CHAPTER 2

Computability

At this point we leave the general setting of logic and aim to get closer to mathematics. We introduce free algebras (for example, the natural numbers) as basic data structures and consider function spaces based on them. The functional objects are viewed as limits of their finite approximations. We call a functional computable if it is the limit of a recursively enumerable set of finite approximations. To work with such objects in a formal theory, we need to have a language to denote them. Again lambda calculus is the appropriate tool, this time extended by constants for particular functionals defined by equations.

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 $\mathcal{H}(\Phi')$ 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 of Φ, 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^0_1-definable).

The general theory of computability concerns partial functions and partial operations on them. However, we might be interested in particular objects only, so once the theory of general (partial) objects is developed, we can look for ways to restrict attention to the particular ones. Examples for interesting sets of objects are (i) the total functions on natural numbers, (ii) so-called cototal objects like "streams" of signed digits $\{-1, 0, 1\}$ (useful to represent real numbers), and (iii) extensional functionals mapping functions on natural numbers to natural numbers.

2.1. Abstract computability via information systems

We need to define appropriate domains for our to-be-defined computable functionals, viewed as limits of their finite approximations. Information systems are a convenient setting to introduce and study the latter.

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 an at most countable non-empty set (the *tokens*), Con is a set of finite subsets of A (the *consistent* sets) and $\vdash$ is a subset of $\mathrm{Con} \times A$ (the *entailment* relation), which satisfy

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

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

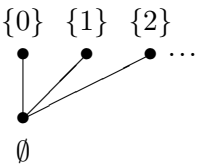
$$
\begin{aligned}
&U \vdash V \quad \text{for} \quad U \in \mathrm{Con} \wedge \forall_{a \in V}(U \vdash a),\\
&a \uparrow b \quad \text{for} \quad \{a, b\} \in \mathrm{Con} \qquad (a, b \text{ are } \textit{consistent}),\\
&U \uparrow V \quad \text{for} \quad \forall_{a \in U, b \in V}(a \uparrow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

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

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

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 easy to verify that this defines an information system whose ideals are (the graphs of) all partial functions from A to B.

Remark. One can show that for an arbitrary information system $\boldsymbol{A} = (A, \mathrm{Con}, \vdash)$ the structure $(|\boldsymbol{A}|, \subseteq, \overline{\emptyset})$ is a "domain" (also called Scott-Ershov domain, or "bounded complete algebraic cpo"), whose set of "compact elements" can be represented as $|\boldsymbol{A}|_c = \{\, \overline{U} \mid U \in \mathrm{Con} \,\}$. The converse holds as well: every countable domain can be represented as an information system. We will not need this relation to standard (non-constructive) domain theory, and hence not even define these notions here.

2.1.2. Function spaces. We 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

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

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 $WU \vdash_B b$.

Remark. 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 2.1.1. *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 $W'U \vdash_B b$ by definition. □

We shall now give an alternative characterization of the ideals in $\boldsymbol{A} \to \boldsymbol{B}$, as "approximable maps". 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 $(U, b) \in r$ 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 $(U, b_1), \ldots, (U, b_n) \in r$, then $\{b_1, \ldots, b_n\} \in \mathrm{Con}_B$;
(b) if $(U, b_1), \ldots, (U, b_n) \in r$ and $\{b_1, \ldots, b_n\} \vdash_B b$, then $(U, b) \in r$;
(c) if $(U', b) \in r$ and $U \vdash_A U'$, then $(U, b) \in r$.

THEOREM 2.1.2. *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 $(U, b_1), \ldots, (U, b_n) \in r$. We must show that $\{b_1, \ldots, b_n\} \in \mathrm{Con}_B$. But this clearly follows from the consistency of r.

(b) Let $(U, b_1), \ldots, (U, b_n) \in r$ and $\{b_1, \ldots, b_n\} \vdash_B b$. We must show that $(U, b) \in r$. 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 $(U, b) \in r$ since r is deductively closed.

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

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

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

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

Consistency of r. Suppose $(U_1, b_1), \ldots, (U_n, b_n) \in r$ and $U = \bigcup\{ U_i \mid i \in I \} \in \mathrm{Con}_A$ for some $I \subseteq \{1, \ldots, n\}$. We must show that $\{ b_i \mid i \in I \} \in \mathrm{Con}_B$. Now from $(U_i, b_i) \in r$ and $U \vdash_A U_i$ we obtain $(U, b_i) \in r$ 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. Suppose $(U_1, b_1), \ldots, (U_n, b_n) \in r$ and

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

We must show $(U, b) \in r$. 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 $(U_i, b_i) \in r$ we know $(U, b_i) \in r$ by axiom (c) for all i with $U \vdash_A U_i$. Hence $(U, b) \in r$ by axiom (b). □

2.1.3. Continuous functions. We can also characterize approximable maps in a different way, which is closer to usual characterizations of continuity[1]:

LEMMA 2.1.3. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be information systems and $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ monotone (i.e., $x \subseteq y$ implies $f(x) \subseteq f(y)$). Then the following are equivalent.*

[1]In fact, approximable maps are exactly the continuous functions w.r.t. the so-called Scott topology. However, we will not enter this subject here.

(a) *f satisfies the "principle of finite support" PFS: If $b \in f(x)$, then $b \in f(\overline{U})$ for some $U \subseteq x$.*
(b) *f 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 $x, y \subseteq z$)*

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

Note that in (b) the set $\{\, f(x) \mid x \in D \,\}$ is directed by monotonicity of f; hence its union is indeed an ideal in $|\boldsymbol{B}|$. 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 $U \subseteq x$.

Proof. 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 $U \subseteq x$. □

We call a function $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ continuous if it satifies the conditions in Lemma 2.1.3. 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 $f(\overline{U}) \subseteq f(x)$.

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 2.1.4. *Let $\boldsymbol{A} = (A, \mathrm{Con}_A, \vdash_A)$, $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$ 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 $r \subseteq \mathrm{Con}_A \times B$ we can associate a continuous function $|r|\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ by*

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

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* $\hat{f} \subseteq \mathrm{Con}_A \times B$ *by*

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

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 Theorem 2.1.2 we know that 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 $(U_i, b_i) \in r$. Hence $U := U_1 \cup \dots \cup U_n \subseteq z$ and $(U, b_i) \in r$ 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 $(U, b_i) \in r$. Now from axiom (b) we can conclude $(U, b) \in r$ and hence $b \in |r|(z)$.

Continuity of $|r|$ follows immediately from part (a) of Lemma 2.1.3 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

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

Finally, for any approximable map $r \subseteq \mathrm{Con}_A \times B$ we have

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

hence $r = \widehat{|r|}$. □

Consequently we can (and will) view approximable maps $r \subseteq \mathrm{Con}_A \times B$ as continuous functions from $|\boldsymbol{A}|$ to $|\boldsymbol{B}|$.

Equality of two subsets $r, s \subseteq \mathrm{Con}_A \times B$ means that they consist of the same tokens (U, b). We can characterize equality $r = s$ by extensional equality of the associated functions $|r|, |s|$. It even suffices that $|r|$ and $|s|$ coincide on all compact elements $\overline{U}$ for $U \in \mathrm{Con}_A$.

Lemma 2.1.5 (Extensionality). *Assume that* $\boldsymbol{A} = (A, \mathrm{Con}_A, \vdash_A)$ *and* $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$ *are information systems and* $r, s \subseteq \mathrm{Con}_A \times B$ *approximable maps. Then the following are equivalent.*

(a) $r = s$,
(b) $|r|(z) = |s|(z)$ *for all* $z \in |\boldsymbol{A}|$,
(c) $|r|(\overline{U}) = |s|(\overline{U})$ *for all* $U \in \mathrm{Con}_A$.

PROOF. It suffices to prove (c) $\to$ (a). As above this follows from

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

□

Moreover, one can easily check that

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

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

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

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

2.1.4. 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, $\tau \to \sigma$. As domains for the base types we choose non-flat free algebras, given by their constructors. The 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. Let α, β, ξ denote type variables.

DEFINITION (Constructor types and algebra forms). *Constructor types* κ have the form

$$\vec{\alpha} \to (\xi)_{i<n} \to \xi$$

with all type variables α_i distinct from each other and from ξ. Iterated arrows are understood as associated to the right. An argument type of a constructor type is called a *parameter* argument type if it is different from ξ, and a *recursive* argument type otherwise. A constructor type κ is *nullary* if it has no recursive argument types. We call

$$\iota := \mu_\xi \vec{\kappa}$$

with $\vec{\kappa}$ not empty an *algebra form*. An algebra form is *non-recursive* (or *explicit*) if it does not have recursive argument types.

EXAMPLES. We list some algebra forms without parameters, with standard names for the constructors added to each constructor type.

$$\begin{aligned}
&\mathbb{U} := \mu_\xi(\mathrm{Dummy}\colon \xi) && \text{(unit)},\\
&\mathbb{B} := \mu_\xi(\mathtt{tt}\colon \xi, \mathtt{ff}\colon \xi) && \text{(booleans)},\\
&\mathbb{N} := \mu_\xi(0\colon \xi, S\colon \xi \to \xi) && \text{(natural numbers, unary)},\\
&\mathbb{P} := \mu_\xi(1\colon \xi, S_0\colon \xi \to \xi, S_1\colon \xi \to \xi) && \text{(positive numbers, binary)},\\
&\mathbb{Y} := \mu_\xi(-\colon \xi, \mathrm{Branch}\colon \xi \to \xi \to \xi) && \text{(binary trees, or derivations)}.
\end{aligned}$$

Algebra forms with type parameters are

$$\begin{aligned}
&\mathbb{I}(\alpha) && := \mu_\xi(\mathrm{Id}\colon \alpha \to \xi) && \text{(identity)},\\
&\mathbb{L}(\alpha) && := \mu_\xi(\mathrm{Nil}\colon \xi, \mathrm{Cons}\colon \alpha \to \xi \to \xi) && \text{(lists)},\\
&\mathbb{S}(\alpha) && := \mu_\xi(\mathrm{SCons}\colon \alpha \to \xi \to \xi) && \text{(streams)},\\
&\alpha \times \beta && := \mu_\xi(\mathrm{Pair}\colon \alpha \to \beta \to \xi) && \text{(product)},\\
&\alpha + \beta && := \mu_\xi(\mathrm{InL}\colon \alpha \to \xi, \mathrm{InR}\colon \beta \to \xi) && \text{(sum)},\\
&\mathrm{uysum}(\alpha) && := \mu_\xi(\mathrm{DummyL}\colon \xi, \mathrm{Inr}\colon \alpha \to \xi) && (\text{for } \mathbb{U} + \alpha),\\
&\mathrm{ysumu}(\alpha) && := \mu_\xi(\mathrm{Inl}\colon \alpha \to \xi, \mathrm{DummyR}\colon \xi) && (\text{for } \alpha + \mathbb{U}).
\end{aligned}$$

The default name for the i-th constructor of an algebra form is C_i.

DEFINITION (Type).

$$\rho, \sigma, \tau ::= \alpha \mid \iota(\vec{\rho}) \mid \tau \to \sigma,$$

where ι is an algebra form with $\vec{\alpha}$ its parameter type variables, and $\iota(\vec{\rho})$ the result of substituting the (already generated) types $\vec{\rho}$ for $\vec{\alpha}$. Types of the form $\iota(\vec{\rho})$ are called *algebras*. An algebra is *closed* if it has no type variables. The *level* of a type is defined by

$$\begin{aligned}
\mathrm{lev}(\alpha) &:= 0,\\
\mathrm{lev}(\iota(\vec{\rho})) &:= \max(\mathrm{lev}(\vec{\rho})),\\
\mathrm{lev}(\tau \to \sigma) &:= \max(\mathrm{lev}(\sigma), 1 + \mathrm{lev}(\tau)).
\end{aligned}$$

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

EXAMPLES. 1. $\mathbb{L}(\alpha)$, $\mathbb{L}(\mathbb{L}(\alpha))$, $\alpha \times \beta$ are algebras.
2. $\mathbb{L}(\mathbb{L}(\mathbb{N}))$, $\mathbb{N} + \mathbb{B}$, $\mathbb{Z} := \mathbb{P} + \mathbb{U} + \mathbb{P}$, $\mathbb{Q} := \mathbb{Z} \times \mathbb{P}$ are closed base types.
3. $\mathbb{R} := (\mathbb{N} \to \mathbb{Q}) \times (\mathbb{P} \to \mathbb{N})$ is a closed algebra of level 1.

There can be many equivalent ways to define a particular type. For instance, we could take $\mathbb{U} + \mathbb{U}$ to be the type of booleans, $\mathbb{L}(\mathbb{U})$ to be the type of natural numbers, and $\mathbb{L}(\mathbb{B})$ to be the type of positive binary numbers.

2.1.5. Partial continuous functionals. For every closed type τ we define an information system $\boldsymbol{C}_\tau = (C_\tau, \mathrm{Con}_\tau, \vdash_\tau)$. The definition is by induction on τ, and in case of an algebra $\iota(\vec{\rho})$ by a side inductive definition.

DEFINITION (Information system of type τ). *Case* $\iota(\rho)$. For simplicity assume that there is only one parameter type ρ.

(a) *Tokens* $a \in C_{\iota(\rho)}$ are the type correct constructor expressions $\mathrm{C}Va_1^* \dots a_n^*$ where a_i^* is an *extended token*, i.e., a token or the special symbol $*$ which carries no information, and V is a consistent set of tokens in C_ρ.

(b) A finite set U of tokens in $C_{\iota(\rho)}$ is *consistent* (i.e., $\in \mathrm{Con}_{\iota(\rho)}$) if all its elements start with the same constructor C, say of arity $\rho \to \iota(\rho) \dots \to \iota(\rho) \to \iota(\rho)$. Let $U = \{\mathrm{C}V_1 a_{11}^* \dots a_{1n}^*, \dots, \mathrm{C}V_m a_{m1}^* \dots a_{mn}^*\}$. Then we require that (i) $V_1 \cup \dots \cup V_m$ is consistent (i.e., $\in \mathrm{Con}_\rho$) and (ii) the sets U_i consisting of all (proper) tokens at the i-th argument position of some token in U are consistent (i.e., $\in \mathrm{Con}_{\iota(\rho)}$).

(c) $\{\mathrm{C}V_1 a_{11}^* \dots a_{1n}^*, \dots, \mathrm{C}V_m a_{m1}^* \dots a_{mn}^*\} \vdash_{\iota(\rho)} \mathrm{C}Va_1^* \dots a_n^*$ if and only if (i) $V_1 \cup \dots \cup V_m \vdash_\rho V$ and (ii) for each set U_i as in (b) above we have $U_i \vdash_{\iota(\rho)} a_i^*$ (where $U_i \vdash *$ is taken to be true).

Case $\tau \to \sigma$. Tokens, consistency and entailment for $\boldsymbol{C}_{\tau\to\sigma}$ are defined as done generally in Section 2.1.2 (on function spaces). In more detail:

(a) Tokens in $\boldsymbol{C}_{\tau\to\sigma}$ are pairs (U, b) with $U \in \mathrm{Con}_\tau$ and $b \in C_\sigma$.

(b) $\{\,(U_i, b_i) \mid i \in I\,\} \in \mathrm{Con}_{\tau\to\sigma}$ is defined to mean

$$\forall_{J\subseteq I}\Big(\bigcup_{j\in J} U_j \in \mathrm{Con}_\tau \to \{\,b_j \mid j \in J\,\} \in \mathrm{Con}_\sigma\Big).$$

(c) $W \vdash_{\tau\to\sigma} (U, b)$ is defined to mean $WU \vdash_\sigma b$.

LEMMA 2.1.6. $(C_\tau, \mathrm{Con}_\tau, \vdash_\tau)$ *is an information system.*

PROOF. *Case* $\iota(\vec{\rho})$. For every constructor, for instance $\mathrm{C}\colon \rho \to \iota(\rho) \to \iota(\rho)$, we need to prove the axioms of information systems. We only give details for the last one, transitivity. By definition we can assume

$$\{\mathrm{C}U_1 a_1^*, \dots \mathrm{C}U_n a_n^*\} \vdash_{\iota(\rho)} \mathrm{C}V_j b_j^* \quad \text{and} \quad \{\mathrm{C}V_1 b_1^*, \dots, \mathrm{C}V_m b_m^*\} \vdash_{\iota(\rho)} \mathrm{C}Wc^*$$

By definition we have

$$\begin{array}{ll} U_1 \cup \dots \cup U_n \vdash_\rho V_j & \\ \{a_1^*, \dots, a_n^*\} \vdash_{\iota(\rho)} b_j^* & \text{and} \\ & V_1 \cup \dots \cup V_m \vdash_\rho W \\ & \{b_1^*, \dots, b_m^*\} \vdash_{\iota(\rho)} c^* \end{array}$$

By the (main and side) induction hypotheses we obtain

$$\begin{array}{l} U_1 \cup \dots \cup U_n \vdash_\rho W \\ \{a_1^*, \dots, a_n^*\} \vdash_{\iota(\rho)} c^* \end{array}$$

Hence the goal $\{\mathrm{C}U_1 a_1^*, \dots \mathrm{C}U_n a_n^*\} \vdash_{\iota(\rho)} \mathrm{C}Wc^*$ follows by definition.

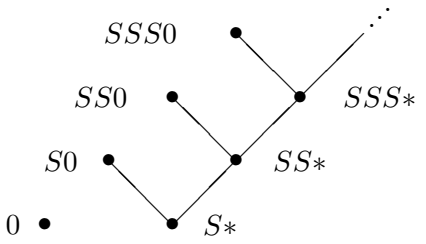


FIGURE 1. Tokens and entailment for $\mathbb{N}$

Case $\tau \to \sigma$. As in Section 2.1.2. □

Observe that all the notions involved are computable: $a \in C_\tau$, $U \in \mathrm{Con}_\tau$ and $U \vdash_\tau a$.

DEFINITION (Partial continuous functionals). The ideals $x \in |\boldsymbol{C}_\tau|$ are called *partial continuous functionals* of type τ. Since $\boldsymbol{C}_{\tau\to\sigma} = \boldsymbol{C}_\tau \to \boldsymbol{C}_\sigma$, the partial continuous functionals of type $\tau \to \sigma$ correspond to the continuous functions from $|\boldsymbol{C}_\tau|$ to $|\boldsymbol{C}_\sigma|$. A partial continuous functional $x \in |\boldsymbol{C}_\tau|$ is *computable* if it is recursively enumerable when viewed as a set of tokens.

DEFINITION (Cototal and total ideals of closed base type). Let $\iota(\vec{\rho})$ be a closed base type. Its tokens can be seen as constructor trees with some recursive argument positions occupied by $*$. An ideal x in $\boldsymbol{C}_\rho$ is *cototal* if for each of its tokens $P(*)$ with a distinguished occurrence of $*$ there is another token of the form $P(\mathrm{C}\vec{\emptyset}\vec{*})$ in x. We call x *total* if it is cototal and finite.

2.1.6. Examples. The tokens for the algebra $\mathbb{N}$ 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).

Dyadic rational numbers in the interval $(-1, 1)$ are those of the form

$$\sum_{n<m} \frac{k_n}{2^{n+1}} \qquad \text{with } k_n \in \{-1, 1\}.$$

A pictorial representation is in Figure 2. Irrational real numbers like $\frac{1}{2}\sqrt{2}$ then can be seen as infinite paths (or "streams"). Both of these objects appear in our present setting as total or cototal ideals of certain closed base types. This connection will make it possible to extract algorithms on stream-represented real numbers from proofs talking about ordinary reals given by Cauchy sequences with moduli.

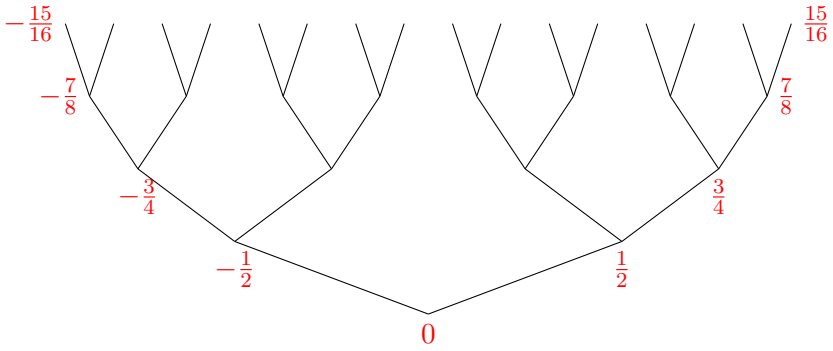
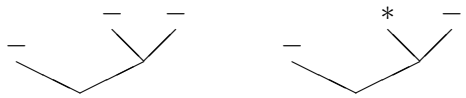


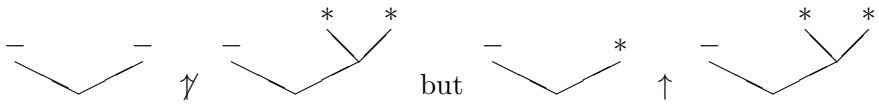
FIGURE 2. Dyadic rationals.

To establish such a connection we first consider the algebra $\mathbb{Y}$ of binary trees. Tokens in $\boldsymbol{C}_{\mathbb{Y}}$:

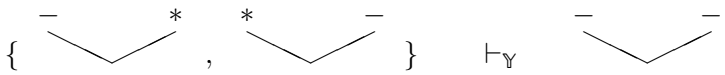


Consistency in $\boldsymbol{C}_{\mathbb{Y}}$:

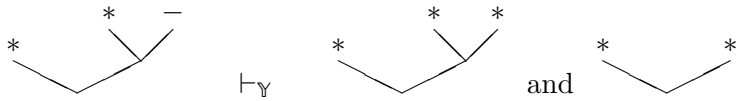
Moreover



Entailment in $\boldsymbol{C}_{\mathbb{Y}}$:

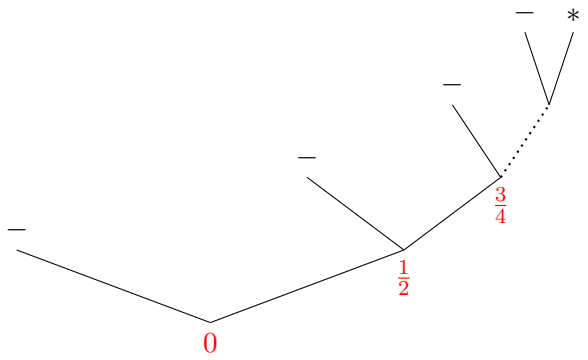


and also

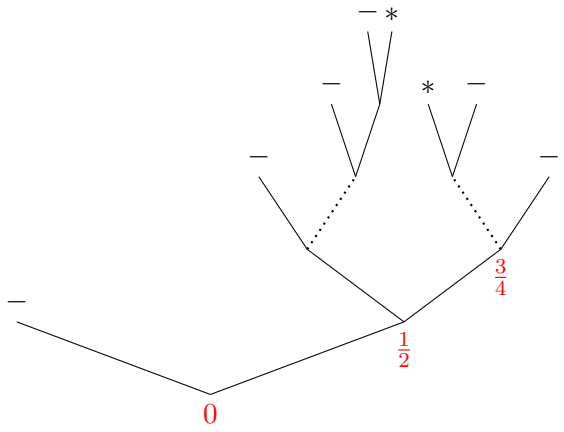


Ideals in $\boldsymbol{C}_{\mathbb{Y}}$:

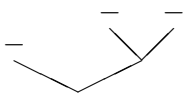
(1) $R :=$ closure of all



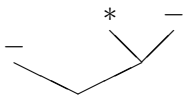
(2) L is defined similarly
(3) $L \cup R$
(4) $\frac{1}{2} :=$ closure of all



(5) Closure of



(6) Closure of



Among these are

$(1)-(4)$	cototal ideals,
(5)	a total ideal,
(6)	a general ideal.

Next we consider the algebra $\mathbb{L}(\mathbb{B})$ of lists of booleans, to represent finite or infinite paths. Tokens in $\boldsymbol{C}_{\mathbb{L}(\mathbb{B})}$ are for instance

$$\{\mathtt{ff}\} :: \{\mathtt{ff}\} :: \{\mathtt{tt}\} :: \mid \qquad \text{abbreviated} \qquad RRL \mid$$

Consistency in $\boldsymbol{C}_{\mathbb{L}(\mathbb{B})}$:

$$RL \mid \not\vdash RRL*, \qquad RR* \uparrow RRL*$$

Entailment in $\boldsymbol{C}_{\mathbb{L}(\mathbb{B})}$:

$$RRL* \vdash RR*, R* \qquad \text{(and also e.g. } RR\emptyset*),$$
$$RRL \mid \vdash RRL*, RR*, R*$$

Ideals in $\boldsymbol{C}_{\mathbb{L}(\mathbb{B})}$:

total:	Closure of $RLRR \mid$
cototal:	Closure of all $RLRR \ldots R*$ and all $RRLL \ldots L*$
general:	Closure of $RLRR*$

2.1.7. Bisimilarity. For closed ground types equality of cototal ideals can be characterized by *bisimilarity*. As an example we consider the algebra $\mathbb{Y}$ of binary trees. We define bisimilarity $\approx_{\mathbb{Y}}$ as the largest relation on $\mathcal{C}_{\mathbb{Y}}$ satisfying the *closure* axiom $\approx_{\mathbb{Y}}^{-}$:

$$\forall_{x,x'}(x \approx x' \to (x \equiv - \wedge x' \equiv -) \vee \exists_{x_1,x_2,x_1',x_2'}(x_1 \approx x_1' \wedge x_2 \approx x_2' \wedge x \equiv \mathrm{C}x_1x_2 \wedge x' \equiv \mathrm{C}x_1'x_2'))$$

with C for the Branch constructor. Being the "largest" relation means that any other relation ("competitor") X satisfying the same closure property is below $\approx_{\mathbb{Y}}$, i.e., we require the *greatest-fixed-point* property $\approx_{\mathbb{Y}}^{+}$:

$$\forall_{x,x'}(Xxx' \to (x \equiv - \wedge x' \equiv -) \vee \exists_{x_1,x_2,x_1',x_2'}((x_1 \approx x_1' \vee Xx_1x_1') \wedge (x_2 \approx x_2' \vee Xx_2x_2') \wedge x \equiv \mathrm{C}x_1x_2 \wedge x' \equiv \mathrm{C}x_1'x_2')) \to$$
$$X \subseteq \approx.$$

For ideals $x, x' \in \mathcal{C}_{\mathbb{Y}}$ we show

LEMMA 2.1.7 (Bisimilarity). *$x \approx_{\mathbb{Y}} x'$ implies $x \equiv x'$.*

PROOF. Let a range over tokens for $\mathbb{Y}$, and define the *height* $|a^*|$ of an extended token a^* by $|*| := 0$, $|-| := 1$, $|\mathrm{C}a_1^*a_2^*| := 1 + \max(|a_1^*|, |a_2^*|)$. By induction on the height $|a^*|$ of extended tokens a^* we prove that for all ideals x, x' and extended tokens $a^* \in x$ we have $a^* \in x'$. It suffices to consider the case $\mathrm{C}a_1^*a_2^*$. From $x \approx_{\mathbb{Y}} x'$ we obtain by the closure axiom x_1, x_2, x_1', x_2' with

$$x_1 \approx x_1' \wedge x_2 \approx x_2' \wedge x \equiv \mathrm{C}x_1x_2 \wedge x' \equiv \mathrm{C}x_1'x_2'.$$

Then $a_i^* \in x_i$ (for $i = 1, 2$), and by IH $a_i^* \in x_i'$. Thus $\mathrm{C}a_1^*a_2^* \in x'$. □

From the Bisimilarity Lemma we obtain the following characterization of $\approx_{\mathbb{Y}}$ on $\mathcal{C}_{\mathbb{Y}}$. We define ${}^{\mathrm{co}}T_{\mathbb{Y}}$ as the largest subset of $\mathcal{C}_{\mathbb{Y}}$ satisfying the *closure* axiom ${}^{\mathrm{co}}T_{\mathbb{Y}}^-$:

$$\forall_x(x \in {}^{\mathrm{co}}T \to x \equiv - \vee \exists_{x_1,x_2}(x_1 \in {}^{\mathrm{co}}T \wedge x_2 \in {}^{\mathrm{co}}T \wedge x \equiv \mathrm{C}x_1x_2)).$$

Again we require the *greatest-fixed-point* property ${}^{\mathrm{co}}T_{\mathbb{Y}}^+$:

$$\forall_x(x \in X \to (x \equiv -) \vee \exists_{x_1,x_2}(x_1 \in {}^{\mathrm{co}}T \cup X \wedge x_2 \in {}^{\mathrm{co}}T \cup X \wedge x \equiv \mathrm{C}x_1x_2)) \to X \subseteq {}^{\mathrm{co}}T.$$

For ideals $x, x' \in \mathcal{C}_{\mathbb{Y}}$ we show

LEMMA 2.1.8 (Characterization of $\approx_{\mathbb{Y}}$).

$$x \approx_{\mathbb{Y}} x' \leftrightarrow x, x' \in {}^{\mathrm{co}}T_{\mathbb{Y}} \wedge x \equiv x'.$$

PROOF. "$\to$". By Lemma 2.1.7 it remains to prove $x \approx_{\mathbb{Y}} x' \to x \in {}^{\mathrm{co}}T_{\mathbb{Y}}$. To this end we apply ${}^{\mathrm{co}}T_{\mathbb{Y}}^+$ with competitor $X := \{\, x \mid \exists_{x'}(x \approx_{\mathbb{Y}} x') \,\}$. It suffices to prove the premise. Fix x, x' with $x \approx_{\mathbb{Y}} x'$. The goal is

$$(x \equiv -) \vee \exists_{x_1,x_2}((x_1 \in {}^{\mathrm{co}}T \vee \exists_{x_1'}(x_1 \approx x_1')) \wedge (x_2 \in {}^{\mathrm{co}}T \vee \exists_{x_2'}(x_2 \approx x_2')) \wedge x \equiv \mathrm{C}x_1x_2).$$

By the closure property $\approx_{\mathbb{Y}}^-$ we have

$$(x \equiv - \wedge x' \equiv -) \vee \exists_{x_1,x_2,x_1',x_2'}(x_1 \approx x_1' \wedge x_2 \approx x_2' \wedge x \equiv \mathrm{C}x_1x_2 \wedge x' \equiv \mathrm{C}x_1'x_2')).$$

In the first case we have $x \equiv -$ and are done. In the second case we have x_1, x_2, x_1', x_2' with $x_1 \approx x_1'$, $x_2 \approx x_2'$ and $x \equiv \mathrm{C}x_1x_2$, and are done as well.

"$\leftarrow$". We prove $x \in {}^{\mathrm{co}}T_{\mathbb{Y}} \to x \equiv x' \to x \approx_{\mathbb{Y}} x'$ by the greatest-fixed-point property $\approx_{\mathbb{Y}}^+$ with competitor $X := \{\, x, x' \mid x \in {}^{\mathrm{co}}T_{\mathbb{Y}} \wedge x \equiv x' \,\}$. It suffices to prove the premise. Fix x, x' with $x \in {}^{\mathrm{co}}T_{\mathbb{Y}} \wedge x \equiv x'$. The goal is

$$(x \equiv - \wedge x' \equiv -) \vee \exists_{x_1,x_2,x_1',x_2'}((x_1 \approx x_1' \vee (x_1 \in {}^{\mathrm{co}}T_{\mathbb{Y}} \wedge x_1 \equiv x_1')) \wedge (x_2 \approx x_2' \vee (x_2 \in {}^{\mathrm{co}}T_{\mathbb{Y}} \wedge x_2 \equiv x_2')) \wedge x \equiv \mathrm{C}x_1x_2 \wedge x' \equiv \mathrm{C}x_1'x_2')).$$

By the closure property ${}^{\mathrm{co}}T_{\mathbb{Y}}^{-}$ applied to $x \in {}^{\mathrm{co}}T_{\mathbb{Y}}$ we have

$$(x \equiv -) \vee \exists_{x_1,x_2}(x_1 \in {}^{\mathrm{co}}T_{\mathbb{Y}} \wedge x_2 \in {}^{\mathrm{co}}T \wedge x \equiv \mathrm{C}x_1x_2).$$

In the first case we have $x \equiv -$ and are done, since $x \equiv x'$. In the second case we have $x_1, x_2 \in {}^{\mathrm{co}}T_{\mathbb{Y}}$ with $x \equiv \mathrm{C}x_1x_2$. Then we are done as well with $x_1' := x_1$ and $x_2' := x_2$, since again $x \equiv x'$. □

2.1.8. Constructors as continuous functions. Let ι be an algebra. Every constructor C generates the following ideal in the function space determined by the type of the constructor:

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

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

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

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

(For $\mathbb{N}$ we have $|r_S|(\{0\}) = \{S0, S*\}$ and $|r_S|(\{S0, S*\}) = \{SS0, SS*, S*\}$.) 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 2.1.9 (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

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

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

2.2. A term language for computable functionals

To work with computable functionals in a formal theory we need to have a language to denote them. Again lambda calculus is the appropriate tool, this time extended by constants for computable functionals.

Recall that a partial continuous functional is defined to be computable if it is the limit of a recursively enumerable set of finite approximations. We introduce a convenient way to define computable functionals, by means of defining equations or more precisely, computation rules. Therefore we extend the term language by constants D defined by certain "computation rules". The resulting term system can be seen as a common extension of Gödel's T (1958) and Plotkin's PCF (1977); we call it T^+.

2.2.1. A common extension T^+ of Gödel's T and Plotkin's PCF.

Definition (Terms). *Terms* of T^+ are built from (typed) variables and (typed) constants (constructors C or defined constants D; see the definition below) by application and abstraction:

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

The set $\mathrm{FV}(M)$ of free variables of a term M is defined by

$$\mathrm{FV}(x) := \{x\}, \qquad \mathrm{FV}(C) := \mathrm{FV}(D) := \emptyset,$$
$$\mathrm{FV}(\lambda_x M) := \mathrm{FV}(M) \setminus \{x\}, \qquad \mathrm{FV}(MN) := \mathrm{FV}(M) \cup \mathrm{FV}(N).$$

Definition (Conversion). We define a conversion relation $\mapsto_\beta$ for terms similarly to what we did for derivation terms:

$$(\lambda_x M(x))^{\tau\to\sigma} N^\tau \mapsto_\beta M(N)^\sigma.$$

In addition we will employ another conversion relation $\mapsto_\eta$ defined by

$$\lambda_x(Mx) \mapsto_\eta M \quad \text{if } x \notin \mathrm{FV}(M) \ (M \text{ not an abstraction}).$$

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

$$D\vec{P}_i(\vec{y}_i) = M_i \qquad (i = 1, \dots, n \text{ where } n \geq 0) \tag{7}$$

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 (7) 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 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 $\mathrm{C}\vec{P}$.

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 (structural) recursion and corecursion operators will be defined by computation rules.

The simplest example is the constant $\bot_\iota$ of type ι with no computation rules. 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* $=_\iota \colon \iota \to \iota \to \mathbb{B}$ for a closed base type ι can be defined easily by computation rules. For example,

$$\begin{array}{ll} (0 =_{\mathbb{N}} 0) = \mathtt{tt}, & (Sn =_{\mathbb{N}} 0) = \mathtt{ff}, \\ (0 =_{\mathbb{N}} Sm) = \mathtt{ff}, & (Sn =_{\mathbb{N}} Sm) = \ (n =_{\mathbb{N}} m). \end{array}$$

For the algebra $\mathbb{Y}$ of binary trees with constructors 0 (leaf) and C (construct a new tree from two given ones) we have

$$\begin{array}{ll} (0 =_{\mathbb{Y}} 0) = \mathtt{tt}, & (\mathrm{C}ts =_{\mathbb{Y}} 0) = \mathtt{ff}, \\ (0 =_{\mathbb{Y}} \mathrm{C}ts) = \mathtt{ff}, & (\mathrm{C}ts =_{\mathbb{Y}} \mathrm{C}t's') = \ (t =_{\mathbb{Y}} t'\ \mathrm{andb}\ s =_{\mathbb{Y}} s'). \end{array}$$

For the algebra $\mathbb{N}$ of natural numbers we have the doubling function

$$\begin{aligned}\mathrm{Double}(0) &:= 0,\\ \mathrm{Double}(S(n)) &:= S(S(\mathrm{Double}(n))).\end{aligned}$$

Addition (written infix) is defined similarly, this time with a parameter m:

$$\begin{aligned}m + 0 &:= m,\\ m + S(n) &:= S(m + n).\end{aligned}$$

Multiplication (again written infix) is defined using addition by

$$\begin{aligned}m \cdot 0 &:= 0,\\ m \cdot S(n) &:= (m \cdot n) + m.\end{aligned}$$

Similarly we can define all primitive recursive functions.

Up to now we have only considered examples of total functions, in the sense that total arguments are mapped to total values. But recall that in our setting functions need not be total. To give an example consider the algebra of streams defined by

$$\mathbb{S}(\alpha) := \mu_\xi(\alpha \to \xi \to \xi)$$

with a single constructor $\mathrm{C}\colon \alpha \to \mathbb{S}(\alpha) \to \mathbb{S}(\alpha)$. We write $\mathrm{C}_a u$ or $a :: u$ for $\mathrm{C}au$. This algebra differs from the ones previously considered by not having a nullary constructor. As a consequence it does not have non-empty total ideals, but clearly cototal ones. An example is $\{\, \mathrm{C}_a^n(*) \mid n \geq 1 \,\}$. We define the function Map of type $(\rho \to \sigma) \to \mathbb{S}(\rho) \to \mathbb{S}(\sigma)$ mapping its function argument $h\colon \rho \to \sigma$ over a stream u of type $\mathbb{S}(\rho)$ by the computation rule

$$\mathrm{Map}_h(a :: u) := (ha) :: \mathrm{Map}_h(u).$$

2.2.2. Recursion operators. Important examples of such constants D are the (structural) higher type *recursion operators* $\mathcal{R}_\iota^\tau$ introduced by Hilbert (1925) and Gödel (1958). They are used to construct maps from the algebra ι to the value type τ, by recursion on the structure of ι.

For instance, $\mathcal{R}_{\mathbb{N}}^\tau$ has type $\mathbb{N} \to \tau \to (\mathbb{N} \to \tau \to \tau) \to \tau$. It is defined by the computation rules

$$\begin{aligned}\mathcal{R}_{\mathbb{N}}^\tau 0af &:= a,\\ \mathcal{R}_{\mathbb{N}}^\tau (Sn)af &:= fn(\mathcal{R}_{\mathbb{N}}^\tau naf).\end{aligned}$$

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, $\mathcal{R}_{\mathbb{N}}^{\mathbb{N}} nm\lambda_{n,l}(Sl)$ defines addition $m+n$ by recursion on n. For $\lambda_{n,l}(Sl)$ we often write $\lambda_{\_,l}(Sl)$ since the bound variable n is not used.

It will be convenient to write an algebra form

$$\mu_\xi(\vec{\alpha}_i \to (\xi)_{\nu<n_i} \to \xi)_{i<k} \quad \text{as} \quad \mu_\xi((\rho_{i\nu}(\xi))_{\nu<n_i} \to \xi)_{i<k}.$$

Definition (Type of $\mathcal{R}^\tau_\iota$). For the algebra $\iota = \mu_\xi((\rho_{i\nu}(\xi))_{\nu<n_i} \to \xi)_{i<k}$ and the result type τ we define the type of the recursion operator $\mathcal{R}^\tau_\iota$ 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 recursive argument types we avoid the product type in $\iota \times \tau$ and take the two argument types ι and τ instead ("duplication").

Definition (Computation rules for $\mathcal{R}^\tau_\iota$). Let

$$\alpha_0 \to \ldots \to \alpha_{m-1} \to (\xi)_{i<n} \to \xi$$

be the type of the i-th constructor C_i of ι and consider a term $\mathrm{C}_i\vec{x}$ of type ι. We write $\vec{x}^P = x^P_0, \ldots, x^P_{m-1}$ for the *parameter arguments* $x^{\alpha_0}_0, \ldots, x^{\alpha_{m-1}}_{m-1}$ and $\vec{x}^R = x^R_0, \ldots, x^R_{n-1}$ for the *recursive arguments* $x^\iota_m, \ldots, x^\iota_{m+n-1}$. Writing $\mathcal{R}$ for $\mathcal{R}^\tau_\iota$ we take as its computation rules

$$\mathcal{R}(\mathrm{C}_i\vec{x})\vec{f} := f_i\vec{x}(\mathcal{R}x^R_0\vec{f}\,)\ldots(\mathcal{R}x^R_{n-1}\vec{f}\,).$$

Examples.

$$\begin{aligned}
&\mathcal{R}^\tau_{\mathbb{B}}\colon \mathbb{B} \to \tau \to \tau \to \tau,\\
&\mathcal{R}^\tau_{\mathbb{N}}\colon \mathbb{N} \to \tau \to (\mathbb{N} \to \tau \to \tau) \to \tau,\\
&\mathcal{R}^\tau_{\mathbb{P}}\colon \mathbb{P} \to \tau \to (\mathbb{P} \to \tau \to \tau) \to (\mathbb{P} \to \tau \to \tau) \to \tau,\\
&\mathcal{R}^\tau_{\mathbb{Y}}\colon \mathbb{Y} \to \tau \to (\mathbb{Y} \to \tau \to \mathbb{Y} \to \tau \to \tau) \to \tau,\\
&\mathcal{R}^\tau_{\mathbb{L}(\rho)}\colon \mathbb{L}(\rho) \to \tau \to (\rho \to \mathbb{L}(\rho) \to \tau \to \tau) \to \tau,\\
&\mathcal{R}^\tau_{\rho+\sigma}\colon \rho + \sigma \to (\rho \to \tau) \to (\sigma \to \tau) \to \tau,\\
&\mathcal{R}^\tau_{\rho\times\sigma}\colon \rho \times \sigma \to (\rho \to \sigma \to \tau) \to \tau.
\end{aligned}$$

It is a helpful exercise to write out the computation rules for these particular recursion operators.

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 $\mathbb{B}$) is *if-then-else*. Another example is

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

It could be used to define the *predecessor* function on $\mathbb{N}$ by the term

$$\mathrm{P}m := \mathcal{C}^{\mathbb{N}}_{\mathbb{N}} m 0 (\lambda_n n).$$

However, it is easier to define the predecessor function by the computation rules $\mathrm{P}0 := 0$ and $\mathrm{P}(Sn) := n$.

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 could be written in the form

$$[\mathbf{case}\ m^{\mathbb{N}}\ \mathbf{of}\ (0 \mapsto 0 \mid Sn \mapsto n)].$$

2.2.3. Corecursion. The computation rules for $\mathcal{R}$ work from the leaves towards the root, and terminate because total ideals are finite. If, however, we deal with cototal ideals, then a similar operator is available to define functions with cototal ideals as values, namely "corecursion".

To understand the type of a corecursion operator recall the constructor types $\kappa_i(\iota)$ of an algebra $\iota = \mu_\xi(\kappa_0, \dots, \kappa_{k-1})$, written as

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

The product of these k constructor types is isomorphic to

$$\sum_{i<k} \prod_{\nu<n_i} \rho_{i\nu}(\iota) \to \iota$$

and the type of the recursion operator $\mathcal{R}^\tau_\iota$ is isomorphic to

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

Dually for the algebra ι the type of its *destructor* D_ι (disassembling a constructor-built object into its parts) is

$$\iota \to \sum_{i<k} \prod_{\nu<n_i} \rho_{i\nu}(\iota).$$

The corecursion operator ${}^{\mathrm{co}}\mathcal{R}^\tau_\iota$ is used to construct a map from τ to ι by "corecursion" on the structure of ι. Its type is

$$\tau \to \Big(\tau \to \sum_{i<k} \prod_{\nu<n_i} \rho_{i\nu}(\iota+\tau)\Big) \to \iota. \tag{8}$$

Examples (Types and computation rules of corecursion operators).

$$\begin{aligned}
&{}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{B}} \colon \tau \to (\tau \to \mathbb{U}+\mathbb{U}) \to \mathbb{B},\\
&{}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{N}} \colon \tau \to (\tau \to \mathbb{U}+(\mathbb{N}+\tau)) \to \mathbb{N},\\
&{}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{P}} \colon \tau \to (\tau \to \mathbb{U}+(\mathbb{P}+\tau)+(\mathbb{P}+\tau)) \to \mathbb{P},\\
&{}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{Y}} \colon \tau \to (\tau \to \mathbb{U}+(\mathbb{Y}+\tau)\times(\mathbb{Y}+\tau)) \to \mathbb{Y},\\
&{}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{L}(\rho)} \colon \tau \to (\tau \to \mathbb{U}+\rho\times(\mathbb{L}(\rho)+\tau)) \to \mathbb{L}(\rho),\\
&{}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{S}(\rho)} \colon \tau \to (\tau \to \rho\times(\mathbb{S}(\rho)+\tau)) \to \mathbb{S}(\rho).
\end{aligned}$$

The computation rule for each of these is defined below. For $f\colon \rho\to\tau$ and $g\colon \sigma\to\tau$ we denote $\lambda_x(\mathcal{R}^\tau_{\rho+\sigma}xfg)$ of type $\rho+\sigma\to\tau$ by $[f,g]$, and similary for ternary sumtypes etcetera. The identity functions id below are of type $\iota\to\iota$ with ι the respective algebra.

$$\begin{aligned}
{}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{B}}xf &:= [\lambda_\mathtt{tt}, \lambda_\mathtt{ff}](fx),\\
{}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{N}}xf &:= [\lambda_0, \lambda_y(S([\mathrm{id}^{\mathbb{N}\to\mathbb{N}}, P_{\mathbb{N}}]y))](fx),\\
{}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{P}}xf &:= [\lambda_1, \lambda_y(S_0([\mathrm{id}, P_{\mathbb{P}}]y)), \lambda_y(S_1([\mathrm{id}, P_{\mathbb{P}}]y))](fx),\\
{}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{Y}}xf &:= [\lambda_0, \lambda_{y_0,y_1}(\mathrm{C}([\mathrm{id}, P_{\mathbb{Y}}]y_0)([\mathrm{id}, P_{\mathbb{Y}}]y_1))](fx),\\
{}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{L}(\rho)}xf &:= [\lambda_[], \lambda_{y_0,y_1}(y_0 :: [\mathrm{id}, P_{\mathbb{L}(\rho)}]y_1)](fx),\\
{}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{S}(\rho)}xf &:= (fx)_0 :: [\mathrm{id}, P_{\mathbb{S}(\rho)}](fx)_1
\end{aligned}$$

with $(fx)_i$ the i-th component of the pair fx, and $P_\alpha := \lambda_x({}^{\mathrm{co}}\mathcal{R}^\tau_\alpha xf)$ for $\alpha \in \{\mathbb{N}, \mathbb{P}, \mathbb{Y}, \mathbb{L}(\rho), \mathbb{S}(\rho)\}$.

Definition. The (single) computation rule for ${}^{\mathrm{co}}\mathcal{R}^\tau_\iota$ of type (8) is

$${}^{\mathrm{co}}\mathcal{R}^\tau_\iota xf := [g_0, \dots, g_{k-1}](fx)$$

where g_i of type $\prod_{\nu<n_i}\rho_{i\nu}(\iota+\tau)\to\iota$ is defined as

$$g_i := \lambda_{\vec{x}}(\mathrm{C}_i(N_\nu)_{\nu<n_i}) \qquad \text{with } x_\nu\colon \rho_{i\nu}(\iota+\tau),$$

$$N_\nu := \begin{cases} x_\nu & \text{if } \rho_{i\nu}(\xi) \text{ is a parameter arg. type,}\\ [\mathrm{id}^{\iota\to\iota}, P^{\tau\to\iota}]x^{\iota+\tau}_\nu) & \text{otherwise,}\end{cases}$$

and $P := \lambda_x({}^{\mathrm{co}}\mathcal{R}^\tau_\iota xf)$ contains the corecursive call.

REMARK. It can be difficult to read the computation rules for corecursion operators. However, it helps if we know some properties of the "step" function f. For instance we have

$$
{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{N}} x f = \begin{cases} 0 & \text{if } fx = \mathrm{DummyL}^{\mathbb{U}+(\mathbb{N}+\tau)} \\ Sn & \text{if } fx = \mathrm{Inr}(\mathrm{InL}^{\mathbb{N}\to\mathbb{N}+\tau} n) \\ S({}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{N}} x' f) & \text{if } fx = \mathrm{Inr}(\mathrm{InR}^{\tau\to\mathbb{N}+\tau} x') \end{cases}
$$

$$
{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{Y}} x f = \begin{cases} 0 & \text{if } fx = \mathrm{DummyL}^{\mathbb{U}+(\mathbb{Y}+\tau)\times(\mathbb{Y}+\tau)} \\ \mathrm{C} t s & \text{if } fx = \mathrm{Inr}\langle \mathrm{InL}^{\mathbb{Y}\to\mathbb{Y}+\tau} t, \mathrm{InL}^{\mathbb{Y}\to\mathbb{Y}+\tau} s\rangle \\ \mathrm{C} t({}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{Y}} y f) & \text{if } fx = \mathrm{Inr}\langle \mathrm{InL}^{\mathbb{Y}\to\mathbb{Y}+\tau} t, \mathrm{InR}^{\tau\to\mathbb{Y}+\tau} y\rangle \\ \mathrm{C}({}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{Y}} y f) s & \text{if } fx = \mathrm{Inr}\langle \mathrm{InR}^{\tau\to\mathbb{Y}+\tau} y, \mathrm{InL}^{\mathbb{Y}\to\mathbb{Y}+\tau} s\rangle \\ \mathrm{C}({}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{Y}} y f)({}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{Y}} z f) & \text{if } fx = \mathrm{Inr}\langle \mathrm{InR}^{\tau\to\mathbb{Y}+\tau} y, \mathrm{InR}^{\tau\to\mathbb{Y}+\tau} z\rangle \end{cases}
$$

$$
{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{S}(\rho)} x f = \begin{cases} a :: u & \text{if } fx = \langle a, \mathrm{InL}^{\mathbb{S}(\rho)\to\mathbb{S}(\rho)+\tau} u\rangle \\ a :: {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{S}(\rho)} x' f & \text{if } fx = \langle a, \mathrm{InR}^{\tau\to\mathbb{S}(\rho)+\tau} x'\rangle. \end{cases}
$$

Recall that Map of type $(\rho \to \sigma) \to \mathbb{S}(\rho) \to \mathbb{S}(\sigma)$ maps its function argument $h\colon \rho \to \sigma$ over a stream u of type $\mathbb{S}(\rho)$ (see Section 2.2.1, page 38). It is an easy exercise to give an alternative definition of the function Map by means of the corecursion operator.

REMARK. It is possible to define interesting cototal objects by means of corecursion operators. For instance the rightmost infinite path in the algebra $\mathbb{Y}$ of binary trees is $t_R := {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{Y}} x_0 f_0$ with τ, x_0 arbitrary (for instance $\tau := \mathbb{U}$, $x_0 := \mathrm{Dummy}^{\mathbb{U}}$) and

$$
f_0 x^{\tau} := \mathrm{Inr}\langle \mathrm{InL}^{\mathbb{Y}\to\mathbb{Y}+\tau} 0, \mathrm{InR}^{\tau\to\mathbb{Y}+\tau} x\rangle.
$$

For the leftmost path we similary have t_L.

Another example is a function converting a real number given as a Cauchy sequence of rationals together a Cauchy modulus into an infinite stream of signed digits $\{-1, 0, 1\}$. Such a function can be defined by a simple corecursion. One can extract it from a proof (using "coinduction") of the fact that such a conversion exists.

2.3. Denotational semantics

We now set up a connection between the model $(|\mathbf{C}_\rho|)_\rho$ of partial continuous functionals described in Section 2.1 and the term system T^+ from Section 2.2. The main point is to clarify how we can use computation rules to define an ideal z in a function space. The general idea is to inductively define the set of tokens (U, a) 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 $(\vec{U}, a) \in [\![\lambda_{\vec{x}} M]\!]$.

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

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

We use the following notation. $(\vec{U}, a)$ means $(U_1, (U_2, \dots (U_n, a)) \dots)$, and $(\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} M]\!]$ means $(\vec{U}, a) \in [\![\lambda_{\vec{x}} M]\!]$ for all (finitely many) $a \in V$.

Definition (Inductive, of $(\vec{U}, a) \in [\![\lambda_{\vec{x}} M]\!]$).

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

For every constructor C and defined constant D we have

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

with one such rule (D) for every computation rule $D\vec{P}(\vec{y}\,) = M$.