# Alternative models of computation

Complexity Theory

Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic

Lukáš Charvát

*This material was created with the support of the Czech Ministry of Education, Youth and Sports (project FRVŠ 166/2013/G1).*

# Motivation

- Turing machines (TMs):
  - Weak data structure and “instruction” set.
  - Easy to analyse.
- Can TMs implement arbitrary algorithms?
  - Does their computing strength suffice?
  - How efficient are they compared to real-world systems?
- We will look at other models of computation, especially at *random access machines* (RAMs) which model computers capable of handling arbitrarily large integers.

# Motivation – Assembly Language

- C-language code:

```
unsigned int getmax(unsigned int* a) {
    unsigned int max = *a;
    while (*a > 0) {
        if (*a > max)
            max = *a;
        ++a; }
 return max; }
```

- Assembly language (x86-64):

```
getmax:
    pushq   %rbp
    movq    %rsp, %rbp
    movq    %rdi, -24(%rbp)
    movq    -24(%rbp), %rax
    movl    (%rax), %eax
    movl    %eax, -4(%rbp)
    jmp .L2
.L4:
    movq    -24(%rbp), %rax
    movl    (%rax), %eax
    cmpl    -4(%rbp), %eax
    jbe .L3
    movq    -24(%rbp), %rax
    movl    (%rax), %eax
    movl    %eax, -4(%rbp)
.L3:
    addq    $4, -24(%rbp)
.L2:
    movq    -24(%rbp), %rax
    movl    (%rax), %eax
    testl   %eax, %eax
    jne .L4
    movl    -4(%rbp), %eax
    popq    %rbp
    ret
```

# Random Access Machines (RAMs)

- The process of computation of TMs is very distant from real computing systems.
- The architecture of RAMs is, on the other hand, very similar to current computers.
- The results of a complexity analysis of a RAM program is close to the behaviour of real-world computers.
- Moreover, we will observe that computing power of RAMs is equivalent to the power of TMs.

# Basic Definition

- Random Access Machine (RAM):
  - An (infinite) array of registers $R = (r_0, r_1, \ldots)$.
  - Each register is capable of containing an arbitrarily large integer (possibly negative).
  - The possibility to directly access an arbitrary register.
  - Register 0 serves as an accumulator.
  - Program counter $\kappa$.
- A RAM program $\Pi = (\pi_1, \ldots, \pi_m)$ is a finite sequence of instructions.
- The input is placed in a finite array of input registers $I = (i_1, \ldots, i_n)$.
- Three addressing modes: $j$, $\uparrow j$, and $= j$.

# Instruction Set

| Instr. | Op | Semantics | Instr. | Op | Semantics |
|---|---|---|---|---|---|
| READ | $j$ | $r_0 \leftarrow i_j$ | HALF | | $r_0 \leftarrow r_0/2$ |
| READ | $\uparrow j$ | $r_0 \leftarrow i_{r_j}$ | JUMP | $= j$ | $\kappa \leftarrow j$ |
| STORE | $j$ | $r_j \leftarrow r_0$ | JPOS | $= j$ | if $r_0 > 0$ then $\kappa \leftarrow j$ |
| STORE | $\uparrow j$ | $r_{r_j} \leftarrow r_0$ | JZERO | $= j$ | if $r_0 = 0$ then $\kappa \leftarrow j$ |
| LOAD | $x$ | $r_0 \leftarrow x'$ | JNEG | $= j$ | if $r_0 < 0$ then $\kappa \leftarrow j$ |
| ADD | $x$ | $r_0 \leftarrow r_0 + x'$ | HALT | | $\kappa \leftarrow 0$ |
| SUB | $x$ | $r_0 \leftarrow r_0 - x'$ | | | |

note: $x$ (resp. $x'$) is one of $j$, $\uparrow j$, $= j$ (resp. $r_j, r_{r_j}, j$)

# Configurations of a RAM

- A configuration is a pair $C = (\kappa, R)$ where:
  - $\kappa \in \mathbb{N}_0$ is the program counter,
  - $R = \{(j_1, r_{j_1}), \dots, (j_k, r_{j_k})\}$ is a finite set of register-value pairs.
- The initial configuration is $(1, \emptyset)$, i.e., all registers are zeroed.
- Let Π be a RAM program and $I = (i_1, \dots, i_n)$ an input array. Then, the relation $(\kappa, R) \xrightarrow{\Pi, I} (\kappa', R')$ ("yields in one step") is defined as follows:
  - $\kappa'$ is the new value of $\kappa$ after executing the $\kappa$-th instruction of Π,
  - $R'$ is a modified version of $R$ according to the semantics of the $\kappa$-th instruction of Π.
- The relations $(\kappa, R) \xrightarrow{\Pi, I}{}^{k} (\kappa', R')$ and $(\kappa, R) \xrightarrow{\Pi, I}{}^{*} (\kappa', R')$ are defined analogously as for TMs.

# The Function Computed by a RAM

## Definition

*Let* $\Pi$ *be a program,* $D \subseteq \mathbb{Z}^*$ *be a set of finite sequences of integers, and* $\phi$ *be a function* $\phi : D \to \mathbb{Z}$*.* $\Pi$ ***computes*** $\phi$ *iff*

$$\forall I \in D \colon (1, \emptyset) \xrightarrow{\Pi, I}{}^{*} (0, R) \Rightarrow (0, \phi(I)) \in R.$$

## Example (A RAM program computing $\phi(a, b) = |a - b|$ with $I = (5, 8)$)

| # | Program | Configurations |
|---|---|---|
| | | $(1, \emptyset)$ |
| 1 | READ 2 | $(2, \{(0, 8)\})$ |
| 2 | STORE 2 | $(3, \{(0, 8), (2, 8)\})$ |
| 3 | READ 1 | $(4, \{(0, 5), (2, 8)\})$ |
| 4 | STORE 1 | $(5, \{(0, 5), (2, 8), (1, 5)\})$ |
| 5 | SUB 2 | $(6, \{(0, -3), (2, 8), (1, 5)\})$ |
| 6 | JNEG 8 | |
| 7 | HALT | |
| 8 | LOAD 2 | $(9, \{(0, 8), (2, 8), (1, 5)\})$ |
| 9 | SUB 1 | $(10, \{(0, 3), (2, 8), (1, 5)\})$ |
| 10 | HALT | $(0, \{(0, 3), (2, 8), (1, 5)\})$ |

# Time and Space Complexity

- Because RAMs use arbitrarily large integers, the computation cost of an instruction can differ according to the size of the operand.
- Possible approaches:
  1. The size of the operand is ignored:
     - The execution of a RAM instruction can be counted as one time step.
     - A uniform cost of an operation.
  2. The size of the operand is taken into account:
     - The cost of an operation rises logarithmically with the size of the operand.
     - Closer to the behaviour of real-world programs.

# Uniform Time and Space Complexity

- The uniform time complexity of the computation of a RAM program $\Pi$ on the input $I \in D$ is the function $t_\Pi^{uni} : D \to \mathbb{N} \cup \{\infty\}$:
  - $t_\Pi^{uni}(I) = k \iff (1, \emptyset) \xrightarrow{\Pi, I}{}^{k} (0, R)$,
    i.e., a RAM with the program $\Pi$ stops on the input $I$ after $k$ steps.
  - $t_\Pi^{uni}(I) = \infty \iff (1, \emptyset) \not\xrightarrow{\Pi, I}{}^{*} (0, R)$,
    i.e., a RAM with the program $\Pi$ does not stop on the input $I$.
- The uniform space complexity of the computation of a program $\Pi$ on the input $I = (i_1, \ldots, i_n) \in D$ is the function $s_\Pi^{uni} : D \to \mathbb{N} \cup \{\infty\}$:
  - $s_\Pi^{uni}(I) = n + \max\{|R| \mid \exists k \in \mathbb{N}_0 \colon (1, \emptyset) \xrightarrow{\Pi, I}{}^{*} (k, R)\}$,
  - i.e., the length of the input plus the number of used registers.

# Logarithmic Time Complexity

- For an integer $i \in \mathbb{Z}$, let $bin(i)$ be its binary representation[1].
- The length of $i$ is defined as $len(i) = |bin(i)|$.
- The logarithmic time cost function $c^{log}_{(\Pi,I)}$ for a RAM program $\Pi$ and its input $I$, s.t. $(1,\emptyset) \xrightarrow{\Pi,I}{}^{k} (0,R)$, is a mapping

$$c^{log}_{(\Pi,I)} : \begin{cases} \{1,\dots,k\} \to \mathbb{N} \\ i \mapsto \max\{len(x_i) \mid x_i \in X_i(\Pi,I)\} \end{cases}$$

  where $X_i(\Pi,I) \subseteq \mathbb{Z}$ is the set of all register indices, register values, and constants used in the $i$-th step of the program $\Pi$ on the input $I$.
- The logarithmic time complexity of the computation of a RAM program $\Pi$ on the input $I$ is the function $t^{log}_{\Pi} : D \to \mathbb{N} \cup \{\infty\}$:
  - $t^{log}_{\Pi}(I) = \sum_{1 \leq i \leq k} c^{log}_{(\Pi,I)}(i) \iff (1,\emptyset) \xrightarrow{\Pi,I}{}^{k} (0,R)$,
  - $t^{log}_{\Pi}(I) = \infty \iff (1,\emptyset) \not\xrightarrow{\Pi,I}{}^{*} (0,R)$.

[1] We assume no redundant leading 0s and a minus sign in front if negative.

# Logarithmic Space Complexity

- The length of $I = (i_1, \ldots, i_n)$ is defined as

$$len(I) = \sum_{j=1}^{n} len(i_j)$$

- The logarithmic space complexity for register $r$ during the computation of a program $\Pi$ on the input $I = (i_1, \ldots, i_n) \in D$ is the function $s^{log}_{\Pi,r} : D \to \mathbb{N} \cup \{\infty\}$:
  - $s^{log}_{\Pi,r}(I) = \max\{len(v) \mid \exists k \in \mathbb{N}_0 \colon (1, \emptyset) \xrightarrow{\Pi, I}{}^{*} (k, R) \wedge (r, v) \in R\}$
- The logarithmic space complexity of the computation of a program $\Pi$ on the input $I = (i_1, \ldots, i_n)$ is the function $s^{log}_{\Pi} : D \to \mathbb{N} \cup \{\infty\}$:
  - $s^{log}_{\Pi}(I) = len(I) + s^{log}_{\Pi,r_0}(I) + s^{log}_{\Pi,r_1}(I) + \ldots$

# Time and Space Complexity – Size of Input

- The time complexity of the computation of a RAM program $\Pi$ is the function $T_\Pi$:

$$T_\Pi : \begin{cases} \mathbb{N} \rightarrow \mathbb{N} \cup \{\infty\} \\ k \mapsto \max\{t_\Pi(I) \mid len(I) = k\} \end{cases}$$

- The space complexity of the computation of a RAM program $\Pi$ is the function $S_\Pi$:

$$S_\Pi : \begin{cases} \mathbb{N} \rightarrow \mathbb{N} \cup \{\infty\} \\ k \mapsto \max\{s_\Pi(I) \mid len(I) = k\} \end{cases}$$

- $t_\Pi$ (resp. $s_\Pi$) can be either uniform $t_\Pi^{uni}$ ($s_\Pi^{uni}$) or logarithmic $t_\Pi^{log}$ ($s_\Pi^{log}$) time (resp. space) complexity functions.

# Simulation of a TM using a RAM

- We will observe that each TM can be simulated by a RAM program with a linear loss in efficiency.
- For a single-tape TM $M$ with the input alphabet $\Sigma = \{a_1, \ldots, a_k\}$, the input domain $D_\Sigma$ of the simulating RAM program is defined as

$$D_\Sigma = \{(i_1, \ldots, i_n, 0) \mid n \in \mathbb{N} \wedge \forall 1 \leq j \leq n \colon 1 \leq i_j \leq k\}$$

- Then, for each language $L \in \Sigma^*$, we can define $\phi_L : D_\Sigma \to \{0, 1\}$:

$$\phi_L((i_1, \ldots, i_n, 0)) = 1 \iff a_{i_1} \cdots a_{i_n} \in L$$

$$\phi_L((i_1, \ldots, i_n, 0)) = 0 \iff a_{i_1} \cdots a_{i_n} \notin L$$

- Computing $\phi_L$ is equivalent to deciding $L$.

# Simulation of a TM using a RAM

## Proposition

*Let $L \in \mathbf{DTIME}(f(n))$. Then there is a RAM program that computes $\phi_L$ with the uniform (resp. logarithmic) complexity in $O(f(n))$ (resp. $O(f(n) * \log(f(n)))$).*

## Proof (Idea)

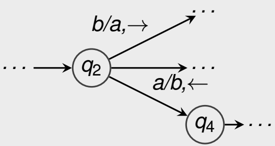
- *For each state q of M, we construct in $\Pi_M$ a subroutine that simulates the behaviour of M in q.*

cells:

| | | | | | | |
|---|---|---|---|---|---|---|
| $\Delta$ | $a$ | $a$ | $b$ | $a$ | $b$ | $\cdots$ |
| | | $\curlywedge$ | | | | |

regs:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 6 | | | $\Delta$ | $a$ | $a$ | $b$ | $a$ | $b$ | $\cdots$ |
| | | | | | | $\curlywedge$ | | | | |




```
q2,a?:   LOAD  ^1
         SUB   =a
         JZERO q2,a
         JUMP  q2,b?
q2,a:    LOAD  =b
         STORE ^1
         LOAD  1
         ADD   =-1
         STORE 1
         JUMP  q4,a?
q2,b?:   LOAD  ^1
         ...
reject:  LOAD  =0
         HALT
accept:  LOAD  =1
         HALT
```

# Simulation of a RAM using a TM

- Now, we will try to show that each RAM can be simulated by a TM with a polynomial loss in efficiency.
- A sequence of integers $I = (i_1, \ldots, i_n)$ can be encoded into the binary representation $code(I)$ as the string $bin(i_1)|\ldots|bin(i_n)$ where the symbol "|" serves as the delimiter.

**Proposition**

*Let $\Pi$ be a RAM program that computes a function $\phi : D \to \mathbb{Z}$ with the uniform time complexity $f(n)$. Then, there exists a 7-tape TM $M_\Pi$ that computes the function $f_{M_\Pi} : \Sigma^* \to \Sigma^*$ for which*

$$f_{M_\Pi}(code(I)) = bin(\phi(I)).$$

*Moreover, $M_\Pi$ computes $\phi$ in the time $O(f(n)^3)$.*

# Simulation of a RAM using a TM – Proof idea 1/3

## Proof (Idea)

- *The tapes of the TM $M_\Pi$ serve for the following purposes:*
  1. *The input $I$*
  2. *Holds the registers' contents*[a] $\Delta(0 : [r_0]) \diamond (1 : [r_1]) \diamond \ldots (n : [r_n])\triangleleft$
  3. *The program counter $\kappa$*
  4. *The currently sought register address*
  
  5-7 *Extra space for the execution*
- *Each instruction of the RAM program $\Pi$ is implemented by a group of states of $M_\Pi$.*
- *Simulating an instruction of $\Pi$ on $M_\Pi$ takes $O(f(n)^2)$ steps.*
  - *Fetching the values of the registers from the second tape takes $O(f(n)^2)$ time (there are $O(f(n))$ pairs, each of the length $O(f(n))$).*
  - *Computation of the result of the instruction on integers of the length $O(f(n))$ can be done in $O(f(n))$ time.*

[a] Update: the old value is replaced by $\# \ldots \#$ and a new one is appended.

# Simulation of a RAM using a TM – Proof idea 2/3

**Proof (Idea)**

- *Based on the previous observation, the simulation of $f(n)$ steps of $\Pi$ takes $O(f(n)^3)$ steps of $M_\Pi$.*
- *It remains to show that after simulating $f(n)$ steps of $\Pi$, the largest integer in the registers has the maximum size of $O(f(n))$.*

**Proposition**

*After the $t$-th step of the computation of a RAM program $\Pi$ on the input $I$, the contents of any register have the length at most $t + len(I) + len(b)$ where $b$ is the largest integer referred to in an instruction of $\Pi$.*

# Simulation of a RAM using a TM – Proof idea 3/3

## Proof (Idea)

- *Base case: the claim is true when $t = 0$.*
- *Induction hypothesis: the claim is true after the $(t-1)$-th step.*
- *Case analysis over instruction types of the $t$-th instruction:*
  - *Most of the instructions do not create new values (jumps, HALT, LOAD, STORE, READ). For these, the claim holds.*
  - *For arithmetic instructions involving a pair of integers, the length of the result is one plus the length of the longest operand, which is by the induction hypothesis at most $t - 1 + len(I) + len(b)$. Thus, the result has the length at most $t + len(I) + len(b)$.*

# Random Access Stored Program (RASP) Machines

- Analogy to universal Turing machines (UTMs):
  - Input: The code of a TM $M$ and an input word $w$.
  - UTM simulates $M$ on the word $w$.
- Universal RAM program:
  - Input registers of $I$ contain an encoded RAM program $\Pi$ and input registers $I_\Pi$ of $\Pi$.
  - The RASP machine simulates the RAM program $\Pi$ with the input $I_\Pi$.

# Random Access Stored Program (RASP) Machines

- A possible encoding of a RAM program:
  - Assign unique code to each instruction-modifier combination.
  - Example: "LOAD $\uparrow$" $\mapsto$ 1, "LOAD $=$" $\mapsto$ 2, ...
  - Keep operand value of instruction separated.
  - Compose the input $I$ as follows:
    $(code_1, op_1, \ldots, code_k, op_k, 0, i_{\Pi_1}, \ldots, i_{\Pi_n})$
- The purpose of the registers of a RAM program:
  - $r_1$ – the program counter of the RAM program $\Pi$,
  - $r_2$ – the pointer to input $I_\Pi$ of the RAM program $\Pi$,
  - $r_3$ – the current instruction,
  - $r_4$ – the current operand value,
  - $r_5$ – an extra register,
  - $r_6, r_7, \ldots$ – registers $r_0, r_1, \ldots$ of the RAM program $\Pi$.

# Random Access Stored Program (RASP) Machines

- Simulation (main loop):
  1. Write the position of the delimiter ("0") to $r_2$.
  2. Read the instruction from the position $(\uparrow 1) * 2$ to $r_3$.
  3. Read the operand value from the position $(\uparrow 1) * 2 + 1$ to $r_4$.
  4. Simulate the instruction with the code in $r_3$. Increment the operand value $j$ by 5, when the addressing modes $\uparrow j$ or $j$ are used.
  5. Update $r_1$ properly.
  6. Goto step 2.

**Proposition**

*If the uniform time complexity of a RAM program $\Pi$ is $O(f(n))$, then the number of the steps of a RASP machine simulating $\Pi$ is upper bounded by $c * f(len(I_\Pi))$.*