11 Partial Combinatory Algebras

So far we have considered models of intuitionistic logic based on topology and more generally on complete Heyting algebras. The idea behind realizability is to instead work with an abstract notion of computation, called *partial combinatory algebra* (pca), which is the topic of this section.

11.1 Some notation and terminology for partial functions

The "computable functions" of a pca will be in particular partial functions, so we will first define some notation and terminology that will help us to deal with partial functions better.

Definition 11.1. Let X and Y be sets. A *partial function* from X to Y is a subset f of $X \times Y$ such that whenever $(x, y) \in f$ and $(x, y') \in f$ we have $y = y'$. We write $f : X \rightharpoonup Y$.

We write $f(x) \downarrow$ to mean that there exists $y \in Y$ (necessarily unique) such that $(x, y) \in f$. We say f *is defined at* x. We write $f(x) = y$ to mean $(x, y) \in f$. Note in particular that writing $f(x) = y$ implicitly implies $f(x) \downarrow$. However, we often write out $f(x) \downarrow$ explicitly anyway to draw attention to the fact that f is defined at x.

Suppose that $f : X \rightharpoonup Y$ and $g : Z \rightharpoonup Y$. For $x \in X$ and $z \in Z$, we write $f(x) \simeq g(z)$ to mean that $f(x) \downarrow$ if and only if $g(z) \downarrow$, and that if they are defined, then $f(x) = g(z)$.

In particular, if X is a set with one element, say $*$, then partial functions $X \rightharpoonup Y$ are just subsets of singletons of Y. In this case $f(*) \downarrow$ means the subsingleton is inhabited, and $f(*) = y$ means the subsingleton contains the element y (and so is an actual singleton).

Also note that we have the following propositions.

Proposition 11.2. *Any function $X \to Y$ is in particular a partial function.*

Proposition 11.3. *Given partial functions $f : X \rightharpoonup Y$ and $g : Y \rightharpoonup Z$, we define the composition $g \circ f$ to be the partial function $X \rightharpoonup Z$, which is defined at x if and only if $f(x) \downarrow$ and $g(f(x)) \downarrow$, and in this case $(g \circ f)(x) = g(f(x))$.*

11.2 Partial applicative structures

Before defining partial combinatory algebras, we will first define a weaker notion, partial applicative structure.

Definition 11.4. A *partial applicative structure* (pas) is a set $\mathcal{A}$, together with a partial binary operation $\cdot : \mathcal{A} \times \mathcal{A} \rightharpoonup \mathcal{A}$.

Definition 11.5. Let $\mathcal{A}$ be a pas. We say a partial function $f : \mathcal{A} \rightharpoonup \mathcal{A}$ is *representable* if there exists $a \in \mathcal{A}$ such that for all $b \in \mathcal{A}$ we have $f(b) \simeq a \cdot b$.

The intuitive idea behind representable partial functions is *programs-as-data*. The elements of $\mathcal{A}$ are mathematical objects that we wish to discuss (e.g. we can take $\mathcal{A} = \mathbb{N}$ or $\mathcal{A} = \mathbb{N}^{\mathbb{N}}$). We think of representable functions as programs that operate on this data. By definition, every representable function can be "coded" or "represented" as an element of $\mathcal{A}$ itself.

Example 11.6. Any total binary operation is in particular a partial binary operation. Applying this to $\mathbb{N}$ with the binary operation $+$, we get that the representable partial function $\mathbb{N} \rightharpoonup \mathbb{N}$ are precisely the total functions of the form $\lambda m.m + n$ for each $n \in \mathbb{N}$.

Example 11.7. Note that we can define an injective function i from closed terms of $\mathbf{HA}_\omega$ of sort $N \to N$ to $\mathbb{N}$. Given a closed term t of sort $N \to N$, we write $\ulcorner t \urcorner$ for the corresponding element of $\mathbb{N}$. Define the partial operation by setting $n \cdot m = k$ when $n = \ulcorner t \urcorner$ and $\mathbf{HA}_\omega \vdash tm = k$ (and otherwise it is undefined). An encoding of terms as natural numbers in this way is often referred to as a *Gödelnumbering*

Definition 11.8. Let $\mathcal{A}$ be a partial applicative structure and assume that we have a supply of free variables $x_1, x_2, \ldots$. The set of $\mathcal{A}$-*terms* is defined inductively as follows.

1. Every free variable x_i is an $\mathcal{A}$-term
2. Every element of $\mathcal{A}$ is an $\mathcal{A}$-term
3. If s and t are $\mathcal{A}$-terms, then so is $s \cdot t$

Definition 11.9. Given an $\mathcal{A}$-term t and a list of free variables $x_1, \ldots, x_n$ including all those that occur free in t, we define a partial function $|t| : \mathcal{A}^n \rightharpoonup \mathcal{A}$ by induction on terms.

1. $|x_i|$ is a total function, defined as $|x_i|(a_1, \ldots, a_n) := a_i$
2. $|a|$ is a total function, defined as $|a|(a_1, \ldots, a_n) := a$
3. We define $|s \cdot t|(a_1, \ldots, a_n)$ to be defined only when $|s|(a_1, \ldots, a_n)$ and $|t|(a_1, \ldots, a_n)$ are defined, and in this case

$$|s \cdot t|(a_1, \ldots, a_n) \simeq |s|(a_1, \ldots, a_n) \cdot |t|(a_1, \ldots, a_n)$$

In particular if an $\mathcal{A}$-term is closed (contains no free variables), then it defines a partial function on a set with one element.

We will deal with $\mathcal{A}$-terms quite a lot, so we will use some notational conventions to make them easier to write:

1. We will often omit the binary operator, writing $s \cdot t$ simply as st.
2. Write $\cdot$ left associatively. That is, we write rst to mean $(rs)t$.
3. We will often write $|t|$ simply as t, omitting the bars.

4. We write substitution into $\mathcal{A}$-terms the same way as substitution of terms in intuitionistic logic (i.e. $t[x/s]$ where s is an $\mathcal{A}$-term).

5. If t is a closed term and $t\downarrow$, then there is a unique element a of $\mathcal{A}$ such that $t \simeq a$. We say t *denotes* a.

11.3 Partial combinatory algebras

Definition 11.10. A *partial combinatory algebra* (pca) is a pas $(\mathcal{A}, \cdot)$ such that there are $\mathbf{k}, \mathbf{s} \in \mathcal{A}$ such that for all $a, b, c \in \mathcal{A}$,

1. $\mathbf{k}a\downarrow$ and $\mathbf{k}ab = a$ (in particular $\mathbf{k}ab$ is always defined).

2. $\mathbf{s}a\downarrow$, $\mathbf{s}ab\downarrow$ and
$$\mathbf{s}abc \simeq ac(bc)$$

We say a partial combinatory algebra is *non-trivial* if $\mathbf{s} \neq \mathbf{k}$.

Similarly to the definition of $\mathbf{HA}_\omega$, we can use $\mathbf{k}$ to generate constant functions and $\mathbf{s}$ to define the "dependent composition" of two functions. In particular, we have the following proposition.

Proposition 11.11. *Let $\mathcal{A}$ be a pca. Then every constant function $\mathcal{A} \to \mathcal{A}$ is representable. If partial functions $f : \mathcal{A} \rightharpoonup \mathcal{A}$ and $g : \mathcal{A} \rightharpoonup \mathcal{A}$ are representable, then so is $g \circ f$. The identity function $\mathcal{A} \to \mathcal{A}$ is also representable (by $\mathbf{i} := \mathbf{skk}$).*

Similarly to in $\mathbf{HA}_\omega$ we can prove a λ-abstraction lemma. Since we are now working with partial functions, we phrase the lemma a bit differently. It is particularly important that $\lambda x.t$ is always a defined term, even if (for example) t is already closed and not defined.

Definition 11.12. Let $\mathcal{A}$ be a pca, and t an $\mathcal{A}$-term whose only free variable is x or with no free variables at all. Then there is a closed $\mathcal{A}$-term $\lambda x.t$ such that $\lambda x.t\downarrow$ and for all $a \in \mathcal{A}$ we have

$$(\lambda x.t)a \simeq t[x/a]$$

Proof. We define $\lambda x.t$ and check that it works by induction on terms.

1. If $t = x$, we define $\lambda x.t$ to be $\mathbf{i}$

2. If $t = a$ for $a \in \mathcal{A}$, we define $\lambda x.t$ to be $\mathbf{k}a$

3. If $t = r \cdot s$, we define $\lambda x.t$ to be $\mathbf{s}(\lambda x.r)(\lambda x.s)$.

□

We can use λ-abstraction to define $\mathbf{y}$ combinators. This construction can only be carried out in "untyped" settings, such as in pcas. It does not work for $\mathbf{HA}_\omega$, for example. This is not the most common definition of $\mathbf{y}$-combinator. It is however the most appropriate definition when working with a partially defined application operator, and therefore the only definition we will consider in this course.

Theorem 11.13. *For any pca, $\mathcal{A}$, there is an element $\mathbf{y} \in \mathcal{A}$ with the following properties. For all $a \in \mathcal{A}$, we have $\mathbf{y}a\downarrow$, and for all $b \in \mathcal{A}$, we have*

$$\mathbf{y}ab \simeq a(\mathbf{y}a)b$$

Proof. We first define

$$t := \lambda x.\lambda y.\lambda z.y(xx)z$$

We then define $\mathbf{y}$ to be the closed term tt.

First note that we can calculate

$$\begin{aligned}\mathbf{y} &= tt \\ &= (\lambda x.\lambda y.\lambda z.y(xx)z)t \\ &\simeq \lambda y.\lambda z.y(tt)z\end{aligned}$$

In particular, we can see that $\mathbf{y}$ is defined, since λ-abstraction is always defined and so it does denote an element of $\mathcal{A}$.

Now given $a \in \mathcal{A}$, we have

$$\mathbf{y}a \simeq \lambda z.a(tt)z$$

Again using the fact that λ-abstraction is always defined, we can see in particular that $\mathbf{y}a\downarrow$.

Finally for any $b \in \mathcal{A}$, we have

$$\begin{aligned}\mathbf{y}ab &\simeq a(tt)b \\ &= a(\mathbf{y}a)b\end{aligned}$$

□

Neither of the two examples of partial applicative structures we have seen are pcas. For example 11.6 it is clear that constant functions are not representable, for example. It is harder to see why example 11.7 is not a pca. For now we just remark that the main difficultly is that the $\mathbf{s}$ combinator of $\mathbf{HA}_\omega$ is a typed operation of sort $(N \to N \to N) \to (N \to N) \to N \to N$. However, to get a pca we would need a term of sort $N \to N$ that takes Gödelnumbers as input, which turns out to be more difficult.

11.4 Two examples of term pcas

To get some non trivial examples of pcas, we will use *term models*. We first define what we mean by term.

Definition 11.14. We define the set $\mathcal{T}$ of *pca-terms* to be inductively defined as follows:

1. $\mathbf{k}$ is a pca-term
2. $\mathbf{s}$ is a pca-term

3. If s and t are pca-terms, then so is $s \cdot t$

We define our first term model as follows. Let $\sim$ be the smallest equivalence relation on $\mathcal{T}$ satisfying the following conditions for all $r, s, t \in \mathcal{T}$:

1. $\mathbf{k}rs \sim r$
2. $\mathbf{s}rst \sim rt(st)$
3. If $r \sim s$, then $r \cdot t \sim s \cdot t$
4. If $s \sim t$, then $r \cdot s \sim r \cdot t$

Example 11.15. We can give $\mathcal{T}/\sim$ the structure of a pca, where we define $[r] \cdot [s] := [r \cdot s]$, $\mathbf{k} := [\mathbf{k}]$ and $\mathbf{s} := [\mathbf{s}]$.

The preceding example has the advantage of being simple. However, it is difficult to say anything concrete about it, even to show it is non trivial.

We therefore consider a second term model, which has a bit more complicated definition, but is easier to describe concretely. It also has the advantage of being easy to implement on electronic computers, especially so with the help of modern programming languages with a built in notion of inductively defined type, such as Haskell.

We first define a subset of terms that we refer to as normal.

Definition 11.16. We say a term t is *reducible* if it satisfies any of the conditions below:

1. $t = \mathbf{k}rs$ for some terms r and s
2. $t = \mathbf{s}rsu$ for some terms r, s and t
3. $t = rs$, for some terms r and s where either r or s is reducible

We say a term t is *normal* if it is not reducible. We write the set of all normal terms as $\mathcal{T}_0$.

Definition 11.17. We define a ternary relation on terms, natural numbers and normal terms. Given a term t, number n and normal form s, we will write the relation as $t \to_n s$ and say t *reduces to* s *at stage* n. We define the relation as the smallest one satisfying the conditions below.

1. If t is normal, then $t \to_n t$.
2. If either t or r is reducible and $t \to_n t'$ and $r \to_n r'$ and $t'r' \to_n v$, then $tr \to_n v$ (note that $t'r'$ is either normal or one of the two remaining cases below).
3. If t and r are normal, then $\mathbf{k}tr \to_n t$.
4. If t, r, u and v are normal, then $\mathbf{s}tru \to_{n+1} v$ whenever $tu(ru) \to_n v$.

Note that we have chosen the definition to ensure we have the following lemma.

Lemma 11.18. *Reducibility at n satisfies the following statements.*

1. *If $t \to_n t'$, then $t \to_{n+1} t'$.*
2. *If $t \to_n t'$ and $t \to_n t''$, then $t' = t''$.*
3. *If t is normal, then $t \to_n t$ for all n.*

We can now give $\mathcal{T}_0$ the structure of a pas by defining $t \cdot s$ to be r if there exists n such that $t \cdot s \to_n r$ (and $t \cdot s$ is undefined if there is no such n).

Theorem 11.19. *$\mathcal{T}_0$ with the above application operator is a pca.*

Proof. By lemma 11.18 we can see that the above definition does give a well defined partial binary operator $\cdot$. It remains to check the axioms for $\mathbf{k}$ and $\mathbf{s}$. First note that if r and t are normal, then so are $\mathbf{k}r$, and $\mathbf{s}rt$, so we do have $\mathbf{k}r \downarrow$ and $\mathbf{s}rt \downarrow$. We clearly have $\mathbf{k}rt = r$. It only remains to check $\mathbf{s}rtu \simeq ru(tu)$ for normal forms r, t, u. However, by definition $\mathbf{s}rtu \to_{n+1} v$ for some normal form v if and only if $ru(tu) \to_n v$. We can see from this that $\mathbf{s}rtu \downarrow$ if and only if $ru(tu) \downarrow$, and when they are defined we clearly have $\mathbf{s}rtu = ru(tu)$, as required. □

11.5 Extended pcas and computable functions

We now have a non trivial example of a pca, giving us some kind of notion of computation. However, at the moment the only thing we can "compute" is normal terms. For this reason, we also consider a variant of the definition of partial combinatory algebra that allows us to encode numbers as a subset of the pca.

Definition 11.20. An extended pca, or pca$^+$ is a partial combinatory algebra $(\mathcal{A}, \cdot)$ with the additional constants $\mathbf{p}$, $\mathbf{p}_0$, $\mathbf{p}_1$, 0, S, P, and $\mathbf{d}$, satisfying the axioms below. Given $n \in \mathbb{N}$ we will write $\underline{n}$ for the element of $\mathcal{A}$ defined recursively by $\underline{0} := 0$ and $\underline{n+1} := S\underline{n}$.

1. For all $a, b \in \mathcal{A}$, $\mathbf{p}ab \downarrow$, $\mathbf{p}_0(\mathbf{p}ab) \simeq a$ and $\mathbf{p}_1(\mathbf{p}ab) \simeq b$
2. $\underline{n+1} \neq \underline{0}$ for any $n \in \mathbb{N}$
3. For all $n \in \mathbb{N}$, $P(S\underline{n}) = \underline{n}$
4. For all $n, m \in \mathbb{N}$ and all $a, b \in \mathcal{A}$, $\mathbf{d}\underline{nm}ab = a$ if $n = m$ and $\mathbf{d}\underline{nm}ab = b$ if $n \neq m$

In fact, any non trivial pca can be made into an extended pca. However, there are often many ways to choose the pca$^+$ structure, and it can be useful to ensure the structure is simple to describe explicitly.

To get a term model with a simple description of the pca$^+$ structure we adjust the definition of $\mathcal{T}_0$ from the previous section by explicitly adding new constants. Namely we define terms as follows:

Definition 11.21. We define the set $\mathcal{T}^+$ of *pca$^+$-terms* to be inductively defined as follows:

1. $\mathbf{k}$ is a pca$^+$-term
2. $\mathbf{s}$ is a pca$^+$-term
3. If s and t are pca$^+$-terms, then so is $s \cdot t$
4. 0 is a pca$^+$-term
5. S is a pca$^+$-term
6. P is a pca$^+$-term
7. $\mathbf{d}$ is pca$^+$-term

Definition 11.22. We say a term t is *reducible* if it satisfies any of the conditions below:

1. $t = \mathbf{k}rs$ for some terms r and s
2. $t = \mathbf{s}rsu$ for some terms r, s and t
3. $t = rs$, for some terms r and s where either r or s is reducible
4. $t = \mathbf{p}_0(\mathbf{p}rs)$ or $t = \mathbf{p}_1(\mathbf{p}rs)$ for some terms r and s
5. $t = P(S\underline{n})$ for some $n \in \mathbb{N}$
6. $t = \mathbf{d}\underline{nm}rs$ for some $n, m \in NN$ and terms r and s

We say a term t is *normal* if it is not reducible. We write the set of all normal terms as $\mathcal{T}_0^+$.

We adjust the definition of reducibility at stage n (definition 11.17) by adding the following clauses:

1. If t and s are normal terms, then $\mathbf{p}_0(\mathbf{p}ts) \to_n t$ and $\mathbf{p}_1(\mathbf{p}ts) \to_n s$
2. If $m \in \mathbb{N}$, then $P(S\underline{m}) \to_n \underline{m}$
3. If $m, l \in \mathbb{N}$ and t, s are normal terms, then $\mathbf{d}\underline{ml}ts \to_n t$ if $m = l$ and $\mathbf{d}\underline{ml}ts \to_n s$ if $m \neq l$

As before, for normal terms t and s, we define $t \cdot s$ to be r if there exists $n \in \mathbb{N}$ such that $t \cdot s \to_n r$ (and $t \cdot s$ is undefined if there is no such n). As before, this gives us a pca, and we can make this into an extended pca by interpreting each constant as "itself."

Definition 11.23. We say a partial function $\mathbb{N} \rightharpoonup \mathbb{N}$ is *computable* relative to a pca$^+$ $(\mathcal{A}, \cdot, \mathbf{s}, \mathbf{k}, \mathbf{p}, \mathbf{p}_0, \mathbf{p}_1, 0, S, P, \mathbf{d})$ if there exists $a \in \mathcal{A}$ such that the following holds. For all $n \in \mathbb{N}$, if $f(n) \downarrow$, then $t\underline{n} \downarrow$ and $t\underline{n} = \underline{f(n)}$, and whenever there exists $m \in \mathbb{N}$ such that $t\underline{n} = \underline{m}$, we have $f(n) \downarrow$ with $\overline{f(n)} = m$.

In fact if a partial function $\mathbb{N} \rightharpoonup \mathbb{N}$ is computable relative to $\mathcal{T}_0^+$, then it is computable relative to *any* extended pca, justifying the following definition.

Definition 11.24. We refer to partial functions $\mathbb{N} \rightharpoonup \mathbb{N}$ that are computable relative to $\mathcal{T}_0^+$ simply as *computable*.

The class of computable partial functions can be defined many different ways. This is the main definition we will see in this course, but in other courses you may have seen definitions in terms abstract versions of mechanical or electronic computers, such as Turing machines or register machines, definitions in terms of definability in the λ-calculus or combinatory logic, or the definition of recursive functions.