7. The Recursion Theorem

- Main result in this section: **Kleene’s Recursion Theorem**.
  - Recursive functions are closed under a very general form of recursion.
  - For proof we will use the **S-m-n-theorem**.
  - Used in many proofs in computability theory.

---

The S-m-n Theorem

- Assume $f : \mathbb{N}^{m+n} \rightarrow \mathbb{N}$ partial recursive.
- Fix the first $m$ arguments (say $\bar{l} := l_0, \ldots, l_{m-1}$).
- Then we obtain a partial recursive function
  \[ g : \mathbb{N}^n \rightarrow \mathbb{N}, \quad g(\bar{x}) \simeq f(\bar{l}, \bar{x}) \, . \]
- The S-m-n theorem expresses that we can compute a Kleene index of $g$
  - i.e. an $e'$ s.t. $g = \{e'\}^n$
  - from a Kleene index of $f$ and $\bar{l}$ **primitive recursively**.

---

Theorem 7.1 (S-m-n Theorem)

- Assume $m, n \in \mathbb{N}$.
- There exists a primitive recursive function
  \[ S^m_n : \mathbb{N}^{m+1} \rightarrow \mathbb{N} \]
  s.t. for all $\bar{l} \in \mathbb{N}^m, \bar{x} \in \mathbb{N}^n$
  \[ \{S^m_n(e, \bar{l})\}^n(\bar{x}) \simeq \{e\}^{m+n}(\bar{l}, \bar{x}) \, . \]
Proof of S-m-n Theorem

Let $T$ be a TM encoded as $e$.

A Turing machine $T'$ corresponding to $S^m_n(e, l)$ should be s.t.

$$T'^m(x) \simeq T^{n+m}(l, x).$$

---

Proof of S-m-n Theorem

$T$ is TM for $e$.

Want to define $T'$ s.t. $T'^m(x) \simeq T^{n+m}(l, x)$

Initial configuration:

$$\ldots \underline{\ldots} \underline{\ldots} \underline{\bin(x_0)} \underline{\ldots} \underline{\ldots} \underline{\bin(x_{n-1})} \underline{\ldots} \underline{\ldots} \underline{\ldots}$$

1. The initial configuration is:
   - $x$ written on the tape,
   - head pointing to the left most bit:

$$\ldots \underline{\ldots} \underline{\bin(x_0)} \underline{\ldots} \underline{\ldots} \underline{\bin(x_{n-1})} \underline{\ldots} \underline{\ldots} \underline{\ldots}$$

2. $T'$ writes first binary representation of $l = l_0, \ldots, l_{n-1}$ in front of this.

   terminates this step with the head pointing to the most significant bit of $\bin(l_0)$.

So configuration after this step is:

$$\underline{\bin(l_0)} \underline{\ldots} \underline{\ldots} \underline{\bin(l_{m-1})} \underline{\bin(x_0)} \underline{\ldots} \underline{\ldots} \underline{\bin(x_{n-1})}$$

Then $T'$ runs $T$, starting in this configuration.

It terminates, if $T$ terminates.

The result is

$$\simeq T^{m+n}(l, x),$$

and we get therefore

$$T'^m(x) \simeq T^{n+m}(l, x)$$

as desired.
Proof of the S-m-n Theorem

T is TM for e.
T' is a TM s.t. \( T'^m(x) \simeq T^{n+m}(\bar{l}, \bar{x}) \)

- From a code for T one can now obtain a code for T' in a primitive recursive way.
- \( S^m_n \) is the corresponding function.
- The details will not be given in the lecture Jump over details

Proof of the S-m-n Theorem

A code for T' can be obtained from a code for T and from \( \bar{l} \) as follows:

- One takes a Turing machine T'', which writes the binary representations of \( \bar{l} = l_0, \ldots, l_{m-1} \)
  in front of its initial position (separated by a blank and with a blank at the end), and terminates at the left most bit.
- It's a straightforward exercise to write a code for the instructions of such a Turing machine, depending on \( \bar{l} \), and show that the function defining it is primitive recursive.

Proof of the S-m-n Theorem

\begin{itemize}
  \item Assume, the terminating state of T'' has Gödel number (i.e. code) s, and that all other states have Gödel numbers < s.
  \item Then one appends to the instructions of T'' the instructions of T, but with the states shifted, so that the new initial state of T is the final state s of T'' (i.e. we add s to all the Gödel numbers of states occurring in T).
  \item This can be done as well primitive recursively.
\end{itemize}

Proof of the S-m-n Theorem

So a code for T'' can be defined primitive recursively depending on a code e for T and \( \bar{l} \), and \( S^m_n \) is the primitive recursive function computing this. With this function it follows now that, if e is a code for a TM, then

\[ \{S^m_n(e, \bar{l})\}^n(x) \simeq \{e\}^{n+m}(\bar{l}, \bar{x}) \]

This equation holds, even if e is not a code for a TM: In this case \( \{e\}^{m+n} \) interprets e as if it were the code for a valid TM T.
Proof of the S-m-n Theorem

(A code for such a valid TM is obtained by
1. deleting any instructions \(\text{encode}(q, a, q', a', D)\) in \(e\)
   s.t. there exists an instruction \(\text{encode}(q, a, q'', a'', D')\)
   occurring before it in the sequence \(e\),
2. and by replacing all directions \(> 1\) by \([R] = 1\).

\[ e_0 := S_m^n(e, \overline{l}) \]

will have the same deficiencies as \(e\), but when applying the Kleene-brackets, it will be interpreted as a TM \(T'\) obtained from \(e'\) in the same way as we obtained \(T\) from \(e\), and therefore

\[ \{e'\}^n(x) \simeq T'^m(x) \simeq T^{n+m}(\overline{l}, x) \simeq \{e\}^{n+m}(\overline{l}, x) . \]

So we obtain the desired result in this case as well.

Notation

We will in the following often omit the superscript \(n\) in \(\{e\}^n(m_0, \ldots, m_{n-1})\).

I.e. we will write

\[ \{e\}(m_0, \ldots, m_{n-1}) \]

instead of

\[ \{e\}^n(m_0, \ldots, m_{n-1}) \]

Further \(\{e\}\) not applied to arguments and without superscript means usually \(\{e\}^1\).

Notation

We introduce as well some notations for dealing with \(\overline{x}, \overline{y}\) etc.

\[ \forall \overline{x} \in \mathbb{N}, \varphi(\overline{x}) \]

stands for

\[ \forall x_1, \ldots, x_n \in \mathbb{N}, \varphi(x_1, \ldots, x_n) \]

where the number of variables is implicit (and usually unimportant).

\[ \exists \overline{x} \in \mathbb{N}, \varphi(\overline{x}) \]

is to be understood similarly.
Notation

\[ \{ \bar{x} \in \mathbb{N}^n \mid \varphi(\bar{x}) \} \]

is to be understood as
\[ \{(x_1, \ldots, x_n) \in \mathbb{N}^n \mid \varphi(x_1, \ldots, x_n) \} \]

\[ \{(x, y, z) \in \mathbb{N}^n \mid \varphi(x, y, z) \} \]
is to be understood as
\[ \{(x_1, \ldots, x_n, y, z) \in \mathbb{N}^n \mid \varphi(x_1, \ldots, x_n, y, z) \} \]

Similar notations are to be understood analogously.

### Examples

Kleene’s Recursion Theorem: Assume \( f : \mathbb{N}^{n+1} \rightarrow \mathbb{N} \) partial recursive.

Then there exists an \( e \in \mathbb{N} \) s.t.
\[ \{e\}^n(\bar{x}) \simeq f(e, \bar{x}) . \]

(Here \( \bar{x} = x_0, \ldots, x_{n-1} \).)
Examples

2. The function computing the **Fibonacci-numbers** `fib` is recursive.
   - (This is a weaker result than what we obtained above –
     above we showed that it is even prim.-rec.)

Fibonacci Numbers

Remember the defining equations for `fib`:

\[
\begin{align*}
\text{fib}(0) &= 1, \\
\text{fib}(1) &= 1, \\
\text{fib}(n+2) &= \text{fib}(n) + \text{fib}(n+1).
\end{align*}
\]

From these equations we obtain

\[
\text{fib}(n) = \begin{cases} 
1, & \text{if } n = 0 \text{ or } n = 1, \\
\text{fib}(n-2) + \text{fib}(n-1), & \text{otherwise.}
\end{cases}
\]

We show that there exists a recursive function `g : \mathbb{N} \to \mathbb{N}`, s.t.

\[
g(n) \simeq \begin{cases} 
1, & \text{if } n = 0 \text{ or } n = 1, \\
g(n-2) + g(n-1), & \text{otherwise.}
\end{cases}
\]

Fibonacci Numbers

Show: Exists `g` rec. s.t. `g(n) \simeq \begin{cases} 
1, & \text{if } n = 0 \text{ or } n = 1, \\
g(n-2) + g(n-1), & \text{otherwise.}
\end{cases}`

Shown as follows: Define a recursive `f : \mathbb{N}^2 \to \mathbb{N}` s.t.

\[
f(e,n) \simeq \begin{cases} 
1, & \text{if } n = 0 \text{ or } n = 1, \\
\{e\}(n-2) + \{e\}(n-1), & \text{otherwise.}
\end{cases}
\]

Now let `e` be s.t.

\[
\{e\}(n) \simeq f(e, n) .
\]

Then `e` fulfils the equations

\[
\{e\}(n) \simeq \begin{cases} 
1, & \text{if } n = 0 \text{ or } n = 1, \\
\{e\}(n-2) + \{e\}(n-1), & \text{otherwise.}
\end{cases}
\]

These are the defining equations for `fib`.

Let `g = \{e\}`.

Then we get

\[
g(n) \simeq \begin{cases} 
1, & \text{if } n = 0 \text{ or } n = 1, \\
g(n-2) + g(n-1), & \text{otherwise.}
\end{cases}
\]

One can show by induction on `n` that `g(n) = \text{fib}(n)` for all `n \in \mathbb{N}`.

Therefore `fib` is recursive.
Similarly, one can introduce arbitrary partial recursive functions \( g \), where
\( g(\vec{n}) \) refers to arbitrary other values \( g(\vec{m}) \).
This corresponds to the recursive definition of functions in programming.
E.g. in Java one defines

```java
public static int fib(int n){
    if (n == 0 || n == 1){
        return 1;
    }else{
        return fib(n-1) + fib(n-2);
    }
}
```

**Example 3**

As in general programming, recursively defined functions need not be total:
- There exists a partial recursive function \( g : \mathbb{N} \not
\rightarrow \mathbb{N} \) s.t.
  \( g(x) \simeq g(x) + 1 \).
- We get \( g(x) \uparrow \).
- The definition of \( g \) corresponds to the following Java definition:
  ```java
  public static int g(int n){
      return g(n) + 1;
  }
  ```
- When executing \( g(x) \), Java loops.

**Example 4**

There exists a partial recursive function \( g : \mathbb{N} \rightarrow \mathbb{N} \) s.t.

\[
g(x) \simeq g(x + 1) + 1 .
\]

Note that that’s a “black hole recursion”, which is not solvable by a total function.
- It is solved by \( g(x) \uparrow \).
- Note that a recursion equation for a function \( f \) cannot always be solved by setting \( f(x) \uparrow \).
- E.g. the recursion equation for \( \text{fib} \) can’t be solved by setting \( \text{fib}(n) \uparrow \).

**Ackermann Function**

The Ackermann function is recursive:
Remember the defining equations:

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

From this we obtain

\[
\text{Ack}(x, y) = \begin{cases} 
  y + 1, & \text{if } x = 0, \\
  \text{Ack}(x-1, 1), & \text{if } x > 0 \text{ and } y = 0, \\
  \text{Ack}(x-1, \text{Ack}(x, y-1)), & \text{otherwise}.
\end{cases}
\]
Ackermann Function

\[ \text{Ack}(x, y) = \begin{cases} 
  y + 1, & \text{if } x = 0, \\
  \text{Ack}(x - 1, 1), & \text{if } x > 0 \text{ and } y = 0, \\
  \text{Ack}(x - 1, \text{Ack}(x, y - 1)), & \text{otherwise}.
\end{cases} \]

Define \( g \) partial recursive s.t.

\[ g(x, y) = \begin{cases} 
  y + 1, & \text{if } x = 0, \\
  g(x - 1, 1), & \text{if } x > 0 \text{ and } y = 0, \\
  g(x - 1, g(x, y - 1)), & \text{if } x > 0 \text{ and } y > 0.
\end{cases} \]

\( g \) fulfills the defining equations of \( \text{Ack} \).

Proof that \( g(x, y) \simeq \text{Ack}(x, y) \) follows by main induction on \( x \), side-induction on \( y \). The details will not be given in the lecture Jump over details.

Proof of Correctness of \( \text{Ack} \)

We show by induction on \( x \) that \( g(x, y) \) is defined and equal to \( \text{Ack}(x, y) \) for all \( x, y \in \mathbb{N} \):

- **Base case** \( x = 0 \).
  \[ g(0, y) = y + 1 = \text{Ack}(0, y) \]

- **Induction Step** \( x \rightarrow x + 1 \). Assume
  \[ g(x, y) = \text{Ack}(x, y) \]

We show

\[ g(x + 1, y) = \text{Ack}(x + 1, y) \]

by side-induction on \( y \):

Proof of the Rec. Theorem

Assume

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

We have to find an \( e \) s.t.

\[ \forall \bar{x} \in \mathbb{N}. \{e\}^n(\bar{x}) \simeq f(e, \bar{x}) \]

**Idea:**

Define \( e = S^1_n(e_1, e_2) \) for some (yet unknown) \( e_1, e_2 \).

We get:

\[ \{e\}^n(\bar{x}) \simeq \{S^1_n(e_1, e_2)\}^n(\bar{x}) \simeq \{e_1\}^{n+1}(e_2, \bar{x}) \]

So we need to find \( e_1, e_2 \) s.t.

\[ \forall \bar{x}. \{e_1\}^{n+1}(e_2, \bar{x}) \simeq f(S^1_n(e_1, e_2), \bar{x}) \]
Proof of Rec. Theorem

Need to solve:
\( \forall \vec{x}, \{e_1\}^{n+1}(e_2, \vec{x}) \simeq f(S_n^1(e_1, e_2), \vec{x}). \)

Let \( e_1 \) s.t.
\[ \{e_1\}^{n+1}(y, \vec{x}) \simeq f(S_n^1(y, y), \vec{x}). \]

So we need to solve
\[
\begin{align*}
  f(S_n^1(e_1, e_2), \vec{x}) & \simeq f(S_n^1(e_2, e_2), \vec{x}) \\
  \simeq & \{e_1\}^{n+1}(e_2, \vec{x})
\end{align*}
\]

Can be fulfilled by setting \( e_2 := e_1 \).
So, an index solving the problem is
\[ e = S_n^1(e_1, e_2) = S_n^1(e_1, e_1), \]

and we are done.

---

Short Version of Proof

Let \( e_1 \) be s.t.
\[ \{e_1\}^{n+1}(y, \vec{x}) \simeq f(S_n^1(y, y), \vec{x}). \]

Let \( e := S_n^1(e_1, e_1) \).
Then we have
\[
\begin{align*}
  \{e\}^n(\vec{x}) & \simeq S_n^1(e_1, e_1) \\
  \simeq & \{S_n^1(e_1, e_1)\}^n(\vec{x}) \\
  \simeq & \{e_1\}^{n+1}(e_1, \vec{x}) \\
  \simeq & f(S_n^1(e_1, e_1), \vec{x}) \\
  \simeq & f(e, \vec{x})
\end{align*}
\]