Instruction

Type	Instruction	Response of the URM
Zero	$Z(n)$	Replace r_n by 0. ($0 \to R_n$, or $r_n := 0$)
Successor	$S(n)$	Add 1 to r_n. ($r_n + 1 \to R_n$, or $r_n := r_n + 1$)
Transfer	$T(m, n)$	Copy r_m to R_n. ($r_m \to R_n$, or $r_n := r_m$)
Jump	$J(m, n, q)$	If $r_m = r_n$, go to the q-th instruction; otherwise go to the next instruction.

$Z(n)$, $S(n)$, $T(m, n)$ are arithmetic instructions.

Outline

Configuration and Instructions

Example: The initial registers are:

R_1	R_2	R_3	R_4	R_5	R_6	R_7	$\cdots$
9	7	0	0	0	0	0	$\ldots$

The program is:

$$
\begin{aligned}
I_1 &: J(1,2,6)\\
I_2 &: S(2)\\
I_3 &: S(3)\\
I_4 &: J(1,2,6)\\
I_5 &: J(1,1,2)\\
I_6 &: T(3,1)
\end{aligned}
$$

Configuration and Computation

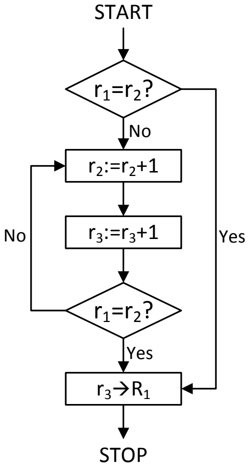
Configuration:
the contents of the registers + the current instruction number.

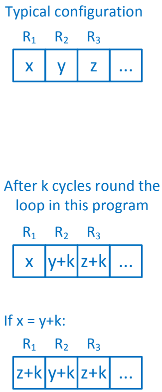
Initial configuration, computation, final configuration.

Operation of URM under a program P

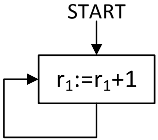
- $P = \{I_1, I_2, \cdots, I_s\} \to$ URM
- URM starts by obeying instruction I_1
- When URM finishes obeying I_k, it proceeds to the next instruction in the computation,
  - if I_k is not a jump instruction, then the next instruction is I_{k+1};
  - if $I_k = J(m, n, q)$ then next instruction is (1) I_q, if $r_m = r_n$; or (2) I_{k+1}, otherwise.
- Computation stops when the next instruction is I_v, where $v > s$.
  - if $k = s$, and I_s is an arithmetic instruction;
  - if $I_k = J(m, n, q)$, $r_m = r_n$ and $q > s$;
  - if $I_k = J(m, n, q)$, $r_m \neq r_n$ and $k = s$.

Flow Diagram





- $J(m, m, q)$ is alertunconditional jump
- Computations that never stop



Some Notation

Suppose P is the program of a URM and $a_1, a_2, a_3, \ldots$ are the numbers stored in the registers.

- $P(a_1, a_2, a_3, \ldots)$ is the initial configuration.
- $P(a_1, a_2, a_3, \ldots) \downarrow$ means that the computation converges.
- $P(a_1, a_2, a_3, \ldots) \uparrow$ means that the computation diverges.
- $P(a_1, a_2, \ldots, a_m)$ is $P(a_1, a_2, \ldots, a_m, 0, 0, \ldots)$.

Outline

URM-Computable Function

URM-Computable Function

What does it mean that a URM computes a (partial) n-ary function f?

URM-Computable Function

What does it mean that a URM computes a (partial) n-ary function f?

Let P be the program of a URM and $a_1, \ldots, a_n, b \in \mathbb{N}$. When computation $P(a_1, \ldots, a_n)$ converges to b if $P(a_1, \ldots, a_n) \downarrow$ and $r_1 = b$ in the final configuration. We write $P(a_1, \ldots, a_n) \downarrow b$.

- P URM-computes f if, for all $a_1, \ldots, a_n, b \in \mathbb{N}$,

$$P(a_1, \ldots, a_n) \downarrow b \text{ iff } f(a_1, \ldots, a_n) = b$$

- Function f is URM-computable if there is a program that URM-computes f.
- (We abbreviate "URM-computable" to "computable")

Computable Functions

Let

$\mathscr{C}$ be the set of computable functions and

$\mathscr{C}_n$ be the set of n-ary computable functions.

Examples

Construct a URM that computes $x + y$.

Examples

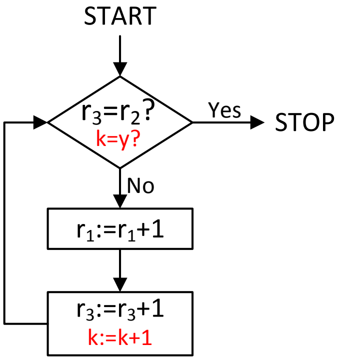
Construct a URM that computes $x + y$.

$$
\begin{aligned}
I_1 &: \; J(3,2,5) \\
I_2 &: \; S(1) \\
I_3 &: \; S(3) \\
I_4 &: \; J(1,1,1)
\end{aligned}
$$

Examples

Construct a URM that computes $x + y$.

$$
\begin{aligned}
I_1 &: J(3,2,5) \\
I_2 &: S(1) \\
I_3 &: S(3) \\
I_4 &: J(1,1,1)
\end{aligned}
$$



Examples

Construct a URM that computes $x \dot{-} 1 = \begin{cases} x-1, & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases}$

Examples

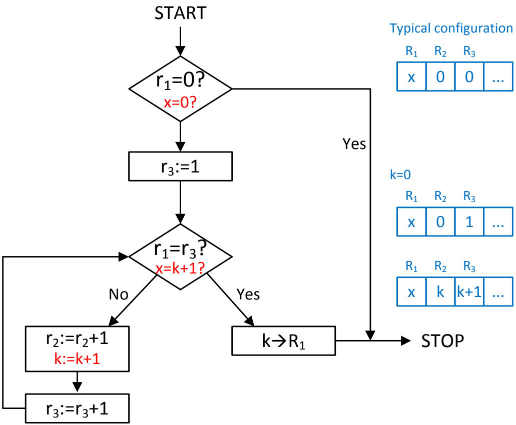
Construct a URM that computes $x \dot{-} 1 = \begin{cases} x-1, & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases}$

$$
\begin{aligned}
I_1 &: \ J(1,4,8) \\
I_2 &: \ S(3) \\
I_3 &: \ J(1,3,7) \\
I_4 &: \ S(2) \\
I_5 &: \ S(3) \\
I_6 &: \ J(1,1,3) \\
I_7 &: \ T(2,1)
\end{aligned}
$$

Examples

Construct a URM that computes $x \dot{-} 1 = \begin{cases} x-1, & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases}$

$$\begin{aligned} I_1 &: J(1,4,8) \\ I_2 &: S(3) \\ I_3 &: J(1,3,7) \\ I_4 &: S(2) \\ I_5 &: S(3) \\ I_6 &: J(1,1,3) \\ I_7 &: T(2,1) \end{aligned}$$



Examples

Construct a URM that computes $x \div 2 = \begin{cases} x/2, & \text{if } x \text{ is even,} \\ \text{undefined}, & \text{if } x \text{ is odd.} \end{cases}$

Examples

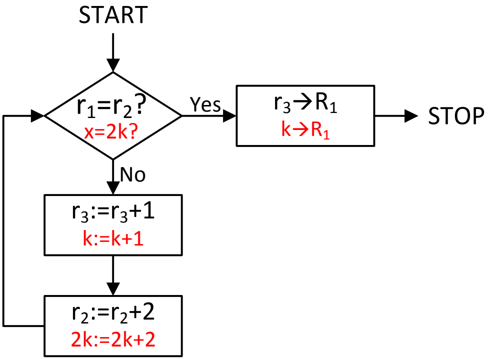
Construct a URM that computes $x \div 2 = \begin{cases} x/2, & \text{if } x \text{ is even,} \\ \text{undefined,} & \text{if } x \text{ is odd.} \end{cases}$

$$
\begin{aligned}
I_1 &: J(1,2,6) \\
I_2 &: S(3) \\
I_3 &: S(2) \\
I_4 &: S(2) \\
I_5 &: J(1,1,1) \\
I_6 &: T(3,1)
\end{aligned}
$$

Examples

Construct a URM that computes $x \div 2 = \begin{cases} x/2, & \text{if } x \text{ is even,} \\ \text{undefined,} & \text{if } x \text{ is odd.} \end{cases}$

$$\begin{aligned} I_1 &: J(1,2,6) \\ I_2 &: S(3) \\ I_3 &: S(2) \\ I_4 &: S(2) \\ I_5 &: J(1,1,1) \\ I_6 &: T(3,1) \end{aligned}$$



Function Defined by Program

Given any program P and $n \geq 1$, by thinking of the effect of P on initial configurations of the form $a_1, \cdots, a_n, 0, 0, \cdots$, there is a unique n-ary function that P computes, denoted by $f_P^{(n)}$.

$$f_P^{(n)}(a_1, \ldots, a_n) = \begin{cases} b, & \text{if } P(a_1, \ldots, a_n) \downarrow b, \\ \text{undefined}, & \text{if } P(a_1, \ldots, a_n) \uparrow. \end{cases}$$

Outline

Predicate and Decision Problem

The value of a predicate is either 'true' or 'false'.

The answer of a *decision problem* is either 'yes' or 'no'.

Example: Given two numbers x, y, check whether x is a multiple of y.
Input: x, y;
Output: 'Yes' or 'No'.

The operation amounts to calculation of the function

$$f(x,y) = \begin{cases} 1, & \text{if } x \text{ is a multiple of } y, \\ 0, & \text{if otherwise.} \end{cases}$$

Thus the property or predicate 'x is a multiple of y' is algorithmically or effectively decidable, or just decidable if function f is computable.

Decidable Predicate and Decidable Problem

Suppose that $M(x_1, \ldots, x_n)$ is an n-ary predicate of natural numbers. The characteristic function $c_M(\mathbf{x})$, where $\mathbf{x} = x_1, \ldots, x_n$, is given by

$$f_P^{(n)}(a_1, \ldots, a_n) = \begin{cases} 1, & \text{if } M(\mathbf{x}) \text{ holds}, \\ 0, & \text{if otherwise}. \end{cases}$$

The predicate $M(\mathbf{x})$ is decidable if c_M is computable; it is undecidable otherwise.

Computability on other Domains

Suppose D is an object domain. A coding of D is an explicit and effective injection $\alpha : D \to \mathbb{N}$. We say that an object $d \in D$ is coded by the natural number $\alpha(d)$.

A function $f : D \to D$ extends to a numeric function $f^* : \mathbb{N} \to \mathbb{N}$. We say that f is computable if f^* is computable.

$$f^* = \alpha \circ f \circ \alpha^{-1}$$

Example

Consider the domain $\mathbb{Z}$. An explicit coding is given by the function α where

$$\alpha(n) = \begin{cases} 2n, & \text{if } n \geq 0, \\ -2n-1, & \text{if } n < 0. \end{cases}$$

Then α^{-1} is given by

$$\alpha^{-1}(m) = \begin{cases} \frac{1}{2}m, & \text{if } m \text{ is even}, \\ -\frac{1}{2}(m+1), & \text{if } m \text{ is odd}. \end{cases}$$

Example (Continued)

Consider the function $f(x) = x - 1$ on $\mathbb{Z}$, then $f^* : \mathbb{N} \to \mathbb{N}$ is given by

$$f^*(x) = \begin{cases} 1 & \text{if } x = 0 \text{ (i.e. } x = \alpha(0)), \\ x - 2 & \text{if } x > 0 \text{ and } x \text{ is even } (\text{i.e. } x = \alpha(n), n > 0), \\ x + 2 & \text{if } x \text{ is odd } (\text{i.e. } x = \alpha(n), n < 0). \end{cases}$$

It is a routine exercise to write a program that computes f^*, hence $x - 1$ is a computable function on $\mathbb{Z}$.

Outline

Remark

Register Machines are more advanced than Turing Machines.

Remark

Register Machines are more advanced than Turing Machines.

Register Machine Models can be classified into three groups:

- CM (Counter Machine Model).
- RAM (Random Access Machine Model).
- RASP (Random Access Stored Program Machine Model).

Remark

Register Machines are more advanced than Turing Machines.

Register Machine Models can be classified into three groups:

- CM (Counter Machine Model).
- RAM (Random Access Machine Model).
- RASP (Random Access Stored Program Machine Model).

The Unlimited Register Machine Model belongs to the CM class.

Finiteness

Every URM uses only a fixed finite number of registers, no matter how large an input number is.

Finiteness

Every URM uses only a fixed finite number of registers, no matter how large an input number is.

This is a fine property of Counter Machine Model.

Outline

Sequential Composition

Given Programs P and Q, how do we construct the sequential composition $P; Q$?

The jump instructions of P and Q must be modified.

Sequential Composition

Given Programs P and Q, how do we construct the sequential composition $P; Q$?

The jump instructions of P and Q must be modified.

Standard Form: A program $P = I_1, \ldots, I_s$ is in *standard form* if, for every jump instruction $J(m, n, q)$ we have $q \leq s + 1$.

Lemma

For any program P there is a program P^* in standard form such that any computation under P^* is identical to the corresponding computation under P. In particular, for any $a_1, \cdots, a_n, b$,

$$P(a_1, \cdots, a_n) \downarrow b \text{ if and only if } P^*(a_1, \cdots, a_n) \downarrow b,$$

and hence $f_P^{(n)} = f_{P^*}^{(n)}$ for every $n > 0$.

Proof

Suppose that $P = I_1, I_2, \cdots, I_s$. Put $P^* = I_1^*, I_2^*, \cdots, I_s^*$ where

if I_k is not a jump instruction, then $I_k^* = I_k$;

if I_k is not a jump instruction, then $I_k^* = \begin{cases} I_k & \text{if } q \leq s+1, \\ J(m, n, s+1) & \text{if } q > s+1. \end{cases}$

Join/Concatenation

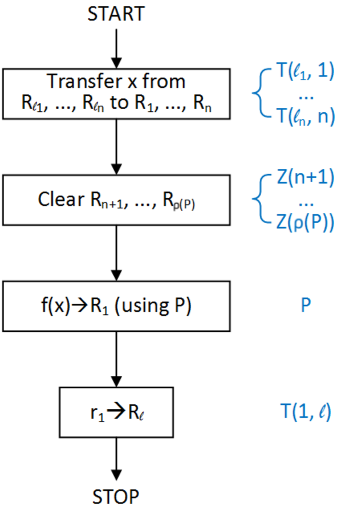
Let P and Q be programs of lengths s, t respectively, in standard form. The *join* or *concatenation* of P and Q, written PQ or $\frac{P}{Q}$, is a program $I_1, I_2, \cdots, I_s, I_{s+1}, \cdots, I_{s+t}$ where $P = I_1, \cdots, I_s$ and the instructions $I_{s+1}, \cdots, I_{s+t}$ are the instructions of Q with each jump $J(m, n, q)$ replaced by $J(m, n, s + q)$.

Program as Subroutine

Suppose the program P computes f.

Let $\rho(P)$ be the least number i such that the register R_i is not used by the program P.

The notation $P[l_1, \ldots, l_n \to l]$ stands for the following program:



$$\begin{array}{rcl}
I_1 & : & T(l_1, 1) \\
 & \vdots & \\
I_n & : & T(l_n, n) \\
I_{n+1} & : & Z(n+1) \\
 & \vdots & \\
I_{\rho(P)} & : & Z(\rho(P)) \\
_ & : & P \\
_ & : & T(1, l)
\end{array}$$