CHAPTER 2

Computability in higher types

In this chapter we will develop a somewhat general view of computability theory, where not only numbers and functions appear as arguments, but also functionals of any finite type.

2.1. Abstract computability via information systems

There are two principles on which our notion of computability will be based: finite support and monotonicity.

It is a fundamental property of computation that evaluation must be finite. So in any evaluation of $\Phi(\varphi)$ the argument φ can be called upon only finitely many times, and hence the value – if defined – must be determined by some finite subfunction of φ. This is the principle of finite support.

Let us carry this discussion somewhat further and look at the situation one type higher up. Let $\mathcal{H}$ be a partial functional of type-3, mapping type-2 functionals Φ to natural numbers. Suppose Φ is given and $\mathcal{H}(\Phi)$ evaluates to a defined value. Again, evaluation must be finite. Hence the argument Φ can only be called on finitely many functions φ. Furthermore each such φ must be presented to Φ in a finite form (explicitly say, as a set of ordered pairs). In other words, $\mathcal{H}$ and also any type-2 argument Φ supplied to it must satisfy the finite support principle, and this must continue to apply as we move up through the types.

To describe this principle more precisely, we need to introduce the notion of a "finite approximation" Φ_0 of a functional Φ. By this we mean a finite set X of pairs (φ_0, n) such that (i) φ_0 is a finite function, (ii) $\Phi(\varphi_0)$ is defined with value n, and (iii) if (φ_0, n) and (φ_0', n') belong to X where φ_0 and φ_0' are "consistent", then $n = n'$. The essential idea here is that Φ should be viewed as the union of all its finite approximations. Using this notion of a finite approximation we can now formulate the

> *Principle of finite support.* If $\mathcal{H}(\Phi)$ is defined with value n, then there is a finite approximation Φ_0 of Φ such that $\mathcal{H}(\Phi_0)$ is defined with value n.

The monotonicity principle formalizes the simple idea that once $\mathcal{H}(\Phi)$ is evaluated, then the same value will be obtained no matter how the argument Φ is extended. This requires the notion of "extension". Φ' extends Φ if for any piece of data (φ_0, n) in Φ there exists another (φ_0', n) in Φ' such that φ_0 extends φ_0' (note the contravariance!). The second basic principle is then

> *Monotonicity principle*. If $\mathcal{H}(\Phi)$ is defined with value n and Φ' extends Φ, then also $\mathcal{H}(\Phi')$ is defined with value n.

An immediate consequence of finite support and monotonicity is that the behaviour of any functional is indeed determined by its set of finite approximations. For if Φ, Φ' have the same finite approximations and $\mathcal{H}(\Phi)$ is defined with value n, then by finite support, $\mathcal{H}(\Phi_0)$ is defined with value n for some finite approximation Φ_0, and then by monotonicity $\mathcal{H}(\Phi')$ is defined with value n. Thus $\mathcal{H}(\Phi) = \mathcal{H}(\Phi')$, for all $\mathcal{H}$.

This observation now allows us to formulate a notion of abstract computability:

> *Effectivity principle*. An object is computable just in case its set of finite approximations is (primitive) recursively enumerable (or equivalently, Σ_1^0-definable).

This is an "externally induced" notion of computability, and it is of definite interest to ask whether one can find an "internal" notion of computability coinciding with it. This can be done by means of a fixed point operator introduced into this framework by Platek; and the result mentioned is due to Plotkin (1978).

The general theory of computability concerns partial functions and partial operations on them. However, we are primarily interested in total objects, so once the theory of partial objects is developed, we can look for ways to extract the total ones. Then one can prove Kreisel's density theorem, wich says that the total functionals are dense in the space of all partial "continuous" functionals.

2.1.1. Information systems. The basic idea of information systems is to provide an axiomatic setting to describe approximations of abstract objects (like functions or functionals) by concrete, finite ones. We do not attempt to analyze the notion of "concreteness" or finiteness here, but rather take an arbitrary countable set A of "bits of data" or "tokens" as a basic notion to be explained axiomatically. In order to use such data to build approximations of abstract objects, we need a notion of "consistency", which determines when the elements of a finite set of tokens are consistent with

each other. We also need an "entailment relation" between consistent sets U of data and single tokens a, which intuitively expresses the fact that the information contained in U is sufficient to compute the bit of information a. The axioms below are a minor modification of Scott's (1982), due to Larsen and Winskel (1991).

DEFINITION. An *information system* is a structure $(A, \mathrm{Con}, \vdash)$ where A is a countable set (the *tokens*), Con is a non-empty set of finite subsets of A (the *consistent* sets) and $\vdash$ is a subset of $\mathrm{Con} \times A$ (the *entailment* relation), which satisfy

$$\begin{aligned}
&U \subseteq V \in \mathrm{Con} \to U \in \mathrm{Con},\\
&\{a\} \in \mathrm{Con},\\
&U \vdash a \to U \cup \{a\} \in \mathrm{Con},\\
&a \in U \in \mathrm{Con} \to U \vdash a,\\
&U, V \in \mathrm{Con} \to \forall_{a \in V}(U \vdash a) \to V \vdash b \to U \vdash b.
\end{aligned}$$

The elements of Con are called *formal neighborhoods*. We use U, V, W to denote *finite* sets, and write

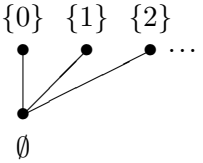
$$\begin{aligned}
&U \vdash V \quad \text{for} \quad U \in \mathrm{Con} \wedge \forall_{a \in V}(U \vdash a),\\
&a \uparrow b \quad \text{for} \quad \{a, b\} \in \mathrm{Con} \qquad (a, b \text{ are } \textit{consistent}),\\
&U \uparrow V \quad \text{for} \quad \forall_{a \in U, b \in V}(a \uparrow b).
\end{aligned}$$

DEFINITION. The *ideals* (also called *objects*) of an information system $\boldsymbol{A} = (A, \mathrm{Con}, \vdash)$ are defined to be those subsets x of A which satisfy

$$\begin{aligned}
&U \subseteq x \to U \in \mathrm{Con} \quad (x \text{ is } \textit{consistent}),\\
&x \supseteq U \vdash a \to a \in x \quad (x \text{ is } \textit{deductively closed}).
\end{aligned}$$

For example the *deductive closure* $\overline{U} := \{\, a \in A \mid U \vdash a \,\}$ of $U \in \mathrm{Con}$ is an ideal. The set of all ideals of $\boldsymbol{A}$ is denoted by $|\boldsymbol{A}|$.

EXAMPLES. Every countable set A can be turned into a *flat* information system by letting the set of tokens be A, $\mathrm{Con} := \{\emptyset\} \cup \{\, \{a\} \mid a \in A \,\}$ and $U \vdash a$ mean $a \in U$. In this case the ideals are just the elements of Con. For $A = \mathbb{N}$ we have the following picture of the Con-sets.



A rather important example is the following, which concerns approximations of functions from a countable set A into a countable set B. The tokens are the pairs (a, b) with $a \in A$ and $b \in B$, and

$$\mathrm{Con} := \{\, \{\, (a_i, b_i) \mid i < k \,\} \mid \forall_{i,j<k}(a_i = a_j \to b_i = b_j) \,\},$$
$$U \vdash (a, b) := (a, b) \in U.$$

It is not difficult to verify that this defines an information system whose ideals are (the graphs of) all partial functions from A to B.

2.1.2. Function spaces. We now define the "function space" $\boldsymbol{A} \to \boldsymbol{B}$ between two information systems $\boldsymbol{A}$ and $\boldsymbol{B}$.

Definition. Let $\boldsymbol{A} = (A, \mathrm{Con}_A, \vdash_A)$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$ be information systems. Define $\boldsymbol{A} \to \boldsymbol{B} = (C, \mathrm{Con}, \vdash)$ by

$$C := \mathrm{Con}_A \times B,$$
$$\{\, (U_i, b_i) \mid i \in I \,\} \in \mathrm{Con} := \forall_{J \subseteq I}(\bigcup_{j \in J} U_j \in \mathrm{Con}_A \to \{\, b_j \mid j \in J \,\} \in \mathrm{Con}_B).$$

For the definition of the entailment relation $\vdash$ it is helpful to first define the notion of an *application* of $W := \{\, (U_i, b_i) \mid i \in I \,\} \in \mathrm{Con}$ to $U \in \mathrm{Con}_A$:

$$\{\, (U_i, b_i) \mid i \in I \,\}U := \{\, b_i \mid U \vdash_A U_i \,\}.$$

From the definition of Con we know that this set is in Con_B. Now define $W \vdash (U, b)$ by $WU \vdash_B b$.

Clearly application is *monotone in the second argument*, in the sense that $U \vdash_A U'$ implies ($WU' \subseteq WU$, hence also) $WU \vdash_B WU'$. In fact, application is also *monotone in the first argument*, i.e.,

$$W \vdash W' \quad \text{implies} \quad WU \vdash_B W'U.$$

To see this let $W = \{\, (U_i, b_i) \mid i \in I \,\}$ and $W' = \{\, (U'_j, b'_j) \mid j \in J \,\}$. By definition $W'U = \{\, b'_j \mid U \vdash_A U'_j \,\}$. Now fix j such that $U \vdash_A U'_j$; we must show $WU \vdash_B b'_j$. By assumption $W \vdash (U'_j, b'_j)$, hence $WU'_j \vdash_B b'_j$. Because of $WU \supseteq WU'_j$ the claim follows.

Lemma. *If $\boldsymbol{A}$ and $\boldsymbol{B}$ are information systems, then so is $\boldsymbol{A} \to \boldsymbol{B}$ defined as above.*

Proof. Let $\boldsymbol{A} = (A, \mathrm{Con}_A, \vdash_A)$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$. The first, second and fourth property of the definition are clearly satisfied. For the third, suppose

$$\{(U_1, b_1), \ldots, (U_n, b_n)\} \vdash (U, b), \quad \text{i.e.,} \quad \{\, b_j \mid U \vdash_A U_j \,\} \vdash_B b.$$

We have to show that $\{(U_1, b_1), \ldots, (U_n, b_n), (U, b)\} \in \mathrm{Con}$. So let $I \subseteq \{1, \ldots, n\}$ and suppose

$$U \cup \bigcup_{i \in I} U_i \in \mathrm{Con}_A.$$

We must show that $\{b\} \cup \{\, b_i \mid i \in I \,\} \in \mathrm{Con}_B$. Let $J \subseteq \{1, \ldots, n\}$ consist of those j with $U \vdash_A U_j$. Then also

$$U \cup \bigcup_{i \in I} U_i \cup \bigcup_{j \in J} U_j \in \mathrm{Con}_A.$$

Since

$$\bigcup_{i \in I} U_i \cup \bigcup_{j \in J} U_j \in \mathrm{Con}_A,$$

from the consistency of $\{(U_1, b_1), \ldots, (U_n, b_n)\}$ we can conclude that

$$\{\, b_i \mid i \in I \,\} \cup \{\, b_j \mid j \in J \,\} \in \mathrm{Con}_B.$$

But $\{\, b_j \mid j \in J \,\} \vdash_B b$ by assumption. Hence

$$\{\, b_i \mid i \in I \,\} \cup \{\, b_j \mid j \in J \,\} \cup \{b\} \in \mathrm{Con}_B.$$

For the final property, suppose

$$W \vdash W' \quad \text{and} \quad W' \vdash (U, b).$$

We have to show $W \vdash (U, b)$, i.e., $WU \vdash_B b$. We obtain $WU \vdash_B W'U$ by monotonicity in the first argument, and $W'U \vdash b$ by definition. □

We shall now give two alternative characterizations of the function space: firstly as "approximable maps", and secondly as continuous maps w.r.t. the so-called Scott topology.

The basic idea for approximable maps is the desire to study "information respecting" maps from $\boldsymbol{A}$ into $\boldsymbol{B}$. Such a map is given by a relation r between Con_A and B, where $r(U, b)$ intuitively means that whenever we are given the information $U \in \mathrm{Con}_A$, then we know that at least the token b appears in the value.

Definition. Let $\boldsymbol{A} = (A, \mathrm{Con}_A, \vdash_A)$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$ be information systems. A relation $r \subseteq \mathrm{Con}_A \times B$ is an *approximable map* if it satisfies the following:

(a) if $r(U, b_1), \ldots, r(U, b_n)$, then $\{b_1, \ldots, b_n\} \in \mathrm{Con}_B$;
(b) if $r(U, b_1), \ldots, r(U, b_n)$ and $\{b_1, \ldots, b_n\} \vdash_B b$, then $r(U, b)$;
(c) if $r(U', b)$ and $U \vdash_A U'$, then $r(U, b)$.

We write $r \colon \boldsymbol{A} \to \boldsymbol{B}$ to mean that r is an approximable map from $\boldsymbol{A}$ to $\boldsymbol{B}$.

THEOREM. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be information systems. Then the ideals of $\boldsymbol{A} \to \boldsymbol{B}$ are exactly the approximable maps from $\boldsymbol{A}$ to $\boldsymbol{B}$.*

PROOF. Let $\boldsymbol{A} = (A, \mathrm{Con}_A, \vdash_A)$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$. If $r \in |\boldsymbol{A} \to \boldsymbol{B}|$ then $r \subseteq \mathrm{Con}_A \times B$ is consistent and deductively closed. We have to show that r satisfies the axioms for approximable maps.

(a) Let $r(U, b_1), \ldots, r(U, b_n)$. We must show that $\{b_1, \ldots, b_n\} \in \mathrm{Con}_B$. But this clearly follows from the consistency of r.

(b) Let $r(U, b_1), \ldots, r(U, b_n)$ and $\{b_1, \ldots, b_n\} \vdash_B b$. We must show that $r(U, b)$. But

$$\{(U, b_1), \ldots, (U, b_n)\} \vdash (U, b)$$

by the definition of the entailment relation $\vdash$ in $\boldsymbol{A} \to \boldsymbol{B}$. Hence $r(U, b)$ since r is deductively closed.

(c) Let $U \vdash_A U'$ and $r(U', b)$. We must show that $r(U, b)$. But

$$\{(U', b)\} \vdash (U, b)$$

since $\{(U', b)\}U = \{b\}$ (which follows from $U \vdash_A U'$). Hence $r(U, b)$, again since r is deductively closed.

For the other direction suppose that $r \colon \boldsymbol{A} \to \boldsymbol{B}$ is an approximable map. We must show that $r \in |\boldsymbol{A} \to \boldsymbol{B}|$.

Consistency of r. Let $r(U_1, b_1), \ldots, r(U_n, b_n)$ and $U = \bigcup\{ U_i \mid i \in I \} \in \mathrm{Con}_A$ for some $I \subseteq \{1, \ldots, n\}$. We must show $\{ b_i \mid i \in I \} \in \mathrm{Con}_B$. From $r(U_i, b_i)$ and $U \vdash_A U_i$ we obtain $r(U, b_i)$ by axiom (c) for all $i \in I$, and hence $\{ b_i \mid i \in I \} \in \mathrm{Con}_B$ by axiom (a).

Deductive closure of r. Let $r(U_1, b_1), \ldots, r(U_n, b_n)$ and

$$W := \{(U_1, b_1), \ldots, (U_n, b_n)\} \vdash (U, b).$$

We must show $r(U, b)$. By definition of $\vdash$ for $\boldsymbol{A} \to \boldsymbol{B}$ we have $WU \vdash_B b$, which is $\{ b_i \mid U \vdash_A U_i \} \vdash_B b$. Further by our assumption $r(U_i, b_i)$ we know $r(U, b_i)$ by axiom (c) for all i with $U \vdash_A U_i$. Hence $r(U, b)$ by axiom (b). □

DEFINITION. Suppose $\boldsymbol{A} = (A, \mathrm{Con}, \vdash)$ is an information system and $U \in \mathrm{Con}$. Define $\mathcal{O}_U \subseteq |\boldsymbol{A}|$ by

$$\mathcal{O}_U := \{ x \in |\boldsymbol{A}| \mid U \subseteq x \}.$$

Note that, since the ideals $x \in |\boldsymbol{A}|$ are deductively closed, $x \in \mathcal{O}_U$ implies $\overline{U} \subseteq x$.

LEMMA. *The system of all $\mathcal{O}_U$ with $U \in \mathrm{Con}$ forms the basis of a topology on $|\boldsymbol{A}|$, called the Scott topology.*

PROOF. Suppose $U, V \in \mathrm{Con}$ and $x \in \mathcal{O}_U \cap \mathcal{O}_V$. We have to find $W \in \mathrm{Con}$ such that $x \in \mathcal{O}_W \subseteq \mathcal{O}_U \cap \mathcal{O}_V$. Choose $W = U \cup V$. □

LEMMA. *Let $\boldsymbol{A}$ be an information system and $\mathcal{O} \subseteq |\boldsymbol{A}|$. Then the following are equivalent.*

(a) *$\mathcal{O}$ is open in the Scott topology.*
(b) *$\mathcal{O}$ satisfies*
  (i) *If $x \in \mathcal{O}$ and $x \subseteq y$, then $y \in \mathcal{O}$ (Alexandrov condition).*
  (ii) *If $x \in \mathcal{O}$, then $\overline{U} \in \mathcal{O}$ for some $U \subseteq x$ (Scott condition).*
(c) *$\mathcal{O} = \bigcup_{\overline{U} \in \mathcal{O}} \mathcal{O}_U$.*

Hence open sets $\mathcal{O}$ may be seen as those determined by a (possibly infinite) system of finitely observable properties, namely all U such that $\overline{U} \in \mathcal{O}$.

PROOF. (a) → (b). If $\mathcal{O}$ is open, then $\mathcal{O}$ is the union of some $\mathcal{O}_U$'s, $U \in \mathrm{Con}$. Since each $\mathcal{O}_U$ is upwards closed, also $\mathcal{O}$ is; this proves the Alexandrov condition. For the Scott condition assume $x \in \mathcal{O}$. Then $x \in \mathcal{O}_U \subseteq \mathcal{O}$ for some $U \in \mathrm{Con}$. Note that $\overline{U} \in \mathcal{O}_U$, hence $\overline{U} \in \mathcal{O}$, and $U \subseteq x$ since $x \in \mathcal{O}_U$.

(b) → (c). Assume that $\mathcal{O} \subseteq |\boldsymbol{A}|$ satisfies the Alexandrov and Scott conditions. Let $x \in \mathcal{O}$. By the Scott condition, $\overline{U} \in \mathcal{O}$ for some $U \subseteq x$, so $x \in \mathcal{O}_U$ for this U. Conversely, let $x \in \mathcal{O}_U$ for some $\overline{U} \in \mathcal{O}$. Then $\overline{U} \subseteq x$. Now $x \in \mathcal{O}$ follows from $\overline{U} \in \mathcal{O}$ by the Alexandrov condition.

(c) → (a). The $\mathcal{O}_U$'s are the basic open sets of the Scott topology. □

We now give some simple characterizations of the continuous functions $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$. Call f *monotone* if $x \subseteq y$ implies $f(x) \subseteq f(y)$.

LEMMA. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be information systems and $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$. Then the following are equivalent.*

(a) *f is continuous w.r.t. the Scott topology.*
(b) *f is monotone and satisfies the "principle of finite support" PFS: If $b \in f(x)$, then $b \in f(\overline{U})$ for some $U \subseteq x$.*
(c) *f is monotone and commutes with directed unions: for every directed $D \subseteq |\boldsymbol{A}|$ (i.e., for any $x, y \in D$ there is a $z \in D$ such that $x, y \subseteq z$)*

$$f(\bigcup_{x \in D} x) = \bigcup_{x \in D} f(x).$$

Note that in (c) the set $\{\, f(x) \mid x \in D \,\}$ is directed by monotonicity of f; hence its union is indeed an ideal in $|\boldsymbol{A}|$. Note also that from PFS and monotonicity of f it follows immediately that if $V \subseteq f(x)$, then $V \subseteq f(\overline{U})$ for some $U \subseteq x$.

Hence continuous maps $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ are those that can be completely described from the point of view of finite approximations of the abstract objects $x \in |\boldsymbol{A}|$ and $f(x) \in |\boldsymbol{B}|$: Whenever we are given a finite approximation V to the value $f(x)$, then there is a finite approximation U to the argument x such that already $f(\overline{U})$ contains the information in V; note that by monotonicity $f(\overline{U}) \subseteq f(x)$.

PROOF. (a) $\to$ (b). Let f be continuous. Then for any basic open set $\mathcal{O}_V \subseteq |\boldsymbol{B}|$ (so $V \in \mathrm{Con}_B$) the set $f^{-1}[\mathcal{O}_V] = \{\, x \mid V \subseteq f(x) \,\}$ is open in $|\boldsymbol{A}|$. To prove monotonicity assume $x \subseteq y$; we must show $f(x) \subseteq f(y)$. So let $b \in f(x)$, i.e., $\{b\} \subseteq f(x)$. The open set $f^{-1}[\mathcal{O}_{\{b\}}] = \{\, z \mid \{b\} \subseteq f(z) \,\}$ satisfies the Alexandrov condition, so from $x \subseteq y$ we can infer $\{b\} \subseteq f(y)$, i.e., $b \in f(y)$. To prove PFS assume $b \in f(x)$. The open set $\{\, z \mid \{b\} \subseteq f(z) \,\}$ satisfies the Scott condition, so for some $U \subseteq x$ we have $\{b\} \subseteq f(\overline{U})$.

(b) $\to$ (a). Assume that f satisfies monotonicity and PFS. We must show that f is continuous, i.e., that for any fixed $V \in \mathrm{Con}_B$ the set $f^{-1}[\mathcal{O}_V] = \{\, x \mid V \subseteq f(x) \,\}$ is open. We prove

$$\{\, x \mid V \subseteq f(x) \,\} = \bigcup \{\, \mathcal{O}_U \mid U \in \mathrm{Con}_A \text{ and } V \subseteq f(\overline{U}) \,\}.$$

Let $V \subseteq f(x)$. Then by PFS $V \subseteq f(\overline{U})$ for some $U \in \mathrm{Con}_A$ such that $U \subseteq x$, and $U \subseteq x$ implies $x \in \mathcal{O}_U$. Conversely, let $x \in \mathcal{O}_U$ for some $U \in \mathrm{Con}_A$ such that $V \subseteq f(\overline{U})$. Then $\overline{U} \subseteq x$, hence $V \subseteq f(x)$ by monotonicity.

For (b) $\leftrightarrow$ (c) assume that f is monotone. Let f satisfy PFS, and $D \subseteq |\boldsymbol{A}|$ be directed. $f(\bigcup_{x\in D} x) \supseteq \bigcup_{x\in D} f(x)$ follows from monotonicity. For the reverse inclusion let $b \in f(\bigcup_{x\in D} x)$. Then by PFS $b \in f(\overline{U})$ for some $U \subseteq \bigcup_{x\in D} x$. From the directedness and the fact that U is finite we obtain $U \subseteq z$ for some $z \in D$. From $b \in f(\overline{U})$ and monotonicity infer $b \in f(z)$. Conversely, let f commute with directed unions, and assume $b \in f(x)$. Then

$$b \in f(x) = f(\bigcup_{U\subseteq x} \overline{U}) = \bigcup_{U\subseteq x} f(\overline{U}),$$

hence $b \in f(\overline{U})$ for some $U \subseteq x$. □

Clearly the identity and constant functions are continuous, and also the *composition* $g \circ f$ of continuous functions $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ and $g\colon |\boldsymbol{B}| \to |\boldsymbol{C}|$.

THEOREM. *Let $\boldsymbol{A}$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$ be information systems. Then the ideals of $\boldsymbol{A} \to \boldsymbol{B}$ are in a natural bijective correspondence with the continuous functions from $|\boldsymbol{A}|$ to $|\boldsymbol{B}|$, as follows.*

(a) *With any approximable map* $r\colon \boldsymbol{A} \to \boldsymbol{B}$ *we can associate a continuous function* $|r|\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ *by*

$$|r|(z) := \{\, b \in B \mid r(U,b) \text{ for some } U \subseteq z \,\}.$$

We call $|r|(z)$ *the* application *of* r *to* z.

(b) *Conversely, with any continuous function* $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ *we can associate an approximable map* $\hat{f}\colon \boldsymbol{A} \to \boldsymbol{B}$ *by*

$$\hat{f}(U,b) := (b \in f(\overline{U})).$$

These assignments are inverse to each other, i.e., $f = |\hat{f}|$ *and* $r = \widehat{|r|}$.

Proof. Let r be an ideal of $\boldsymbol{A} \to \boldsymbol{B}$; then by the theorem just proved r is an approximable map. We first show that $|r|$ is well-defined. So let $z \in |\boldsymbol{A}|$.

$|r|(z)$ is consistent: let $b_1, \dots, b_n \in |r|(z)$. Then there are $U_1, \dots, U_n \subseteq z$ such that $r(U_i, b_i)$. Hence $U := U_1 \cup \dots \cup U_n \subseteq z$ and $r(U, b_i)$ by axiom (c) of approximable maps. Now from axiom (a) we can conclude that $\{b_1, \dots, b_n\} \in \mathrm{Con}_B$.

$|r|(z)$ is deductively closed: let $b_1, \dots, b_n \in |r|(z)$ and $\{b_1, \dots, b_n\} \vdash_B b$. We must show $b \in |r|(z)$. As before we find $U \subseteq z$ such that $r(U, b_i)$. Now from axiom (b) we can conclude $r(U, b)$ and hence $b \in |r|(z)$.

Continuity of $|r|$ follows immediately from part (b) of the lemma above, since by definition $|r|$ is monotone and satisfies PFS.

Now let $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ be continuous. It is easy to verify that $\hat{f}$ is indeed an approximable map. Furthermore

$$\begin{aligned} b \in |\hat{f}|(z) &\leftrightarrow \hat{f}(U,b) && \text{for some } U \subseteq z \\ &\leftrightarrow b \in f(\overline{U}) && \text{for some } U \subseteq z \\ &\leftrightarrow b \in f(z) && \text{by monotonicity and PFS.} \end{aligned}$$

Finally, for any approximable map $r\colon \boldsymbol{A} \to \boldsymbol{B}$ we have

$$\begin{aligned} r(U,b) &\leftrightarrow \exists_{V \subseteq \overline{U}} r(V,b) \quad \text{by axiom (c) for approximable maps} \\ &\leftrightarrow b \in |r|(\overline{U}) \\ &\leftrightarrow \widehat{|r|}(U,b), \end{aligned}$$

so $r = \widehat{|r|}$. □

Moreover, one can easily check that

$$r \circ s := \{\, (U,c) \mid \exists_V ((U,V) \subseteq s \wedge (V,c) \in r) \,\}$$

is an approximable map (where $(U,V) := \{\, (U,b) \mid b \in V \,\}$), and

$$|r \circ s| = |r| \circ |s|, \quad \widehat{f \circ g} = \hat{f} \circ \hat{g}.$$

We usually write $r(z)$ for $|r|(z)$, and similarly $f(U,b)$ for $\hat{f}(U,b)$. It should always be clear from the context where the mods and hats should be inserted.

2.1.3. Algebras and types. We now consider concrete information systems, our basis for continuous functionals.

Types will be built from base types by the formation of function types, $\rho \to \sigma$. As domains for the base types we choose non-flat and possibly infinitary free algebras, given by their constructors. The main reason for taking non-flat base domains is that we want the constructors to be injective and with disjoint ranges. This generally is not the case for flat domains.

We inductively define *type forms*

$$\rho, \sigma ::= \alpha \mid \rho \to \sigma \mid \mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k}$$

with α, ξ type variables and $k \geq 1$ (since we want our algebras to be inhabited). Note that $(\rho_\nu)_{\nu<n} \to \sigma$ means $\rho_0 \to \ldots \to \rho_{n-1} \to \sigma$, associated to the right.

Let $\mathrm{FV}(\rho)$ denote the set of type variables free in ρ. We define $\mathrm{SP}(\alpha,\rho)$ "α occurs at most *strictly positive* in ρ" by induction on ρ.

$$\mathrm{SP}(\alpha,\beta) \qquad \frac{\alpha \notin \mathrm{FV}(\rho) \quad \mathrm{SP}(\alpha,\sigma)}{\mathrm{SP}(\alpha,\rho\to\sigma)} \qquad \frac{\mathrm{SP}(\alpha,\rho_{i\nu}) \text{ for all } i<k,\, \nu<n_i}{\mathrm{SP}(\alpha,\mu_\xi((\rho_{i\nu})_{\nu<n_i}\to\xi)_{i<k})}$$

Now we can define $\mathrm{Ty}(\rho)$ "ρ is a *type*", again by induction on ρ.

$$\mathrm{Ty}(\alpha) \qquad \frac{\mathrm{Ty}(\rho) \quad \mathrm{Ty}(\sigma)}{\mathrm{Ty}(\rho\to\sigma)}$$

$$\frac{\mathrm{Ty}(\rho_{i\nu}) \text{ and } \mathrm{SP}(\xi,\rho_{i\nu}) \text{ for all } i<k,\, \nu<n_i \qquad \xi \notin \mathrm{FV}(\rho_{0\nu}) \text{ for all } \nu<n_0}{\mathrm{Ty}(\mu_\xi((\rho_{i\nu})_{\nu<n_i}\to\xi)_{i<k})}$$

We call

$$\iota := \mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k}$$

an *algebra*. Sometimes it is helpful to display the type parameters and write $\iota(\vec{\alpha},\vec{\beta}\,)$, where $\vec{\alpha},\vec{\beta}$ are all type variables except ξ free in some $\rho_{i\nu}$, and $\vec{\alpha}$ are the ones occuring only strictly positive. If we write the i-th component of ι in the form $(\rho_\nu(\xi))_{\nu<n} \to \xi$, then we call

$$(\rho_\nu(\iota))_{\nu<n} \to \iota$$

the i-th *constructor type* of ι.

In $(\rho_\nu(\xi))_{\nu<n} \to \xi$ we call $\rho_\nu(\xi)$ a *parameter* argument type if ξ does not occur in it, and a *recursive* argument type otherwise. A recursive argument type $\rho_\nu(\xi)$ is *nested* if it has an occurrence of ξ in a strictly positive parameter position of another (previously defined) algebra, and unnested otherwise. An algebra ι is called *nested* if it has a constructor with at least one nested recursive argument type, and *unnested* otherwise.

Every type ρ should have a *total inhabitant*, i.e., a closed term of this type built solely from constructors, variables and assumed total inhabitants of some of its (type) variables. To ensure this we have required that for every algebra $\mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k}$ the initial $(\rho_{0\nu})_{\nu<n_0} \to \xi$ has no recursive argument types. Note that it might not be necessary to actually use assumed total inhabitants for all variables of a type. An example is the list type $\mathbf{L}(\alpha)$, which has the Nil constructor as a total inhabitant. However, for the type $\mathbf{L}(\alpha)^+$ $(:= \mu_\xi(\alpha \to \xi, \alpha \to \xi \to \xi))$ we need to assume a total inhabitant of α.

Here are some examples of algebras.

$$\begin{aligned}
\mathbf{U} &:= \mu_\xi \xi \qquad \text{(unit)},\\
\mathbf{B} &:= \mu_\xi(\xi, \xi) \qquad \text{(booleans)},\\
\mathbf{N} &:= \mu_\xi(\xi, \xi \to \xi) \qquad \text{(natural numbers, unary)},\\
\mathbf{P} &:= \mu_\xi(\xi, \xi \to \xi, \xi \to \xi) \qquad \text{(positive numbers, binary)},\\
\mathbf{D} &:= \mu_\xi(\xi, \xi \to \xi \to \xi) \qquad \text{(binary trees, or derivations)},\\
\mathbf{O} &:= \mu_\xi(\xi, \xi \to \xi, (\mathbf{N} \to \xi) \to \xi) \qquad \text{(ordinals)},\\
\mathbf{T}_0 &:= \mathbf{N}, \quad \mathbf{T}_{n+1} := \mu_\xi(\xi, (\mathbf{T}_n \to \xi) \to \xi) \qquad \text{(trees)}.
\end{aligned}$$

Examples of algebras strictly positive in their type parameters are

$$\begin{aligned}
\mathbf{L}(\alpha) &:= \mu_\xi(\xi, \alpha \to \xi \to \xi) \qquad \text{(lists)},\\
\alpha \times \beta &:= \mu_\xi(\alpha \to \beta \to \xi) \qquad \text{(product)},\\
\alpha + \beta &:= \mu_\xi(\alpha \to \xi, \beta \to \xi) \qquad \text{(sum)}.
\end{aligned}$$

An example of a nested algebra is

$$\mathbf{T} := \mu_\xi(\mathbf{L}(\xi) \to \xi) \quad \text{(finitely branching trees)}.$$

Note that $\mathbf{T}$ has a total inhabitant since $\mathbf{L}(\alpha)$ has one (given by the Nil constructor).

Let ρ be a type; we write $\rho(\vec{\alpha}\,)$ for ρ to indicate its dependence on the type parameters $\vec{\alpha}$. We can substitute types $\vec{\sigma}$ for $\vec{\alpha}$, to obtain $\rho(\vec{\sigma}\,)$. Examples are $\mathbf{L}(\mathbf{B})$, the type of lists of booleans, and $\mathbf{N} \times \mathbf{N}$, the type of pairs of natural numbers.

Note that often there are many equivalent ways to define a particular type. For instance, we could take $\mathbf{U}+\mathbf{U}$ to be the type of booleans, $\mathbf{L}(\mathbf{U})$ to be the type of natural numbers, and $\mathbf{L}(\mathbf{B})$ to be the type of positive binary numbers.

For every constructor type of an algebra we provide a (typed) *constructor symbol* C_i. In some cases they have standard names, for instance

$\mathrm{tt}^{\mathbf{B}}, \mathrm{ff}^{\mathbf{B}}$ for the two constructors of the type $\mathbf{B}$ of booleans,

$0^{\mathbf{N}}, \mathrm{S}^{\mathbf{N}\to\mathbf{N}}$ for the type $\mathbf{N}$ of (unary) natural numbers,

$1^{\mathbf{P}}, \mathrm{S}_0^{\mathbf{P}\to\mathbf{P}}, \mathrm{S}_1^{\mathbf{P}\to\mathbf{P}}$ for the type $\mathbf{P}$ of (binary) positive numbers,

$\mathrm{Nil}^{\mathbf{L}(\rho)}, \mathrm{Cons}^{\rho\to\mathbf{L}(\rho)\to\mathbf{L}(\rho)}$ for the type $\mathbf{L}(\rho)$ of lists,

$(\mathrm{Inl}_{\rho\sigma})^{\rho\to\rho+\sigma}, (\mathrm{Inr}_{\rho\sigma})^{\sigma\to\rho+\sigma}$ for the sum type $\rho+\sigma$,

$\mathrm{Branch}\colon \mathbf{L}(\mathbf{T})\to\mathbf{T}$ for the type $\mathbf{T}$ of finitely branching trees.

An algebra form ι is *structure-finitary* if all its argument types $\rho_{i\nu}$ are not of arrow form. It is *finitary* if in addition it has no type variables. In the examples above $\mathbf{U}$, $\mathbf{B}$, $\mathbf{N}$, $\mathbf{P}$ and $\mathbf{D}$ are all finitary, but $\mathbf{O}$ and $\mathbf{T}_{n+1}$ are not. $\mathbf{L}(\rho)$, $\rho\times\sigma$ and $\rho+\sigma$ are structure-finitary, and finitary if their parameter types are. The nested algebra $\mathbf{T}$ above is finitary.

An algebra is *explicit* if all its constructor types have parameter argument types only (i.e., no recursive argument types). In the examples above $\mathbf{U}$, $\mathbf{B}$, $\rho\times\sigma$ and $\rho+\sigma$ are explicit, but $\mathbf{N}$, $\mathbf{P}$, $\mathbf{L}(\rho)$, $\mathbf{D}$, $\mathbf{O}$, $\mathbf{T}_{n+1}$ and $\mathbf{T}$ are not.

We will also need the notion of the *level* of a type, which is defined by

$$\mathrm{lev}(\iota) := 0, \qquad \mathrm{lev}(\rho\to\sigma) := \max\{\mathrm{lev}(\sigma), 1+\mathrm{lev}(\rho)\}.$$

Base types are types of level 0, and a *higher* type has level at least 1.

2.1.4. Partial continuous functionals. For every type ρ we define the information system $\boldsymbol{C}_\rho = (C_\rho, \mathrm{Con}_\rho, \vdash_\rho)$. The ideals $x\in|\boldsymbol{C}_\rho|$ are the *partial continuous functionals* of type ρ. Since we will have $\boldsymbol{C}_{\rho\to\sigma} = \boldsymbol{C}_\rho\to\boldsymbol{C}_\sigma$, the partial continuous functionals of type $\rho\to\sigma$ will correspond to the continuous functions from $|\boldsymbol{C}_\rho|$ to $|\boldsymbol{C}_\sigma|$ w.r.t. the Scott topology. It will not be possible to define $\boldsymbol{C}_\rho$ by recursion on the type ρ, since we allow algebras with constructors having function arguments (like $\mathbf{O}$ and Sup). Instead, we shall use recursion on the "height" of the notions involved, defined below.

Definition (Information system of type ρ). We simultaneously define C_ι, $C_{\rho\to\sigma}$, Con_ι and $\mathrm{Con}_{\rho\to\sigma}$.

(a) The *tokens* $a \in C_\iota$ are the type correct constructor expressions $\mathrm{C}a_1^* \dots a_n^*$ where a_i^* is an *extended token*, i.e., a token or the special symbol $*$ which carries no information.
(b) The tokens in $C_{\rho\to\sigma}$ are the pairs (U, b) with $U \in \mathrm{Con}_\rho$ and $b \in C_\sigma$.
(c) A finite set U of tokens in C_ι is *consistent* (i.e., $\in \mathrm{Con}_\iota$) if all its elements start with the same constructor C, say of arity $\tau_1 \to \dots \to \tau_n \to \iota$, and all $U_i \in \mathrm{Con}_{\tau_i}$ for $i = 1, \dots, n$, where U_i consists of all (proper) tokens at the i-th argument position of some token in $U = \{\mathrm{C}\vec{a_1^*}, \dots, \mathrm{C}\vec{a_m^*}\}$.
(d) $\{\, (U_i, b_i) \mid i \in I \,\} \in \mathrm{Con}_{\rho\to\sigma}$ is defined to mean $\forall_{J\subseteq I}(\bigcup_{j\in J} U_j \in \mathrm{Con}_\rho \to \{\, b_j \mid j \in J \,\} \in \mathrm{Con}_\sigma)$.

Building on this definition, we define $U \vdash_\rho a$ for $U \in \mathrm{Con}_\rho$ and $a \in C_\rho$.

(e) $\{\mathrm{C}\vec{a_1^*}, \dots, \mathrm{C}\vec{a_m^*}\} \vdash_\iota \mathrm{C}'\vec{a^*}$ is defined to mean $\mathrm{C} = \mathrm{C}'$, $m \geq 1$ and $U_i \vdash a_i^*$, with U_i as in (c) above (and $U \vdash *$ taken to be true).
(f) $W \vdash_{\rho\to\sigma} (U, b)$ is defined to mean $WU \vdash_\sigma b$, where application WU of $W = \{\, (U_i, b_i) \mid i \in I \,\} \in \mathrm{Con}_{\rho\to\sigma}$ to $U \in \mathrm{Con}_\rho$ is defined to be $\{\, b_i \mid U \vdash_\rho U_i \,\}$; recall that $U \vdash V$ abbreviates $\forall_{a\in V}(U \vdash a)$.

If we define the *height* of the syntactic expressions involved by

$$|\mathrm{C}a_1^* \dots a_n^*| := 1 + \max\{\, |a_i^*| \mid i = 1, \dots, n \,\}, \qquad |*| := 0,$$
$$|(U, b)| := \max\{1 + |U|, 1 + |b|\},$$
$$|\{\, a_i \mid i \in I \,\}| := \max\{\, 1 + |a_i| \mid i \in I \,\},$$
$$|U \vdash a| := \max\{1 + |U|, 1 + |a|\},$$

these are definitions by recursion on the height.

It is easy to see that $(C_\rho, \mathrm{Con}_\rho, \vdash_\rho)$ is an information system. Observe that all the notions involved are computable: $a \in C_\rho$, $U \in \mathrm{Con}_\rho$ and $U \vdash_\rho a$.

Definition (Partial continuous functionals). For every type ρ let $\boldsymbol{C}_\rho$ be the information system $(C_\rho, \mathrm{Con}_\rho, \vdash_\rho)$. The set $|\boldsymbol{C}_\rho|$ of ideals in $\boldsymbol{C}_\rho$ is the set of *partial continuous functionals* of type ρ. A partial continuous functional $x \in |\boldsymbol{C}_\rho|$ is *computable* if it is recursively enumerable when viewed as a set of tokens.

Notice that $\boldsymbol{C}_{\rho\to\sigma} = \boldsymbol{C}_\rho \to \boldsymbol{C}_\sigma$ as defined generally for information systems.

For example, the tokens for the algebra $\mathbf{N}$ are shown in Figure 1. For tokens a, b we have $\{a\} \vdash b$ if and only if there is a path from a (up) to b (down). As another (more typical) example, consider the algebra $\mathbf{D}$ of derivations with a nullary constructor 0 and a binary C. Then $\{\mathrm{C}0*, \mathrm{C}*0\}$ is consistent, and $\{\mathrm{C}0*, \mathrm{C}*0\} \vdash \mathrm{C}00$.

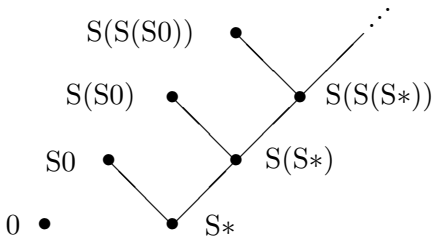


FIGURE 1. Tokens and entailment for **N**

2.1.5. Constructors as continuous functions. Let ι be an algebra. Every constructor C generates the following ideal in the function space:

$$r_{\mathrm{C}} := \{\, (\vec{U}, \mathrm{C}\vec{a^*}\,) \mid \vec{U} \vdash \vec{a^*}\,\}.$$

Here $(\vec{U}, a)$ abbreviates $(U_1, (U_2, \dots (U_n, a) \dots))$.

According to the general definition of a continuous function associated to an ideal in a function space the continuous map $|r_{\mathrm{C}}|$ satisfies

$$|r_{\mathrm{C}}|(\vec{x}\,) = \{\, \mathrm{C}\vec{a^*} \mid \exists_{\vec{U} \subseteq \vec{x}}(\vec{U} \vdash \vec{a^*})\,\}.$$

An immediate consequence is that the (continuous maps corresponding to) constructors are injective and their ranges are disjoint, which is what we wanted to achieve by associating non-flat rather than flat information systems with algebras.

LEMMA (Constructors are injective and have disjoint ranges). *Let ι be an algebra and* C *be a constructor of ι. Then*

$$|r_{\mathrm{C}}|(\vec{x}\,) \subseteq |r_{\mathrm{C}}|(\vec{y}\,) \leftrightarrow \vec{x} \subseteq \vec{y}.$$

If $\mathrm{C}_1, \mathrm{C}_2$ *are distinct constructors of ι, then* $|r_{\mathrm{C}_1}|(\vec{x}\,) \neq |r_{\mathrm{C}_2}|(\vec{y}\,)$*, since the two ideals are non-empty and disjoint.*

PROOF. Immediate from the definitions. □

REMARK. Notice that neither property holds for flat information systems, since for them, by monotonicity, constructors need to be *strict* (i.e., if one argument is the empty ideal, then the value is as well). But then we have

$$|r_{\mathrm{C}}|(\emptyset, y) = \emptyset = |r_{\mathrm{C}}|(x, \emptyset),$$
$$|r_{\mathrm{C}_1}|(\emptyset) = \emptyset = |r_{\mathrm{C}_2}|(\emptyset)$$

where in the first case we have one binary and, in the second, two unary constructors.

2.1.6. Total and cototal ideals in a finitary algebra. In the information system $\boldsymbol{C}_\iota$ associated with an algebra ι, the "total" and "cototal" ideals are of special interest. Here we give an explicit definition for finitary algebras. For general algebras totality can be defined inductively and cototality coinductively (cf. 3.1.4).

Recall that a token in ι is a constructor tree P possibly containing the special symbol $*$. Because of the possibility of parameter arguments we need to distinguish between "structure-" and "fully" total and cototal ideals. For the definition it is easiest to refer to a constructor tree $P(*)$ with a distinguished occurrence of $*$. This occurrence is called *non-parametric* if the path from it to the root does not pass through a parameter argument of a constructor. For a constructor tree $P(*)$, an arbitrary $P(\mathrm{C}\vec{a^*})$ is called *one-step extension* of $P(*)$, written $P(\mathrm{C}\vec{a^*}) \succ_1 P(*)$.

Definition. Let ι be an algebra, and $\boldsymbol{C}_\iota$ its associated information system. An ideal $x \in |\boldsymbol{C}_\iota|$ is *cototal* if every constructor tree $P(*) \in x$ has a $\succ_1$-predecessor $P(\mathrm{C}\vec{*}) \in x$; it is called *total* if it is cototal and the relation $\succ_1$ on x is well-founded. It is called *structure-cototal* (*structure-total*) if the same holds with $\succ_1$ defined w.r.t. $P(*)$ with a non-parametric distinguished occurrence of $*$.

If there are no parameter arguments, we shall simply speak of total and cototal ideals. For example, for the algebra $\mathbf{N}$ every total ideal is the deductive closure of a token $\mathrm{S}(\mathrm{S}\ldots(\mathrm{S0})\ldots)$, and the set of all tokens $\mathrm{S}(\mathrm{S}\ldots(\mathrm{S}*)\ldots)$ is a cototal ideal. For the algebra $\mathbf{L}(\mathbf{N})$ of lists of natural numbers the total ideals are the finite lists and the cototal ones the finite or infinite lists. For the algebra $\mathbf{D}$ of derivations the total ideals can be viewed as the finite derivations, and the cototal ones as the finite or infinite "locally correct" derivations of Mints (1978); arbitrary ideals can be viewed as "partial" or "incomplete" derivations, with "holes".

Remark. From a categorical perspective (as in Hagino (1987); Rutten (2000)) finite lists of natural numbers can be seen as making up the initial algebra of the functor $TX = 1 + (\mathbf{N} \times X)$, and infinite lists (or streams) of natural numbers as making up the terminal coalgebra of the functor $TX = \mathbf{N} \times X$. In the present setting both finite and infinite lists of natural numbers appear as cototal ideals in the algebra $\mathbf{L}(\mathbf{N})$, with the finite ones the total ideals. However, to properly deal with computability we need to accommodate partiality, and hence there are more ideals in the algebra $\mathbf{L}(\mathbf{N})$.

2.2. Denotational and operational semantics

For every type ρ, we have defined what a partial continuous functional of type ρ is: an ideal consisting of tokens at this type. These tokens or rather the formal neighborhoods formed from them are syntactic in nature; they are reminiscent to Kreisel's "formal neighborhoods" (Kreisel, 1959; Martin-Löf, 1983; Coquand and Spiwack, 2006). However – in contrast to Martin-Löf (1983) – we do not have to deal separately with a notion of consistency for formal neighborhoods: this concept is built into information systems.

Let us now turn our attention to a formal (functional programming) language, in the style of Plotkin's PCF (1977), and see how we can provide a denotational semantics (that is, a "meaning") for the terms of this language. A closed term M of type ρ will denote a partial continuous functional of this type, that is, a consistent and deductively closed set of tokens of type ρ. We will define this set inductively.

It will turn out that these sets are recursively enumerable. In this sense every closed term M of type ρ denotes a computable partial continuous functional of type ρ. However, it is not a good idea to *define* a computable functional in this way, by providing a recursive enumeration of its tokens. We rather want to be able to use recursion equations for such definitions. Therefore we extend the term language by constants D defined by certain "computation rules", as in (Berger et al., 2003; Berger, 2005). Our semantics will cover these as well. The resulting term system can be seen as a common extension of Gödel's T (1958) and Plotkin's PCF; we call it T^+.

2.2.1. Structural recursion operators and Gödel's T. We begin with a discussion of particularly important examples of such constants D, the (structural) higher type *recursion operators* $\mathcal{R}^\tau_\iota$ introduced by Hilbert (1925) and Gödel (1958). They are used to construct maps from the algebra ι to τ, by recursion on the structure of ι. For instance, $\mathcal{R}^\tau_{\mathbf{N}}$ has type $\mathbf{N} \to \tau \to (\mathbf{N} \to \tau \to \tau) \to \tau$. The first argument is the recursion argument, the second one gives the base value, and the third one gives the step function, mapping the recursion argument and the previous value to the next value. For example, $\mathcal{R}^{\mathbf{N}}_{\mathbf{N}} nm\lambda_{n,p}(\mathrm{S}p)$ defines addition $m + n$ by recursion on n. For $\lambda_{n,p}(\mathrm{S}p)$ we often write $\lambda_{\_,p}(\mathrm{S}p)$ since the bound variable n is not used.

Generally, we define the type of the recursion operator $\mathcal{R}^\tau_\iota$ for the algebra $\iota = \mu_\xi((\rho_{i\nu}(\xi))_{\nu<n_i} \to \xi)_{i<k}$ and result type τ to be

$$\iota \to ((\rho_{i\nu}(\iota \times \tau))_{\nu<n_i} \to \tau)_{i<k} \to \tau.$$

Here ι is the type of the recursion argument, and each $(\rho_{i\nu}(\iota\times\tau))_{\nu<n_i}\to\tau$ is called a *step type*. Usage of $\iota\times\tau$ rather than τ in the step types can be seen as a "strengthening", since then one has more data available to construct the value of type τ. Moreover, for unnested recursive argument types $\vec{\sigma}\to\tau$ we avoid the product type in $\vec{\sigma}\to\iota\times\tau$ and take the two argument types $\vec{\sigma}\to\iota$ and $\vec{\sigma}\to\tau$ instead ("duplication").

For some algebras we spell out the type of their recursion operators:

$$\mathcal{R}^\tau_{\mathbf{B}}\colon \mathbf{B}\to\tau\to\tau\to\tau,$$
$$\mathcal{R}^\tau_{\mathbf{N}}\colon \mathbf{N}\to\tau\to(\mathbf{N}\to\tau\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\mathbf{P}}\colon \mathbf{P}\to\tau\to(\mathbf{P}\to\tau\to\tau)\to(\mathbf{P}\to\tau\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\mathbf{D}}\colon \mathbf{D}\to\tau\to(\mathbf{D}\to\tau\to\mathbf{D}\to\tau\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\mathbf{O}}\colon \mathbf{O}\to\tau\to(\mathbf{O}\to\tau\to\tau)\to((\mathbf{N}\to\mathbf{O})\to(\mathbf{N}\to\tau)\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\mathbf{L}(\rho)}\colon \mathbf{L}(\rho)\to\tau\to(\rho\to\mathbf{L}(\rho)\to\tau\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\rho+\sigma}\colon \rho+\sigma\to(\rho\to\tau)\to(\sigma\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\rho\times\sigma}\colon \rho\times\sigma\to(\rho\to\sigma\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\mathbf{T}}\colon \mathbf{T}\to(\mathbf{L}(\mathbf{T}\times\tau)\to\tau)\to\tau.$$

There is an important variant of recursion, where no recursive calls occur. This variant is called the *cases operator*; it distinguishes cases according to the outer constructor form. For the algebra $\iota=\mu_\xi((\rho_{i\nu}(\xi))_{\nu<n_i}\to\xi)_{i<k}$ and result type τ the type of the cases operator $\mathcal{C}^\tau_\iota$ is

$$\iota\to((\rho_{i\nu}(\iota))_{\nu<n_i}\to\tau)_{i<k}\to\tau.$$

The simplest example (for type $\mathbf{B}$) is *if-then-else*. Another example is

$$\mathcal{C}^\tau_{\mathbf{N}}\colon \mathbf{N}\to\tau\to(\mathbf{N}\to\tau)\to\tau.$$

It can be used to define the *predecessor* function on $\mathbf{N}$, i.e., $\mathrm{P}0:=0$ and $\mathrm{P}(\mathrm{S}n):=n$, by the term

$$\mathrm{P}m:=\mathcal{C}^{\mathbf{N}}_{\mathbf{N}}m0(\lambda_n n).$$

Remark. When computing the value of a cases term, we do not want to (eagerly) evaluate all arguments, but rather compute the test argument first and depending on the result (lazily) evaluate at most one of the other arguments. This phenomenon is well known in functional languages; for instance, in Scheme the `if`-construct is called a *special form* (as opposed to an operator). Therefore instead of taking the cases operator applied to a full list of arguments, one rather uses a `case`-construct to build this term; it differs from the former only in that it employs lazy evaluation. Hence the

predecessor function is written in the form $[\mathbf{case}\ m\ \mathbf{of}\ 0 \mid \lambda_n n]$. If there are exactly two cases, we also write $\lambda_m[\mathbf{if}\ m\ \mathbf{then}\ 0\ \mathbf{else}\ \lambda_n n]$ instead.

We shall also need *map operators*. Let $\rho(\vec{\alpha}\,)$ be a type and $\vec{\alpha}$ strictly positive type parameters. We define

$$\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho(\vec{\alpha}\,)} \colon \rho(\vec{\sigma}\,) \to (\vec{\sigma} \to \vec{\tau}\,) \to \rho(\vec{\tau}\,)$$

(where $(\vec{\sigma} \to \vec{\tau}) \to \rho(\vec{\tau}\,)$ means $(\sigma_1 \to \tau_1) \to \ldots \to (\sigma_n \to \tau_n) \to \rho(\vec{\tau}\,)$). If none of $\vec{\alpha}$ appears free in $\rho(\vec{\alpha}\,)$ let

$$\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho(\vec{\alpha}\,)} x \vec{f} := x.$$

Otherwise we use an outer recursion on $\rho(\vec{\alpha}\,)$ and if $\rho(\vec{\alpha}\,)$ is $\iota(\vec{\alpha}\,)$ an inner one on x. In case $\rho(\vec{\alpha}\,)$ is $\iota(\vec{\alpha}\,)$ we abbreviate $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\iota(\vec{\alpha}\,)}$ by $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\iota}$ or $\mathcal{M}^{\vec{\tau}}_{\iota(\vec{\sigma}\,)}$.

The immediate cases for the outer recursion are

$$\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\alpha_i} x \vec{f} := f_i x, \qquad \mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}(\sigma\to\rho)} h \vec{f} x := \mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho}(hx)\vec{f}.$$

It remains to consider $\iota(\vec{\pi}(\vec{\alpha}\,))$. In case $\vec{\pi}(\vec{\alpha}\,)$ is not $\vec{\alpha}$ let

$$\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\iota(\vec{\pi}(\vec{\alpha}\,))} x \vec{f} := \mathcal{M}^{\vec{\pi}(\vec{\sigma}\,)\to\vec{\pi}(\vec{\tau}\,)}_{\iota} x (\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\pi_i(\vec{\alpha}\,)} \cdot \vec{f}\,)_{i<|\vec{\pi}\,|}$$

with $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\pi_i(\vec{\alpha}\,)} \cdot \vec{f} := \lambda_x \mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\pi_i(\vec{\alpha}\,)} x \vec{f}$. In case $\vec{\pi}(\vec{\alpha}\,)$ is $\vec{\alpha}$ we use recursion on x and define for a constructor $\mathrm{C}_i \colon (\rho_\nu(\vec{\sigma}, \iota(\vec{\sigma}\,)))_{\nu<n} \to \iota(\vec{\sigma}\,)$

$$\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\iota}(\mathrm{C}_i \vec{x}\,)\vec{f}$$

to be the result of applying C'_i of type $(\rho_\nu(\vec{\tau}, \iota(\vec{\tau}\,)))_{\nu<n} \to \iota(\vec{\tau}\,)$ (the same constructor as C_i with only the type changed) to, for each $\nu < n$,

$$\mathcal{M}^{\vec{\sigma},\iota(\vec{\sigma}\,)\to\vec{\tau},\iota(\vec{\tau}\,)}_{\lambda_{\vec{\alpha},\beta}\rho_\nu(\vec{\alpha},\beta)} x_\nu \vec{f}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\iota} \cdot \vec{f}\,).$$

Note that the final function argument provides the recursive call w.r.t. the recursion on x.

Example.

$$\mathcal{M}^{\tau}_{\mathbf{L}(\sigma)} \mathrm{Nil} f^{\sigma\to\tau} := \mathrm{Nil},$$
$$\mathcal{M}^{\tau}_{\mathbf{L}(\sigma)}(x^\sigma :: l^{\mathbf{L}(\sigma)}) f^{\sigma\to\tau} := (fx) :: (\mathcal{M}\, l\, f).$$

Definition. *Terms of Gödel's* T for nested algebras are inductively defined from typed variables x^ρ and constants for constructors C^ι_i, recursion operators $\mathcal{R}^\tau_\iota$ and map operators $\mathcal{M}^{\vec{\rho}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\pi}$ by abstraction $\lambda_{x^\rho} M^\sigma$ and application $M^{\rho\to\sigma} N^\rho$.

2.2.2. Conversion. We define a *conversion relation* $\mapsto_\rho$ between terms of type ρ by

$$(\lambda_x M(x))N \mapsto M(N), \tag{2.1}$$

$$\lambda_x(Mx) \mapsto M \quad \text{if } x \notin \mathrm{FV}(M) \ (M \text{ not an abstraction}), \tag{2.2}$$

$$\mathcal{R}^\tau_\iota(\mathrm{C}^\iota_i\vec{N})\vec{M} \mapsto M_i(\mathcal{M}^{\iota\to\iota\times\tau}_{\lambda_\alpha\rho_\nu(\alpha)}N_\nu\lambda_x\langle x^\iota,\mathcal{R}^\tau_\iota x\vec{M}\rangle)_{\nu<n} \tag{2.3}$$

where $(\rho_\nu(\iota))_{\nu<n}\to\iota$ is the type of the i-th constructor C_i.

In the special case $\rho_\nu(\alpha)=\alpha$ we can avoid the product type and instead of the pair

$$\mathcal{M}^{\iota\to\iota\times\tau}_{\lambda_\alpha\alpha}N_\nu\lambda_x\langle x^\iota,\mathcal{R}^\tau_\iota x\vec{M}\rangle \quad \text{i.e.,} \quad \langle N^\iota_\nu,\mathcal{R}^\tau_\iota N_\nu\vec{M}\rangle$$

take its two components N^ι_ν and $\mathcal{R}^\tau_\iota N_\nu\vec{M}$ as separate arguments of M_i.

The rule (2.1) is called β-conversion, and (2.2) η-conversion; their left hand sides are called *β-redexes* or *η-redexes*, respectively. The left hand side of (2.3) is called *$\mathcal{R}$-redex*; it is a special case of a redex associated with a constant D defined by "computation rules" (cf. 2.2.3), and hence also called a *D-redex*.

2.2.3. A common extension T^+ of Gödel's T and Plotkin's PCF. *Terms* of T^+ are built from (typed) variables and (typed) constants (constructors C or defined constants D, see below) by (type-correct) application and abstraction:

$$M,N ::= x^\rho \mid \mathrm{C}^\rho \mid D^\rho \mid (\lambda_{x^\rho}M^\sigma)^{\rho\to\sigma} \mid (M^{\rho\to\sigma}N^\rho)^\sigma.$$

Definition (Computation rule). Every defined constant D comes with a system of *computation rules*, consisting of finitely many equations

$$D\vec{P}_i(\vec{y}_i) = M_i \qquad (i=1,\dots,n) \tag{2.4}$$

with free variables of $\vec{P}_i(\vec{y}_i)$ and M_i among $\vec{y}_i$, where the arguments on the left hand side must be "constructor patterns", i.e., lists of applicative terms built from constructors and distinct variables. To ensure consistency of the defining equations, we require that for $i\neq j$ $\vec{P}_i$ and $\vec{P}_j$ have disjoint free variables, and either $\vec{P}_i$ and $\vec{P}_j$ are non-unifiable (i.e., there is no substitution which identifies them), or else for the most general unifier ϑ of $\vec{P}_i$ and $\vec{P}_j$ we have $M_i\vartheta = M_j\vartheta$. Notice that the substitution ϑ assigns to the variables $\vec{y}_i$ in M_i constructor patterns $\vec{R}_k(\vec{z})$ $(k=i,j)$. A further requirement on a system of computation rules $D\vec{P}_i(\vec{y}_i)=M_i$ is that the lengths of all $\vec{P}_i(\vec{y}_i)$ are the same; this number is called the *arity* of D, denoted by $\mathrm{ar}(D)$. A substitution instance of a left hand side of (2.4) is called a *D-redex*.

More formally, constructor patterns are defined inductively by (we write $\vec{P}(\vec{x}\,)$ to indicate all variables in $\vec{P}$):

(a) x is a constructor pattern.
(b) The empty list $\langle\rangle$ is a constructor pattern.
(c) If $\vec{P}(\vec{x}\,)$ and $Q(\vec{y}\,)$ are constructor patterns whose variables $\vec{x}$ and $\vec{y}$ are disjoint, then $(\vec{P}, Q)(\vec{x}, \vec{y}\,)$ is a constructor pattern.
(d) If C is a constructor and $\vec{P}$ a constructor pattern, then so is $\mathrm{C}\vec{P}$, provided it is of ground type.

REMARK. The requirement of disjoint variables in constructor patterns $\vec{P}_i$ and $\vec{P}_j$ used in computation rules of a defined constant D is needed to ensure that applying the most general unifier produces constructor patterns again. However, for readability we take this as an implicit convention, and write computation rules with possibly non-disjoint variables.

Examples of constants D defined by computation rules are abundant. In particular, the map and (structural) recursion operators can be viewed as defined by computation rules, which in this case are called *conversion* rules; cf. 2.2.2.

The boolean connectives andb, impb and orb are defined by

$$\begin{array}{lll} \mathtt{tt}\ \mathrm{andb}\ y = y, & \mathtt{ff}\ \mathrm{impb}\ y = \mathtt{tt}, & \mathtt{tt}\ \mathrm{orb}\ y = \mathtt{tt}, \\ x\ \mathrm{andb}\ \mathtt{tt} = x, & \mathtt{tt}\ \mathrm{impb}\ y = y, & x\ \mathrm{orb}\ \mathtt{tt} = \mathtt{tt}, \\ \mathtt{ff}\ \mathrm{andb}\ y = \mathtt{ff}, & x\ \mathrm{impb}\ \mathtt{tt} = \mathtt{tt}, & \mathtt{ff}\ \mathrm{orb}\ y = y, \\ x\ \mathrm{andb}\ \mathtt{ff} = \mathtt{ff}, & & x\ \mathrm{orb}\ \mathtt{ff} = x. \end{array}$$

Notice that when two such rules overlap, their right hand sides are equal under any unifier of the left hand sides.

Decidable *equality* $=_\iota \colon \iota \to \iota \to \mathbf{B}$ for a finitary algebra ι can be defined easily by computation rules. For example,

$$\begin{array}{ll} (0 =_{\mathbf{N}} 0) = \mathtt{tt}, & (\mathrm{S}m =_{\mathbf{N}} 0) = \mathtt{ff}, \\ (0 =_{\mathbf{N}} \mathrm{S}n) = \mathtt{ff}, & (\mathrm{S}m =_{\mathbf{N}} \mathrm{S}n) = (m =_{\mathbf{N}} n). \end{array}$$

For the algebra $\mathbf{D}$ of binary trees with constructors L (leaf) and C (construct a new tree from two given ones) we have

$$\begin{array}{ll} (L =_{\mathbf{D}} L) = \mathtt{tt}, & (\mathrm{C}m =_{\mathbf{D}} L) = \mathtt{ff}, \\ (L =_{\mathbf{D}} \mathrm{C}n) = \mathtt{ff}, & (\mathrm{C}a_1a_2 =_{\mathbf{D}} \mathrm{C}b_1b_2) = (a_1 =_{\mathbf{D}} b_1\ \mathrm{andb}\ a_2 =_{\mathbf{D}} b_2). \end{array}$$

2.2.4. Ideals as denotation of terms. How can we use computation rules to define an ideal z in a function space? The general idea is to inductively define the set of tokens (U, b) that make up z. It is convenient to define the value $[\![\lambda_{\vec{x}} M]\!]$, where M is a term with free variables among $\vec{x}$. Since this value is a token set, we can define inductively the relation $(\vec{U}, b) \in [\![\lambda_{\vec{x}} M]\!]$.

For a constructor pattern $\vec{P}(\vec{x}\,)$ and a list $\vec{V}$ of the same length and types as $\vec{x}$ we define a list $\vec{P}(\vec{V})$ of formal neighborhoods of the same length and types as $\vec{P}(\vec{x}\,)$, by induction on $\vec{P}(\vec{x}\,)$. $x(V)$ is the singleton list V, and for $\langle\rangle$ we take the empty list. $(\vec{P}, Q)(\vec{V}, \vec{W})$ is covered by the induction hypothesis. Finally

$$(\mathrm{C}\vec{P})(\vec{V}) := \{\, \mathrm{C}\vec{b^*} \mid b_i^* \in P_i(\vec{V_i}) \text{ if } P_i(\vec{V_i}) \neq \emptyset, \text{ and } b_i^* = * \text{ otherwise} \,\}.$$

We use the following notation. $(\vec{U}, b)$ means $(U_1, \dots (U_n, b) \dots)$, and $(\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} M]\!]$ means $(\vec{U}, b) \in [\![\lambda_{\vec{x}} M]\!]$ for all (finitely many) $b \in V$.

Definition (Inductive, of $a \in [\![\lambda_{\vec{x}} M]\!]$). *Case* $\lambda_{\vec{x},y,\vec{z}} M$ with $\vec{x}$ free in M, but not y.

$$\frac{(\vec{U}, \vec{W}, a) \in [\![\lambda_{\vec{x},\vec{z}} M]\!]}{(\vec{U}, V, \vec{W}, a) \in [\![\lambda_{\vec{x},y,\vec{z}} M]\!]}(K).$$

Case $\lambda_{\vec{x}} M$ with $\vec{x}$ the free variables in M.

$$\frac{U \vdash a}{(U, a) \in [\![\lambda_x x]\!]}(V), \quad \frac{(\vec{U}, V, a) \in [\![\lambda_{\vec{x}} M]\!] \quad (\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} N]\!]}{(\vec{U}, a) \in [\![\lambda_{\vec{x}} (MN)]\!]}(A).$$

For every constructor C and defined constant D:

$$\frac{\vec{U} \vdash \vec{a^*}}{(\vec{U}, \mathrm{C}\vec{a^*}) \in [\![\mathrm{C}]\!]}(\mathrm{C}), \quad \frac{(\vec{V}, a) \in [\![\lambda_{\vec{x}} M]\!] \quad \vec{U} \vdash \vec{P}(\vec{V})}{(\vec{U}, a) \in [\![D]\!]}(D),$$

with one rule (D) for every defining equation $D\vec{P}(\vec{x}\,) = M$.

This "denotational semantics" has good properties; however, we do not carry out the proofs here (cf. Schwichtenberg and Wainer (2012)). First of all, one can prove that $[\![\lambda_{\vec{x}} M]\!]$ is an ideal . Moreover, our definition above of the denotation of a term is reasonable in the sense that it is not changed by an application of the standard (β- and η-) conversions or a computation rule. For the β-conversion part of this proof it is helpful to first introduce a more standard notation, which involves variable environments.

Definition. $[\![M]\!]^{\vec{U}}_{\vec{x}} := \{\, b \mid (\vec{U}, b) \in [\![\lambda_{\vec{x}} M]\!] \,\}$, $[\![M]\!]^{\vec{u},\vec{V}}_{\vec{x},\vec{y}} := \bigcup_{\vec{U} \subseteq \vec{u}} [\![M]\!]^{\vec{U},\vec{V}}_{\vec{x},\vec{y}}$.

We have a useful monotonicity property, which follows from the deductive closure of $[\![\lambda_{\vec{x}} M]\!]$.

LEMMA. (a) *If* $\vec{V} \vdash \vec{U}$, $b \vdash c$ *and* $b \in [\![M]\!]^{\vec{U}}_{\vec{x}}$, *then* $c \in [\![M]\!]^{\vec{V}}_{\vec{x}}$.
(b) *If* $\vec{v} \supseteq \vec{u}$, $b \vdash c$ *and* $b \in [\![M]\!]^{\vec{u}}_{\vec{x}}$, *then* $c \in [\![M]\!]^{\vec{v}}_{\vec{x}}$.

LEMMA. (a) $[\![x_i]\!]^{\vec{U}}_{\vec{x}} = \overline{U}_i$ *and* $[\![x_i]\!]^{\vec{u}}_{\vec{x}} = u_i$.
(b) $[\![\lambda_y M]\!]^{\vec{U}}_{\vec{x}} = \{\, (V, b) \mid b \in [\![M]\!]^{\vec{U},V}_{\vec{x},y} \,\}$ *and* $[\![\lambda_y M]\!]^{\vec{u}}_{\vec{x}} = \{\, (V, b) \mid b \in [\![M]\!]^{\vec{u},V}_{\vec{x},y} \,\}$.
(c) $[\![MN]\!]^{\vec{U}}_{\vec{x}} = [\![M]\!]^{\vec{U}}_{\vec{x}} [\![N]\!]^{\vec{U}}_{\vec{x}}$ *and* $[\![MN]\!]^{\vec{u}}_{\vec{x}} = [\![M]\!]^{\vec{u}}_{\vec{x}} [\![N]\!]^{\vec{u}}_{\vec{x}}$.

COROLLARY. $[\![\lambda_y M]\!]^{\vec{u}}_{\vec{x}} v = [\![M]\!]^{\vec{u},v}_{\vec{x},y}$.

LEMMA (Substitution). $[\![M(z)]\!]^{\vec{u},[\![N]\!]^{\vec{u}}_{\vec{x}}}_{\vec{x},z} = [\![M(N)]\!]^{\vec{u}}_{\vec{x}}$.

LEMMA (Preservation of values, β). $[\![(\lambda_y M(y))N]\!]^{\vec{u}}_{\vec{x}} = [\![M(N)]\!]^{\vec{u}}_{\vec{x}}$.

LEMMA (Preservation of values, η). $[\![\lambda_y (My)]\!]^{\vec{u}}_{\vec{x}} = [\![M]\!]^{\vec{u}}_{\vec{x}}$ *if* $y \notin \mathrm{FV}(M)$.

Then it follows that values are preserved under computation rules:

LEMMA. *For every computation rule* $D\vec{P}(\vec{y}) = M$ *of a defined constant* D, $[\![\lambda_{\vec{y}} (D\vec{P}(\vec{y}))]\!]^{\vec{u}}_{\vec{x}} = [\![\lambda_{\vec{y}} M]\!]^{\vec{u}}_{\vec{x}}$.