5. The Primitive Recursive Functions

In this module we consider **3 models of computation**.
- The **URMs**, which captures computation as it happens on a computer.
- The **Turing Machines**, which capture computation on a piece of paper.
- The **partial recursive functions**, developed in this and the next section.

Partial recursive functions were first proposed by Gödel and Kleene 1936.

There are many other models of computation.

---

### Algebraic View of Computation

**Main motivation** for partial recursive functions:
- **Algebraic view of computation**.
- The class of partial computable functions in this model is defined by certain **combinators**.
- We have some initial functions and close them under operations which form from partial computable functions new partial computable functions.
- So in this model of computation we define directly a set of functions (rather than defining first a programming language and then the functions defined by it).

---

### Primitive Recursive Functions

In this section we will first start introducing the **primitive recursive functions**.

They form an important **subclass of the partial recursive functions**.

Main property of the primitive recursive functions.
- All primitive recursive functions are **total**.
- Therefore **not all computable functions** are **primitive recursive**.
- There exists no programming language, such that all definable functions are total, which allows to define all computable functions.
Primitive Recursive Functions

- The primitive recursive functions contain all feasible functions (and many infeasible functions as well).
- Therefore all realistic functions can be defined primitive recursively.
- The principle of primitive recursion is closely related to the principle of induction.
- In the dependently typed programming language Agda induction and primitive recursion are the same principle.
- Extensions of the principle of primitive recursion form the main ingredient of many functional programming languages.

Overview

(a) Introduction of primitive recursive functions.
(b) Closure Properties of the primitive rec. functions
   - We will show that the set of primitive recursive functions is a rich set of functions, closed under many operations.
   - This will show as well extend our intuition of how powerful URM computable functions are.

Def. Prim. Rec. Functions

If
- \( f : \mathbb{N}^k \to \mathbb{N} \) is primitive recursive,
- \( g_i : \mathbb{N}^n \to \mathbb{N} \) are primitive recursive, \((i = 0, \ldots, k - 1)\),
so is
\[
\begin{align*}
f \circ (g_0, \ldots, g_{k-1}) : \mathbb{N}^n & \to \mathbb{N} .
\end{align*}
\]
Remember that \( h := f \circ (g_0, \ldots, g_{k-1}) \) is defined as
\[
\begin{align*}
h(x) &= f(g_0(x), \ldots, g_{k-1}(x)) .
\end{align*}
\]
Especially, if \( f : \mathbb{N} \to \mathbb{N} \) and \( g : \mathbb{N} \to \mathbb{N} \) are primitive recursive, so is
\[
\begin{align*}
f \circ g : \mathbb{N} & \to \mathbb{N} .
\end{align*}
\]
Def. Prim. Rec. Functions

If
- \( g : \mathbb{N}^n \rightarrow \mathbb{N} \),
- \( h : \mathbb{N}^{n+2} \rightarrow \mathbb{N} \) are primitive recursive,
so is the function \( f : \mathbb{N}^{n+1} \rightarrow \mathbb{N} \) 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 \( \text{primrec}(g, h) \).

Def. Prim. Rec. Functions

If
- \( k \in \mathbb{N} \),
- \( h : \mathbb{N}^2 \rightarrow \mathbb{N} \) is primitive recursive,
so is the function \( f : \mathbb{N} \rightarrow \mathbb{N} \), defined by primitive recursion from \( k \) and \( h \).

Remember that \( f := \text{primrec}(k, h) \) is defined by
- \( f(0) = k \),
- \( f(y + 1) = h(y, f(y)) \).

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

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, succ, \( \text{proj}^n \),
  - using composition \( \circ \),
  - i.e. by forming from \( f, g \), \( f \circ (g_0, \ldots, g_{n-1}) \),
  - and \( \text{primrec} \).

Inductively Defined Sets

E.g.
- \( \text{primrec}(\text{proj}^1, \text{succ} \circ \text{proj}^2) : \mathbb{N}^2 \rightarrow \mathbb{N} \) is prim. rec.
- \( \text{primrec}(0, \text{proj}^2) : \mathbb{N} \rightarrow \mathbb{N} \) is prim. rec.

(= addition)

(= pred)
Primitive Rec. Relations and Sets

- A relation $R \subseteq \mathbb{N}^n$ is **primitive recursive**, if
  
  $$
  \chi_R : \mathbb{N}^n \rightarrow \mathbb{N}
  $$

  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:**
  
  $$
  f(x_0, x_1, y + 1) = (x_0 + x_1 + y + f(x_0, x_1, y)) \cdot 3
  $$

  (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 $\vec{x}$
  - 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

- id : N → N, id(y) = y is primitive recursive:
- id = proj^1_0:
  proj^1_0 : N^1 → N,
  proj^1_0(y) = y = id(y).

Constant Function

- const_n : N → N, const_n(x) = n is primitive recursive:
- const_n = succ ○ · · · ○ succ ○ zero:
  n times
  succ ○ · · · ○ succ ○ zero(x) = succ(succ(· · · succ(zero(x)))))
  n times
  = succ(succ(· · · succ(0))))
  n times
  = 0 +1 + 1 · · · + 1
  n times
  = n
  = const_n(x).

Addition

- add : N^2 → N, add(x, y) = x + y is primitive recursive.
  We have the laws:
  
  \[
  \begin{align*}
  \text{add}(x, 0) &= x + 0 \\
  &= x \\
  \text{add}(x, y + 1) &= x + (y + 1) \\
  &= (x + y) + 1 \\
  &= \text{add}(x, y) + 1 \\
  &= \text{succ}(\text{add}(x, y))
  \end{align*}
  \]
  
  \[
  \begin{align*}
  \text{add}(x, 0) &= x, \\
  \text{add}(x, y + 1) &= \text{succ}(\text{add}(x, y)).
  \end{align*}
  \]

Addition

- add(x, 0) = g(x),
  where
  g : N → N, g(x) = x,
  i.e. g = id = proj^1_0.
Addition

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

where

\[ h : \mathbb{N}^3 \rightarrow \mathbb{N}, h(x, y, z) := \text{succ}(z). \]
\[ h = \text{succ} \circ \text{proj}_2^3:\]
\[ (\text{succ} \circ \text{proj}_2^3)(x, y, z) = \text{succ}(\text{proj}_2^3(x, y, z)) = \text{succ}(z) = h(x, y, z). \]

Multiplication

\[ \text{mult} : \mathbb{N}^2 \rightarrow \mathbb{N}, \text{mult}(x, y) = x \cdot y \]
is primitive recursive.
We have the laws:

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

Therefore

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

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

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

Therefore

\[
\text{mult} = \text{primrec}(0, \text{add} \circ (\text{proj}_2^3, \text{proj}_0^3)).
\]

Predecessor Function

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

Subtraction

\[
\begin{align*}
\text{sub}(x, y) = x \div y \text{ is prim. rec.:} \\
\text{sub}(x, 0) &= x, \\
\text{sub}(x, y + 1) &= x \div (y + 1) \\
&= (x \div y) \div 1 \\
&= \text{pred} \left( \text{sub}(x, y) \right).
\end{align*}
\]
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
\[ x \div (x \div 1) = x - (x - 1) = x - x + 1 = 1 = \text{sig}(x). \]

\[ x < y \text{ is Prim. Rec.} \]

\[ A(x, y) :\iff x < y \text{ is primitive recursive, since } \chi_A(x, y) = \text{sig}(y - x): \]

\[ \text{If } x < y, \text{ then } y \div x = y - x > 0, \]

therefore
\[ \text{sig}(y \div x) = 1 = \chi_A(x, y). \]

\[ \text{If } \neg(x < y), \text{ i.e. } x \geq y, \text{ then } \]
\[ y \div x = 0, \]
\[ \text{sig}(y \div x) = 0 = \chi_A(x, y). \]

Add., Mult., Exp.

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

Consider the sequence of definitions of addition, multiplication, exponentiation:

Addition:
\[ x + 0 = x, \]
\[ x + (y + 1) = (x + y) + 1, \]
Therefore, if we write \((+ 1)\) for the function \( \mathbb{N} \rightarrow \mathbb{N}, \)
\[ ((+ 1))(x) = x + 1, \text{ then } \]
\[ x + y = ((+ 1)^y(x). \]
Remark on Notation

The notation \( (\cdot \cdot \cdot (\cdot 1) \cdot \cdot \cdot ) ) \) is to be understood as follows:

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

\[
f^n(x) := f(f(\cdot \cdot \cdot f(x) \cdot \cdot \cdot ))
\]

This is not to be confused with exponentiation

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

So

\[
((\cdot \cdot \cdot (\cdot 1) \cdot \cdot \cdot )) = ((\cdot \cdot \cdot (\cdot 1) \cdot \cdot \cdot ))
\]

Add., Mult., Exp.

Multiplication:

\[
x \cdot 0 = 0,
x \cdot (y + 1) = x \cdot y + x.
\]

Therefore, if we write \( ((\cdot \cdot \cdot ) \cdot x) \) for the function \( \mathbb{N} \rightarrow \mathbb{N} \),
\[
((\cdot \cdot \cdot ) \cdot x)(y) = y + x,
\]

then

\[
x \cdot y = ((\cdot \cdot \cdot ) \cdot x)(0).
\]

Exponentiation:

\[
x^0 = 1,
x^{y+1} = (x^y) \cdot x.
\]

Therefore, if we write \( ((\cdot ) \cdot x) \) for the function \( \mathbb{N} \rightarrow \mathbb{N} \),
\[
((\cdot ) \cdot x)(y) = x \cdot y,
\]

then

\[
x^y = ((\cdot ) \cdot x)(1).
\]

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

Superexponentiation

Extend this sequence further, by defining

Superexponentiation:

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

\[
\text{superexp}(x, y + 1) = x^{\text{superexp}(x,y)}.
\]

Therefore, if we write \( ((\uparrow ) \cdot n) \) for the function \( \mathbb{N} \rightarrow \mathbb{N} \),
\[
((\uparrow ) \cdot n)(k) = n^k,
\]

then

\[
\text{superexp}(x, y) = ((\uparrow ) \cdot x)(1).
\]
Supersuperexponentiation

- **Supersuperexponentiation:**
  \[
  \text{supersupperexp}(x, 0) = 1, \\
  \text{supersupperexp}(x, y + 1) = \text{superexp}(x, \text{supersupperexp}(x, y)),
  \]
  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 \( \lor, \land, \neg \)**
- If \( R, S \subseteq \mathbb{N}^n \) are prim. rec., so are
  \[
  R \lor S, \\
  R \land S, \\
  \neg R.
  \]

Closure under Prop. Connectives

- **Here**
  \[
  (R \lor S)(\vec{x}) \iff R(\vec{x}) \lor S(\vec{x}), \\
  (R \land S)(\vec{x}) \iff R(\vec{x}) \land S(\vec{x}), \\
  (\neg R)(\vec{x}) \iff \neg R(\vec{x}).
  \]
- 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:
  \[
  x \leq y \iff \neg(y < x), \\
  x = y \iff x \leq y \land y \leq x.
  \]

**Remark** \( \land, \lor, \neg \mathbb{N}^n \setminus \)

- We have
  \[
  R \lor S = R \cup S \text{ (the set theoretic union of } R \text{ and } S), \\
  R \land S = R \cap S, \\
  \neg R = \mathbb{N}^n \setminus R.
  \]
Closure under $\lor$, $\land$, $\neg$

**Proof of $R \cup S = R \lor S$:**

$$(R \cup S)(\vec{x}) \iff \vec{x} \in R \cup S \iff \vec{x} \in R \lor \vec{x} \in S \iff R(\vec{x}) \lor S(\vec{x})$$

Jump over Rest

**Proof of $R \cap S = R \land S$:**

$$(R \cap S)(\vec{x}) \iff \vec{x} \in R \cap S \iff \vec{x} \in R \land \vec{x} \in S \iff R(\vec{x}) \land S(\vec{x})$$

Proof of Closure under $\lor$

**Proof of Closure under $\lor$**

$$\chi_{R \lor S}(\vec{x}) = \text{sig}(\chi_R(\vec{x}) + \chi_S(\vec{x})), \quad (\text{therefore } R \lor S \text{ is primitive recursive}):$$

- If $R(\vec{x})$ holds, then
  $$\text{sig}(\chi_R(\vec{x}) + \chi_S(\vec{x})) = 1 = \chi_{R \lor S}(\vec{x})$$

- Similarly, if $S(\vec{x})$ holds, then
  $$\text{sig}(\chi_R(\vec{x}) + \chi_S(\vec{x})) = 1 = \chi_{R \lor S}(\vec{x})$$
Proof of Closure under $\lor$

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

\[
\text{sig}(\chi_R(\vec{x}) + \chi_S(\vec{x})) = 0 = \chi_{R \lor S}(\vec{x}) .
\]

Proof of Closure under $\land$

- If $\neg R(\vec{x})$ holds, then $\chi_R(\vec{x}) = 0$, therefore

\[
\chi_R(\vec{x}) \cdot \chi_S(\vec{x}) = 0 = \chi_R \land S(\vec{x}) .
\]

- Similarly, if $\neg S(\vec{x})$, we have

\[
\chi_R(\vec{x}) \cdot \chi_S(\vec{x}) = 0 = \chi_R \land S(\vec{x}) .
\]

Proof of Closure under $\neg$

- $\chi_{\neg R}(\vec{x}) = 1 \div \chi_R(\vec{x})$

(and therefore primitive recursive):

Jump over Rest of Proof

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

\[
1 \div \chi_R(\vec{x}) = 1 = \chi_{\neg R}(\vec{x}) .
\]

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

\[
1 \div \chi_R(\vec{x}) = 1 = \chi_{\neg R}(\vec{x}) .
\]
**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**

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

is primitive recursive.

---

**Definition by Cases**

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

is primitive recursive.

---

**Bounded Sums**

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

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

where

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

and for \( y > 0 \),

\[
\sum_{z<y} g(\vec{x}, z) := g(\vec{x}, 0) + g(\vec{x}, 1) + \cdots + g(\vec{x}, y-1).
\]
Bounded Sums

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

Proof that \( f \) is prim. rec.:

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

Jump over rest of proof

The last equations follows from

\[
\begin{align*}
f(\vec{x}, y + 1) &= \sum_{z<y+1} g(\vec{x}, z) \\
&= (\sum_{z<y} g(\vec{x}, z)) + g(\vec{x}, y) \\
&= f(\vec{x}, y) + g(\vec{x}, y).
\end{align*}
\]

Bounded Products

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

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 and Example Factorial Function

Example

We have above

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

etc.

Bounded Products

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

Proof that \( f \) is prim. rec.:

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

Here, the last equations follows by

\[
\begin{align*}
f(\vec{x}, y + 1) &= \prod_{z<y+1} g(\vec{x}, z) \\
&= (\prod_{z<y} g(\vec{x}, z)) \cdot g(\vec{x}, y) \\
&= f(\vec{x}, y) \cdot g(\vec{x}, y).
\end{align*}
\]

Jump over next Example
Example

Example for closure under bounded products:
$f : \mathbb{N} \to \mathbb{N}$,

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

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

is primitive recursive, since

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

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

(Note that in the special case \( x = 0 \) 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:

\[ 0! = 1 \]

\[ (x + 1)! = x! \cdot (x + 1) \]

Bounded Quantification

\[ \text{If } R \subseteq \mathbb{N}^{n+1} \text{ is prim. rec., so are} \]

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

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

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

\[ R_1(\vec{x}, y) :\iff \forall z < y. R(\vec{x}, z) \]  
Show \( \chi_{R_1}(\vec{x}, y) = \prod_{z<y} \chi_R(\vec{x}, z) \).

- If \( \neg R(\vec{x}, z) \) for one \( z < y \),  
  then \( \chi_R(\vec{x}, z) = 0 \), therefore  
  \[ \prod_{z<y} \chi_R(\vec{x}, z) = 0 = \chi_{R_1}(\vec{x}, y) \].

Bounded Quantification

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

- If \( R(\vec{x}, z) \), for some \( z < y \),  
  then \( \chi_R(\vec{x}, z) = 1 \), therefore  
  \( \sum_{z<y} \chi_R(\vec{x}, y) \geq \chi_R(\vec{x}, z) = 1 \),  
  therefore  
  \[ \text{sig}(\sum_{z<y} \chi_R(\vec{x}, y)) = 1 = \chi_{R_2}(\vec{x}, y) \].

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

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

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

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

Jump over Alternative Proof

---

### Alternative Proof of Closure under Bounded Search

**Define**

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

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

**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
    
    \[
    (\sum_{z<y} \chi Q(\vec{x}, z) \cdot z) + \chi Q'(\vec{x}, y) \cdot y = z = \mu z' < y. R(\vec{x}, z') .
    \]

Jump over details.
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

Omit 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 “$z = z’$”.

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 coding and decoding functions for pairs, tuples and sequences of natural numbers are primitive recursive.

More precisely, the following functions are primitive recursive:

- (a) $\pi : \mathbb{N}^2 \to \mathbb{N}.$
  (Remember, $\pi(x, y)$ encodes two natural numbers as one.)

- (b) $\pi_0, \pi_1 : \mathbb{N} \to \mathbb{N}.$
  (Remember $\pi_0(\pi(x, y)) = x, \pi_1(\pi(x, y)) = y).$

- (c) $\pi^k : \mathbb{N}^k \to \mathbb{N}$ ($k \geq 1$).
  (Remember $\pi^k(x_0, \ldots, x_{k-1})$ encodes the sequence $(x_0, \ldots, x_{k-1})$.)
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 \( \pi^k_i(\pi^k(x_0, \ldots , x_{k-1})) = x_i \) for \( i < k \).)

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

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

\[
f_k(x_0, \ldots , x_{k-1}) = \langle x_0, \ldots , x_{k-1} \rangle.
\]

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

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

(Remember that \( \text{lh}(\langle x_0, \ldots , x_{k-1} \rangle) = k \).)

Proof of Lemma 5.1 (a), (b)

(a)

\[
\pi(x, y) = (\sum_{i \leq x+y} i) + y = (\sum_{i < x+y+1} i) + y
\]

is primitive recursive.

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

\[
\pi_0(x) := \mu y < x + 1.3z < x + 1.x = \pi(y, z),
\]

\[
\pi_1(x) := \mu z < x + 1.3y < x + 1.x = \pi(y, z).
\]

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

Proof of Lemma 5.1 (c)

(c) Proof by induction on \( k \):

- \( k = 1 \): \( \pi^1(x) = x \), so \( \pi^1 \) is primitive recursive.
- \( k \to k + 1 \): Assume that \( \pi^k \) 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)

We have
\[ \pi_0^1(x) = x, \]
\[ \pi_i^{k+1}(x) = \pi_i^k(\pi_0(x)), \] if \( i < k \),
\[ \pi_i^{k+1}(x) = \pi_1(x), \] if \( i = k \).

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

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

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 \( g : \mathbb{N}^2 \to \mathbb{N} \),
\[ g(x, 0) := x, \]
\[ g(x, k + 1) := \pi_0(g(x, k)), \]

which 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) \[ (x)_i = \begin{cases} \pi_i^{\text{lh}(x)}(\pi_1(x - 1)), & \text{if } i \neq 0, \\ f(\pi_1(x - 1), \text{lh}(x), i) & \text{if } i = 0 \end{cases} \]
is primitive recursive.
Lemma and Definition 5.2

(Technical Lemma needed in the proof of closure under course-of-value primitive recursion below.) Prim. rec. functions as follows do exist:

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

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

**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.

$$\text{last}(\text{snoc}(x, y)) = y ,$$

$$\text{beginning}(\text{snoc}(x, y)) = x .$$

Jump over proof.

### Proof of Lemma 5.2 (a)

Define

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

so $\text{snoc}$ is primitive recursive.

### Proof of Lemma 5.2 (b)

We have

$$\text{snoc}(\langle \rangle, y) = \langle y \rangle ,$$

$$\text{snoc}(0, y) = \langle y \rangle ,$$

$$\text{snoc}(\langle x_0, \ldots, x_k \rangle, y) = 1 + \pi(k + 1, \pi(k, \pi(k+1)(x_0, \ldots, x_k)), y)\)$$

(by $\text{lh}(\langle x_0, \ldots, x_k \rangle) = k + 1$)

$$= 1 + \pi(k + 1, \pi(k, \pi(k+1)(x_0, \ldots, x_k), y))$$

$$= 1 + \pi(k + 1, \pi(k, \pi(k+1)(x_0, \ldots, x_k), y))$$

$$= 1 + \pi(k + 1, \pi(k+2)(x_0, \ldots, x_k, y))$$

$$= \langle x_0, \ldots, x_k, y \rangle .$$

**Proof for beginning:**

Define

$$\text{beginning}(x) := \begin{cases} \langle \rangle, & \text{if } \text{lh}(x) \leq 1, \\ \langle (x)_0 \rangle, & \text{if } \text{lh}(x) = 2, \\ 1 + \pi((\text{lh}(x) - 1) - 1, \pi_0(\pi_1(y) - 1)), & \text{otherwise.} \end{cases}$$
Proof of Lemma 5.2 (b)

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** \( \text{lh}(y) > 1 \): 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
\[
\text{snoc}(y, z) = 1 + \pi(n + 2, \pi(\pi_1(y - 1), z))
\]

Therefore
\[
\text{beginning}(\text{snoc}(y, z))
\begin{align*}
&= 1 + \pi((\text{lh}(x) - 1) - 1, \pi_0(\pi_1(\text{snoc}(y, z) - 1))) \\
&= 1 + \pi(n, \pi_0(\pi_1((1 + \pi(n + 2, \pi(\pi_1(y - 1), z))) - 1))) \\
&= 1 + \pi(n, \pi_0(\pi_1(y - 1), z)) \\
&= 1 + \pi(n, \pi_1(y - 1)) \\
&= 1 + \pi(n, \pi_1(1 + \pi(n + 1, \pi^{n+2}(y_0, \ldots, y_{n+1}))) - 1)) \\
&= 1 + \pi(n, \pi_1(1 + \pi(n + 1, \pi^{n+2}(y_0, \ldots, y_{n+1})))) \\
&= 1 + \pi(n, \pi^{n+2}(y_0, \ldots, y_{n+1})) \\
&= y.
\end{align*}
\]
Proof of Lemma 5.2 (b)

**Proof for last:**
Define

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

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

\[
\text{last}(\text{snoc}(y, z)) = \text{last}(\langle y_0, \ldots, y_{n-1}, z \rangle) \\
= (\langle y_0, \ldots, y_{n-1}, z \rangle)_{h(\langle y_0, \ldots, y_{n-1}, z \rangle) - 1} \\
= (\langle y_0, \ldots, y_{n-1}, z \rangle)_{n} \\
= z.
\]

Definition Course-Of-Value

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

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

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 \( g : \mathbb{N}^{n+2} \rightarrow \mathbb{N} \)

is primitive recursive. Then

\[
f : \mathbb{N}^{n+1} \rightarrow \mathbb{N} \\
f(\bar{x}, k) = g(\bar{x}, k, \overline{f}(\bar{x}, k))
\]

is prim. rec.

Informal meaning of course-of-value primitive recursion:
If we can express \( f(\bar{x}, y) \) by an expression using

- constants,
- \( \bar{x}, y \),
- previously defined prim. rec. functions,
- \( f(\bar{x}, z) \) for \( z < y \),

then \( f \) is prim. rec.
Example

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

\[
\begin{align*}
\text{fib}(0) & := 1, \\
\text{fib}(1) & := 1, \\
\text{fib}(x) & := \text{fib}(x-2) + \text{fib}(x-1), \text{ if } x > 1,
\end{align*}
\]

Definable by course-of-value primitive recursion:

\[\text{fib}(x) = \begin{cases} 
1 & \text{if } x \leq 1, \\
(\text{fib}(x))_{x-2} + (\text{fib}(x))_{x-1} & \text{otherwise.}
\end{cases}\]

using \((\text{fib}(x))_{x-2} = \text{fib}(x-2), (\text{fib}(x))_{x-1} = \text{fib}(x-1)\).

---

Proof

\( f(\vec{x}, y) = g(\vec{x}, y, \overline{f}(\vec{x}, y)) \)

\[
\begin{align*}
\overline{f}(\vec{x}, 0) &= \langle \rangle, \\
\overline{f}(\vec{x}, y + 1) &= \langle f(\vec{x}, 0), f(\vec{x}, 1), \ldots, f(\vec{x}, y - 1), f(\vec{x}, y) \rangle \\
&= \text{snoc}(\langle f(\vec{x}, 0), f(\vec{x}, 1), \ldots, f(\vec{x}, y - 1) \rangle, f(\vec{x}, y)) \\
&= \text{snoc}(\overline{f}(\vec{x}, y), f(\vec{x}, y)) \\
&= \text{snoc}(\overline{f}(\vec{x}, y), g(\vec{x}, y, \overline{f}(\vec{x}, y))) \).
\]

Therefore \( \overline{f} \) is primitive recursive.

---

Proof

\( f(\vec{x}, y) = g(\vec{x}, y, \overline{f}(\vec{x}, y)) \)

Now we have that

\[
\begin{align*}
f(\vec{x}, y) &= \langle f(\vec{x}, 0), \ldots, f(\vec{x}, y) \rangle_y \\
&= (\overline{f}(\vec{x}, y + 1))_y \\
&= \text{last}(\overline{f}(\vec{x}, y + 1))
\end{align*}
\]

is primitive recursive.
Lemma and Definition 5.3

(Technical Lemma used later to simulate Turing Machines using primitive recursive/partial recursive functions).

There exist prim. rec. functions as follows:

(a) append : \( \mathbb{N}^2 \to \mathbb{N} \) s.t.

\[
\text{append}(\langle x_0, \ldots, x_{k-1}, y_0, \ldots, y_{l-1} \rangle) = \langle x_0, \ldots, x_{k-1}, y_0, \ldots, y_{l-1} \rangle.
\]

We write \( x \ast y \) for \( \text{append}(x, y) \).

(b) subst : \( \mathbb{N}^3 \to \mathbb{N} \), s.t. if \( i < n \) then

\[
\text{subst}(\langle x_0, \ldots, x_{n-1}, i, y \rangle) = \langle x_0, \ldots, x_{i-1}, y, x_{i+1}, x_{i+2}, \ldots, x_{n-1} \rangle,
\]

and if \( i \geq n \), then

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

We write \( x[i/y] \) for \( \text{subst}(x, i, y) \).

Lemma and Definition 5.3

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

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

and if \( i \geq n \),

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

Proof of Lemma 5.3 (a)

We have

\[
\text{append}(\langle x_0, \ldots, x_n \rangle, 0) = \langle x_0, \ldots, x_n \rangle,
\]

and for \( m > 0 \)
\[
\text{append}(\langle x_0, \ldots, x_n \rangle, \langle y_0, \ldots, y_m \rangle) = \langle x_0, \ldots, x_n, y_0, \ldots, y_m \rangle.
\]

Lemma and Definition 5.3

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

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

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

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

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

s.t. \( x = (b_0, \ldots, b_k)_2 \)

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

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

The proof will be omitted in the lecture.

Jump over proof.
Proof of Lemma 5.3 (a)

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.

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 (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 (d), (e)

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

(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 (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.