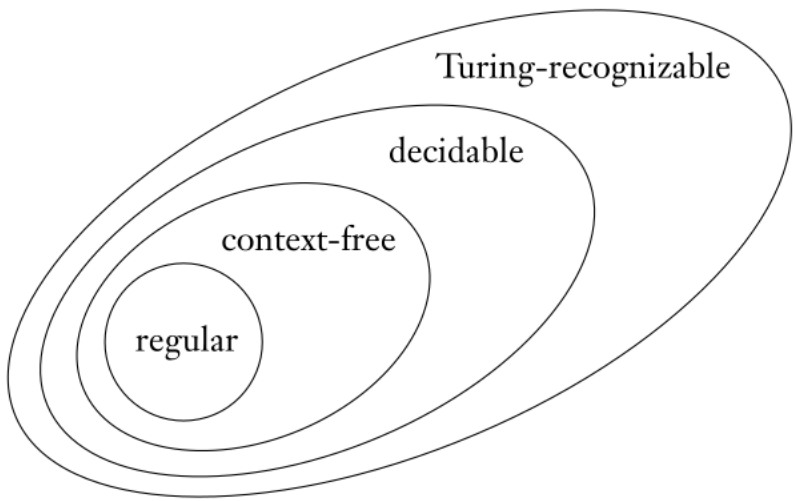
Sri AravindaKrishnan Thyagarajan (Aravind)

September 12, 2024

THE UNIVERSITY OF SYDNEY

Last lecture

- TMs are a machine model of computation, equivalent to many other models of computation.
- They are different from simpler models (e.g., DFA) because they have unrestricted access to unlimited memory.
- The **Church-Turing Thesis** says that the intuitive idea of an algorithm equals Turing machine algorithms.



Turing machines are fairly robust, i.e., variations and extensions of the basic model do not change the languages that can be recognised.[1]

- Let's call our TMs the basic TMs.
  - deterministic
  - single doubly-infinite tape
  - can move left, right, or stay
- Two machines are equivalent if they recognise the same language.

[1] Although "robustness" is not a formally defined concept, we will justify it with theorems about variations of TMs.

Variation 1: Must-move TMs

- Basic TMs can move left, right or stay put in one step.
- Sipser's variation disallows 'stay' — let's call them must-move TMs.
- So, obviously every must-move TM is a basic TM.
- Is every basic TM equivalent to a must-move TM?

Theorem

Every basic TM is equivalent to a must-move TM.

Proof idea: simulate a basic TM by a must-move TM

- Replace each S-transition by an R-transition followed by an L-transition, i.e., replace

q_i	a	b	S	q_j

by

q_i	a	b	R	q_i,j
q_i,j	*	*	L	q_j

* is any symbol

Variation 2: Left-bounded TM

- The head starts on the left-most square of the tape with the head on the first letter of the input.
- Should the transition function suggest to move left when the head is already at the left-most position, the head stays put.

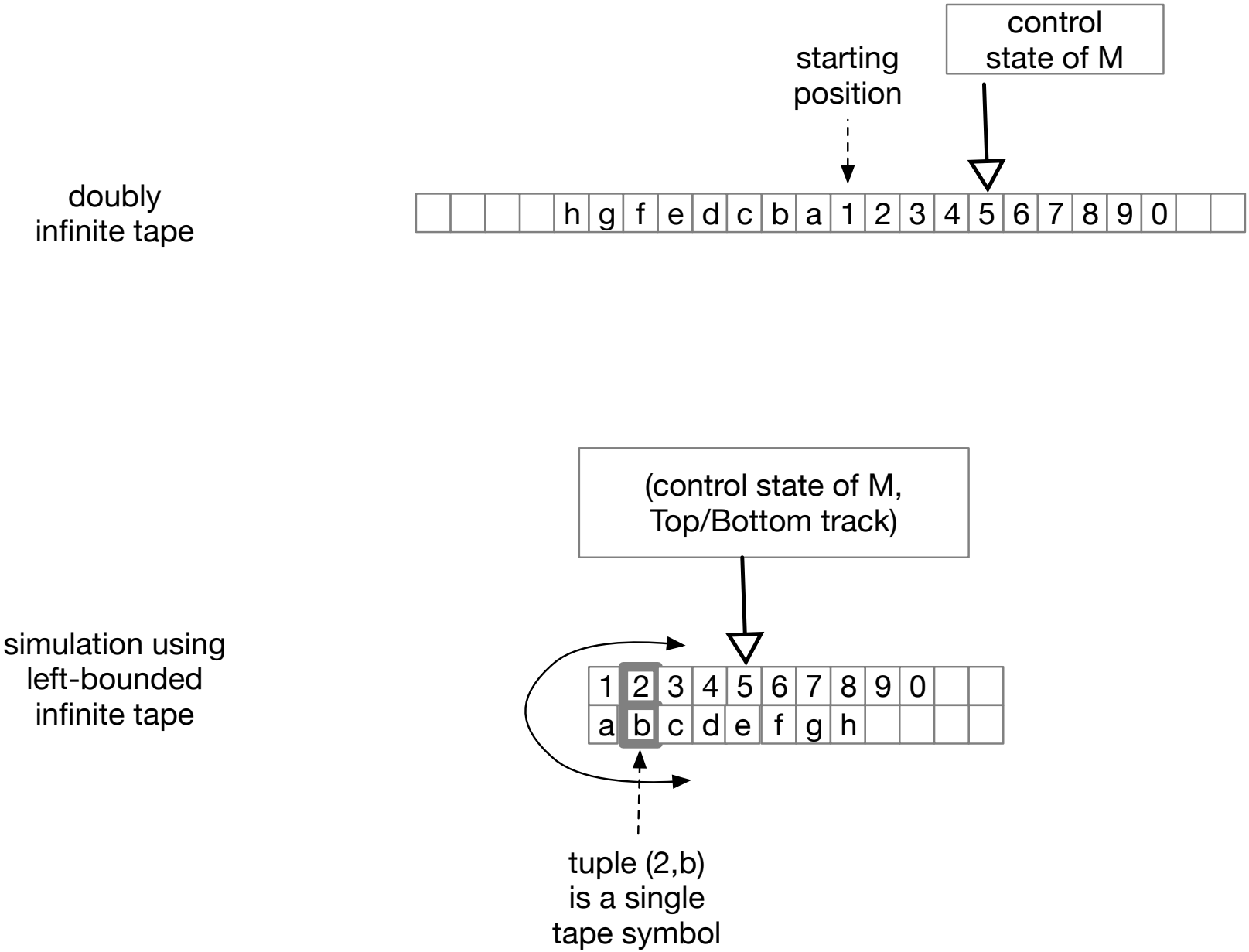
Theorem

Every basic TM is equivalent to a left-bounded TMs.[2]

Proof idea: simulate a basic TM by a left-bounded one

- Split the left-bounded tape into two tracks — upper and lower.
- Upper track represents the right half of the tape (from the initial head position)
- Lower track represents the left half of the tape
- An extra state component keeps track of the half in which the head currently is.
- At the start, mark the (leftmost) position so the TM knows when to switch between upper and lower.

[2] The vice-versa is a tutorial question.


control
state of M
starting
position
doubly
infinite tape
h g f e d c b a 1 2 3 4 5 6 7 8 9 0
(control state of M,
Top/Bottom track)
simulation using
left-bounded
infinite tape
1 2 3 4 5 6 7 8 9 0
a b c d e f g h
tuple (2,b)
is a single
tape symbol

Variation 3: Multitape TMs

- A multitape TM has multiple tapes, each with its own head for reading and writing.
- Initially the input appears on tape 1, and the others start out blank.
- The heads move simultaneously (not necessarily same direction).
- The type of the transition function of a k-tape TM becomes:

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

- Obviously every basic TM is a multitape TM (with $k = 1$).
- Is every multitape TM equivalent to a basic TM?

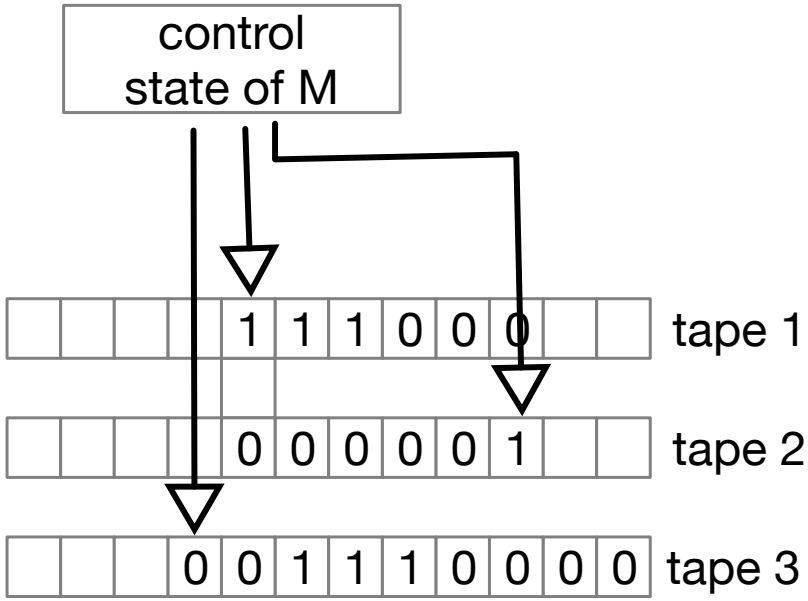
Multitape TMs

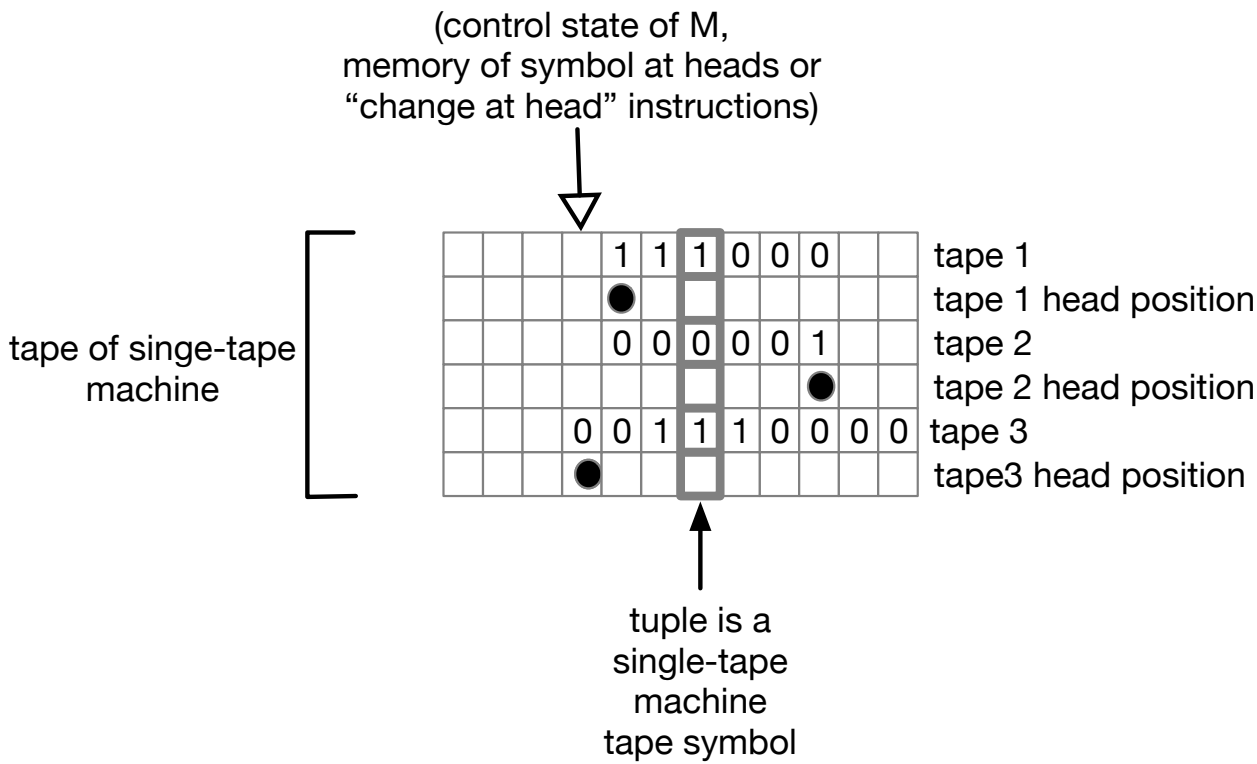
Theorem

Every multitape TM M has an equivalent basic TM B.

Proof Idea.

- Split the tape into $2k$-many tracks of the single tape of B.
- For each tape of M, use one track to store tape contents, and one track to mark head position on that tape.
- Each transition of M is simulated by a series of transitions of B.


control
state of M
1 1 1 0 0 0
tape 1
0 0 0 0 0 1
tape 2
0 0 0 1 1 1 0 0 0 0
tape 3


(control state of M,
memory of symbol at heads or
“change at head” instructions)
tape of singe-tape
machine
1 1 1 0 0 0
tape 1
tape 1 head position
0 0 0 0 0 1
tape 2
tape 2 head position
0 0 1 1 1 0 0 0 0
tape 3
tape3 head position
tuple is a
single-tape
machine
tape symbol

Takeaway

Turing machines are fairly robust, i.e., variations and extensions of the basic model do not change the languages that can be recognised.

1. The set of recognisable languages doesn't change if one makes certain variations (e.g., more tapes).
2. The set of decidable languages doesn't change if one makes certain variations (e.g., more tapes).

   This is because the simulations we just saw do not introduce diverging computations.

Closure properties

You will look at closure properties of the recognisable languages and the decidable languages in the tutorial. Let's get started...

We can think of a TM M as a procedure that takes an input x and outputs 1 ('halt-accept'), outputs 0 ('halt-reject'), or runs forever.

For instance:

```
1 def my_fun(x):
2   return not M(x)
```

- outputs 1 if M halts and rejects x
- outputs 0 if M halts and accepts x
- runs forever if M runs forever on x.

Note that if M is a decider, the third option never happens. What strings does `my_fun` accept?

Closure properties

Theorem

The complement of a decidable language is decidable.

Proof. Suppose that TM M decides language L. Build a TM from M as follows:

```
1  def my_fun(x):
2    return not M(x)
```

This TM decides the complement of L. □

Closure properties

Theorem

The union of decidable languages is decidable.

Proof.

Let L_1 and L_2 be languages that are decided by TMs M_1 and M_2 respectively. We construct a TM that decides $L_1 \cup L_2$ as follows:

```
1  def my_fun(x):
2    if M1(x):
3      return 1
4    if M2(x):
5      return 1
6    return 0
```

□

If L_1 and L_2 are recognisable, what language does the previous program recognise?

1. $L_1 \cup L_2$
2. L_1
3. some set X where $L_1 \subseteq X \subseteq L_1 \cup L_2$.

Corollary

The intersection of decidable languages is decidable.

Why? By De Morgan's Law for sets (see T0) which says that

$$X \cap Y = \overline{\overline{X} \cup \overline{Y}}$$

Theorem

A language is decidable exactly when it and its complement are recognisable.

Proof.

$\Rightarrow$: follows from the closure properties of decidable languages.

$\Leftarrow$: Say M_1 recognises L and M_2 recognises $\Sigma^* \setminus L$. We construct a decider for L as follows. On input w:

- Run M_1 and M_2 in parallel on w.
  - use one tape for each machine.
  - at each step, simulate M_1 on tape 1 and M_2 on tape 2.
- If M_1 ever accepts, then accept; and if M_2 ever accepts then reject.

The point is that exactly one of M_1, M_2 must eventually accept w, and so we have built a decider for L. $\square$

COMP2022|2922
Models of Computation

Decidability
Sipser Chapter 4

Sri AravindaKrishnan Thyagarajan (Aravind)

September 12, 2024

THE UNIVERSITY OF SYDNEY

Decidability

Recall:

- If you run a TM on an input one of three things can occur:
  1. the TM eventually halts in an accept state;
  2. the TM evenutally halts in a reject state;
  3. the TM doesn't enter a halting state (aka it diverges, i.e., 'runs forever').
- A TM is a decider if it halts on every input.
- A language L is called Turing-decidable if $L = L(M)$ for a decider M.

Encoding objects as strings

- The input to a Turing Machine is always a string, but it can be useful to work with objects other than strings over Σ.
- We can encode almost any object as a string (integers, sets of integers, graphs, programs, Turing machines!)
- The encoding of an object X may be written
  - $\langle X \rangle$ as in Sipser
  - str(X) as in Python
  - or just X
- The encoding of more than one object, say X and Y may be written $\langle X, Y \rangle$ or $X; Y$

Decidable problems about automata

The acceptance problem for DFAs[3] is the language

$$L_{\text{DFA-acceptance}} = \{D, w \mid D \text{ is a DFA that accepts } w\}$$

This problem is decidable.

Here is a high-level description of a decider for this language:

1. Simulate D on word w.
2. If D ends in an accepting state then accept, else reject.

[3] aka membership problem for DFAs

Decidable Problems about RE, Automata, CFGs

1. The acceptance-problems for DFA,NFA, RE, and CFG are decidable.
2. The emptiness-problems for DFA, NFA, RE, and CFG are decidable.
3. The equivalence-problems for DFA, NFA, and RE are decidable.[4]

[4] Fact: The equivalence problem for CFGs is not decidable.

Are there any decidable problems about TMs?

Acceptance problem for TMs

The acceptance problem for TMs is the language

$$L_{\text{TM-acceptance}} = \{M, w \mid M \text{ is a TM that accepts } w\}$$

- Read M as the string encoding of the source code of a TM M.
- This language is recognisable.
- To show this we will build a TM U that recognises it.

High level description of U:

- on input M, w:
- simulate M on w
- accept if M enters q_{accept} and rejects if M enters q_{reject} (and diverge otherwise).

The TM U is called a universal TM since it can do what any other TM can do.

Universal TM

Implementation level description of U:

- Tape 1 holds the transition function δ_M of the input TM M.
- Tape 2 holds the simulated contents of M's tape.
- Tape 3 holds the current state of M, and the current position of M's tape head.
- U simulates M on input x one step at a time:

  In each step, it updates M's state and simulated tape contents and head position as dictated by δ_M. If ever M halts and accepts or halts and rejects, then U does the same.

Universal TM

The universal TM U is a blueprint for an interpreter — it takes TMs as input and executes them.

This is similar to an interpreter of `Python` written in `Python`: a `Python` program that takes `Python` programs as input and executes them.

You can also view a universal TM as analogous to a computer.

Self-test

Is the universal TM U a decider?

1. Yes.
2. No.
3. It depends on the input.

Universal TM

The acceptance problem for TMs is the language

$$L_{\text{TM-acceptance}} = \{M, w \mid M \text{ is a TM that accepts } w\}$$

- TM U recognises $L_{\text{TM-acceptance}}$, but it doesn't decide it (since U is not a decider).
- So, is the acceptance problem for TMs decidable?

Theorem (Turing)

$L_{\text{TM-acceptance}} = \{M, w \mid M$ is a TM that accepts $w\}$ is not decidable.