5. The Primitive Recursive Functions

- URM and TM based on universal programming languages.
- In this and the next section we introduce a third model of computation.
- It is given as a set of partial functions
  - basic functions
  - by using certain operations.
- First proposed by Gödel and Kleene 1936.
- Best model for showing that functions are computable.
- In this section we introduce the primitive-recursive functions, which form a subset of the partial-recursive functions.

Overview

(a) Introduction of primitive recursive functions.
  - Will be total.
  - Includes all functions which can be computed realistically, and many more.
  - But not all computable functions are primitive recursive.

(b) Closure Properties of the primitive rec. functions
  - We will show that the set of primitive recursive functions is a reach set of functions, closed under many operations.

Def. Prim. Rec. Functions

Inductive definition of the primitive recursive functions

\( f : \mathbb{N}^k \to \mathbb{N} \).

The following basic Functions are primitive recursive:
- \( \text{zero} : \mathbb{N} \to \mathbb{N} \),
- \( \text{succ} : \mathbb{N} \to \mathbb{N} \),
- \( \text{proj}_i : \mathbb{N}^k \to \mathbb{N} \ (0 \leq i < k) \).

Remember that these functions have defining equations
- \( \text{zero}(n) = 0 \),
- \( \text{succ}(n) = n + 1 \),
- \( \text{proj}_i(a_0, \ldots, a_{k-1}) = a_i. \)
Def. Prim. Rec. Functions

If

- \( g : \mathbb{N}^n \to \mathbb{N} \),
- \( h : \mathbb{N}^{n+2} \to \mathbb{N} \) are primitive recursive,

so is the function \( f : \mathbb{N}^{n+1} \to \mathbb{N} \) defined by primitive recursion from \( g, h \).

Remember that \( f \) is defined by

- \( f(\vec{x}, 0) = g(\vec{x}) \),
- \( f(\vec{x}, n + 1) = h(\vec{x}, n, f(\vec{x}, n)) \).

\( f \) is denoted by \( \text{primrec}(g, h) \).

Inductively Defined Sets

That the set of primitive recursive functions is inductively defined means:

- It is the least set
  - containing basic functions
  - and closed under the operations.
- Or: It is the set generated by the above.
- Or: The primitive recursive functions are those we can write as terms formed
  - from zero, \( \text{succ} \), \( \text{proj}^n \),
  - using composition \( \circ \)
  - i.e. by forming from \( f, g \), \( f \circ (g_0, \ldots, g_{n-1}) \)
  - and \( \text{primrec} \).

E.g.

- \( \text{primrec}(\text{proj}_1^1, \text{succ} \circ \text{proj}_2^3) : \mathbb{N}^2 \to \mathbb{N} \) is prim. rec.
- \( \text{primrec}(0, \text{proj}_1^2) : \mathbb{N} \to \mathbb{N} \) is prim. rec.
Primitive Rec. Relations and Sets

- A relation $R \subseteq \mathbb{N}^n$ is **primitive recursive**, if
  \[ \chi_R : \mathbb{N}^n \to \mathbb{N} \]
  is primitive recursive.
- Note that we identified a set $A \subseteq \mathbb{N}^n$ with the relation $R \subseteq \mathbb{N}^n$ given by
  \[ R(\vec{x}) : \iff \vec{x} \in A \]
  Therefore a set $A \subseteq \mathbb{N}^n$ is primitive recursive if the corresponding relation $R$ is.

Remark

- Unless demanded explicitly, for showing that $f$ is defined by the principle of primitive recursion (i.e. by `primrec`), it suffices to express:
  - $f(\vec{x}, 0)$ as an expression built from
    - previously defined prim. rec. functions,
    - $\vec{x}$,
    - the recursion argument $y$,
    - the recursion hypothesis $f(\vec{x}, y)$,
    - and constants.
  **Example:**
  \[ f(x_0, x_1, 0) = (x_0 + x_1) \cdot 3 . \]
  (Assuming that $+, \cdot$ have already been shown to be primitive recursive).

Remark

- $f(\vec{x}, y + 1)$ as an expression built from
  - previously defined prim. rec. functions,
  - $\vec{x}$,
  - the recursion argument $y$,
  - the recursion hypothesis $f(\vec{x}, y)$,
  - and constants.
  **Example:**
  \[ f(x_0, x_1, y + 1) = (x_0 + x_1 + y + f(x_0, x_1, y)) \cdot 3 . \]
  (Assuming that $+, \cdot$ have already been shown to be primitive recursive).

Remark

- Similarly, for showing $f$ is prim. rec. by using previously defined functions using composition, it suffices to express $f(\vec{x})$ in terms of
  - previously defined prim. rec. functions,
  - parameters $\vec{x}$
  - and constants.
  **Example:**
  \[ f(x, y, z) = (x + y) \cdot 3 + z . \]
  (Assuming that $+, \cdot$ have already been shown to be primitive recursive).
  When looking at the first examples, we will express primitive recursive functions directly by using the basic functions, `primrec` and $\circ$. 
Identity Function

\[ \text{id} : \mathbb{N} \rightarrow \mathbb{N}, \quad \text{id}(n) = n \text{ is primitive recursive:} \]

- \[ \text{id} = \text{proj}_0^1 : \mathbb{N}^1 \rightarrow \mathbb{N}, \]
- \[ \text{proj}_0^1(n) = n = \text{id}(n). \]

Constant Function

\[ \text{const}_n : \mathbb{N} \rightarrow \mathbb{N}, \quad \text{const}_n(k) = n \text{ is primitive recursive:} \]

\[ \text{const}_n = \text{succ} \circ \cdots \circ \text{succ} \circ \text{zero} : \]
\[ n \text{ times} \]

\[ \text{succ} \circ \cdots \circ \text{succ} \circ \text{zero}(k) = \text{succ}(\text{succ}(\cdots \text{succ}(\text{zero}(k)))) \]
\[ n \text{ times} \]
\[ = \text{succ}(\text{succ}(\cdots \text{succ}(0))) \]
\[ n \text{ times} \]
\[ = 0 + 1 + 1 \cdots + 1 \]
\[ n \text{ times} \]
\[ = n \]
\[ = \text{const}_n(k). \]

Addition

\[ \text{add} : \mathbb{N}^2 \rightarrow \mathbb{N}, \quad \text{add}(x, y) = x + y \text{ is primitive recursive.} \]

We have the laws:

\[ \text{add}(x, 0) = x + 0 = x \]
\[ \text{add}(x, y + 1) = x + (y + 1) = (x + y) + 1 = \text{add}(x, y) + 1 \]

\[ \text{add}(x, 0) = g(x), \]
where
\[ g : \mathbb{N} \rightarrow \mathbb{N}, \quad g(x) = x, \]
i.e. \[ g = \text{id} = \text{proj}_0^1. \]
**Addition**

\[
\begin{align*}
\text{add}(x, 0) &= x + 0 = g(x), \\
\text{add}(x, y + 1) &= \text{add}(x, y) + 1.
\end{align*}
\]

\[\text{add}(x, y + 1) = h(x, y, \text{add}(x, y)),\]

where

\[h : \mathbb{N}^3 \rightarrow \mathbb{N}, h(x, y, z) := z + 1.\]

\[h = \text{succ} \circ \text{proj}_2^3: \]

\[(\text{succ} \circ \text{proj}_2^3)(x, y, z) = \text{succ} (\text{proj}_2^3(x, y, z))
= \text{succ}(z)
= z + 1
= h(x, y, z).\]

Therefore

\[\text{add} = \text{primrec}(\text{proj}_0^1, \text{succ} \circ \text{proj}_2^3).\]

---

**Multiplication**

\[\text{mult} : \mathbb{N}^2 \rightarrow \mathbb{N}, \text{mult}(x, y) = x \cdot y\]

is primitive recursive.

We have the laws:

\[
\begin{align*}
\text{mult}(x, 0) &= x \cdot 0 = 0, \\
\text{mult}(x, y + 1) &= x \cdot (y + 1)
= x \cdot y + x
= \text{mult}(x, y) + x
= \text{add}(\text{mult}(x, y), x).
\end{align*}
\]

Jump over rest

---

**Multiplication**

\[
\begin{align*}
\text{mult}(x, 0) &= 0, \\
\text{mult}(x, y + 1) &= \text{add}(\text{mult}(x, y), x).
\end{align*}
\]

\[\text{mult}(x, 0) = g(x), \text{where } g : \mathbb{N} \rightarrow \mathbb{N}, g(x) = 0,\]

i.e. \( g = \text{zero}, \)
Multiplication

\[
\text{mult}(x, 0) = 0 = g(x) , \\
\text{mult}(x, y + 1) = \text{add} (\text{mult}(x, y), x) .
\]

\[
\text{mult}(x, y + 1) = h(x, y, \text{mult}(x, y)), \\
\text{where} \\
h : \mathbb{N}^3 \to \mathbb{N}, h(x, y, z) := \text{add}(z, x). \\
h = \text{add} \circ (\text{proj}_2^3, \text{proj}_0^3): \\
(\text{add} \circ (\text{proj}_2^3, \text{proj}_0^3))(x, y, z) = \text{add}(\text{proj}_2^3(x, y, z), \text{proj}_0^3(x, y, z)) \\
= \text{add}(z, x) \\
= h(x, y, z) .
\]

Therefore

\[
\text{mult} = \text{primrec}(\text{zero}, \text{add} \circ (\text{proj}_2^3, \text{proj}_0^3)) .
\]

Predecessor Function

\[
\text{pred} \text{ is prim. rec.:} \\
pred(0) = 0 , \\
pred(x + 1) = x .
\]

Subtraction

\[
\text{sub}(x, y) = x - y \text{ is prim. rec.:} \\
\text{sub}(x, 0) = x , \\
\text{sub}(x, y + 1) = x - (y + 1) = (x - y) - 1 = \text{pred}(\text{sub}(x, y)) .
\]
**Signum Function**

- \( \text{sig} : \mathbb{N} \rightarrow \mathbb{N}, \)
  \[
  \text{sig}(x) := \begin{cases} 
  1, & \text{if } x > 0, \\
  0, & \text{if } x = 0
  \end{cases}
  \]

  is prim. rec.:

  \( \text{sig}(x) = x \div (x \div 1): \)

  For \( x = 0 \) we have
  \[
  x \div (x \div 1) = 0 \div (0 \div 1) = 0 \div 0 = 0 = \text{sig}(x).
  \]

  For \( x > 0 \) we have
  \[
  x \div (x \div 1) = x - (x - 1) = x - x + 1 = 1 = \text{sig}(x).
  \]

- Note that
  \[ \text{sig} = \chi_{x>0} \]
  where \( x > 0 \) stands for the unary predicate, which is true for \( x \) iff \( x > 0: \)
  \[
  \chi_{x>0}(y) = \begin{cases} 
  1, & \text{if } y > 0, \\
  0, & \text{if } y = 0.
  \end{cases} = \text{sig}(y)
  \]

**Add., Mult., Exp.**

- Consider the sequence of definitions of addition, multiplication, exponentiation:
  - **Addition:**
    \[
    n + 0 = n, \\
    n + (m + 1) = (n + m) + 1.
    \]
    Therefore, if we write \(((+) 1)\) for the function \( \mathbb{N} \rightarrow \mathbb{N}, \)
    \[
    ((+) 1)(n) = n + 1, \text{ then}
    \]
    \[
    n + m = ((+) 1)^m(n).
    \]
Remark on Notation

The notation \((+(+) 1)^m(n)\) is to be understood as follows:

Let \(f\) be a function (e.g. \((+(+) 1)\)). Then we define

\[ f^n(m) := f(f(\cdots f(m)\cdots)) \]

\(n\) times

This is not to be confused with exponentiation

\[ n^m = n \cdot \cdots \cdot n \]

\(n\) times

So

\[ \((+(+) 1)^m(n) = \underbrace{((+(+) 1)((+(+) 1)(\cdots (((+(+) 1)(n)\cdots)))}_{m \text{ times}} \]

\[ = \underbrace{\cdots ((m+1)+1)\cdots +1}_{m \text{ times}} = m + n \]

Add., Mult., Exp.

Exponentiation:

\[ n^0 = 1, \]

\[ n^{m+1} = (n^m) \cdot n, \]

Therefore, if we write \((n)\) for the function \(\mathbb{N} \to \mathbb{N} \]

\[ (n)(m) = n \cdot m, \]

then

\[ n^m = ((n)^m)^{(1)}. \]

Note that above, we have both occurrences of \(n^m\) for exponentation and of \((n)^m(1)\) for iterated function application.

Superexponentiation

Extend this sequence further, by defining

Superexponentiation:

\[ \text{superexp}(n, 0) = 1, \]

\[ \text{superexp}(n, m + 1) = n^{\text{superexp}(n,m)}, \]

Therefore, if we write \((\uparrow n)\) for the function \(\mathbb{N} \to \mathbb{N} \]

\[ (\uparrow n)(k) = n^k, \]

then

\[ \text{superexp}(n, m) = ((\uparrow n)^m)^{(1)}. \]
Supersuperexponentiation

Supersuperexponentiation:
\[
\text{supersuperexp}(n, 0) = 1, \\
\text{supersuperexp}(n, m + 1) = \text{superexp}(n, \text{supersuperexp}(n, m)),
\]
Etc.
One obtains sequence of extremely fast growing functions.
These functions will exhaust the primitive recursive functions.
We will reconsider this sequence at the beginning of Sect. 6 (a).

(b) Closure of the Prim. Rec. Func.

Closure under \(\cup, \cap, \setminus\)

If \(R, S \subseteq \mathbb{N}^n\) are prim. rec., so are
\(R \cup S,\)
\(R \cap S,\)
\(\mathbb{N}^n \setminus R.\)

Closure under Prop. Connectives

Note:
\[
(R \cup S)(\vec{x}) \Leftrightarrow R(\vec{x}) \lor S(\vec{x}),
\]
\[
(R \cap S)(\vec{x}) \Leftrightarrow R(\vec{x}) \land S(\vec{x}),
\]
\[
(\mathbb{N}^n \setminus R)(\vec{x}) \Leftrightarrow \neg R(\vec{x}).
\]
So the prim. rec. predicates are closed under the propositional connectives \(\land, \lor, \neg.\)

Example:
Above we have seen that “\(x < y\)” is primitive recursive.
Therefore the predicates “\(x \leq y\)” and “\(x = y\)” are primitive recursive:
\[
x \leq y \Leftrightarrow \neg(y < x).
\]
\[
x = y \Leftrightarrow x \leq y \land y \leq x.
\]

Proof of \((R \cup S)(\vec{x}) \Leftrightarrow R(\vec{x}) \lor S(\vec{x})\):
\[
\begin{align*}
(R \cup S)(\vec{x}) & \Leftrightarrow \vec{x} \in R \cup S \\
& \Leftrightarrow \vec{x} \in R \lor \vec{x} \in S \\
& \Leftrightarrow R(\vec{x}) \lor S(\vec{x})
\end{align*}
\]

Jump over Rest

Proof of \((R \cap S)(\vec{x}) \Leftrightarrow R(\vec{x}) \land S(\vec{x})\):
\[
\begin{align*}
(R \cap S)(\vec{x}) & \Leftrightarrow \vec{x} \in R \cap S \\
& \Leftrightarrow \vec{x} \in R \land \vec{x} \in S \\
& \Leftrightarrow R(\vec{x}) \land S(\vec{x})
\end{align*}
\]
Closure under $\cup$, $\cap$, $\setminus$

**Proof of Closure under $\cap$**

Proof of $(\mathbb{N}^n \setminus R)(\vec{x}) \iff \neg R(\vec{x})$:

$(\mathbb{N}^n \setminus R)(\vec{x}) \iff \vec{x} \in (\mathbb{N}^n \setminus R)$

$\iff \vec{x} \notin R$

$\iff \neg R(\vec{x})$

---

**Proof of Closure under $\cup$**

**Similarly, if $S(\vec{x})$ holds, then**

$\operatorname{sig}(\chi_R(\vec{x}) + \chi_S(\vec{x})) = 1 = \chi_{R \cup S}(\vec{x})$

If neither $R(\vec{x})$ nor $S(\vec{x})$ holds, then we have

$\operatorname{sig}(\chi_R(\vec{x}) + \chi_S(\vec{x})) = 0 = \chi_{R \cup S}(\vec{x})$.
Proof of Closure under $\cap$

$\chi_{R \cap S}(\vec{x}) = \chi_R(\vec{x}) \cdot \chi_S(\vec{x})$
(and therefore $R \cap S$ is primitive recursive):

Jump over Rest of Proof

If $R(\vec{x})$ and $S(\vec{x})$ hold, then

$$\chi_R(\vec{x}) \cdot \chi_S(\vec{x}) = 1 = \chi_{R \cap S}(\vec{x}).$$

Proof of Closure under $\setminus$

$\chi_{N^n \setminus R}(\vec{x}) = 1 - \chi_R(\vec{x})$
(and therefore primitive recursive):

Jump over Rest of Proof

If $R(\vec{x})$ holds, then $\chi_R(\vec{x}) = 1$, therefore

$$1 - \chi_R(\vec{x}) = 1 = \chi_{N^n \setminus R}(\vec{x}).$$

If $R(\vec{x})$ does not hold, then $\chi_R(\vec{x}) = 0$, therefore

$$1 - \chi_R(\vec{x}) = 1 = \chi_{N^n \setminus R}(\vec{x}).$$

Definition by Cases

The primitive recursive functions are closed under definition by cases:

Assume

- $g_1, g_2 : \mathbb{N}^n \to \mathbb{N}$ are primitive recursive,
- $R \subseteq \mathbb{N}^n$ is primitive recursive.

Then $f : \mathbb{N}^n \to \mathbb{N}$,

$$f(\vec{x}) := \begin{cases} g_1(\vec{x}), & \text{if } R(\vec{x}), \\ g_2(\vec{x}), & \text{if } \neg R(\vec{x}), \end{cases}$$

is primitive recursive.
Definition by Cases

\[
f(\vec{x}) := \begin{cases} 
  g_1(\vec{x}), & \text{if } R(\vec{x}), \\
  g_2(\vec{x}), & \text{if } \neg R(\vec{x}), 
\end{cases}
\]

Jump over rest of proof.

If \( R(\vec{x}) \) holds, then \( \chi_R(\vec{x}) = 1 \),
\( \chi_{\mathbb{N}^n \setminus R}(\vec{x}) = 0 \), therefore
\[
g_1(\vec{x}) \cdot \chi_R(\vec{x}) + g_2(\vec{x}) \cdot \chi_{\mathbb{N}^n \setminus R}(\vec{x}) = g_1(\vec{x}) = f(\vec{x})
\]

If \( \neg R(\vec{x}) \) holds,
then \( \chi_R(\vec{x}) = 0 \), \( \chi_{\mathbb{N}^n \setminus R}(\vec{x}) = 1 \),
\[
g_1(\vec{x}) \cdot \chi_R(\vec{x}) + g_2(\vec{x}) \cdot \chi_{\mathbb{N}^n \setminus R}(\vec{x}) = g_2(\vec{x}) = f(\vec{x})
\]

Bounded Sums

If \( g : \mathbb{N}^{n+1} \rightarrow \mathbb{N} \) is prim. rec., so is
\[
f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}, \quad f(\vec{x}, y) := \sum_{z<y} g(\vec{x}, z)
\]
where
\[
\sum_{z<0} g(\vec{x}, z) := 0
\]
and for \( y > 0 \),
\[
\sum_{z<y} g(\vec{x}, z) := g(\vec{x}, 0) + g(\vec{x}, 1) + \cdots + g(\vec{x}, y-1)
\]
Example

We have above

\[
\begin{align*}
  f(\vec{x}, 0) &= g(\vec{x}, 0) \\
  f(\vec{x}, 1) &= g(\vec{x}, 0) + g(\vec{x}, 1) \\
  f(\vec{x}, 2) &= g(\vec{x}, 0) + g(\vec{x}, 1) + g(\vec{x}, 2) \\

e tc. 
\end{align*}
\]

Bounded Products

If \( g : \mathbb{N}^{n+1} \to \mathbb{N} \) is prim. rec., so is

\[
f : \mathbb{N}^{n+1} \to \mathbb{N}, \quad f(\vec{x}, y) := \prod_{z < y} g(\vec{x}, z) ,
\]

where

\[
\prod_{z < 0} g(\vec{x}, z) := 1 ,
\]

and for \( y > 0 \),

\[
\prod_{z < y} g(\vec{x}, z) := g(\vec{x}, 0) \cdot g(\vec{x}, 1) \cdots g(\vec{x}, y - 1) .
\]

Omit Proof

Example for closure under bounded products:

\( f : \mathbb{N} \to \mathbb{N}, \)

\[
f(n) := n! = 1 \cdot 2 \cdots n
\]

(\( f(0) = 0! = 1 \)),

is primitive recursive, since

\[
f(n) = \prod_{i < n} (i + 1) = \prod_{i < n} g(i) ,
\]

where \( g(i) := i + 1 \) is prim. rec..

(Note that in the special case \( n = 0 \) we have

\[
f(0) = 0! = 1 = \prod_{i < 0} (i + 1) .
\]
Remark on Factorial Function

Alternatively, the factorial function can be defined directly by using primitive recursion as follows:

\[
\begin{align*}
0! & = 1 \\
(n + 1)! & = n! \cdot (n + 1)
\end{align*}
\]

Bounded Quantification

If \( R \subseteq \mathbb{N}^{n+1} \) is prim. rec., so are

\[
R_1(\vec{x}, y) : \iff \forall z < y. R(\vec{x}, z)
\]

\[
R_2(\vec{x}, y) : \iff \exists z < y. R(\vec{x}, z)
\]

Proof for \( R_1 \):

\[
\chi_{R_1}(\vec{x}, y) = \prod_{z < y} \chi_{R}(\vec{x}, z)
\]

Jump over details.

- If \( \forall z < y. R(\vec{x}, z) \) holds, then \( \forall z < y. \chi_{R}(\vec{x}, z) = 1 \), therefore

\[
\prod_{z < y} \chi_{R}(\vec{x}, y) = \prod_{z < y} 1 = 1 = \chi_{R_1}(\vec{x}, y)
\]

Bounded Quantification

If \( \neg R(\vec{x}, z) \) for one \( z < y \), then \( \chi_{R}(\vec{x}, z) = 0 \), therefore

\[
\prod_{z < y} \chi_{R}(\vec{x}, y) = 0 = \chi_{R_1}(\vec{x}, y)
\]
Bounded Quantification

\[ R_2(\vec{x}, y) :\equiv \exists z < y.R(\vec{x}, z) . \]

**Proof for \( R_2 \):**

\[ \chi_{R_2}(\vec{x}, y) = \text{sig}(\sum_{z<y} \chi_R(\vec{x}, z)) : \]

Jump over Rest of Proof

- If \( \forall z < y. \neg R(\vec{x}, z) \), then

\[ \text{sig}(\sum_{z<y} \chi_R(\vec{x}, y)) = \text{sig}(\sum_{z<y} 0) = \text{sig}(0) = 0 = \chi_{R_2}(\vec{x}, y) . \]

Bounded Search

If \( R \subseteq \mathbb{N}^{n+1} \) is a prim. rec. predicate, so is

\[ f(\vec{x}, y) := \mu z < y.R(\vec{x}, z) , \]

where

\[ \mu z < y.R(\vec{x}, z) := \begin{cases} 
\text{the least } z \text{ s.t. } R(\vec{x}, z) \text{ holds, if such } z \text{ exists,} \\
y \text{otherwise.}
\end{cases} \]

**Bounded Search**

\[ f(\vec{x}, y) := \mu z < y.R(\vec{x}, z) \]

**Proof:**

Define

\[ Q(\vec{x}, y) :\equiv R(\vec{x}, y) \land \forall z < y.\neg R(\vec{x}, z) , \]

\[ Q'(\vec{x}, y) :\equiv \forall z < y.\neg R(\vec{x}, z) \]

\( Q \) and \( Q' \) are primitive recursive.

\( Q(\vec{x}, y) \) holds, if \( y \) is minimal s.t. \( R(\vec{x}, y) \).

We show

\[ f(\vec{x}, y) = \left( \sum_{z<y} \chi_Q(\vec{x}, z) \cdot z \right) + \chi_{Q'}(\vec{x}, y) \cdot y . \]

Jump over details.
Bounded Search

\[ Q(\vec{x}, y) : \Leftrightarrow R(\vec{x}, y) \land \forall z < y. \neg R(\vec{x}, z) , \]
\[ Q'(\vec{x}, y) : \Leftrightarrow \forall z < y. \neg R(\vec{x}, z) , \]
Show \[ f(\vec{x}, y) = (\sum_{z<y} \chi_{Q(\vec{x}, z)} \cdot z) + \chi_{Q'(\vec{x}, y)} \cdot y . \]

Assume \( \exists z < y. R(\vec{x}, z) \).
Let \( z \) be minimal s.t. \( R(\vec{x}, z) \).
\[ \Rightarrow Q(\vec{x}, z) , \]
\[ \Rightarrow \chi_{Q(\vec{x}, z)} \cdot z = z . \]

For \( z \neq z' \) we have \( \neg Q(\vec{x}, z') \),
therefore \( \chi_{Q(\vec{x}, z')} \cdot z' = 0 (z' \neq z) \).
Furthermore, \( \neg Q'(\vec{x}, y) \), therefore \( \chi_{Q'(\vec{x}, y)} \cdot y = 0 . \)
Therefore
\[ (\sum_{z<y} \chi_{Q(\vec{x}, z)} \cdot z) + \chi_{Q'(\vec{x}, y)} \cdot y = z = \mu z' < y. R(\vec{x}, z') . \]

Bounded Search

\[ Q(\vec{x}, y) : \Leftrightarrow R(\vec{x}, y) \land \forall z < y. \neg R(\vec{x}, z) , \]
\[ Q'(\vec{x}, y) : \Leftrightarrow \forall z < y. \neg R(\vec{x}, z) , \]
Show \[ f(\vec{x}, y) = (\sum_{z<y} \chi_{Q(\vec{x}, z)} \cdot z) + \chi_{Q'(\vec{x}, y)} \cdot y . \]

Assume \( \forall z < y. \neg R(\vec{x}, z) \).
\[ \Rightarrow \neg Q(\vec{x}, z) \text{ for } z < y , \]
\[ \Rightarrow \forall z < y. \chi_{Q(\vec{x}, z)} \cdot z = 0 . \]
Furthermore, \( Q'(\vec{x}, y) \),
therefore \( \chi_{Q'(\vec{x}, y)} \cdot y = y . \)
Therefore
\[ (\sum_{z<y} \chi_{Q(\vec{x}, z)} \cdot z) + \chi_{Q'(\vec{x}, y)} \cdot y = y = \mu z' < y. R(\vec{x}, z') . \]

Bounded Search

\[ f(\vec{x}, y) := \mu z < y. R(\vec{x}, z) \]

Alternatively, \( f \) can be defined by primitive recursion directly using the equations:
\[ f(\vec{x}, 0) = 0 \]
\[ f(\vec{x}, y + 1) = \begin{cases} f(\vec{x}, y) & \text{if } f(\vec{x}, y) < y , \\ y & \text{if } f(\vec{x}, y) = y \land R(\vec{x}, y) , \\ y + 1 & \text{otherwise.} \end{cases} \]

Exercise: Show \( f \) fulfills those equations.
From these equations it follows that \( f \) is primitive recursive, provided \( R \) is.

Example

Let \( P \subseteq \mathbb{N} \) be a primitive recursive predicate, and define
\[ f : \mathbb{N} \to \mathbb{N} , \]
\[ f(x) := |\{y < x \mid P(y)\}| . \]
\( f(x) \) is the number of \( y < x \) s.t. \( P(y) \) holds.
\( f \) is primitive recursive, since
\[ f(x) = \sum_{y<x} \chi_{P(y)} . \]
Example 2

Let $Q \subseteq \mathbb{N}$ be a primitive recursive predicate.

We show how to determine primitive recursively the second least $y < x$ s.t. $Q(y)$ holds.

**Step 1**: Express the property to be the second least $y < x$ s.t. $Q(y)$ holds as a prim. rec. predicate $P(y)$:

$$P(y) :\iff Q(y) \land (\exists z < y.Q(z)) \land \neg (\exists z < y.\exists z' < y.(Q(z) \land Q(z') \land z \neq z'))$$

$P(y)$ is primitive recursive, since it is defined from $Q$ using $\land$, $\neg$, bounded quantification and “$z = z$’”.

---

Example 2

**Step 2**: Let $f(y)$ be the second least $y < x$ s.t. $Q(y)$ holds:

$$f(x) = \begin{cases} y, & \text{if } y < x \text{ and } P(y), \\ x, & \text{if there is no } y < x \text{ s.t. } P(y). \end{cases}$$

Then $f(x) = \mu y < x.P(y)$

so $f$ is primitive recursive.

(We could have defined instead

$$P'(y) :\iff Q(y) \land \exists z < y.Q(z)$$

Then $f(x) = \mu y < x.P'(y)$ holds.)

---

Lemma 5.1

The following functions are primitive recursive:

(a) $\pi : \mathbb{N}^2 \to \mathbb{N}$.

(Remember, $\pi(n, m)$ encodes two natural numbers as one.)

(b) $\pi_0, \pi_1 : \mathbb{N} \to \mathbb{N}$.

(Remember $\pi_0(\pi(n, m)) = n$, $\pi_1(\pi(n, m)) = m$).

(c) $\pi^k : \mathbb{N}^k \to \mathbb{N}$ ($k \geq 1$).

(Remember $\pi^k(n_0, \ldots, n_{k-1})$ encodes the sequence $(n_0, \ldots, n_k)$.

---

Lemma 5.1

(d) $f : \mathbb{N}^3 \to \mathbb{N}$,

$$f(x, k, i) = \begin{cases} \pi^k_i(x), & \text{if } i < k, \\ x, & \text{otherwise}. \end{cases}$$

(Remember that $\pi^k_i(\pi^k(n_0, \ldots, n_{k-1})) = n_i$ for $i < k$.)

We write $\pi^k_i(a)$ for $f(x, k, i)$, even if $i \geq k$.

(e) $f_k : \mathbb{N}^k \to \mathbb{N}$,

$$f_k(x_0, \ldots, x_{k-1}) = (x_0, \ldots, x_{k-1})$$

(Remember that $(x_0, \ldots, x_{k-1})$ encodes the sequence $x_0, \ldots, x_{k-1}$ as one natural number.

(f) $lh : \mathbb{N} \to \mathbb{N}$.

(Remember that $lh((x_0, \ldots, x_{k-1})) = k$.)
Lemma 5.1

(g) \( g : \mathbb{N}^2 \rightarrow \mathbb{N}, g(x, i) = (x)_i \).
(remember that \( (x_0, \ldots, x_{k-1})_i = x_i \) for \( i < k \).)

The proof will be omitted in the lecture.
Jump over proof.

Proof of Lemma 5.1 (a), (b)

(a)

\[
\pi(n, m) = (\sum_{i \leq n+m} i) + m
\]

\[
= (\sum_{i < n+m+1} i) + m
\]

is primitive recursive.

(b) One can easily show that \( n, m \leq \pi(n, m) \).
Therefore we can define

\[
\pi_0(n) := \\mu k < n + 1.3l < n + 1.n = \pi(k, l),
\]

\[
\pi_1(n) := \\mu l < n + 1.3k < n + 1.n = \pi(k, l).
\]

Therefore \( \pi_0, \pi_1 \) are primitive recursive.

Proof of Lemma 5.1 (c)

(c) Proof by induction on \( k \):
\[
\bullet \quad k = 1: \pi^1(x) = x, \text{ so } \pi^1 \text{ is primitive recursive.}
\]
\[
\bullet \quad k \rightarrow k + 1: \text{ Assume that } \pi^k \text{ is primitive recursive.}
\]

Show that \( \pi^{k+1} \) is primitive recursive as well:

\[
\pi^{k+1}(x_0, \ldots, x_k) = \pi(\pi^k(x_0, \ldots, x_{k-1}), x_k).
\]

Therefore \( \pi^{k+1} \) is primitive recursive
(Using that \( \pi, \pi^k \) are primitive recursive).

Proof of Lemma 5.1 (d)

(d) We have

\[
\pi^1_0(x) = x,
\]

\[
\pi^1_i(x) = \pi^k_i(\pi^0(x)), \text{ if } i < k,
\]

\[
\pi^1_i(x) = \pi_1(x), \text{ if } i = k,
\]

Therefore

\[
\pi^k_i(x) = \begin{cases} 
\pi_1((\pi^0)^{k-i}(x)), & \text{if } i > 0, \\
(\pi^0)^k(x), & \text{if } i = 0.
\end{cases}
\]
Proof of Lemma 5.1 (d)

and

\[ f(x, k, i) = \begin{cases} 
  x, & \text{if } i \geq k, \\
  \pi_1((\pi_0)^{k-i}(x)), & \text{if } 0 < i < k, \\
  (\pi_0)^k(x), & \text{if } i = 0 < k. 
\end{cases} \]

Define \( g : \mathbb{N}^2 \rightarrow \mathbb{N} \),

\[
g(x, 0) := x, \\
g(x, k + 1) := \pi_0(g(x, k)),
\]

which is primitive recursive.

Proof of Lemma 5.1 (d)

Then we get \( g(x, k) = (\pi_0)^k(x) \), therefore

\[ f(x, k, i) = \begin{cases} 
  x, & \text{if } i \geq k, \\
  \pi_1(g(x, k-i)), & \text{if } 0 < i < k, \\
  g(x, k), & \text{if } i = 0 < k. 
\end{cases} \]

So \( f \) is primitive recursive.

Proof of Lemma 5.1 (e), (f), (g)

(e) \[ f_k(x_0, \ldots, x_{k-1}) = 1 + \pi(k - 1, \pi^k(x_0, \ldots, x_{k-1})) \]

is primitive recursive.

(f) \[ \text{lh}(x) = \begin{cases} 
  0, & \text{if } x = 0, \\
  \pi_0(x - 1) + 1, & \text{if } x \neq 0. 
\end{cases} \]

(g) \[
(x)_i = \pi_i^{\text{lh}(x)}(\pi_1(x - 1)) \\
= f(\pi_1(x - 1), \text{lh}(x), i)
\]

is primitive recursive.

Lemma and Definition 5.2

Prim. rec. functions as follows do exist:

(a) \( \text{snoc} : \mathbb{N}^2 \rightarrow \mathbb{N} \) s.t.

\[
\text{snoc}(\langle x_0, \ldots, x_{n-1}, x \rangle) = \langle x_0, \ldots, x_{n-1}, x \rangle.
\]

\textbf{Remark:} \( \text{snoc} \) is the word \( \text{cons} \) reversed.

\( \text{snoc} \) is like \( \text{cons} \), but adds an element to the end rather than to the beginning of a list.

(b) \( \text{last} : \mathbb{N} \rightarrow \mathbb{N} \) and \( \text{beginning} : \mathbb{N} \rightarrow \mathbb{N} \) s.t.

\[
\text{last}(\text{snoc}(x, y)) = y, \\
\text{beginning}(\text{snoc}(x, y)) = x.
\]

The proof will be omitted in the lecture.

Jump over proof.
Proof of Lemma 5.2 (a)

Define

\[ \text{snoc}(x, y) = \begin{cases} 
\langle y \rangle, & \text{if } x = 0, \\
1 + \pi(\text{lh}(x), \pi_1(x - 1), y), & \text{otherwise}, 
\end{cases} \]

so snoc is primitive recursive.

Proof of Lemma 5.2 (b)

Proof for beginning:

Define

\[ \text{beginning}(x) := \begin{cases} 
\langle \rangle, & \text{if } \text{lh}(x) \leq 1, \\
\langle(x)_0 \rangle, & \text{if } \text{lh}(x) = 2, \\
1 + \pi((\text{lh}(x) - 1) - 1, \pi_0(\pi_1(y - 1))), & \text{otherwise}. 
\end{cases} \]

Let \( x = \text{snoc}(y, z) \). Show \( \text{beginning}(x) = y \).

Case \( \text{lh}(y) = 0 \):

Then \( x = \text{snoc}(y, z) = \langle z \rangle \)

therefore \( \text{lh}(x) = 1 \), and

\[ \text{beginning}(x) = \langle \rangle = y \]
Case \( \text{lh}(y) = 1 \): Then \( y = \langle y' \rangle \) for some \( y' \), \( \text{snoc}(y, z) = \langle y', z \rangle \),

\[
\begin{align*}
\text{beginning}(x) & = \langle (x)_0 \rangle \\
& = \langle \langle y', z \rangle \rangle_0 \\
& = \langle y' \rangle \\
& = y
\end{align*}
\]

Case \( \text{lh}(y) > 1 \): Let \( \text{lh}(y) = n + 2 \),

\[
y = \langle y_0, \ldots, y_{n+1} \rangle = 1 + \pi(n + 1, \pi^{n+2}(y_0, \ldots, y_{n+1}))
\]

Then

\[
\text{snoc}(y, z) = 1 + \pi(n + 2, \pi(y - 1), z)
\]

Proof for last:
Define

\[
\text{last}(x) := (x)_{\text{lh}(x) - 1}
\]

If \( y = \langle y_0, \ldots, y_{n-1} \rangle \), then

\[
\begin{align*}
\text{last}(\text{snoc}(y, z)) & = \text{last}(\langle y_0, \ldots, y_{n-1}, z \rangle) \\
& = (\langle y_0, \ldots, y_{n-1}, z \rangle)_{\text{lh}(y_0, \ldots, y_{n-1}, z) - 1} \\
& = (\langle y_0, \ldots, y_{n-1}, z \rangle)_{n} \\
& = z
\end{align*}
\]
**Definition Course-Of-Value**

- Assume \( f : \mathbb{N}^{n+1} \to \mathbb{N} \). Then we define

\[
\overline{f}(\bar{x}, n) := \langle f(\bar{x}, 0), f(\bar{x}, 1), \ldots, f(\bar{x}, n-1) \rangle
\]

Especially \( \overline{f}(\bar{x}, 0) = \langle \rangle \).

- \( \overline{f} \) is called the course-of-value function associated with \( f \).

---

**Course-of-Value Prim. Recursion**

The prim. rec. functions are closed under **course-of-value primitive recursion**:

Assume

\[
g : \mathbb{N}^{n+2} \to \mathbb{N}
\]

is primitive recursive. Then

\[
f : \mathbb{N}^{n+1} \to \mathbb{N}
\]

\[
f(\bar{x}, k) = g(\bar{x}, k, \overline{f}(\bar{x}, k))
\]

is prim. rec.

---

**Example**

Fibonacci numbers are prim. rec.\( \text{fib} : \mathbb{N} \to \mathbb{N} \) given by:

\[
\text{fib}(0) := 1, \\
\text{fib}(1) := 1, \\
\text{fib}(n) := \text{fib}(n-1) + \text{fib}(n-2), \text{ if } n > 1
\]

Definable by course-of-value primitive recursion:

- We have

\[
\text{fib}(n) = \begin{cases} 
1 & \text{if } n \leq 1, \\
(\text{fib}(n))_{n-2} + (\text{fib}(n))_{n-1} & \text{otherwise}.
\end{cases}
\]
Proof that prim. rec. functions are closed under course-of-value primitive recursion:
Let \( f \) be defined by
\[
f(x, y) = g(x, y, \overline{f}(x, y))
\]
Show \( f \) is prim. rec.
We show first that \( \overline{f} \) is primitive recursive.

Lemma and Definition 5.3
There exists prim. rec. functions as follows:

(a) \( \text{append} : \mathbb{N}^2 \rightarrow \mathbb{N} \) s.t.
\[
\text{append}((a_0, \ldots, a_{k-1}), (b_0, \ldots, b_{l-1})) = (a_0, \ldots, a_{k-1}, b_0, \ldots, b_{l-1})
\]
We write \( n \ast m \) for \( \text{append}(n, m) \).

(b) \( \text{subst} : \mathbb{N}^3 \rightarrow \mathbb{N} \) s.t. if \( i < n \) then
\[
\text{subst}(x_0, \ldots, x_{n-1}, i, y) = x_0, \ldots, x_{i-1}, y, x_{i+1}, x_{i+2}, \ldots, x_{n-1}
\]
and if \( i \geq n \), then
\[
\text{subst}(x_0, \ldots, x_{n-1}, i, y) = x_0, \ldots, x_{n-1}
\]
We write \( x[i/y] \) for \( \text{subst}(x, i, y) \).
Lemma and Definition 5.3

(c) \( \text{subseq} : \mathbb{N}^3 \to \mathbb{N} \) s.t., if \( i < n \),

\[
\text{subseq}(\langle x_0, \ldots, x_{n-1} \rangle, i, j) = \langle x_i, x_{i+1}, \ldots, x_{\min(j-1,n-1)} \rangle ,
\]

and if \( i \geq n \),

\[
\text{subseq}(\langle x_0, \ldots, x_{n-1} \rangle, i, j) = \langle \rangle .
\]

Proof of Lemma 5.3 (a)

We have

\[
\begin{align*}
\text{append}(\langle x_0, \ldots, x_n \rangle, 0) &= \text{append}(\langle x_0, \ldots, x_n \rangle, \langle \rangle ) \\
&= \langle x_0, \ldots, x_n \rangle ,
\end{align*}
\]

and for \( m > 0 \)

\[
\begin{align*}
\text{append}(\langle x_0, \ldots, x_n \rangle, \langle y_0, \ldots, y_m \rangle) &= \langle x_0, \ldots, x_n, y_0, \ldots, y_m \rangle \\
&= \text{snoc}(\langle x_0, \ldots, x_n, y_0, \ldots, y_{m-1}, y_m \rangle) \\
&= \text{snoc}(\text{append}(\langle x_0, \ldots, x_n \rangle, \langle y_0, \ldots, y_{m-1} \rangle), y_m) \\
&= \text{snoc}(\text{append}(\langle x_0, \ldots, x_n \rangle), \\
&\quad \text{beginning}(\langle y_0, \ldots, y_m \rangle)), \\
&\quad \text{last}(\langle y_0, \ldots, y_m \rangle)) .
\end{align*}
\]

Therefore we have

\[
\begin{align*}
\text{append}(x, 0) &= x , \\
\text{append}(x, y) &= \text{snoc}(\text{append}(x, \text{beginning}(y)), \text{last}(y)) ,
\end{align*}
\]

One can see that \( \text{beginning}(x) < x \) for \( x > 0 \), therefore the last equations give a definition of \( \text{append} \) by course-of-value primitive recursion, therefore \( \text{append} \) is primitive recursive.

Lemma and Definition 5.3

(d) \( \text{half} : \mathbb{N} \to \mathbb{N} \),

s.t. \( \text{half}(n) = k \) if \( n = 2k \) or \( n = 2k + 1 \).

(e) The function \( \text{bin} : \mathbb{N} \to \mathbb{N} \), s.t.

\[
\text{bin}(n) = \langle b_0, \ldots, b_k \rangle ,
\]

for \( b_i \) in normal form (no leading zeros, unless \( n = 0 \)),

s.t. \( n = (b_0, \ldots, b_k) \)

(f) A function \( \text{bin}^{-1} : \mathbb{N} \to \mathbb{N} \), s.t.

\[
\text{bin}^{-1}(\langle b_0, \ldots, b_k \rangle) = n , \text{ if } (b_0, \ldots, b_k)_2 = n .
\]

The proof will be omitted in the lecture.

Jump over proof.
Proof of Lemma 5.3 (b)

We have

\[
\text{subst}(x, i, y) := \begin{cases} 
  x, & \text{if } \text{lh}(x) \leq i, \\
  \text{snoc}(\text{beginning}(x), y), & \text{if } i + 1 = \text{lh}(x), \\
  \text{snoc}(\text{subst}(\text{beginning}(x), i, y), \text{last}(x)) & \text{if } i + 1 < \text{lh}(x).
\end{cases}
\]

Therefore \(\text{subst}\) is definable by course-of-value primitive recursion.

Proof of Lemma 5.3 (d), (e)

(d) \(\text{half}(x) = \mu y < x.(2 \cdot y = x \lor 2 \cdot y + 1 = x)\).

(e)

\[
\text{bin}(x) = \begin{cases} 
  \langle 0 \rangle, & \text{if } x = 0, \\
  \langle 1 \rangle & \text{if } x = 1, \\
  \text{snoc}(\text{half}(x), x - (2 \cdot \text{half}(x))), & \text{if } x > 1.
\end{cases}
\]

therefore definable by course-of-value primitive recursion.

Proof of Lemma 5.3 (c)

We can define

\[
\text{subseq}(x, i, j) = \begin{cases} 
  \langle \rangle, & \text{if } i \geq \text{lh}(x), \\
  \text{subseq}(\text{beginning}(x), i, j), & \text{if } i < \text{lh}(x) \text{ and } j < \text{lh}(x), \\
  \text{snoc}(\text{subseq}(\text{beginning}(x), i, j), \text{last}(x)) & \text{if } i < \text{lh}(x) \leq j,
\end{cases}
\]

which is a definition by course-of-value primitive recursion.

Proof of Lemma 5.3 (f)

\[
\text{bin}^{-1}(x) = \begin{cases} 
  0, & \text{if } \text{lh}(x) = 0, \\
  (x)_0 & \text{if } \text{lh}(x) = 1, \\
  \text{bin}^{-1}(\text{beginning}(x)) \cdot 2 + \text{last}(x) & \text{if } \text{lh}(x) > 1,
\end{cases}
\]

therefore definable by course-of-value primitive recursion.