Fundamental Computer Science
Non deterministic TM

Denis Trystram

February, 2021
A Non-deterministic Turing Machine ($M$) is a sixtuple $(K, \Sigma, \Gamma, \Delta, s, H)$, where $K$, $\Sigma$, $\Gamma$, $s$ and $H$ are similar to the definition of the Deterministic Turing Machine. 

$\Delta$ describes the transitions, it is a **subset** of 

$$( (K \setminus H) \times \Gamma ) \times (K \times (\Gamma \cup \{\leftarrow, \rightarrow\}))$$
Non-deterministic Turing Machine

Definition.
A Non-deterministic Turing Machine ($M$) is a sixtule $(K, \Sigma, \Gamma, \Delta, s, H)$, where $K$, $\Sigma$, $\Gamma$, $s$ and $H$ are similar to the definition of the Deterministic Turing Machine
$\Delta$ describes the transitions, it is a subset of
\[
((K \setminus H) \times \Gamma) \times (K \times (\Gamma \cup \{\leftarrow, \rightarrow\}))
\]

- $\Delta$ is not a function
  - a single pair of $(q, \sigma)$ can lead to multiple pairs $(q', \sigma')$
  - the empty string $\epsilon$ is allowed as a transition symbol
**Definition.**

A Non-deterministic Turing Machine \( M \) is a sixtuple \( (K, \Sigma, \Gamma, \Delta, s, H) \), where \( K, \Sigma, \Gamma, s \) and \( H \) are similar to the definition of the Deterministic Turing Machine.

\( \Delta \) describes the transitions, it is a subset of

\[
((K \setminus H) \times \Gamma) \times (K \times (\Gamma \cup \{\leftarrow, \rightarrow\}))
\]

- \( \Delta \) is not a function
  - a single pair of \( (q, \sigma) \) can lead to multiple pairs \( (q', \sigma') \)
  - the empty string \( \epsilon \) 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

- start

- accept or reject
**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

$$(s, \sqcup w) \vdash^*_M (h, u\sigma v)$$

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 

$$(s, \sqcup w) \triangleright^* M (h, u\sigma v)$$

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. $w \in L$ if and only if $(s, \sqcup w) \triangleright^* M (h, u\sigma v)$ for some $\sigma \in \Sigma$ and $u, v \in \Sigma^*$

2. there is natural number $N \in \mathbb{N}$ (depending on $M$ and $|w|$) such that there is no configuration $C$ satisfying $(s, \sqcup w) \triangleleft^N M C$
Let \( M = (K, \Sigma, \Gamma, \Delta, s, H) \) be a Non-deterministic Turing Machine.

We say that \( M \) \text{ computes} a function \( f : \Sigma^* \to \Sigma^* \) if for each \( w \in \Sigma^* \) the following condition holds:

\[
\vdash (s, \sqcup w) \Downarrow^* M (h, \sqcup v) \text{ if and only if } v = f(w)
\]
A natural number \( m \in \mathbb{N} \) is called \textit{composite} if it can be written as the product of two natural numbers \( p_1, p_2 \in \mathbb{N} \), i.e., \( m = p_1 \cdot p_2 \).

Describe (high-level) a Non-deterministic Turing Machine that recognizes the language \( L = \{1^m : m \text{ is a composite number}\} \).
Example (1)

- A natural number $m \in \mathbb{N}$ is called *composite* if it can be written as the product of two natural numbers $p_1, p_2 \in \mathbb{N}$, i.e., $m = p_1 \cdot p_2$

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 $p_1$ and $p_2$ **non-deterministically**
2. multiply $p_1$ and $p_2$
3. compare $m$ with $p_1 \cdot p_2$ and if they are equal then *accept*
Example (2)

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

- choose \((p_1, p_2) \in \{(1, 1), (1, 11), (1, 111), \ldots, (11, 1), (11, 11), \ldots\}\)
Example (2)

- What does non-deterministically mean?
  - choose \((p_1, p_2) \in \{(1, 1), (1, 11), (1, 111), \ldots, (11, 1), (11, 11), \ldots\}\)

- How to transform the above machine to decide the same language?
Example (2)

- What does **non-deterministically** mean?
  - choose \((p_1, p_2) \in \{(1, 1), (1, 11), (1, 111), \ldots, (11, 1), (11, 11), \ldots\}\)

- How to transform the above machine to decide the same language?
  1. choose two integers \(p_1 < m\) and \(p_2 < m\) **non-deterministically**
  2. multiply \(p_1\) and \(p_2\)
  3. compare \(m\) with \(p_1 \cdot p_2\) and if they are equal then accept, else reject
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):
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 \( \{1, 2, \ldots, c\}^* \)
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!
Discussion

- Any non-deterministic TM can be simulated by a deterministic one.
- However, Non-deterministic TM seem to be more powerful than deterministic ones.
- We pay this in computation time.
Any non-deterministic TM can be simulated by a deterministic one.

However, Non-deterministic TM seem to be more powerful than deterministic ones.

We pay this in computation time.

We will see what does it mean in the next lectures.