Register machines

Algorithms, or "effective procedures"

No precise definition at time Hilbert posed the Entscheidungsproblem, just examples...

Common features of the examples:

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

The "elementary operations" for register machines will be:
- add 1 to a natural number
- test whether a natural number is 0
- subtract 1 from a positive natural number
- jump ("goto")
- conditional ("if-then-else")

\[ 0, 1, 2, 3, \ldots \] stored in (idealised) registers

**Definition:** a register machine consists of:

- finitely many registers \( R_0, \ldots, R_n \) (each capable of storing a natural number)
- a program, consisting of a finite list of instructions of the form

<table>
<thead>
<tr>
<th>Label</th>
<th>Body of instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (i+1)^{th} ) instruction in the list is labelled ( L_i ) ( (i=0,1,\ldots) )</td>
<td></td>
</tr>
</tbody>
</table>
- instructions take one of three forms:

| \( L : R^+ \rightarrow L' \) | add 1 to contents of register \( R \) and jump to instruction labelled \( L'' \) |
| \( L : R^- \rightarrow L', L'' \) | if contents of \( R \) is \( >0 \), subtract 1 from it and jump to \( L' \); otherwise, jump to \( L'' \) |
| \( L : \text{HALT} \) | stop executing instructions |
Graphical notation for programs

Increment instruction \( L : R^+ \rightarrow L' \) represented by \( R^+ \rightarrow L' \)

Conditional decrement instruction \( L : R^- \rightarrow L', L'' \) represented by \( R^- \rightarrow L' \rightarrow L'' \)

Halt instruction represented by \( \text{HALT} \).

So:
- nodes of graph indicate register operations (and halting)
- arcs of graph represent jumps between instructions
- labels of instructions become implicit
- loose sequential order of instructions, which is no problem so long as the START instruction of the graph is indicated.

Example: register machine for addition

Registers \( R_0 \), \( R_1 \), \( R_2 \)

Program:
- \( L_0 : R_1^- \rightarrow L_1, L_2 \)
- \( L_1 : R_0^+ \rightarrow L_0 \)
- \( L_2 : R_2^- \rightarrow L_3, L_4 \)
- \( L_3 : R_0^+ \rightarrow L_2 \)
- \( L_4 : \text{HALT} \)

Starting with initial values \( 0, x, y \) in \( R_0, R_1, R_2 \)

the machine reaches \( L_4 : \text{HALT} \)

with values \( x+y, 0, 0 \)

in \( R_0, R_1, R_2 \).
Example: register machine for addition

registers R0, R1, R2

Program:

L0: R1 → L1, L2
L1: R0+ → L0
L2: R2 → L3, L4
L3: R0+ → L2
L4: HALT

Starting with initial values 0, x, y in R0, R1, R2
the machine reaches L4: HALT
with values x+y, 0, 0 in R0, R1, R2.

Example: register machine for addition

registers R0, R1, R2

Program:

L0: R1 → L1, L2
L1: R0+ → L0
L2: R2 → L3, L4
L3: R0+ → L2
L4: HALT

Starting with initial values 0, x, y in R0, R1, R2
the machine reaches L4: HALT
with values x+y, 0, 0 in R0, R1, R2.
Having loaded registers with initial values (from $N_S = \{0, 1, 2, \ldots\}$)

a computation (or run) of the register machine consists of obeying the program instructions, starting with the first in the list

Either

the computation continues forever

E.g. L1: $R1^+ \rightarrow L1$
L2: HALT
never halts

Or

it halts, because

Either

a HALT instruction is obeyed

"proper halt"

E.g. L1: $R1^+ \rightarrow L3$
L2: HALT
halts erroneously

Or

we are asked to jump to a label not in the list

"erroneous halt"

Graphical representation of

L1: $R1^+ \rightarrow L1$
L2: HALT

is

\begin{align*}
R1^+ \\
\text{HALT}
\end{align*}

L1: $R1^+ \rightarrow L3$
L2: HALT

is

\begin{align*}
R1^+ \\
\text{HALT}
\end{align*}

N.B. Can always modify programs to turn erroneous halts into proper halts by adding extra HALT instructions to the list with appropriate labels.
Note that the operation of a register machine is deterministic, in the sense that the next instruction to be obeyed (if any) is uniquely determined.

Because of this determinism and the possibility that computations do not halt, 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 $f \subseteq X \times Y$ (of the set $X \times Y \overset{\text{def}}{=} \{(x,y) | x \in X \land y \in Y\}$ of ordered pairs) satisfying $(x,y) \in f \land (x,y') \in f \Rightarrow y = y'$ (i.e. for all $x \in X$ there is at most one $y \in Y$ with $(x,y) \in f$).

**NOTATION**:

- $f(x) = y \quad \text{means} \quad (x,y) \in f$
- $f(x) \downarrow \quad \text{means there is some} \quad y \quad \text{such that} \quad (x,y) \in f$
- $f(x) \uparrow \quad \text{means there is no} \quad \cdots \quad \cdots \quad \cdots$

$Pfn(X,Y) \overset{\text{def}}{=} \text{set of all partial functions from} \ X \ \text{to} \ Y$

$Fun(X,Y) \overset{\text{def}}{=} \text{set of all (total) functions from} \ X \ \text{to} \ Y$

Those $f \in Pfn(X,Y)$ such that $f(x) \downarrow$ for all $x \in X$
**Definition:**

\( f \in \text{Pfn}(\mathbb{N}^n, \mathbb{N}) \) is (register machine) **computable** if and only if there is a register machine \( M \) with at least \( n+1 \) registers, \( R_0, R_1, R_2, \ldots, R_n \) say, (and maybe some other registers as well) with the property that for all \( (x_1, \ldots, x_n) \in \mathbb{N}^n \) and all \( y \in \mathbb{N} \)

\[ f(x_1, \ldots, x_n) = y \text{ if and only if the computation of } M \text{ starting with } R_1 = x_1, \ldots, R_n = x_n, \text{ and all other registers } = 0, \text{ halts with } R_0 = y. \]

E.g. the example register machine on Slide 18 shows that the addition function \( f(x, y) \equiv x + y \) is computable.

**N.B.** there may be many different register machines that compute the same partial function.

Of course the investigation of what kinds of function are computable, and what kinds are not, is a major concern of this course. For the moment let’s just see some more examples...
Multiplication: \( f(x, y) = xy \) is computable

Hence if the machine is started with \((R_1, R_2, R_3, R_0) := (x, y, 0, 0)\), it halts with \((R_1, R_2, R_3, R_0) := (0, y, 0, xy)\) under the assumption \(R_3 = 0\).
Proposition.
The following arithmetic functions are all computable:

1. projection \( f(x, y) \equiv x \)
2. constant \( f(x) \equiv n \)
3. truncated subtraction \( x - y \equiv \begin{cases} x - y, & \text{if } y \leq x \\ 0, & \text{if } y > x \end{cases} \)
4. integer division \( x \div y \equiv \begin{cases} \text{integer part of } x/y, & \text{if } y > 0 \\ 0, & \text{if } y = 0 \end{cases} \)
5. integer remainder \( x \mod y \equiv x - y \cdot (x \div y) \)
6. exponentiation (base 2) \( f(x) = 2^x \)
7. \( \log_{base 2} \) \( \log_2(x) \equiv \begin{cases} \text{greatest } y \text{ s.t. } 2^y \leq x, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \end{cases} \)

Proof - left as an exercise in register machine programming!

One can solve these kinds of problems in a more systematic way by compiling algorithms for the functions written using higher-level control constructs into register machine language.

First note that there is no loss of generality (i.e. the class of computable functions remains unchanged) if we work with register machine programs with exactly one \texttt{HALT} instruction (because...). So graphs of programs look schematically like

```
  START → M → (HALT)
```

Then we have...
Sequential composition $M_1; M_2$

IF $R = 0$ THEN $M_1$ ELSE $M_2$

WHILE $R \neq 0$ DO $M$