2 Computability theory

2.1 Intuitive idea of computability and Church Thesis

The computability main effect

The surest recipe non computer scientist to be horrified:

A hot debate over the right program language

- All program languages and machine models are „equally powerful“
- In every model there are the same non computable problems

2.2 The computability in the intuitive sense

$f : \mathbb{N}^k \longrightarrow \mathbb{N}$ is called(partial) function.

f is computable if

$\exists$ an effective procedure($=$algorithm) which computes f.

effective procedure $=$ Java-program (, ..., “appropriate” program language)

Input: $(x_1, \ldots, x_k) \in \mathbb{N}^k$

Output: $f(x_1, \ldots, x_k)$

program halts in finitely many steps in case of $(x_1, \ldots, x_k) \in$ domain of f.

infinite loop otherwise.

Example

input n

repeat

until false

computes the total not defined function Ω

Example

$$f_\pi(n) = \begin{cases} 1 & \text{if } n \text{ is an initial segment in the decimal representation of } \pi. \\ 0 & \text{otherwise} \end{cases}$$

f_π is computable:

Use large number arithmetic, and apply an appropriate approximate computation “large enough”.

Excursus: some approximations of π

Archimedes: Approximation by right polygons.

Ancient Indian: 1682 of Leibniz rediscovered

$$\pi = 4\sum_{i=0}^{\infty} \frac{-1^i}{2i+1} = 1 - \frac{1}{3} + \frac{1}{5} - \cdots$$

("appropriate stop condition")

Baile-Borwein-Plouffe 1996:

$$\pi = \sum_{i=0}^{\infty} \frac{1}{16^i}\left(\frac{4}{8i-1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6}\right)$$

Example

$$f(n) = \begin{cases} 1 & \text{if } n \text{ appears somewhere in the decimal representation of } \pi. \\ 0 & \text{otherwise} \end{cases}$$

It is unknown if f is computable!

It is not known till now $\forall n : f(n) = 1$ (“normality” of π).

Example

$$f(n) = \begin{cases} 1 & \text{if } n \times `7' \text{ appears somewhere in the decimal representation of } \pi \\ 0 & \text{otherwise} \end{cases}$$

Is f computable ?

Example

$$f(n) = \begin{cases} 1 & \text{if } n \times `7' \text{ appears somewhere in the decimal representation of } \pi \\ 0 & \text{otherwise} \end{cases}$$

Is f computable ? Yes !

If $\forall n : n \times `7'$ occurs: $\forall n : f(n) = 1$

If $`7'$ occurs maximum n_0 times somewhere:

$$\longrightarrow f(n) = \begin{cases} 1 & \text{if } n \leq n_0 \\ 0 & \text{otherwise} \end{cases}$$

Also: computability $\neq$ we know the function definitive specified !

Example

LBA: linear bounded Turing machine

DLBA: deterministic linear bounded Turing machine

$$i(n) = \begin{cases} 1 & \text{if } \forall LBA\, M \exists DLBA\, D : L(M) = L(D) \\ 0 & \text{otherwise} \end{cases}$$

We don't know the function $i(n)$.

But, $i(n)$ is a constant function and hence obviously computable.

Non computable functions

Let $r \in \mathbb{R}$ be arbitrary.

$$f_r(n) = \begin{cases} 1 & \text{if } n \text{ is an initial segment in the decimal representation of } r. \\ 0 & \text{otherwise} \end{cases}$$

Assume: $\forall r : f_r$ is computable.

$\longrightarrow \exists$ at least as much computable functions as the real numbers.

A contradiction:

- The set of all computable functions is countable
  (since it is described by finitely long text).
- $\mathbb{R}$ is not countable.

Non computable functions

There are some. But could we write out one concrete?

todo

Church thesis

Functions computable by Turing machine

are exactly those computable in the intuitive sense.

Not a proposition but everybody accepted.

The reasons

- All known computable models are weaker or equivalent.

  this we can prove

- All „intuitive“ computable known function are Turing-computable.

Turing machines compute functions

$T = (Q, \Sigma, \Gamma, \delta, s, F)$ computes the partial function $f_T : \Sigma^* \to \Gamma^* \Leftrightarrow$

$$f_T(w) := \begin{cases} v & \text{if } T \text{ halts by input of } w \text{ with output } v \\ & ((s)w \Rightarrow u(q)v), q \in F \\ \perp = \text{(not defined)} & \text{otherwise} \end{cases}$$

g is Turing computable $\Leftrightarrow \exists T : f_T = g$

Remark: when $g(x) = \perp$, T does not halt.

Turing machines compute numerical functions

$f:\mathbb{N}^k\to\mathbb{N}$ is Turing computable $\Leftrightarrow$

$$\exists T=(Q,\Sigma,\Gamma,\delta,s,F):\forall n_1,\ldots,n_k,m\in\mathbb{N}:$$
$$f(n_1,\ldots,n_k)=m\Leftrightarrow$$
$$(s)\mathrm{bin}(n_1)\#\cdots\#\mathrm{bin}(n_k)\vdash^* u(q)\mathrm{bin}(m),q\in F$$

Acceptance $\rightsquigarrow$ function

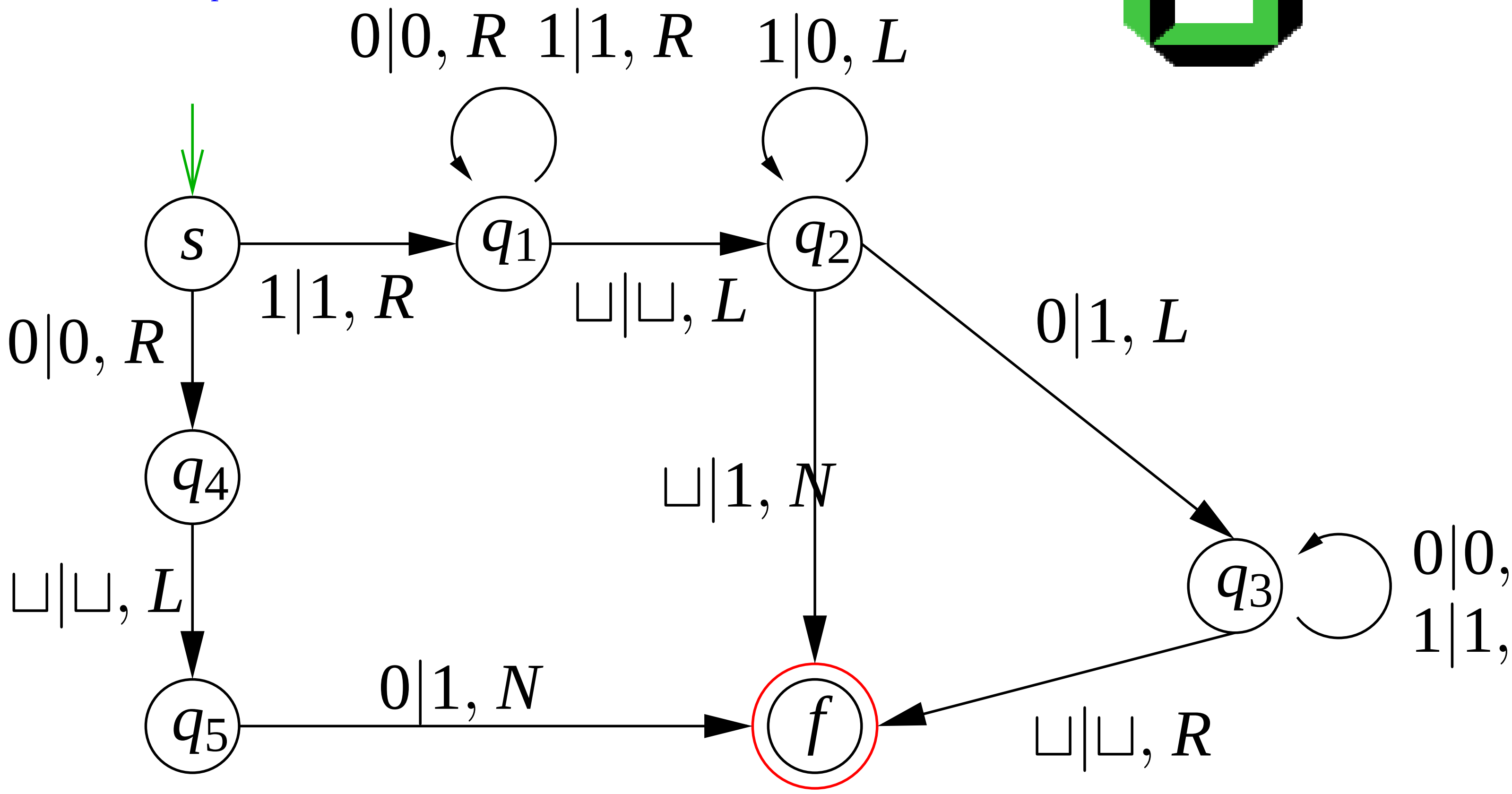
Computability of a function is the main idea.

Instead of acceptor for $L \subseteq \Sigma^*$ consider TM, which computes a „half“ characteristic function

$$\chi'_L(w) = \begin{cases} 1 & \text{if } w \in L \\ \perp & \text{otherwise} \end{cases}$$

But as we said, all „interesting“ could we perform as acceptors.

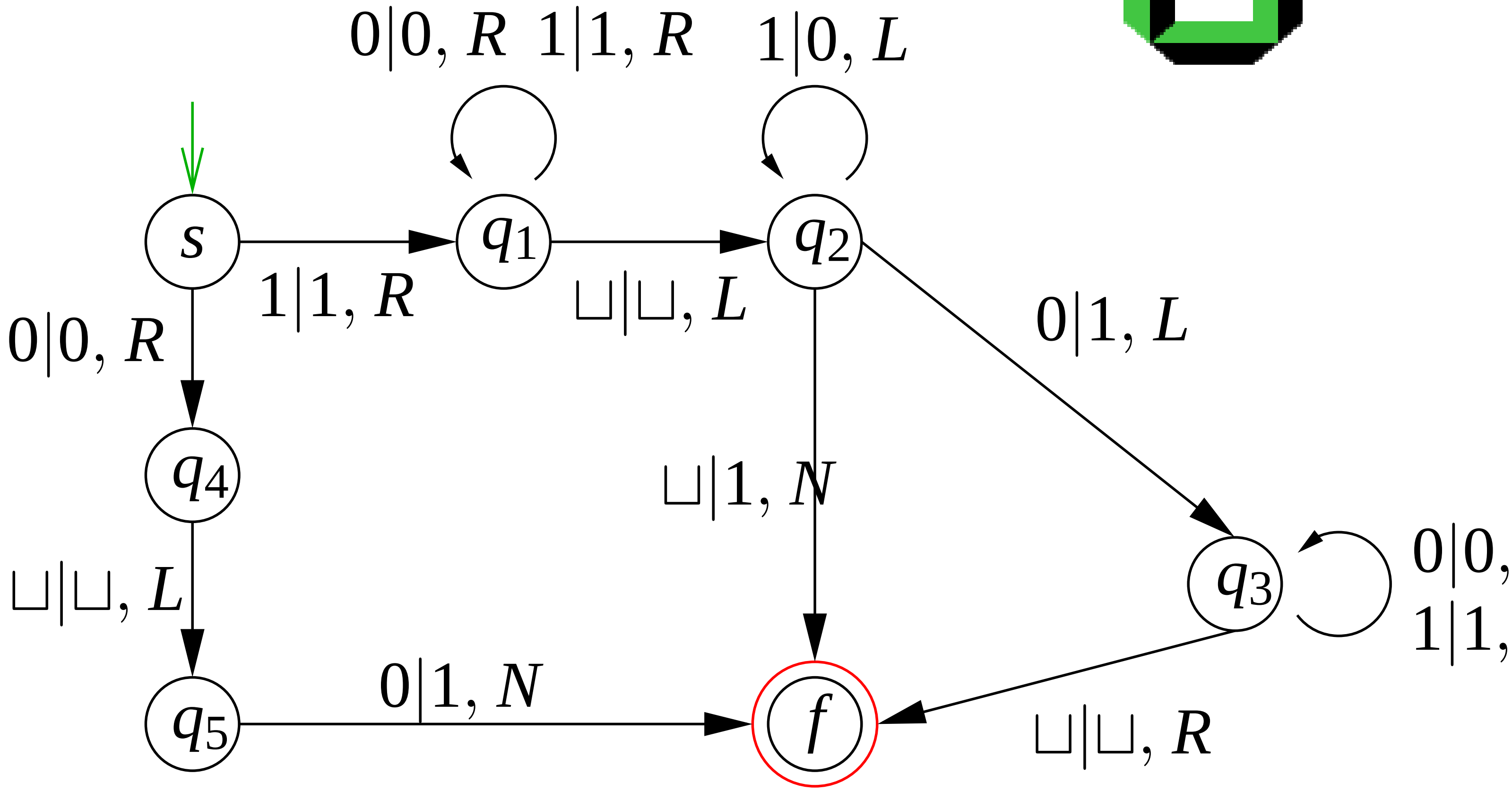
Example



$$f(w) = \begin{cases} w+1 & \text{if } w \in 0 \cup 1(0 \cup 1)^*, \\ & w \text{ interpreted as binary number} \\ \text{undefined} & \text{otherwise} \end{cases}$$

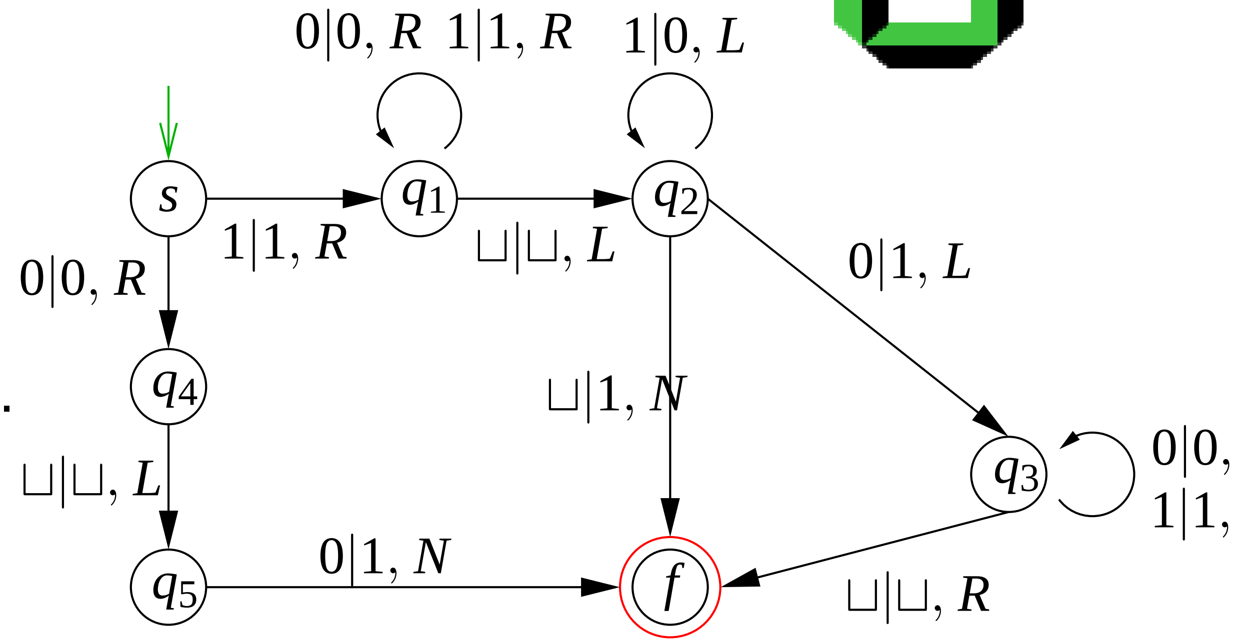
Remark: Not displayed movement are valid here as infinite loops.

Example



$$(s)11 \vdash 1(q_1)1 \vdash 11(q_1) \vdash 1(q_2)1 \vdash (q_2)10 \vdash (q_2)\sqcup 00 \vdash (f)100$$

Functionality



definition by cases

on the structure of the input.

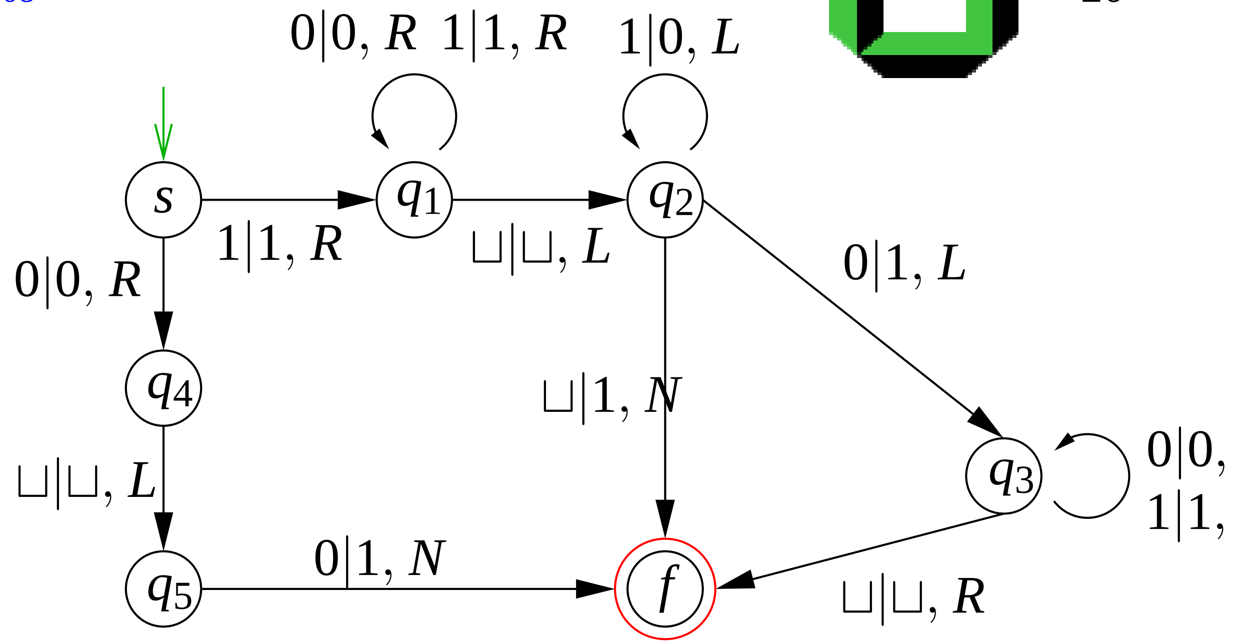
Let $w \in \{0,1\}^*$,

$a \in \{0,1\}$,$n \geq 1$.

0: $(s)0 \vdash 0(q_4) \vdash (q_5)0 \vdash (f)1$

$0aw$: $(s)0aw \vdash 0(q_4)aw$ not defined

Functionality



Definition by cases

on the structure of input.

Let $w \in \{0,1\}^*$,

$a \in \{0,1\}$,$n \geq 1$.

1^n: $(s)1^n \vdash 1(q_1)1^{n-1} \overset{n-1}{\vdash} 1^n(q_1) \vdash 1^{n-1}(q_2)1 \overset{n}{\vdash} (q_2)\sqcup 0^n \vdash (f)10^n$

$1w0$: $(s)1w0 \vdash 1(q_1)w0 \overset{|w|+1}{\vdash} 1w0(q_1) \vdash 1w(q_2)0 \vdash 1w(q_3)1 \overset{|w|+1}{\vdash}$
$(q_3)\sqcup 1w1 \vdash (f)1w1$

$1w01^n$:

$(s)1w01^n \vdash 1(q_1)w01^n \overset{|w|+1+n}{\vdash} 1w01^n(q_1) \vdash 1w01^{n-1}(q_2)1 \overset{n}{\vdash}$

$1w(q_2)00^n \vdash 1w(q_3)10^n \overset{|w|+1}{\vdash} (q_3)\sqcup 1w10^n \vdash (f)1w10^n$

Example: The overall not defined function

$$T = (\{s\}, \Sigma, \Gamma, \delta, s, \{\})$$

$$\forall a \in \Gamma : \delta(s, a) = (s, a, R)$$

Program technics for Turing machines

- Local variables
- Serial shifts
- Traces
- While-loops

Local Variables

Local variable accumulates $x \in A$, ($|A| < \infty$!):

$Q \rightsquigarrow Q \times A$

Example: M is TM, such that memorizes the first symbol of the word and halts if it is not in another place in the word.

$$\delta([s,\sqcup],0) = ([q,0],0,R) \quad \delta([s,\sqcup],1) = ([q,1],1,R)$$

$$\delta([q,0],1) = ([q,0],1,R) \quad \delta([q,1],0) = ([q,1],0,R)$$

$$\delta([q,0],\sqcup) = (f,\sqcup,N) \quad \delta([q,1],\sqcup) = (f,\sqcup,N)$$

Serial shifts

Given: $T = (Q, \Sigma, \Gamma, \delta, s, F)$

Let: $(s)w \vdash^* (r) f_T(w)$ for one $r \in F$ if $f_T(w) \neq \perp$.,

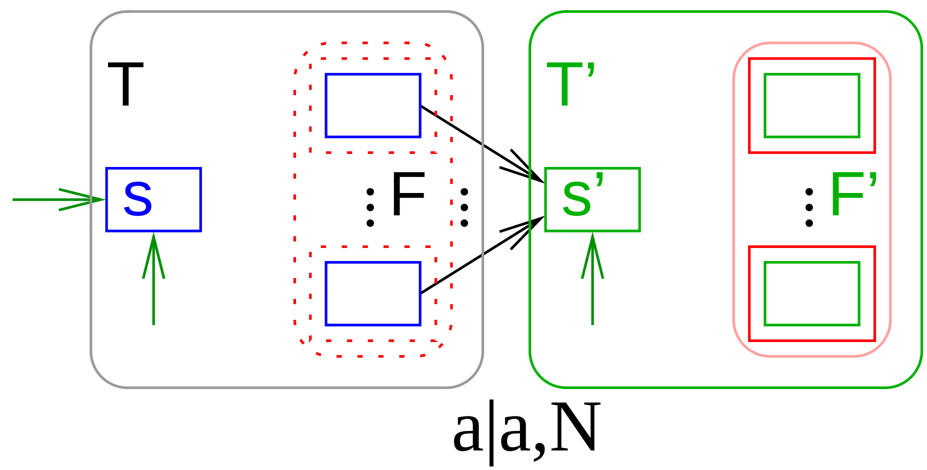
$T' = (Q', \Sigma, \Gamma', \delta', s', F')$.

output: Turing machine $T^\circ = (Q^\circ, \Sigma, \Gamma^\circ, \delta^\circ, s, F')$ for $f_{T'}(f_T(x))$:

$Q^\circ = Q \dot\cup Q'$

$\Gamma^\circ = \Gamma \cup \Gamma'$

$$\delta^\circ(q,a) = \begin{cases} \delta(q,a) & \text{if } q \in Q \setminus F \\ (s', a, N) & \text{if } q \in F \\ \delta'(q,a) & \text{if } q \in Q' \end{cases}$$



if then else

Given: $T=(Q,\Sigma,\Gamma,\delta,s,F)$, $T'=(Q',\Sigma,\Gamma',\delta',s',F')$
$T''=(Q'',\Sigma,\Gamma'',\delta'',s'',F'')$.

Output: Turing machine $T^\circ=(Q^\circ,\Sigma,\Gamma^\circ,\delta^\circ,s,F'\cup F'')$

$Q^\circ=Q\,\dot\cup\,Q'\,\dot\cup\,Q''$, $\Gamma^\circ=\Gamma\cup\Gamma'\cup\Gamma''$

$$f_{T^\circ}(x)=\begin{cases} f_{T'}(f_T(x)) & \text{if } f_T(x)=a \\ f_{T''}(f_T(x)) & \text{if } \downarrow f_T(x)\neq a \end{cases}.$$

$$\delta^\circ(q,b)=\begin{cases}\delta(q,b) & \text{if } q\in Q\setminus F\\ (s',b,N) & \text{if } q\in F \;\&\; b=a\\ (s'',b,N) & \text{if } q\in F \;\&\; b\neq a\\ \delta'(q,b) & \text{if } q\in Q'\\ \delta''(q,b) & \text{if } q\in Q''\end{cases}$$

Tracks

$\Gamma = \Gamma_1 \times \cdots \times \Gamma_k$.

Example: Arithmetical operations of 2 binary numbers,

Marking...

A little complication: the input alphabet will be modified.

Solution:

- $\Gamma = \Sigma \,\dot{\cup}\, \Gamma_1 \times \cdots \times \Gamma_k$
- replace $a \in \Sigma$ through $(a, 0, \ldots, 0)$ in the input.

While-loops: While $i \neq 0$ Do tape:= f_T(tape)

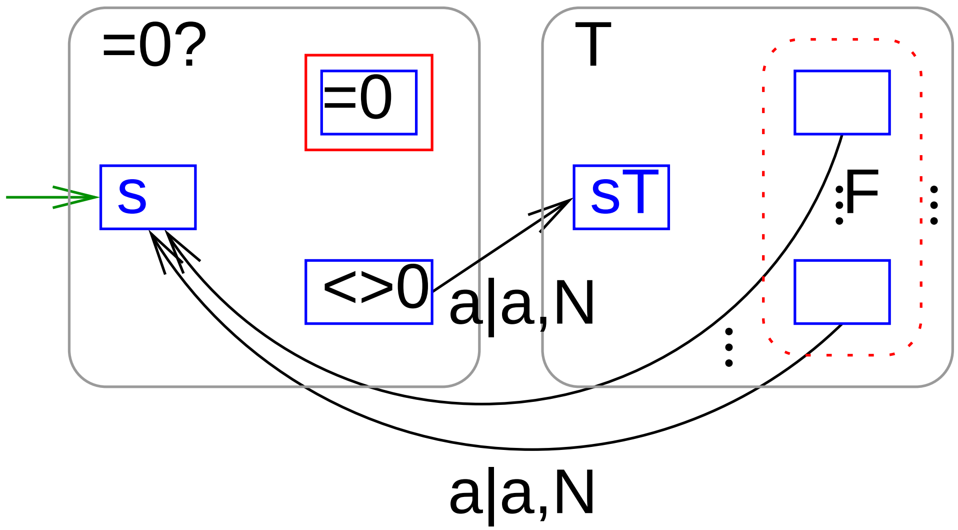
Track i defines a number (unary or binary)

Subprogram: test on track $i = 0$.

When yes: halt

Leave T moving

back to the start state. (the transition $\delta(f,a) = (s,a,N)$)



Examples:

$R_{\sqcup}$ - scans right until finding $\sqcup$

(the same $L_{\sqcup}$)

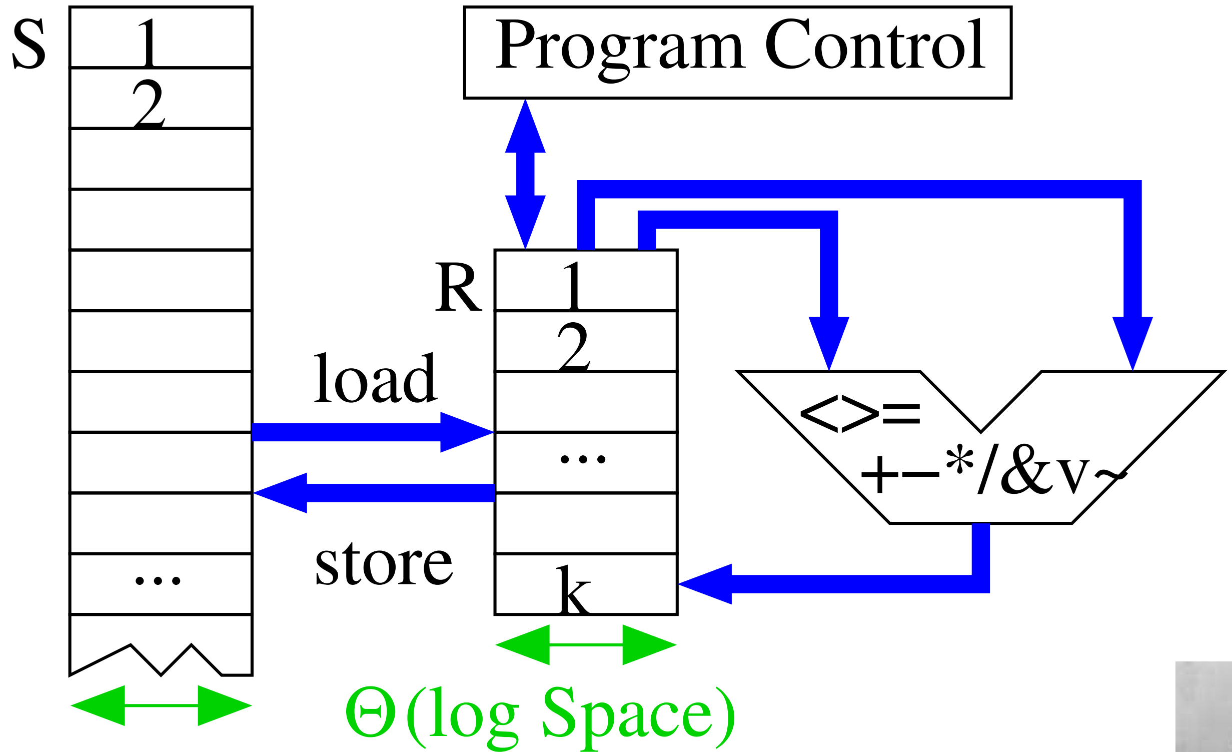
Copy: $(q)w\sqcup \vdash (f)w \sqcup w$ - copy the word

Shift R: $(q)\sqcup w\sqcup \vdash (f)w \sqcup w$ - move the word one position to the right.

2.3 LOOP-, WHILE-, GOTO (=Register machines) and RAM- computability

- More familiar computational models
- All up to one are Turing powerful

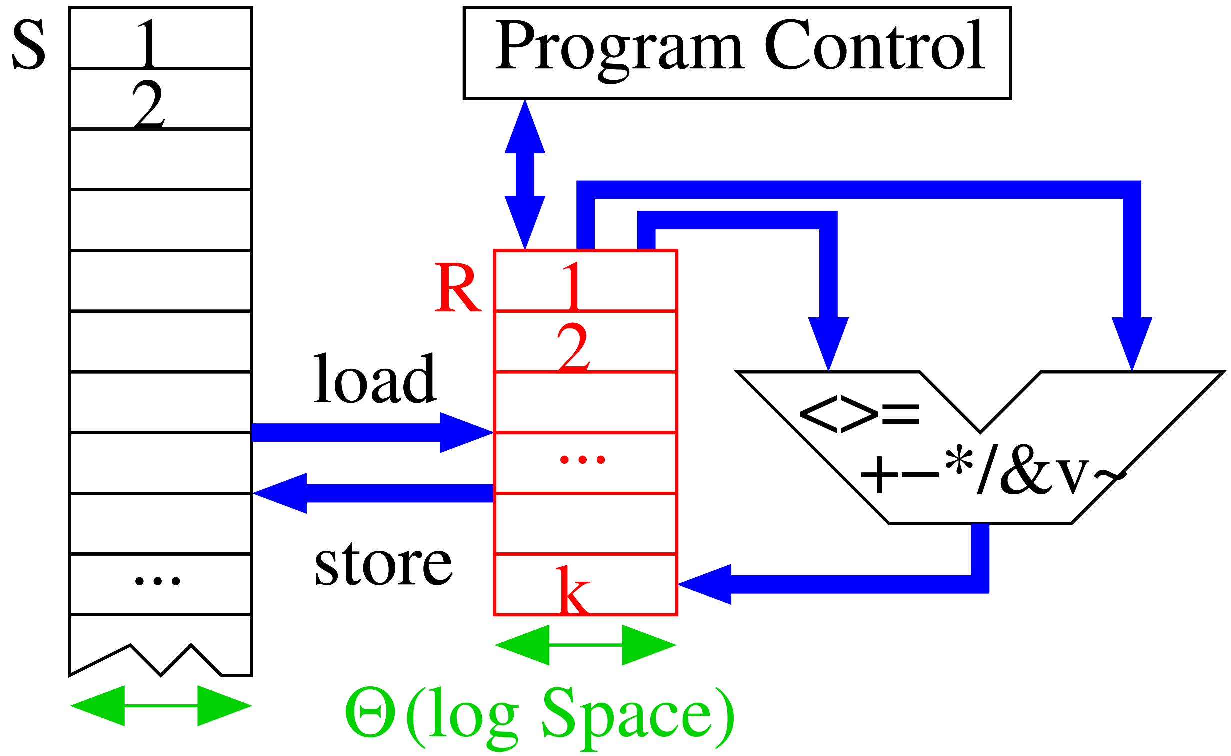
RAM: Random Access Machine



Modern (RISC) adaptation

of Neumann-models [of Neumann 1945]

Register

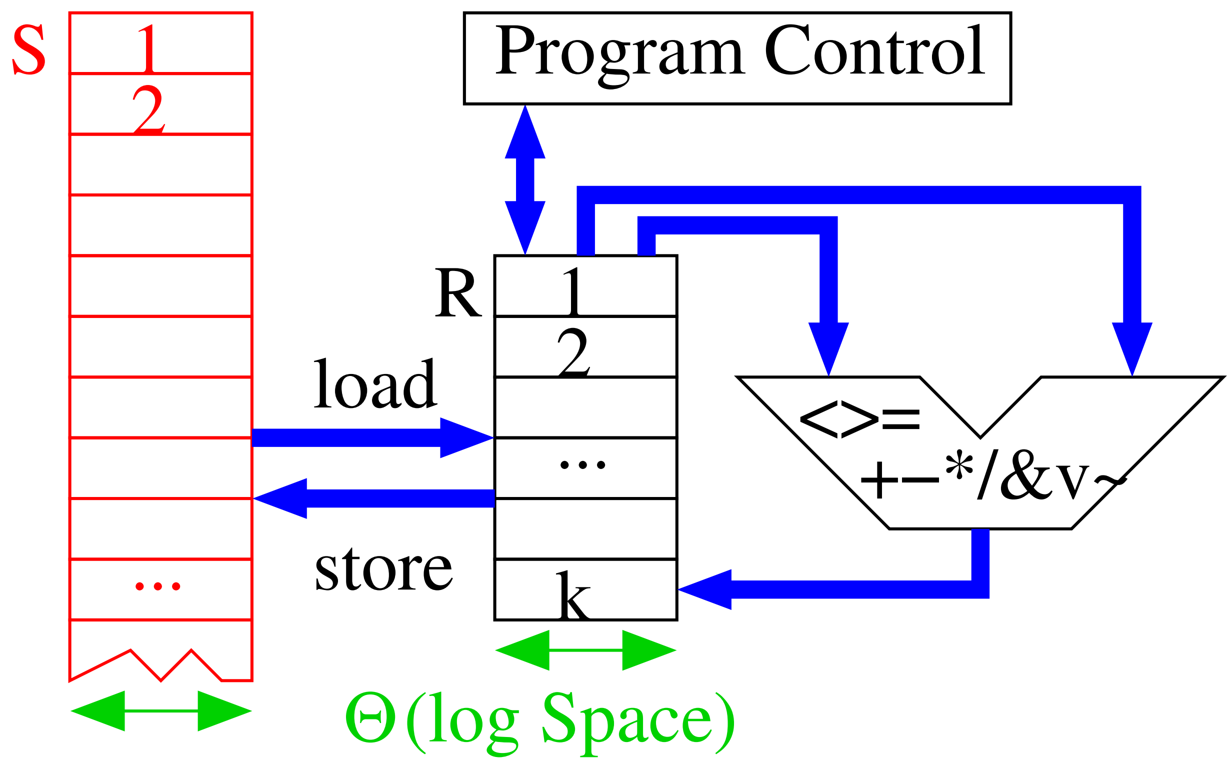


k (any constant) memory

$R_1, \ldots, R_k$ for

(small) integers

The main memory

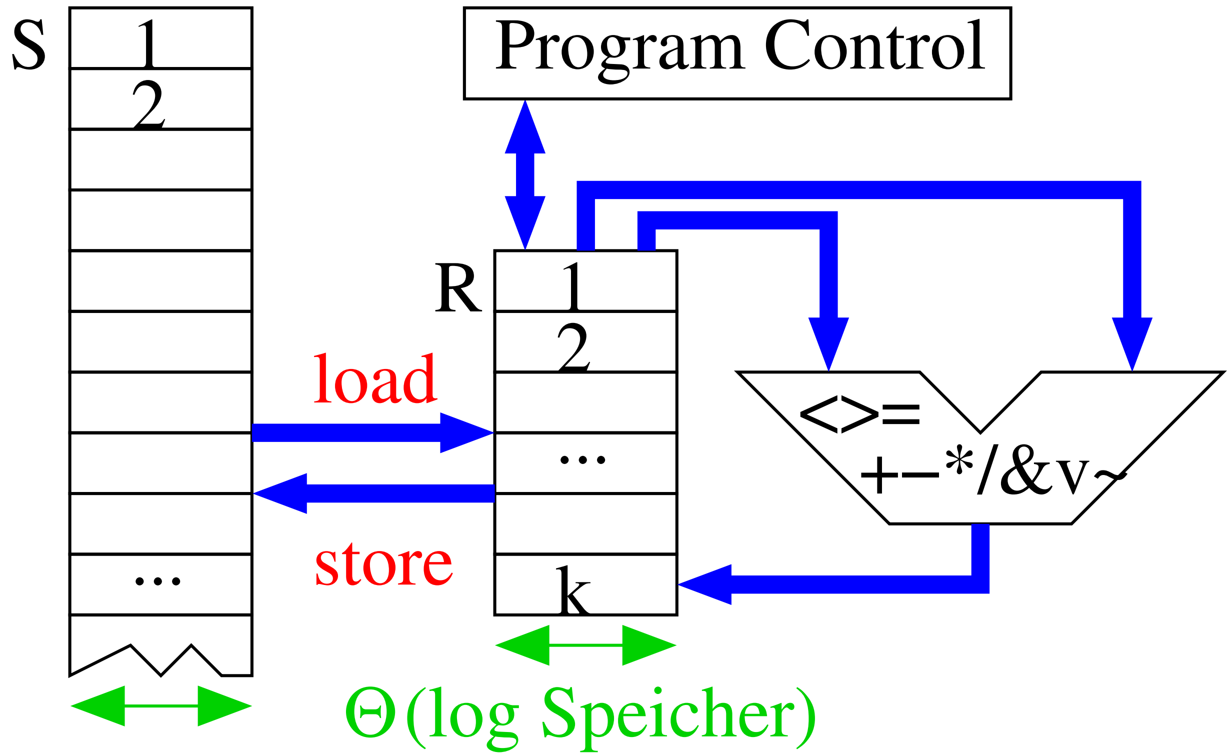


Non bounded supply of memory cells

$S[1], S[2]\ldots$ for

(small) integers

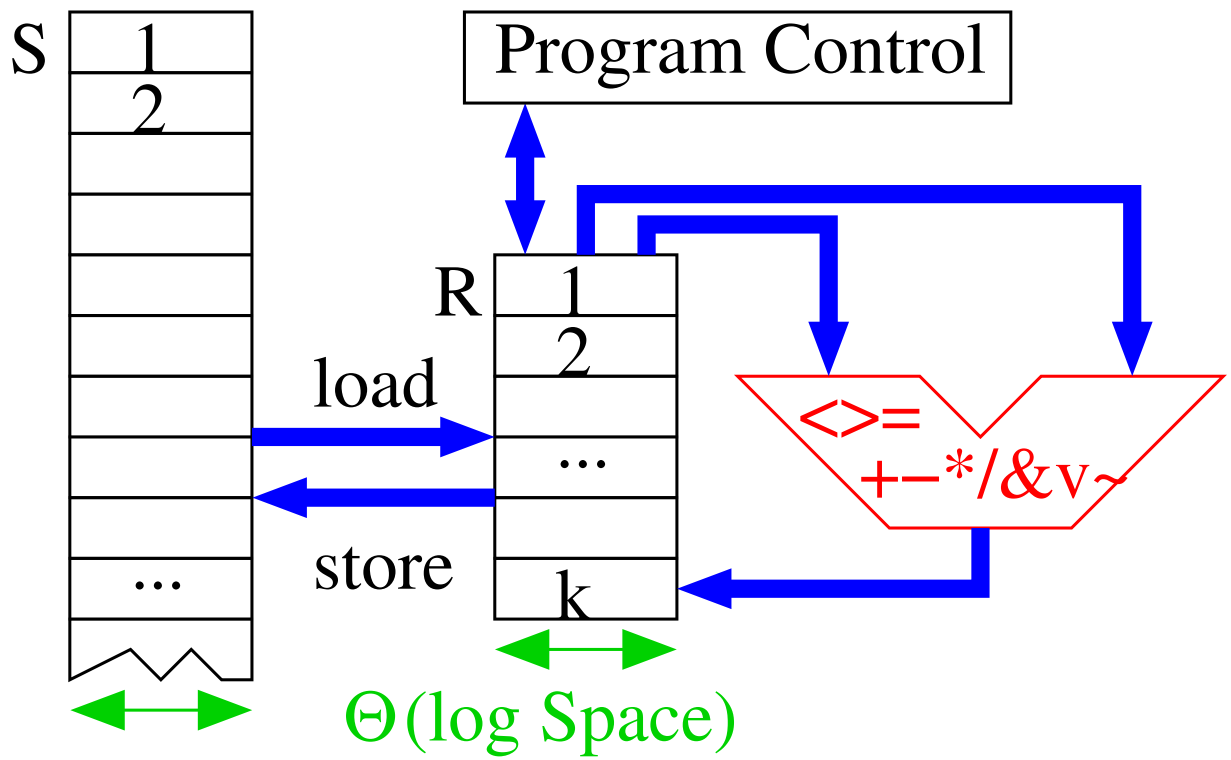
Memory access



$R_i := S[R_j]$ loads the content of the memory cell $S[R_j]$ in Register R_i.

$S[R_j] := R_i$ stores Register R_i in memory cell $S[R_j]$.

Calculation

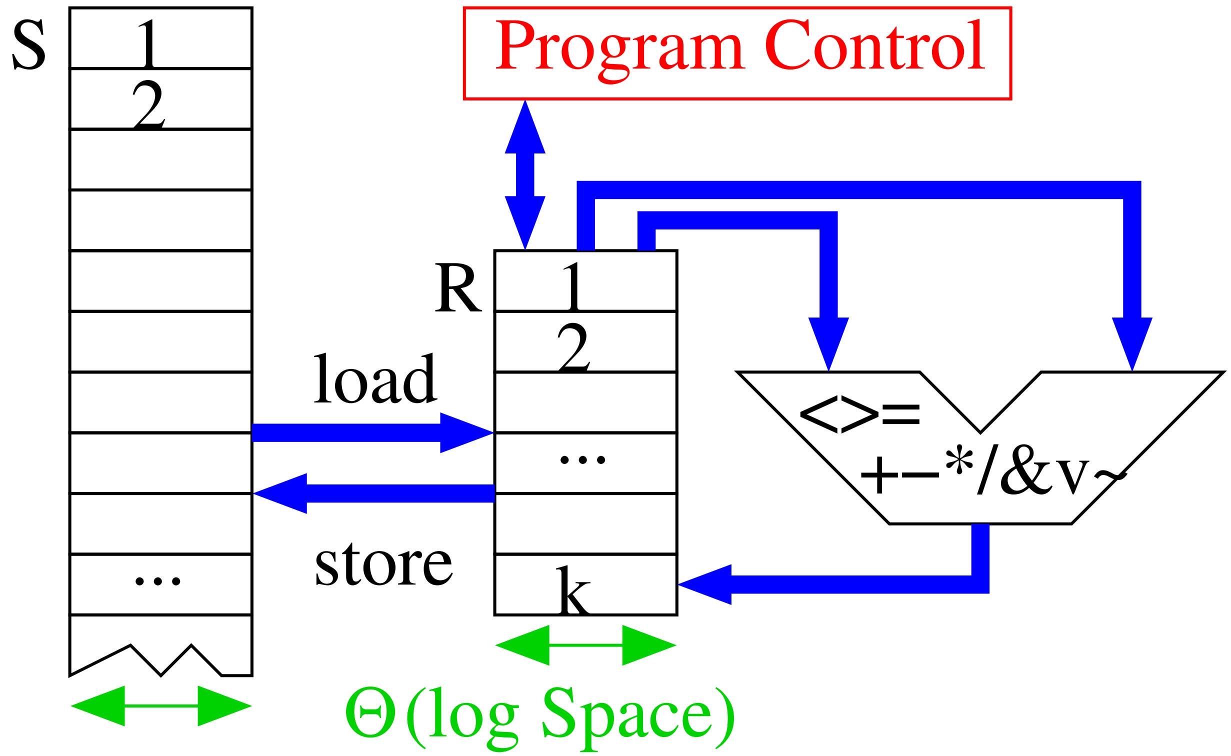


$R_i := R_j \odot R_\ell$ Register arithmetic.

'$\odot$' is a placeholder for a huge number of operations

Arithmetic, Comparison, Logic

Conditional jump



JZ j, R_i Puts the program execution an label j (goto j) if $R_i = 0$

„Small“ integers?

Alternatives:

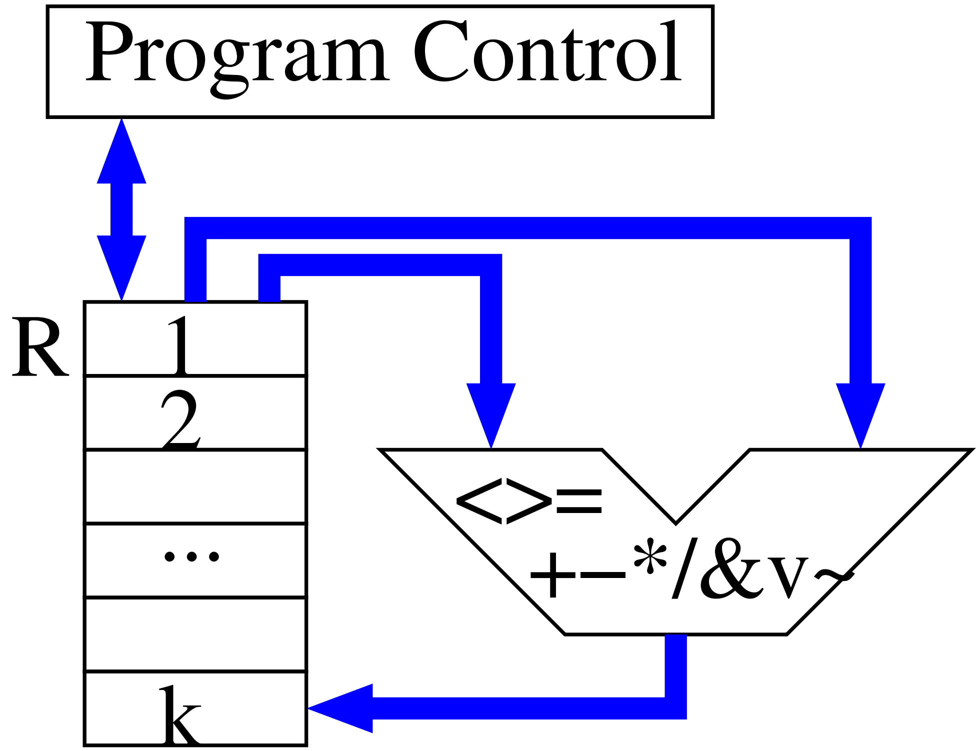
Constantly many bits (64?): theoretic unsatisfactory since only finite memory is addressed $\rightsquigarrow$ finite automaton

Arbitrary large: too optimistic

Enough for all used memory cells to be addressed: The best compromise.

Register machine

$\approx$ RAM $-$ memory $+$ arbitrary large



Register machines-computability

Configuration: $(q, R_1, \ldots, R_k)$

q is a counter for the program commands

„$\vdash^*$“ we have defined.

$f : \mathbb{N}^{k'} \to \mathbb{N}$, $k' \leq k$ is Register machines computable $\Leftrightarrow$

$\exists$RM $M : \forall n_1, \ldots, n_{k'}, m \in \mathbb{N}$:

$$f(n_1, \ldots, n_{k'}) = m \Leftrightarrow$$

$$(1, n_1, \ldots, n_{k'}, 0^{k-k'}) \vdash^* (q, f(n_1, \ldots, n_k), \ldots)$$

with PROGRAM$[q]$ $=$HALT

RAM-computability

Configuration: $(q, R_1, \ldots, R_k, S)$

Let M be a RAM:

input: $w \in \Sigma^n$ in $S[1]$,...,$S[n]$

output: $f_M(w)$ in $S[1]$,...,$S[|f_M(w)|]$

when HALT-command is executed.

Natural numbers are considered and we have to code them! Analog TM

High level program languages

Java, C/C++, Pascal,. . .

ML, Lisp,. . .

Prolog, Oz,. . .

. . .

are the most popular program models for us.

Compilers translate the programs in RAM Code.

LOOP-Program

Minimal program language for computability theory:

$\mathbb{N}$ main$(\mathbb{N}x_1,\ldots,\mathbb{N}x_k)${

$\mathbb{N}\ x_0 = 0;\ \mathbb{N}\ x_{k+1} = 0;\ \mathbb{N}\ x_{k+2} = 0;\ \ldots$

body;

return x_0;

}

body is allowed to use

Assignment: $x_i := x_j + c$, $c \in \{-1, 0, 1\}$ $\quad 0 - 1 := 0$

Schöning: $c \in \mathbb{Z}$

';': Sequence of instructions

loop x_i: Loop. Repeat x_i times. The contents of x_i is relevant before the first execution of the loop.

LOOP-Program

Observation: The Loop-Program always terminates.

Definition: f is Loop-computable if

$\exists$ Loop-Program P, which computes f.

Question: which functions are Loop-computable?

Are there total computable functions, which are not Loop-computable?

Loop-Programs.

$x := x + c$ // $c \in \mathbb{Z}$

$x := y + z$

$x := y \dot{-} z$ // $y \dot{-} z = y - z$ if $y \geq z$, 0 otherwise

$x := y \cdot z$

$x := y$ div z

$x := y$ mod z

arbitrary arithmetical expressions

if $x \neq 0$ **then** …

Example Addition $x_0 := x_1 + x_2$

$x_0 := x_1$

loop x_2

$\quad x_0{+}{+}$

Example Multiplication $x_0 := x_1 \cdot x_2$

loop x_1

$$x_0 := x_0 + x_2$$

Example **if** $x = 0$ **then** A

$y := 1$

loop x

$y--$

loop y

A

While-program

Minimal program language for computability theory:

$\mathbb{N}$ main$(\mathbb{N}x_1,\ldots,\mathbb{N}x_k)\{$

$\mathbb{N}\ x_0 = 0;\ \mathbb{N}\ x_{k+1} = 0;\ \mathbb{N}\ x_{k+2} = 0;\ \ldots$

body;

return x_0;

}

in the body we can use

Assignments: $x_i := x_j + c$, $c \in \{-1,0,1\}$ $\qquad 0 - 1 := 0$

';': Sequence of instructions

while$(x_i \neq 0)$: loops

WHILE simulates LOOP

loop x **do**

  P

$\rightsquigarrow$

$y := x$ // y does not occur in P

while $y \neq 0$ **do**

  $y := y - 1$

  P

Could we While by LOOP simulate ?

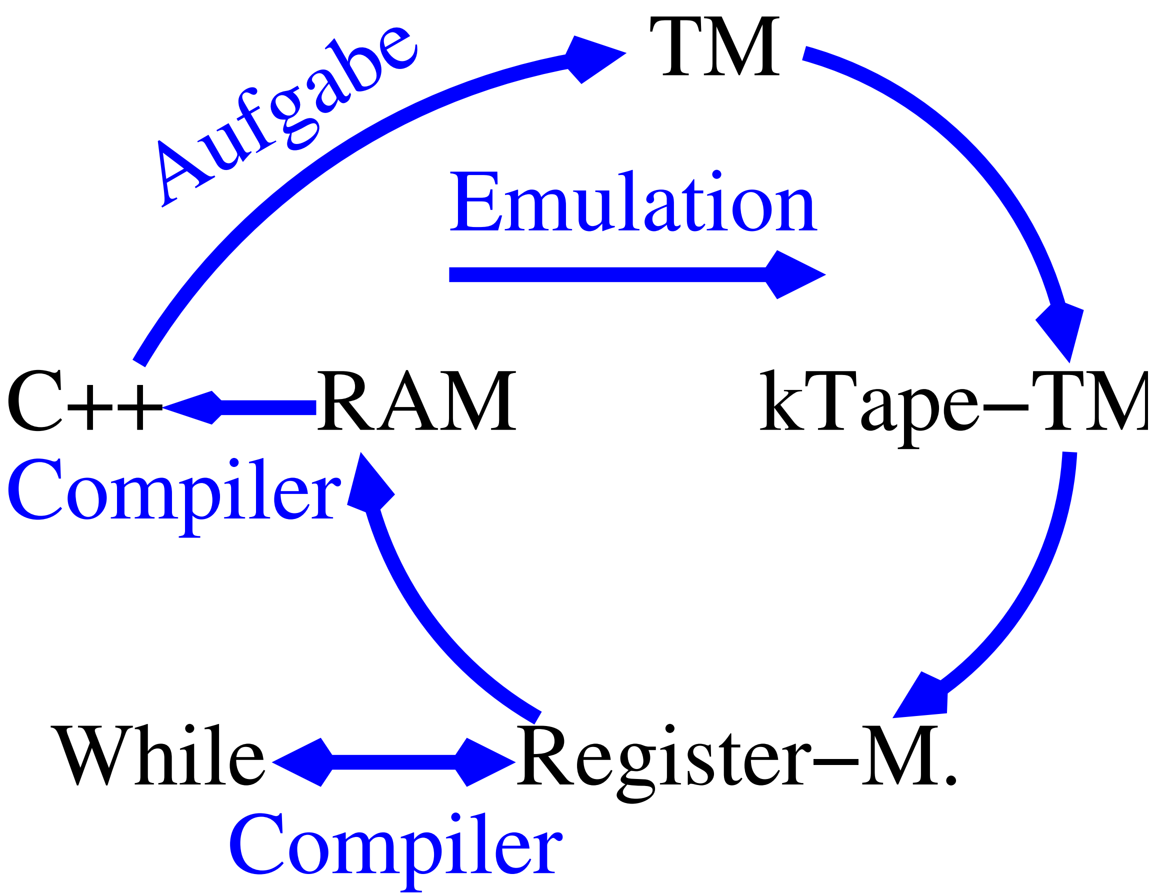
No !

Way not?

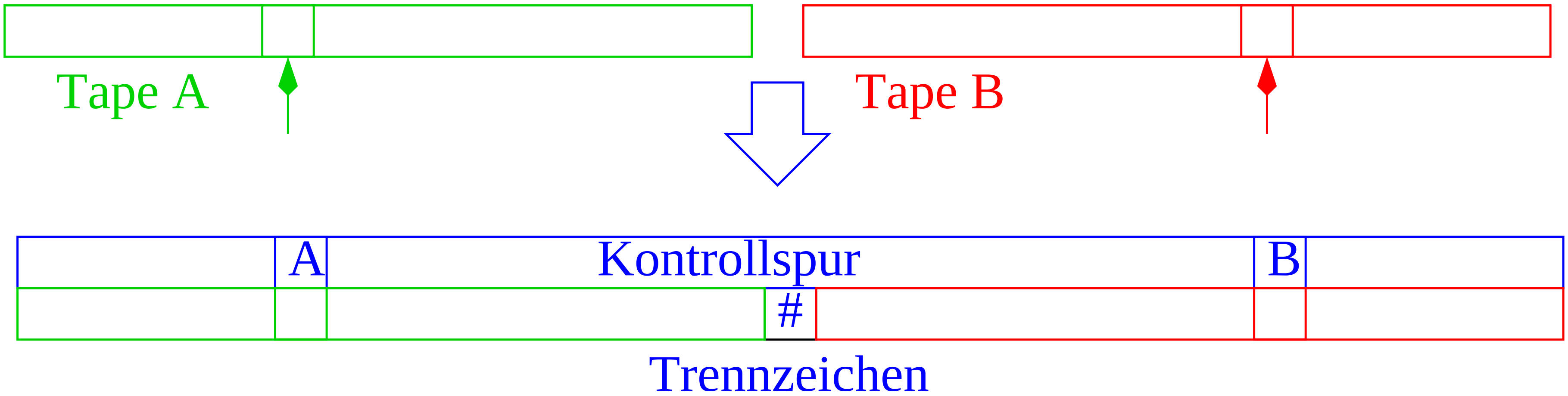
At least all total Turing computable functions ?

$\rightsquigarrow$ later

Equivalence of the machine models



Turing machine emulates k-tape-TM



- Non empty tape parts are hanged one by one (separators used)
- The position of the head is marked
- A state memorizes $k-1$ tape symbols

Proposition: The time T with k-tape-TM $\longrightarrow$ Time $\mathcal{O}\left(T^2\right)$ for one-tape-TM.

k-tape-TM emulates Register machine

- One tape per register (binary format or unary format)
- Different states for every program line
- Subprograms for arithmetic
- Assignment $\longrightarrow$ copy tape

Register machine emulates RAM

Idea: an additional register R_S represents the memory:

$$R_S = \sum_i S[i] \cdot 2^{bi}$$

with $b =$ number of RAM bits

$S[i]$ in R_j loading:

$$R_j := \frac{R_S}{2^{bi}} \bmod 2^b \quad .$$

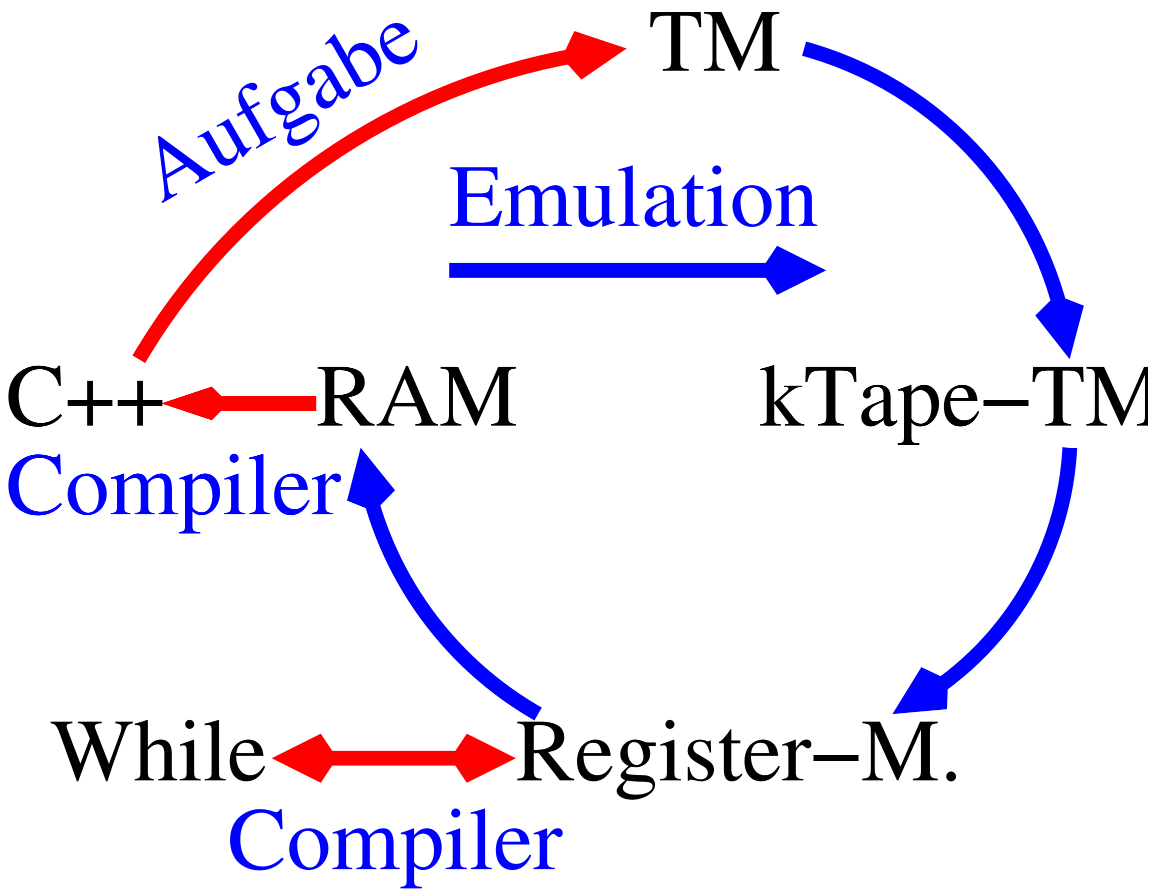
$S[i] := 0$:

$$R_S := R_S - \left(\frac{R_S}{2^{bi}} \bmod 2^b\right) 2^{bi}$$

R_j in $S[i]$ saving:

$$S[i] := 0;\ R_S := R_S + R_j \cdot 2^{bi}$$

Equivalence of machine models



Markov-Algorithms

Deterministic rules for string-rewriting.

Given: input $w \in \Sigma^*$

The set of rules $\Delta \in \left(\Gamma^* \times \Gamma^*\right)^*$

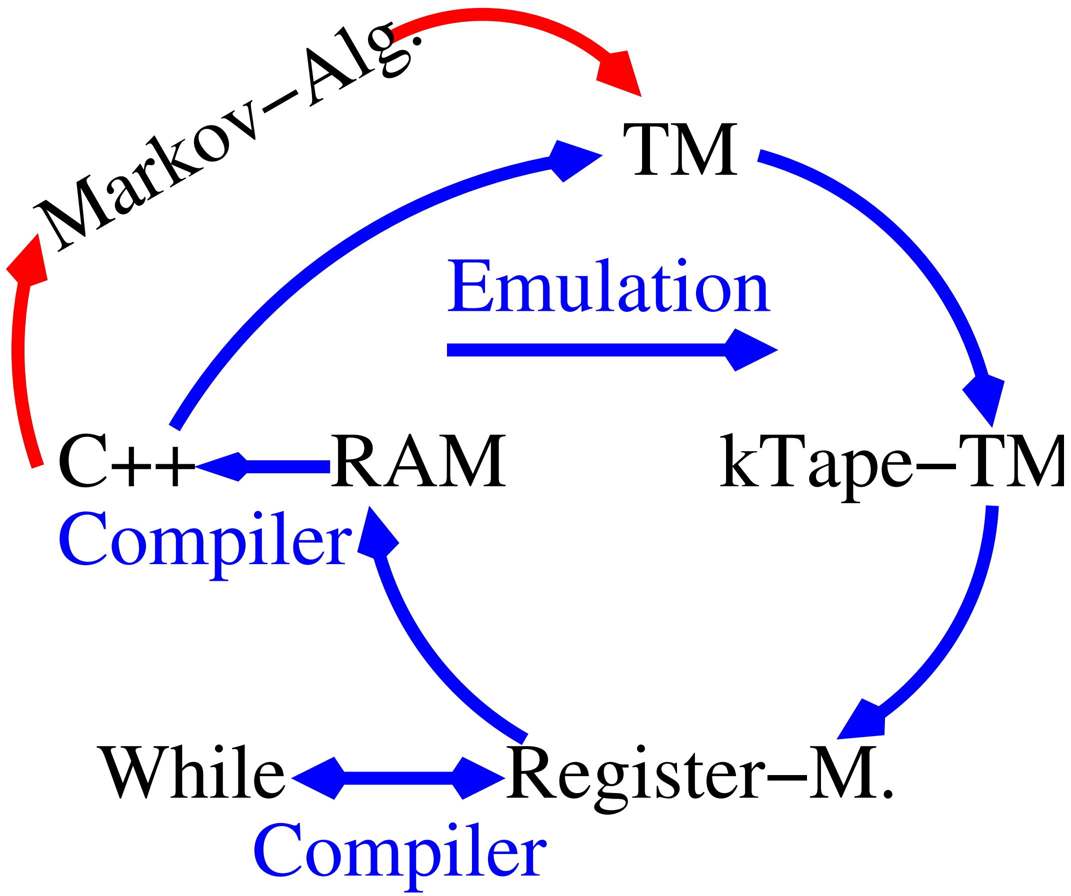
while $\exists(\ell, r) \in \Delta, u, v \in \Gamma^* : w = u\ell v$ **do**

find the first rule $\left(\ell, r\right) \in R$

and the shortest $u \in \Gamma^*$ such that $w = u\ell v$ for some $v \in \Gamma^*$

$w := urv$

Markov-Algorithms: Turing powerful



Markov-algorithms: Turing powerful

Given: TM $M = (Q, \Sigma, \Gamma, \delta, s, F)$ with input w.

Let max 1 $\sqcup$ left and right of input will be looked at.

Consider Markov-algorithm for the alphabet $Q \cup \Gamma \cup \{(,)\}$.

$$\begin{aligned}
\Delta = \cdots & \text{Special rules for the edge} \\
& \circ \left\langle ((q)a, (q')a') : \delta(q,a) = (q', a', N) \right\rangle \\
& \circ \left\langle (c(q)a, (q')ca') : \delta(q,a) = (q', a', L) \right\rangle \\
& \circ \left\langle ((q)ac, a'(q')c) : \delta(q,a) = (q', a', R) \right\rangle
\end{aligned}$$

input the Markov-algorithm $\sqcup(s)w\sqcup$

The consequence of the produced strings is the exact result of the configurations of the Turing machine!

Semi-Thue-System

Like nondeterministic Markov-algorithms.

Even so Turing powerful.

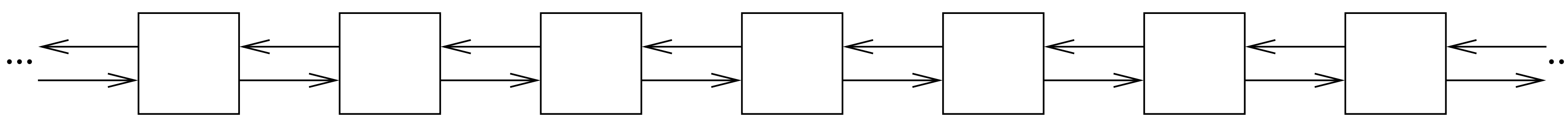
Our TM-simulation has always one applicable rule.

Cellular automata

Consider the finite automaton $(\{0,1\},\{0,1\}^2,\delta,\emptyset)$ with

Q	0	1	0	1	0	1	0	1
$L \times R$	$(0,0)$	$(0,0)$	$(0,1)$	$(0,1)$	$(1,0)$	$(1,0)$	$(1,1)$	$(1,1)$
δ	0	1	1	1	0	1	1	0

Connect infinitely many of such automata to a chain.



[M. Cook 2002]: This machine is Turing-powerful.

see also in Wikipedia „rule 110 cellular automaton“

Quantum computer

- One Qubit stores the superpositions of 0 and 1.
- Computations with n Qubits compute superposition of 2^n classical computations
- Quantum computer could in polynomial time factorizes and could obtain discrete logarithms
- This became many cryptographic algorithms to compromise

Quantum computer: computability and complexity theory

- Assumption of the complexity theory:
  - Factorizing, DLog are not in P (the same with randomizing)
  - Factorizing, DLog not NP hard.
- Turing machines could simulate Quantum computer

Result: Quantum computers were faster however not more powerful than classical computers

2.4 Primitive recursive functions

Basic functions

- $O(x) = 0$
- $S(x) = x + 1$
- $I_k^n(x_1, \ldots, x_n) = x_k, k \leq n$

Basic operations

- Superposition

  $h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$

- Primitive recursion

  $h(x_1, \ldots, x_n, 0) = f(x_1, \ldots, x_n)$

  $h(x_1, \ldots, x_n, y + 1) = g(x_1, \ldots, x_n, y, h(x_1, \ldots, x_n, y))$

Primitive recursive functions

A function is primitive recursive if it could be obtained from the basic functions by means of the operations superposition and primitive recursion applied finitely many times.

Examples:

- $x+0=0$
  $x+(y+1)=(x+y)+1$
- $x.0=0$
  $x.(y+1)=x.y+x$

Primitive recursive and μ-recursive functions

μ-operation:

$$f(x_1,\ldots,x_n) = \mu z[g(x_1,\ldots,x_n,z) = 0] \Leftrightarrow$$
$$(\forall y < z)(g(x_1,\ldots,x_n,y) > 0)\ \&\ g(x_1,\ldots,x_n,z) = 0$$

A function is μ-recursive if it could be obtained from the basic functions by means of the operations superposition, primitive recursion and μ-operation applied finitely many times.

Example: nowhere defined function $\emptyset(x) = \mu z[S(x) = 0]$.

Theorem The class of μ-recursive functions is exactly the class of the computable functions with TM.

2.5 The Ackermann function

[Ackermann 1928, Hermes]

Function $a(x,y)$

if $x=0$ **then return** $y+1$

if $y=0$ **then return** $a(x-1,1)$

return $a(x-1,a(x,y-1))$

Proposition: a is a total, TM-computable function

Proof:

Recursion $\rightsquigarrow$ Stack by RAM $\rightsquigarrow$ TM

Totality of the Ackermann function

Proof: Induction on the lexicographical order of (x,y):

Base of induction: $a(0,y) = y+1$

Induction step for $y = 0$:

$a(x,0) = a(x-1,1)$,

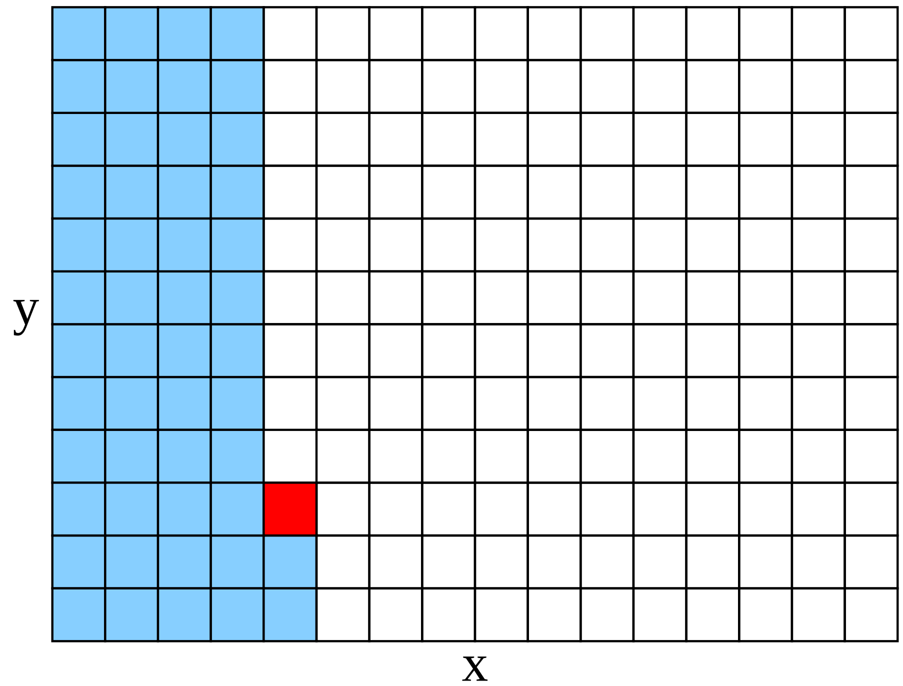
terminates,as $(x-1,1) < (x,0)$

Induction step for $x,y > 0$:

$a(x-1,a(x,y-1))$ terminates as

$(x,y-1) < (x,y)$ and

$(x-1,a(x,y-1)) < (x,y)$



How big numbers could compute a Loop program?

Definition:

Let for one Loop program P

$\mathbf{x} = (x_0, x_1, \ldots)$ the variable vector by which the program starts. Here arbitrary!

$\mathbf{x}' = (x'_0, x'_1, \ldots)$ the variable vector by which the program ends.

$$f_P(\mathbf{x}) := \sum_{i \geq 0} x'_i$$

$$f_P(n) := \max\left\{ f_P(\mathbf{x}) : \sum_{i \geq 0} x_i \leq n \right\}$$

The Ackermann function is not Loop-computable

Proof: Assume that a is Loop-computable.

$\longrightarrow a(n,n) = g(n)$ is computable by one Loop-program G.

$\longrightarrow a(n,n) = g(n) \leq f_G(n)$

But we will show:

Lemma E: $\forall$Loop-program $P : \exists k : \forall n \in \mathbb{N} : f_P(n) < a(k,n)$.

A contradiction.

Example

$$
\begin{aligned}
a(0,y) &= y+1 \\
a(1,y) &= a(0,a(1,y-1)) = a(1,y-1)+1 = \\
&\quad a(0,a(1,y-2))+1 = a(1,y-2)+2 = \cdots = \\
&\quad a(1,0)+y = y+a(0,1) = y+2 \\
a(2,y) &= a(1,a(2,y-1) = 2+a(2,y-1) = \cdots \\
&= 2y+a(2,0) = 2y+a(1,1) = 2y+3
\end{aligned}
$$

Example

$$
\begin{aligned}
a(2,y) &= 2y+3\\
a(3,y) &= a(2,a(3,y-1)) = 2a(3,y-1)+3\\
&= 2a(2,a(3,y-2))+3 = 4a(3,y-2)+3(1+2)\\
&= 4a(2,a(3,y-3))+3(1+2) = 8a(3,y-3)+3(1+2+4)\\
&= \cdots = 2^y \underbrace{a(3,0)}_{=5} + 3(\underbrace{1+2+\cdots+2^{y-1}}_{=2^y-1})\\
&= 2^{y+3}-3
\end{aligned}
$$

Example

$$a(3,y) = 2^{y+3} - 3$$

$$a(4,y) = a(3, a(4, y-1)) = 2^{a(4,y-1)+3} - 3$$

$$= 2^{a(3,a(4,y-2))+3} - 3 = 2^{2^{a(4,y-2)+3}-3+3} - 3$$

$$= 2^{2^{a(3,a(4,y-3))+3}} - 3 = 2^{2^{2^{a(4,y-3)+3}-3+3}} - 3$$

$$= \cdots = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \ \overbrace{a(4,0)}^{=a(3,1)=2^{1+3}-3} \ {}^{+3} - 3 = 2^{\cdot^{\cdot^{\cdot^{2^{16}}}}} - 3$$

$$a(4,2) = 2^{2^{16}} - 3 = 2^{65536} - 3$$

Monotonicity of the Ackermann function

Lemma A: $y < a(x,y)$

Lemma B: $a(x,y) < a(x,y+1)$

Lemma C: $a(x,y+1) \leq a(x+1,y)$

Lemma D: $a(x,y) < a(x+1,y)$

Lemma BD: $a(x,y) \leq a(x',y')$ if $x \leq x'$ and $y \leq y'$

Proof: Exercise. Induction,...

Lemma E: $\forall$Loop-program $P : \exists k : \forall n \in \mathbb{N} : f_P(n) < a(k,n)$.

Proof: Induction on the definition of the Loop-Programs.

Base of Induction:

$$f_{\text{emptyprogram}}(n) = n < n+1 = a(0,n).$$
$$f_{x:=\,y+c}(n) \leq 2n+1 < 2n+3 = a(2,n)$$

Induction step for $P = P_1; P_2$:

By IH $\exists k_1, k_2 : f_{P_1}(n) < a(k_1, n) \wedge f_{P_2}(n) < a(k_2, n)$.

Let now $k_3 = \max\{k_1 - 1, k_2\}$. It holds:

$$
\begin{aligned}
f_P(n) &\leq f_{P_2}(f_{P_1}(n)) && \text{Def. } f_P \\
&< a(k_2, f_{P_1}(n)) && \text{IH} \\
&< a(k_2, a(k_1, n)) && \text{IH,monotone} \\
&\leq a(k_3, a(k_3 + 1, n)) && \text{monotone} \\
&= a(k_3 + 1, n + 1) && \text{Def. } a \\
&\leq a(k_3 + 2, n) && \text{Lemma B}
\end{aligned}
$$

Induction step for $P = \textbf{loop}\ x_i\ \textbf{do}\ Q$:

Let x_i be not in Q (ex. in a new variable copied).

By IH $\exists k : f_Q(n) < a(k,n)$.

Let $\mathbf{x}$ be an input by $f_P(\mathbf{x})$ and it will be maximized by $\sum_j x_j \leq n$.

Let $m \leq n$ be the value of x_i in $\mathbf{x}$

$$
\begin{aligned}
f_P(n) &= f_P(\mathbf{x}) \\
&\leq \underbrace{f_Q(f_Q(\cdots f_Q}_{m \text{ times}}(n-m)\cdots))+m && \text{Def. } m \\
&\leq a(k, \underbrace{f_Q(f_Q(\cdots f_Q}_{m-1 \text{ times}}(n-m)\cdots))) + m - 1 && \text{IH}\cdots \\
&\leq \underbrace{a(k, a(k, \cdots a(k}_{m \text{ times}}, n-m)\cdots)) + m - m && \text{IH}
\end{aligned}
$$

Induction step for $P =$ loop x_i do Q:

$$
\begin{aligned}
f_P(n) &\le \underbrace{a(k,a(k,\cdots a(k}_{m \text{ times}},n-m)\cdots)) \\
&< \underbrace{a(k,a(k,\cdots a(k}_{m-1 \text{ times}},a(k+1,n-m)\cdots)) && \text{monotone} \\
&= \underbrace{a(k,a(k,\cdots a(k}_{m-2 \text{ times}},a(k+1,n-m+1)\cdots)) && \text{Def.}a \\
= \cdots &= a(k+1,n-1) && \text{Def.}a \\
&\le a(k+1,n) && \text{monotone}
\end{aligned}
$$

qed.

More fast growing functions

$k \mapsto \max\{f_P(0) : P$ is term. While-program with k instr.$\}$

Busy Beaver

$\Sigma(n)$: $\max_\delta \#$ ones, those on the tape stand after a DTM $(\{1,\ldots,n,Z\},\emptyset,\{0,1\},\delta,1,\{Z\})$ halts (empty input).

$S(n)$: $\max_\delta \#$transition, which one DTM $(\{1,\ldots,n,Z\},\emptyset,\{0,1\},\delta,1,\{Z\})$ that halts made (empty input).

Busy Beaver

Suppose that S(n) is a computable function.

EvalS : TM, evaluating S(n)

Clean cleaning the sequence of 1s

Double $n+n$

Double | EvalS | Clean - with n_0 states

Create n_0 - creating n_0 1s on an initially blank tape.

BadS : Create n_0 | Double | EvalS | Clean - $N = n_0 + n_0$ states

Starting with an initially blank tape it first creates a sequence of n_0 1s and then doubles it, producing a sequence of N 1s. Then BadS will produce S(N) 1s on tape, and at last it will clear all 1's and then halt. But the cleaning will continue at least S(N) steps, so the time of BadS is strictly greater than S(N), a contradiction with the def. of S(n).

More knowledge about the Busy Beaver

$\Sigma(n)$, $S(n)$ are total not computable functions.

n	$\Sigma(n)$	$S(n)$
1	1	1
2	4	6
3	6	21
4	13	107
5	$\geq$4 098	$\geq$47 176 870
6	$> 1.29 \cdot 10^{865}$	$> 3 \cdot 10^{1730}$

[`http://www.drb.insel.de/~heiner/BB/`],

[`http://www.logique.jussieu.fr/~michel/ha.html`]

Record holder

```
n:       6           5           4           3           2
q/in 0   1       0   1       0   1       0   1       0   1
A:   B1R F0L; B1R C1L; B1R B1L; B1R Z1L; B1R B1L
B:   C0R D0R; C1R B1R; A1L C0L; B1L C0R; A1L Z1R
C:   D1L E1R; D1R E0L; Z1R D1L; C1L A1L;
D:   E0L D0L; A1L D1L; D1R A0R;
E:   A0R C1R; Z1R A0L;                   n=1:
F:   A1L Z1R;                            H1N ---
```

Slowly growing functions

Inverse Ackermann function

$$\alpha(m,n) := \min\{i \geq 0 : a(i, \lfloor m/n \rfloor) > \log_2 n\}$$

For each realistic case is valid $\alpha(m,n) \leq 5$.

But $\alpha(m,n) \notin O(1)$.

An important data structure has overall complexity $m\Theta(\alpha(m,n))$ for m operations and n objects:

Union-Find Data Structure

Class UnionFind$(n : \mathbb{N})$ // Maintains a partition of $1..n$

Function find$(i : 1..n) : 1..n$

assert $\forall i, j \in \{1, \ldots, n\}$:

find$(i) =$ find$(j) \Leftrightarrow i, j$ are in the same part

Procedure union$(i, j : 1..n)$

A:= the part with $i \in A$

B:= the part with $j \in B$

join A and B to a single part

Application: for example Kruskal's algorithm for minimal spanning tree

Class UnionFind$(n:\mathbb{N})$

parent$=[1,2,\ldots,n]$: **Array** $[1..n]$ **of** $1..n$

gen$=[0,\ldots,0]$: **Array** $[1..n]$ **of** $0..\log n$ // generation of leaders

Function find$(i:1..n):1..n$

if parent$[i]=i$ **then return** i

else $i' :=$ find(parent$[i]$)

parent$[i] := i'$ // path compression

return i'

Procedure link$(i,j:1..n)$

assert i and j are leaders of different subsets

if gen$[i]<$ gen$[j]$ **then** parent$[i] := j$ // balance

else parent$[j] := i$; **if** gen$[i]=$ gen$[j]$ **then** gen$[i]$++

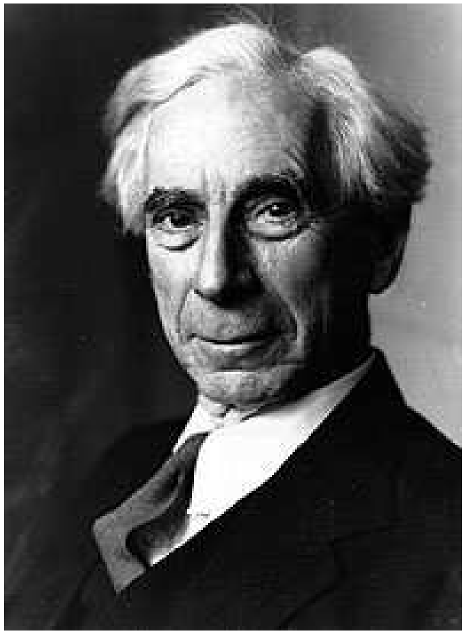
Procedure union$(i,j:1..n)$

if find$(i)\neq$ find(j) **then** link(find(i), find(j))

2.6 Halting problem, Undecidability, Reducibility

- Gödel numbering: TMs could processed themselves as an input
- Important example: Universal TM
- Diagonal argument: a undecidable language
- Reductions: it shows that other problems are undecidable.

Paradoxes and Self reference



The barber of a small town

shaves all and only those men

who do not shave themselves.

.

Does the barber shave himself?

Paradoxes and Self reference

Maintained Daniel Dösentrieb an all-knowing machine to have invented.

Yes No

One places a yes/no Question and the answer lights up.

Dagobert Duck buys the machine.

Wants to pay however only with more correct function.

It places the Question to the machine:

Will you answer with no?

What happens?

Decidability

$A \subseteq \Sigma^*$ is decidable (computable) if

the characteristic function χ_A is computable.

$$\chi_A(w) = \begin{cases} 1 & \text{if } w \in A \\ 0 & \text{if } w \notin A \end{cases}$$

Semi-decidability

$A \subseteq \Sigma^*$ is semi-decidable if

the „half“ characteristic function χ_A is computable.

$$\chi_A(w) = \begin{cases} 1 & \text{if } w \in A \\ \perp & \text{if } w \notin A \end{cases}$$

Proposition: $A \subseteq \Sigma^*$ decidable $\Leftrightarrow$
A and $\bar{A}$ are both semi-decidable

Proof: Let TM

M_A acceptor for A and

$M_{\bar{A}}$ acceptor for $\bar{A}$

for $s := 1$ **to** ∞ **do**

if M_A halts in s steps **then** Accept

if $M_{\bar{A}}$ halts in s steps **then** Reject

Recursive enumerability

$A \subseteq \Sigma^*$ recursively enumerable if

$A = \emptyset$ or $\exists$total computable function $f : \mathbb{N} \to \Sigma^*$:

$$A = \{f(1), f(2), f(3), \ldots\}$$

Proposition: A is recursively enumerable $\Leftrightarrow$ A is semi-decidable

Recursive enumerability $\longrightarrow$ semi-decidable

Let A is recursively enumerable by means of f.

Function $\chi'_A(x)$

  for $s := 1$ **to** ∞ **do**

    if $f(n) = x$ **then return** 1

Semi-decidable $\longrightarrow$ recursively enumerable

- Consider $\pi(k,m) = 2^k(2m+1) - 1$ - a codding function for all pairs of natural numbers.
- Each natural number n is a code of exactly one pair $n = \pi(k,m)$.
- Let $L(\pi(m,k)) = m$ and $R(\pi(m,k)) = k$ be the decoding functions.
- π, L, R are computable functions.
- Consider the sequence of all words in Σ^*:
  $\alpha_0, \alpha_1, \ldots, \alpha_i, \ldots$
  in the following order $|\alpha_i| < |\alpha_{i+1}|$ or $|\alpha_i| = |\alpha_{i+1}|$ and α_i is lexicographically less than α_{i+1}.
- For example: $a, b, aa, ab, ba, bb, \ldots$

Semi-decidable $\longrightarrow$ recursively enumerable

Case $A = \emptyset$: trivial.

Otherwise we give one function $f : \mathbb{N} \to \Sigma^*$ with the range A.

Function $f(n)$

a:$=$ some fixed element of A

interpret n as a pair $n = \pi(m, k)$

Consider the word $u = \alpha_m$

if an acceptor M for A accepts u in $\leq k$ steps **then return** u

else return a

Semi-decidable $\longrightarrow$ recursively enumerable

- f ist total
- f gets only values from A
- $\forall u \in A \exists k : M$ accepts u in k steps
- $f(\pi(m,k)) = \alpha_m$

■

Exercise: Prove that if A is infinite, then A is decidable iff there exists a total computable function $f : \mathbb{N} \rightarrow \Sigma^* :$

$$A = \{f(0) < f(1) < f(2) < \ldots\}$$ in a lexicographical order.

Equivalent statements

- A is recursively enumerable
- A is semi-decidable
- A is of Chomsky type 0
- $A = L(M)$ for TM M
- χ'_A is Turing-, While-, RegM., RAM, $\ldots$ computable
- A is a domain of one (partial) computable function
- A is a range of a computable function

2.7 Non decidable Problems

Enumeration of Turing-machines

Consider $T = (Q, \Sigma, \Gamma, \delta, s, F)$. Let:

- $Q = \{1, \ldots, n\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, \sqcup\}$, $\sqcup = 2$
- $s = 1$
- $F = \{2\}$

for appropriate constant n

Goödel number $\langle M\rangle$ of Turing machine M

Define the following strings in $\{0,1\}$:

Code $\delta(q,a)=(r,b,d)$ by $0^q\,1\,0^{a+1}\,1\,0^r\,1\,0^{b+1}\,1\,0^d$

where d is the code of the directions: $N=1, L=2, R=3$.

The Turing-machine will be coded by binary numbers:

$$111\text{code}_1 11\text{code}_2 11\ldots 11\text{code}_z 111,$$

code_i for $i=1,\ldots,z$: all values of function δ in arbitrary order are written.

Convention:

n is not a Goödel number of a TM,

$\longrightarrow n$ descibes one TM, which accepts the $\emptyset$

The Gödel numbers

Observation

The Goöel numbering describes an

injective mapping of standardised TMs to natural numbers

Example

Let $M = (\{1,2,3\},\{0,1\},\{0,1,\sqcup\},\delta,1,\{2\})$, with

$\delta(1,1) = (3,0,R)$

$\delta(3,0) = (1,1,R)$

$\delta(3,1) = (2,0,R)$

$\delta(3,\sqcup) = (3,1,L)$

$\langle M \rangle$ is then:

1110100100010100011000101010010001100010010010100011

0001000100010010011111

Universal Turing machine

$$U = (Q_u, \{0,1\}, \{0,1,\sqcup\}, \delta_u, s_u, F_u)$$

input: $\langle M\rangle w$

M is the simulated TM, w is the binary coded input.

U simulates M on w.

U accepts $\langle M\rangle w$ if M accepts w

Universal Turing machine

3 Tapes:

1. $\langle M \rangle$
2. the state q_M of M unary coded
3. the content of the tape w of M

Universal Turing machine

if prefix v of w represents a TM **then** // 111tuple111

move v on the tape $\langle M \rangle$

$q_M := 1$ // the initial state of M

while $q_M \neq 2$ **do** // final state of M

run to the beginning of $\langle M \rangle$

foreach $(q, a, r, b, d) \in \langle M \rangle$ **do** // field by field

if $q = q_M$ **then** // compare with q_M

if input symbol of the tape 3$=a$ **then**

$q_M := r$ // copy on the state tape

put b on the tape 3

the moving on the tape 3 is according to the chosen d

Universal Turing machine: 3tape→1tape

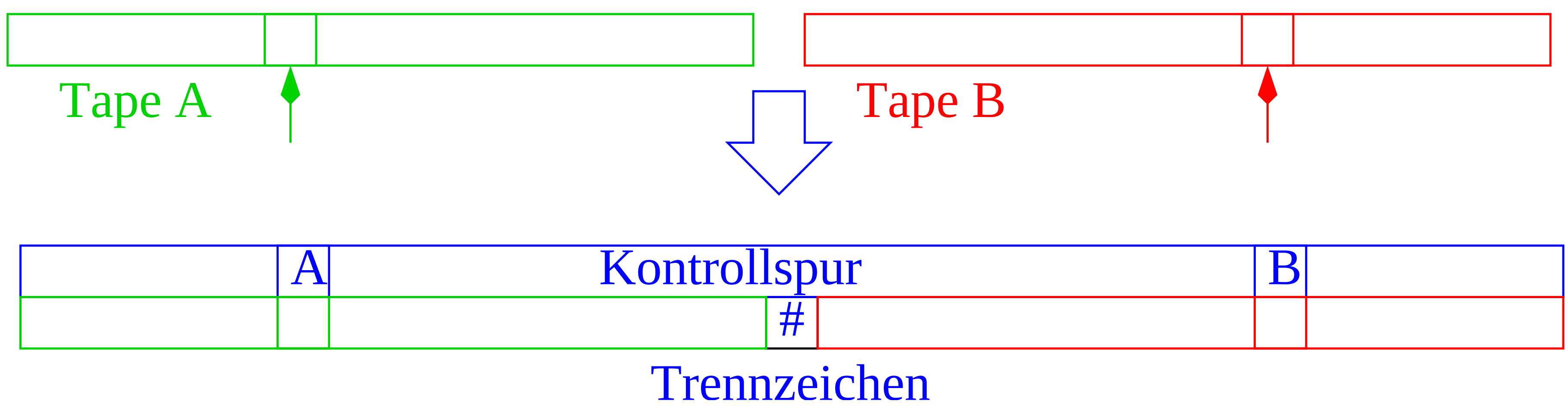
Actually we know how it works.

Problem: tape alphabet independently of M but $> \{0,1\}$

Code tape alphabet by constantly many $\{0,1\}$.

Problem: input has to be coded too.

This settles a upstream coding TM.



The diagonal language L_d

Let M_i is the TM with $\langle M_i \rangle = i$.

Let w_i is the binary representation of i.

$L_d := \{w_i : M_i$ does not accept $w_i\}$

Proposition: L_d is undecidable

Proof:

Assume:

$L_d = \{w_i : M_i$ does not accept $w_i\}$ is decidable.

$\overset{\text{Def. „decidable“}}{\longrightarrow}$ $\exists M_i : M_i$ accepts L_d and halts always.

What does M_i do with w_i?

$w_i \in L_d \overset{\text{Def. } M_i}{\longrightarrow} w_i$ will be accepted. $\overset{\text{Def. } L_d}{\longrightarrow} w_i \notin L_d$

$w_i \notin L_d \overset{\text{Def. } M_i}{\longrightarrow} w_i$ will not be accepted. $\overset{\text{Def. } L_d}{\longrightarrow} w_i \in L_d$

Both lead to a contradiction.

Corollary:

$\overline{L_d} = \{w_i : M_i$ accepts $w_i\}$ is undecidable

Assume: $\overline{L_d}$ is decidable.

$\longrightarrow \exists M : M$ accepts $\overline{L_d}$

modify $M \rightsquigarrow M'$ so M' accepts L_d

(Exchange accepts/does not accept for the final state).

A contradiction.

Note that $\overline{L_d}$ is semi-decidable. Run the universal machine on $\langle M_i \rangle w_i$.

Corollary: $\overline{L_d}$ is semi-decidable and not decidable.

Undecidable problems

Does not exist a program P, such that

$$\text{halts}(\langle P\rangle, X)) = \begin{cases} yes & \text{if } P(X) \text{ halts} \\ no & \text{otherwise} \end{cases}$$

Assume that there is:

D(X)= **if** halts(X,X) **then** loop(X) **else** halt

D($\langle D\rangle$)= **if** halts($\langle D\rangle$,$\langle D\rangle$) **then** loop($\langle D\rangle$) **else** halt

If halts($\langle D\rangle, \langle D\rangle$)=yes, then $\downarrow$D($\langle D\rangle$), but $\uparrow$D($\langle D\rangle$).

If halts($\langle D\rangle, \langle D\rangle$)=no, then $\downarrow$D($\langle D\rangle$), but $\uparrow$D($\langle D\rangle$).

Undecidable problems

There in no program is-safe, which is not a virus and:

$$\text{is}-\text{safe}(\langle P\rangle, X)) = \begin{cases} \mathit{yes} & \text{if } P(X) \text{ does not start virus} \\ \mathit{no} & \text{otherwise} \end{cases}$$

D(X)= **if** is-safe(X,X) **then** virus(X) **else** "Hello"

D($\langle D\rangle$)= **if** halts($\langle D\rangle, \langle D\rangle$) **then** loop($\langle D\rangle$) **else** halt

If is-safe($\langle D\rangle, \langle D\rangle$)=yes, then D($\langle D\rangle$) is not activated virus, but virus($\langle D\rangle$) is activated.

If is-safe($\langle D\rangle, \langle D\rangle$)=no, then virus is activated $\Rightarrow$ is-safe is activated a virus.

Halting problem

$H := \{w_i v : M_i \text{ halts on } v\}$

Proposition: H is not decidable.

Proof: Assume that H is decidable.

We construct one TM, by which $\overline{L_d}$ will be accepted.

$w_i \in \overline{L_d}$?

$\Leftrightarrow M_i$ accepts w_i.

$\Leftrightarrow w_i w_i \in H \wedge M_i$ accepts w_i.

This we could do by means of one TM for H and one universal TM.

A contradiction.

The bounded Halting problem

Proposition:

$\{w_i v \# w_j : M_i$ halts on v in at most j steps$\}$

is decidable.

Proof:

Let the universal TM U run on $w_i v$

in j simulated steps.

More undecidable problems

Given Turing machines T, T'

- $L(T) = \emptyset$? emptyness
- $|L(T)| = \infty$? infiniteness
- $L(T) = \Sigma^*$? completeness
- $L(T) = L(T')$? equivalence

Undecidability of emptyness

Assume that M accepts $\{i : L(M_i) = \emptyset\}$

We will show that then $\overline{L_d}$ will be decidable.

$w_i \in \overline{L_d} = \{w_i : M_i$ accepts $w_i\}$?

Construct a Turing machine $T(i)$:

erase input

run M_i on w_i

if state$(M_i) \neq 2$ **then** endless loop

Now is $L(T(i)) \neq \emptyset$ if $w_i \in \overline{L_d}$.

Also $\overline{L_d}$ is decidable.

A contradiction

Undecidability of completeness

$L(T) = \Sigma^*$?

Similar proof as the emptyness! Here $T(i)$ ignores its input!

Meta programming

The proof of the emptyness get one program and transforms it to another.

Important programming technics.

Rice theorem

Let $\mathbf{R}$ be the class of all Turing computable functions.

Theorem Let $\mathbf{S}$ be a nontrivial class of Turing computable functions ($S \neq \emptyset, S \neq R$). Then the set

$$C(S) = \{w \mid M_w \text{ computes a function } \in S\}$$

is not decidable.

Proof:

Assume that $C(S)$ is decidable.

Case 1. $\emptyset \notin S$ and $f \in S$. Then there is a Turing machine M_f that computes f.

Let M be a Turing machine and w is a word.

$$T_{M,w}(x) = \begin{cases} M_f(x) & \text{if} \downarrow M(w) \\ \perp & \text{otherwise} \end{cases}$$

Then:

if $\downarrow M(w) \Rightarrow (\forall x) T_{M,w}(x) = f(x)$

if $\uparrow M(w) \Rightarrow T_{M,w}(x) = \emptyset$.

$$T_{M,w} \in S \Leftrightarrow \downarrow M(w).$$

$$\langle T_{M,w} \rangle \in C(S) \Leftrightarrow \langle M \rangle w \in H.$$

A contradiction.

Case 2. $\emptyset \in S$. Consider: $R \setminus S$ is non trivial.

Then $C(\overline{S})$ is undecidable, and hence $C(S)$ is undecidable.

Self-reference

Construct a TM which ignores the input an prints out a copy of its own description. We call this machine SELF.

Lemma : There is a computable function $q(w) = \langle P_w \rangle$-the description of P_w and halts, where $\forall x P_w(x) = w$.

(a) On input w construct the following TM

P_w:

1. Erase the input.

2. Write w on the tape.

3. Halt.

(b) Output $\langle P_w \rangle$.

Self-reference

The construction of SELF: $\langle SELF\rangle = \langle AB\rangle$.

The job of A: print $\langle B\rangle$

The job of B: print $\langle AB\rangle$

$A = P_{\langle B\rangle}$ [$\Rightarrow q(\langle B\rangle) = \langle A\rangle$].

$B =$ On input $\langle M\rangle$:

1. Compute $q(\langle M\rangle) = \langle M'\rangle$.
2. Combine the result with $\langle M\rangle$ to receive $\langle M'M\rangle$.
3. Print and halt.

Then SELF:

1. $A : \langle B\rangle$;
2. $B : q(\langle B\rangle) = \langle A\rangle$;
3. $B : \langle A\rangle\langle B\rangle \rightsquigarrow \langle AB\rangle$

Post Correspondence Problem (PCP)

Given: finite sequences of pairs of words:

$$K = (x_1, y_1) \cdots (x_n, y_n) \in (\Sigma^+ \times \Sigma^+)^*$$

Question:

$$\exists i_1, \ldots, i_k \in \{1, \ldots, n\} : x_{i_1} \ldots x_{i_k} = y_{i_1} \ldots y_{i_k}$$

?

Example

- $K = ((1, 111), (10111, 10), (10, 0))$ has the solution $(2, 1, 1, 3)$, it holds:

$$x_2 x_1 x_1 x_3 = 101111110 = 101111110 = y_2 y_1 y_1 y_3$$

- $K = ((10, 101), (011, 11), (101, 011))$

  has no solution:

  $(133\cdots)$

Example [Mirko Rahn]

- $K = ((0, 011), (001, 1), (1, 00), (11, 110))$

  has the shortest solution of length 595:

  1211212112112121121203212112130321203311213111212031212121121312112

  0321211210321213032120211112033112132121212131112121121111203203121

  2120321211212121213131203213032120320321031213033112131302103201111

  1212112111200210121212121203212112121212021203203213032121120321321

  3130330321213030312113113032001032121112121131212303212120321210321

  0110321303230212123033101120313102121213121020320312021313121321112

  1032111121212021111212121203213212121211203213031120321121213033121

  2131121211203310320312120321213102101032130321020212303302132101100

  30212122113203121210103202123132110311212312120303213303003

PCP is semidecidable

Algorithm:

Procedure PCP$((x_1,y_1)\cdots(x_n,y_n))$

  for $k := 1$ **to** ∞ **do**

    foreach $i_1\cdots i_k \in \{1..n\}^k$ **do**

      if $x_{i_1}\cdots x_{i_k} = y_{i_1}\cdots y_{i_k}$ **then**

        output $i_1\cdots i_k$

        return

PCP is undecidable

Proof see Schöning.

Idee: assume solvable $\longrightarrow$ Halting problem solvable

$x_{i_1}\dots x_{i_k} = y_{i_1}\dots y_{i_k} = (s)w\#\cdots\#u(f)v$

describes an accepting sequence of TM-configurations

Hilbert's 10. Problem — Diophantine equations

Given:

multivariable polynom p

with integer coefficients.

Question [Hilbert 1900]:

$$\exists x_1,\ldots,x_n \in \mathbb{Z} : p(x_1,\ldots,x_n) = 0?$$

[Matiyasevich 1970]: The problem is undecidable.

Closureness of decidable languages

Closed under

$\cup$

$\cap$

$\overline{}$

$\cdot$

Closureness of semidecidable languages

Closed under

$\cup$

$\cap$

nicht abgeschlossen unter

$\overline{\cdot}$

Closureness of semidecidable languages under union

Let M_1 and M_2 be acceptors for L_1 and L_2

Acceptor for $L_1 \cup L_2$:

for $j := 1$ **to** ∞ **do**

  if M_1 accepts w after j steps **then** accept

  if M_2 accepts w after j steps **then** accept

Nonclosureness of semidecidable languages under complement

Assume : closed under complement.

Let M be an acceptor for L_d, $\overline{M}$ acceptor for $\overline{L_d}$

Function isInLd(w)

    for $j := 1$ **to** ∞ **do**

        if M accepts w after j steps **then return** true

        if $\overline{M}$ accepts w after j steps **then return** false

Either of them halts.

$\longrightarrow L_d$ decidable.

A contradiction.

Application of the parallel realization

Several algorithms $A_1,\ldots,A_k$, to solve a difficult problem (slowly, fast, never).

Load all algorithms (pseudo) simultaneously out.

− If all are equivalent fast we have wasted factor k cost of computation

+ If one is never ready we have infinitely much won.

+ With very different running time we could win on the average.

+ We could use parallel processors.

+ Often we could save a part of the redundant work.

Parallel realization

Application: Theorem prover, Program/Hardware-verifier, difficult planning and optimization problems

Example: Prepossessing predicate formulas in time $\mathcal{O}\left((4/3)^{\text{\#Variables}}\right)$.

[U. Schöningh, A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems, FOCS, 1999]