COS210 - Theoretical Computer Science
Turing Machines and the Church-Turing Thesis (Part 3)

Equivalence of Multi- and Single-Tape Turing Machines

It may be easier and more efficient to solve a computational problem with a multi-tape Turing machine rather than with a single-tape Turing machine

However, single- and multi-tape Turing machines have the same descriptive power:

Theorem

Let $k \geq 1$ be an integer. Any k-tape Turing machine can be converted to an equivalent single-tape Turing machine

Equivalence of Multi- and Single-Tape Turing Machines

Informal proof:

How to convert a 2-tape Turing machine to an equivalent 1-tape machine

- Let $M = (Q, \Sigma, \Gamma, \delta, q, q_{accept}, q_{reject})$ be a **2**-tape Turing machine
- we construct an equivalent **1**-tape Turing machine $N = (Q', \Sigma, \Gamma', \delta', q', q'_{accept}, q'_{reject})$
- the machines M and N are **equivalent** if for every input string w over Σ the following holds:
  - M **accepts** w if and only if N **accepts** w
  - M **rejects** w if and only if N **rejects** w
  - M does **not terminate** on input w if and only if N does **not terminate** on input w

Equivalence of Multi- and Single-Tape Turing Machines

From $M = (Q, \Sigma, \Gamma, \delta, q, q_{accept}, q_{reject})$ to $N = (Q', \Sigma, \Gamma', \delta', q', q'_{accept}, q'_{reject})$

For N we use the following **extended tape alphabet**:

$$\Gamma' \;=\; \Gamma \cup \{\dot{a} : a \in \Gamma\} \cup \{\#\}$$

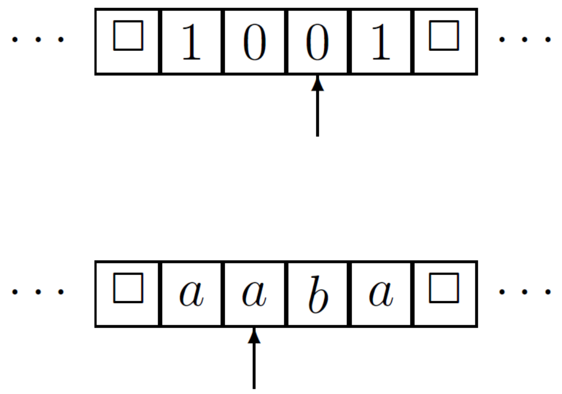
Example:

$$\text{If } \Gamma = \{0, 1, a, b, \square\}, \;\text{ then } \Gamma' = \{0, 1, a, b, \square, \dot{0}, \dot{1}, \dot{a}, \dot{b}, \dot{\square}, \#\}$$

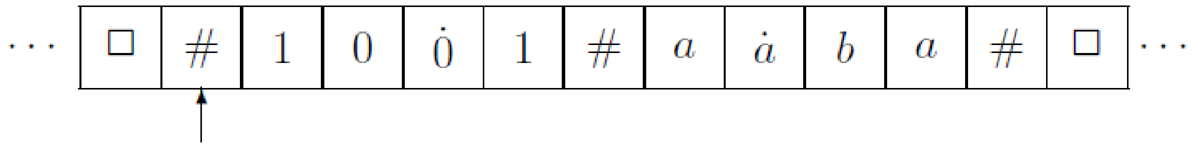
dotted copy or each original symbol, and extra symbol $\#$

Equivalence of Multi- and Single-Tape Turing Machines

A **two-tape configuration** like



shall be **encoded on a single tape** based on the extended alphabet



- The # **separates** the contents of the original tapes
- **Dotted symbols** represent positions of the **original tape heads**

Equivalence of Multi- and Single-Tape Turing Machines

$\cdots$	$\square$	$\#$	1	0	$\dot{0}$	1	$\#$	a	$\dot{a}$	b	a	$\#$	$\square$	$\cdots$

The machine N simulates each computational step of M as follows:

- At the start the tape head of N is at the leftmost symbol $\#$
- The tape head of N moves along the string from left to right
- While moving, the machine N **identifies and memorizes** the current **configuration of the original** M by means of states
  - N starts in state $q_{?,?}$ where original tape head positions are unknown
  - first dotted symbol is $\dot{0}$
    $\Rightarrow$ tape head 1 of M points at 0
    N switches to state $q_{0,?}$
  - second dotted symbol is $\dot{a}$
    $\Rightarrow$ tape head 2 of M points at a
    N switches to state $q_{0,a}$

Equivalence of Multi- and Single-Tape Turing Machines

$\cdots$	$\square$	$\#$	1	0	$\dot{0}$	1	$\#$	a	$\dot{a}$	b	a	$\#$	$\square$	$\cdots$

- In the state $q_{0,a}$ the machine N knows the current configuration of the original M
- Hence, N knows how M would update the contents of tape 1 and 2
- Possible updates of M:
  - replace symbols at tape heads
  - move tape heads
- How N simulates updates of M:
  - replace dotted symbols
  - 'move' the dots (via replacements)
- Simulation of updates may require to shift a part of the tape content to the right
- After the update, N identifies and memorizes updated configuration and simulates the next computational step of M

Example – Copy from First to Second Tape

Step:

1:

□ a b □
↑
□ □ □ □
↑

input string on tape 1
tape head 1 on leftmost symbol
tape 2 empty

2:

□ a b □
↑
□ a □ □
↑

copy first symbol to tape 2
move both tape heads to the right

3:

□ a b □
↑
□ a b □
↑

Copy second symbol to tape 2
move both tape heads to the right

copying done, switch to accept state

Example – Copy from First to Second Tape

Step:		
1:	□ □ # ȧ b # □̇ # □ □	tape head points at leftmost #
2:	□ □ # ȧ b # □̇ # □ □	head moves to first dotted symbol remember **a**
3:	□ □ # ȧ b # □̇ # □ □	head moves to second dotted symbol remember □
4:	□ □ # ȧ b # □̇ # □ □	tape head moves back to first dotted symbol

Example – Copy from First to Second Tape

Step:		
4:	□ □ # ȧ b # □̇ # □ □	tape head moves back to first dotted symbol
5:	□ □ # a b # □̇ # □ □	replace **ȧ** by **a**
6:	□ □ # a ḃ # □̇ # □ □	replace **b** by **ḃ**
7:	□ □ # a ḃ # □̇ # □ □	tape head moves to second dotted symbol

Example – Copy from First to Second Tape

Step:

7: $\square\ \square\ \#\ a\ \dot{b}\ \#\ \dot{\square}\ \#\ \square\ \square$

tape head moves to second dotted symbol

8: $\square\ \square\ \#\ a\ \dot{b}\ \#\ a\ \#\ \square\ \square$

replace $\dot{\square}$ by **a**

9: $\square\ \square\ \#\ a\ \dot{b}\ \#\ a\ \dot{\square}\ \square\ \square$

replace **#** by $\dot{\square}$

10: $\square\ \square\ \#\ a\ \dot{b}\ \#\ a\ \dot{\square}\ \#\ \square$

replace $\square$ by **#**
start moving to the left

Step:

Example – Copy from First to Second Tape

11: $\square\ \square\ \#\ a\ \dot{b}\ \#\ a\ \dot{\square}\ \#\ \square$

tape head moves to first dotted symbol, remember **b**

...

simulate that first dot moves to the right:

replace $\dot{\mathbf{b}}$ by **b**

insert $\dot{\square}$ after the **b**

shift remaining tape content to the right

12: $\square\ \square\ \#\ a\ b\ \dot{\square}\ \#\ a\ \dot{\square}\ \#\ \square$

13: $\square\ \square\ \#\ a\ b\ \dot{\square}\ \#\ a\ \dot{\square}\ \#\ \square$

tape head moves to second dotted symbol, remember $\square$

Step:

Example – Copy from First to Second Tape

13: $\square\ \square\ \#\ a\ b\ \dot{\square}\ \#\ a\ \dot{\square}\ \#\ \square$

tape head moves to second dotted symbol

...

simulate the insertion of **b** between **a** and $\dot{\square}$:
replace $\dot{\square}$ by **b**
replace **#** by $\dot{\square}$
replace $\square$ by **#**

14: $\square\ \square\ \#\ a\ b\ \dot{\square}\ \#\ a\ b\ \dot{\square}\ \#$

copying done,
switch to accept state

Turing Machines and Computability

Theorem

The following computational models are equivalent, i.e., any one of them can be converted to any of the other:

- *Single-tape Turing machines*
- *k-tape Turing machines*
- *Non-deterministic Turing machines*
- *Java programs*
- *$C++$ programs*

If some computational problem is solvable in general, then it is solvable by a Turing machine

Turing Machines and Computability

Knowing that these computation models are equivalent is important when trying to answer questions of the form

- Does these exist an algorithm X to solve problem Y
- Due to the equivalence between models showing there does or does not exist an algorithm under one model is sufficient
- E.g. if a problem cannot be solved by a Turing machine, then it also cannot be solved by a Java program

History Computability Theory

In 1900, the mathematician David Hilbert presented a list of problems that he considered crucial for the further development of mathematics. One of these problems is the following:

- Does there exist a finite process that decides whether or not any given polynomial equation with integer coefficients has an integer solution?

  Example:

  $$12x^3y^7z^5 + 7x^2y^4z - x^4 + y^2z^7 - z^3 + 10 = 0$$

- In our context this asks if there exists an algorithm that can solve the problem
- In 1970 it was proven that the answer is no

History Computability Theory

- In the beginning of the twentieth century, mathematicians gave several definitions of computational models, such as Turing machines (1936) and the λ-calculus (1936), and they proved that all these are equivalent
- Later, after programming languages were invented, it was shown that these older notions of an algorithm are equivalent to notions of an algorithm that are based on C programs, Java programs, etc.

The Church-Turing Thesis

In other words, all attempts to give a rigorous definition of the notion of an algorithm led to the same concept. Because of this, computer scientists nowadays agree on what is called the:

Definition (Church-Turing Thesis)

Every computational process that is intuitively considered to be an algorithm can be converted to a Turing machine.

- Some researchers claim it to be a theorem, others a definition.