

## Systems for arithmetic

### 1. Gödel's $\mathcal{T}$

In an earlier chapter we introduced typed combinatory logic over a set of base types  $A$ . In this chapter on arithmetic we will consider this system again, but we only work with one base type, which we will think of as the set of natural numbers  $\mathbb{N}$  and which is traditionally denoted by  $0$ . We will also have three new constants and two new reduction rules and the result will be a rewriting system which is called Gödel's  $\mathcal{T}$ .

**DEFINITION 1.1.** The finite types are defined by induction as follows:  $0$  is a finite type, and if  $\sigma$  and  $\tau$  are finite types, then so are  $\sigma \rightarrow \tau$  and  $\sigma \times \tau$ .

Besides the usual combinators  $\mathbf{k}$ ,  $\mathbf{s}$ ,  $\mathbf{p}$ ,  $\mathbf{p}_0$  and  $\mathbf{p}_i$ , Gödel's  $\mathcal{T}$  includes the following constants: a constant  $0$  of type  $0$ , which stands for the natural number zero, a constant  $S$  of type  $0 \rightarrow 0$  which stands for the successor function and for each type  $\sigma$  a "recursor"  $\mathbf{R}^\sigma$  of type  $\sigma \rightarrow ((0 \rightarrow (\sigma \rightarrow \sigma)) \rightarrow (0 \rightarrow \sigma))$ , whose meaning will be explained shortly. So the constants (combinators) of Gödel's  $\mathcal{T}$  are:

- (1) for each pair of types  $\sigma, \tau$  a combinator  $\mathbf{k}^{\sigma, \tau}$  of sort  $\sigma \rightarrow (\tau \rightarrow \sigma)$ .
- (2) for each triple of types  $\rho, \sigma, \tau$  a combinator  $\mathbf{s}^{\rho, \sigma, \tau}$  of type  $(\rho \rightarrow (\sigma \rightarrow \tau)) \rightarrow ((\rho \rightarrow \sigma) \rightarrow (\rho \rightarrow \tau))$ .
- (3) for each pair of types  $\rho, \sigma$  combinators  $\mathbf{p}^{\rho, \sigma}, \mathbf{p}_0^{\rho, \sigma}, \mathbf{p}_1^{\rho, \sigma}$  of types  $\rho \rightarrow (\sigma \rightarrow \rho \times \sigma)$ ,  $\rho \times \sigma \rightarrow \rho$  and  $\rho \times \sigma \rightarrow \sigma$ , respectively.
- (4) a constant  $0$  of type  $0$  and a constant  $S$  of type  $0 \rightarrow 0$ .
- (5) for each type  $\sigma$  a combinator  $\mathbf{R}^\sigma$  ("the recursor") of type  $\sigma \rightarrow ((0 \rightarrow (\sigma \rightarrow \sigma)) \rightarrow (0 \rightarrow \sigma))$ .

The terms are built as before:

**DEFINITION 1.2.** The terms of a certain type are defined inductively as follows:

- each variable or constant of type  $\sigma$  will be a term of type  $\sigma$ .
- if  $f$  is a term of type  $\sigma \rightarrow \tau$  and  $x$  is a term of type  $\sigma$ , then  $f(x)$  is a term of type  $\tau$ .

The convention is that an expression like  $fxyz$  has to be read as  $((f(x)y)z)$ .

Besides the old reduction rules there will be two new ones, as follows:

**DEFINITION 1.3.** An expression on the left of the table below is called a *redex*. If  $t$  is a redex and  $t'$  is the corresponding expression on the right of the table, then we will say that  $t$

converts to  $t'$  and we will write  $t \text{ conv } t'$ .

$t$	$t'$
$\mathbf{k}t_1t_2$	$t_1$
$\mathbf{s}t_1t_2t_3$	$t_1t_3(t_2t_3)$
$\mathbf{p}_i(\mathbf{p}t_0t_1)$	$t_i$
$\mathbf{R}t_1t_20$	$t_1$
$\mathbf{R}t_1t_2(Sn)$	$t_2n(\mathbf{R}t_1t_2n)$

The definition of the reduction relation, a reduction sequence, a normal form *et cetera* are as before.

The final two reduction rules tell us how we should think of the recursor. The recursor of type  $\sigma$  yields for every pair of values  $t_1$  of type  $\sigma$  and  $t_2$  of type  $0 \rightarrow (\sigma \rightarrow \sigma)$  a sequence of objects of type  $\sigma$ , defined by recursion. That is,  $\mathbf{R}t_1t_2$  is a function of type  $0 \rightarrow \sigma$ , so expects a natural number  $m$  as input. If  $m = 0$ , then  $\mathbf{R}t_1t_2m$  is  $t_1$ , which is the starting point of the recursion. If  $m = Sn$ , then we may assume that  $\mathbf{R}t_1t_2n$  has already been defined and the recursion should tell us how the new value  $\mathbf{R}t_1t_2(Sn)$  can be obtained from the value  $\mathbf{R}t_1t_2n$  and the natural number  $n$ . In fact, that is where  $t_2$  comes in, in telling us how the next value is to be computed from the previous:  $\mathbf{R}t_1t_2(Sn)$  is given by feeding  $t_2$  with  $n$  and the previous value  $\mathbf{R}t_1t_2n$ .

As for typed combinatory logic, we have that the rewriting rules for Gödel's  $\mathcal{T}$  make it both strongly normalising and confluent.

**THEOREM 1.4.** (Strong normalisation for Gödel's  $\mathcal{T}$ , Tait, 1967) *The rewriting system given by Gödel's  $\mathcal{T}$  is strongly normalising and confluent. That is, for any term  $t$  in Gödel's  $\mathcal{T}$  there is a number  $n$  such that every reduction sequence starting from  $t$  has length at most  $n$  and ends with the same term in normal form.*

**PROOF.** The proof is an adaptation of the strong normalisation proof for typed combinatory logic. For the student who is interested: besides the neutral expressions we had before, we now also have to include  $0$  and  $St$  as neutral expressions. With these modifications we can adapt the proof of Lemma 1.10 from Chapter 8. Also, one has to prove that  $\mathbf{R}t_1t_2t_3$  is computable whenever the  $t_i$  are, which one does by induction on  $\nu(t_1) + \nu(t_2) + \nu(t_3) + m$ , where  $m$  is the number of symbols in the normal form of  $t_3$ .  $\square$

And we can again implement  $\lambda$ -abstraction:

**PROPOSITION 1.5.** *For any variable  $x$  and term  $t$  in Gödel's  $\mathcal{T}$  there is another term denoted by  $\lambda x.t$  such that*

$$(\lambda x.t)t' \succeq t[t'/x].$$

**PROOF.** We use the same algorithm as before.  $\square$

One can repeat  $\lambda$ -abstraction as in  $\lambda x.\lambda y.t$ ; but instead of  $\lambda x.\lambda y.t$  we will usually write  $\lambda xy.t$ .

The result is a system in which one can do quite a bit of arithmetic. For example, there is a constant **plus** of type  $0 \rightarrow (0 \rightarrow 0)$  such that:

$$\begin{aligned} \mathbf{plus} \ m \ 0 &\succeq m \\ \mathbf{plus} \ m \ Sn &\succeq S(\mathbf{plus} \ m \ n) \end{aligned}$$

Indeed, one may define **plus** as

$$\mathbf{plus} = \lambda mn. \mathbf{R}m(\lambda xy. Sy)n$$

(please check!).

For example, we have:

$$\mathbf{plus}(SS0)(SS0) \succeq S[\mathbf{plus}(SS0)(S0)] \succeq SS[\mathbf{plus}(SS0)0] \succeq SSSS0.$$

So  $2 + 2 = 4$ , which you probably already knew.

In a similar way one can define terms for multiplication and exponentiation, for instance. Since the system is strongly normalising and normal forms are unique, and because the only closed terms in normal form of type 0 are numerals  $S^m 0$ , one can really do arithmetic inside Gödel's  $\mathcal{T}$ .

## 2. Arithmetic in all finite types

In this section we combine Gödel's  $\mathcal{T}$  with first-order logic to give us a system in which one can also prove some arithmetical theorems.

The system  $\mathbf{HA}^\omega$  is formulated in multi-sorted intuitionistic logic, with the sorts being the finite types and the terms being those of Gödel's  $\mathcal{T}$ .

DEFINITION 2.1. The formulas of  $\mathbf{HA}^\omega$  are defined inductively as follows:

- $\perp$  is a formula and if  $s$  and  $t$  are terms of Gödel's  $\mathcal{T}$  of the same type  $\sigma$ , then  $s =_\sigma t$  is a formula.
- if  $\varphi$  and  $\psi$  are formulas, then so are  $\varphi \wedge \psi$ ,  $\varphi \vee \psi$ ,  $\varphi \rightarrow \psi$ .
- if  $x$  is a variable of type  $\sigma$  and  $\varphi$  is a formula, then  $\exists x^\sigma \varphi$  and  $\forall x^\sigma \varphi$  are formulas.

The axioms and rules of  $\mathbf{HA}^\omega$  are:

- (i) All the axioms and rules of intuitionistic logic (say in Hilbert-style).
- (ii) Rules for equality at all types:

$$x =_\sigma x, \quad x =_\sigma y \rightarrow y =_\sigma x, \quad x =_\sigma y \wedge y =_\sigma z \rightarrow x =_\sigma z,$$

$$f =_{\sigma \rightarrow \tau} f' \wedge x =_\sigma x' \rightarrow fx =_\tau f'x'.$$

- (iii) The successor axioms:

$$\neg S(x) =_0 0, \quad S(x) =_0 S(y) \rightarrow x =_0 y$$

- (iv) For any formula  $\varphi$  in the language of  $\mathbf{HA}^\omega$ , the induction axiom:

$$\varphi(0) \rightarrow (\forall x^0 (\varphi(x) \rightarrow \varphi(Sx)) \rightarrow \forall x^0 \varphi(x)).$$

- (v) The axioms for the combinators:

$$\begin{aligned} \mathbf{k}xy &= x \\ \mathbf{s}xyz &= xz(yz) \\ \mathbf{p}_0(\mathbf{p}xy) &= x \\ \mathbf{p}_1(\mathbf{p}xy) &= y \end{aligned}$$

as well as for the recursor:

$$\begin{aligned}\mathbf{R}xy0 &= x \\ \mathbf{R}xy(Sn) &= yn(\mathbf{R}xyn)\end{aligned}$$

In  $\mathbf{HA}^\omega$  we cannot prove the following *extensionality axioms*:

$$\begin{aligned}\forall f^{\sigma \rightarrow \tau}, g^{\sigma \rightarrow \tau} \left( (\forall x^\sigma fx =_\tau gx) \rightarrow f =_{\sigma \rightarrow \tau} g \right) \\ \forall x^{\sigma \times \tau}, y^{\sigma \times \tau} \left( \mathbf{p}_0x =_\sigma \mathbf{p}_0y \wedge \mathbf{p}_1x =_\tau \mathbf{p}_1y \rightarrow x =_{\sigma \times \tau} y \right)\end{aligned}$$

The result of adding these axioms to  $\mathbf{HA}^\omega$  will be denoted by  $\mathbf{E-HA}^\omega$ .

To both  $\mathbf{HA}^\omega$  and  $\mathbf{E-HA}^\omega$  we can add the Law of Excluded Middle  $\varphi \vee \neg \varphi$  or Double Negation Elimination  $\neg \neg \varphi \rightarrow \varphi$ : we will denote the resulting systems by  $\mathbf{PA}^\omega$  and  $\mathbf{E-PA}^\omega$ , respectively.

Some remarks about these systems:

- (1) All these systems satisfy the deduction theorem, so one can freely use natural deduction to prove things in these systems.
- (2) For any formula  $\varphi(x)$  in the language of  $\mathbf{HA}^\omega$  we have

$$\mathbf{HA}^\omega \vdash x =_\sigma y \wedge \varphi(x) \rightarrow \varphi(y).$$

Indeed, this is quite easy to prove by induction on the structure of  $\varphi$ . And from this it follows that the same is provable in all other systems, because  $\mathbf{HA}^\omega$  is a subsystem of all of them.

- (3) If  $t \succeq t'$  in Gödel's  $\mathcal{T}$ , then  $\mathbf{HA}^\omega \vdash t = t'$ .
- (4) It is sometimes convenient to regard disjunction as a defined connective in  $\mathbf{HA}^\omega$ . The reason for this is that  $\varphi \vee \psi$  is provably equivalent to

$$\exists n^0 \left[ (n = 0 \rightarrow \varphi) \wedge (\neg n = 0 \rightarrow \psi) \right]$$

in  $\mathbf{HA}^\omega$ . Moreover, if we regard  $\varphi \vee \psi$  as an abbreviation for this expression, the logical axioms for disjunction can be proved on the basis of the other axioms of  $\mathbf{HA}^\omega$ .

In  $\mathbf{HA}^\omega$  one can now prove things like: addition is associative and commutative. Please try!