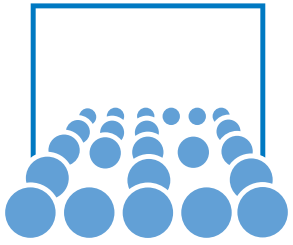
Fundamental Algorithms

Chapter 5: Models and Complexity

Dirk Pflüger

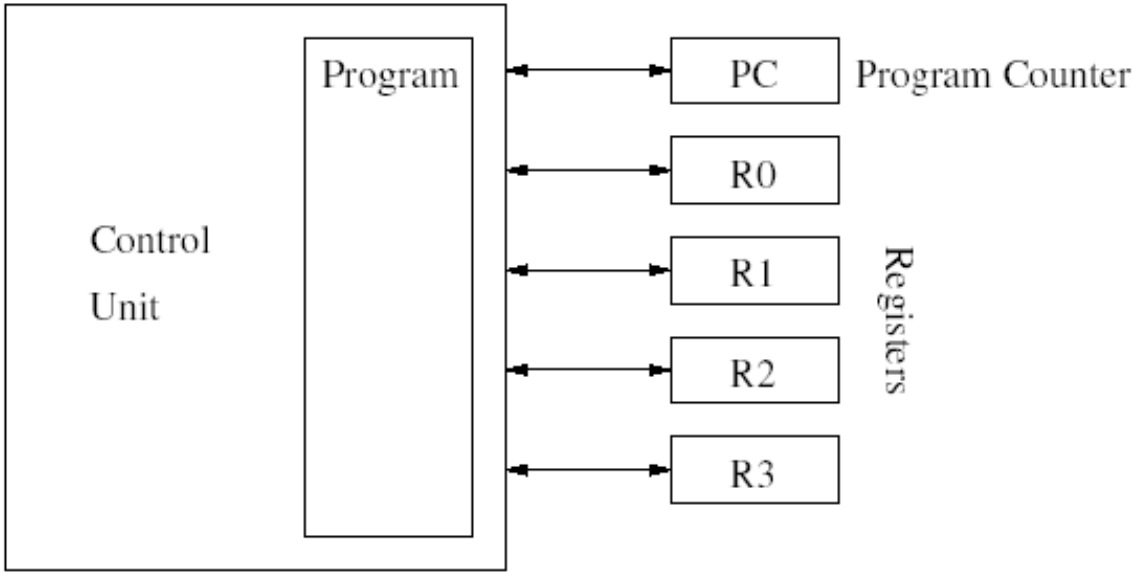
Winter 2010/11



Random Access Machine (RAM)

A **random access machine** (RAM) is a simple model of computation:

- Its memory consists of an unbounded sequence of registers. Each register may hold an integer value.
- The control unit of a RAM holds a program, which consists of a numbered list of statements.
- A program counter determines the next statement.



RAM Programs

Rules for executing a RAM-program:

- in each work cycle the RAM executes one statement of the program
- a program counter specifies the number of the statement that is to be executed
- the program ends when the program counter takes an invalid value

Running a RAM Program:

1. define the program, i.e. the exact list of statements
2. define starting values for the registers (the input)
3. define a starting value for the program counter

Statements in a RAM Program

Statement	Effect on registers	Program Counter
$R_i \leftarrow R_j$	$\langle R_i \rangle := \langle R_j \rangle$	$\langle PC \rangle := \langle PC \rangle + 1$
$R_i \leftarrow RR_j$	$\langle R_i \rangle := \langle R\langle R_j \rangle \rangle$	$\langle PC \rangle := \langle PC \rangle + 1$
$RR_i \leftarrow R_j$	$\langle R\langle R_i \rangle \rangle := \langle R_j \rangle$	$\langle PC \rangle := \langle PC \rangle + 1$
$R_i \leftarrow k$	$\langle R_i \rangle := k$	$\langle PC \rangle := \langle PC \rangle + 1$
$R_i \leftarrow R_j + R_k$	$\langle R_i \rangle := \langle R_j \rangle + \langle R_k \rangle$	$\langle PC \rangle := \langle PC \rangle + 1$
$R_i \leftarrow R_j - R_k$	$\langle R_i \rangle := \max\{0, \langle R_j \rangle - \langle R_k \rangle\}$	$\langle PC \rangle := \langle PC \rangle + 1$
GOTO m		$\langle PC \rangle := m$
IF Ri = 0 GOTO m		$\langle PC \rangle := \begin{cases} m & \text{if } \langle R_i \rangle = 0 \\ \langle PC \rangle + 1 & \text{otherwise} \end{cases}$
IF Ri > 0 GOTO m		$\langle PC \rangle := \begin{cases} m & \text{if } \langle R_i \rangle > 0 \\ \langle PC \rangle + 1 & \text{otherwise} \end{cases}$

Example: Multiplication on a RAM

Idea: multiply x by y by adding up x exactly y times

```
mult (x: Integer, y; Integer) : Integer {
    sum := 0;
    while y > 0 do {
        sum := sum + x;
        y := y − 1;
    }
    return sum;
}
```

Example: Multiplication on a RAM (2)

The RAM program:

1. R3 $\leftarrow$ 1
2. IF R1 = 0 GOTO 6
3. R2 $\leftarrow$ R2 + R0
4. R1 $\leftarrow$ R1 - R3
5. GOTO 2
6. R0 $\leftarrow$ R2
7. STOP

Starting Configuration:

- <R0> := x, <R1> := y, all other registers = 0
- <PC> := 1
- desired result (xy) in <R0>

Uniform Time Complexity

- consider input vectors $(x_1, \ldots, x_m), x_i \in \mathbb{N}$
- starting configuration: Ri = x_{i+1} (otherwise 0)

Definition

Let M be a RAM and $\vec{x} = (x_1, \ldots, x_m)$ be the input of the RAM.

The **uniform size of the input** is defined as $\|\vec{x}\|_{\text{uni}} := m$.

The **uniform time complexity** $T_M^{\text{uni}}(\vec{x})$ of M on $\vec{x}$ is then defined as the number of work cycles M performs on $\vec{x}$.

Example: multiplication RAM for $\vec{x} = (x, y)$

- uniform size of the input: $\|\vec{x}\|_{\text{uni}} = 2$
- uniform time complexity: $T_M^{\text{uni}}((x, y)) = 3 + 4y$ (independent of x)

Logarithmic Time Complexity

Ideas:

- uniform size of the input does not reflect the size of the input numbers (important for multiplication RAM)
- uniform size does not reflect that operations might be more expensive for large operands.

Definition

The **uniform logarithmic size of the input** of a RAM is defined as $\|\vec{x}\|_{\log} := \sum_{i=1}^{m} l(x_i)$, where $l(z)$ is the number of digits to represent z.

- number of digits depends on numbering system
- typically: $l(z) = \log_2 z$

Definition: Logarithmic Time Complexity

The **logarithmic costs** of the RAM's work cycles are defined as:

Statement	log. cost
Ri $\leftarrow$ Rj	$l(\langle Ri \rangle) + 1$
Ri $\leftarrow$ RRj	$l(\langle Rj \rangle) + l(\langle R\langle Rj \rangle \rangle) + 1$
RRi $\leftarrow$ Rj	$l(\langle Ri \rangle) + l(\langle Rj \rangle) + 1$
Ri $\leftarrow$ k	$l(k) + 1$
Ri $\leftarrow$ Rj + Rk	$l(\langle Rj \rangle) + l(\langle Rk \rangle) + 1$
Ri $\leftarrow$ Rj - Rk	$l(\langle Rj \rangle) + l(\langle Rk \rangle) + 1$
GOTO m	1
IF Ri = 0 GOTO m	$l(\langle Ri \rangle) + 1$
IF Ri > 0 GOTO m	$l(\langle Ri \rangle) + 1$

The **logarithmic time complexity** $T_M^{\log}(\vec{x})$ of M on $\vec{x}$ is defined as the sum of the logarithmic costs of all working steps M performs on $\vec{x}$.

"uniformly and logarithmically time-bounded"

Consider a function $t: \mathbb{N} \to \mathbb{N}$,
$\to$ typically $t(n) = n, t(n) = n^2, t(n) = 2^n$, etc.

Definition

A RAM M is called **uniformly $t(n)$-time-bounded**, if $T_M^{\text{uni}}(\vec{x}) \leq t(n)$ for all $\vec{x}: \|\vec{x}\|_{\text{uni}} = n$.

(for all inputs of uniform size n, the uniform time complexity has to be bounded by $t(n)$)

Definition

A RAM M is called **logarithmically $t(n)$-time-bounded**, if $T_M^{\log}(\vec{x}) \leq t(n)$ for all $\vec{x}: \|\vec{x}\|_{\log} = n$.

(same definition with logarithmical size and time complexity)

"uniformly and logarithmically time-bounded"

Example: Multiplication RAM

- uniform time complexity is $T_M^{\text{uni}}((x,y)) = 4y + 3$
- uniform input size is $\|\vec{x}\|_{\text{uni}} = 2$
- hence, there is no such function $t(n)$
  (y becomes arbitrarily large)

Thus, **M is not uniformly $t(n)$-time-bounded** for any function t

- logarithmic time complexity is
  $T_M^{\log}((x,y)) \leq 5 + y(5 + 3\|(x,y)\|_{\log}) + \|(x,y)\|_{\log}$
- logarithmic input size is $\|(x,y)\|_{\log} = l(x) + l(y)$
- $T_M^{\log}((x,y))$ grows with y, while $\|(x,y)\|_{\log}$ grows with $\log(y)$

M has an exponential time complexity w.r.t. the logarithmic complexity measure.