First recursion theorem

The First Recursion Theorem, also known as Kleene's Recursion Theorem, states that every recursive functional has a least fixed point that is computable. In other words, given a recursive functional Φ : FUN(N^k) → FUN(N^k), there exists a computable function f_Φ : N^k → N such that:

1. Φ(f*Φ*) = *f*Φ (fixed point property)
2. For any g ∈ FUN(N^k), if Φ(g) = g, then f_Φ ⊆ g (least fixed point property)

THEOREM 17.9 (First recursion theorem (Kleene)). *Let* $\Phi : \mathcal{F}(\mathbb{N}^k) \to \mathcal{F}(\mathbb{N}^h)$ *be a* ***recursive functional****. Then* Φ *has a* ***least fixed point*** f_Φ *which is* ***computable****, i.e.*

$$(1)\ \Phi(f_\Phi) = f_\Phi$$

$$(2)\ \forall g \in \mathcal{F}(\mathbb{N}^k) \quad \Phi(g) = g \Rightarrow f_\Phi \subseteq g$$

$$(3)\ f_\Phi \text{ is computable}$$

and we can see that $f_\Phi = \bigcup_n \Phi^n(\varnothing)$.

This theorem is used in the course to:

1. Prove the existence of fixed points for recursive functionals, such as the Ackermann functional. By showing that the Ackermann functional is recursive, we can conclude that it has a computable least fixed point, which is the Ackermann function itself.
2. Classify sets as recursively enumerable or not. For example, the set A = {x | φ_x(y) = x^2 for infinitely many y} can be shown to be recursively enumerable but not recursive using the First Recursion Theorem.
3. Prove undecidability results. As a corollary of the First Recursion Theorem, we can prove Rice's Theorem, which states that any non-trivial property of computable functions is undecidable. This, in turn, is used to prove the undecidability of the Halting Problem.

Second recursion theorem

The Second Recursion Theorem, also known as Kleene's Fixed Point Theorem, states that for any total computable function f : N → N, there exists a fixed point e ∈ N such that φ_e = φ_f(e). In other words, there is a program e that, when executed, behaves exactly like the program obtained by applying f to e.

Let $f : \mathbb{N} \to \mathbb{N}$ be total, computable and extensional i.e.

$$\forall e, e' \ \varphi_e = \varphi_{e'} \Rightarrow \varphi_{f(e)} = \varphi_{f(e')}$$

Then, by Theorem 17.8 (Myhill-Shephedson) there exists a unique recursive functional Φ such that

$$\forall e \in \mathbb{N} \quad \Phi(\varphi_e) = \varphi_{f(e)}$$

Since Φ is recursive, by the First Recursion Theorem (Theorem 17.9) it has a least fixed point $f_\Phi : \mathbb{N} \to \mathbb{N}$ computable. Therefore there is $e_0 \in \mathbb{N}$ such that

$$\varphi_{e_0} = f_\Phi = \Phi(f_\Phi) = \Phi(\varphi_{e_0}) = \varphi_{f(e_0)}$$

This means that if f is total computable and extensional, then there exists e_0 such that

$$\varphi_{e_0} = \varphi_{f(e_0)}$$

The second recursion theorem states that this holds also when f is not extensional.

THEOREM 18.1 (Second recursion theorem (Kleene)). *Let $f : \mathbb{N} \to \mathbb{N}$ a total computable function. Then there exists $e_0 \in \mathbb{N}$ such that*

$$\varphi_{e_0} = \varphi_{f(e_0)}$$

This theorem is used in the course to:

1. Show the non-extensionality of certain sets. For example, the Halting Set K = {x | φ_x(x)↓} can be proven to be non-extensional using the Second Recursion Theorem. The proof involves constructing a program that behaves differently when given its own index as input.
2. Demonstrate the existence of self-referential programs. The Second Recursion Theorem allows us to create programs that can access their own source code or index during execution. This is a powerful technique for constructing counterexamples and proving the limitations of computable functions.