LECTURE 7 – SPECIAL TURING MACHINES

Overview

- Linear Speedup Turing Computation
- Random Access Turing Machine
- Non-deterministic Turing Machine

Linear Speedup Turing Computation

Theorem. Let the language L $\in$ TIME(f(n)). Then, for any C>0, L $\in$ TIME(f'(n)), where f'(n)=C·f(n)+n+2.

The above theorem on the complexity of multi-tape Turing machines states that multiplicative constants are not important given the adoption of a multi-tape Turing machine for defining time complexity.

The proof of the above theorem is by demonstrating that:

For a Turing machine TM = $(Q,\Sigma,\Gamma,\delta,q_0,F)$ with multiple-tapes which decides L and operates in time f(n):

It is possible to construct a k-tape Turing machine TM' = $(Q,\Sigma,\Gamma,\delta,q_0,F)$ which operates in time bound f'(n) and simulates TM:

> Each symbol of TM' encodes several symbols of TM and therefore TM' can simulate several moves of TM by one move of its own (this corresponds to increasing the word length of a computer).

The above means that the constant in a linear f(n) (e.g. 5n) or superlinear f(n) (e.g. $17n^3$) can be made arbitrarily small by increasing the number of tapes in the Turing machine.

Then, by having a multiple-tape computation model, the time bound f(n) can have a linear speedup.

The lower order terms in f(n) can be completely discarded.

Linear speedup corresponds to advances in hardware. Hence, algorithm design is only positively affected from improvements in the rate of growth of the time complexity f(n).

Then, if L is a language which is decidable in polynomial time, its complexity is in TIME($f(n^k)$) for some integer k>0.

The union of all these classes, i.e. the set of all languages that are decidable in polynomial time by deterministic Turing machines is denoted by **P**.

Random Access Turing Machines

A random access Turing machine (RAM) is a computational model that consists of a program action on a data structure which is an array of registers.

Each register can contain an arbitrary large integer which can be negative.
Formally, a RAM program $\Pi=(\pi_1, \pi_2, \pi_3,\ldots, \pi_m)$ is a finite sequence of instructions, where each instruction π_i has a form of the following types:

Instruction	Operand	Semantics
READ	j	$r_0=i_j$
READ	↑j	$r_0=i_{rj}$
STORE	j	$r_j=r_0$
STORE	↑j	$r_{rj}=r_0$
LOAD	x	$r_0=x$
ADD	x	$r_0= r_0 + x$
SUB	x	$r_0= \max\{r_0 - x , 0\}$
HALF		$r_0= \lfloor r_0/2 \rfloor$
JUMP	j	k=j
JPOS	j	if $r_0>0$ then k=j
JZERO	j	if $r_0=0$ then k=j
JNEG	j	if $r_0<0$ then k=j
HALT		k=0

The instructions in a RAM machine simulate those in an actual computer.

Register r_0 is the accumulator, where all the arithmetic and logic operations are performed.

There are three types of addressing the operand in a RAM machine:

j means address the contents of register j

↑j means address the contents of register whose index is in register j

=x means the integer itself

In a RAM machine, k is the program counter, it starts in k=1 and each instruction changes k to k+1 unless otherwise stated.

The HALT instruction and any other instruction that is semantically wrong, causes the RAM computation to stop.

The input of a RAM program is a finite sequence of integers contained in input registers $I=(i_1, i_2, \ldots, i_n)$.

Upon halting, the output of a RAM program is in the accumulator (r_0 register).
A configuration of a RAM program with n registers and m instructions is given by a n+2-tuple $(k;r_0, r_1,\ldots, r_{n-1};T)$ where:

$k \in \aleph$ is the program counter, an integer between 0 and m

each r_j is the value of register j

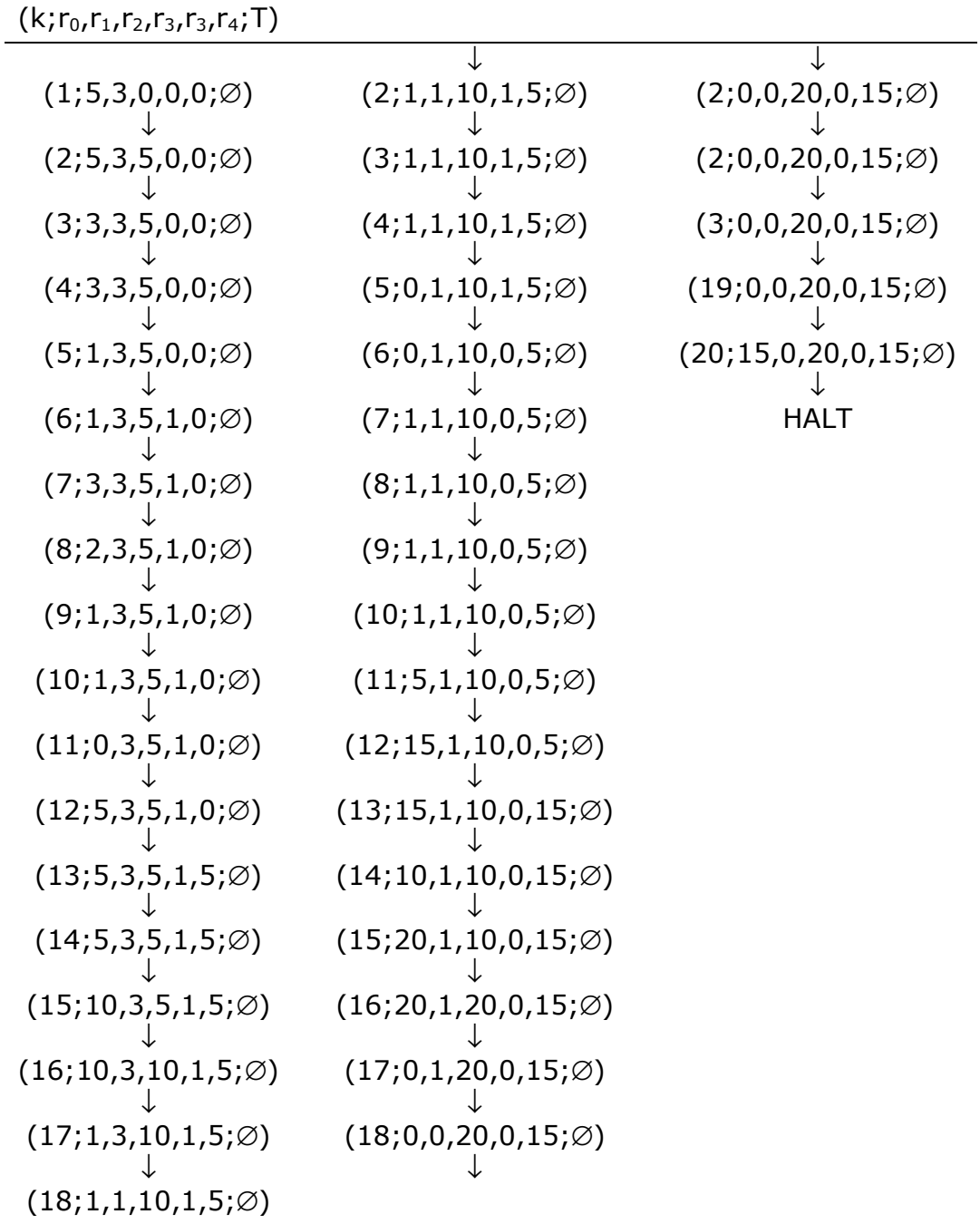
T is the tape contents

Example. The following RAM program computes the function $\phi(\aleph x \aleph)\rightarrow\aleph$ where the function $\phi(x,y)=x\cdot y$. To implement the computation, this RAM program uses ordinary binary multiplication. In the initial configuration, register r_0 contains the number x and register r_1 contains the number y.

1. STORE 2	11. ADD 2
2. LOAD 1	12. STORE 4
3. JZERO 19	13. LOAD 2
4. HALF	14. ADD 2
5. STORE 3	15. STORE 2
6. LOAD 1	16. LOAD 3
7. SUB 3	17. STORE 1
8. SUB 3	18. JUMP 2
9. JZERO 13	19. LOAD 4
10. LOAD 4	20. HALT

The above RAM program computes the product of two numbers in O(n) time. In this program, O(n) means the total number of instructions that is proportional to the logarithm of the input integers.

For example, the sequence of configurations for a computation $\phi(5,3)=15$ with the above RAM program using 5 registers is as follows:



Theorem. For a language L in TIME(f(n)) there is Random Access Turing Machine which decides L in time O(f(n)).

The above theorem means than any ordinary deterministic Turing machine can be simulated by a RAM program. See proof in (Papadimitriou, p. 43).

Non-deterministic Turing Machine

A non-deterministic Turing machine is given by NTM = $(Q,\Sigma,\Gamma,\delta,q_0,F)$ where the program δ is not a function anymore, instead δ is a relation:

$$\delta \subset [(Q \times \Gamma) \times (Q \times \Gamma \times \{ L , R \})]$$

which means that for each state-symbol combination there may be more than one possible transition or no transition at all.

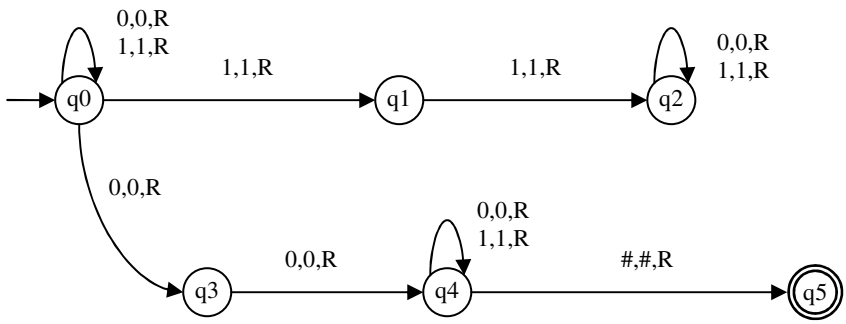
In a non-deterministic Turing machine the input string is accepted if there is some sequence of non-deterministic choices that result in halt and succeed. Other choices may result in rejection, just one accepting computation is enough.

The input string is rejected only is there is no sequence of choices that can lead to acceptance (halt and succeed).

Example. The following non-deterministic Turing machine recognises strings of the form {0,1}*00{0,1}* but rejects strings of the form {0,1}*11{0,1}*.

NTM = $(\{q_0,q_1,q_2,q_3,q_4,q_5\},\{0,1\},\{0,1,\#\},\delta,q_0,\{q_5\})$ where δ is given by:

Q \ Σ	0	1	#
q_0	$\{q_0,0,R\}\{q_3,0,R\}$	$\{q_0,1,R\}\{q_1,1,R\}$	∅
q_1	∅	$\{q_2,1,R\}$	∅
q_2	$\{q_2,0,R\}$	$\{q_2,1,R\}$	∅
q_3	$\{q_4,0,R\}$	∅	∅
q_4	$\{q_4,0,R\}$	$\{q_4,1,R\}$	$\{q_5,\#,R\}$
q_5	∅	∅	∅



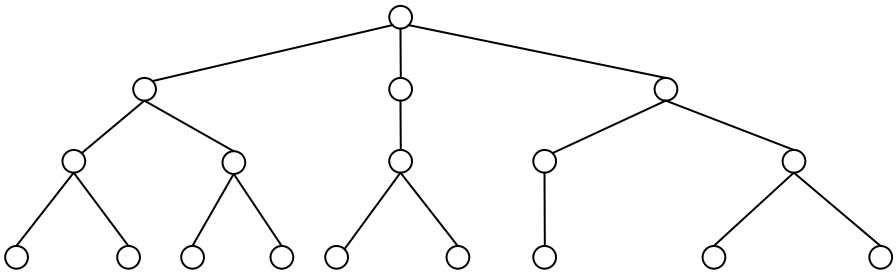
Processing the string "10010" is as follows:

$q_0 10010\# \rightarrow 1\{q_0,q_1\}0010\# \rightarrow 10\{q_0,q_3\}010\# \rightarrow 100\{q_0,q_3,q_4\}10\# \rightarrow$
$1001\{q_0,q_1,q_4\}0\# \rightarrow 10010\{q_0,q_3,q_4\}\# \rightarrow 10010\#\{q_5\} \rightarrow$ halts and succeeds

Processing the string "10110" is as follows:

$q_0 10110\# \rightarrow 1\{q_0,q_1\}0110\# \rightarrow 10\{q_0,q_3\}110\# \rightarrow 101\{q_0,q_1\}10\# \rightarrow$
$1011\{q_0,q_1,q_2\}0\# \rightarrow 10110\{q_0,q_3,q_2\}\# \rightarrow$ halts and fails

The computation of a non-deterministic machine can be seen as the depth of the total computational activity that is being carried out.



The set of languages decided by a non-deterministic Turing machine within time f(n) is a new complexity class denoted NTIME(f(n)).

Then, the union of all NTIME($f(n^k)$) forms the non-deterministic class **NP**.

We can say that **P** $\subseteq$ **NP** because deterministic Turing machines are a special case of non-deterministic ones.

By converting a non-deterministic computation into a practical, polynomial time algorithm for a given problem we could solve problems like TSP in polynomial time.

Theorem. Suppose that language L is decided by a non-deterministic Turing machine N in time f(n). Then, L is decided by a 3-tape deterministic Turing machine M in time $O(c^{f(n)})$ where c>1 is some constant depending of the non-deterministic machine N.

The non-deterministic Turing machine is not a true model of computation.

Non-deterministic computation (unlimited parallelism) is closely related to some applications of computation such as logic, combinatorial optimisation and artificial intelligence.

If each branch in a non-deterministic computation can be verified in polynomial time (for example the decision version of the TSP, is there a tour of length at most B?), then it is said that the non-deterministic program solves the decision problem.

That is, the non-deterministic algorithm is said to operate in polynomial time if there is a polynomial p such that for every instance I there is some guess that leads the deterministic checking stage to respond YES in polynomial time of p(n) where n is the length of the instance.

The **NP** class is the class of decision problems that can be solved by polynomial time by non-deterministic algorithms.

However, there are problems that are not verifiable in polynomial time (for example the complement of TSP, is it true that no tour has length B or less?) and this happens for many problems in the class **NP**.

The **NP** class is associated to polynomial time verifiability while the class **P** is associated to polynomial time solvability.

Reading: (Papadimitriou, chapter 2), (Lewis&Papadimitriou, chapter 4)