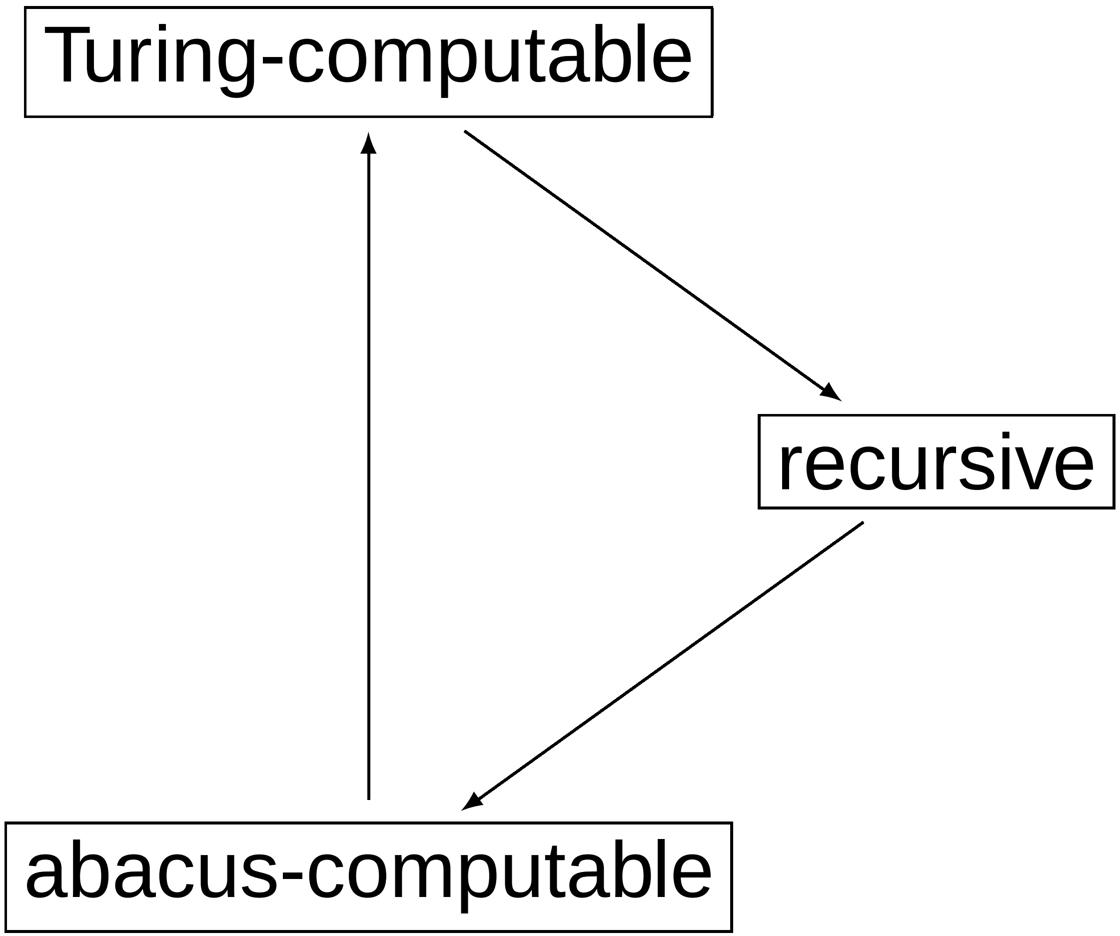
About the exam

- **I will hand out example exam questions next week.**
- Also relevant: the exercises on my slides.
- Also worth looking at: exercises in Boolos/Burgess/Jeffrey.
- It is worth looking at the exams of the last two years (Prof. Pym): enter

  `http://www.bath.ac.uk/library/exampapers/search.html`

  and search for “comp0020”. Ca. half of the questions are relevant for this year’s exam. My exam will have fewer essay-style questions, more calculations.

Consequences of the circle



Universal function

Definition. A $(k+1)$-place recursive function F is called a **universal function** if, for every k-place Turing-computable function g, there is an m such that

$$F(m, x) = g(x),$$

where x stands for $x_1, \ldots, x_k$.

Universal function

Theorem. A universal function exists.

Proof. If g is computed by the TM with code m, then $g(x)$ is equal to the function

$$F(m,x) = value(conf(m,x,halt(m,x))).$$

defined in the last lecture.

A universal Turing machine

- Let U be the TM that computes our universal function $F(m, x)$.
- So, for every TM M with code m, instead of running M on x, we can run U on (m, x).
- U is called a **universal Turing machine** (first discovered by Turing in 1937/38, before the age of general-purpose computers).
- This was the first theoretical assurance that a general-purpose computer could be designed that could mimic **any** special-purpose computer.

Kleene's normal form theorem

Theorem. Every recursive function can be obtained from the basic functions (zero, successor, projections) by composition, primitive recursion, and **at most one use of minimization**.

Assuming Church's thesis, this means that "every effectively computable function requires not more than one while-loop".

Proof

Proof. Let g be a recursive function, and let m be the code of the TM that computes g. So we have

$$g(x) = F(m, x) = value(conf(m, x, halt(m, x))).$$

The functions $value$ and $conf$ do not involve minimization. The function $halt$ is defined by minimization over the function $stdh$, and the latter does not involve minimization.

Stability under perturbation

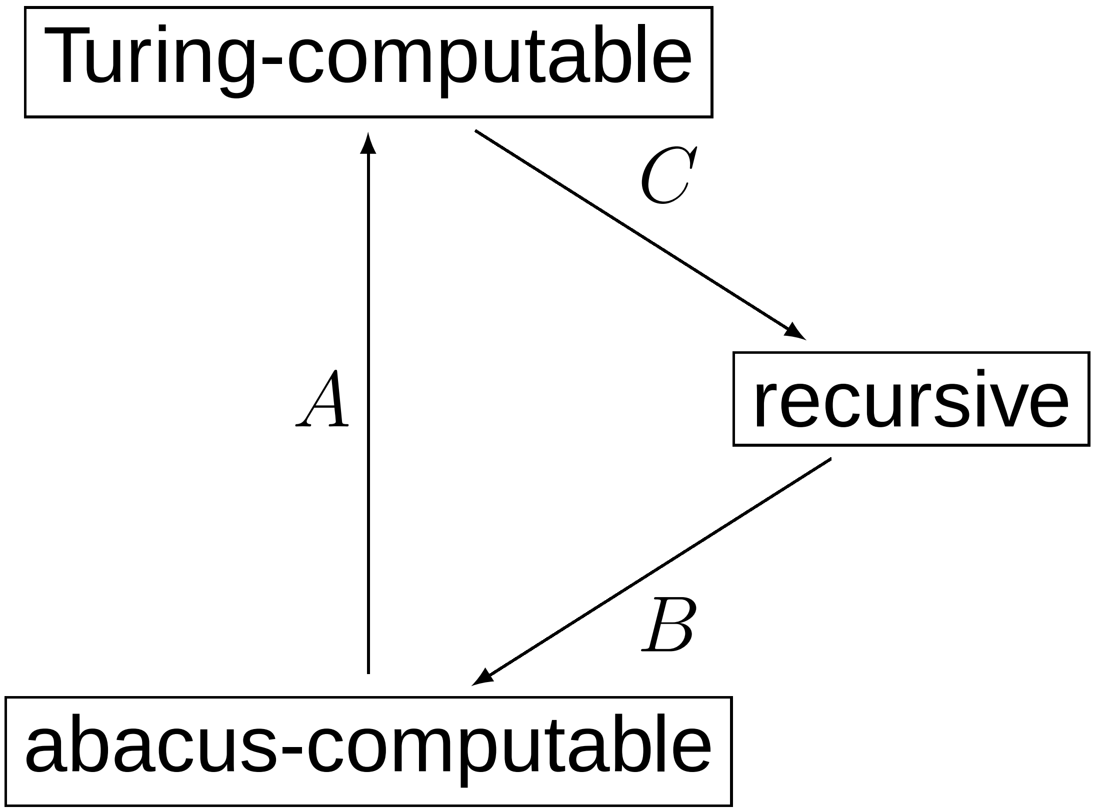
Theorem. The same functions are Turing computable whether one defines TM's to have

1. a tape infinite in both directions or in only one direction;
2. only two symbols (0 and 1) or a greater finite number of symbols that can be on the tape;
3. a two-dimensional grid or an ordinary tape.

One says Turing machines **are stable under perturbation of definition**. This is typical for a natural class of objects.

Proof (part 1/2)

To understand the stability theorem, recall the cycle of simulations:



Proof (part 2/2)

1. The Turing machine used in (A) to simulate an abacus machine never needs to go left of the starting square.
2. Minor changes in the coding using in (C) show that we can cope with any finite number of tape symbols.
3. Showing this is trickier and beyond this lecture.

Semi-recursive relations

(= recursively enumerable relations)

Basic idea

Intuitively, a set A is called **semi-decidable** if there is a Turing machine (or abacus machine or recursive function…) that

- halts if $x \in A$, and
- does not halt otherwise.

Semi-recursive relations

Here is the technical version of that basic idea:

Definition. A relation R is called **semi-recursive** if and only if it is the domain of some recursive function f—that is, if

$$R(x) \quad \text{iff} \quad f(x) \text{ is defined.}$$

Example

The set E of even numbers is semi-recursive. To see this, consider the function

$$f(x) = \begin{cases} 1 & \text{if } rem(x, 2) = 0 \\ undefined & \text{otherwise.} \end{cases}$$

Because rem is recursive and we use definition by cases, f is recursive. The set E is semi-recursive, because it is the domain of f.

Recursive vs. semi-recursive

Every recursive relation is semi-recursive. To see this, let R be any relation, and define

$$f(x) = \begin{cases} 1 & \text{if } R(x) \\ \text{undefined} & \text{if not } R(x). \end{cases}$$

This is definition by cases, so f is a recursive function. And $x \in R$ iff x is in the domain of f.

Recursive vs. semi-recursive

For any m, let f_m be the function computed by the TM with code m. Recall the relation $self$, which we defined as follows

$$self(x) \quad \text{iff} \quad f_x(x) \text{ is defined.}$$

This relation $self$ is not recursive (as shown earlier), but semi-recursive.

Proof. $self$ is the domain of $F(x, x)$, where F is the universal function.

Semi-recursive relations and $\exists$

Proposition. Let R by a k-place relation. The following are equivalent:

1. R is semi-recursive;
2. for some recursive $(k+1)$-place relation S,

$$R(x) \text{ iff } \exists t.S(x,t).$$

Semi-recursive relations and $\exists$

Proposition. Let $S(x, y)$ be a semi-recursive $(k+1)$-place relation, and let R be the k-place relation R is given by

$$R(x) \quad \text{iff} \quad \exists y. S(x, y)$$

Then R too is semi-recursive.

Proof. See Boolos/Burgess/Jeffrey.

Recursively enumerable sets

Definition. A set A is called **recursively enumerable** if it is the range of a recursive function $f : N \to N$.

(So a subset A of N is recursively enumerable if it is enumerable, and the enumeration function is recursive.)

Example

The set E of even numbers is enumerable, because it is the range of the recursive function

$$g(x) = 2 \cdot x.$$

Alternatively, we could use the enumeration

$$f(x) = \begin{cases} x & \text{if } rem(x,2) = 0 \\ undefined & \text{otherwise.} \end{cases}$$

(This corresponds to a “list with holes”.)

Theorem

Theorem. Let R be 1-place relation on the natural numbers. The following are equivalent:

1. R is semi-recursive;
2. R is the empty set, or recursively enumerable by a **total** recursive function;
3. R is recursively enumerable.