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 the proof we will use the **S-m-n-theorem.**

Used in many proofs in computability theory.

However, both the S-m-n theorem and the proof of the Recursion theorem will be omitted this year.

Jump to Kleene’s Recursion Theorem.

---

The S-m-n Theorem

- Assume \( f : \mathbb{N}^{m+n} \rightarrow \mathbb{N} \) partial recursive.
- Fix the first \( m \) arguments (say \( \vec{l} := l_0, \ldots, l_{m-1} \)).
- Then we obtain a partial recursive function

\[
g : \mathbb{N}^n \rightarrow \mathbb{N}, \quad g(\vec{x}) \simeq f(\vec{l}, \vec{x})\]

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 \( \vec{l} \) primitive recursively.

---

Notation

\[
\{S_n^m(e, \vec{l})\}^n(\vec{x}) \simeq \{e\}^{m+n}(\vec{l}, \vec{x}).
\]

Assume \( t \) is an expression depending on \( n \) variables \( \vec{x} \),

s.t. we can compute \( t \) from \( \vec{x} \) partial recursively.

Then \( \lambda \vec{x}.t \) is any natural number \( e \) s.t. \( \{e\}^n(\vec{x}) \simeq t \).

Then we will have

\[
S_n^m(e, \vec{l}) = \lambda \vec{x}.\{e\}^{m+n}(\vec{l}, \vec{x}).
\]
**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 \( \vec{l} \in \mathbb{N}^m, \vec{x} \in \mathbb{N}^n \)
  \( \{S^m_n(e, \vec{l})\}^n(\vec{x}) \simeq \{e\}^{m+n}(\vec{l}, \vec{x}) \).

**Proof of S-m-n Theorem**

Let \( T \) be a TM encoded as \( e \).
- A Turing machine \( T' \) corresponding to \( S^m_n(e, \vec{l}) \) should be
  s.t.
  \( T'^{(n)}(\vec{x}) \simeq T^{(n+m)}(\vec{l}, \vec{x}) \).

1. **Initial configuration:**
   - \( \vec{x} \) written on the tape,
   - head pointing to the leftmost bit:
   \[
   \cdots \| \| \| \text{bin}(x_0) \| \| \cdots \| \| \text{bin}(x_{n-1}) \| \| \cdots
   \]

2. **T' writes first binary representation of \( \vec{l} = l_0, \ldots, l_{n-1} \)**
   - terminates this step with the head pointing to the most significant bit of bin(\( l_0 \)).
   - So configuration after this step is:
   \[
   \| \| \| \text{bin}(l_0) \| \| \cdots \| \| \text{bin}(l_{m-1}) \| \| \text{bin}(x_0) \| \| \cdots \| \| \text{bin}(x_{n-1})
   \]

---

**Proof of S-m-n Theorem**

- \( T \) is TM for \( e \).
- Want to define \( T' \) s.t. \( T'^{(n)}(\vec{x}) \simeq T^{(n+m)}(\vec{l}, \vec{x}) \)
- \( T' \) can be defined as follows:

1. **The initial configuration is:**
   - \( \vec{x} \) written on the tape,
   - head pointing to the left most bit:
     \[
     \cdots \| \| \| \text{bin}(x_0) \| \| \cdots \| \| \text{bin}(x_{n-1}) \| \| \cdots
     \]
     \[
     \]

2. **\( T' \) writes first binary representation of \( \vec{l} = l_0, \ldots, l_{n-1} \)**
   - 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:
     \[
     \| \| \| \text{bin}(l_0) \| \| \cdots \| \| \text{bin}(l_{m-1}) \| \| \text{bin}(x_0) \| \| \cdots \| \| \text{bin}(x_{n-1})
     \]

---
Proof of the S-m-n Theorem

T is TM for e.
Want to define T' s.t. T'(n)(x) \simeq T(n+m)(\bar{l}, x).
Configuration after first step:
\[
\begin{array}{cccccccc}
\text{bin}(l_0) & \ddots & \ldots & \text{bin}(l_{m-1}) & \text{bin}(x_0) & \ddots & \ldots & \text{bin}(x_{n-1}) \\
\uparrow
\end{array}
\]

Then T' runs T, starting in this configuration.
It terminates, if T terminates.
The result is
\[
\simeq T(m+n)(\bar{l}, x),
\]
and we get therefore
\[
T'(n)(x) \simeq T(m+n)(\bar{l}, x)
\]
as desired.

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 S-m-n Theorem

T is TM for e.

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

Configuration after first step:
\[
\begin{array}{cccccccc}
\text{bin}(l_0) & \ddots & \ldots & \text{bin}(l_{m-1}) & \text{bin}(x_0) & \ddots & \ldots & \text{bin}(x_{n-1}) \\
\uparrow
\end{array}
\]

Then T' runs T, starting in this configuration.
It terminates, if T terminates.
The result is
\[
\simeq T(m+n)(\bar{l}, x),
\]
and we get therefore
\[
T'(n)(x) \simeq T(m+n)(\bar{l}, x)
\]
as desired.

Assume, the terminating state of T'' has Gödel number (i.e. code) s, and that all other states have Gödel numbers < s.

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).
This can be done as well primitive recursively.
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 \( \vec{l} \), 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

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

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 \( T \).

\[ e' := S^m_n(e, \vec{l}) \] 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

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

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

Proof of the S-m-n Theorem

(A code for such a valid TM is obtained by 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 \( e \), and by replacing all directions \( > 1 \) by \( [R] = 1 \).)

Kleene’s Recursion Theorem

Assume \( f : \mathbb{N}^{n+1} \simeq \mathbb{N} \) partial recursive.

Then there exists an \( e \in \mathbb{N} \) s.t.

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

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

Kleene’s Rec. Theorem: \( \exists e. \forall \vec{x}. \{e\}^n(\vec{x}) \simeq f(e, \vec{x}). \)

There exists an \( e \) s.t.

\[ \{e\}(x) \simeq e + 1 . \]

For showing this take in the Recursion Theorem

\[ f(e, n) := e + 1. \]

Then

\[ \{e\}(x) \simeq f(e, x) \simeq e + 1 . \]

Remark

Kleene’s Rec. Theorem: \( \exists e. \forall \vec{x}. \{e\}^n(\vec{x}) \simeq f(e, \vec{x}). \)

Applications as Example 1 are usually not very useful.

Usually, when using the Rec. Theorem, one

- doesn’t use the index \( e \) directly,
- but only the application of \( \{e\} \) to arguments.

Example 2

- The function computing the **Fibonacci-numbers** \( \text{fib} \) 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 \( \text{fib} \):

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

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.

\[
\text{fib}(n) = \begin{cases} 
1, & \text{if } n = 0 \text{ or } n = 1, \\
g(n - 2) + g(n - 1), & \text{otherwise.}
\end{cases}
\]
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 \( f : \mathbb{N}^2 \rightarrow \mathbb{N} \) 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 \( e \) fulfills the equations

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

Let \( g = \{e\} \).

Then we get

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

These are the defining equations for \( \text{fib} \).

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

Therefore \( \text{fib} \) is recursive.

General Applic. of Rec. Theorem

Similarly, one can introduce arbitrary partial recursive functions \( g \), where

\( g(\vec{m}) \) refers to arbitrary other values \( g(\vec{m}) \).

So, instead of arguing as before that \( \text{fib} \) is partial recursive, it suffices to say the following

By the recursion theorem, there exists a partial recursive function \( \text{fib} : \mathbb{N} \rightarrow \mathbb{N} \) s.t.

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

We can prove by induction on \( n \) that \( \forall n : \mathbb{N}. \text{fib}(n) \downarrow \) holds.

Therefore \( \text{fib} \) is total and therefore recursive.

This use of the the recursion theorem 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);
    }
}
```

General Applic. of Rec. Theorem
Example 3

As in general programming, recursively defined functions need not be total:

- There exists a partial recursive function $g : \mathbb{N} \to \mathbb{N}$ 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 $g : \mathbb{N} \to \mathbb{N}$ s.t.
  
  \[ g(x) \simeq g(x) + 1 \]

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

- Define $g$ partial recursive s.t.

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

  \[ g \text{ fulfils the defining equations of } \text{Ack}. \]

  - 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 remaining slides.**
Proof of Correctness of **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 Correctness of Ack**

Show $g(x + 1, y) = \text{Ack}(x + 1, y)$

- **Base case** $y = 0$:
  
  \[ g(x + 1, 0) \simeq g(x, 1) \, \text{Main-IH} \]
  
  \[ \simeq \text{Ack}(x, 1) = \text{Ack}(x + 1, 0) \, . \]
- **Induction Step** $y \rightarrow y + 1$:
  
  \[ g(x + 1, y + 1) \simeq g(x, g(x + 1, y)) \, \text{Main-IH} \]
  
  \[ \simeq g(x, \text{Ack}(x + 1, y)) \, \text{Side-IH} \]
  
  \[ \simeq \text{Ack}(x, \text{Ack}(x + 1, y)) \]
  
  \[ = \text{Ack}(x + 1, y + 1) \, . \]

Jump over remaining slides

(Proof of the Recursion Theorem)

---

Idea of Proof of the Recursion Theorem

Assume

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

We have to find an $e$ s.t.

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

- We set $e = \lambda \bar{x}. \{e_1\}^{n+1}(e_1, \bar{x})$ for some $e_1$ to be determined.
- Then the left and right hand side of the equation of the recursion theorem reads

\[
\{e\}^n(\bar{x}) \simeq \{\lambda \bar{x}. \{e_1\}^{n+1}(e_1, \bar{x})\}^n(\bar{x})
\]

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

\[
f(e, \bar{x}) \simeq f(\lambda \bar{x}. \{e_1\}^{n+1}(e_1, \bar{x}), \bar{x})
\]

- **So $e_1$ needs to fulfill the following equation:**

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

\[
\simeq f(e, \bar{x})
\]

\[
\simeq f(\lambda \bar{x}. \{e_1\}^{n+1}(e_1, \bar{x}), \bar{x})
\]

- This can be fulfilled if we define $e_1$ s.t.

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

\[ \{e_1\}^{n+1}(e_2, \bar{x}) \simeq f(\lambda \bar{x}. \{e_2\}^{n+1}(e_2, \bar{x})). \]

- By the S-m-n Theorem we can obtain this if we have \( e_1 \) s.t.
  \[ \{e_1\}^{n+1}(e_2, \bar{x}) \simeq f(S^1_n(e_2, e_2), \bar{x}) \]
- There exists a partial recursive function \( g : \mathbb{N}^{n+1} \simeq \mathbb{N} \) s.t.
  \[ g(e_2, \bar{x}) \simeq f(S^1_n(e_2, e_2), \bar{x}) \]
- If \( e_1 \) is an index for \( g \) we obtain the desired equation.
  \[ \{e_1\}^{n+1}(e_2, \bar{x}) \simeq f(S^1_n(e_2, e_2), \bar{x}) \]

Complete Proof of Rec. Theorem

Let \( e_1 \) be s.t.

\[ \{e_1\}^{n+1}(y, \bar{x}) \simeq f(S^1_n(y, y), \bar{x}) . \]

Let \( e := S^1_n(e_1, e_1) \).

Then we have

\[
\begin{align*}
\{e\}^n(\bar{x}) & \leq S^1_n(e_1, e_1) \simeq \{S^1_n(e_1, e_1)\}^n(\bar{x}) \\
\text{S-m-n theorem} & \simeq \{e_1\}^{n+1}(e_1, \bar{x}) \\
\text{Def of } e_1 & \simeq f(S^1_n(e_1, e_1), \bar{x}) \\
e & \leq S^1_n(e_1, e_1) \simeq f(\bar{x}).
\end{align*}
\]