22 Arithmetic Representability

In the previous lectures we have introduced axiomatizations of various fragments of mathematics, including integer arithmetic. We have given two formalizations – the axioms of an inductively ordered integral domain and the Peano axioms. In the process the number of axioms has grown tremendously. If we were to rely just on the pure tableau calculus for first-order logic, we would hardly be able to prove anything from these axioms in a decent amount of time. The size of the proofs will just be too big for that.

From a practical perspective, therefore, it makes little sense to continue the path of adding axioms to first-order logic. Instead, we have to turn these axioms into proof rules that we can apply ad lib and even extend the proof system by decision procedures for certain fragments of mathematics, which are guaranteed to be correct but use extra-logical features to construct a proof. Doing so will bring us to a formalism that we call **P**roof **R**efinement **L**ogic, and in particular to a version of it that we call λ-PRL, which enables us to reason about programs, integers and lists of integers. There is also a successor of λ-PRL that is based on a much richer formal logic called type theory, but introducing that logic and its applications is a course by itself.

However, before we do so, let us explore the theoretical consequences of the axiomatizations we have so far. The Peano axioms appear so utterly simple – *can we express anything interesting in them*? They include substitution and induction as axiom *schemes*, which potentially means an infinite number of axioms – *how far would we get if we require the number of axioms to be truly finite*? These two questions will lead us to the investigation of computable functions, issues of decidability, and eventually to what is known as Gödel's incompleteness theorem.

22.1 Representability of Functions in a Formal Theory

The theory of computable functions has been the subject of mathematical studies since the early 1930's. A variety of formal models has been developed, such as the *λ-calculus* (functional programming), *Turing machines* (imperative programming), *recursive functions*, and a huge number of programming languages. All these models have turned out to be equivalent. None is stronger than the others, so we can be quite sure that they do represent the class of all computable functions completely – if a function is computable, then it can be represented in each of these formalisms (this became known as *Church's thesis*). Most theoretical models of computability focus on functions on natural numbers, so we will only consider the domain $\mathbb{N}$ from now on.

In the following we will show that the Peano axioms are strong enough to express all computable functions as well. For this purpose, of course, we have to define what it means for a logical formula to represent a function. Let us keep in mind that the language of Peano Arithmetic includes only the language of first-order logic (predicates, variables, parameters, logical connectives, and quantifiers), the designated parameters `0` and `1`, the designated equality predicate `=`, and the designated function symbols `+` and `*`. So, while the numbers 0 and 1 have a direct representation by a symbol of Peano Arithmetic, all the other numbers have to be represented by terms. The number 2 is represented by `0+1+1`, 3 by `0+1+1+1`, and so on.[3] In the following discourse we need to be able to denote the term that represents an arbitrary natural number n. We choose the notation $\underline{\boldsymbol{n}}$ for this purpose and define

$$\underline{\boldsymbol{n}} \equiv \underbrace{\texttt{0+1+1+ ... 1}}_{n \text{ times}}$$

Note that $\underline{\boldsymbol{n}}$ is a term of the object language of Peano Arithmetic while n itself is a meta-level symbol representing a natural number.

How can we represent computable functions by logical formulas? Recall that in the axiomatization of integers as inductively ordered integral domain we introduced function symbols as more conventional notation for predicates that satisfied the functionality axiom. A term of the form f`(`x`)=`y is only a different way of writing the predicate R_f`(`x`,`y`)`. Does that mean that the predicate R_f is a representation of the function f?

Not exactly. We have to keep in mind that a (computable) function is a *semantical* construct that maps natural numbers onto natural numbers, while the symbol f is still an object of the formal language (an abbreviation for an object level term, to be precise). However, we can relate a computational function to the semantical meaning of a predicate R_f by saying that $f(x) = y$ must hold if the corresponding formula is *valid*. The following definition makes this concept precise.

A formal *theory* $\mathcal{T}$ is a set of formulas over some formal language. A formula X is *valid in* $\mathcal{T}$ (Notation $\models_{\mathcal{T}} X$) if it is true in every model of $\mathcal{T}$. Often a theory $\mathcal{T}$ is defined by a set of axioms and the elements of $\mathcal{T}$, called *theorems*, are the logical consequences of these axioms. Occasionally people identify the theory with this set of axioms, but this is not entirely accurate. In the following we will focus on theories whose language includes a representation of natural numbers.

Definition: An n-ary function $f : \mathbb{N} \to \mathbb{N}$ is called *representable in a theory* $\mathcal{T}$ if there is an (n+1)-ary predicate R_f in the formal language of $\mathcal{T}$, such that for all $x_1, .., x_n, y \in \mathbb{N}$

- $f(x_1, .., x_n) = y$ implies $\models_T R_f(\underline{\boldsymbol{x_1}}, \ldots, \underline{\boldsymbol{x_n}}, \underline{\boldsymbol{y}})$
- $f(x_1, .., x_n) \neq y$ implies $\models_T \sim R_f(\underline{\boldsymbol{x_1}}, \ldots, \underline{\boldsymbol{x_n}}, \underline{\boldsymbol{y}})$

In a weak sense, representability in a theory can be viewed as some form of logic programming. Note that the requirements on the predicate R_f representing a function f are quite strict. It is not sufficient that $R_f(\underline{\boldsymbol{x_1}}, \ldots, \underline{\boldsymbol{x_n}}, \underline{\boldsymbol{y}})$ is not valid in $\mathcal{T}$ if $f(x_1, .., x_n) \neq y$.

[3] It should be noted that 0, 1, 2, 3 etc. are only symbols of the meta-language that we use to denote the actual natural number. They are not the numbers themselves. However, as it is more convenient to write 0, 1, 2, 3 than "the number zero", "the number one", "the number two", "the number three", we will use the conventional numbers as placeholders.

As a result, it becomes more difficult to distinguish between the meta-level symbol that represents a number, and the symbol of the logical object language that is supposed to express the same. In this text we use `blue typewriter font` for object language expressions while the other fonts are reserved for the meta-language. Such a distinction, however, cannot be made on the blackboard. Note also that $math{-}font$ in formulas indicates a placeholder for an arbitrary term or formula.

Instead, the negation of this formula must be valid, which means that R_f must be a very exact representation and that the theory $\mathcal{T}$ is capable of expressing that.

Q: $\boxed{\textit{What kind of functions can be represented in Peano Arithmetic?}}$

Let us consider a few examples:

- Obviously addition, successor, and multiplication can be represented in Peano arithmetic.
  - *Addition* $+$ is represented by the predicate R_+ with $R_+(x,y,z) \equiv x+y=z$.
  - The *successor function* s is represented by the predicate R_s with $R_s(x,y) \equiv x+1=y$.
  - *Multiplication* $*$ is represented by the predicate R_* with $R_*(x,y,z) \equiv x*y=z$.

  In all three cases it is obvious that the representation is correct.

- The *predecessor function* p inverts the successor function on non-zero inputs. That means hat $p(x)=y$ iff $x=0$ and $y=0$ or if $x=y+1$. Based on this analysis we define the predicate R_p as $R_p(x,y) \equiv (x=\underline{0} \wedge y=\underline{0}) \vee y+1=x$.

  What remains to show is that this representation is in fact correct, that is that $p(x) = y$ implies the validity of $R_p(\underline{\boldsymbol{x}},\underline{\boldsymbol{y}})$ in Peano Arithmetic and that $p(x) \neq y$ implies the validity of $\sim R_p(\underline{\boldsymbol{x}},\underline{\boldsymbol{y}})$. Fortunately, we are not required to give a *formal* proof for the validity of these formulas, which would be almost impossible. Instead, we can use the axioms of Peano Arithmetic within a *semantical* argument that the formulas are valid in every model of Peano Arithmetic. We proceed by induction over the number x.

  1. For $x=0$ we know $p(x)=0$. Consider two possibilities for the number y.

     If $y=0$ then $p(x)=y$ and $R_p(\underline{\boldsymbol{x}},\underline{\boldsymbol{y}}) = (\underline{0}=\underline{0} \wedge \underline{0}=\underline{0}) \vee \underline{0}+1=\underline{0}$ is valid because of the reflexivity of equality.

     If $y \neq 0$, then $p(x) \neq y$ and $\sim R_p(\underline{\boldsymbol{x}},\underline{\boldsymbol{y}}) = \sim((\underline{0}=\underline{0} \wedge \underline{\boldsymbol{y}}=\underline{0}) \vee \underline{\boldsymbol{y}}+1=\underline{0})$ is valid because $\underline{\boldsymbol{y}}$ has the form `0+..+1`. As a consequence the axiom `non-surjective` makes both disjuncts false. Thus their negation becomes true in every model of Peano Arithmetic.

  2. Assume the claim holds for $x=n$. For $x=n+1$ we know $p(x)=n$, and $\underline{\boldsymbol{x}} = \underline{\boldsymbol{n}}+1$. Again we consider two possibilities for the number y.

     If $y=n$ then $p(x)=y$ and $R_p(\underline{\boldsymbol{x}},\underline{\boldsymbol{y}}) = (\underline{\boldsymbol{n}}+1=\underline{0} \wedge \underline{\boldsymbol{y}}=\underline{0}) \vee \underline{\boldsymbol{n}}+1=\underline{\boldsymbol{n}}+1$ is valid because of the reflexivity of equality.

     If $y \neq n$, then $p(x) \neq y$ and $\sim R_p(\underline{\boldsymbol{x}},\underline{\boldsymbol{y}}) = \sim((\underline{\boldsymbol{n}}+1=\underline{0} \wedge \underline{\boldsymbol{y}}=\underline{0}) \vee \underline{\boldsymbol{y}}+1=\underline{\boldsymbol{n}}+1)$. The left disjunct in this formula is false because of the axiom `non-surjective`. Since $y \neq n$, $\underline{\boldsymbol{y}}$ must have a form different from $\underline{\boldsymbol{n}}$, which means that the right disjunct either has the form $\underline{\boldsymbol{n}+\boldsymbol{i}}+1+1=\underline{\boldsymbol{n}}+1$ for some number i if $y>n$ or the form $\underline{\boldsymbol{y}}+1=\underline{\boldsymbol{y}+\boldsymbol{j}}+1$ for some number j if $y<n$. Thus by repeatedly applying the axiom `injective` we get either $\underline{\boldsymbol{i}}+1=0$ or $0=\underline{\boldsymbol{j}}+1$. Because of the axiom `non-surjective` and the symmetry of equality, both possibilities evaluate to false, which means that the overall formula must become true in every model of Peano Arithmetic.

  The example of the predecessor function shows that the representation of a function is often easy to find, while proving the correctness of this representation turns out to be complicated even in simple cases, since we have to investigate the validity of two formulas for all possible values for the input and output of the function. Usually, this requires an inductive proof, but fortunately this induction takes place on the meta-level – we are not required to use the induction axiom. This insight is very important in the study of formal theories that do not have an induction axiom (like the theory $\mathcal{Q}$ that we will discuss in the next lecture).

Writing that a function f is represented by a predicate R_f, where $R_f(x_1,..,x_n,y)$ is defined as some specific formula becomes somewhat tedious on the long run. In the following we will use a more sloppy notation and say that f is represented by that formula. For instance, we would simply say that the predecessor function is represented by the formula $(x{=}\underline{\mathbf{0}} \wedge y{=}\underline{\mathbf{0}}) \vee y{+}1{=}x$.

- *Subtraction* on natural numbers is an inverse of addition that floors a zero. That means hat $x{-}y{=}z$ iff $x < y$ and $z{=}0$ or if $x{=}y{+}z$. Thus subtraction can be represented by the formula $(\mathtt{x{<}y} \wedge \mathtt{z{=}\underline{0}}) \vee \mathtt{y{+}z{=}x}$. For the sake of clarity we used an abbreviation $x{<}y$ in this formula, which is defined as $x{<}y \equiv (\exists \mathtt{z})((x\mathtt{+z})\mathtt{+1} = y)$.

22.2 Representing Computable Functions in Peano Arithmetic

We now want to show that Peano Arithmetic is sufficiently strong to represent all computable functions. We will do this by looking at the so-called μ-recursive functions, a mathematical model for computability that is closest to formal logic. μ-recursive functions are as expressive as any other model of computability and thus well-suited for our purpose. They are defined as follows.

Definition: The class of *μ-recursive functions* is recursively defined as follows.

- The *successor function* $s:\mathbb{N}{\to}\mathbb{N}$ with $s(x) = x{+}1$ is μ-recursive.
- The *constant function* $c_k:\mathbb{N}^0{\to}\mathbb{N}$ with $c_k() = k$ is μ-recursive for all $k \in \mathbb{N}$.
- The *projection function* $\pi^n{}_i:\mathbb{N}^n{\to}\mathbb{N}$ with $\pi^n{}_i(x_1,..x_n) = x_i$ is μ-recursive for all $1{\leq}i{\leq}n \in \mathbb{N}$.
- If the functions $f_1,..,f_k:\mathbb{N}^n{\to}\mathbb{N}$ and $g:\mathbb{N}^k{\to}\mathbb{N}$ are recursive then the *composition* $h = g{\circ}f_1,..,f_k:\mathbb{N}^n{\to}\mathbb{N}$, defined as
  – $h(x_1,..x_n) = g(\,f_1(x_1,..x_n),...,f_k(x_1,..x_n))$
  is μ-recursive.
- If the functions $f:\mathbb{N}^n{\to}\mathbb{N}$ and $g:\mathbb{N}^{n+2}{\to}\mathbb{N}$ are recursive then the *primitive recursion* $h = \mathsf{pr}(f,g):\mathbb{N}^{n+1}{\to}\mathbb{N}$, defined as
  – $h(x_1,..x_n,0) = f(x_1,..x_n)$ and
  – $h(x_1,..x_n,k{+}1) = g(x_1,..x_n,\ k,\ h(x_1,..x_n,k))$
  is μ-recursive.
- If $f:\mathbb{N}^{n+1}{\to}\mathbb{N}$ is recursive then the *minimization* $h = \mu f:\mathbb{N}^n{\to}\mathbb{N}$, defined as
  – $h(x_1,..x_n) = \mathsf{min}\{\,z\,|\,f(x_1,..x_n,z){=}0\}$
  is μ-recursive.

Note that minimization doesn't terminate if $f(x_1, ..x_n, y){\neq}0$ for all y. μ-recursive functions are often called *recursive functions*, as long as this notion is unambiguous.

Theorem: All μ-recursive functions are representable in Peano Arithmetic.

Proof: We demonstrate how to represent all the constructs mentioned above.

- The *successor function* can be represented by the formula x+1=y.
- The *constant function* c_k can be represented by the formula y=$\underline{\boldsymbol{k}}$.
- The *projection function* $\pi^n{}_i$ can be represented by the formula y=x_i, or, to be precise, by a predicate $\mathtt{R}_{\pi_i^n}$ with $\mathtt{R}_{\pi_i^n}(x_1,..x_n,y) \equiv y{=}x_i$. It is easy to see that
  - $\pi^n{}_i(x_1, ..x_n){=}x_i{=}y$ implies that $\mathtt{R}_{\pi_i^n}(\underline{\boldsymbol{x_1}},..,\underline{\boldsymbol{x_n}},\underline{\boldsymbol{y}}) \equiv \underline{\boldsymbol{x_i}}{=}\underline{\boldsymbol{x_i}}$ is valid.
  - $\pi^n{}_i(x_1, ..x_n){=}x_i{\neq}y$ implies that $\sim\mathtt{R}_{\pi_i^n}(\underline{\boldsymbol{x_1}},..,\underline{\boldsymbol{x_n}},\underline{\boldsymbol{y}}) \equiv \sim(\underline{\boldsymbol{y}}{=}\underline{\boldsymbol{x_i}})$ is valid.
- The *composition* $h = g{\circ}f_1, .., f_k$ can be represented by the formula
  $(\exists z_1,..z_k)(\ \mathtt{R}_{f_1}(x_1,..x_n,z_1)\ \wedge\ \dots\ \wedge\ \mathtt{R}_{f_k}(x_1,..x_n,z_k)\ \wedge\ \mathtt{R}_g(z_1,..z_k,y)$),
  where $\mathtt{R}_{f_1}$,..., $\mathtt{R}_{f_k}$, and $\mathtt{R}_g$ are the predicates representing $f_1, .., f_k$, and g respectively. The formula encodes the forwarding of results from the f_i into g
- The *minimization* $h = \mu f$ can be represented by the formula
  $(\forall \mathtt{z})(\mathtt{z}{\leq}y \supset (\exists \mathtt{t})((\mathtt{R}_f(x_1,..x_n,\mathtt{z},\mathtt{t}) \wedge \mathtt{t}{=}0) \Leftrightarrow \mathtt{z}{=}y))$,
  where $\mathtt{R}_f$ represents f. The formula encodes an unrestriced search, stating that if y is the first value with $f(x_1, ..x_n, y){=}0$, which means that for $z{\leq}y$ we know $f(x_1, ..x_n, z){=}0$ if and only if $z{=}y$. It is not necessary to express the results of the search for $z{>}y$, asince representability only requires an answer for the question whether a given number y can be the desired result.
- Representing he *primitive recursion* $h = \mathsf{pr}(f, g)$ by a formula in Peano Arithmetic is the most demanding part of the proof. The language of Peano Arithmetic has addition, multiplication, conjunction for composition, quantifiers for search, but no inductive constructor – only an axiom that allows us to *prove* formulas by induction.

  Thus, rather than trying to give a direct representation of primitive recursion in Peano Arithmetic, we will describe how to express it in terms of addition, multiplication, and the remaining constructs for building μ-recursive functions. Since all these constructs are representable in Peano Arithmetic, primitive recursion must be representable a well.

  How can we compute the value of $h(x_1, ..x_n, k)$ without using primitive recursion? We first calculate the computations sequence $h(x_1, ..x_n, 0)$, $h(x_1, ..x_n, 1), \dots h(x_1, ..x_n, k)$ and then select the last element of that sequence. To compute this sequence, we enumerate all possible sequences $y_0, y_1, \dots, y_k$ and check whether they satisfy the requirements of primitive recursion, i.e. $f(x_1, ..x_n){=}y_0$ and $g(x_1, ..x_n, i, y_i){=}y_{i+1}$.

  Enumerating sequences requires us to represent them by numbers that can be decoded back into sequences. A straightforward, though computationally not very effective way is to represent a sequence $y_0, y_1, \dots, y_k$ by a uniquely decodable polynomial.[4]

[4] Computationally, the representation of primitive recursion in Peano Arithmetic is extremely inefficient. But for our proof this is irrelevant, as we only need to show *that* every computable function can be represented.

1. We first search for the smallest prime number p that is larger than all the y_i
2. We then represent $y_0, y_1, \ldots, y_k$ by the number $\sigma \equiv y_0{}^* + y_1{}^* p + \ldots + y_k{}^* p^k$
3. To make sure that σ can be decoded again, we couple it with the prime p and define $\hat{y} \equiv \langle p,\sigma \rangle$, where $\langle\rangle : \mathbb{N}^2 \to \mathbb{N}$ is a bijective encoding of pairs of numbers, defined by $\langle i,j \rangle \equiv j + (i+j)(i+j+1)/2$.
4. Given $\hat{y}$ and a number i we can compute the value of the element y_i as follows. We first extract p and σ by using the inverses of $\langle\rangle$, which can be expressed as
   – $\langle z \rangle_1 \equiv \mathsf{min}\{ i \mid (\exists j)(j \leq z \wedge \langle i, j \rangle = z)\}$ and
   – $\langle z \rangle_2 \equiv \mathsf{min}\{ j \mid (\exists i)(i \leq z \wedge \langle i, j \rangle = z)\}$.

   We then divide $\hat{y}$ by p^{i+1} to isolate the elements from i upwards and compute the remainder of dividing the result by p again to get the smallest of these elements. Let us call the result of this computation $\hat{y}{\star}i$.

To determine the sequence we need, we then have to search for the smallest number z that encodes a sequence and satisfies the recursive equations. We can then select $z{\star}k$ to get the final result of the computation. Altogether we get

$$\begin{aligned}
&\mathsf{pr}(f,g)(x_1,..x_n,k)\\
&= \mathsf{min}\{ z \mid f(x_1,..x_n) = z{\star}0 \wedge (\forall i)(i \leq k \supset g(x_1,..x_n,i,z{\star}i) = z{\star}(i+1))\} \star k
\end{aligned}$$

The above expression can be expressed in terms of addition, multiplication, minimization, composition, projections, and constants. To prove this, we only have to show how to express a primality test, summation of polynomials, exponentiation, division, quotient remainder, the operation $\hat{y}{\star}i$, and limited quantification using only these constructs. Since all of these are are already shown to be representable in Peano Arithmetic, primitive recursion can be represented in Peano Arithmetic as well. $\square$