CHAPTER 2

Partial continuous functionals

The objects studied in mathematics have types, which in many cases are function types, possibly of a higher type. Such objects in most cases are infinite, and we intend to describe them in terms of their finite approximations. An appropriate framework for such an approach are the partial continuous functionals of Scott (1982) and Ershov (1977). Continuity of a function f here means that for every approximation V of the value $f(x)$ there is an approximation U of the argument x such that $f[U]$ has more information than V. We define the partial continuous functionals via Scott's information systems.

2.1. Information systems

The basic idea of information systems is to provide an axiomatic setting to describe approximations of abstract objects by concrete, finite ones. We 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).

2.1.1. Ideals.

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

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

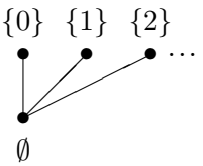
Definition. The *ideals* (also called *objects*) of an information system $\boldsymbol{A} = (A, \mathrm{Con}, \vdash)$ are defined to be those subsets x of A which satisfy

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

We write $x \in |\boldsymbol{A}|$ to mean that x is an ideal of $\boldsymbol{A}$.

Examples. The *deductive closure* $\overline{U} := \{\, a \in A \mid U \vdash a \,\}$ of $U \in \mathrm{Con}$ is an ideal.

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

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

It is easy 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 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 $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$. Assume $r \in |\boldsymbol{A} \to \boldsymbol{B}|$. 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.*

(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

[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.

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

$|r|(z)$ is deductively closed: let $b_1, \ldots, b_n \in |r|(z)$ and $\{b_1, \ldots, b_n\} \vdash_B b$. 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 one can easily show

$$b \in |\hat{f}|(z) \leftrightarrow b \in f(z)$$

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) → (a). As above this follows from

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

□

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.

2.2. Objects of a given type

We now use information systems to define the objects of the Scott-Ershov model of partial continuous functionals, each of a given "type".

2.2.1. Types. If τ and σ are types, we clearly want that $\tau \to \sigma$ is a type as well, to be called "function type". But we have to start somewhere. The basic idea is that we consider finite lists of (named) "constructor types".

Types may involve type variables $\alpha, \beta, \gamma, \xi, \zeta$. Iterated arrows are understood as associated to the right. For example, $\alpha \to \beta \to \gamma$ means $\alpha \to (\beta \to \gamma)$, not $(\alpha \to \beta) \to \gamma$.

Definition. *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. 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 is *recursive* if it has a recursive argument type. Each list of named constructor types with all of its parameter argument types distinct determines a *base type* $\iota_{\vec{\kappa}}$. Base types given by a list of named constructors are called *algebras*.

For some common lists of named constructor types there are standard names for the corresponding base types:

$\mathrm{Dummy}\colon \xi$	$\mathbb{U}$ (unit),
$\mathtt{tt}\colon \xi, \mathtt{ff}\colon \xi$	$\mathbb{B}$ (booleans),
$\mathrm{SdL}\colon \xi, \mathrm{SdM}\colon \xi, \mathrm{SdR}\colon \xi$	$\mathbb{D}$ (signed digits),
$\mathrm{Zero}\colon \xi, \mathrm{Succ}\colon \xi \to \xi$	$\mathbb{N}$ (natural numbers, unary),
$\mathrm{One}\colon \xi, \mathrm{S}_0\colon \xi \to \xi, \mathrm{S}_1\colon \xi \to \xi$	$\mathbb{P}$ (positive numbers, binary),
$\mathrm{L}\colon \xi, \mathrm{B}\colon \xi \to \xi \to \xi$	$\mathbb{Y}$ (binary trees)

and with parameter types

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

DEFINITION. *Types* are inductively defined by

(a) Every type variable α is a type.
(b) If $\vec{\kappa}(\vec{\alpha})$ is a list of named constructor types and $\vec{\tau}$ are types where the length of $\vec{\tau}$ is the number of parameters in $\vec{\kappa}$, then $\iota_{\vec{\kappa}(\vec{\tau})}$ is a type.
(c) It τ and σ are types, then so is $\tau \to \sigma$.

Types of the form $\tau \to \sigma$ are called *function types*, and types of the form $\iota_{\vec{\kappa}(\vec{\tau})}$ *base types*.

If the base type corresponding to a list of named constructor types has a standard name, then we use this name to denote the base type.

A type is *closed* it it has no parameters. Let $\tau(\vec{\alpha})$ be a type with $\vec{\alpha}$ its parameters, and let $\vec{\rho}$ be closed types. We define the *level* of $\tau(\vec{\rho})$ by

$$\begin{aligned}\mathrm{lev}(\iota_{\vec{\kappa}(\vec{\rho})}) &:= \max(\mathrm{lev}(\vec{\rho})),\\ &\qquad \text{where the length of } \vec{\rho} \text{ is the number of parameters in } \vec{\kappa}(\vec{\alpha}),\\ \mathrm{lev}(\tau \to \sigma) &:= \max(\mathrm{lev}(\sigma), 1 + \mathrm{lev}(\tau)).\end{aligned}$$

Examples of base types:

- $\mathbb{L}(\alpha)$, $\mathbb{L}(\mathbb{L}(\alpha))$, $\alpha \times \beta$ are base types of level 0.
- $\mathbb{L}(\mathbb{L}(\mathbb{N}))$, $\mathbb{N} + \mathbb{B}$, $\mathbb{Z} := \mathbb{P} + \mathbb{U} + \mathbb{P}$, $\mathbb{Q} := \mathbb{Z} \times \mathbb{P}$ are closed base types of level 0.
- $\mathbb{R} := (\mathbb{N} \to \mathbb{Q}) \times (\mathbb{P} \to \mathbb{N})$ is a closed base type of level 1.

2.2.2. The information system of a given type. For every closed type τ we define the information system $\boldsymbol{A}_\tau = (A_\tau, \mathrm{Con}_\tau, \vdash_\tau)$. The ideals $x \in |\boldsymbol{A}_\tau|$ are the *partial continuous functionals* of type τ. Since we will have $\boldsymbol{A}_{\tau\to\sigma} = \boldsymbol{A}_\tau \to \boldsymbol{A}_\sigma$, the partial continuous functionals of type $\tau \to \sigma$ will correspond to the continuous functions from $|\boldsymbol{A}_\tau|$ to $|\boldsymbol{A}_\sigma|$.

DEFINITION (Information system of a closed type τ). We simultaneously define $A_{\iota_{\vec{\kappa}(\vec{\tau})}}$, $A_{\tau\to\sigma}$, $\mathrm{Con}_{\iota_{\vec{\kappa}(\vec{\tau})}}$ and $\mathrm{Con}_{\tau\to\sigma}$.

(a) The *tokens* $a \in A_{\iota_{\vec{\kappa}(\vec{\tau})}}$ are the type correct constructor expressions

$$\mathrm{C}U_1 \dots U_m a_1^* \dots a_n^*$$

with C the name of a constructor type $\vec{\alpha} \to (\xi)_{i<n} \to \xi$ from $\vec{\kappa}$, all U_j $(1 \le j \le m)$ from Con_{τ_j} and each a_i^* $(1 \le i \le n)$ an *extended token*, i.e., a token or the special symbol $*$ which carries no information.

(b) The tokens in $A_{\tau\to\sigma}$ are the pairs (U, b) with $U \in \mathrm{Con}_\tau$ and $b \in A_\sigma$.

(c) A finite set U of tokens in $A_{\iota_{\vec{\kappa}(\vec{\tau})}}$ is *consistent* (i.e., $U \in \mathrm{Con}_{\iota_{\vec{\kappa}(\vec{\tau})}}$) if
   (i) all its elements start with the same constructor C, say of arity $\vec{\tau} \to (\iota_{\vec{\kappa}(\vec{\tau})})_{i<n} \to \iota_{\vec{\kappa}(\vec{\tau})}$,
   (ii) the union V_j of all Con-sets at the j-th $(1 \le j \le m)$ argument position of some token in U is in Con_{τ_j}, and
   (iii) all $U_i \in \mathrm{Con}_{\iota_{\vec{\kappa}(\vec{\tau})}}$ $(1 \le i \le n)$, where U_i consists of all (proper) tokens at the $(m+i)$-th argument position of some token in U.

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

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

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

(e) $\{\mathrm{C}\vec{U}_1\vec{a}_1^*, \dots, \mathrm{C}\vec{U}_l\vec{a}_l^*\} \vdash_{\iota_{\vec{\kappa}(\vec{\tau})}} \mathrm{C}'\vec{W}\vec{a^*}$ is defined to mean $\mathrm{C} = \mathrm{C}'$, $l \ge 1$, $V_j \vdash W_j$ with V_j as in (c) above and $U_i \vdash a_i^*$ with U_i as in (c) above (and $U \vdash *$ defined to be true).

(f) $W \vdash_{\tau\to\sigma} (U, b)$ is defined to mean $WU \vdash_\sigma b$, where application WU of $W = \{\,(U_i, b_i) \mid i \in I\,\} \in \mathrm{Con}_{\tau\to\sigma}$ to $U \in \mathrm{Con}_\tau$ is defined to be $\{\,b_i \mid U \vdash_\tau U_i\,\}$.

Note that the present definition is by recursion on the *height* of the syntactic expressions involved, defined by

$$\begin{aligned}
|\alpha| &:= 0,\\
|\iota_{\vec{\kappa}(\vec{\tau})}| &:= 1 + \max\{\,|\tau_i| \mid \tau_i \in \vec{\tau}\,\},\\
|\tau \to \sigma| &:= \max\{1 + |\tau|, |\sigma|\},\\
|\mathrm{C}U_1 \dots U_m a_1^* \dots a_n^*| &:= 1 + \max(\{\,|U_j| \mid 1 \le j \le m\,\} \cup \{\,|a_i^*| \mid 1 \le i \le n\,\}),\\
|*| &:= 0,\\
|(U, b)| &:= 1 + \max\{|U|, |b|\},\\
|\{\,a_i \mid i \in I\,\}| &:= 1 + \max\{\,|a_i| \mid i \in I\,\},\\
|U \vdash a| &:= 1 + \max\{|U|, |a|\}.
\end{aligned}$$

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

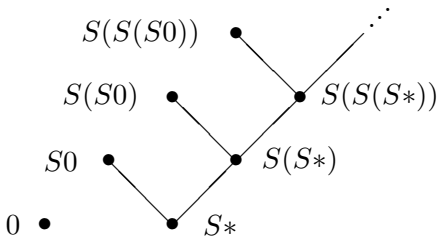


FIGURE 1. Tokens and entailment for $\mathbb{N}$

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

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

For example, the tokens for the base type $\mathbb{N}$ are shown in Figure 1 (with 0 for Zero and S for Succ). 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 example, consider the base type $\mathbb{Y}$ of binary trees with a nullary constructor L (for Leaf) and a binary B (for Branch). Then $\{B(L,*), B(*,L)\}$ is consistent, and it entails $B(L,L)$.

2.2.3. Cototal and total ideals of a closed base type. Let τ be a closed base type, for simplicity without parameters. An example is the type $\mathbb{Y}$ of binary trees. We want to take a closer look at the elements of $|\boldsymbol{A}_\tau|$, i.e., the ideals in $\boldsymbol{A}_\tau$. Among them it seems natural to single out those with the following property, to be called "cototality":

DEFINITION (Cototal ideal). Consider a token in the ideal, and in this token (a constructor expression) a position occupied by the symbol $*$ (indicating "no information"). Then it must be possible to further analyze the ideal, in the following sense. There must be another token in the ideal where this symbol $*$ is replaced by a constructor expression $\mathrm{C}\vec{*}$ with C the name of a constructor of the underlying base type.

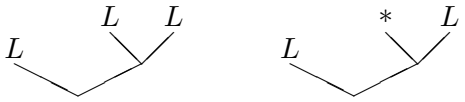
Clearly cototal ideals may be infinite. However, they can be analyzed (or "destructed") up to an arbitrary depth. It may also happen that a cototal ideal is finite. In this case it is called "total".

Hence in our model of partial continuous functionals already at base types we have ideals (i.e., objects) which are either

- cototal and infinite, or

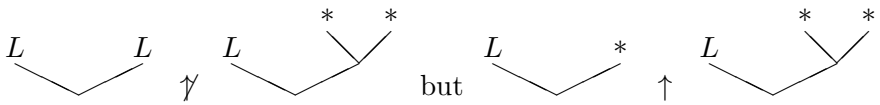
- total (i.e., cototal and finite), or
- neither.

We give some examples for the base type $\mathbb{Y}$, with a more pictorial representation. Tokens in $\boldsymbol{A}_{\mathbb{Y}}$ (omitting B):

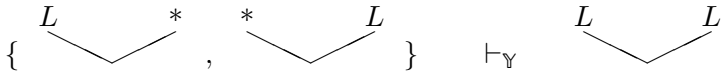


Consistency in $\boldsymbol{A}_{\mathbb{Y}}$:

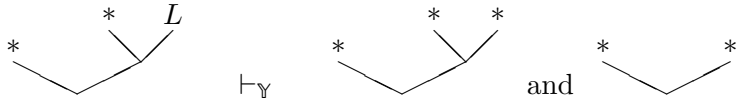
Moreover



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

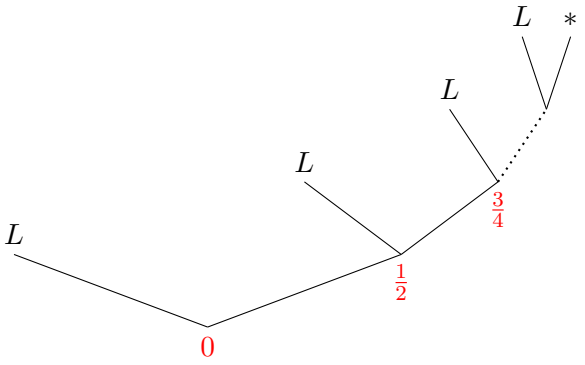


and also



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

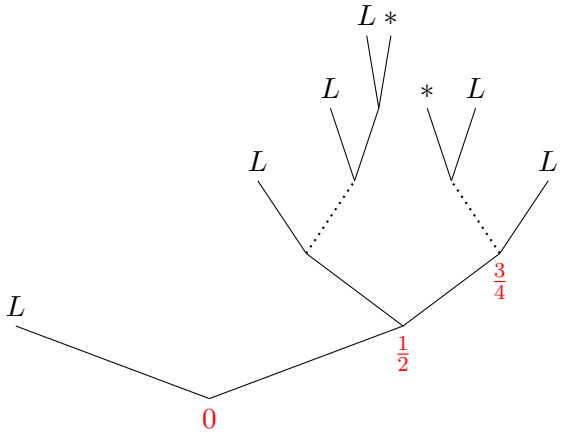
(i) $1 :=$ closure of all



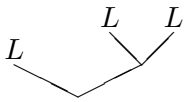
(ii) -1 is defined similarly

(iii) $-1 \cup 1$

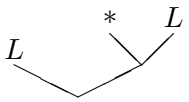
(iv) $\frac{1}{2}$:= closure of all



(v) Closure of



(vi) Closure of



Among these

(i) – (iv)	are infinite cototal ideals,
(v)	is a total ideal, and
(vi)	is an ideal but not cototal.

Cototality and totality of ideals in $\boldsymbol{A}_{\mathbb{Y}}$ can also be characterized differently. We can represent the set of total ideals in $\boldsymbol{A}_{\mathbb{Y}}$ as a predicate "inductively defined" by the two "clauses"

$$\begin{aligned}&(T_{\mathbb{Y}})^+_0 \colon L \in T_{\mathbb{Y}},\\ &(T_{\mathbb{Y}})^+_1 \colon \forall_{x_1,x_2}(x_1, x_2 \in T_{\mathbb{Y}} \to Bx_1x_2 \in T_{\mathbb{Y}})\end{aligned} \tag{1}$$

To state that $T_{\mathbb{Y}}$ is the *least* set satisfying the two clauses we use the *elimination* or *least fixed point* property

$$T^-_{\mathbb{Y}} \colon L \in X \to \forall_{x_1,x_2}(x_1, x_2 \in T_{\mathbb{Y}} \cap X \to Bx_1x_2 \in X) \to T_{\mathbb{Y}} \subseteq X. \tag{2}$$

It says that every "competitor" X satisfying the same clauses contains $T_{\mathbb{Y}}$.

We now want to represent the set of *cototal* binary trees by a "coinductively" defined predicate ${}^{\mathrm{co}}T_{\mathbb{Y}}$. Note that the conjunction of the two clauses of $T_{\mathbb{Y}}$ is equivalent to

$$\forall_x((x \equiv L) \vee \exists_{x_1,x_2}(x_1, x_2 \in T_{\mathbb{Y}} \wedge x \equiv Bx_1x_2) \to x \in T_{\mathbb{Y}}).$$

Since $T_{\mathbb{Y}}$ is the least predicate with this property we even have the equivalence

$$\forall_x(x \in T_{\mathbb{Y}} \leftrightarrow (x \equiv L) \vee \exists_{x_1,x_2}(x_1, x_2 \in T_{\mathbb{Y}} \wedge x \equiv Bx_1x_2))$$

(called inversion property). How can we formally represent cototality of a binary tree? The idea is to define ${}^{\mathrm{co}}T_{\mathbb{Y}}$ as the *largest* set satisfying the equivalence. Formulated differently, cototal ideals are not built from initial objects by construction (synthesized), but rather defined by the property that they can always be destructed (analysed). Therefore we require the "closure property"

$$(3) \quad {}^{\mathrm{co}}T^{-}_{\mathbb{Y}} : \forall_x(x \in {}^{\mathrm{co}}T_{\mathbb{Y}} \to (x \equiv L) \vee \exists_{x_1,x_2}(x_1, x_2 \in {}^{\mathrm{co}}T_{\mathbb{Y}} \wedge x \equiv Bx_1x_2)).$$

A set built by construction steps (synthesis) is meant to be the least set closed under these steps. Similary, a set described by destruction (analysis) is meant to be the largest set closed under destruction. Hence we require the "greatest fixed point" property

$$(4) \quad {}^{\mathrm{co}}T^{+}_{\mathbb{Y}} : \forall_x(x \in X \to (x \equiv L) \vee \exists_{x_1,x_2}(x_1, x_2 \in {}^{\mathrm{co}}T_{\mathbb{Y}} \cup X \wedge x \equiv Bx_1x_2)) \to X \subseteq {}^{\mathrm{co}}T_{\mathbb{Y}}.$$

${}^{\mathrm{co}}T^{+}_{\mathbb{Y}}$ expresses that every competitor X satisfying the closure property is below ${}^{\mathrm{co}}T_{\mathbb{Y}}$.

We also consider binary versions of $T_{\mathbb{Y}}$ and ${}^{\mathrm{co}}T_{\mathbb{Y}}$, called *similarity* $\sim_{\mathbb{Y}}$ and *bisimilarity* $\approx_{\mathbb{Y}}$. The clauses for $\sim_{\mathbb{Y}}$ are

$$(5) \quad \begin{array}{l} (\sim_{\mathbb{Y}})^{+}_0 : L \sim_{\mathbb{Y}} L, \\ (\sim_{\mathbb{Y}})^{+}_1 : \forall_{x_1,x_1'}(x_1 \sim_{\mathbb{Y}} x_1' \to \forall_{x_2,x_2'}(x_2 \sim_{\mathbb{Y}} x_2' \to Bx_1x_2 \sim_{\mathbb{Y}} Bx_1'x_2')) \end{array}$$

and the elimination or least-fixed-point property is

$$(6) \quad \begin{array}{l} \sim^{-}_{\mathbb{Y}} : X(L,L) \to \forall_{x_1,x_2}((x_1 \sim_{\mathbb{Y}} x_2 \wedge Xx_1x_2) \to \\ \qquad\qquad \forall_{x_1',x_2'}((x_1' \sim_{\mathbb{Y}} x_2' \wedge Xx_1'x_2') \to \\ \qquad\qquad X(Bx_1x_2, Bx_1'x_2'))) \to \\ \qquad \sim_{\mathbb{Y}} \subseteq X. \end{array}$$

The elimination (or closure) property for $\approx_{\mathbb{Y}}$ is

$$\begin{aligned}\approx^{-}_{\mathbb{Y}}\colon \forall_{x,x'}(x \approx_{\mathbb{Y}} x' \to ((x \equiv L) \land (x' \equiv L)) \lor \\ \exists_{x_1,x_2,x_1',x_2'}(x_1 \approx_{\mathbb{Y}} x_1' \land x_2 \approx_{\mathbb{Y}} x_2' \land \\ x \equiv Bx_1x_2 \land x' \equiv Bx_1'x_2'))\end{aligned} \tag{7}$$

and the introduction (or greatest-fixed-point or coinduction) property is

(8)

$$\begin{aligned}\approx^{+}_{\mathbb{Y}}\colon \forall_{x,x'}(Xxx' \to ((x \equiv L) \land (x' \equiv L)) \lor \\ \exists_{x_1,x_2,x_1',x_2'}((x_1 \approx_{\mathbb{Y}} x_1' \lor Xx_1x_1') \land (x_2 \approx_{\mathbb{Y}} x_2' \lor Xx_2x_2') \land \\ x \equiv Bx_1x_2 \land x' \equiv Bx_1'x_2')) \to \\ X \subseteq \approx_{\mathbb{Y}}.\end{aligned}$$

We show that $x \approx_{\mathbb{Y}} x'$ implies $x \equiv x'$. Generally we have

Lemma 2.2.1 (Bisimilarity). *For every closed base type bisimilarity implies equality.*

Proof. As an example we prove this for $\mathbb{Y}$. Let a range over tokens for $\mathbb{Y}$. By induction on the height $|a^*|$ of extended tokens a^* we prove that for all ideals x, x' and extended tokens a^* from $a^* \in x$ we can infer $a^* \in x'$. It suffices to consider the case $Ba_1^*a_2^*$. From $x \approx_{\mathbb{Y}} x'$ we obtain by the closure property x_1, x_2, x_1', x_2' with

$$x_1 \approx x_1' \land x_2 \approx x_2' \land x \equiv Bx_1x_2 \land x' \equiv Bx_1'x_2'.$$

Then $a_i^* \in x_i$ (for $i = 1, 2$), and by IH $a_i^* \in x_i'$. Thus $Ba_1^*a_2^* \in x'$. □

From the Bisimilarity Lemma we obtain

Proposition 2.2.2 (Characterization of $\approx_{\mathbb{Y}}$).

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

Proof. "$\to$". By Lemma 2.2.1 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

$$\begin{aligned}(x \equiv L) \lor \exists_{x_1,x_2}((x_1 \in {}^{\mathrm{co}}T_{\mathbb{Y}} \lor \exists_{x_1'}(x_1 \approx x_1')) \land \\ (x_2 \in {}^{\mathrm{co}}T_{\mathbb{Y}} \lor \exists_{x_2'}(x_2 \approx x_2')) \land x \equiv Bx_1x_2).\end{aligned}$$

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

$$(x \equiv L \land x' \equiv L) \lor \exists_{x_1,x_2,x_1',x_2'}(x_1 \approx x_1' \land x_2 \approx x_2' \land x \equiv Bx_1x_2 \land x' \equiv Bx_1'x_2')).$$

In the first case we have $x \equiv L$ 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 $x \equiv Bx_1x_2$, and are done as well.

"←". 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 $X := \{\, x, x' \mid x \in {}^{\mathrm{co}}T_{\mathbb{Y}} \land x \equiv x' \,\}$. It suffices to prove the premise. Fix x, x' with $x \in {}^{\mathrm{co}}T_{\mathbb{Y}} \land x \equiv x'$. The goal is

$$\begin{aligned}(x \equiv L \land x' \equiv L) \lor \exists_{x_1,x_2,x_1',x_2'}((x_1 \approx x_1' \lor (x_1 \in {}^{\mathrm{co}}T_{\mathbb{Y}} \land x_1 \equiv x_1')) \land \\ (x_2 \approx x_2' \lor (x_2 \in {}^{\mathrm{co}}T_{\mathbb{Y}} \land x_2 \equiv x_2')) \land \\ x \equiv Bx_1x_2 \land x' \equiv Bx_1'x_2')).\end{aligned}$$

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

$$(x \equiv L) \lor \exists_{x_1,x_2}(x_1 \in {}^{\mathrm{co}}T_{\mathbb{Y}} \land x_2 \in {}^{\mathrm{co}}T_{\mathbb{Y}} \land x \equiv Bx_1x_2).$$

In the first case we have $x \equiv L$ and are done, since $x \equiv x'$. In the second case we have $x_1, x_2 \in {}^{\mathrm{co}}T_{\mathbb{Y}}$ with $x \equiv Bx_1x_2$. Then we are done as well with $x_1' := x_1$ and $x_2' := x_2$, since again $x \equiv x'$. □

2.2.4. Constructors as continuous functions. Let ι be a closed base type. Every constructor C generates the following ideal in the function space determined by the type of the constructor:

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

where $(\vec{U}, a)$ 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

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

(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.

Lemma 2.2.3 (Constructors are injective and have disjoint ranges). *Let ι be a closed base type 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. □

Lemma (Ideals of a closed base type). *Every non-empty ideal in the information system associated to a closed base type has the form $|r_{\mathrm{C}}|(\vec{x})$ with a constructor* C *and ideals $\vec{x}$.*

Proof. Let z be a non-empty ideal and $\mathrm{C}a_0^*b_0^* \in z$, where for simplicity we assume that C is a binary constructor. Let $x := \{\, a \mid \mathrm{C}a* \in z \,\}$ and $y := \{\, b \mid \mathrm{C}*b \in z \,\}$; clearly x, y are ideals. We claim that $z = |r_{\mathrm{C}}|(x, y)$.

For $\supseteq$ consider $\mathrm{C}a^*b^*$ with $a^* \in x \cup \{*\}$ and $b^* \in y \cup \{*\}$. Then by definition $\{\mathrm{C}a^**, \mathrm{C}*b^*\} \subseteq z$, hence $\mathrm{C}a^*b^* \in z$ by deductive closure. Conversely, notice that an arbitrary element of z must have the form $\mathrm{C}a^*b^*$, because of consistency. Then $\{\mathrm{C}a^**, \mathrm{C}*b^*\} \subseteq z$ again by deductive closure. Hence $a^* \in x \cup \{*\}$ and $b^* \in y \cup \{*\}$, and therefore $\mathrm{C}a^*b^* \in |r_{\mathrm{C}}|(x, y)$. □

It is in this proof that we need entailment to be a relation between finite sets of tokens and single tokens, not just a binary relation between tokens. Information systems with the latter property are called *atomic*.

The information systems $\boldsymbol{C}_\tau$ enjoy the pleasant property of *coherence*, which amounts to the possibility to locate inconsistencies in two-element sets of data objects. Generally, an information system $\boldsymbol{A} = (A, \mathrm{Con}, \vdash)$ is *coherent* if it satisfies: $U \subseteq A$ is consistent if and only if all of its two-element subsets are.

LEMMA 2.2.4. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be information systems. If $\boldsymbol{B}$ is coherent, then so is $\boldsymbol{A} \to \boldsymbol{B}$.*

PROOF. Let $\boldsymbol{A} = (A, \mathrm{Con}_A, \vdash_A)$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$ be information systems, and consider $\{(U_1, b_1), \dots, (U_n, b_n)\} \subseteq \mathrm{Con}_A \times B$. Assume

$$\forall_{1 \le i < j \le n}(\{(U_i, b_i), (U_j, b_j)\} \in \mathrm{Con}).$$

We have to show $\{(U_1, b_1), \dots, (U_n, b_n)\} \in \mathrm{Con}$. Let $I \subseteq \{1, \dots, n\}$ and $\bigcup_{i \in I} U_i \in \mathrm{Con}_A$. We must show $\{\, b_i \mid i \in I \,\} \in \mathrm{Con}_B$. Now since $\boldsymbol{B}$ is coherent by assumption, it suffices to show that $\{b_i, b_j\} \in \mathrm{Con}_B$ for all $i, j \in I$. So let $i, j \in I$. By assumption we have $U_i \cup U_j \in \mathrm{Con}_A$, and hence $\{b_i, b_j\} \in \mathrm{Con}_B$. □

2.3. Terms

We now set up a term language to denote partial continuous functionals. It can be seen as a as a common extension of Gödel's T (1958) and Plotkin's PCF (1977); we call it T^+.

2.3.1. 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

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

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

$$D\vec{P}_i(\vec{y}_i) := M_i \qquad (i = 1, \dots, n \text{ where } n \geq 0) \tag{9}$$

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[2] (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 (9) 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.

[2]A detailed treatment of unification is in Appendix A

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,

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

and similarly for $\mathbb{Y}$ with constructors L (leaf) and B (branch)

$$\begin{array}{ll} (L =_{\mathbb{Y}} L) := \mathtt{tt}, & (Bxy =_{\mathbb{Y}} L) := \mathtt{ff}, \\ (L =_{\mathbb{Y}} Bxy) := \mathtt{ff}, & (Bxy =_{\mathbb{Y}} Bx'y') :=\ (x =_{\mathbb{Y}} x'\ \mathrm{andb}\ y =_{\mathbb{Y}} y'). \end{array}$$

For $\mathbb{L}(\iota)$ with ι a closed base type we define $=_{\mathbb{L}(\iota)} \colon \mathbb{L}(\iota) \to \mathbb{L}(\iota) \to \mathbb{B}$ by

$$\begin{array}{ll} ([] =_{\mathbb{L}(\iota)} []) := \mathtt{tt}, & (a{::}\ell =_{\mathbb{L}(\iota)} []) := \mathtt{ff}, \\ ([] =_{\mathbb{L}(\iota)} a{::}\ell) := \mathtt{ff}, & (a{::}\ell =_{\mathbb{L}(\iota)} a'{::}\ell') :=\ (a =_\iota a'\ \mathrm{andb}\ \ell =_{\mathbb{L}(\iota)} \ell'). \end{array}$$

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

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

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

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

Multiplication (again written infix) is defined using addition by

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

Similarly we can define all primitive recursive functions.

Up to now we have mainly 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 closed base type $\mathbb{S}(\mathbb{D})$ of streams defined by the single constructor type

$$\mathrm{SCons} \colon \mathbb{D} \to \mathbb{S}(\mathbb{D}) \to \mathbb{S}(\mathbb{D}).$$

We write $\mathrm{C}_d u$ or $d :: u$ for $\mathrm{SCons}(d, u)$. This base type 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 $\{\, \mathrm{C}_d^n(*) \mid n \geq 1 \,\}$.

We define Map of type $(\tau \to \sigma) \to \mathbb{S}(\tau) \to \mathbb{S}(\sigma)$ mapping its function argument $h\colon \tau \to \sigma$ over a stream u of type $\mathbb{S}(\tau)$ by the computation rule

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

As an example of how to define standard arithmetical functions in T^+ we consider the quotient-and-remainder function qr of type $\mathbb{N} \to \mathbb{N} \to \mathbb{N} \times \mathbb{N}$. Its defining equarions are

$$\begin{aligned} \mathrm{qr}(0, m) &:= (0, 0), \\ \mathrm{qr}(n+1, m) &:= \begin{cases} (q, r+1) & \text{if } r+1 < m, \\ (q+1, 0) & \text{else} \end{cases} \qquad \text{with } (q, r) := \mathrm{qr}(n, m). \end{aligned}$$

Lemma 2.3.1 (NatQRProp). *Given n, m with $0 < m$. Let $(q, r) := \mathrm{qr}(n, m)$. Then $n = q * m + r$ and $r < m$.*

Proof. By induction on n. The base case is clear. In the step we distinguish cases. *Case* $r + 1 < m$. Then $\mathrm{qr}(n+1, m) = (q, r+1)$. The claim follows by induction hypothesis. *Case* $m \leq r + 1$. Then $m = r + 1$ since $r < m$ (by induction hypothesis) and $m \leq r + 1$ (by case assumption). By definition we have $\mathrm{qr}(n+1, m) = (q+1, 0)$. We obtain

$$n + 1 = q * m + r + 1 = q * m + m = (q + 1) * m + 0. \qquad \square$$

2.3.2. 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 base type 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

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

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, $\mathcal{R}_{\mathbb{N}}^{\mathbb{N}} n m \lambda_{n,l}(Sl)$ 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 a list of constructor types

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

DEFINITION (Type of $\mathcal{R}^\tau_\iota$). Let a base type ι be given by a list of constructor types $((\rho_{i\nu}(\xi))_{\nu<n_i} \to \xi)_{i<k}$. Let τ be a type. 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 a base type ι given by a list of constructor types $((\rho_{i\nu}(\xi))_{\nu<n_i} \to \xi)_{i<k}$ and a 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 is $[\texttt{if}\ m\ 0\ \lambda_n n]$ (which is often written in the form $[\mathbf{case}\ m\ \mathbf{of}\ 0 \mid \lambda_n n]$).

General recursion with respect to a measure. In practice it often happens that one needs to recur to an argument which is not an immediate component of the present constructor object; this is not allowed in structural recursion. Of course, in order to ensure that the recursion terminates we have to assume that the recurrence is w.r.t. a given well-founded set; for simplicity we restrict ourselves to the algebra $\mathbb{N}$. However, we do allow that the recurrence is with respect to a measure function μ, with values in $\mathbb{N}$. The operator $\mathcal{F}$ of *general recursion* then is defined by

$$\mathcal{F}\mu x G = Gx(\lambda_y[\mathbf{if}\ \mu y < \mu x\ \mathbf{then}\ \mathcal{F}\mu y G\ \mathbf{else}\ \varepsilon]), \tag{10}$$

where ε denotes a canonical inhabitant of the range. We leave it as an exercise to prove that $\mathcal{F}$ is definable from an appropriate structural recursion operator.

As an example for the use of $\mathcal{F}$ we define a function NatToPos converting a natural number ≥ 1 written in unary (i.e., built from the constructors 0 and S) into the same natural number written in binary (i.e., built from the constructors 1, S_0 and S_1). This uses the auxiliary functions $\mathrm{Even}\colon \mathbb{N} \to \mathbb{B}$ defined by

$$\begin{aligned} \mathrm{Even}(0) &:= \mathtt{tt}, \\ \mathrm{Even}(S(0)) &:= \mathtt{ff}, \\ \mathrm{Even}(S(S(n))) &:= \mathrm{Even}(n) \end{aligned}$$

and $\mathrm{Half}\colon \mathbb{N} \to \mathbb{N}$ defined by

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

Then $\mathrm{NatToPos}\colon \mathbb{N} \to \mathbb{P}$ is defined by

$$\mathrm{NatToPos}(n) = \mathcal{F}(\mathrm{id}, n, G)$$

with $\mathrm{id}(n) = n$ and $G\colon \mathbb{N} \to (\mathbb{N} \to \mathbb{P}) \to \mathbb{P}$ defined by

$$G(n, f) = \begin{cases} S_0(f(\mathrm{Half}(n))) & \text{if } \mathrm{Even}(n), \\ 1 & \text{if } n = S(0), \\ S_1(f(\mathrm{Half}(n))) & \text{otherwise.} \end{cases}$$

2.3.3. 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 let a base type ι be given by a list of constructor types

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

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

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

This way of subsuming the types of all constructors of a base type into a single type suggests the following definition of the *destructor* $\mathcal{D}_\iota$ of a base type ι.

Definition (Destructor). For a base type ι given by a list of constructor types $((\rho_{i\nu}(\xi))_{\nu<n_i} \to \xi)_{i<k}$ we define the type of the destructor $\mathcal{D}_\iota$ to be

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

The computation rules for $\mathcal{D}_\iota$ disassemble a constructor-built pattern into its parts. For instance, the computation rules for $\mathcal{D}_\mathbb{N}$ of type $\mathbb{N} \to \mathrm{uysum}(\mathbb{N})$

(or $\mathbb{U}+\mathbb{N}$) are

$$\begin{aligned}\mathcal{D}_{\mathbb{N}}(0) &:= \mathrm{DummyL},\\ \mathcal{D}_{\mathbb{N}}(S(n)) &:= \mathrm{Inr}(n).\end{aligned}$$

The computation rules for $\mathcal{D}_{\mathbb{P}}$ of type $\mathbb{P}\to$ uysum$(\mathbb{P}+\mathbb{P})$ (or $\mathbb{U}+(\mathbb{P}+\mathbb{P})$) are

$$\begin{aligned}\mathcal{D}_{\mathbb{P}}(1) &:= \mathrm{DummyL},\\ \mathcal{D}_{\mathbb{P}}(S_0(p)) &:= \mathrm{Inr}(\mathrm{InL}^{\mathbb{P}\to\mathbb{P}+\mathbb{P}}(p)),\\ \mathcal{D}_{\mathbb{P}}(S_1(p)) &:= \mathrm{Inr}(\mathrm{InR}^{\mathbb{P}\to\mathbb{P}+\mathbb{P}}(p)).\end{aligned}$$

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

$$\tau\to\Big(\tau\to\sum_{i<k}\prod_{\nu<n_i}\rho_{i\nu}(\iota+\tau)\Big)\to\iota. \tag{11}$$

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 base type.

$$\begin{aligned}{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{B}}xf &:= [\lambda\_\,\mathtt{tt},\lambda\_\,\mathtt{ff}](fx),\\ {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{N}}xf &:= [\lambda\_\,0,\lambda_y(S([\mathrm{id}^{\mathbb{N}\to\mathbb{N}},P_{\mathbb{N}}]y))](fx),\\ {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{P}}xf &:= [\lambda\_\,1,\lambda_y(S_0([\mathrm{id},P_{\mathbb{P}}]y)),\lambda_y(S_1([\mathrm{id},P_{\mathbb{P}}]y))](fx),\\ {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{Y}}xf &:= [\lambda\_L,\lambda_{y_0,y_1}(B([\mathrm{id},P_{\mathbb{Y}}]y_0)([\mathrm{id},P_{\mathbb{Y}}]y_1))](fx),\\ {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{L}(\rho)}xf &:= [\lambda\_[],\lambda_{y_0,y_1}(y_0::[\mathrm{id},P_{\mathbb{L}(\rho)}]y_1)](fx),\\ {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbb{S}(\rho)}xf &:= (fx)_0::[\mathrm{id},P_{\mathbb{S}(\rho)}](fx)_1\end{aligned}$$

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 (11) is

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

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

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

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

and $P := \lambda_x({}^{\mathrm{co}}\mathcal{R}^\tau_\iota x f)$ 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

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

$$ {}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{Y}} x f = \begin{cases} L & \text{if } fx = \mathrm{DummyL}^{\mathbb{U}+(\mathbb{Y}+\tau)\times(\mathbb{Y}+\tau)} \\ Buv & \text{if } fx = \mathrm{Inr}\langle \mathrm{InL}^{\mathbb{Y}\to\mathbb{Y}+\tau} u, \mathrm{InL}^{\mathbb{Y}\to\mathbb{Y}+\tau} v\rangle \\ Bu({}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{Y}} y f) & \text{if } fx = \mathrm{Inr}\langle \mathrm{InL}^{\mathbb{Y}\to\mathbb{Y}+\tau} u, \mathrm{InR}^{\tau\to\mathbb{Y}+\tau} y\rangle \\ B({}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{Y}} y f)v & \text{if } fx = \mathrm{Inr}\langle \mathrm{InR}^{\tau\to\mathbb{Y}+\tau} y, \mathrm{InL}^{\mathbb{Y}\to\mathbb{Y}+\tau} v\rangle \\ B({}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{Y}} y f)({}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbb{Y}} z f) & \text{if } fx = \mathrm{Inr}\langle \mathrm{InR}^{\tau\to\mathbb{Y}+\tau} y, \mathrm{InR}^{\tau\to\mathbb{Y}+\tau} z\rangle \end{cases} $$

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

Recall that Map of type $(\tau \to \sigma) \to \mathbb{S}(\tau) \to \mathbb{S}(\sigma)$ maps its function argument $h \colon \tau \to \sigma$ over a stream u of type $\mathbb{S}(\tau)$ (see Section 2.3.1, page 24). 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 type $\mathbb{Y}$ 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.

2.4. Denotational semantics

We now set up a connection between the model $(|\mathbf{A}_\rho|)_\rho$ of partial continuous functionals described in Section 2.1 and the term system T^+ from Section 2.3. 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 $D\vec{P}(\vec{y}) = M$.

This "denotational semantics" has good properties; however, we do not carry out the proofs here but rather refer to the literature. Some of these proofs are given in Appendix B. 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.