Fundamental Computer Science

Giorgio Lucarelli

giorgio.lucarelli@imag.fr

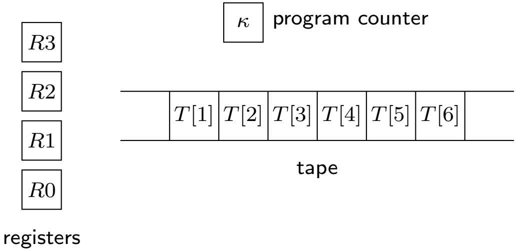
February 5, 2018

Random Access Turing Machines

- Random Access Memory
  - access any position of the tape in a single step

Random Access Turing Machines

- Random Access Memory
  - access any position of the tape in a single step
- we also need:
  - finite number of *registers* $\rightarrow$ manipulate addresses of the tape
  - *program counter* $\rightarrow$ current instruction to execute



- program: a set of instructions

Random Access Turing Machines: Instructions set

instruction	operand	semantics
read	j	$R_0 \leftarrow T[R_j]$
write	j	$T[R_j] \leftarrow R_0$
store	j	$R_j \leftarrow R_0$
load	j	$R_0 \leftarrow R_j$
load	$= c$	$R_0 = c$
add	j	$R_0 \leftarrow R_0 + R_j$
add	$= c$	$R_0 \leftarrow R_0 + c$
sub	j	$R_0 \leftarrow \max\{R_0 + R_j, 0\}$
sub	$= c$	$R_0 \leftarrow \max\{R_0 + c, 0\}$
half		$R_0 \leftarrow \lfloor \frac{R_0}{2} \rfloor$
jump	s	$\kappa \leftarrow s$
jpos	s	if $R_0 > 0$ then $\kappa \leftarrow s$
jzero	s	if $R_0 = 0$ then $\kappa \leftarrow s$
halt		$\kappa = 0$

- register R_0: *accumulator*

Random Access Turing Machines: Formal definition

A Random Access Turing Machine is a pair $M = (k, \Pi)$, where

- $k > 0$ is the finite number of registers, and
- $\Pi = (\pi_1, \pi_2, \ldots, \pi_p)$ is a finite sequence of instructions (program).

Random Access Turing Machines: Formal definition

A Random Access Turing Machine is a pair $M = (k, \Pi)$, where

- $k > 0$ is the finite number of registers, and
- $\Pi = (\pi_1, \pi_2, \ldots, \pi_p)$ is a finite sequence of instructions (program).

Notations

- the last instruction π_p is always a *halt* instruction
- $(\kappa; R_0, R_1, \ldots, R_{k-1}; T)$: a **configuration**, where
  - κ: program counter
  - R_j, $0 \leq j < k$: the current value of register j
  - T: the contents of the tape
    (each $T[j]$ contains a non-negative integer, i.e. $T[j] \in \mathbb{N}$)
- **halted configuration:** $\kappa = 0$

Examples

$$(1; 0, 5, 3; \emptyset)$$

1: load 1
2: add 2
3: sub =1
4: store 1
5: halt

Examples

1: load 1
2: add 2
3: sub =1
4: store 1
5: halt

$$\begin{aligned}(1;0,5,3;\emptyset) &\vdash (2;5,5,3;\emptyset) \vdash (3;8,5,3;\emptyset) \vdash (4;7,5,3;\emptyset)\\ &\vdash (5;7,7,3;\emptyset) \vdash (0;7,7,3;\emptyset)\end{aligned}$$

Examples

1: load 1
2: add 2
3: sub =1
4: store 1
5: halt

$$
\begin{aligned}
(1;0,5,3;\emptyset) \quad &\vdash \quad (2;5,5,3;\emptyset) \vdash (3;8,5,3;\emptyset) \vdash (4;7,5,3;\emptyset) \\
&\vdash \quad (5;7,7,3;\emptyset) \vdash (0;7,7,3;\emptyset)
\end{aligned}
$$

$$R_1 \leftarrow R_2 + R_1 - 1$$

Examples

1: load 1
2: add 2
3: sub =1
4: store 1
5: halt

$$(1;0,5,3;\emptyset) \vdash (2;5,5,3;\emptyset) \vdash (3;8,5,3;\emptyset) \vdash (4;7,5,3;\emptyset)$$
$$\vdash (5;7,7,3;\emptyset) \vdash (0;7,7,3;\emptyset)$$

$$R_1 \leftarrow R_2 + R_1 - 1$$

1: load 1
2: jzero 6
3: sub =3
4: store 1
5: jump 2
6: halt

$$(1;0,7;\emptyset)$$

Examples

1: load 1
2: add 2
3: sub =1
4: store 1
5: halt

$$
\begin{aligned}
(1;0,5,3;\emptyset) \quad &\vdash \quad (2;5,5,3;\emptyset) \vdash (3;8,5,3;\emptyset) \vdash (4;7,5,3;\emptyset) \\
&\vdash \quad (5;7,7,3;\emptyset) \vdash (0;7,7,3;\emptyset)
\end{aligned}
$$

$$R_1 \leftarrow R_2 + R_1 - 1$$

1: load 1
2: jzero 6
3: sub =3
4: store 1
5: jump 2
6: halt

$$
\begin{aligned}
(1;0,7;\emptyset) \quad &\vdash \quad (2;7,7;\emptyset) \vdash (3;7,7;\emptyset) \vdash (4;4,7;\emptyset) \vdash (5;4,4;\emptyset) \\
&\vdash \quad (2;4,4;\emptyset) \vdash (3;4,4;\emptyset) \vdash (4;1,4;\emptyset) \vdash (5;1,1;\emptyset) \\
&\vdash \quad (2;1,1;\emptyset) \vdash (3;1,1;\emptyset) \vdash (4;0,1;\emptyset) \vdash (5;0,0;\emptyset) \\
&\vdash \quad (2;0,0;\emptyset) \vdash (6;0,0;\emptyset) \vdash (0;0,0;\emptyset)
\end{aligned}
$$

Examples

1: load 1
2: add 2
3: sub =1
4: store 1
5: halt

$$(1;0,5,3;\emptyset) \vdash (2;5,5,3;\emptyset) \vdash (3;8,5,3;\emptyset) \vdash (4;7,5,3;\emptyset)$$
$$\vdash (5;7,7,3;\emptyset) \vdash (0;7,7,3;\emptyset)$$

$$R_1 \leftarrow R_2 + R_1 - 1$$

1: load 1
2: jzero 6
3: sub =3
4: store 1
5: jump 2
6: halt

$$(1;0,7;\emptyset) \vdash (2;7,7;\emptyset) \vdash (3;7,7;\emptyset) \vdash (4;4,7;\emptyset) \vdash (5;4,4;\emptyset)$$
$$\vdash (2;4,4;\emptyset) \vdash (3;4,4;\emptyset) \vdash (4;1,4;\emptyset) \vdash (5;1,1;\emptyset)$$
$$\vdash (2;1,1;\emptyset) \vdash (3;1,1;\emptyset) \vdash (4;0,1;\emptyset) \vdash (5;0,0;\emptyset)$$
$$\vdash (2;0,0;\emptyset) \vdash (6;0,0;\emptyset) \vdash (0;0,0;\emptyset)$$

while $R_1 > 0$ do $R_1 \leftarrow R_1 - 3$

Exercise

- Write a program for a Random Access Turing Machine that multiplies two integers.
  Tip: assume that the initial configuration is $(1; 0, a_1, a_2, 0; \emptyset)$

Exercise

- Write a program for a Random Access Turing Machine that multiplies two integers.
  Tip: assume that the initial configuration is $(1; 0, a_1, a_2, 0; \emptyset)$

while $R_1 > 0$ **do**
    $R_1 \leftarrow R_1 - 1$
    $R_3 \leftarrow R_3 + R_2$

Exercise

- Write a program for a Random Access Turing Machine that multiplies two integers.
  Tip: assume that the initial configuration is $(1; 0, a_1, a_2, 0; \emptyset)$

while $R_1 > 0$ **do**
    $R_1 \leftarrow R_1 - 1$
    $R_3 \leftarrow R_3 + R_2$

or (all computations should pass through R_0)

$R_0 \leftarrow R_1$
while $R_0 > 0$ **do**
    $R_0 \leftarrow R_0 - 1$
    $R_1 \leftarrow R_0$
    $R_0 \leftarrow R_3$
    $R_0 \leftarrow R_0 + R_2$
    $R_3 \leftarrow R_3$

Exercise

- Write a program for a Random Access Turing Machine that multiplies two integers.
  Tip: assume that the initial configuration is $(1; 0, a_1, a_2, 0; \emptyset)$

while $R_1 > 0$ **do**
    $R_1 \leftarrow R_1 - 1$
    $R_3 \leftarrow R_3 + R_2$

or (all computations should pass through R_0)

$R_0 \leftarrow R_1$
while $R_0 > 0$ **do**
    $R_0 \leftarrow R_0 - 1$
    $R_1 \leftarrow R_0$
    $R_0 \leftarrow R_3$
    $R_0 \leftarrow R_0 + R_2$
    $R_3 \leftarrow R_3$

```
1: load 1
2: jzero 9
3: sub =1
4: store 1
5: load 3
6: add 2
7: store 3
8: jump 1
9: halt
```

Another exercise

- Write a program for a Random Access Turing Machine that finds the maximum of a sequence of ℓ non-zero positive integers.
  Tip: initial configuration $(1; 0, \&a_1, 0; a_1 a_2 \ldots a_\ell 0)$

Another exercise

- Write a program for a Random Access Turing Machine that finds the maximum of a sequence of ℓ non-zero positive integers.
  Tip: initial configuration $(1; 0, \&a_1, 0; a_1 a_2 \dots a_\ell 0)$

```
 1: read 1
 2: jzero 11
 3: sub 2
 4: jzero 7
 5: read 1
 6: store 2
 7: load 1
 8: add =1
 9: store 1
10: jump 1
11: halt
```

Random Access Turing Machines

Theorem

Every Random Access Turing Machine $M = (\kappa, \Pi)$ *has an equivalent single tape Turing Machine* $M' = (K, \Sigma, \Gamma, \delta, s, H)$.

If M *halts on input of size* n *after* t *steps, then* M' *halts on after* $O(poly(t, n))$ *steps.*

Random Access Turing Machines

Theorem

Every Random Access Turing Machine $M = (\kappa, \Pi)$ has an equivalent single tape Turing Machine $M' = (K, \Sigma, \Gamma, \delta, s, H)$.

If M halts on input of size n after t steps, then M' halts on after $O(poly(t, n))$ steps.

Proof (sketch):

- we pass through the multiple tape model
  - use $k + 3$ tapes
  - tape 1: the contents of the tape of M
  - tape 2: the program counter
  - tape 3: auxiliary
  - tape $3 + j$, $1 \leq j \leq k$: corresponds to R_j
- add appropriate delimiters
- simulate instructions

Random Access Turing Machines

Proof (sketch):

- add 4
  1. copy the contents of tape 8 (R_4) on tape 3 (auxiliary)
  2. use the Turing Machine with two tapes seen in previous lecture to add the numbers in tapes 8 and 4 (R_0)
  3. store the result in tape 4
  4. increase the contents of tape 2 (program counter) by 1

Random Access Turing Machines

Proof (sketch):

- add 4
  1. copy the contents of tape 8 (R_4) on tape 3 (auxiliary)
  2. use the Turing Machine with two tapes seen in previous lecture to add the numbers in tapes 8 and 4 (R_0)
  3. store the result in tape 4
  4. increase the contents of tape 2 (program counter) by 1

- write 2
  1. move the head of tape 1 (tape of M) to the position (address) indicted by tape 6 (R_2)
  2. copy the contents of tape 4 (R_0) in the indicated position of tape 1
  3. increase the contents of tape 2 (program counter) by 1

Random Access Turing Machines

Proof (sketch):

- add 4
  1. copy the contents of tape 8 (R_4) on tape 3 (auxiliary)
  2. use the Turing Machine with two tapes seen in previous lecture to add the numbers in tapes 8 and 4 (R_0)
  3. store the result in tape 4
  4. increase the contents of tape 2 (program counter) by 1

- write 2
  1. move the head of tape 1 (tape of M) to the position (address) indicted by tape 6 (R_2)
  2. copy the contents of tape 4 (R_0) in the indicated position of tape 1
  3. increase the contents of tape 2 (program counter) by 1

- jpos 19
  1. scan tape 4 (R_0)
  2. if all cells are zero then increase the contents of tape 2 (program counter) by 1
  3. else replace the contents of tape 2 by 19

Random Access Turing Machines

Proof (sketch):

- the size of the contents of all tapes cannot be bigger that a polynomial to t and n
  - initially: n
  - at each step: the size of the contents is increased by at most a constant c (instruction add $= c$)

Random Access Turing Machines

Proof (sketch):

- the size of the contents of all tapes cannot be bigger that a polynomial to t and n
  - initially: n
  - at each step: the size of the contents is increased by at most a constant c (instruction add $= c$)
- each instruction can be implemented in time polynomial in the size of the contents of all tapes

Random Access Turing Machines

Proof (sketch):

- the size of the contents of all tapes cannot be bigger that a polynomial to t and n
  - initially: n
  - at each step: the size of the contents is increased by at most a constant c (instruction add $= c$)
- each instruction can be implemented in time polynomial in the size of the contents of all tapes

- Thus, complexity polynomial in t and n

Random Access Turing Machines

Proof (sketch):

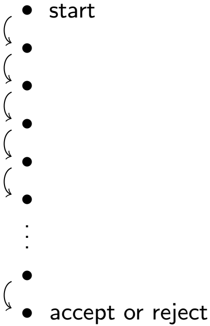
- the size of the contents of all tapes cannot be bigger that a polynomial to t and n
  - initially: n
  - at each step: the size of the contents is increased by at most a constant c (instruction add $= c$)
- each instruction can be implemented in time polynomial in the size of the contents of all tapes

- Thus, complexity polynomial in t and n

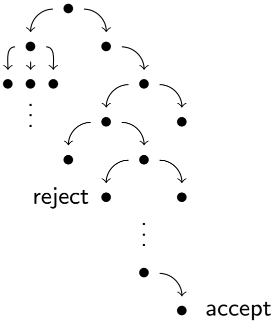
Random Access is not more powerful !!!

Non-determinism

- the next step is **not unique**



deterministic computation



non-deterministic computation

Non-deterministic Turing Machine

A Non-deterministic Turing Machine (M) is a sextuple $(K, \Sigma, \Gamma, \Delta, s, H)$, where K, Σ, Γ, s and H are as in the definition of the Deterministic Turing Machine, and Δ describes the transitions and it is a *subset* of

$$((K \setminus H) \times \Gamma) \quad \times \quad (K \times (\Gamma \cup \{\leftarrow, \rightarrow\}))$$

Non-deterministic Turing Machine

A Non-deterministic Turing Machine (M) is a sextuple $(K, \Sigma, \Gamma, \Delta, s, H)$, where K, Σ, Γ, s and H are as in the definition of the Deterministic Turing Machine, and Δ describes the transitions and it is a *subset* of

$$((K \setminus H) \times \Gamma) \quad \times \quad (K \times (\Gamma \cup \{\leftarrow, \rightarrow\}))$$

- Δ is not a function
  - a single pair of (q, σ) can lead to multiple pairs (q', σ')
  - the empty string ϵ is allowed as a transition symbol

Non-deterministic Turing Machine

A Non-deterministic Turing Machine (M) is a sextuple $(K, \Sigma, \Gamma, \Delta, s, H)$, where K, Σ, Γ, s and H are as in the definition of the Deterministic Turing Machine, and Δ describes the transitions and it is a *subset* of

$$((K \setminus H) \times \Gamma) \quad \times \quad (K \times (\Gamma \cup \{\leftarrow, \rightarrow\}))$$

- Δ is not a function
  - a single pair of (q, σ) can lead to multiple pairs (q', σ')
  - the empty string ϵ is allowed as a transition symbol
- A configuration may *yield* several configurations in a single step
  - $\vdash_M$ is not necessarily uniquely identified

Non-deterministic Turing Machine

Definitions

Let $M = (K, \Sigma, \Gamma, \Delta, s, H)$ be a Non-deterministic Turing Machine.
We say that M **accepts** an input $w \in \Sigma^*$ if

$$(s, \underline{\sqcup} w) \vdash_M^* (h, u\underline{\sigma}v)$$

for some $h \in H$, $\sigma \in \Sigma$ and $u, v \in \Sigma^*$.

Non-deterministic Turing Machine

Definitions

Let $M = (K, \Sigma, \Gamma, \Delta, s, H)$ be a Non-deterministic Turing Machine.
We say that M **accepts** an input $w \in \Sigma^*$ if

$$(s, \underline{\sqcup}w) \vdash_M^* (h, u\underline{\sigma}v)$$

for some $h \in H$, $\sigma \in \Sigma$ and $u, v \in \Sigma^*$.

We say that M **recognizes** (or **semidecides**) a language L if for each $w \in \Sigma^*$ the following holds: $w \in L$ if and only if M accepts w.

Non-deterministic Turing Machine

Definitions

Let $M = (K, \Sigma, \Gamma, \Delta, s, H)$ be a Non-deterministic Turing Machine.
We say that M **accepts** an input $w \in \Sigma^*$ if

$$(s, \underline{\sqcup} w) \vdash_M^* (h, u\underline{\sigma}v)$$

for some $h \in H$, $\sigma \in \Sigma$ and $u, v \in \Sigma^*$.

We say that M **recognizes** (or **semidecides**) a language L if for each $w \in \Sigma^*$ the following holds: $w \in L$ if and only if M accepts w.

We say that M **decides** a language L if for each $w \in \Sigma^*$ the following two conditions hold:

1. there is natural number $N \in \mathbb{N}$ (depending on M and $|w|$) such that there is no configuration c satisfying $(s, \underline{\sqcup} w) \vdash_M^N c$
2. $w \in L$ if and only if $(s, \underline{\sqcup} w) \vdash_M^* (y, u\underline{\sigma}v)$ for some $\sigma \in \Sigma$ and $u, v \in \Sigma^*$

Non-deterministic Turing Machine

Definitions (cont'd)

Let $M = (K, \Sigma, \Gamma, \Delta, s, H)$ be a Non-deterministic Turing Machine.

We say that M **computes** a function $f : \Sigma^* \to \Sigma^*$ if for each $w \in \Sigma^*$ the following two conditions hold:

1. there is natural number $N \in \mathbb{N}$ (depending on M and $|w|$) such that there is no configuration c satisfying $(s, \sqcup w) \vdash_M^N c$
2. $(s, \sqcup w) \vdash_M^* (h, \sqcup v)$ if and only if $v = f(w)$

Example

- A natural number $m \in \mathbb{N}$ is called *composite* if it can be written as the product of two natural numbers $p, q \in \mathbb{N}$, i.e., $m = p \cdot q$. Describe (high-level) a Non-deterministic Turing Machine that recognizes the language $L = \{1^m : m \text{ is a composite number}\}$.

Example

- A natural number $m \in \mathbb{N}$ is called *composite* if it can be written as the product of two natural numbers $p, q \in \mathbb{N}$, i.e., $m = p \cdot q$. Describe (high-level) a Non-deterministic Turing Machine that recognizes the language $L = \{1^m : m$ is a composite number$\}$.

  1. choose two integers p and q **non-deterministically**
  2. multiply p and q
  3. compare a with $p \cdot q$ and if they are equal then *accept*

Example

- A natural number $m \in \mathbb{N}$ is called *composite* if it can be written as the product of two natural numbers $p, q \in \mathbb{N}$, i.e., $m = p \cdot q$. Describe (high-level) a Non-deterministic Turing Machine that recognizes the language $L = \{1^m : m \text{ is a composite number}\}$.
  1. choose two integers p and q **non-deterministically**
  2. multiply p and q
  3. compare a with $p \cdot q$ and if they are equal then *accept*

- What does **non-deterministically** mean?

Example

- A natural number $m \in \mathbb{N}$ is called *composite* if it can be written as the product of two natural numbers $p, q \in \mathbb{N}$, i.e., $m = p \cdot q$. Describe (high-level) a Non-deterministic Turing Machine that recognizes the language $L = \{1^m : m \text{ is a composite number}\}$.
  1. choose two integers p and q **non-deterministically**
  2. multiply p and q
  3. compare a with $p \cdot q$ and if they are equal then *accept*

- What does **non-deterministically** mean?
  - choose $(p, q) \in \{(1, 1), (1, 11), (1, 111), \ldots, (11, 1), (11, 11), \ldots\}$

Example

- A natural number $m \in \mathbb{N}$ is called *composite* if it can be written as the product of two natural numbers $p, q \in \mathbb{N}$, i.e., $m = p \cdot q$. Describe (high-level) a Non-deterministic Turing Machine that recognizes the language $L = \{1^m : m \text{ is a composite number}\}$.
  1. choose two integers p and q **non-deterministically**
  2. multiply p and q
  3. compare a with $p \cdot q$ and if they are equal then *accept*
- What does **non-deterministically** mean?
  - choose $(p, q) \in \{(1, 1), (1, 11), (1, 111), \ldots, (11, 1), (11, 11), \ldots\}$
- How to transform the above machine to decide the same language?

Example

- A natural number $m \in \mathbb{N}$ is called *composite* if it can be written as the product of two natural numbers $p, q \in \mathbb{N}$, i.e., $m = p \cdot q$. Describe (high-level) a Non-deterministic Turing Machine that recognizes the language $L = \{1^m : m \text{ is a composite number}\}$.

  1. choose two integers p and q **non-deterministically**
  2. multiply p and q
  3. compare a with $p \cdot q$ and if they are equal then *accept*

- What does **non-deterministically** mean?

  - choose $(p, q) \in \{(1, 1), (1, 11), (1, 111), \ldots, (11, 1), (11, 11), \ldots\}$

- How to transform the above machine to decide the same language?

  1. choose two integers $p < m$ and $q < m$ **non-deterministically**
  2. multiply p and q
  3. compare a with $p \cdot q$ and if they are equal then *accept*, else *reject*

Exercise

- Consider a set $A = \{a_1, a_2, \ldots, a_n\}$ of positive integers and an integer $w \in \mathbb{N}$.
  Give a Non-deterministic Turing Machine that *recognizes* the language $L = \{A' \subseteq A : \sum_{a_i \in A'} a_i = w\}$.

Exercise

- Consider a set $A = \{a_1, a_2, \ldots, a_n\}$ of positive integers and an integer $w \in \mathbb{N}$.
  Give a Non-deterministic Turing Machine that *recognizes* the language $L = \{A' \subseteq A : \sum_{a_i \in A'} a_i = w\}$.

1. choose non-deterministically a set $A' \subseteq A$
2. add the elements of A'
3. if they sum up to w, then *accept*

Exercise

- Consider a set $A = \{a_1, a_2, \dots, a_n\}$ of positive integers and an integer $w \in \mathbb{N}$.
  Give a Non-deterministic Turing Machine that *recognizes* the language $L = \{A' \subseteq A : \sum_{a_i \in A'} a_i = w\}$.

1. choose non-deterministically a set $A' \subseteq A$
2. add the elements of A'
3. if they sum up to w, then *accept*

- How to choose A' non-deterministically?
  - produce all binary numbers of n digits
  - start from $00\dots0$ and add 1 at each iteration

Non-deterministic Turing Machine

Theorem

Every Non-deterministic Turing Machine $NDTM = (K, \Sigma, \Gamma, \Delta, s, H)$ *has an equivalent Deterministic Turing Machine* DTM.

Proof (sketch):

Non-deterministic Turing Machine

Theorem

Every Non-deterministic Turing Machine $NDTM = (K, \Sigma, \Gamma, \Delta, s, H)$ *has an equivalent Deterministic Turing Machine* DTM.

Proof (sketch):

- Use a multiple tape deterministic Turing Machine

tape 1: input (never changes)
tape 2: simulation
tape 3: address

Non-deterministic Turing Machine

Theorem

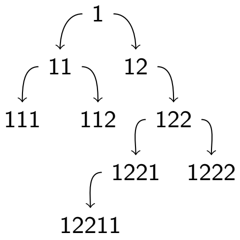
Every Non-deterministic Turing Machine $NDTM = (K, \Sigma, \Gamma, \Delta, s, H)$ has an equivalent Deterministic Turing Machine DTM.

Proof (sketch):

- Use a multiple tape deterministic Turing Machine

tape 1: input (never changes)
tape 2: simulation
tape 3: address

- data on tape 3:
  - each node of the computation tree of $NDTM$ has at most c children: $c \leq |K| \cdot (|\Sigma| + 2)$
  - address of a node in $\{1, 2, \ldots, c\}^*$



Non-deterministic Turing Machine

Proof (sketch):

1. Initialize tape 1 with the input w and tapes 2 & 3 to be empty.
2. Copy the contents of tape 1 to tape 2.
3. Simulate NDTM on tape 2 using the sequence of computations described in tape 3. If an accepting configuration is yielded, then *accept*.
4. Update the string in tape 3 with the lexicographic next string and go to 2.

Non-deterministic Turing Machine

Proof (sketch):

1. Initialize tape 1 with the input w and tapes 2 & 3 to be empty.
2. Copy the contents of tape 1 to tape 2.
3. Simulate NDTM on tape 2 using the sequence of computations described in tape 3. If an accepting configuration is yielded, then *accept*.
4. Update the string in tape 3 with the lexicographic next string and go to 2.

- Observations:
  - we perform a Breadth First Search of the computation tree

Non-deterministic Turing Machine

Proof (sketch):

1. Initialize tape 1 with the input w and tapes 2 & 3 to be empty.
2. Copy the contents of tape 1 to tape 2.
3. Simulate NDTM on tape 2 using the sequence of computations described in tape 3. If an accepting configuration is yielded, then *accept*.
4. Update the string in tape 3 with the lexicographic next string and go to 2.

- Observations:
  - we perform a Breadth First Search of the computation tree
  - we need exponential time of steps with respect to NDTM!

Non-deterministic Turing Machine

Discussion

- Non-deterministic Turing Machines seem to be more powerful than deterministic ones
- we pay this in computation time

Non-deterministic Turing Machine

Discussion

- Non-deterministic Turing Machines seem to be more powerful than deterministic ones
- we pay this in computation time

- next lectures: we will see what does this mean

More exercises

- Give a Random Access Turing Machine that *decides* the language $L = \{a^n b^n c^n : n \geq 0\}$.
- Give a Random Access Turing Machine that *decides* the language $L = \{wcw : w \in \{a, b\}^*\}$.
- Give a Non-deterministic Turing Machine that *recognizes* the language $L = \{a^* abb^* aa^*\}$ (use simple machines).
- Give a Non-deterministic Turing Machine that *recognizes* the language $L = \{ww^R uu^R : w, u \in \{a, b\}^*\}$ (give high-level definition).
- Consider a graph $G = (V, E)$ and an positive integer k. Give a Non-deterministic Turing Machine that *recognizes* the language $L = \{V' \subseteq V : |V'| \geq k$ and $(u, v) \notin E$ for any two $u, v \in V'\}$ (give high-level definition).