Theory of Computation

Thomas Zeugmann

Hokkaido University
Laboratory for Algorithmics

http://www-alg.ist.hokudai.ac.jp/~thomas/ToC/

Lecture 11: Models of Computation

Motivation I

Looking roughly 100 years back, mathematics faced the problem that there have been problems for which nobody could find an algorithm solving them.

Motivation I

Looking roughly 100 years back, mathematics faced the problem that there have been problems for which nobody could find an algorithm solving them.

One of the famous examples is Hilbert's tenth problem, formulated as

Design an algorithm deciding whether a given Diophantine equation has an integral solution.

Motivation II

Finally, the idea emerged that there may be problems which cannot be solved algorithmically.

Motivation II

Finally, the idea emerged that there may be problems which cannot be solved algorithmically.

Let us first see, how this idea could emerge. As a matter of fact, without Cantor's work, it would have been much more difficult.

Motivation III

Looking at all the algorithms we know, we can say that an algorithm is a computation method having the following properties.

(1) The instruction is a finite text.

(2) The computation is done step by step, where each step performs an elementary operation.

(3) In each step of the execution of the computation it is uniquely determined which elementary operation we have to perform.

(4) The next computation step depends only on the input and the intermediate results computed so far.

Motivation III

Looking at all the algorithms we know, we can say that an algorithm is a computation method having the following properties.

(1) The instruction is a finite text.

(2) The computation is done step by step, where each step performs an elementary operation.

(3) In each step of the execution of the computation it is uniquely determined which elementary operation we have to perform.

(4) The next computation step depends only on the input and the intermediate results computed so far.

Motivation III

Looking at all the algorithms we know, we can say that an algorithm is a computation method having the following properties.

(1) The instruction is a finite text.

(2) The computation is done step by step, where each step performs an elementary operation.

(3) In each step of the execution of the computation it is uniquely determined which elementary operation we have to perform.

(4) The next computation step depends only on the input and the intermediate results computed so far.

Motivation III

Looking at all the algorithms we know, we can say that an algorithm is a computation method having the following properties.

(1) The instruction is a finite text.

(2) The computation is done step by step, where each step performs an elementary operation.

(3) In each step of the execution of the computation it is uniquely determined which elementary operation we have to perform.

(4) The next computation step depends only on the input and the intermediate results computed so far.

Motivation III

Looking at all the algorithms we know, we can say that an algorithm is a computation method having the following properties.

(1) The instruction is a finite text.

(2) The computation is done step by step, where each step performs an elementary operation.

(3) In each step of the execution of the computation it is uniquely determined which elementary operation we have to perform.

(4) The next computation step depends only on the input and the intermediate results computed so far.

Motivation IV

Now, we can also assume that there is a finite alphabet Σ such that every algorithm can be represented as a string from Σ^*. Since the number of all strings from Σ^* is *countably infinite* there are at most *countably infinite* many algorithms.

Motivation IV

Now, we can also assume that there is a finite alphabet Σ such that every algorithm can be represented as a string from Σ^*. Since the number of all strings from Σ^* is *countably infinite* there are at most *countably infinite* many algorithms.

Recalling Cantor's famous result that

$$\{f \mid f\colon \mathbb{N} \to \{0,1\}\}$$

is *uncountably infinite*, we directly arrive at the following theorem.

Motivation IV

Now, we can also assume that there is a finite alphabet Σ such that every algorithm can be represented as a string from Σ^*. Since the number of all strings from Σ^* is *countably infinite* there are at most *countably infinite* many algorithms.

Recalling Cantor's famous result that

$$\{f \mid f\colon \mathbb{N} \to \{0,1\}\}$$

is *uncountably infinite*, we directly arrive at the following theorem.

Theorem 1

There exists a noncomputable function $f\colon \mathbb{N} \to \{0,1\}$.

Motivation V

While this result is of fundamental epistemological importance, it is telling nothing about any particular function. For achieving results in this regard, we have to do much more. Thus, modern computation theory starts with the question

Which problems can be solved algorithmically ?

Motivation V

While this result is of fundamental epistemological importance, it is telling nothing about any particular function. For achieving results in this regard, we have to do much more. Thus, modern computation theory starts with the question

Which problems can be solved algorithmically ?

In order to answer it, first of all, the *intuitive notion of an algorithm has to be formalized mathematically*.

Motivation V

While this result is of fundamental epistemological importance, it is telling nothing about any particular function. For achieving results in this regard, we have to do much more. Thus, modern computation theory starts with the question

Which problems can be solved algorithmically ?

In order to answer it, first of all, the *intuitive notion of an algorithm has to be formalized mathematically*.

Within this course, we shall study Gödel's and Turing's formalization; i.e., partial recursive functions and Turing machines, respectively.

Defining Partial Recursive Functions I

For all $n \in \mathbb{N}^+$ we write $\mathcal{P}^n$ to denote the set of all partial recursive functions from $\mathbb{N}^n$ into $\mathbb{N}$. Here we define $\mathbb{N}^1 = \mathbb{N}$ and $\mathbb{N}^{n+1} = \mathbb{N}^n \times \mathbb{N}$, i.e., $\mathbb{N}^n$ is the set of all ordered n-tuples of natural numbers.

Gödel's idea to define the set $\mathcal{P}$ of all partial recursive functions is as follows:

Step (1): Define some basic functions which are intuitively computable.

Step (2): Define some rules that can be used to construct new computable functions from functions that are already known to be computable.

Defining Partial Recursive Functions I

For all $n \in \mathbb{N}^+$ we write $\mathcal{P}^n$ to denote the set of all partial recursive functions from $\mathbb{N}^n$ into $\mathbb{N}$. Here we define $\mathbb{N}^1 = \mathbb{N}$ and $\mathbb{N}^{n+1} = \mathbb{N}^n \times \mathbb{N}$, i.e., $\mathbb{N}^n$ is the set of all ordered n-tuples of natural numbers.

Gödel's idea to define the set $\mathcal{P}$ of all partial recursive functions is as follows:

Step (1): Define some basic functions which are intuitively computable.

Step (2): Define some rules that can be used to construct new computable functions from functions that are already known to be computable.

Defining Partial Recursive Functions I

For all $n \in \mathbb{N}^+$ we write $\mathcal{P}^n$ to denote the set of all partial recursive functions from $\mathbb{N}^n$ into $\mathbb{N}$. Here we define $\mathbb{N}^1 = \mathbb{N}$ and $\mathbb{N}^{n+1} = \mathbb{N}^n \times \mathbb{N}$, i.e., $\mathbb{N}^n$ is the set of all ordered n-tuples of natural numbers.

Gödel's idea to define the set $\mathcal{P}$ of all partial recursive functions is as follows:

Step (1): Define some basic functions which are intuitively computable.

Step (2): Define some rules that can be used to construct new computable functions from functions that are already known to be computable.

Defining Partial Recursive Functions II

In order to complete Step (1), we define the following functions $Z,\ S,\ V\colon \mathbb{N} \to \mathbb{N}$ by setting:

$$
\begin{aligned}
Z(n) &=_{df} 0 \quad \text{for all } n \in \mathbb{N},\\
S(n) &=_{df} n+1 \quad \text{for all } n \in \mathbb{N},\\
V(n) &=_{df} \begin{cases} 0, & \text{if } n = 0\,;\\ n-1, & \text{for all } n \geqslant 1\,. \end{cases}
\end{aligned}
$$

That is, Z is the constant 0 function, S is the successor function and V is the predecessor function. Clearly, these functions are intuitively computable. Therefore, by definition we have $Z,\ S,\ V \in \mathcal{P}^1$. This completes Step (1) of the outline given above.

Defining Partial Recursive Functions III

Next, we define the rules (cf. Step (2)).

Defining Partial Recursive Functions III

Next, we define the rules (cf. Step (2)).

(2.1) (Introduction of fictitious variables)
Let $n \in \mathbb{N}^+$; then we have: if $\tau \in \mathcal{P}^n$ and
$\psi(x_1, \dots, x_n, x_{n+1}) =_{df} \tau(x_1, \dots, x_n)$, then $\psi \in \mathcal{P}^{n+1}$.

Defining Partial Recursive Functions III

Next, we define the rules (cf. Step (2)).

(2.1) (Introduction of fictitious variables)
Let $n \in \mathbb{N}^+$; then we have: if $\tau \in \mathcal{P}^n$ and
$\psi(x_1, \ldots, x_n, x_{n+1}) =_{df} \tau(x_1, \ldots, x_n)$, then $\psi \in \mathcal{P}^{n+1}$.

(2.2) (Identifying variables)
Let $n \in \mathbb{N}^+$; then we have: if $\tau \in \mathcal{P}^{n+1}$ and
$\psi(x_1, \ldots, x_n) =_{df} \tau(x_1, \ldots, x_n, x_n)$, then $\psi \in \mathcal{P}^n$.

Defining Partial Recursive Functions III

Next, we define the rules (cf. Step (2)).

(2.1) (Introduction of fictitious variables)
Let $n \in \mathbb{N}^+$; then we have: if $\tau \in \mathcal{P}^n$ and $\psi(x_1, \ldots, x_n, x_{n+1}) =_{df} \tau(x_1, \ldots, x_n)$, then $\psi \in \mathcal{P}^{n+1}$.

(2.2) (Identifying variables)
Let $n \in \mathbb{N}^+$; then we have: if $\tau \in \mathcal{P}^{n+1}$ and $\psi(x_1, \ldots, x_n) =_{df} \tau(x_1, \ldots, x_n, x_n)$, then $\psi \in \mathcal{P}^n$.

(2.3) (Permuting variables) Let $n \in \mathbb{N}^+$, $n \geqslant 2$ and let $i \in \{1, \ldots, n\}$; then we have: if $\tau \in \mathcal{P}^n$ and $\psi(x_1, \ldots, x_i, x_{i+1}, \ldots, x_n) =_{df} \tau(x_1, \ldots, x_{i+1}, x_i, \ldots, x_n)$, then $\psi \in \mathcal{P}^n$.

Defining Partial Recursive Functions IV

(2.4) **(Composition)**

Let $n \in \mathbb{N}$ and $m \in \mathbb{N}^+$. Furthermore, let $\tau \in \mathcal{P}^{n+1}$, let $\psi \in \mathcal{P}^m$ and define

$$\phi(x_1, \ldots, x_n, y_1, \ldots, y_m) =_{df} \tau(x_1, \ldots, x_n, \psi(y_1, \ldots, y_m)).$$

Then $\phi \in \mathcal{P}^{n+m}$.

Defining Partial Recursive Functions IV

(2.4) (Composition)
Let $n \in \mathbb{N}$ and $m \in \mathbb{N}^+$. Furthermore, let $\tau \in \mathcal{P}^{n+1}$, let $\psi \in \mathcal{P}^m$ and define
$\phi(x_1, \ldots, x_n, y_1, \ldots, y_m) =_{df} \tau(x_1, \ldots, x_n, \psi(y_1, \ldots, y_m))$.
Then $\phi \in \mathcal{P}^{n+m}$.

(2.5) (Primitive recursion)
Let $n \in \mathbb{N}$, let $\tau \in \mathcal{P}^n$ and let $\psi \in \mathcal{P}^{n+2}$. Then we have: if

$$\begin{aligned} \phi(x_1, \ldots, x_n, 0) &=_{df} \tau(x_1, \ldots, x_n); \\ \phi(x_1, \ldots, x_n, y+1) &=_{df} \psi(x_1, \ldots, x_n, y, \phi(x_1, \ldots, x_n, y)), \end{aligned}$$

then $\phi \in \mathcal{P}^{n+1}$.

Defining Partial Recursive Functions V

(2.6) (μ-recursion)
Let $n \in \mathbb{N}^+$; then we have: if $\tau \in \mathcal{P}^{n+1}$ and
$\psi(x_1, \ldots, x_n) =_{df} \mu y[\tau(x_1, \ldots, x_n, y) = 1]$

$$=_{df} \begin{cases} \text{the smallest } y \text{ such that} & \\ (1)\ \ \tau(x_1, \ldots, x_n, v) \quad \text{is defined} & \text{for all } v \leqslant y; \\ (2)\ \ \tau(x_1, \ldots, x_n, v) \neq 1 & \text{for all } v < y; \\ (3)\ \ \tau(x_1, \ldots, x_n, y) = 1\,, & \text{if such a } y \text{ exists;} \\ \text{not defined}\,, & \text{otherwise;} \end{cases}$$

then $\psi \in \mathcal{P}^n$.

Dedekind's Justification Theorem I

Note that all operations given above except Operation (2.5) are explicit. Operation (2.5) itself constitutes an *implicit* definition, since ϕ appears on both the left and right hand side. Thus, before we can continue, we need to verify whether or not Operation (2.5) does *always* defines a function. This is by no means obvious. Recall that every implicit definition needs a justification.

Therefore, we have to show the following theorem:

Dedekind's Justification Theorem II

Theorem 2

If τ and ψ are functions, then there is precisely one function ϕ satisfying the scheme given in Operation (2.5).

Dedekind's Justification Theorem II

Theorem 2

If τ and ψ are functions, then there is precisely one function ϕ satisfying the scheme given in Operation (2.5).

Proof. We have to show uniqueness and existence of ϕ.

Dedekind's Justification Theorem III

Claim 1. *There is at most one function ϕ satisfying the scheme given in Operation (2.5).*

Dedekind's Justification Theorem III

Claim 1. There is at most one function ϕ satisfying the scheme given in Operation (2.5).

Suppose there are functions ϕ_1 and ϕ_2 satisfying the scheme given in Operation (2.5). We show by induction over y that

$$\phi_1(x_1, \ldots, x_n, y) = \phi_2(x_1, \ldots, x_n, y) \text{ for all } x_1, \ldots, x_n, y \in \mathbb{N} .$$

For the induction basis, let $y = 0$. Then we directly get for all $x_1, \ldots, x_n \in \mathbb{N}$

$$\begin{aligned} \phi_1(x_1, \ldots, x_n, 0) &= \tau(x_1, \ldots, x_n) \\ &= \phi_2(x_1, \ldots, x_n, 0) . \end{aligned}$$

Dedekind's Justification Theorem IV

Now, we assume as induction hypothesis (abbr. IH) that for all $x_1, \ldots, x_n \in \mathbb{N}$ and some $y \in \mathbb{N}$

$$\phi_1(x_1, \ldots, x_n, y) = \phi_2(x_1, \ldots, x_n, y) \,.$$

The induction step is done from y to $y + 1$. Using the scheme provided in Operation (2.5) we obtain

$$\begin{aligned}
\phi_1(x_1, \ldots, x_n, y + 1) &= \psi(x_1, \ldots, x_n, y, \phi_1(x_1, \ldots, x_n, y)) \\
&\quad \text{by def.} \\
&= \psi(x_1, \ldots, x_n, y, \phi_2(x_1, \ldots, x_n, y)) \\
&\quad \text{by the IH} \\
&= \phi_2(x_1, \ldots, x_n, y + 1) \quad \text{by def.} \,.
\end{aligned}$$

Consequently $\phi_1 = \phi_2$, and Claim 1 is proved.

Dedekind's Justification Theorem V

Claim 2. There is a function ϕ satisfying the scheme given in Operation (2.5).

For showing the existence of ϕ we replace the inductive and implicit definition of ϕ by an infinite sequence of explicit definitions; i.e., let

$$\phi_0(x_1, \ldots, x_n, y) =_{df} \begin{cases} \tau(x_1, \ldots, x_n), & \text{if } y = 0\,; \\ \text{not defined}, & \text{otherwise}\,. \end{cases}$$

$$\phi_1(x_1, \ldots, x_n, y) =_{df} \begin{cases} \phi_0(x_1, \ldots, x_n, y), & \text{if } y < 1\,; \\ \psi(x_1, \ldots, x_n, 0, \phi_0(x_1, \ldots, x_n, 0)), & \text{if } y = 1\,; \\ \text{not defined}, & \text{otherwise}\,. \end{cases}$$

$$\ldots$$

Dedekind's Justification Theorem VI

$$\phi_{i+1}(x_1,\ldots,x_n,y) =_{df} \begin{cases} \phi_i(x_1,\ldots,x_n,y), & \text{if } y < i+1\,; \\ \psi(x_1,\ldots,x_n,i,\phi_i(x_1,\ldots,x_n,i)), & \text{if } y = i+1\,; \\ \text{not defined}, & \text{otherwise}\,. \end{cases}$$

$$\ldots$$

All definitions of the functions ϕ_i are *explicit*, and thus the functions ϕ_i exist by the set forming axiom. Consequently, for $y \in \mathbb{N}$ and every $x_1,\ldots,x_n \in \mathbb{N}$ the function ϕ defined by

$$\phi(x_1,\ldots,x_n,y) =_{df} \phi_y(x_1,\ldots,x_n,y)$$

does exist.

Dedekind's Justification Theorem VII

Furthermore, by construction we directly get

$$
\begin{aligned}
\phi(x_1, \ldots, x_n, 0) &= \phi_0(x_1, \ldots, x_n, 0) \\
&= \tau(x_1, \ldots, x_n) \quad \text{and} \\
\phi(x_1, \ldots, x_n, y+1) &= \phi_{y+1}(x_1, \ldots, x_n, y+1) \\
&= \psi(x_1, \ldots, x_n, y, \phi_y(x_1, \ldots, x_n, y)) \\
&= \psi(x_1, \ldots, x_n, y, \phi(x_1, \ldots, x_n, y)) \,,
\end{aligned}
$$

and thus, ϕ is satisfying the scheme given in Operation (2.5). ∎

Defining Partial Recursive Functions VI

Definition 1

We define the class $\mathcal{P}$ of all *partial recursive functions* to be the smallest function class containing the functions Z, S and V and all functions that can be obtained from Z, S and V by finitely many applications of the Operations (2.1) through (2.6).

Defining Partial Recursive Functions VI

Definition 1

We define the class $\mathcal{P}$ of all ***partial recursive functions*** to be the smallest function class containing the functions Z, S and V and all functions that can be obtained from Z, S and V by finitely many applications of the Operations (2.1) through (2.6).

Furthermore, we define the important subclass of primitive recursive functions as follows.

Definition 2

We define the class *Prim* of all ***primitive recursive functions*** to be the smallest function class containing the functions Z, S and V and all functions that can be obtained from Z, S and V by finitely many applications of the Operations (2.1) through (2.5).

Example 1 (Identity Function)

The identity function $I\colon \mathbb{N} \to \mathbb{N}$ *defined by* $I(x) = x$ *for all* $x \in \mathbb{N}$ *is primitive recursive.*

Proof. We want to apply Operation (2.4). Let $n = 0$ and $m = 1$. By our definition (cf. Step (1)), we know that $V, S \in \mathcal{P}^1$. So, V serves as the τ (note that $n + 1 = 0 + 1 = 1$) and S serves as the ψ in Operation (2.4) (note that $m = 1$). Consequently, the desired function I is the ϕ in Operation (2.4) (note that $n + m = 0 + 1 = 1$) and we can set

$$I(x) =_{df} V(S(x)) .$$

Hence, the identity function I is primitive recursive.

Example 2 (Binary Addition)

The binary addition function $\alpha\colon \mathbb{N}\times\mathbb{N}\to\mathbb{N}$ *given by* $\alpha(n,m) = n+m$ *for all* $n, m \in \mathbb{N}$ *is is primitive recursive.*

Proof. By assumption, $S \in \mathcal{P}$. As shown in Example 1, $I \in \mathit{Prim}$. First, we define some auxiliary functions by using the operations indicated below.

$$\begin{aligned}
\psi(x_1,x_2) &=_{df} S(x_1) && \text{by using Operation (2.1);}\\
\tilde{\psi}(x_1,x_2) &=_{df} \psi(x_2,x_1) && \text{by using Operation (2.3);}\\
\tau(x_1,x_2,x_3) &=_{df} \tilde{\psi}(x_1,x_2) && \text{by using Operation (2.1);}\\
\tilde{\tau}(x_1,x_2,x_3) &=_{df} \tau(x_1,x_3,x_2) && \text{by using Operation (2.3)}\,.
\end{aligned}$$

Now, we apply Operation (2.5) for defining α, i.e., we set

$$\begin{aligned}
\alpha(n,0) &=_{df} I(n),\\
\alpha(n,m+1) &=_{df} \tilde{\tau}(n,m,\alpha(n,m))\,.
\end{aligned}$$

Binary Addition II

Since we only used Operations (2.1) through (2.5), we see that $\alpha \in \mathit{Prim}$.

Binary Addition II

Since we only used Operations (2.1) through (2.5), we see that $\alpha \in \mathit{Prim}$.

So, let us compute $\alpha(n, 1)$. Then we get

$$
\begin{aligned}
\alpha(n,1) &= \alpha(n, 0+1) = \tilde{\tau}(n, 0, \alpha(n,0)) \\
&= \tilde{\tau}(n, 0, I(n)) \quad \text{by using } \alpha(n,0) = I(n)\,, \\
&= \tilde{\tau}(n, 0, n) \quad \text{by using } I(n) = n\,, \\
&= \tau(n, n, 0) \quad \text{by using the definition of } \tilde{\tau}\,, \\
&= \tilde{\psi}(n, n) \quad \text{by using the definition of } \tau\,, \\
&= \psi(n, n) \quad \text{by using the definition of } \tilde{\psi}\,, \\
&= S(n) = n + 1 \quad \text{by using the definition of } \psi \text{ and } S\,.
\end{aligned}
$$

Binary Addition III

So, our definition *may* look more complex than necessary. In order to see, it is not, we compute $\alpha(n, 2)$.

$$\begin{aligned}
\alpha(n,2) &= \alpha(n,1+1) = \tilde{\tau}(n,1,\alpha(n,1)) \\
&= \tilde{\tau}(n,1,n+1) \quad \text{by using } \alpha(n,1) = n+1\,, \\
&= \tau(n,n+1,1) \\
&= \tilde{\psi}(n,n+1) \\
&= \psi(n+1,n) \\
&= S(n+1) = n+2\,.
\end{aligned}$$

Binary Multiplication

In the following we shall often omit some of the tedious technical steps. For example, in order to clarify that binary multiplication is primitive recursive, we simply point out that is suffices to set

$$\begin{aligned} m(x, 0) &= Z(x)\,, \\ m(x, y+1) &= \alpha(x, m(x, y))\,. \end{aligned}$$

Binary Multiplication

In the following we shall often omit some of the tedious technical steps. For example, in order to clarify that binary multiplication is primitive recursive, we simply point out that is suffices to set

$$\begin{aligned} m(x,0) &= Z(x)\,, \\ m(x,y+1) &= \alpha(x,m(x,y))\,. \end{aligned}$$

Also note that the constant 1 function c is primitive recursive; i.e., $c(n) = 1$ for all $n \in \mathbb{N}$. For seeing this, we set

$$\begin{aligned} c(0) &= S(0)\,, \\ c(n+1) &= c(n)\,. \end{aligned}$$

In the following, instead of $c(n)$ we just write 1.

Signum and Arithmetic Difference

Now, it is easy to see that the signum function sg is primitive recursive, since we have

$$\begin{aligned} sg(0) &= 0\,, \\ sg(n+1) &= 1\,. \end{aligned}$$

Signum and Arithmetic Difference

Now, it is easy to see that the signum function sg is primitive recursive, since we have

$$\begin{aligned} sg(0) &= 0\,, \\ sg(n+1) &= 1\,. \end{aligned}$$

Since the natural numbers are not closed under subtraction, one conventionally uses the so-called arithmetic difference defined as $m \dot{-} n = m - n$ if $m \geqslant n$ and 0 otherwise. The arithmetic difference is primitive recursive, too, since for all $n, m \in \mathbb{N}$ we have

$$\begin{aligned} m \dot{-} 0 &= I(m)\,, \\ m \dot{-} (n+1) &= V(m \dot{-} n)\,. \end{aligned}$$

Generalizations of the examples given so far are in the book.

Case Distinctions I

Quite often one is defining functions by making case distinctions (cf., e.g., our definition of the predecessor function V). So, it is only natural to ask under what circumstances definitions by case distinctions do preserve primitive recursiveness. A convenient way to describe properties is the usage of *predicates*. An n-ary predicate p over the natural numbers is a subset of $\mathbb{N}^n$. Usually, one writes $p(x_1, \dots, x_n)$ instead of $(x_1, \dots, x_n) \in p$. The characteristic function of n-ary predicate p is the function $\chi_p \colon \mathbb{N}^n \to \{0, 1\}$ defined by

$$\chi_p(x_1, \dots, x_n) = \begin{cases} 1, & \text{if } p(x_1, \dots, x_n)\,; \\ 0, & \text{otherwise}\,. \end{cases}$$

Case Distinctions II

Definition 3

A predicate p is said to be ***primitive recursive*** if χ_p is primitive recursive.

Case Distinctions II

Definition 3

A predicate p is said to be ***primitive recursive*** if χ_p is primitive recursive.

Definition 4

Let p, q be n-ary predicates, then we define $p \wedge q$ to be the set $p \cap q$, $p \vee q$ to be the set $p \cup q$, and $\neg p$ to be the set $\mathbb{N}^n \setminus p$.

Case Distinctions III

Lemma 3

Let p, q *be any primitive recursive* n*-ary predicates. Then* $p \wedge q$, $p \vee q$, *and* $\neg p$ *are also primitive recursive.*

Case Distinctions III

Lemma 3

Let p, q *be any primitive recursive* n*-ary predicates. Then* $p \wedge q$*,* $p \vee q$*, and* $\neg p$ *are also primitive recursive.*

Proof. Obviously, it holds

$$\begin{aligned}
\chi_{p \wedge q}(x_1, \ldots, x_n) &= \chi_p(x_1, \ldots, x_n) \cdot \chi_q(x_1, \ldots, x_n)\,, \\
\chi_{p \vee q}(x_1, \ldots, x_n) &= sg(\chi_p(x_1, \ldots, x_n) + \chi_q(x_1, \ldots, x_n))\,, \\
\chi_{\neg p}(x_1, \ldots, x_n) &= 1 \dot{-} \chi_p(x_1, \ldots, x_n)\,.
\end{aligned}$$

Since we already know addition, multiplication and the arithmetic difference to be primitive recursive, the assertion of the lemma follows. ∎

Case Distinctions IV

Now, we can show our theorem concerning function definitions by making case distinctions.

Theorem 4

Let $p_1, \ldots, p_k$ be pairwise disjoint n-ary primitive recursive predicates, and let $\psi_1, \ldots, \psi_k \in \mathcal{P}^n$ be primitive recursive functions. Then the function $\gamma \colon \mathbb{N}^n \to \mathbb{N}$ defined by

$$\gamma(x_1, \ldots, x_n) =_{df} \begin{cases} \psi_1(x_1, \ldots, x_n), & \text{if } p_1(x_1, \ldots, x_n)\,; \\ \cdot & \\ \cdot & \\ \cdot & \\ \psi_k(x_1, \ldots, x_n), & \text{if } p_k(x_1, \ldots, x_n)\,; \\ 0, & \text{otherwise}\,; \end{cases}$$

is primitive recursive.

Case Distinctions V

Proof. Since we can write γ as

$$\gamma(x_1,\ldots,x_n) = \sum_{i=1}^{k} \chi_{p_i}(x_1,\ldots,x_n)\cdot\psi_i(x_1,\ldots,x_n)\,,$$

the theorem follows from the primitive recursiveness of general addition and multiplication. ∎

Pairing Functions I

Quite often it would be very useful to have a bijection from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. So, first we have to ask whether or not such a bijection does exist.

Pairing Functions I

Quite often it would be very useful to have a bijection from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. So, first we have to ask whether or not such a bijection does exist.

This is indeed the case.

Recall that the elements of $\mathbb{N} \times \mathbb{N}$ are ordered pairs of natural numbers. So, we may easily represent all elements of $\mathbb{N} \times \mathbb{N}$ in a two dimensional array, where row x contains all pairs (x, y), i.e., having x in the first component and $y = 0, 1, 2, \ldots$ (cf. Figure 1).

Pairing Functions II

$$
\begin{array}{cccccc}
(0,0) & (0,1) & (0,2) & (0,3) & (0,4) & \dots \\
(1,0) & (1,1) & (1,2) & (1,3) & (1,4) & \dots \\
(2,0) & (2,1) & (2,2) & (2,3) & (2,4) & \dots \\
(3,0) & (3,1) & (3,2) & (3,3) & (3,4) & \dots \\
(4,0) & (4,1) & (4,2) & (4,3) & (4,4) & \dots \\
(5,0) & \dots & & & & \\
\dots & \dots & & & &
\end{array}
$$

Figure 1: A two dimensional array representing $\mathbb{N} \times \mathbb{N}$.

Pairing Functions III

Now, the idea is to arrange all these pairs in a sequence starting

$$(0,0),(0,1),(1,0),(0,2),(1,1),(2,0),(0,3),(1,2),(2,1),(3,0),\ldots \tag{1}$$

In this order, all pairs (x,y) appear before all pairs (x',y') if and only if $x+y < x'+y'$. That is, they are arranged in order of incrementally growing component sums. The pairs with the same component sum are ordered by the first component starting with the smallest one. That is, pair (x,y) is located in the segment

$$(0,x+y),\quad (1,x+y-1),\quad \ldots,\quad (x,y),\quad \ldots,\quad (x+y,0)\,.$$

Note that there are $x+y+1$ many pairs having the component sum $x+y$. Thus, in front of pair $(0,x+y)$ we have in the Sequence (1) $x+y$ many segments containing a total of

$$1+2+3+\cdots+(x+y)$$

many pairs.

Pairing Functions IV

Taking into account that

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} = \sum_{i=1}^{n} i$$

we thus can define the desired bijection $c\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ by setting

$$\begin{aligned} c(x,y) &= \frac{(x+y)(x+y+1)}{2} + x \\ &= \frac{(x+y)^2 + 3x + y}{2} . \end{aligned} \tag{2}$$

Pairing Functions V

All the operations involved in computing c have already been shown to be primitive recursive, thus we can conclude that c is primitive recursive, too.

Pairing Functions V

All the operations involved in computing c have already been shown to be primitive recursive, thus we can conclude that c is primitive recursive, too.

Exercise. *Determine the functions* d_1 *and* d_2 *such that for all* $x, y \in \mathbb{N}$, *if* $z = c(x, y)$ *then* $x = d_1(z)$ *and* $y = d_2(z)$.

Pairing Functions V

All the operations involved in computing c have already been shown to be primitive recursive, thus we can conclude that c is primitive recursive, too.

Exercise. *Determine the functions* d_1 *and* d_2 *such that for all* $x, y \in \mathbb{N}$, *if* $z = c(x, y)$ *then* $x = d_1(z)$ *and* $y = d_2(z)$.

Exercise. *Show that for every fixed* $k \in \mathbb{N}$, $k > 2$, *there is a primitive recursive bijection* $c_k \colon \mathbb{N}^k \to \mathbb{N}$.

Pairing Functions V

All the operations involved in computing c have already been shown to be primitive recursive, thus we can conclude that c is primitive recursive, too.

Exercise. *Determine the functions* d_1 *and* d_2 *such that for all* $x, y \in \mathbb{N}$*, if* $z = c(x, y)$ *then* $x = d_1(z)$ *and* $y = d_2(z)$.

Exercise. *Show that for every fixed* $k \in \mathbb{N}$, $k > 2$*, there is a primitive recursive bijection* $c_k \colon \mathbb{N}^k \to \mathbb{N}$.

Exercise. *Let* $\mathbb{N}^*$ *be the set of all finite sequences of natural numbers. Show that there is a primitive recursive bijection* $c_* \colon \mathbb{N}^* \to \mathbb{N}$.

General Recursive Functions

Next, we define the class of *general recursive functions*.

Definition 5

For all $n \in \mathbb{N}^+$ we define $\mathcal{R}^n$ to be the set of all functions $f \in \mathcal{P}^n$ such that $f(x_1, \ldots, x_n)$ is defined for all $x_1, \ldots, x_n \in \mathbb{N}$. Furthermore, we set $\mathcal{R} = \bigcup_{n \in \mathbb{N}^+} \mathcal{R}^n$.

In other words, $\mathcal{R}$ is the set of all functions that are total and partial recursive. Now, we can show the following theorem:

General Recursive Functions

Next, we define the class of *general recursive functions*.

Definition 5

For all $n \in \mathbb{N}^+$ we define $\mathcal{R}^n$ to be the set of all functions $f \in \mathcal{P}^n$ such that $f(x_1, \ldots, x_n)$ is defined for all $x_1, \ldots, x_n \in \mathbb{N}$. Furthermore, we set $\mathcal{R} = \bigcup_{n \in \mathbb{N}^+} \mathcal{R}^n$.

In other words, $\mathcal{R}$ is the set of all functions that are total and partial recursive. Now, we can show the following theorem:

Theorem 5

$Prim \subset \mathcal{R} \subset \mathcal{P}$.

Proof I

Proof. Clearly $Z, S, V \in \mathcal{R}$. Furthermore, after a bit of reflection it should be obvious that any finite number of applications of Operations (2.1) through (2.5) results only in total functions. This shows $Prim \subseteq \mathcal{R}$.

Also, $\mathcal{R} \subseteq \mathcal{P}$ is obvious by definition. So, it remains to show that the two inclusions are *proper*.

Proof I

Proof. Clearly Z, S, $V \in \mathcal{R}$. Furthermore, after a bit of reflection it should be obvious that any finite number of applications of Operations (2.1) through (2.5) results only in total functions. This shows $\mathit{Prim} \subseteq \mathcal{R}$.

Also, $\mathcal{R} \subseteq \mathcal{P}$ is obvious by definition. So, it remains to show that the two inclusions are *proper*.

Claim 1. $\mathcal{P} \setminus \mathcal{R} \neq \emptyset$.

By definition, $S \in \mathcal{P}$ and using Operation (2.4) it is easy to see that $\delta(n) =_{df} S(S(n))$ is in $\mathcal{P}$, too. Now, note that $\delta(n) = n + 2 > 1$ for all $n \in \mathbb{N}$.

Using Operation (2.1) we define $\tau(x, y) = \delta(y)$, and thus $\tau \in \mathcal{P}$. Consequently,

$$\psi(x) = \mu y[\tau(x, y) = 1]$$

is the nowhere defined function and hence $\psi \notin \mathcal{R}$. On the other hand, by construction $\psi \in \mathcal{P}$. Therefore, we get $\psi \in \mathcal{P} \setminus \mathcal{R}$.

Proof II

Claim 2. $\mathcal{R} \setminus Prim \neq \emptyset$.

Showing this claim is much more complicated. First, we define a function

$$
\begin{aligned}
ap(0, m) &=_{df} m + 1 \,, \\
ap(n + 1, 0) &=_{df} ap(n, 1) \,, \\
ap(n + 1, m + 1) &=_{df} ap(n, ap(n + 1, m)) \,,
\end{aligned}
$$

which is the so-called Ackermann-Péter function. Hilbert conjectured in 1926 that every total and computable function is also primitive recursive. This conjecture was disproved by Ackermann in 1928 and Péter simplified Ackermann's definition in 1955.

Proof III

Now, it suffices to show that function ap is *not* primitive recursive and that function ap is general recursive. Both parts are not easy to prove. So, due to the lack of time, we must skip some parts. But before we start, let us confine ourselves that the function ap is *intuitively* computable. For doing this, consider the following fragment of pseudo-code implementing the function ap as `peter`.

Proof IV

```
function peter(n, m)
             if n = 0
               return m + 1
             else if m = 0
               return peter(n - 1, 1)
             else
               return peter(n - 1, peter(n, m - 1))
```

Proof V

Next, we sketch the proof that ap cannot be primitive recursive. First, for every primitive recursive function ϕ, one defines a function f_ϕ as follows. Let k be the arity of ϕ; then we set

$$f_\phi(n) = \max\left\{\phi(x_1, \ldots, x_k) \mid \sum_{i=1}^{k} x_i \leqslant n\right\} .$$

Then, by using the inductive construction of the class *Prim*, one can show by structural induction that for every primitive recursive function ϕ there is a number $n_\phi \in \mathbb{N}$ such that

$$f_\phi(n) < ap(n_\phi, n) \quad \text{for all } n \geqslant n_\phi .$$

Intuitively, the latter statement shows that the Ackermann-Péter function grows faster than every primitive recursive function.

Proof VI

The rest is then easy. Suppose $\mathrm{ap} \in Prim$. Then, taking into account that the identity function I is primitive recursive, one directly sees by application of Operation (2.4) that

$$\kappa(n) = \mathrm{ap}(I(n), I(n))$$

is primitive recursive, too. Now, for κ there is a number $n_\kappa \in \mathbb{N}$ such that

$$f_\kappa(n) < \mathrm{ap}(n_\kappa, n) \quad \text{for all } n \geqslant n_\kappa\,.$$

But now,

$$\kappa(n_\kappa) \leqslant f_\kappa(n_\kappa) \ < \ \mathrm{ap}(n_\kappa, n_\kappa) = \kappa(n_\kappa)\ ,$$

a contradiction.

Proof VII

For the second part, one has to prove that ap $\in \mathcal{R}$ which mainly means to provide a construction to express the function ap using the Operations (2.1) through (2.5) *and* the μ-operator. We refer the interested reader to Hermes. $\blacksquare$

Thank you!