

# The Language

---

- *Individual variables:*  $a, b, c, f, g, h, m, n, x, y, z, \dots$
- Set of *individual constants*, containing at least:  
 $k, s$  (combinators),  $p, p_0, p_1$  (pairing and projections),  $0$  (zero),  $s_{\mathbb{N}}$  (successor),  $p_{\mathbb{N}}$  (predecessor) and  $d_{\mathbb{N}}$  (definition by numerical cases).
- *Terms* ( $r, s, t, \dots$ ) are built up from the variables and constants by means of the function symbol  $\cdot$  for (partial) application.  
We use  $(st)$  or  $st$  as an abbreviation for  $(s \cdot t)$  and adopt the convention of association to the left, i.e.  $s_1 s_2 \dots s_n$  stands for  $(\dots (s_1 \cdot s_2) \dots s_n)$ .
- The *atomic formulas* are  $s \downarrow$ ,  $\mathbb{N}(s)$  and  $s = t$ .  
Since we work with a logic of partial terms, it is not guaranteed that all terms have values, and  $s \downarrow$  has to be read as  $s$  is defined or  $s$  has a value.  
 $\mathbb{N}(s)$  says that  $s$  is a natural number.
- The *formulas* ( $A, B, C, \dots, \varphi, \psi, \chi, \dots$ ) are generated from the atomic formulas by closing against the usual propositional connectives and quantifiers.

# The Logic of Partial Terms

---

We use the following abbreviation:  $t \simeq s :\Leftrightarrow t \downarrow \vee s \downarrow \rightarrow t = s$ .

- A complete axiomatization of the classical propositional logic.
- Quantifier axioms
  - (Q1)  $(\forall x.\varphi) \wedge t \downarrow \rightarrow \varphi[t/x]$ ,
  - (Q2)  $\varphi[t/x] \wedge t \downarrow \rightarrow \exists x.\varphi$ .
- Equality axioms
  - (E1)  $x = x$ ,
  - (E2)  $t \simeq s \wedge \varphi(t) \rightarrow \varphi(s)$ .
- Strictness axioms
  - (S1)  $R(t_1, \dots, t_n) \rightarrow t_1 \downarrow \wedge \dots \wedge t_n \downarrow$ , if R is a  $n$ -ary relation symbol,
  - (S2)  $t s \downarrow \rightarrow t \downarrow \wedge s \downarrow$ ,
  - (S3)  $c \downarrow$ , if  $c$  is a constant,
  - (S4)  $x \downarrow$ .

As rules we have *Modus ponens* and the *usual* quantifier rules.

# ***The basic theory of operations and numbers*** BON

---

## **I. Partial combinatory algebra**

- (1)  $k x y = x$ ,
- (2)  $s x y \downarrow \wedge s x y z \simeq x z (y z)$ .

## **II. Pairing and projection**

- (3)  $p_0 x \downarrow \wedge p_1 x \downarrow$ ,
- (4)  $p_0 (p x y) = x \wedge p_1 (p x y) = y$ .

## **III. Natural numbers**

- (5)  $N(0) \wedge \forall x.N(x) \rightarrow N(s_N x)$ ,
- (6)  $\forall x.N(x) \rightarrow s_N x \neq 0 \wedge p_N (s_N x) = x$ ,
- (7)  $\forall x.N(x) \wedge x \neq 0 \rightarrow N(p_N x) \wedge s_N (p_N x) = x$ .

## **IV. Definition by cases on $\mathbb{N}$**

- (8)  $N(v) \wedge N(w) \wedge v = w \rightarrow d_N x y v w = x$ ,
- (9)  $N(v) \wedge N(w) \wedge v \neq w \rightarrow d_N x y v w = y$ .

# $\lambda$ abstraction

---

**Theorem** For every variable  $x$  and every term  $t$ , there exists a term  $\lambda x.t$  whose free variables are those of  $t$ , excluding  $x$ , such that

$$\text{BON} \vdash \lambda x.t \downarrow \wedge (\lambda x.t) x \simeq t \quad \text{and} \quad \text{BON} \vdash s \downarrow \rightarrow (\lambda x.t) s \simeq t[s/x].$$

$(\lambda x.t)$  is inductively defined as follows:

$$\begin{array}{ll} \lambda x.x := s \mathbf{k} \mathbf{k}, & \widehat{\lambda} x.x := s \mathbf{k} \mathbf{k}, \\ \lambda x.y := \mathbf{k} y, & \text{if } x \neq y, \quad \widehat{\lambda} x.t := \mathbf{k} t, \quad \text{if } x \notin \text{FV}(t), \\ \lambda x.c := \mathbf{k} c, & \text{if } c \text{ is a constant,} \\ \lambda x.r s := s (\lambda x.r) (\lambda x.s). & \widehat{\lambda} x.r s := s (\widehat{\lambda} x.r) (\widehat{\lambda} x.s). \end{array}$$

This definition ensures  $\lambda x.t \downarrow$  even for  $\neg t \downarrow$ . However, we would have  $\neg \widehat{\lambda} x.t \downarrow$  because of strictness.

**Remark** In contrast to the definition of  $\widehat{\lambda}$  — and ordinary  $\lambda$  calculus —  $\lambda$  is not compatible with substitution:

For  $x \neq y$  the equality  $(\lambda x.t)[s/y] = \lambda x.t[s/y]$  does not hold in general.

# Recursion Theorem

---

**Theorem** There exists a term  $\text{rec}$  such that

$$\text{BON} \vdash \text{rec } f \downarrow \wedge \forall x. \text{rec } f x \simeq f (\text{rec } f) x.$$

$$\text{rec} := \lambda g. (\lambda y, z. g (y y) z) (\lambda y, z. g (y y) z).$$

$$\begin{aligned} \text{rec } f &\simeq (\lambda g. (\lambda y, z. g (y y) z) (\lambda y, z. g (y y) z)) f \\ &\simeq (\lambda y, z. f (y y) z) (\lambda y, z. f (y y) z) \end{aligned}$$

$$\begin{aligned} \text{rec } f x &\simeq (\lambda y, z. f (y y) z) (\lambda y, z. f (y y) z) x \\ &\simeq f ((\lambda y, z. f (y y) z) (\lambda y, z. f (y y) z)) x \\ &\simeq f (\text{rec } f) x \end{aligned}$$

**Remark** In the *total* setting one can define a term  $\text{rec}'$  such that  $\text{rec}' f = f (\text{rec}' f)$ . In the *partial* setting the recursion equation holds only *pointwise*.

# Solving recursions

---

$f x \simeq t[f, x]$  can be solved by  $f := \text{rec} (\lambda g, x.t[g, x])$ .

$$\begin{aligned} f x &\simeq \text{rec} (\lambda g, x.t[g, x]) x \\ &\simeq (\lambda g, x.t[g, x]) (\text{rec} (\lambda g, x.t[g, x])) x \\ &\simeq (\lambda g, x.t[g, x]) f x \\ &\simeq t[f, x] \end{aligned}$$

## Example

$$x + y = \begin{cases} x & \text{if } y = 0, \\ (x + (y - 1)) + 1 & \text{if } y \neq 0. \end{cases}$$

$\text{plus } x y = d_s x (s_N (\text{plus } x (p_N y))) y 0$

$\text{plus} := \text{rec} (\lambda f, x, y.d_s x (s_N (f x (p_N y))) y 0)$

Question: What about “Totality”:

$\forall n, m \in \mathbb{N}. (\text{plus } n m) \in \mathbb{N}$ ?

# ***Non-strict case distinction***

---

From Strictness follows

$$d_N r s u v \downarrow \rightarrow r \downarrow \wedge s \downarrow$$

To define a (recursive) function by case definition, we introduce the following “strong” definition by cases:

$$d_s r s u v := d_N (\lambda z.r) (\lambda z.s) u v 0$$

where the variable  $z$  does not occur in the terms  $r$  and  $s$ .

# Induction

---

$$(\mathcal{L}\text{-I}_N) \quad \varphi(0) \wedge (\forall x \in N. \varphi(x) \rightarrow \varphi(s_N x)) \rightarrow \forall x \in N. \varphi(x)$$

$(\mathcal{L}\text{-I}_N)$  can be used to prove *totality* of a “function”  $f$  on  $N$ :

$$\forall x \in N. f x \in N.$$

It is immediate that *Peano Arithmetic* PA can be embedded in  $\text{BON} + (\mathcal{L}\text{-I}_N)$ .

# Models

---

BON +  $(\mathcal{L}\text{-I}_N)$  has a natural recursion-theoretic interpretation, where the term universe is interpreted by the natural numbers and the application  $a \cdot b$  is translated into  $\{a\}(b)$  ( $\{n\}$  for  $n = 0, 1, 2, 3, \dots$  is a standard enumeration of the partial recursive functions).

**Theorem** BON +  $(\mathcal{L}\text{-I}_N)$  and PA are proof-theoretic equivalent.

Another class of models are given by *term models*. The universe consists of all (closed) terms of the language; a reduction relation is defined based on rewrite rules for the combinators; equality is defined as “having a common reduct”; Natural numbers are those terms which reduce to a numeral.

Existence can be interpreted either as “having a normal form”; or it can be trivialized by taking the full universe.

# *Relation to Computer Science*

---

Applicative Theories provide a framework to prove *directly* properties — such as: correctness, termination, etc. — of (the pure functional core) of *Functional programming languages*.

There are two main distinctions for functional programming languages: typed vs. type-free and strict vs. lazy. Applicative Theories follow the type-free and strict paradigm. SCHEME is an example of a functional programming language for which applicative theories can be used.

*Types* are not build in in the definition of the language (which would result in syntactatic restrictions), but represented as *predicates*, for example N.

That a term “has” a certain type has to be proven within the theory.

- Applicative theories allow to introduce new “datatypes” by adding new constants and theirs characteristic axioms, *without any coding*.
- The partial logic allows to speak about *termination* within the language.

# Explicit Mathematics

---

The language of Explicit Mathematics is as for BON, but including additionally

- individual constants  $c_e$ ,  $e \in \mathbb{N}$  (elementary comprehension) and  $j$  (join),
- *Type variables*:  $X, Y, Z, \dots$ ,
- $=$  is used also on the type level,
- Two binary relation symbols  $\in$  (membership) and  $\mathfrak{R}$  (naming; representation); their first arguments are (individual) terms, the second arguments types.

The formulas are build as usual, from the extended set of atomic formulae, including *quantification* over type variables:  $\exists X.\varphi$  and  $\forall X.\varphi$ .

$t \in X$  is read  $t$  belongs to  $X$ ;

$\mathfrak{R}(t, X)$  is read  $t$  is a name of  $X$ .

## Ontological axioms

- Every type has a name:  $\exists x. \mathfrak{R}(x, U)$ ,
- One name names only one type:  $\mathfrak{R}(a, U) \wedge \mathfrak{R}(a, V) \rightarrow U = V$ ,  
(but one type can have different names)
- Types are **extensional**:  $(\forall x. x \in U \leftrightarrow x \in V) \rightarrow U = V$ .

## Elementary Comprehension

An formula  $\varphi$  is called *elementary* if it contains neither the relation symbol  $\mathfrak{R}$  nor bound type variables.

Let  $\varphi(x, \vec{y}, \vec{Z})$  be an *elementary formula* with Gödel number  $m = \ulcorner \varphi(x, \vec{y}, \vec{Z}) \urcorner$

- $\exists X. \forall x. x \in X \leftrightarrow \varphi(x, \vec{y}, \vec{Z})$ ,
- $\mathfrak{R}(\vec{z}, \vec{Z}) \wedge (\forall x. x \in X \leftrightarrow \varphi(x, \vec{y}, \vec{Z})) \rightarrow \mathfrak{R}(c_m \vec{y} \vec{z}, X)$ .

We write  $t \in \{x | \varphi(x, \vec{y}, \vec{Z})\}$  for  $\forall X. \mathfrak{R}(\vec{z}, \vec{Z}) \wedge \mathfrak{R}(c_m \vec{y} \vec{z}, X) \rightarrow t \in X$  where  $m = \ulcorner \varphi(x, \vec{y}, \vec{Z}) \urcorner$ .

---

EET allows the straightforward definition of (names of) several useful types:

- $\text{Nat} := \{x \mid \mathbf{N}(x)\}$   
 $t \in \text{Nat} \leftrightarrow (\forall X. \mathfrak{R}(\mathbf{c}_n, X) \rightarrow t \in X)$  where  $n = \lceil \mathbf{N}(x) \rceil$   
 $t \in \text{Nat} \leftrightarrow (\forall X. \mathfrak{R}(\mathbf{c}_n, X) \rightarrow t \in X \wedge (\forall s. s \in X \leftrightarrow \mathbf{N}(s)))$   
 $t \in \text{Nat} \leftrightarrow \mathbf{N}(t)$
- $X \cup Y := \{x \mid x \in X \wedge x \in Y\}$
- $X \rightarrow Y := \{f \mid \forall x \in X. f x \in Y\}$
- $X \curvearrowright Y := \{f \mid \forall x \in X. f x \downarrow \rightarrow f x \in Y\}$
- $V := \{x \mid x = x\}$

The restriction to elementary formulae is essential, to avoid Russell-like types:

$$R := \{x \mid \exists X. \mathfrak{R}(x, X) \wedge \neg(x \in X)\}$$

- Names can be interpreted by *characteristic functions (algorithms)* of the types.
- While the types are extensional, the names are intensional.

# Join

---

$$\mathfrak{R}(t) := \exists X. \mathfrak{R}(t, X)$$

$$s \dot{\in} t := \exists X. \mathfrak{R}(t, X) \wedge s \in X$$

## Join

- $\mathfrak{R}(a, X) \wedge (\forall x \in X. \mathfrak{R}(f x)) \rightarrow \mathfrak{R}(j a f) \wedge \Sigma(a, f, j a f)$

In this axiom the formula  $\Sigma(a, f, b)$  expresses that  $b$  names the disjoint union of  $f$  over  $a$ , i.e.

$$\Sigma(a, f, b) := \forall x. x \dot{\in} b \leftrightarrow \exists y, z. x = \mathfrak{p} y z \wedge y \dot{\in} a \wedge z \dot{\in} f y.$$

# Inductive Generation

---

## Inductive Generation

$$\text{Closed}(a, b, S) := (\forall x \in a)[(\forall y \in a)((y, x) \in b \rightarrow y \in S) \rightarrow x \in S].$$

Consider  $b$  as the code of a binary relation. Then this definition means that  $S$  is a type which contains a  $c \in a$  if all predecessors of  $c$  in  $a$  with respect to  $b$  belong to  $S$ .

*Inductive generation* is now given by the following axioms

$$\mathfrak{R}(a) \wedge \mathfrak{R}(b) \rightarrow \exists X. \mathfrak{R}(i(a, b), X) \wedge \text{Closed}(a, b, X),$$

$$\mathfrak{R}(a) \wedge \mathfrak{R}(b) \wedge \text{Closed}(a, b, \varphi) \rightarrow \forall x \in i(a, b). \varphi(x)$$

for all formulas  $\varphi(u)$ . Thus inductive generation states the existence of accessible parts (uniform in the corresponding names)

# *Feferman's theory* $T_0$

---

$$T_0 = \text{EET} + (\text{Join}) + (\text{IG}) + (\text{F-}I_N)$$

Theories of Explicit Mathematics play an important role in the proof-theoretic analysis of subsystems of second-order arithmetic; in particular,  $T_0$  was used in the ordinal analysis of  $(\Delta_2^1 - \text{CA}) + (\text{BI})$ .

# Least fixed points

---

**Example**  $f(x) = f(x + 1)$

Obviously, every constant function is a solution of this equation:

$$f_0 = \lambda x.0,$$

$$f_1 = \lambda x.1,$$

$$f_2 = \lambda x.2,$$

...

However, the *least* solution — with respect to the definedness-order — is the total undefined function, i.e. a function  $f_{\perp}$  such that  $\forall x. \neg f_{\perp}(x) \downarrow$

*All programming languages allowing to define recursive functions will give this solution!*

$\text{rec}(\lambda f, x.f(x + 1))$  yields just a fixed point of this functional equation, i.e. all we can prove about it is:

$$\begin{aligned} (\text{rec}(\lambda f, x.f(x + 1))) x &\simeq (\lambda f, x.f(x + 1)) (\text{rec}(\lambda f, x.f(x + 1))) x \\ &\simeq (\text{rec}(\lambda f, x.f(x + 1))) (x + 1) \end{aligned}$$

We can not prove (or disprove) that  $\text{rec}(\lambda f, x.f(x + 1))$  is (extensional) equivalent with  $f_0, f_1, f_2, \dots$  or  $f_{\perp}$ .

# The theory LFP

---

- Everything is a number

$$\forall x.N(x).$$

- Computability

$$\text{(Comp.1)} \quad \forall f \forall x \forall n \in \mathbf{N}. c(f, x, n) = 0 \vee c(f, x, n) = 1),$$

$$\text{(Comp.2)} \quad \forall f \forall x. f x \downarrow \leftrightarrow (\exists n \in \mathbf{N}. c(f, x, n) = 0).$$

$c(f, x, n) = 0$  expresses that the computation of  $f x$  has terminated after  $n$  steps.

$$\text{LFP} := \text{BON} + (\text{Comp}) + \forall x.N(x) + (\mathcal{L}\text{-I}_{\mathbf{N}}).$$

**Remark** All new axioms of LFP are compatible with the recursion-theoretic model (use Kleene's  $T$ -predicate to verify the computability axioms). They do not exceed the strength of Peano Arithmetic.

---

### Lemma

1. There exists a closed term  $\text{not}_N$  so that LFP proves  $\neg N(\text{not}_N)$ .
2. There exists a closed term  $b$  so that LFP proves  $\forall x. \neg b x \downarrow$ .

The first assertion holds already in BON:

$$\text{not}_N := \text{rec} (\lambda f, x. d_N 1 0 (f x) 0) 0.$$

or  $\text{not}_N := \text{rec } g$  with:

$$g x \simeq \begin{cases} 1 & \text{if } g x = 0, \\ 0 & \text{if } g x \in N \wedge g x \neq 0. \end{cases}$$

With the axiom  $\forall x. N(x)$  in LFP it follows that  $\neg \text{not}_N \downarrow$  (which is not provable in BON!).

So for the second assertion we set:

$$b := \lambda x. \text{not}_N$$

# Classes

---

A formula  $A$  containing exactly  $x$  as free variable will be called a **class**.

Let  $A$  and  $B$  be classes and let  $\varphi$  be an arbitrary formula.

$$\begin{aligned}t \in A &:= t \downarrow \wedge A[t/x], \\A \rightarrow B &:= \forall y. y \in A \rightarrow x y \in B, \\A \curvearrowright B &:= \forall y. y \in A \wedge x y \downarrow \rightarrow x y \in B, \\r \sqsubseteq s &:= r \downarrow \rightarrow r = s, \\f \sqsubseteq_{A \curvearrowright B} g &:= \forall x \in A. f x \sqsubseteq g x, \\f \cong_{A \curvearrowright B} g &:= f \sqsubseteq_{A \curvearrowright B} g \wedge g \sqsubseteq_{A \curvearrowright B} f.\end{aligned}$$

$r \sqsubseteq s$  says that if  $r$  has a value, then  $r$  is equal to  $s$ .

$f \sqsubseteq_{A \curvearrowright B} g$  says that for every  $x \in A$  if the computation  $f x$  terminates, then  $g x$  also terminates and both computations yield the same result.

**Definition** Let  $A$  be a class. A function  $f \in (A \rightarrow A)$  is called  **$A$ -monotonic**, if

$$\forall g \in A. \forall h \in A. g \sqsubseteq_A h \rightarrow f g \sqsubseteq_A f h.$$

# The Least Fixed Point Operator

---

**Lemma** There exist closed terms  $l$  and  $h$  such that LFP proves:

1.  $l g \downarrow$ ,
2.  $l g x \downarrow \leftrightarrow \exists n. h g n x \downarrow$ ,
3.  $l g x = z \rightarrow \exists n. h g n x = z$ .

We have to show later that we can replace the term  $g (l g)$  by a “finite approximation”  $g (h g n)$ .

$$h g n \simeq \begin{cases} b & \text{if } n = 0, \\ g(h g(p_N n)) & \text{otherwise.} \end{cases}$$

$$q g x n \simeq \begin{cases} 0 & \text{if } h g(p_0 n) x = p_1 n, \\ \text{not}_N & \text{otherwise.} \end{cases}$$

$$l := \lambda g \lambda x. p_1(p_0(\mu(\lambda y. c^3(q, g, x, p_0(y), p_1(y))))).$$

---

**Lemma** If  $g \in ((A \curvearrowright B) \rightarrow (A \curvearrowright B))$  is  $A \curvearrowright B$ -monotonic, then the following claims hold in LFP:

1.  $\forall n. \mathbf{h} g n \in A \curvearrowright B,$
2.  $\forall n. \mathbf{h} g n \sqsubseteq_{A \curvearrowright B} \mathbf{h} g (n + 1),$
3.  $\mathbf{l} g \in A \curvearrowright B,$
4.  $\forall n. \mathbf{h} g n \sqsubseteq_{A \curvearrowright B} \mathbf{l} g,$
5.  $\mathbf{l} g \sqsubseteq_{A \curvearrowright B} g(\mathbf{l} g),$
6.  $\forall l. \exists n. \forall x \in A. x \leq l \rightarrow \mathbf{l} g x \sqsubseteq \mathbf{h} g n x,$
7.  $\forall x \in A. \exists n. g(\mathbf{l} g) x \sqsubseteq g(\mathbf{h} g n) x.$

**Theorem** Let  $g \in ((A \curvearrowright B) \rightarrow (A \curvearrowright B))$  be  $A \curvearrowright B$ -monotonic

1.  $\mathbf{l} g \cong_{A \curvearrowright B} g(\mathbf{l} g),$
2.  $f \in A \curvearrowright B \wedge f \cong_{A \curvearrowright B} g f \rightarrow \mathbf{l} g \sqsubseteq_{A \curvearrowright B} f.$

# Example 1

---

Consider the following JAVA like method:

```
A m (B x) {  
  return m(x);  
}
```

The semantics of the method  $m$  is given as the least fixed point of the functional  $\lambda f. \lambda x. f x$  which is the everywhere undefined function.

$$\text{BON} \not\vdash \neg \text{rec} (\lambda f. \lambda x. f x) s \downarrow$$

Let  $V$  be the universal class  $x = x$  and  $\emptyset$  the empty class  $x \neq x$ . Then the functional  $\lambda f. \lambda x. f x$  is an element of  $(V \curvearrowright \emptyset) \rightarrow (V \curvearrowright \emptyset)$  and is of course  $V \curvearrowright \emptyset$ -monotonic.

$$\text{LFP} \vdash \lceil (\lambda f. \lambda x. f x) \in (V \curvearrowright \emptyset)$$

$$\text{LFP} \vdash \forall y. \neg \lceil (\lambda f. \lambda x. f x) y \downarrow$$

## Example 2

---

$$\begin{aligned} f_1 x &\simeq \begin{cases} 1 & \text{if } x = 1, \\ \text{not}_N & \text{otherwise} \end{cases} \\ f_2 x &\simeq \begin{cases} \text{not}_N & \text{if } x = 1, \\ 1 & \text{otherwise.} \end{cases} \\ g x &\simeq \begin{cases} f_1 & \text{if } x = f_1, \\ f_2 & \text{otherwise.} \end{cases} \end{aligned}$$

We have  $g \in ((V \curvearrowright V) \rightarrow (V \curvearrowright V))$ .

If  $f$  is a fixed point of  $g$  then we have either  $f = f_1$  or  $\forall x. f x \simeq f_2 x$ .

However,  $g$  does not have a least fixed point in the sense of  $\sqsubseteq_{(V \curvearrowright V)}$ , since it is not

$(V \curvearrowright V)$ -monotonic:

$f_1 \sqsubseteq_{(V \curvearrowright V)} \lambda x. 1$ , but  $\neg(g f_1 \sqsubseteq_{(V \curvearrowright V)} g (\lambda x. 1))$