Journal of Logical and Algebraic Methods in Programming 84 (2015) 259–275

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Monoids with tests and the algebra of possibly non-halting programs

Marcel Jackson [a,1,*], Tim Stokes [b]

[a] Department of Mathematics and Statistics, La Trobe University, Victoria, Australia
[b] Department of Mathematics, University of Waikato, New Zealand

ARTICLE INFO

Article history:
Received 9 July 2013
Received in revised form 19 August 2014
Accepted 21 August 2014
Available online 16 September 2014

Keywords:
Computable partial functions
Algebraic models of computation
Deterministic programs
Domain
Restriction semigroup
If-then-else

ABSTRACT

We study the algebraic theory of computable functions, which can be viewed as arising from possibly non-halting computer programs or algorithms, acting on some state space, equipped with operations of composition, *if-then-else* and *while-do* defined in terms of a Boolean algebra of conditions. It has previously been shown that there is no finite axiomatization of algebras of partial functions under these operations alone, and this holds even if one restricts attention to transformations (representing halting programs) rather than partial functions, and omits *while-do* from the signature. In the halting case, there is a natural "fix", which is to allow composition of halting programs with conditions, and then the resulting algebras admit a finite axiomatization. In the current setting such compositions are not possible, but by extending the notion of *if-then-else*, we are able to give finite axiomatizations of the resulting algebras of (partial) functions, with *while-do* in the signature if the state space is assumed finite. The axiomatizations are extended to consider the partial predicate of equality. All algebras considered turn out to be enrichments of the notion of a (one-sided) restriction semigroup.

1. Motivation and definitions

1.1. Some terminology

Let X, Y be sets. A *function* $X \to Y$ is a partial map from a subset of X into Y, and the set of all such is denoted $\mathcal{P}(X, Y)$. If $Y = X$ this is denoted $\mathcal{P}(X)$, a semigroup under composition (read left to right, so that $(fg)(x) = g(f(x))$ for all $f, g \in \mathcal{P}(X)$ and $x \in X$), and an element of $\mathcal{P}(X)$ is called a *function on* X. Because X is usually some fixed "global domain", we use the name *domain* of f (written $\mathrm{dom}(f)$) to denote the subset of points at which f is actually defined. The *identity map* 1_X on X is the total function on X that fixes every $x \in X$, and the *null map* 0_X is the function on X with empty domain; they are respectively identity and zero elements in the semigroup $\mathcal{P}(X)$. The subscript X will be omitted where the choice is clear. A *transformation on* X is an everywhere-defined function in $\mathcal{P}(X)$ (that is, having domain all of X); the set of all such is $\mathcal{T}(X)$, a submonoid of $\mathcal{P}(X)$. Transformations are also known as total functions.

A *predicate* on X is an everywhere-defined function $X \to \{T, F\}$; the set of all such is 2^X, a Boolean algebra under the usual logical connectives. Denote by $I(X)$ the monoid of restrictions of the identity function under composition; it is isomorphic to the semilattice $(2^X, \cap)$.

* Corresponding author.
E-mail addresses: m.g.jackson@latrobe.edu.au (M. Jackson), stokes@math.waikato.ac.nz (T. Stokes).

[1] The first author was supported by ARC Future Fellowship FT120100666 and ARC Discovery Project DP1094578.

http://dx.doi.org/10.1016/j.jlamp.2014.08.007

1.2. Computable functions

Many authors have investigated algebraic foundations for facets of the theory of computer programs: amongst others we have in mind are *sum-ordered (partial) semirings* (Manes and Benson [29]), *dynamic algebras* (Pratt and others, see [34]), *Kleene algebras with tests* (KAT) (Kozen [25]), *Kleene algebra with domain* (KAD; Desharnais, Möller and Struth [9]; see also Hirsch and Mikulás [18] and Desharnais, Jipsen and Struth [8]), *modal semirings* (Möller and Struth [32]), *refinement algebras* (von Wright [40]), *correctness algebras* (Guttmann [16]), as well as the authors' own contributions such as *modal restriction semigroups* [22]. Approaches based on the full Tarski algebra of relations are detailed by Maddux in [27].

These algebraic approaches have substantial connections with classical program logics, such as Hoare logic and various propositional dynamic logics, enabling straightforward algebraic equational reasoning to supplant the conventional logical approach. Indeed, with the exception of KAT (which does not allow for domain information), the algebraic systems mentioned are designed to interpret at least the modal logic part of dynamic logic. However the family of computable (that is, partial recursive) functions on a set X is not typically a model of any of these algebraic systems.

First, in all but the case of modal restriction semigroups, these algebraic systems allow for union and usually reflexive transitive closure, reflecting the expressiveness of the associated logical systems. This effectively forces a relational semantics rather than a functional one. However, if one sticks to the basic *if-then-else* and *while-do* constructs, there is no need to use a relational semantics: partial functions suffice.

Additionally, many of these systems admit a notion of domain complementation, which is not compatible with the computability assumption. More specifically, in dynamic logic, the proposition $[f]$ false ("necessarily false" as determined by f as a modal relation) holds exactly on the points at which the program f fails to halt, and in general this is clearly not computable. Moreover, in the algebraic formulations mentioned (as well as in dynamic logic itself), these propositions may themselves then become test conditions within other programs. Thus constructions such as "while program f does not halt do program g" can be expressed, and indeed give rise to their own nonhalting proposition and so on.

Test conditions arising in actual programs are typically Boolean combinations of basic tests, and certainly not statements on halting conditions. The goal of the present article is to present algebraic systems that are rich enough to express standard deterministic programming connectives, such as *if-then-else*, as well as tests for equality and non-equality of variable values, yet does not permit the leaching of statements on halting into the test type. The set of all partial recursive functions on $\mathbb{N}$ will provide a model, in contrast to the more common approach in which binary relations model programs. (There are of course other approaches to program algebra, such as predicate transformer semantics, but these work at a lower level in which assignments are modeled for example.)

The main results consist of finite axiomatizations for these systems, which we are able to show are complete (for the full first order theory, not just the equational fragment) with respect to suitable functional semantics. Operations modeling looping are also considered, but for these we are unable to obtain finite axiomatizations, instead making do with axioms at least guaranteeing that certain desirable properties are satisfied, including completeness for finite (and even periodic) algebras.

The basic approach shares features of both the KAT approach of Kozen [25] and the modal restriction semigroups with preferential join approach of Jackson and Stokes [22]. Our algebraic systems consist of a semigroup of functions with an embedded sort B of "tests", which will form a Boolean algebra in which the meet operation is just the underlying multiplication of the semigroup (so that the tests will form a subsemilattice: an idempotent and commutative subsemigroup), and with a Boolean complementation operation (which is not defined outside of the test sort).

So far this is identical to the union- and star-free fragment of the algebraic systems considered in KAT. However, we also consider unary operations modelling domain (as in KAD, or modal restriction semigroups and its predecessors [19]) as well as various equality test, *if-then-else* and *while-do* constructions. All elements of the test sort B are fixed by domain, but in general domain elements form a strictly larger subsemilattice than B. This reflects the fact that for us, tests are to be viewed as conditions in programs (and indeed as special types of programs themselves) rather than as assertions: we have no need to generate weakest preconditions for example, hence no need to view general domain elements as tests, and indeed no need for domain complement (antidomain) at all.

Another significant difference with KAT and KAD is that we use partial functions as our semantics of programs rather than binary relations. This relates to the fact that we seek to model programs themselves rather than assertions about programs: we have no need of union or Kleene closure, which feature in dynamic logic for example, so we do not need binary relations either. (Binary relations are of course closed under both operations, while partial functions are closed under neither.)

The computable functions on a set X will form a model of each of the systems we consider, with B modelling a Boolean algebra of identity maps with *recursive* domains, and with general domain elements corresponding to the identity map on recursively enumerable subsets of X (which are exactly the domains of computable functions).

For the remainder of this section we further motivate and then carefully define the additional operations used to model *if-then-else* and other constructions.

1.3. *Extending* if-then-else

Operations modelling the *if-then-else* command of imperative programming languages have been modelled algebraically by a number of authors. The idea is to model programs as functions $X \to Y$ (or even binary relations in $X \times Y$, though we