Proof Practice

Prove or disprove:
The empty set $\emptyset$ is recursive.

Proof that it is recursive:
In order to prove that a set $B$ is recursive, we have to show that there is a computable predicate $P_B$ such that:

$B = \{ x \in \mathbb{N} \mid P_B(x) \}$

For the empty set, $P_\emptyset$ has to be false for every $x$.

$P_\emptyset$ is computable, because it is computed, for instance, by the empty program.

---

Proof Practice

Prove or disprove:
The set of all natural numbers $\mathbb{N}$ is recursive.

Proof that it is recursive:
In order to prove that a set $B$ is recursive, we have to show that there is a computable predicate $P_B$ such that:

$B = \{ x \in \mathbb{N} \mid P_B(x) \}$

For the set $\mathbb{N}$, $P_\mathbb{N}$ has to be true for every $x$.

$P_\mathbb{N}$ is computable, because it is computed, for instance, by the following program:

$Y \leftarrow Y + 1$

---

Proof Practice

Prove or disprove:
If $A$ and $B$ are recursive sets, then $A \cap B$ is also a recursive set.

Proof that $A \cap B$ is a recursive set:
If $A$ and $B$ are recursive, then there must be computable predicates $P_A$ and $P_B$ such that:

$A = \{ x \in \mathbb{N} \mid P_A(x) \}$

$B = \{ x \in \mathbb{N} \mid P_B(x) \}$

Then we need to show that there is a computable predicate $P_{A \cap B}$ such that:

$A \cap B = \{ x \in \mathbb{N} \mid P_{A \cap B}(x) \}$

The following program computes $P_{A \cap B}$:

IF $\neg P_A(X)$ GOTO E
IF $\neg P_B(X)$ GOTO E
$Y \leftarrow Y + 1$

Or this one:

$Y \leftarrow P_A(X) \cdot P_B(X)$

---

Proof Practice

Prove or disprove:
The set $B$ of all natural numbers whose associated programs halt on inputs 4 and 7 is r.e.

Proof that it is r.e.:
In order to prove that $B$ is r.e., we have to show that there is a partially computable function $f_B(x)$ such that:

$B = \{ x \in \mathbb{N} \mid f_B(x) \downarrow \}$

The following program computes $f_B(x)$:

$Z \leftarrow U_4(4, X)$
$Z \leftarrow U_7(7, X)$

---

Proof Practice

Prove or disprove:
The set $B$ of all natural numbers whose associated programs do not halt on input 5 is r.e.

Proof that it is not r.e.:
Assume that $B$ is r.e. This would require that we can compute for a given program whether it halts on a given input (here: input 5).

In that case, the predicate $\text{HALT}(x, y)$ would be computable. However, we already proved that $\text{HALT}(x, y)$ is not computable.

This contradiction shows that $B$ cannot be r.e.
Theorem 5.1 (Parameter Theorem):
For each \( n, m > 0 \) there is a primitive recursive function \( S_m^n(u_1, \ldots, u_n, y) \) such that
\[
\Phi_{m+n}(x_1, \ldots, x_m, u_1, \ldots, u_n, y) = \Phi_m(x_1, \ldots, x_m, S_m^n(u_1, \ldots, u_n, y)).
\]
Suppose that the values for \( u_1, \ldots, u_n, \) and \( y \) are fixed.
Then the left side of the equation is a partially computable function of the \( m \) arguments \( x_1, \ldots, x_m \).
Let the number of the program that computes this function be \( q \). Then we have:
\[
\Phi_{m+n}(x_1, \ldots, x_m, u_1, \ldots, u_n, y) = \Phi_m(x_1, \ldots, x_m, q).
\]
The parameter theorem tells us that there exists such a \( q \) that can be obtained from \( u_1, \ldots, u_n, \) and \( y \) by a primitive recursive function.

Let us take a look at the case \( n = 1 \):
\[
\Phi_{m+1}(x_1, \ldots, x_m, u, y) = \Phi_m(x_1, \ldots, x_m, S_m(u, y)).
\]
Here, \( S_m(u, y) \) is the number of a program that receives inputs \( x_1, \ldots, x_m \) and computes the same value as program number \( y \) does on inputs \( x_1, \ldots, x_m, u \).
We can easily obtain \( S_m(u, y) \) by writing the instruction \( X_m+1 \leftarrow u \) and then appending the program with number \( y \).
This works similarly for any given \( n \), which can be proven by mathematical induction (see page 86 in the textbook).

Theorem 6.1 (Recursion Theorem):
Let \( g(z, x_1, \ldots, x_m) \) be a partially computable function of \((m + 1)\) variables. Then there is a number \( e \) such that
\[
\Phi_e(x_1, \ldots, x_m) = g(e, x_1, \ldots, x_m).
\]
Proof: Consider the partially computable function \( g(S_m(v, v), x_1, \ldots, x_m) \) where \( S_m^1 \) is the function that occurs in the parameter theorem.
Clearly, there must be a program that takes inputs \( x_1, \ldots, x_m \) and \( v \) and computes function \( g \). Let the number of such a program be \( z_0 \). Then we have:
\[
g(S_m^1(v, v), x_1, \ldots, x_m) = \Phi^{m+1}(x_1, \ldots, x_m, v, z_0)
\]
Applying the Parameter Theorem:
\[
g(S_m^1(v, v), x_1, \ldots, x_m) = \Phi^{m+1}(x_1, \ldots, x_m, S_m^1(v, z_0))
\]
Setting \( v = z_0 \):
\[
g(S_m^1(z_0, z_0), x_1, \ldots, x_m) = \Phi^{m+1}(x_1, \ldots, x_m, S_m^1(z_0, z_0))
\]
Setting \( e = S_m^1(z_0, z_0) \):
\[
g(e, x_1, \ldots, x_m) = \Phi^{m+1}(x_1, \ldots, x_m, e) = \Phi^{m+1}(x_1, \ldots, x_m)
\]
End of proof.

Corollary 6.2:
There is a number \( e \) such that for all \( x \):
\[
\Phi_e(x) = x.
\]
Proof: Consider the computable projection function \( g(z, x) = u_1^2(z, x) = z \).
With the help of the Recursion Theorem, we can find a number \( e \) such that
\[
\Phi_e(x) = g(e, x) = e.
\]
Such programs generate copies of themselves.
Rice’s Theorem

Let $\Gamma$ be some collection of partially computable functions of one variable. We associate with $\Gamma$ the following index set $R_\Gamma$:

$$R_\Gamma = \{ t \in \mathbb{N} \mid \Phi_t \in \Gamma \}.$$ 

$R_\Gamma$ will be a recursive set if there is an algorithm which accepts as input the number $t$ of a program and returns the value TRUE or FALSE depending on whether or not the function computed by this program belongs to $\Gamma$.

Rice’s Theorem

Some examples are:

1. $\Gamma$ is the set of computable functions;
2. $\Gamma$ is the set of primitive recursive functions;
3. $\Gamma$ is the set of partially computable functions which are defined for all but a finite number of values of $x$.

Theorem 7.1 (Rice’s Theorem): Let $\Gamma$ be a collection of partially computable functions of one variable. Let there be partially computable functions $f(x)$ and $g(x)$ such that $f(x)$ belongs to $\Gamma$ but $g(x)$ does not. Then $R_\Gamma$ is not recursive.

Rice’s Theorem

Rice’s Theorem tells us that there is no way to algorithmically determine non-trivial properties of the function computed by another program. Trivial properties are those that apply to all partially computable functions or none of them. The theorem uses functions $f(x)$ and $g(x)$ - such that $f(x)$ belongs to a collection $\Gamma$ but $g(x)$ does not - for the sole purpose of excluding such trivial cases.