Fundamental Computer Science

Denis Trystram (inspired by Giorgio Lucarelli)

February, 2020

Non-deterministic Turing Machine

A Non-deterministic Turing Machine (M) is a sixtuple $(K, \Sigma, \Gamma, \Delta, s, H)$, 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 sixtuple $(K, \Sigma, \Gamma, \Delta, s, H)$, 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

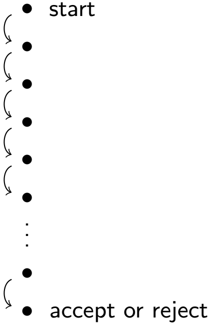
A Non-deterministic Turing Machine (M) is a sixtuple $(K, \Sigma, \Gamma, \Delta, s, H)$, where K, Σ, Γ, s and H are as in the definition of the Deterministic Turing Machine, and Δ describes the transitions and it is a *subset* of

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

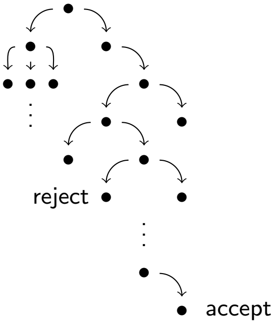
- Δ is not a function
  - a single pair of (q, σ) can lead to multiple pairs (q', σ')
  - the empty string ϵ is allowed as a transition symbol
- A configuration may *yield* several configurations in a single step
  - $\vdash_M$ is not necessarily uniquely identified

Non-determinism

- the next step is **not unique**



deterministic computation



Comparison deterministic vs non-deterministic

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

- $(s, \underline{\sqcup}w) \vdash^*_M (h, \underline{\sqcup}v)$ 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 $A = \{a_1, a_2, \ldots, a_n\}$ 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*

- How to choose A' non-deterministically?
  - produce all binary numbers of n digits
  - start from $00\ldots0$ 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

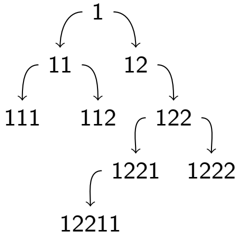
Every Non-deterministic Turing Machine $NDTM = (K, \Sigma, \Gamma, \Delta, s, H)$ has an equivalent Deterministic Turing Machine DTM.

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
  - 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