**3P17/4P17 COMPUTABILITY 2014-2015 Lecture Handout 4**

## Non-deterministic Turing machines continued...

For configurations $C$ and $C'$, and a (state, symbol, direction) triplet $a = (q, x, d)$ that takes $C$ to $C'$, we shall write $C \xrightarrow{a} C'$ or $C \xrightarrow{(q,x,d)} C'$.

We shall refer to them as **action** triplets, so as not to confuse them with configuration triplets. So an action triplet transforms one configuration into another.

In a deterministic TM, the transition function $\delta$ specified only one action for a given (state, symbol) pair. In a non-deterministic TM, it specifies a set of actions.

Let $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}})$ be a non-deterministic TM.

The number of possible action triplets is $b = |Q| \times |\Gamma| \times |\{L, R\}|$, which is finite and so, we can assign each action triplet a unique i.d. number from the set $1, 2, \ldots, b$.

In a **computation tree**, nodes are configurations. A computation tree for $N$ on input $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$ is defined as follows: The root node is the starting configuration $C_0 = (q_0, T_0, 1)$. A configuration $C'$ is a child of configuration $C$ in the tree if $C \xrightarrow{(q,x,d)} C'$ and $(q, x, d) \in \delta(q, T(i))$. If a configuration is halting, it has no children.

A path of from the root to another node in the tree corresponds to a finite sequence $e = (e_1, e_2, \ldots, e_r)$ of action triple i.d.'s., each $e_i, \in \{1, 2, \ldots, b\}$

**Theorem 1.** *Every non-deterministic Turing machine has an equivalent single tape, deterministic Turing machine.*

*Proof.* Let $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}})$ be a non-deterministic TM. We will simulate $N$ with a deterministic multitape TM $D$ by having $D$ do a breath-first search of the computation tree of $N$. By Theorem 1 in Lecture Handout 2, there is a single tape deterministic TM that is equivalent to $D$, and therefore to $N$.

The deterministic TM $D$ has three tapes. The input alphabet $\Sigma_D$ of $D$ is the same as the input alphabet $\Sigma$ of $N$. The tape alphabet $\Gamma_D$ of $D$ is the union $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ of the individual tape alphabets described below.

Tape 1 contains the input string and this tape is never changed. Therefore its alphabet is simply $\Gamma_1 = \Sigma \cup \{\triangleright, \sqcup\}$.

Tape 2 maintains a copy of $N$'s tape on a branch of its non-deterministic computation, it has alphabet $\Gamma_2 = \Gamma$.

Tape 3 acts like a queue - it keeps track of where $D$ needs to try in $N$'s computation tree. It has alphabet $\Gamma_3 = \{\mathtt{1}, \ldots, \mathtt{b}, *\}$, where $*$ is treated as a delimiter symbol. Tape 3 consists of sequences of action triplet i.d.'s separated by $*$. E.g., if $b = 8$ then the tape might look like

$\triangleright \sqcup \ldots \sqcup \mathtt{231} * \mathtt{47} * \mathtt{782} \sqcup \ldots$ which represents the queue $(2,3,1) * (4,7) * (7,8,2)$. A delimiter separates *elements* of the queue, e.g., (2,3,1) is an element, and the *head* of the queue is the queue is the first element.

Processing an element of the queue: An element has form $e = (e_1, e_2, \ldots, e_r)$ where $r \geq 1$ and each $1 \leq e_i \leq b$ is an action triplet i.d. Assume tape 2 has the input $\sigma$ with the head above the first symbol and and is otherwise blank. To *process* a queue element $e = (e_1, e_2, \ldots, e_r)$, $D$ looks up the action $a_i = (q, x, d)$ of the identifier $e_i$, changes the symbol under head 2 to $x$, and moves it in the direction $d$. It does this for each $e_i$ in turn until $e_r$.

Initially tape 1 contains the input $\sigma$ and and tapes 2 and 3 are blank.

1. Copy tape 1 to tape 2.

2. Use tape 2 to simulate the first step of $N$ on $\sigma$, i.e., determine $\delta(q_0, \sigma_1)$. This will give a set action triplets. Add the i.d.'s of those triplets to the queue in some arbitrary order, separated by $*$.

3. Wipe clean tape 2, copy tape 1 to tape 2, and put the head of tape 2 above location 1 (the first square after $\triangleright$).

4. Let $e = (e_1, e_2, \ldots, e_r)$ be the head of the queue. Process $e$.

5.Let $a_r = (q, x, d)$ be the action of $e_r$ in the previous step.

If $q \neq q_{\text{acc}}, q_{\text{rej}}$, then look up $\delta(q, y)$ where $y$ is the current symbol under head 2. This retrieves a set $\{t_1, t_2, \ldots t_s\}$ of new action i.d.'s. Add each of $(e_1, e_2, \ldots, e_r, t_1), (e_1, e_2, \ldots, e_r, t_2), \ldots, (e_1, e_2, \ldots, e_r, t_s)$ to the back of the queue, separated by $*$.

If $q = q_{\text{acc}}$, then halt with accept.

If $q = q_{\text{rej}}$, the continue to the next step.

6. Remove $e$ and its delimiter from the head of the queue. If the queue is now empty, then halt with reject. Otherwise, go to step 3.

$\square$

## Church-Turing Thesis

In summary:

Given an alphabet $\Sigma$, a language $L \subseteq \Sigma^*$ is *recognisable*, or *recursively enumerable* if some Turing machine recognises it. Additionally, $L$ is *decidable*, or *recursive* if some Turing machine recognises it and halts on any input $\sigma \in \Sigma^*$.

A function on $\Sigma$-strings $f : \Sigma^* \to \Sigma^*$ is *computable* or *recursive* if some Turing machine computes it (by which definition, the TM always halts).

For any multitape Turing machine $M$, there is a single tape Turing machine $M'$ that is equivalent, ie., recognises a language $L$ if $M$ recognises it, and decides $L$ if $M$ decides it.

For any multitape Turing machine $M$ that computes a function $f : \Sigma^* \to \Sigma^*$, there is a single tape Turing machine $M'$ that also computes $f$.

For any non-deterministic Turing machine $N$, there is a deterministic Turing machine $M'$ that is equivalent, ie., recognises a language $L$ if $M$ recognises it, and decides $L$ if $M$ decides it.

For any non-deterministic Turing machine $N$ that computes a function $f : \Sigma^* \to \Sigma^*$, there is a deterministic Turing machine $M'$ that also computes $f$.

**Church-Turing Thesis**: *Given any reasonable model of computation, any function that is computable under that model (i.e., terminates in a finite number of steps with the correct answer), is also computable by a single tape, deterministic Turing machine.*

Intuitively, an "algorithm" is a set of instructions for taking some input that, in a finite number of steps, gives some output, i.e. computing a function in a finite number of steps. Therefore, the Church-Turing thesis says that an algorithm is something that can be expressed as a (single tape, deterministic) Turing machine.