Definition. A register machine is specified by: (Recall)

- finitely many registers $\mathtt{R}_0$, $\mathtt{R}_1$, . . . , $\mathtt{R}_n$ (each capable of storing a natural number);
- a program consisting of a finite list of instructions of the form $\mathit{label} : \mathit{body}$, where for $i = 0, 1, 2, \ldots$, the $(i+1)^{\text{th}}$ instruction has label $\mathtt{L}_i$.

Instruction body takes one of three forms:

$R^+ \to L'$	add 1 to contents of register R and jump to instruction labelled L'
$R^- \to L', L''$	if contents of R is > 0, then subtract 1 from it and jump to L', else jump to L''
$\mathtt{HALT}$	stop executing instructions

Recall:

Computable functions

Definition. $f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ is (register machine) computable if there is a register machine M with at least $n+1$ registers R_0, R_1, . . . , R_n (and maybe more) such that for all $(x_1, \ldots, x_n) \in \mathbb{N}^n$ and all $y \in \mathbb{N}$,

> the computation of M starting with $R_0 = 0$, $R_1 = x_1$, . . . , $R_n = x_n$ and all other registers set to 0, halts with $R_0 = y$

if and only if $f(x_1, \ldots, x_n) = y$.

N.B. there may be many different M that compute the same partial function f.

Coding programs as numbers

Turing/Church solution of the Entscheidungsproblem uses the idea that (formal descriptions of) algorithms can be the data on which algorithms act.

To realize this idea with Register Machines we have to be able to code RM programs as numbers. (In general, such codings are often called Gödel numberings.)

"Effective" numerical codes

RM program

initial contents of R1, ..., Rn

final contents of R0 (if halts)

$$\mathrm{Prog}, [x_1, \ldots, x_n] \longmapsto y$$

run the RM

"Effective" numerical codes

$$\mathrm{Prog},\ [x_1,\dots,x_n] \longmapsto y$$

code $\downarrow$ $\uparrow$ decode

$$\langle \ulcorner \mathrm{Prog} \urcorner, \ulcorner [x_1,\dots,x_n] \urcorner \rangle$$

a number

want numerical codings

$$\langle -,- \rangle,\ \ulcorner - \urcorner,\ \ulcorner [-,\dots,-] \urcorner$$

so that

$$\cdot \xrightarrow{\text{decode}} \cdot \overset{\text{run}}{\longmapsto} \cdot$$

is RM computable

Numerical coding of pairs

$\{0, 1, 2, 3, \ldots\}$

For $x, y \in \mathbb{N}$, define $\begin{cases} \langle\!\langle x, y \rangle\!\rangle \triangleq 2^x(2y+1) \\ \langle x, y \rangle \triangleq 2^x(2y+1) - 1 \end{cases}$

left-hand side is equal to
the right-hand side <u>by definition</u>

Numerical coding of pairs

For $x, y \in \mathbb{N}$, define $\begin{cases} \langle\!\langle x, y \rangle\!\rangle \triangleq 2^x(2y+1) \\ \langle x, y \rangle \triangleq 2^x(2y+1) - 1 \end{cases}$

$\langle\!\langle x,y \rangle\!\rangle$	0	1	2	...
0	1	3	5	...
1	2	6	10	...
2	4	12	20	...
⋮	⋮	⋮	⋮	

$\langle x,y \rangle$	0	1	2	...
0	0	2	4	...
1	1	5	9	...
2	3	11	19	...
⋮	⋮	⋮	⋮	

Numerical coding of pairs

For $x, y \in \mathbb{N}$, define $\begin{cases} \langle\!\langle x,y \rangle\!\rangle & \triangleq \; 2^x(2y+1) \\ \langle x,y \rangle & \triangleq \; 2^x(2y+1) - 1 \end{cases}$

So

$$\begin{array}{rcl} \boxed{0\mathrm{b}\langle\!\langle x,y \rangle\!\rangle} & = & \boxed{0\mathrm{b}y}\boxed{1}\boxed{\overbrace{0\cdots0}^{x\ 0\mathrm{s}}} \\ \boxed{0\mathrm{b}\langle x,y \rangle} & = & \boxed{0\mathrm{b}y}\boxed{0}\boxed{\underbrace{1\cdots1}_{x\ 1\mathrm{s}}} \end{array}$$

(Notation: $0\mathrm{b}x \triangleq$ *x in binary*.)

E.g. $27 = 0\mathrm{b}11011 = \langle\!\langle 0,13 \rangle\!\rangle = \langle 2,3 \rangle$

Numerical coding of pairs

For $x, y \in \mathbb{N}$, define $\begin{cases} \langle\langle x, y \rangle\rangle & \triangleq & 2^x(2y+1) \\ \langle x, y \rangle & \triangleq & 2^x(2y+1) - 1 \end{cases}$

So

$$\boxed{0\mathrm{b}\langle\langle x, y \rangle\rangle} = \boxed{0\mathrm{b}y \mid 1 \mid 0 \cdots 0}$$

$$\boxed{0\mathrm{b}\langle x, y \rangle} = \boxed{0\mathrm{b}y \mid 0 \mid 1 \cdots 1}$$

$\langle -, - \rangle$ gives a bijection (one-one correspondence) between $\mathbb{N} \times \mathbb{N}$ and $\mathbb{N}$.

$\langle\langle -, - \rangle\rangle$ gives a bijection between $\mathbb{N} \times \mathbb{N}$ and $\{n \in \mathbb{N} \mid n \neq 0\}$.

Numerical coding of lists

$\mathit{list}\,\mathbb{N} \triangleq$ set of all finite lists of natural numbers, using ML notation for lists:

- empty list: $[]$
- list-cons: $x :: \ell \in \mathit{list}\,\mathbb{N}$ (given $x \in \mathbb{N}$ and $\ell \in \mathit{list}\,\mathbb{N}$)
- $[x_1, x_2, \ldots, x_n] \triangleq x_1 :: (x_2 :: (\cdots x_n :: [] \cdots))$

Numerical coding of lists

$\mathit{list}\,\mathbb{N} \triangleq$ set of all finite lists of natural numbers, using ML notation for lists.

For $\ell \in \mathit{list}\,\mathbb{N}$, define $\ulcorner \ell \urcorner \in \mathbb{N}$ by induction on the length of the list ℓ:

$$\begin{cases} \ulcorner [] \urcorner & \triangleq & 0 \\ \ulcorner x :: \ell \urcorner & \triangleq & \langle\!\langle x, \ulcorner \ell \urcorner \rangle\!\rangle = 2^x (2 \cdot \ulcorner \ell \urcorner + 1) \end{cases}$$

Thus $\ulcorner [x_1, x_2, \ldots, x_n] \urcorner = \langle\!\langle x_1, \langle\!\langle x_2, \cdots \langle\!\langle x_n, 0 \rangle\!\rangle \cdots \rangle\!\rangle \rangle\!\rangle$

Numerical coding of lists

$\mathit{list}\,\mathbb{N} \triangleq$ set of all finite lists of natural numbers, using ML notation for lists.

For $\ell \in \mathit{list}\,\mathbb{N}$, define $\ulcorner \ell \urcorner \in \mathbb{N}$ by induction on the length of the list ℓ:

$$\begin{cases} \ulcorner [] \urcorner & \triangleq & 0 \\ \ulcorner x :: \ell \urcorner & \triangleq & \langle\!\langle x, \ulcorner \ell \urcorner \rangle\!\rangle = 2^x (2 \cdot \ulcorner \ell \urcorner + 1) \end{cases}$$

For example:

$\ulcorner [3] \urcorner = \ulcorner 3 :: [] \urcorner = \langle\!\langle 3, 0 \rangle\!\rangle = 2^3 (2 \cdot 0 + 1) = 8 = 0\mathrm{b}1000$

$\ulcorner [1,3] \urcorner = \langle\!\langle 1, \ulcorner [3] \urcorner \rangle\!\rangle = \langle\!\langle 1, 8 \rangle\!\rangle = 34 = 0\mathrm{b}100010$

$\ulcorner [2,1,3] \urcorner = \langle\!\langle 2, \ulcorner [1,3] \urcorner \rangle\!\rangle = \langle\!\langle 2, 34 \rangle\!\rangle = 276 = 0\mathrm{b}100010100$

Numerical coding of lists

$\mathit{list}\,\mathbb{N} \triangleq$ set of all finite lists of natural numbers, using ML notation for lists.

For $\ell \in \mathit{list}\,\mathbb{N}$, define $\ulcorner \ell \urcorner \in \mathbb{N}$ by induction on the length of the list ℓ:

$$\begin{cases} \ulcorner [] \urcorner & \triangleq \mathbf{0} \\ \ulcorner x :: \ell \urcorner & \triangleq \langle\!\langle x, \ulcorner \ell \urcorner \rangle\!\rangle = 2^x (2 \cdot \ulcorner \ell \urcorner + 1) \end{cases}$$

For example:

$$\ulcorner [3] \urcorner = \ulcorner 3 :: [] \urcorner = \langle\!\langle 3, 0 \rangle\!\rangle = 2^3 (2 \cdot 0 + 1) = 8 = 0\mathrm{b}1\overbrace{000}^{3}$$

$$\ulcorner [1,3] \urcorner = \langle\!\langle 1, \ulcorner [3] \urcorner \rangle\!\rangle = \langle\!\langle 1, 8 \rangle\!\rangle = 34 = 0\mathrm{b}1\overbrace{000}^{3}1\overbrace{0}^{1}$$

$$\ulcorner [2,1,3] \urcorner = \langle\!\langle 2, \ulcorner [1,3] \urcorner \rangle\!\rangle = \langle\!\langle 2, 34 \rangle\!\rangle = 276 = 0\mathrm{b}1\underbrace{000}_{3}1\underbrace{0}_{1}1\underbrace{00}_{2}$$

Numerical coding of lists

$list\,\mathbb{N} \triangleq$ set of all finite lists of natural numbers, using ML notation for lists.

For $\ell \in list\,\mathbb{N}$, define $\ulcorner \ell \urcorner \in \mathbb{N}$ by induction on the length of the list ℓ:

$$\begin{cases} \ulcorner [] \urcorner & \triangleq \mathbf{0} \\ \ulcorner x :: \ell \urcorner & \triangleq \langle\!\langle x, \ulcorner \ell \urcorner \rangle\!\rangle = 2^x (2 \cdot \ulcorner \ell \urcorner + 1) \end{cases}$$

$$\boxed{0\mathrm{b}\ulcorner [x_1, x_2, \ldots, x_n] \urcorner} = \boxed{1}\underbrace{\boxed{0\cdots0}}_{x_n\ 0\text{s}}\boxed{1}\underbrace{\boxed{0\cdots0}}_{x_{n-1}\ 0\text{s}}\cdots\boxed{1}\underbrace{\boxed{0\cdots0}}_{x_1\ 0\text{s}}$$

Numerical coding of lists

$\mathit{list}\,\mathbb{N} \triangleq$ set of all finite lists of natural numbers, using ML notation for lists.

For $\ell \in \mathit{list}\,\mathbb{N}$, define $\ulcorner \ell \urcorner \in \mathbb{N}$ by induction on the length of the list ℓ:

$$\begin{cases} \ulcorner [] \urcorner & \triangleq & \mathbf{0} \\ \ulcorner x :: \ell \urcorner & \triangleq & \langle\!\langle x, \ulcorner \ell \urcorner \rangle\!\rangle = \mathbf{2}^x(\mathbf{2} \cdot \ulcorner \ell \urcorner + \mathbf{1}) \end{cases}$$

$$\boxed{0\mathrm{b}\ulcorner [x_1, x_2, \ldots, x_n] \urcorner} = \boxed{\mathbf{1}\;|\;\mathbf{0}\cdots\mathbf{0}}\;\boxed{\mathbf{1}\;|\;\mathbf{0}\cdots\mathbf{0}}\cdots\boxed{\mathbf{1}\;|\;\mathbf{0}\cdots\mathbf{0}}$$

Hence $\ell \mapsto \ulcorner \ell \urcorner$ gives a bijection from $\mathit{list}\,\mathbb{N}$ to $\mathbb{N}$.

Numerical coding of programs

If P is the RM program $\boxed{\begin{array}{c} \mathrm{L}_0 : body_0 \\ \mathrm{L}_1 : body_1 \\ \vdots \\ \mathrm{L}_n : body_n \end{array}}$

then its numerical code is

$$\ulcorner P \urcorner \triangleq \ulcorner [\ulcorner body_0 \urcorner, \ldots, \ulcorner body_n \urcorner] \urcorner$$

where the numerical code $\ulcorner body \urcorner$ of an instruction body is defined by:
$$\begin{cases} \ulcorner \mathrm{R}_i^+ \to \mathrm{L}_j \urcorner & \triangleq & \langle\langle 2i, j \rangle\rangle \\ \ulcorner \mathrm{R}_i^- \to \mathrm{L}_j, \mathrm{L}_k \urcorner & \triangleq & \langle\langle 2i+1, \langle j, k \rangle \rangle\rangle \\ \ulcorner \mathtt{HALT} \urcorner & \triangleq & 0 \end{cases}$$

Any $x \in \mathbb{N}$ decodes to a unique instruction $body(x)$:

if $x = 0$ then $body(x)$ is $\mathtt{HALT}$,
else ($x > 0$ and) let $x = \langle\langle y, z \rangle\rangle$ in
if $y = 2i$ is even, then
$body(x)$ is $R_i^+ \to L_z$,
else $y = 2i + 1$ is odd, let $z = \langle j, k \rangle$ in
$body(x)$ is $R_i^- \to L_j, L_k$

So any $e \in \mathbb{N}$ decodes to a unique program $prog(e)$, called the register machine program with index e:

$$prog(e) \triangleq \boxed{\begin{array}{c} L_0 : body(x_0) \\ \vdots \\ L_n : body(x_n) \end{array}} \quad \text{where } e = \ulcorner [x_0, \ldots, x_n] \urcorner$$

Example of $prog(e)$

- $786432 = 2^{19} + 2^{18} = 0\mathtt{b}11\underbrace{0\ldots0}_{18\ \text{"0"s}} = \ulcorner[18,0]\urcorner$
- $18 = 0\mathtt{b}10010 = \langle\!\langle 1,4\rangle\!\rangle = \langle\!\langle 1,\langle 0,2\rangle\rangle\!\rangle = \ulcorner \mathtt{R}_0^- \to \mathtt{L}_0,\mathtt{L}_2\urcorner$
- $0 = \ulcorner\mathtt{HALT}\urcorner$

So $prog(786432) = \boxed{\begin{array}{l}\mathtt{L}_0 : \mathtt{R}_0^- \to \mathtt{L}_0,\mathtt{L}_2 \\ \mathtt{L}_1 : \mathtt{HALT}\end{array}}$

Example of $prog(e)$

- $786432 = 2^{19} + 2^{18} = 0b11\underbrace{0\ldots0}_{18\ \text{"0"s}} = \ulcorner[18,0]\urcorner$
- $18 = 0b10010 = \langle\!\langle 1,4 \rangle\!\rangle = \langle\!\langle 1, \langle 0,2 \rangle \rangle\!\rangle = \ulcorner R_0^- \to L_0, L_2 \urcorner$
- $0 = \ulcorner \mathtt{HALT} \urcorner$

So $prog(786432) = \boxed{\begin{array}{l} L_0 : R_0^- \to L_0, L_2 \\ L_1 : \mathtt{HALT} \end{array}}$

N.B. jump to label with no body (erroneous halt)

What function is computed by a RM with prog(786432) as its program?

$$666 = \mathtt{0b}1010011010$$

$$= \ulcorner [1, 1, 0, 2, 1] \urcorner$$

$$\mathrm{prog}(666) = \boxed{\begin{array}{l} L_0 : R_0^+ \to L_0 \\ L_1 : R_0^+ \to L_0 \\ L_2 : \mathrm{HALT} \\ L_3 : R_0^- \to L_0, L_0 \\ L_4 : R_0^+ \to L_0 \end{array}}$$

(never halts!)

What partial function does this compute?

Example of $prog(e)$

- $786432 = 2^{19} + 2^{18} = 0b11\underbrace{0\ldots0}_{18\ ''0''s} = \ulcorner[18,0]\urcorner$
- $18 = 0b10010 = \langle\langle 1,4\rangle\rangle = \langle\langle 1,\langle 0,2\rangle\rangle\rangle = \ulcorner R_0^- \to L_0, L_2\urcorner$
- $0 = \ulcorner \mathtt{HALT}\urcorner$

So $prog(786432) = \boxed{\begin{array}{l} L_0 : R_0^- \to L_0, L_2 \\ L_1 : \mathtt{HALT} \end{array}}$

N.B. In case $e = 0$ we have $0 = \ulcorner[\,]\urcorner$, so $prog(0)$ is the program with an empty list of instructions, which by convention we regard as a RM that does nothing (i.e. that halts immediately).

"Effective" numerical codes

$$\mathrm{Prog},\ [x_1,\dots,x_n] \longmapsto y$$

code $\downarrow$ $\uparrow$ decode

$$\langle \ulcorner \mathrm{Prog} \urcorner, \ulcorner [x_1,\dots,x_n] \urcorner \rangle$$

a number

want numerical codings

$\langle -,- \rangle$, $\ulcorner - \urcorner$, $\ulcorner [-,\dots,-] \urcorner$

so that

$$\cdot \xrightarrow{\text{decode}} \cdot \xmapsto{\text{run}} \cdot$$

is RM computable

Universal register machine, U

High-level specification

Universal RM U carries out the following computation, starting with $\mathtt{R}_0 = 0$, $\mathtt{R}_1 = e$ (code of a program), $\mathtt{R}_2 = a$ (code of a list of arguments) and all other registers zeroed:

- decode e as a RM program P
- decode a as a list of register values $a_1, \ldots, a_n$
- carry out the computation of the RM program P starting with $\mathtt{R}_0 = 0, \mathtt{R}_1 = a_1, \ldots, \mathtt{R}_n = a_n$ (and any other registers occurring in P set to 0).

Mnemonics for the registers of U and the role they play in its program:

$R_1 \equiv \mathtt{P}$ code of the RM to be simulated

$R_2 \equiv \mathtt{A}$ code of current register contents of simulated RM

$R_3 \equiv \mathtt{PC}$ program counter—number of the current instruction (counting from $\mathbf{0}$)

$R_4 \equiv \mathtt{N}$ code of the current instruction body

$R_5 \equiv \mathtt{C}$ type of the current instruction body

$R_6 \equiv \mathtt{R}$ current value of the register to be incremented or decremented by current instruction (if not $\mathtt{HALT}$)

$R_7 \equiv \mathtt{S}$, $R_8 \equiv \mathtt{T}$ and $R_9 \equiv \mathtt{Z}$ are auxiliary registers.

Overall structure of U's program

$\boxed{1}$ copy $\mathtt{PC}$th item of list in $\mathtt{P}$ to $\mathtt{N}$ (halting if $\mathtt{PC} >$ length of list); goto $\boxed{2}$

$\boxed{2}$ if $\mathtt{N} = \mathbf{0}$ then copy $\mathbf{0}$th item of list in $\mathtt{A}$ to $\mathtt{R}_0$ and halt, else (decode $\mathtt{N}$ as $\langle\!\langle y, z \rangle\!\rangle$; $\mathtt{C} ::= y$; $\mathtt{N} ::= z$; goto $\boxed{3}$)

{at this point either $\mathtt{C} = 2i$ is even and current instruction is $\mathtt{R}_i^+ \to \mathtt{L}_z$,
or $\mathtt{C} = 2i + 1$ is odd and current instruction is $\mathtt{R}_i^- \to \mathtt{L}_j, \mathtt{L}_k$ where $z = \langle j, k \rangle$}

$\boxed{3}$ copy ith item of list in $\mathtt{A}$ to $\mathtt{R}$; goto $\boxed{4}$

$\boxed{4}$ execute current instruction on $\mathtt{R}$; update $\mathtt{PC}$ to next label; restore register values to $\mathtt{A}$; goto $\boxed{1}$

Overall structure of U's program

$\boxed{1}$ copy $\mathtt{PC}$th item of list in $\mathtt{P}$ to $\mathtt{N}$ (halting if $\mathtt{PC} >$ length of list); goto $\boxed{2}$

$\boxed{2}$ if $\mathtt{N} = \mathbf{0}$ then copy $\mathbf{0}$th item of list in $\mathtt{A}$ to $\mathtt{R}_0$ and halt, else (decode $\mathtt{N}$ as $\langle\!\langle y, z \rangle\!\rangle$; $\mathtt{C} ::= y$; $\mathtt{N} ::= z$; goto $\boxed{3}$)

{at this point either $\mathtt{C} = 2i$ is even and current instruction is $\mathtt{R}_i^+ \to \mathtt{L}_z$,
or $\mathtt{C} = 2i + 1$ is odd and current instruction is $\mathtt{R}_i^- \to \mathtt{L}_j, \mathtt{L}_k$ where $z = \langle j, k \rangle$}

$\boxed{3}$ copy ith item of list in $\mathtt{A}$ to $\mathtt{R}$; goto $\boxed{4}$

$\boxed{4}$ execute current instruction on $\mathtt{R}$; update $\mathtt{PC}$ to next label; restore register values to $\mathtt{A}$; goto $\boxed{1}$

To implement this, we need RMs for manipulating (codes of) lists of numbers...