A Notion of a Computational Step for Partial Combinatory Algebras

Nathanael L. Ackerman[1] and Cameron E. Freer[2]

[1] Department of Mathematics
Harvard University
nate@math.harvard.edu
[2] Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
freer@math.mit.edu

Abstract. Working within the general formalism of a partial combinatory algebra (or PCA), we introduce and develop the notion of a *step algebra*, which enables us to work with individual computational steps, even in very general and abstract computational settings. We show that every partial applicative structure is the closure of a step algebra obtained by repeated application, and identify conditions under which this closure yields a PCA.

Keywords: partial combinatory algebra, step algebra, computational step

1 Introduction

A key feature of a robust notion of computation is having analogous notions of *efficient* computations. More precisely, given a definition of objects that are computable (given unlimited resources such as time, space, or entropy), one would like corresponding definitions of those that are efficiently computable (given bounds on such resources).

As the notion of computation has been generalized in many directions, the related notions of efficiency have not always followed. One important generalization of computation is that given by partial combinatory algebras. Here our goal is to provide one such corresponding notion of *a single step of a computation*, by introducing *step algebras*. Step algebras attempt to describe, using a similar formalism, what it means to carry out one step of a computation; we believe that they may be useful in the analysis of efficient computation in this general context, as well as in other applications, as we will describe.

1.1 Partial Combinatory Algebras

The class of *partial combinatory algebras* (PCAs) provides a fundamental formulation of one notion of abstract computation. PCAs generalize the combinatorial

calculi of Schönfinkel [Sch24] and Curry [Cur29], and have connections to realizability, topos theory, and higher-type computation. For an introduction to PCAs, see van Oosten [vO08] or Longley [Lon95].

PCAs are flexible and general enough to support many of the standard operations and techniques of (generalized) computation, and despite the austerity of their definition, each PCA will contain a realization of every partial computable function. As has been shown over the years, many of the natural models of computation are PCAs. For example, partial computable functions relative to any fixed oracle, partial continuous functions $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ as in Kleene's higher-type computability, nonstandard models of Peano arithmetic, and certain Scott domains all form PCAs (see, e.g., [vO08, §1.4]), as do the partial α-recursive functions for a given admissible ordinal α.

Furthermore, by work of van Oosten [vO06], there is a notion of reducibility between PCAs extending the notion of Turing reducibility arising from ordinary computation with oracles. Also, by augmenting PCAs with limiting operations, Akama [Aka04] has shown how they can interpret infinitary λ-calculi.

Realizability was initially used as a tool to study concrete models of intuitionistic theories. As a result, the use of PCAs has been expanded to give models of intuitionistic set theory, as in work on realizability toposes (generalizing the effective topos; see, e.g, [vO08]) and, more recently, in work by Rathjen [Rat06].

In all these facets of computation, PCAs provide a clean and powerful generalization of the notion of computation. However, the elements of a PCA represent an entire computation, and PCAs generally lack a notion of a single step of a computation. In this paper we aim to provide one such notion.

1.2 Other Approaches to Abstract Algorithmic Computations

One approach, other than PCAs, to abstract algorithmic computation is the *finite algorithmic procedure* of H. Friedman [Fri71]; for details and related approaches, including its relationship to recursion in higher types, see Fenstad [Fen80, §0.1]. In such procedures, there is a natural notion of a step provided by the ordered list of instructions.

Another, more abstract, attempt to capture the notion of computational step is suggested by Moschovakis' *recursors* [Mos01]. In this setting, the result of an abstract computation is the least fixed point of a continuous operator on a complete lattice. As this least fixed point can be obtained as the supremum of iteratively applying this continuous operator to the minimal element of the lattice, one might consider a computational step to be a single application of this continuous operator.

Still another approach is that of Gurevich's *abstract state machines* [Gur00]. Within this formalism, the notion of a single step of an algorithm has also been an important concept [BDG09].

These and other approaches do provide useful analyses of the notion of a single computation step. Here our goal is analogous but is in the more general setting of PCAs, where one is never explicitly handed a list of fine-grained com-

putational instructions, and must instead treat each potentially powerful and long (even nonhalting) subcomputation as a black box.

1.3 Outline of the Extended Abstract

We begin by defining PCAs and recalling their key properties. In the following section we introduce *step algebras*, the main notion of this abstract. A PCA always gives rise to a step algebra (by considering application to be a single step), but additional hypotheses on a step algebra are needed to ensure that the closure under repeated application is itself a PCA.

In the most general cases, a step algebra gives us little handle on its closure. Therefore we consider additional computable operations such as pairing, the use of registers, and serial application. These operations lead us to conditions under which a step algebra yields a PCA (by closure under repeated application). On the other hand, we conjecture that every PCA (up to isomorphism) comes from a suitable step algebra in this way.

Finally, we briefly discuss potential extensions of this work, including a generalization of computational complexity to the setting of PCAs, and an analysis of reversible computations in PCAs.

2 Preliminaries

Before proceeding to step algebras, we briefly recall the definition and key properties of partial applicative structures and partial combinatory algebras. For many more details, see [vO08] or [Lon95].

Definition 1. *Suppose A is a nonempty set and $\circ : A \times A \to A$ is a partial map. We say that $\mathbb{A} = (A, \circ)$ is a* **partial applicative structure** *(PAS). We write* PAS *to denote the class of partial applicative structures.*

This map $\circ$ is often called application. When the map $\circ$ is total, we say that $\mathbb{A}$ is a *total PAS*. When there is no risk of confusion (e.g., from application in another PAS), we will write ab to denote $a \circ b$. Furthermore, we adopt the standard convention of association to the left, whereby abc denotes $(ab)c$ (but not $a(bc)$ in general).

Given an infinite set of variables, the set of terms over a PAS $\mathbb{A} = (A, \circ)$ is the least set containing these variables and all elements of A that is closed under application. For a closed term t (i.e., without variables) and element $a \in A$, we write $t \downarrow a$, and say that term t denotes element a, when a is the result of repeated reduction of subterms of t. We write $t\downarrow$, and say that t denotes, when there is some a such that $t \downarrow a$. For closed terms t, s, the expression $t = s$ means that they denote the same value, and $t \simeq s$ means that if either t or s denotes, then $t = s$. This notation extends to non-closed terms (and means the corresponding expression of closed terms for every substitution instance).

Definition 2. *Let $\mathbb{A} = (A, \circ)$ be a PAS. We say that $\mathbb{A}$ is combinatorially complete when for every $n \in \mathbb{N}$ and any term $t(x_1, \ldots, x_{n+1})$, there is an element $a \in A$ such that for all $a_1, \ldots, a_{n+1} \in A$, we have $aa_1 \cdots a_n\downarrow$ and*

$$aa_1 \cdots a_{n+1} \simeq t(a_1, \ldots, a_{n+1}).$$

A **partial combinatory algebra** *(PCA) is a combinatorially complete PAS.*

The following lemma is standard.

Lemma 1. *Let $\mathbb{A} = (A, \circ)$ be a PAS. Then $\mathbb{A}$ is a PCA if and only if there are elements $S, K \in A$ satisfying*

- $Kab = a$,
- $Sab\downarrow$, *and*
- $Sabc \simeq ac(bc)$

for all $a, b, c \in A$.

Using the S and K combinators, many convenient objects or methods can be obtained in arbitrary PCAs, including pairing and projection operators, all natural numbers (via Church numerals), and definition by cases. Note that we use $\langle \cdot, \cdot \rangle$ to denote elements of a cartesian product (and not, as in other texts, for pairing in a PCA or for lambda abstraction).

Furthermore, by combinatory completeness, in every PCA each (ordinary) partial computable function corresponds to some element, and a wide range of computational facts about ordinary partial computable functions translate to arbitrary PCAs, including a form of lambda abstraction and the existence of fixed point operators (giving a version of the recursion theorem). However, one key feature that arbitrary PCAs do *not* admit is a sort of induction whereby one is able to iterate over all programs in sequence.

2.1 Examples

For concreteness, we present two of the most canonical examples of PCAs, although PCAs admit many more powerful or exotic models of computation, as described in the introduction.

Example 1. Kleene's first model is $\mathcal{K}_1 = (\mathbb{N}, \circ)$ where $a \circ b = \varphi_a(b)$, i.e., the application of the partial computable function with index a to the input natural number b.

In $\mathcal{K}_1$ there is a natural notion of a computational step, provided by, e.g., a single operation (i.e., overwriting the current cell, making a state transition, and moving the read/write head) on a Turing machine.

Example 2. Kleene's second model is $\mathcal{K}_2 = (\mathbb{N}^{\mathbb{N}}, \circ)$ where application $a \circ b$ involves treating the element a of Baire space as (code for) a partial continuous map $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$, applied to $b \in \mathbb{N}^{\mathbb{N}}$.

While one may typically think of an *element* of $\mathbb{N}^{\mathbb{N}}$ as naturally having a step-by-step representation (e.g., whereby a real is represented as a sequence of nested intervals), application itself does not admit an obvious decomposition in terms of steps, and so even here step algebras may provide additional granularity.

3 Step Algebras

We now proceed to consider a framework for computations where the fundamental "steps" are abstract functions. To do so, we define step algebras, the key notion of this extended abstract. A step algebra is a PAS with certain extra structure and properties, whose elements are an object paired with a state; as with PCAs, these elements can be thought of as either data or code. When an element is treated as code, its state will be irrelevant, but when an element is treated as data, the state of the data can inform what is done to it.

Specifically, if $\mathbb{A} = (A \times I, \circledast)$ is a step algebra, then for $\mathbf{a}, \mathbf{b} \in \mathbb{A}$ we will think of $\mathbf{a} \circledast \mathbf{b}$ as the result of applying code $\mathbf{a}$ to data $\mathbf{b}$ for *one time step*. An intuition that does not translate exactly, but is nonetheless useful, is to imagine the PAS as a "generalized Turing machine". In this context, the code for a Turing machine can be thought of as a function telling the Turing machine how to increment its state and tape by one time step. The result of running the Turing machine is then the result of iterating this fixed code. Another (imperfect) analogy is with lambda calculi, where a single step naturally corresponds with a single β-reduction.

With this intuition in mind, we will be interested in the operation of "iterating" the application of our functions until they halt. In order to make this precise, we will need a notion capturing when the result of a single operation has halted. We will achieve this by requiring one state, $\Downarrow$, to be a "halting state". In particular, we will require that if an element is in the halting state, then any code applied to it does not change it.

Definition 3. *Suppose $\mathbb{A} = (A \times I, \circledast)$ is a total PAS such that I has two (distinct) distinguished elements $0, \Downarrow$, and let $\langle \cdot, \cdot \rangle$ denote elements of the cartesian product $A \times I$. We say that $\mathbb{A}$ is a* **step algebra** *when*

1. $\langle a, i \rangle \circledast \mathbf{b} = \langle a, j \rangle \circledast \mathbf{b}$ *for all* $a \in A, \quad i, j \in I$, *and* $\mathbf{b} \in \mathbb{A}$, *and*
2. $\mathbf{a} \circledast \langle b, \Downarrow \rangle = \langle b, \Downarrow \rangle$ *for all* $\mathbf{a} \in \mathbb{A}$ *and* $b \in A$.

We write Step *to denote the class of step algebras.*

As with PCAs, in step algebras we will use the convention that when $\mathbf{a}, \mathbf{b} \in \mathbb{A}$, then $\mathbf{ab}$ represents the element $\mathbf{a} \circledast \mathbf{b}$, and will associate to the left.

The elements of a step algebra are meant to each describe a single step of a computation. Under this intuition, an entire (partial) computation arises from repeated application. Specifically, suppose $\mathbf{a} = \langle a, 0 \rangle$ and $\mathbf{b} = \langle b, 0 \rangle$ are elements of a step algebra $\mathbb{A} = (A \times I, \circledast)$. Then the computation describing a applied to b corresponds to the sequence

$$\mathbf{b},\ \mathbf{ab},\ \mathbf{a}(\mathbf{ab}),\ \mathbf{a}(\mathbf{a}(\mathbf{ab})),\ \ldots,$$

where if the sequence ever reaches a halting state $(c, \Downarrow)$, the computation is deemed to have finished with output c. Note that because of Definition 3.2, at most one such halting state can ever be reached by repeated application of $\mathbf{a}$ to $\mathbf{b}$. We now make precise this intuition of a partial computation arising from steps.

Definition 4. *We define the* **closure** *map* $\mathcal{P} : \mathrm{Step} \to \mathrm{PAS}$ *as follows. Suppose* $\mathbb{A} = \big(A \times I, \circledast\big)$ *is a step algebra. Let* $\mathcal{P}(\mathbb{A})$ *be the PAS* $(A, \circ)$ *such that*

$$a \circ b = c$$

if and only if there is a sequence of elements $\mathbf{c}_0, \ldots, \mathbf{c}_m \in \mathbb{A}$ *satisfying*

- $\langle b, 0\rangle = \mathbf{c}_0$,
- $\langle a, 0\rangle \circledast \mathbf{c}_i = \mathbf{c}_{i+1}$ *for* $0 \leq i < m$, *and*
- $\langle c, \Downarrow\rangle = \mathbf{c}_m$.

We now show that in fact every PAS is the closure of some step algebra.

Lemma 2. *The map* $\mathcal{P}$ *is surjective. Namely, for every PAS* $\mathbb{B} = (B, \circ)$ *there is a step algebra* $\mathbb{A} = (B \times \{0, \Downarrow\}, \circledast)$ *such that* $\mathcal{P}(\mathbb{A}) = \mathbb{B}$.

Proof. Let $\circledast$ be such that

- $\langle a, i\rangle \circledast \langle b, 0\rangle = \langle ab, \Downarrow\rangle$ for all $a, b \in B$ such that $ab{\downarrow}$ and $i \in \{0, \Downarrow\}$,
- $\langle a, i\rangle \circledast \langle b, 0\rangle = \langle b, 0\rangle$ for all $a, b \in B$ such that $ab{\uparrow}$ and $i \in \{0, \Downarrow\}$, and
- $\mathbf{a} \circledast \langle b, \Downarrow\rangle = \langle b, \Downarrow\rangle$ for all $\mathbf{a} \in \mathbb{A}$ and $b \in B$.

Note that $\mathbb{A}$ is a step algebra, as Definition 3.1 holds because the first and second points in the definition of $\circledast$ here are independent of i. We have $\mathcal{P}(\mathbb{A}) = \mathbb{B}$ because for all $a, b \in B$, if $ab{\downarrow}$ then $\langle a, 0\rangle \circledast \langle b, 0\rangle = \langle ab, \Downarrow\rangle$, and if $ab{\uparrow}$ then $\langle a, 0\rangle$ acts as the constant function on $\langle b, 0\rangle$. □

As we have just seen, every PAS comes from a step algebra where each computation occurs as a single step. Note that the collection of *terms* in a PAS can be made into a PAS itself by repeated reductions that consist of a single application of elements in the underlying PAS. Associated to such a PAS of terms, there is a natural step algebra, in which each step corresponds to a single term reduction.

Example 3. Suppose $\mathbb{A} = (A, \circ)$ is a PAS. Let A^* be the collection of closed terms of $\mathbb{A}$. Suppose the term a contains a leftmost subterm of the form $b \circ c$, for $b, c \in A$. Let a^+ be the result of replacing this subterm with the value of $b \circ c$ in A, if $b \circ c{\downarrow}$, and let a^+ be any value not in A otherwise. If a contains no such subterm, let $a^+ = a$. We define the step algebra $(A^* \times \{0, 1, \Uparrow, \Downarrow\}, \circledast^*)$ by the following equations, for all $a, b \in A^*$ and $i \in \{0, 1, \Uparrow, \Downarrow\}$.

- $\langle a, i\rangle \circledast^* \langle b, 0\rangle = \langle (a) \circ (b), 1\rangle$;
- $\langle a, i\rangle \circledast^* \langle b, 1\rangle = \langle b^+, 1\rangle$, when $b^+ \in A^*$ and $b^+ \neq b$;
- $\langle a, i\rangle \circledast^* \langle b, 1\rangle = \langle b, \Uparrow\rangle$ when $b^+ \notin A^*$;
- $\langle a, i\rangle \circledast^* \langle b, 1\rangle = \langle b, \Downarrow\rangle$ when $b^+ = b$; and
- $\langle a, i\rangle \circledast^* \langle b, \Uparrow\rangle = \langle b, \Uparrow\rangle$.

Now let $(A^*, \circ^*) = \mathcal{P}\big((A^* \times \{0, 1, \Uparrow, \Downarrow\}, \circledast^*)\big)$. It is then easily checked that for any sequences $a, b \in A^*$ and $c \in A$, we have $a \circ^* b = c$ if and only if $(a) \circ (b)$ evaluates to c in $\mathbb{A}$.

We now turn to a context where we are guaranteed to have more concrete tools at our disposal, with the goal of finding conditions that ensure that the closure of a step algebra is a PCA.

4 Complete Step Algebras and PCAs

The notion of a step algebra is rather abstract, and provides relatively little structure for us to manipulate. We now introduce some basic computational operations that will ensure that the closure of a step algebra is a PCA. These are modeled after the standard techniques for programming on a Turing machine (or other ordinary model of computation), but make use of our abstract notion of computation for the basic steps.

Specifically, there are four types of operations that we will consider. First, there is a very abstract notion of "hidden variables"; this allows us to read in and keep track of two elements, for future use. Second, there is the notion of an iterative step algebra; given two pieces of code, this provides code that runs the first until it halts, then runs the second until it halts on the output of the first, and finally returns the result of the second. We also allow for passing the hidden variables from the code running the pair to each of the individual pieces of code it runs. Third, we require code that returns the first hidden variable. Fourth, we require a pairing operation that allows us to either run the first hidden variable on the first element of the pair, or run the second hidden variable on the second element of the pair, or run the first element of the pair on the second.

We will show that the closure of a step algebra having such operations contains S and K combinators. In particular, by Lemma 1, this will show that having such operations ensures that the closure is a PCA.

We now introduce the notion of hidden variables; while this definition is quite general, we will make use of it later in the specific ways we have just sketched.

Definition 5. *Suppose* $\mathbb{A} = (A \times I, \circledast, v_0, v_1, r)$ *is such that*

- $(A \times I, \circledast)$ *is a step algebra,*
- $r : A \to A$ *is total, and*
- $v_0, v_1 : A \to A \cup \{\emptyset\}$ *are total.*

We say $\mathbb{A}$ *has* **hidden variables** *when for all* $b \in A$ *there is a (necessarily unique)* $a^b \in A$ *satisfying*

- $\langle r(a), 0\rangle \circledast \langle b, 0\rangle = \langle a^b, \Downarrow\rangle$ *and*
- $v_1(a^b) = b$ *and* $v_0(a^b) = v_1(a)$.

We will use the notation $a^{b,c}$ *to mean the element* $(a^b)^c$.

The rough idea is to require a stack containing at least two elements (which we sometimes refer to as the *registers*). The code $r(a)$ reads in the next element, b, and returns the code for "a with b pushed on the stack". In this view, $v_1(a)$ is the element most recently read by the code.

Before proceeding to see how this formalism is used, we make a few observations. First, we have not required that the states are preserved (although some step algebras may nonetheless keep track of their state); this is because we will mainly treat the objects we read in only as code, not data. Second, note that we have only assumed that the stack has two elements. (Likewise, some step algebras may happen to keep track of the entire stack — e.g., as part of a reversible computation.)

Definition 6. *Suppose* $\mathbb{A} = (A \times I, \circledast, (v_0, v_1, r), (\pi_0, \pi_1, t))$ *is such that*

- $(A \times I, \circledast, v_0, v_1, r)$ *is a step algebra with hidden variables,*
- $\pi_0, \pi_1 : A \times A \times A \times A \times I \to I$ *with* π_0, π_1 *both total and injective in the last coordinate (i.e.,* $\pi_i(a_0, a_1, l_0, l_1, \cdot)$ *is injective for each* $a_0, a_1, l_0, l_1 \in A$*), and*
- $t : A \times A \to A$ *is total.*

We then say that $\mathbb{A}$ *is an* **iterative step algebra** *if whenever* $a_0, a_1, l_0, l_1, b \in A$ *with* $\pi_0^*(\cdot) = \pi_0(a_0, a_1, l_0, l_1, \cdot)$, $\pi_1^*(\cdot) = \pi_1(a_0, a_1, l_0, l_1, \cdot)$, *and* $t = t(a_0, a_1)^{l_0, l_1}$, *we have*

- $\langle t, 0\rangle \circledast \langle b, 0\rangle = \langle b, \pi_0^*(0)\rangle$,

- $\langle t, 0\rangle \circledast \langle b, \pi_0^*(j)\rangle = \langle b', \pi_0^*(j')\rangle$ *when* $\langle a_0^{l_0,l_1}, 0\rangle \circledast \langle b, j\rangle = \langle b', j'\rangle$ *and* $j' \neq \Downarrow$,
- $\langle t, 0\rangle \circledast \langle b, \pi_0^*(j)\rangle = \langle b', \pi_1^*(0)\rangle$ *when* $\langle a_0^{l_0,l_1}, 0\rangle \circledast \langle b, j\rangle = \langle b', \Downarrow\rangle$,

- $\langle t, 0\rangle \circledast \langle b, \pi_1^*(j)\rangle = \langle b', \pi_1^*(j')\rangle$ *when* $\langle a_1^{l_0,l_1}, 0\rangle \circledast \langle b, j\rangle = \langle b', j'\rangle$ *and* $j' \neq \Downarrow$, *and*
- $\langle t, 0\rangle \circledast \langle b, \pi_1(j)\rangle = \langle b', \Downarrow\rangle$ *when* $\langle a_1^{l_0,l_1}, 0\rangle \circledast \langle b, j\rangle = \langle b', \Downarrow\rangle$.

Intuitively, a step algebra is iterative when for every pair of code a_0 and a_1, there is code such that when it has l_0 and l_1 as its stack variables and is given a piece of data in state 0, it first runs a_0 with stack values (l_0, l_1) until it halts, then resets the state to 0 and runs a_1 with stack values (l_0, l_1) until it halts, and finally returns the result.

Note that while we have only defined this for pairs of code, the definition implies that elements can be found which iteratively run sequences of code of any finite length. We write $t_{a_0,\ldots,a_m}$ for code that first runs a_0 (with appropriate stack values) until it halts, then runs a_1 (with the same stack values) until it halts, and so on.

There are two subtleties worth mentioning about π_0^* and π_1^*. First, these take as input the states of t as well as the code that t is following. This is because we want it to be possible, in some cases, for π_0^*, π_1^* to keep track of the operations being performed.

Second, while we have assumed that π_0^* and π_1^* are always injective, we have not assumed that they have disjoint images (even outside of $\{\Downarrow\}$). One example that might be helpful to keep in mind is the case of $I = \mathbb{N} \cup \{\Downarrow, \Uparrow\}$ where each element of our step algebra is constant on elements whose state is in $\{\Downarrow, \Uparrow\}$, where π_0^*, π_1^* are constant on $\{\Downarrow, \Uparrow\}$, and where $\pi_i^*(n) = 2 \cdot n + i + 1$. In this case we can think of the state $\Uparrow$ as "diverges", i.e., a state that if reached will never halt, and we can think of of the maps π_i as using the natural bijections between even and odd natural numbers to "keep track" of what state we are in as we apply multiple pieces of code.

We are now able to give the two conditions that guarantee the desired combinators.

Definition 7. *Suppose* $\mathbb{A} = (A \times I, \circledast, (v_0, v_1, r))$ *is a step algebra with hidden variables. We say* $\mathbb{A}$ *that has* **constant functions** *when there is some* $c \in A$ *such that for all* $x, y \in A$*, we have* $\langle c^x, 0\rangle \circ \langle y, 0\rangle = \langle x, \Downarrow\rangle$*.*

We can think of c as code that simply returns the value in its first register. In this case, c^x is then code that already has x in its first register and that returns the value in its first register. In particular, we have the following easy lemma.

Lemma 3. *Suppose* $\mathbb{A} = (A \times I, \circledast, (v_0, v_1, r))$ *is an iterative step algebra with a constant function* c*. Let* $(A, \circ) = \mathcal{P}\big((A \times I, \circledast)\big)$ *be the closure of* $\mathbb{A}$*. Then for all* $x, y \in A$*, we have* $(r(c) \circ x) \circ y = x$*.*

Proof. The code $r(c)$ first reads x into its first register, and then returns c^x, which itself is code that returns what is in the first register (i.e., x). □

In particular, if $\mathbb{A}$ is an iterative step algebra with a constant function, then the closure of $\mathbb{A}$ has a K combinator.

Definition 8. *Suppose* $\mathbb{A} = (A \times I, \circledast, (v_0, v_1, r), ([\cdot, \cdot], p, p_0, p_1, p_2))$ *is such that*

- $(A \times I, \circledast, v_0, v_1, r)$ *is a step algebra with hidden variables,*
- $[\,\cdot, \cdot\,] : A \times A \to A$ *is total, and*
- $p, p_0, p_1, p_2 \in A$*.*

We then say that $\mathbb{A}$ *has* **pairing** *when for all* $a_0, a_1, b_0, b_1 \in A$ *and* $j \in I$*,*

- $\langle p^{a_0,a_1}, 0\rangle \circledast \langle b_0, 0\rangle = \langle [b_0, b_0], \Downarrow\rangle$*,*
- $\langle p_0^{a_0,a_1}, 0\rangle \circledast \langle [b_0, b_1], j\rangle = \langle [b', b_1], j'\rangle$ *when* $\langle a_0, 0\rangle \circledast \langle b_0, j\rangle = \langle b', j'\rangle$*,*
- $\langle p_1^{a_0,a_1}, 0\rangle \circledast \langle [b_0, b_1], j\rangle = \langle [b_0, b'], j'\rangle$ *when* $\langle a_1, 0\rangle \circledast \langle b_1, j\rangle = \langle b', j'\rangle$*,*
- $\langle p_2^{a_0,a_1}, 0\rangle \circledast \langle [b_0, b_1], j\rangle = \langle [b_0, b'], j'\rangle$ *when* $\langle b_0, 0\rangle \circledast \langle b_1, j\rangle = \langle b', j'\rangle$ *and* $j' \neq \Downarrow$*, and*
- $\langle p_2^{a_0,a_1}, 0\rangle \circledast \langle [b_0, b_1], j\rangle = \langle b', j'\rangle$ *when* $\langle b_0, 0\rangle \circledast \langle b_1, j\rangle = \langle b', \Downarrow\rangle$*.*

We say that $\mathbb{A} = (A \times I, \circledast, (v_0, v_1, r), (\pi_0, \pi_1, t), ([\cdot, \cdot], p, p_0, p_1, p_2))$ *is an iterative step algebra with pairing when* $(A \times I, \circledast, (v_0, v_1, r), (\pi_0, \pi_1, t))$ *is an iterative step algebra and* $(A \times I, \circledast, (v_0, v_1, r), ([\cdot, \cdot], p, p_0, p_1, p_2))$ *is a step algebra with pairing. We will sometimes abuse notation and speak of the closure of* $\mathbb{A}$ *to mean* $\mathcal{P}\big((A \times I, \circledast)\big)$*.*

Intuitively, we say that $\mathbb{A}$ has pairing when there is an external pairing function $[\,\cdot, \cdot\,]$ along with an element of $\mathbb{A}$ that takes an element and pairs it with itself; an element that applies what is in the first register to the first element of the pair; an element that applies what is in the second register to an element of the pair; and an element that applies the first element of the pair to the second element, returning the answer if it halts.

Lemma 4. *Suppose* $\mathbb{A} = \big(A \times I, \circledast, (v_0, v_1, r), (\pi_0, \pi_1, t), ([\,\cdot\,,\cdot\,], p, p_0, p_1, p_2)\big)$ *is an iterative step algebra with pairing. Then the closure of* $\mathbb{A}$ *has an* S *combinator.*

Proof sketch. Suppose $(A, \circ)$ is the closure of $\mathbb{A}$. Let $S_2 = t_{p,p_0,p_1,p_2}$ and let $S = r(r(S_2))$. Intuitively, S_2 takes an argument d and then runs the following subroutines in succession:

- Return $[d, d]$.
- Return $[v_0(S_2) \circ d, d]$.
- Return $[v_0(S_2) \circ d, v_1(S_2) \circ d]$.
- Return $(v_0(S_2) \circ d) \circ (v_1(S_2) \circ d)$.

But then Sab is code that first reads in a, then reads in b (and moves a to the 0th register), and then performs the above. Hence S is the desired combinator. □

Definition 9. *Let* $\mathbb{A} = (A \times I, \circledast)$ *be a step algebra. We say that* $\mathbb{A}$ *is a* **complete** *step algebra when it can be extended to an iterative step algebra with pairing and with constant functions.*

Theorem 1. *If* $\mathbb{A}$ *is a complete step algebra, its closure is a PCA.*

Proof. This follows immediately from Lemma 1, Lemma 3, and Lemma 4. □

Conjecture 1. Every PCA is isomorphic to a PCA that arises as the closure of a complete step algebra (for a suitable notion of isomorphism).

5 Future Work

Here we have begun developing a notion of a single step of a computation, in the setting of PCAs. Having done so, we can now begin the project of developing robust notions of efficient computation in this general setting. For example, we aim to use step algebras to extend a notion of computational complexity to arbitrary PCAs (e.g., by considering suitably parametrized families of step algebras).

Many questions also remain about the class of step algebras whose closures yield the same PCA. In particular, there are many natural options one might consider within the partition on step algebras induced in this way. For example, the relationship between a step algebra and the one obtained by uniformly collapsing every n-step sequence into a single element, or those obtained by many other transformations, remains unexplored.

Finally, we plan to use step algebras to develop a notion of reversible computation in the general context of PCAs. The fine-grained analysis of computational steps might be used to ensure that each step is injective (whether by requiring that a complete step algebra keep track of its entire stack, or obtained by other means). Under an appropriate formulation of reversibility, one might explore whether, for every PCA, there is an essentially equivalent one in which computation is fully reversible.

Acknowledgements
The authors thank Bob Lubarsky for helpful conversations, and thank Rehana Patel, Dan Roy, and the anonymous referees for comments on a draft. This publication was made possible through the support of grants from the John Templeton Foundation and Google. The opinions expressed in this publication are those of the authors and do not necessarily reflect the views of the John Templeton Foundation.

References

[Aka04] Akama, Y.: Limiting partial combinatory algebras. Theoret. Comput. Sci. 311(1-3), 199–220 (2004)

[BDG09] Blass, A., Dershowitz, N., Gurevich, Y.: When are two algorithms the same? Bull. Symbolic Logic 15(2), 145–168 (2009)

[Cur29] Curry, H.B.: An analysis of logical substitution. Amer. J. Math. 51(3), 363–384 (1929)

[Fen80] Fenstad, J.E.: General recursion theory: an axiomatic approach. Perspectives in Mathematical Logic. Springer-Verlag, Berlin (1980)

[Fri71] Friedman, H.: Algorithmic procedures, generalized Turing algorithms, and elementary recursion theory. In: Logic Colloquium '69 (Proc. Summer School and Colloq., Manchester, 1969). North-Holland, Amsterdam, 361–389 (1971)

[Gur00] Gurevich, Y.: Sequential abstract-state machines capture sequential algorithms. ACM Trans. Comput. Log. 1(1), 77–111 (2000)

[Lon95] Longley, J.: Realizability toposes and language semantics. PhD thesis, University of Edinburgh, College of Science and Engineering, School of Informatics (1995)

[Mos01] Moschovakis, Y.N.: What is an algorithm? In: Mathematics unlimited—2001 and beyond. Springer, Berlin, 919–936 (2001)

[Rat06] Rathjen, M.: Models of intuitionistic set theories over partial combinatory algebras. In: Theory and Applications of Models of Computation (TAMC 2006). Vol. 3959 of LNCS. Springer, Berlin, 68–78 (2006)

[Sch24] Schönfinkel, M.: Über die Bausteine der mathematischen Logik. Math. Ann. 92(3-4), 305–316 (1924)

[vO06] van Oosten, J.: A general form of relative recursion. Notre Dame J. Formal Logic 47(3), 311–318 (2006)

[vO08] van Oosten, J.: Realizability: an introduction to its categorical side. Vol. 152 of Studies in Logic and the Foundations of Mathematics. Elsevier B. V., Amsterdam (2008)