Register machines

Algorithms, informally

No precise definition of "algorithm" at the time Hilbert posed the *Entscheidungsproblem*, just examples.

Common features of the examples:

- finite description of the procedure in terms of elementary operations
- deterministic (next step uniquely determined if there is one)
- procedure may not terminate on some input data, but we can recognize when it does terminate and what the result is.

Register Machines, informally

They operate on natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ stored in (idealized) registers using the following "elementary operations":

- add $\mathbf{1}$ to the contents of a register
- test whether the contents of a register is $\mathbf{0}$
- subtract $\mathbf{1}$ from the contents of a register if it is non-zero
- jumps ("goto")
- conditionals ("if_then_else_")

Definition. A register machine is specified by:

- 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 $\mathtt{L}_i$.

Definition. A register machine is specified by:

- 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 $\boldsymbol{label} : \boldsymbol{body}$, 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''
HALT	stop executing instructions

Example

registers:

$R_0 \ R_1 \ R_2$

program:

$L_0 : R_1^- \to L_1, L_2$

$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : \mathrm{HALT}$

example computation:

L_i	R_0	R_1	R_2
0	**0**	**1**	**2**

Example

registers:

$R_0\ R_1\ R_2$

program:

$L_0 : R_1^- \to L_1, L_2$

$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : \mathtt{HALT}$

example computation:

L_i	R_0	R_1	R_2
0	**0**	**1**	**2**

Example

registers:

R_0 R_1 R_2

program:

$L_0 : R_1^- \to L_1, L_2$

$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : \mathrm{HALT}$

example computation:

L_i	R_0	R_1	R_2
0	0	1	2
1	0	0	2

Example

registers:

R_0 R_1 R_2

program:

$L_0 : R_1^- \to L_1, L_2$

$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 :$ HALT

example computation:

L_i	R_0	R_1	R_2
0	0	1	2
1	0	0	2
0	1	0	2

Example

registers:

$R_0\ R_1\ R_2$

program:

$L_0 : R_1^- \to L_1, L_2$

$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : \mathtt{HALT}$

example computation:

L_i	R_0	R_1	R_2
0	**0**	**1**	**2**
1	**0**	**0**	**2**
0	**1**	**0**	**2**
2	**1**	**0**	**2**

Example

registers:

R_0 R_1 R_2

program:

$L_0 : R_1^- \to L_1, L_2$

$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : \mathtt{HALT}$

example computation:

L_i	R_0	R_1	R_2
0	0	1	2
1	0	0	2
0	1	0	2
2	1	0	2
3	1	0	1

Example

registers:

$R_0\ R_1\ R_2$

program:

$L_0 : R_1^- \to L_1, L_2$

$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : \mathtt{HALT}$

example computation:

L_i	R_0	R_1	R_2
0	0	1	2
1	0	0	2
0	1	0	2
2	1	0	2
3	1	0	1
2	2	0	1

Example

registers:

R_0 R_1 R_2

program:

$L_0 : R_1^- \to L_1, L_2$

$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : \mathtt{HALT}$

example computation:

L_i	R_0	R_1	R_2
0	**0**	**1**	**2**
1	**0**	**0**	**2**
0	**1**	**0**	**2**
2	**1**	**0**	**2**
3	**1**	**0**	**1**
2	**2**	**0**	**1**
3	**2**	**0**	**0**

Example

registers:

$R_0 \; R_1 \; R_2$

program:

$L_0 : R_1^- \to L_1, L_2$

$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : \mathtt{HALT}$

example computation:

L_i	R_0	R_1	R_2
0	0	1	2
1	0	0	2
0	1	0	2
2	1	0	2
3	1	0	1
2	2	0	1
3	2	0	0
2	3	0	0

Example

registers:

R_0 R_1 R_2

program:

$L_0 : R_1^- \to L_1, L_2$

$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : \mathtt{HALT}$

example computation:

L_i	R_0	R_1	R_2
0	0	1	2
1	0	0	2
0	1	0	2
2	1	0	2
3	1	0	1
2	2	0	1
3	2	0	0
2	3	0	0
4	3	0	0

Register machine computation

Register machine configuration:

$$c = (\ell, r_0, \ldots, r_n)$$

where ℓ = current label and r_i = current contents of $\mathtt{R}_i$.

Notation: "$\mathtt{R}_i = x$ [in configuration c]" means $c = (\ell, r_0, \ldots, r_n)$ with $r_i = x$.

Register machine computation

Register machine configuration:

$$c = (\ell, r_0, \ldots, r_n)$$

where ℓ = current label and r_i = current contents of $\mathtt{R}_i$.

Notation: "$\mathtt{R}_i = x$ [in configuration c]" means $c = (\ell, r_0, \ldots, r_n)$ with $r_i = x$.

Initial configurations:

$$c_0 = (0, r_0, \ldots, r_n)$$

where r_i = initial contents of register $\mathtt{R}_i$.

Register machine computation

A computation of a RM is a (finite or infinite) sequence of configurations

$$c_0, c_1, c_2, \ldots$$

where

- $c_0 = (0, r_0, \ldots, r_n)$ is an initial configuration
- each $c = (\ell, r_0, \ldots, r_n)$ in the sequence determines the next configuration in the sequence (if any) by carrying out the program instruction labelled $\mathtt{L}_\ell$ with registers containing r_0,. . . ,r_n.

Halting

For a finite computation $c_0, c_1, \ldots, c_m$, the last configuration $c_m = (\ell, r, \ldots)$ must be a halting configuration, i.e. ℓ must satisfy:

either ℓ^{th} instruction in program has body `HALT` (a "proper halt")

or ℓ is greater than the number of instructions in program, so that there is no instruction labelled $\mathtt{L}_\ell$ (an "erroneous halt")

Halting

For a finite computation $c_0, c_1, \ldots, c_m$, the last configuration $c_m = (\ell, r, \ldots)$ must be a halting configuration, i.e. ℓ must satisfy:

- either ℓ^{th} instruction in program has body $\mathtt{HALT}$ (a "proper halt")
- or ℓ is greater than the number of instructions in program, so that there is no instruction labelled L_ℓ (an "erroneous halt")

E.g. $\boxed{\begin{array}{l} L_0 : R_0^+ \to L_2 \\ L_1 : \mathtt{HALT} \end{array}}$ halts erroneously.

Halting

For a finite computation $c_0, c_1, \ldots, c_m$, the last configuration $c_m = (\ell, r, \ldots)$ must be a halting configuration, i.e. ℓ must satisfy:

<u>either</u> ℓ^{th} instruction in program has body `HALT` (a "proper halt")

<u>or</u> ℓ is greater than the number of instructions in program, so that there is no instruction labelled $\mathtt{L}_\ell$ (an "erroneous halt")

N.B. can always modify programs (without affecting their computations) to turn all erroneous halts into proper halts by adding extra `HALT` instructions to the list with appropriate labels.

Halting

For a finite computation $c_0, c_1, \ldots, c_m$, the last configuration $c_m = (\ell, r, \ldots)$ must be a halting configuration.

Note that computations may never halt. For example,

$$\boxed{\begin{array}{l} L_0 : R_0^+ \to L_0 \\ L_1 : \mathtt{HALT} \end{array}}$$

only has infinite computation sequences

$$(0, r), (0, r+1), (0, r+2), \ldots$$

Graphical representation

- one node in the graph for each instruction
- arcs represent jumps between instructions
- lose sequential ordering of instructions—so need to indicate initial instruction with `START`.

instruction	representation
$R^{+} \to L$	$R^{+} \longrightarrow [L]$
$R^{-} \to L, L'$	$R^{-} \begin{array}{l} \nearrow [L] \\ \\ \searrow\!\!\!\!\rightarrow [L'] \end{array}$
$\mathtt{HALT}$	$\mathtt{HALT}$
$\mathtt{L_0}$	$\mathtt{START} \longrightarrow [\mathtt{L_0}]$

Example

registers:

$R_0\ R_1\ R_2$

program:

$L_0 : R_1^- \rightarrow L_1, L_2$

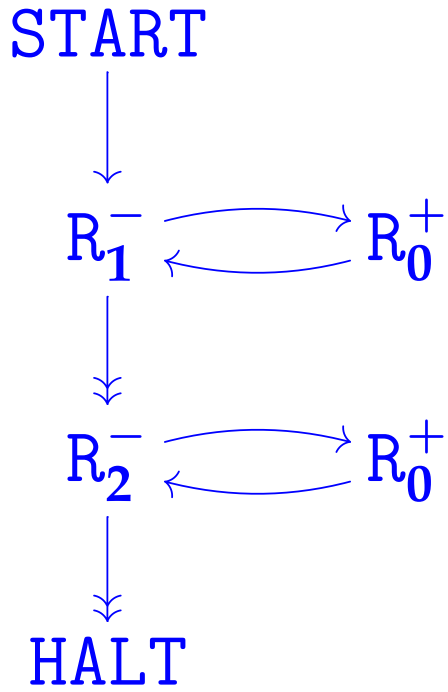
$L_1 : R_0^+ \rightarrow L_0$

$L_2 : R_2^- \rightarrow L_3, L_4$

$L_3 : R_0^+ \rightarrow L_2$

$L_4 : \mathtt{HALT}$

graphical representation:



Example

registers:

R_0 R_1 R_2

program:

$L_0 : R_1^- \to L_1, L_2$

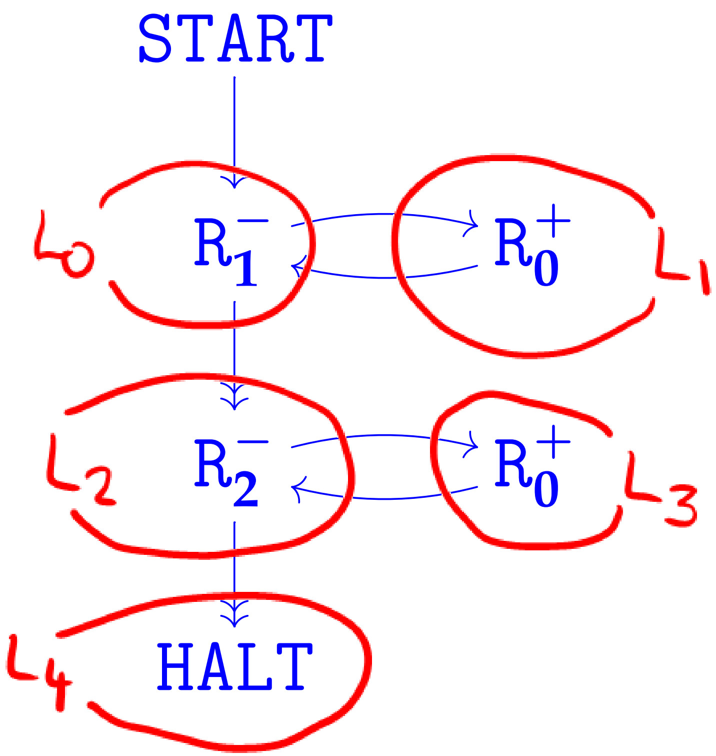
$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 :$ HALT

graphical representation:



Graphical representation is helpful for seeing what function a machine computes...

Example

registers:

$R_0 \; R_1 \; R_2$

program:

$$\begin{aligned}
L_0 &: R_1^- \to L_1, L_2 \\
L_1 &: R_0^+ \to L_0 \\
L_2 &: R_2^- \to L_3, L_4 \\
L_3 &: R_0^+ \to L_2 \\
L_4 &: \mathtt{HALT}
\end{aligned}$$

graphical representation:

$$\begin{tikzcd}
\mathtt{START} \arrow[d] & \\
R_1^- \arrow[r, bend left] \arrow[d, two heads] & R_0^+ \arrow[l, bend left] \\
R_2^- \arrow[r, bend left] \arrow[d, two heads] & R_0^+ \arrow[l, bend left] \\
\mathtt{HALT} &
\end{tikzcd}$$

Claim: starting from initial configuration $(0, 0, x, y)$, this machine's computation halts with configuration $(4, x + y, 0, 0)$.

Example

registers:

$R_0\ R_1\ R_2$

program:

$L_0 : R_1^- \to L_1, L_2$

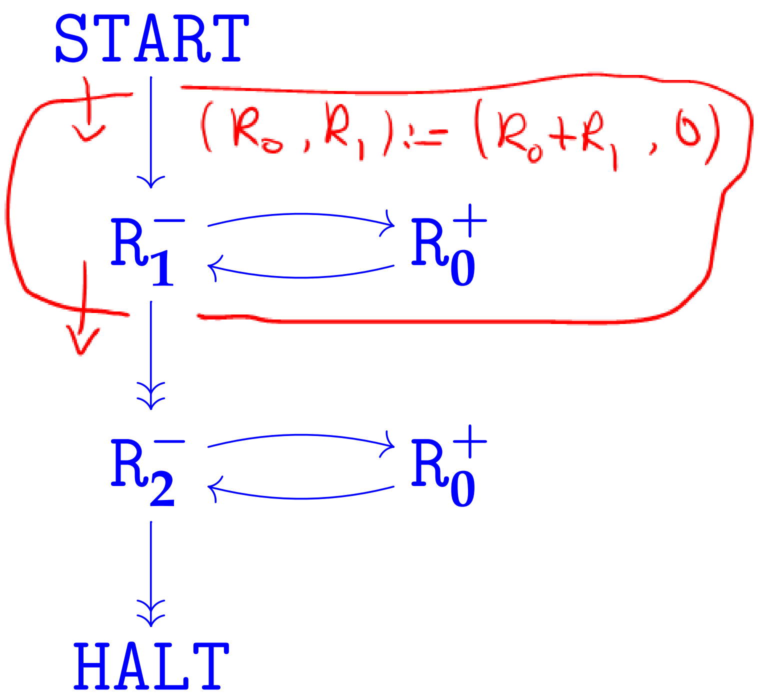
$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : \mathrm{HALT}$

graphical representation:



Claim: starting from initial configuration $(0, 0, x, y)$, this machine's computation halts with configuration $(4, x + y, 0, 0)$.

Example

registers:

$\mathtt{R_0}\ \mathtt{R_1}\ \mathtt{R_2}$

program:

$\mathtt{L_0} : \mathtt{R_1^-} \to \mathtt{L_1}, \mathtt{L_2}$

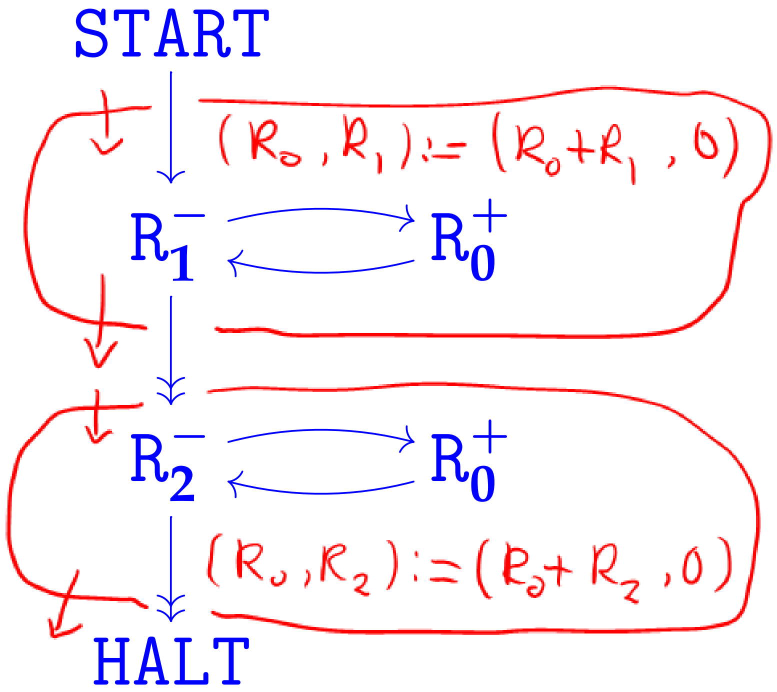
$\mathtt{L_1} : \mathtt{R_0^+} \to \mathtt{L_0}$

$\mathtt{L_2} : \mathtt{R_2^-} \to \mathtt{L_3}, \mathtt{L_4}$

$\mathtt{L_3} : \mathtt{R_0^+} \to \mathtt{L_2}$

$\mathtt{L_4} : \mathtt{HALT}$

graphical representation:



Claim: starting from initial configuration $(0, 0, x, y)$, this machine's computation halts with configuration $(4, x + y, 0, 0)$.

Partial functions

Register machine computation is deterministic: in any non-halting configuration, the next configuration is uniquely determined by the program.

So the relation between initial and final register contents defined by a register machine program is a partial function...

Partial functions

Register machine computation is deterministic: in any non-halting configuration, the next configuration is uniquely determined by the program.

So the relation between initial and final register contents defined by a register machine program is a partial function. . .

Definition. A partial function from a set X to a set Y is specified by any subset $f \subseteq X \times Y$ satisfying

$$(x, y) \in f \wedge (x, y') \in f \to y = y'$$

for all $x \in X$ and $y, y' \in Y$.

Partial functions

ordered pairs $\{(x, y) \mid x \in X \wedge y \in Y\}$

i.e. for all $x \in X$ there is at most one $y \in Y$ with $(x, y) \in f$

Definition. A partial function from a set X to a set Y is specified by any subset $f \subseteq X \times Y$ satisfying

$$(x, y) \in f \wedge (x, y') \in f \rightarrow y = y'$$

for all $x \in X$ and $y, y' \in Y$.

Partial functions

Notation:

- "$f(x) = y$" means $(x, y) \in f$
- "$f(x)\downarrow$" means $\exists y \in Y\,(f(x) = y)$
- "$f(x)\uparrow$" means $\neg\exists y \in Y\,(f(x) = y)$
- $X \rightharpoonup Y$ = set of all partial functions from X to Y
  $X \to Y$ = set of all (total) functions from X to Y

Definition. A partial function from a set X to a set Y is specified by any subset $f \subseteq X \times Y$ satisfying

$$(x, y) \in f \wedge (x, y') \in f \to y = y'$$

for all $x \in X$ and $y, y' \in Y$.

Partial functions

Notation:

- "$f(x) = y$" means $(x, y) \in f$
- "$f(x)\downarrow$" means $\exists y \in Y \, (f(x) = y)$
- "$f(x)\uparrow$" means $\neg\exists y \in Y \, (f(x) = y)$
- $X \rightharpoonup Y$ = set of all partial functions from X to Y
  $X \to Y$ = set of all (total) functions from X to Y

Definition. A partial function from a set X to a set Y is total if it satisfies

$$f(x)\downarrow$$

for all $x \in X$.

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 $R_0, R_1, \ldots, R_n$ (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 $R_0 = 0$, $R_1 = x_1, \ldots, R_n = x_n$ and all other registers set to 0, halts with $R_0 = y$

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

Note the [somewhat arbitrary] I/O convention: in the initial configuration registers $R_1, \ldots, R_n$ store the function's arguments (with all others zeroed); and in the halting configuration register R_0 stores it's value (if any).

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 $R_0, R_1, \ldots, R_n$ (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 $R_0 = 0$, $R_1 = x_1, \ldots, R_n = x_n$ 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.

Example

registers:

$\mathtt{R}_0 \ \mathtt{R}_1 \ \mathtt{R}_2$

program:

$\mathtt{L}_0 : \mathtt{R}_1^- \to \mathtt{L}_1, \mathtt{L}_2$

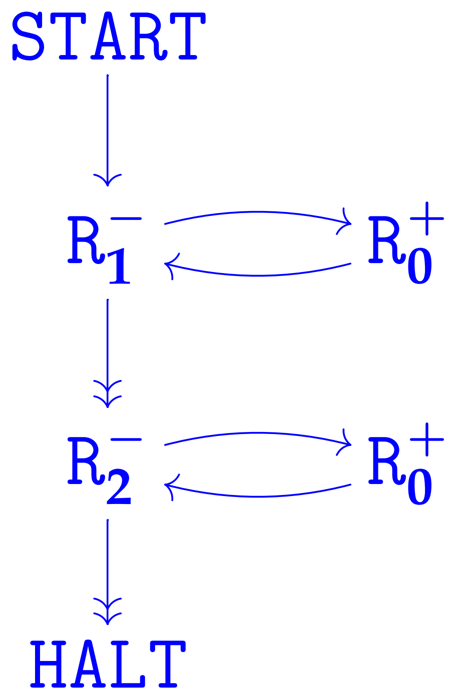
$\mathtt{L}_1 : \mathtt{R}_0^+ \to \mathtt{L}_0$

$\mathtt{L}_2 : \mathtt{R}_2^- \to \mathtt{L}_3, \mathtt{L}_4$

$\mathtt{L}_3 : \mathtt{R}_0^+ \to \mathtt{L}_2$

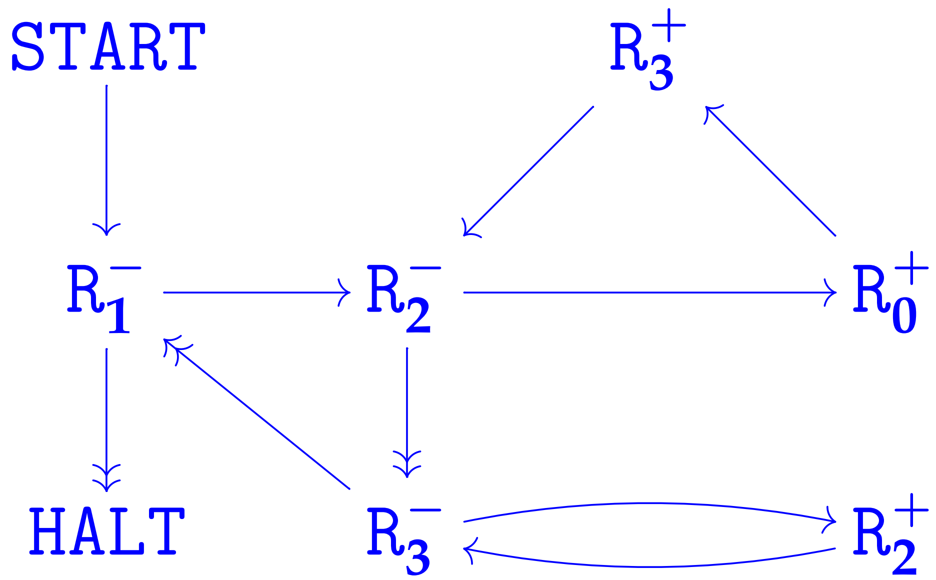
$\mathtt{L}_4 : \mathtt{HALT}$

graphical representation:

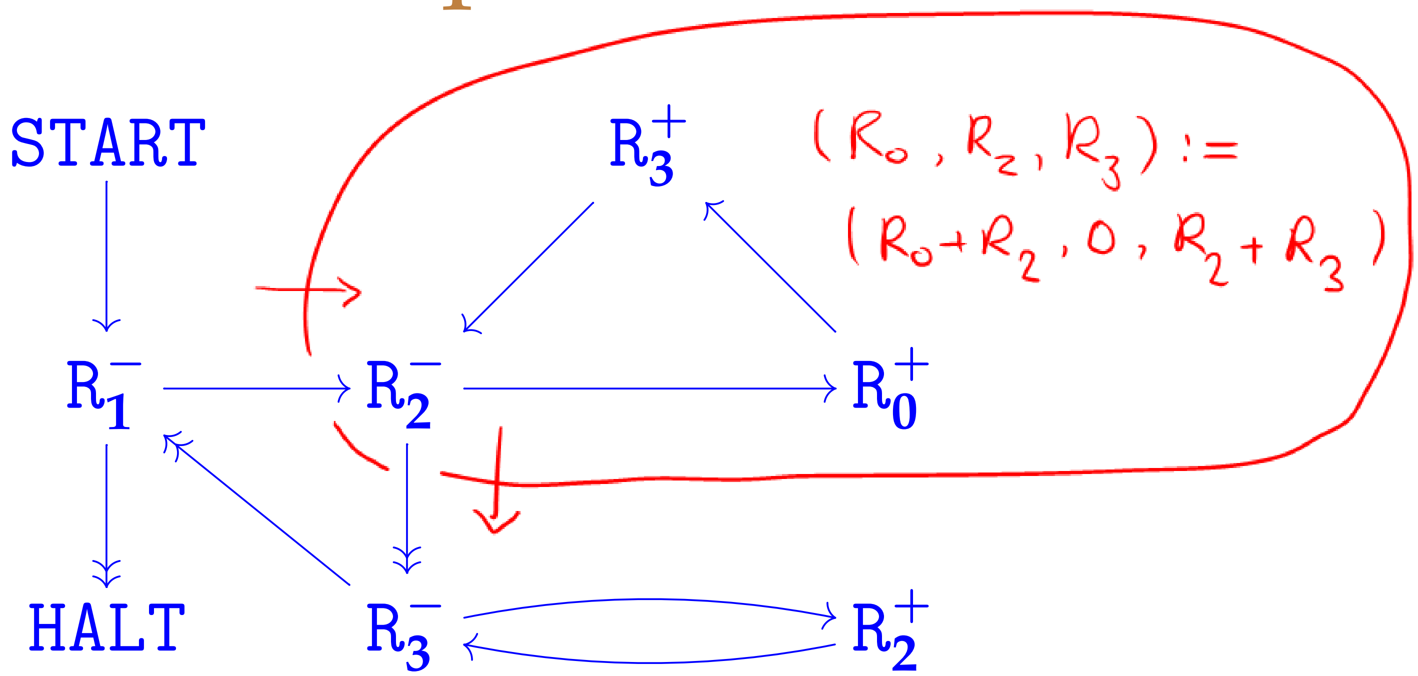


Claim: starting from initial configuration $(0, 0, x, y)$, this machine's computation halts with configuration $(4, x + y, 0, 0)$. So $f(x, y) \triangleq x + y$ is computable.

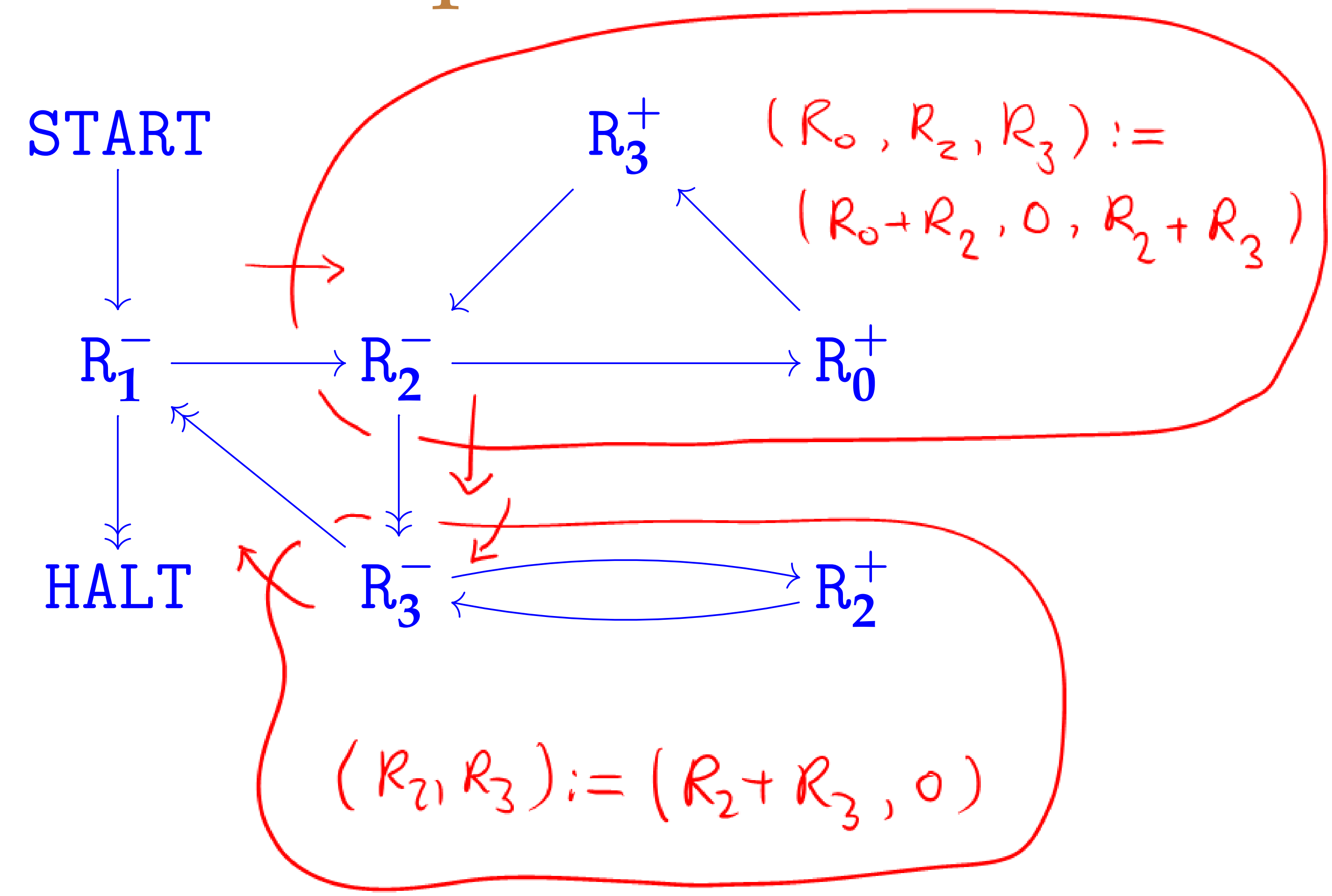
Multiplication $f(x, y) \triangleq xy$ is computable



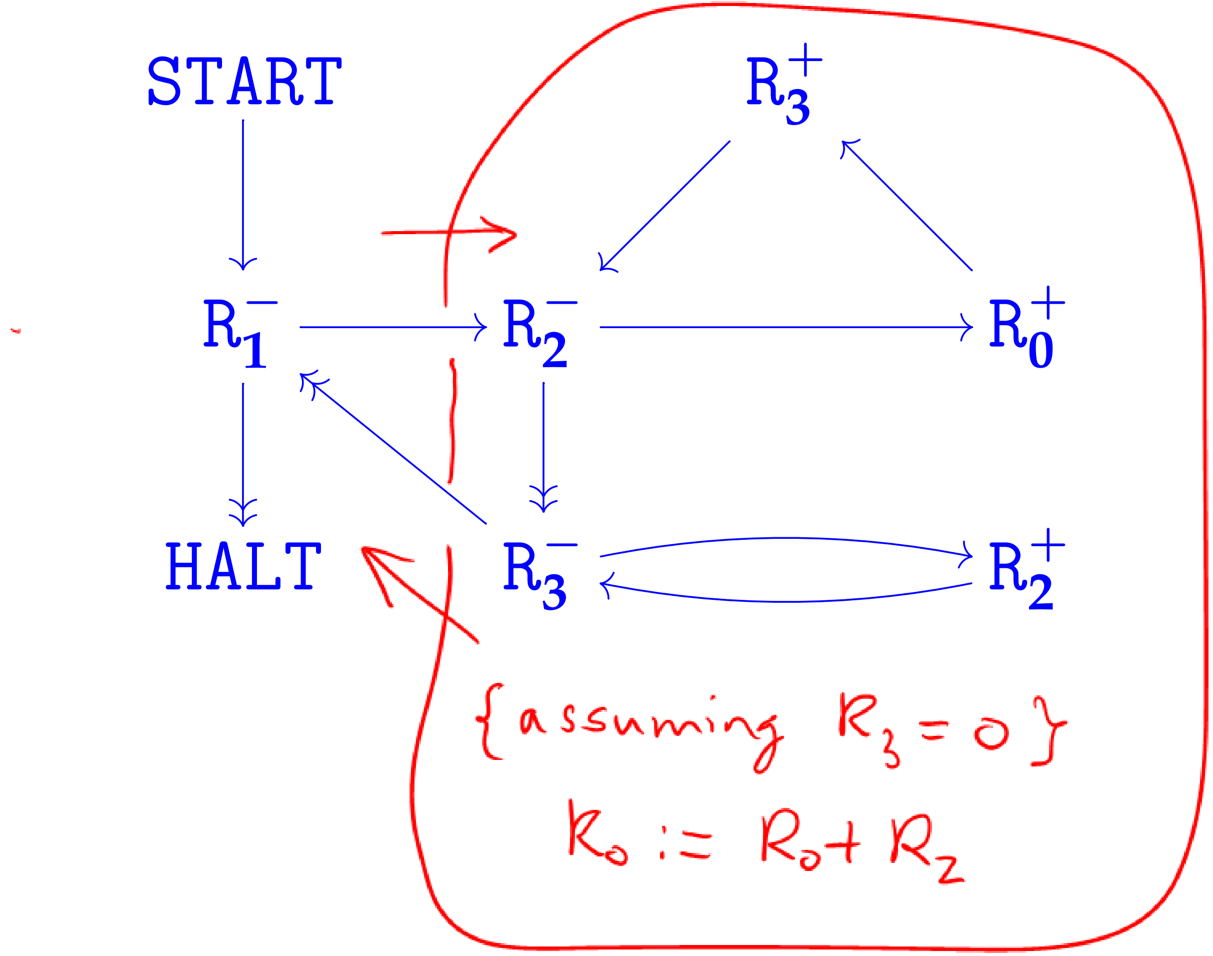
Multiplication $f(x,y) \triangleq xy$ is computable



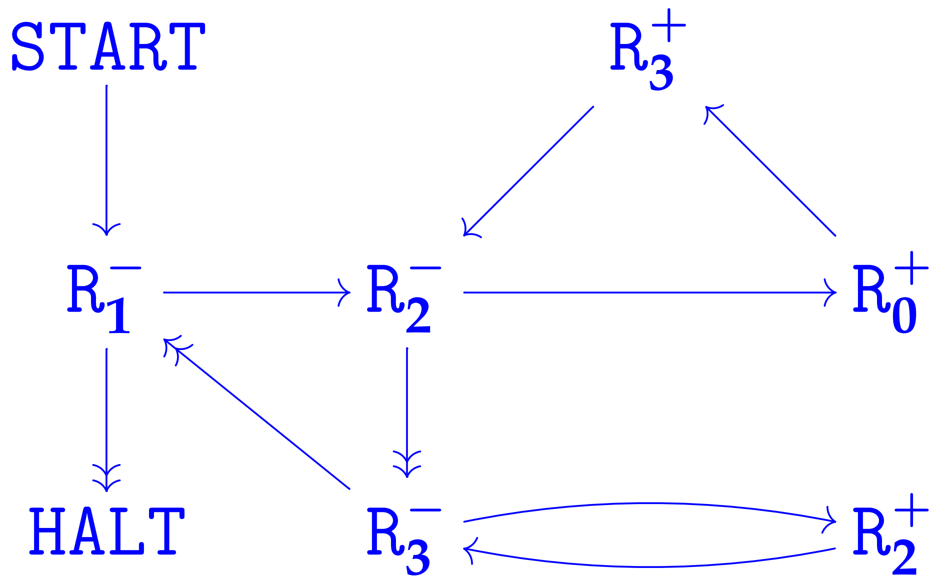
Multiplication $f(x, y) \triangleq xy$ is computable



Multiplication $f(x, y) \triangleq xy$ is computable



Multiplication $f(x,y) \triangleq xy$ is computable



If the machine is started with $(R_0, R_1, R_2, R_3) = (0, x, y, 0)$, it halts with $(R_0, R_1, R_2, R_3) = (xy, 0, y, 0)$.

Further examples

The following arithmetic functions are all computable.
(Proof—left as an exercise!)

projection: $p(x, y) \triangleq x$

constant: $c(x) \triangleq n$

truncated subtraction: $x \dot{-} y \triangleq \begin{cases} x - y & \text{if } y \leq x \\ 0 & \text{if } y > x \end{cases}$

Further examples

The following arithmetic functions are all computable.
(Proof—left as an exercise!)

integer division:

$$x \mathbin{div} y \triangleq \begin{cases} \textit{integer part of } x/y & \text{if } y > 0 \\ 0 & \text{if } y = 0 \end{cases}$$

integer remainder: $x \mathbin{mod} y \triangleq x \mathbin{\dot{-}} y(x \mathbin{div} y)$

exponentiation base 2: $e(x) \triangleq 2^x$

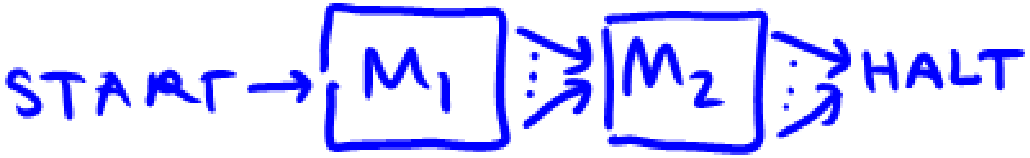
logarithm base 2:

$$\log_2(x) \triangleq \begin{cases} \textit{greatest } y \textit{ such that } 2^y \leq x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \end{cases}$$

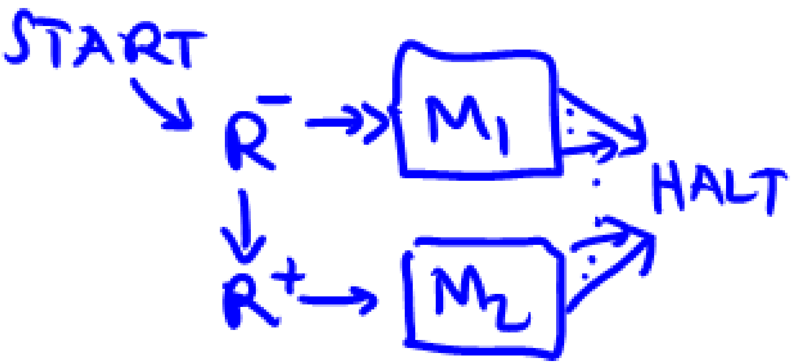
W.l.o.g. can use RMs with only one HALT

N.B. interference

Sequential composition $M_1 ; M_2$



IF $R = 0$ THEN M_1 ELSE M_2



WHILE $R \neq 0$ DO M