ALGEBRAIC PROPERTIES OF `IF-THEN-ELSE` AND COMMUTATIVE THREE-VALUED TESTS

TIM STOKES and ROGER SU

Abstract

This paper establishes a finite axiomatization of possibly non-halting computer programs and tests, with the `if-then-else` operation. The model is a two-sorted algebra, with one sort being the programs and the other being the tests. The main operation on programs is composition, and 1 and 0 represent the programs `skip` and `loop` (*i.e.* never halts) respectively. Programs are modeled as partial functions on some state space X, with tests modeled as partial predicates on X.

The operations on the tests are the usual logical connectives $\wedge$, $\vee$, $\neg$, T and F. In addition, there is the hybrid operation of `if-then-else`, and the test-valued operation H on programs which is true when a program halts, and undefined otherwise. The halting operation H implies that operations of domain D and domain join $\vee$ may also be expressed.

When tests are assumed to be possibly non-halting, the evaluation strategy of the logical connectives affects the result. Here we model parallel evaluation, as opposed to the common sequential (or short-circuit) evaluation strategy. For example, we view $\alpha \wedge \beta$ as false if either α or β is false, even if the other does not halt.

Keywords: if-then-else; algebra of programs; function semigroups.
Mathematics Subject Classification 2010: 08A70, 03G25 and 68Q99.

1 Introduction

There are numerous frameworks to study the algebraic properties of `if-then-else`. McCarthy [1], Bloom and Tindell [2], Guessarian and Meseguer [3], and Pigozzi [4] studied `if-then-else` as a one-sorted algebra, where there are only one set of programs, with the logical constants "true" T and "false" F being distinguished elements. This approach only studies the properties of `if-then-else`, and neither program composition nor logical connectives are covered.

Kozen [5] and Desharnais *et al.* [6] viewed both programs and tests as binary relations, and modeled them by a Kleene algebra. They not only treated both program composition and logical connectives; with the closure operator in a Kleene algebra, they also modeled the `while-do` construct. However, this approach yields no finite axiomatisation.

Another stream of research is by Bergman [7] and Manes [8]. Bergman used sheaf theory to study the action of a Boolean algebra (tests) on a set (programs), and Manes used category theory to study `if-then-else` with both two- and three-valued tests. Both of these two works focus on how the logical connectives interact with the `if-then-else` operation, but do not cover program composition.

In [9], Jackson and Stokes used a two-sorted algebraic approach to study a version similar to those of Bergman and Manes. Jackson and Stokes looked at programs and tests which always halt, and considered the operations of program composition, logical connectives, `if-then-else` and program-test composition. (The composition of a program p and a test α is a test which determines whether α is true after executing p.) Jackson and Stokes modeled programs by total functions on a set, and tests by total predicates on the same set. They obtained a finite equational axiomatization of the algebras of functions and predicates closed under these operations, and showed that without the program-test composition, the resulting semantic models were not finitely axiomatizable.

In [10], Jackson and Stokes studied possibly non-halting programs, modelled as partial functions on a base set. (States at which a program does not halt are modelled by the places where the function is undefined.) They considered a richer signature, which includes halting tests, special programs `skip` and `loop`, and an oracle for the halting problem. This signature means that the domain operation (`skip` restricted to were a program halts) is expressible, along with its complement, and the test sort could effectively be incorporated into the program sort. Here, the authors obtained a finite equational axiomatization, and they extended this to the cases where program intersection is also modeled.

In [11], Jackson and Stokes attempted to model only computable operations (with no oracle for the halting problem). The signature again facilitates the expression of domain as well as program-test composition. As noted there, this induces an algebra of non-halting tests in which tests are evaluated sequentially. Panicker *et al.* [12] generalized [11] in the concatenation-free case to allow arbitrary non-halting tests within `if-then-else` statements, but under sequential evaluation. Recently, they extended this signature by including composition as well, in [13].

This present paper follows the algebraic approach described above, but we adopt the parallel evaluation strategy for the logical connectives in order to maximize the scope of computability. The signature we consider is the same as in [9], with a separate test sort. However, the true and false components of the tests arise as programs (restrictions of `skip`). There is a halting predicate H that can be applied to any program p, and $H(p)$ evaluates to true when p halts (*i.e.* is defined) and, like p, is undefined otherwise. Note that this is a "real-world" test, since affirming halting is always possible, while failure of a program to halt will lead to the affirmation test not to halt.

2 Motivation and Operations Considered

Before building our formal framework, we first give an overview of the objects we use in our algebraic model, and explain the motivations behind them.

2.1 Programs as Functions

We study programs that are deterministic but may not halt. Hence the programs we model give rise to partial functions on the state space X, which we denote $\mathcal{P}(X)$. Different programs can give rise to the same partial functions, so strictly speaking we are only modelling programs up to equivalence (identical outputs for each input). So, in future when we refer to a program p, we really mean the partial function it induces.

2.2 Operations on Programs

An important operation on programs is sequential application, which corresponds to composition of their corresponding partial functions, an operation well-known to be associative. Given programs p and q, we use $p \cdot q$ or simply pq to denote their composition. The programs with composition form a semigroup.

In addition to composition, we include in the signature two nullary operations which represent the special programs of `skip` and `loop`. The `skip` program is the one that always outputs the same value as its input, and it acts like the identity. On the other hand, the `loop` program is the one that never halts, represented by the empty function. It acts like the zero element in the semigroup of programs. Approaches such as those involving domain semirings as in [6] routinely assume the presence of such constant elements.

2.3 Tests

Now, logical tests will be brought into the picture, which is the place where our approach is most distinctive.

A possibly non-halting test gives rise to a partial predicate, which is a partial function from X to $\{T, F\}$. The undefined part of a partial predicate represents where the evaluation of a test has not halted. Denote by $\mathcal{P}Pred(X)$ the set of partial predicates on the set X.

Partial predicates can possess all the usual operations which total predicates have, namely the connectives "and" $\wedge$, "or" $\vee$ and "not" $\neg$, as well as the constants "true" T and "false" F. However, the definitions of the binary connectives $\wedge$ and $\vee$ become dependent on how these connectives are evaluated: whether sequentially or in parallel. The following tables show the connectives under the parallel evaluation strategy, with U denoting the situation when a test "has not halted".

$\wedge$	T	F	U
T	T	F	U
F	F	F	F
U	U	F	U

$\vee$	T	F	U
T	T	T	T
F	T	F	U
U	T	U	U

$\neg$	
T	F
F	T
U	U

The difference between the two evaluation strategies can be seen in the cases of $U \wedge F$ and $U \vee T$. The connectives are not commutative when evaluated sequentially as in [8] and [11], but are commutative under the parallel paradigm used here.

Note that the logic we use above is that used by Kleene in defining the logic K_3 in [14].

A convenient means to specify partial predicates is the notion of *disjoint pairs.* A partial predicate α has a "true" part $\alpha_T = \{x \in X \mid x\alpha = T\}$, as well as a "false" part $\alpha_F = \{x \in X \mid x\alpha = F\}$. Both of these are subsets of X, and $\alpha_T \cap \alpha_F = \emptyset$ because nothing can be both true and false at once. This concept leads to the following high-level definition.

Definition 2.1 *The* ***algebra of disjoint pairs*** *over a bounded distributive lattice* $(L\ ; \cup, \cap, 1, 0)$ *is an algebra* $(L^{\wedge}\ ; \wedge, \vee, \neg, \top, \bot)$*, where*

$$L^{\wedge} := \big\{\big\langle a, b\big\rangle \mid a, b \in L\ ;\ a \cap b = 0\big\},$$

and the operations on these are defined as follows.

- $\big\langle a, b\big\rangle \wedge \big\langle c, d\big\rangle := \big\langle a \cap c, b \cup d\big\rangle$

- $\langle a, b\rangle \vee \langle c, d\rangle := \langle a \cup c, b \cap d\rangle$
- $\neg\langle a, b\rangle := \langle b, a\rangle$
- $\top := \langle 1, 0\rangle$ *and* $\bot := \langle 0, 1\rangle$

The algebra of the subsets of a set correspond directly to a Boolean algebra, and a Boolean algebra is in turn a bounded distributive lattice, so we can also speak of the algebra of disjoint pairs *on a Boolean algebra* or *over a set*. In fact, there is a direct isomorphism between an algebra of partial predicates and an algebra of disjoint pairs.

2.4 The `if-then-else` Operation

The first hybrid operation which involves both tests and programs is `if-then-else`. Formally, if S denotes the programs and K denotes the tests, then the `if-then-else` is a ternary operation

$$\_\,[\,\_\,,\ \_\,]\ :\ K \times S \times S \to S.$$

When the test halts and produces either T or F, the behavior of `if-then-else` is given by

$$T[p, q] := p, \text{ and}$$
$$F[p, q] := q.$$

If the test does not halt, the `if-then-else` statement would be trying to evaluate the test indefinitely, so the whole statement behaves like `loop`. Hence, $U[p, q] := 0$.

2.5 The `Halt` Test, the Domain Operation and Domain Join

Since a program may not halt at certain states, it is natural to ask "whether a program does terminate". This is the `halt` test, denoted H, and it is an operation which takes a program p as input and returns a test $H(p)$ as output. The result $H(p)$ is defined as follows for all $x \in X$:

$$x\big(H(f)\big) := \begin{cases} T & \text{if } xf \text{ is defined;} \\ F & \text{nowhere;} \\ U & \text{if } xf \text{ is undefined.} \end{cases}$$

Note that this `halt` test can never return false. Asking whether $H(p)$ is false is the same as asking whether p is non-halting. This is exactly the halting problem, which is well known not to be computable. Since we seek to model computable programs, such non-computable statements should not be expressible.

When taken together with `if-then-else`, the `halt` test $H(p)$ gives rise to the *domain* operation on programs via the expression $\big(H(p)\big)[1,0]$, which represents "the restriction of the identity function to where p is defined". The domain operation $D(p)$ is thus defined by $\big(H(p)\big)[1,0]$. Conversely, H can be implicitly expressed in terms of D, as $H(p)$ is exactly the test α which satisfies $\alpha[1,0] = D(p)$ and $\alpha[0,1] = 0$. Therefore, in order to model H, it suffices to model D, and then re-introduce H via such an implicit definition.

Now, given two programs p and q, both $H(p) \wedge H(q)$ and $H(p) \vee H(q)$ are also tests, and $\big(H(p) \wedge H(q)\big)[1,0]$ and $\big(H(p) \vee H(q)\big)[1,0]$ should represent the restriction of the identity function to "where both p and q are defined" and "where either p or q is defined" respectively. This means that there are implicitly defined lattice operations *join* and *meet* on the set of identity restrictions, so we might as well add them to the singature from the outset. It is a familiar fact that *meet* on identity restrictions is modeled by functional composition. On the other hand, *join* needs to be introduced as an additional operation. Hence, the algebra of programs will have an extra operation $\vee$ called the *domain join*, defined on domain elements (but easily extendable to arbitrary programs if one wishes by defining $p \vee q = D(p) \vee D(q)$).

Further justification of inclusion of H in our signature comes from the following observation. If the halting test operation H, and hence also D, are omitted from the signature we are considering, and only an abstract collection of tests together with composition and if-then-else remain, we have a signature similar to the one considered in [9, Section 6] for the case of programs with halting tests. There, it was established that no finite axiomatization was possible. Indeed the case considered there can be viewed as a special case of the one we have been considering with the halting test (and hence domain) removed from the signature, satisfying the law that $\alpha \vee \neg\alpha = T$ for all tests α. It follows that this signature without the halting test operation H also has no finite axiomatization.

2.6 Main Theorem

To summarize, we are interested in $S = \mathcal{P}(X)$, the partial functions on some set X, together with $K = \mathcal{P}Pred(X)$, the partial predicates on X, along with the following operations introduced previously in this section:

- composition of partial functions $\cdot\ :\ S \times S \to S$;

- tje distinguished programs $1, 0 \in S$, where 1 is the identity function and 0 the empty function;
- the `if-then-else` operation $\_\,[\,\_\,,\,\_\,] \;:\; K \times S \times S \to S$;
- logical connectives $\wedge, \vee \;:\; K \times K \to K$ and $\neg \;:\; K \to K$, as well as the distinguished tests $T, F \in K$;
- the halt test $H \;:\; S \to K$;
- the domain operation $D \;:\; S \to S$ given by $D(p) := H(p)[1, 0]$;
- domain join operation applying to elements of $D(S) = \{D(s) \mid s \in S\}$ and given by $D(p) \vee D(q) := \big(H(p) \wedge H(q)\big)[1, 0]$.

Our main interest is thus in two-sorted algebras of partial functions and possibly non-halting tests that embed in the two-sorted algebra of all partial functions $\mathcal{P}(X)$ and all tests $\mathcal{P}Pred(X)$ for some X, with the above operations.

In the course of what follows, we prove the following.

Theorem 2.2 *The first-order axioms (S1)—(S3), (D1)—(D7), (T1)—(T6), (G1)—(G2), and (H1)—(H2) characterize the class of two-sorted algebras of programs and tests under the above operations.*

3 Modeling the Program Operations

Our approach starts by modeling the programs, whose operations are composition, domain, 1 and 0, with the additional domain join.

3.1 Composition, Domain, 1 and 0

The programs with composition, domain, 1 and 0 are modeled by an algebra $(S\;;\;\cdot\,, D, 1, 0)$, where

(S1) $(S\;;\;\cdot\,, 1)$ is a monoid;

(S2) 0 is the zero element;

(S3) $(S\;;\;\cdot\,, D)$ forms a left restriction semigroup.

The axioms of a left restriction semigroup are the following. For all $x, y \in S$:

(R1) $D(x)x = x$

(R2) $D(x)D(y) = D(y)D(x)$

(R3) $D\big(D(x)\big) = D(x)$

(R4) $D(xy) = D(x)D(xy)$

(R5) $xD(y) = D(xy)x$

Soundness and completeness of these axioms for a functional interpretation was first shown by Trokhimenko in [15], and rediscovered by others several times since.

The axioms above further imply the following useful identities, first established in [16, Proposition 1.2].

Proposition 3.1 *Let S be a left restriction semigroup, then the following are true for all $x, y \in S$.*

(R6) $D(x)D(y) = D\big(D(x)D(y)\big)$

(R7) $D(x)D(x) = D(x)$

(R8) $D(x)D(y) = D\big(D(x)y\big)$

(R9) $D(xy) = D\big(xD(y)\big)$

We define $D(S)$ to be the set $\{a \in S \mid D(a) = a\}$, or equivalently $\{D(x) \mid x \in S\}$. Then, it is easy to see that $D(S)$ is a subsemigroup of S. In particular, 1 and 0 are both in $D(S)$.

At a later stage of this paper, we require the following proposition too.

Proposition 3.2 *Let $x, a, b \in S$ with $a, b \in D(S)$. Then, $D(xab) = D(xa) \cdot D(xb)$.*

Proof. Starting from the right-hand side:

$$\begin{aligned}
D(xa) \cdot D(xb) &= D\big(D(xa) \cdot xb\big) \cdot D(xa) && \text{(R5)} \\
&= D\big(D(xa)x \cdot b\big) \cdot D(xa) && \\
&= D\big(xD(a) \cdot b\big) \cdot D(xa) && \text{(R5)} \\
&= D(xab) \cdot D(xa) && \text{(because } D(a) = a\text{)} \\
&= D(xa) \cdot D(xab) && \text{(R2)} \\
&= D(xab) && \text{(R4)}
\end{aligned}$$

□

3.2 Domain Join

We introduce a new operation of *domain join* on a left restriction semigroup. Let $\vee : D(S) \times D(S) \to D(S)$ be a binary operation on $D(S)$, which satisfies the following axioms. For all $a, b, c \in D(S)$ and $x, y \in S$:

(D1) $(a \vee b) \vee c = a \vee (b \vee c)$

(D2) $a \vee b = b \vee a$

(D3) $a \vee a = a$

(D4) $a \vee (ab) = a \cdot (a \vee b) = a$

(D5) $a \vee (bc) = (a \vee b) \cdot (a \vee c)$

(D6) $D\big(x(a \vee b)\big) = D(xa) \vee D(xb)$

(D7) $ax = ay$ and $bx = by$ imply $(a \vee b)x = (a \vee b)y$

This operation on $D(S)$ can then be extended into an operation on the whole of S, by setting $x \vee y := D(x) \vee D(y)$ for any two $x, y \in S$.

The domain join models the set-theoretic union of identity restrictions, so the existing facts about sets establish the soundness of the first five axioms. Axioms (D6) and (D7) are required in later parts of this paper. Their soundness is also evident using our functional interpretation.

In fact, (D6) implies the following law involving only domain elements:

(D6') $\forall a, b, c \in D(S) \ : \ a(b \vee c) = ab \vee ac$

Using (D1) to (D5), together with (D6'), it follows that $D(S)$ forms a distributive lattice; indeed $(D(S) \ ; \ \cdot \ , \vee, 1, 0)$ is a bounded distributive lattice. This structure on $D(S)$ is the basis of our representation theorem in the rest of this section.

3.3 Determinative Pairs

To prove that the axioms are complete, we build an embedding from the abstract algebra to a concrete one, and such an embedding is called a representation. In our present conmbox, we map our abstract algebra of programs and tests to a concrete algebra of partial functions and partial predicates.

Following the approach by Jackson and Stokes in [11], we utilize the technique of determinative pairs, first introduced and much used by Schein.

From this point on, let S denote a semigroup.

Definition 3.3 *A determinative pair (abbreviated as* detpair*) on S is $\langle \epsilon, W \rangle$, where*

- *ϵ is a right congruence on S, and*
- *$W \subset S$ is a union of ϵ-classes, and is also a right ideal.*

The detpair induces a homomorphism ϕ from S to the semigroup of partial functions on $(S \setminus W)/\epsilon$. For all $x \in S$, $x\phi := \psi_x$, where for all $\overline{s} \in (S \setminus W)/\epsilon$, we define

$$\overline{s}\psi_x := \begin{cases} \overline{sx} \text{ if } sx \notin W; \\ \text{undefined otherwise.} \end{cases}$$

It is well known and in any case routine to verify that ϕ is a semigroup homomorphism which preserves 1 and 0.

In order to preserve D and $\vee$ and make the homomorphism injective, we need to use a specific detpair with certain properties. It turns out that a detpair family closely related to that used in [11] suits our purpose.

Recall that at the end of Section 3.2, we showed that the semigroup operation $\cdot$ behaves like the 'meet' on $D(S)$. Hence this naturally induces a partial order: for all $x, y \in D(S)$, $x \leq y$ if and only if $xy = x$. Now this partial order allows us to speak of filters in $D(S)$.

Definition 3.4 *Let F be a proper filter of $D(S)$. Define a binary relation θ_F on S by $x \; \theta_F \; y \iff \exists \mathsf{e} \in F : \mathsf{e}x = \mathsf{e}y$. In addition, define $W_F := \{a \in S \mid D(a) \notin F\}$.*

Then, it is routine to verify that $\langle \theta_F, W_F \rangle$ forms a detpair; see also [11].

Notation 3.5 *For the purpose of clarity, we adopt the following convention. Given a left restriction semigroup S and a filter $F \subseteq D(S)$,*

- *x, y, z and s, t denote general elements of S,*
- *a, b, c denote elements of $D(S)$, and*
- *$\mathsf{e}, \mathsf{f}, \mathsf{g}$ denote elements of F.*

Definition 3.6 *Let $s, t \in S$ with $s \neq t$. Then, an (s, t)-separating equivalence relation σ is one that satisfies $(s, t) \notin \sigma$.*

In Definition 3.4, the right congruence is specified by a filter. To ensure that a right congruence specified in this way using a filter is (s,t)-separating, we are led to the following definition.

Definition 3.7 *Let $s,t \in S$ with $s \neq t$. Then, a filter F in $D(S)$ is (s,t)-separating when*

- *$D(s) \in F$, and*
- *$\neg \exists \mathtt{e} \in F : \mathtt{e}s = \mathtt{e}t$.*

It immediately follows that given an (s,t)-separating filter F, the corresponding θ_F is an (s,t)-separating right congruence.

For conciseness, an (s,t)-separating right congruence or filter will be simply referred to as separating, when no ambiguity can arise.

Definition 3.8 *A maximally (s,t)-separating filter is an (s,t)-separating filter which is not properly contained in any other (s,t)-separating filter.*

The passage preceding [11, Lemma 2.4] shows that such maximally separating filters do exist. The next theorem demonstrates the importance of maximally separating.

Lemma 3.9 *Let $s,t \in S$ with $s \not\leq t$, and $F \subset D(S)$ a maximally (s,t)-separating filter. Then, F is prime.*

Proof. Let $a, b \in D(S)$, and assume for contradiction that $a \vee b \in F$ with $a \notin F$ and $b \notin F$. Since $a \notin F$, we can consider $G_a := \uparrow\{a\mathtt{f} \mid \mathtt{f} \in F\}$. According to [17, Exercise 2.23], G_a is a filter which properly contains a and F. Hence, G_a cannot be a maximally (s,t)-separating filter because F already is one, and $F \subset G_a$. This means that there exists some $\mathtt{g}_a \in G_a$ such that $\mathtt{g}_a s = \mathtt{g}_a t$. Moreover, $\mathtt{g}_a \geq a\mathtt{f}_a$ for some $\mathtt{f}_a \in F$, so $a\mathtt{f}_a = a\mathtt{f}_a\mathtt{g}_a$, and

$$\begin{aligned}(a\mathtt{f}_a)s &= (a\mathtt{f}_a\mathtt{g}_a)s \\ &= a\mathtt{f}_a(\mathtt{g}_a s) \\ &= a\mathtt{f}_a(\mathtt{g}_a t) \\ &= (a\mathtt{f}_a\mathtt{g}_a)t \\ &= (a\mathtt{f}_a)t.\end{aligned}$$

Similarly for b, we can obtain $(b\mathtt{f}_b)s = (b\mathtt{f}_b)t$ for some $\mathtt{f}_b \in F$.

Then, multiplying both sides of $(a\mathtt{f}_a)s = (a\mathtt{f}_a)t$ by $\mathtt{f}_b$, and dually for b, we get

$$\begin{aligned}(a\mathtt{f}_a\mathtt{f}_b)s &= (a\mathtt{f}_a\mathtt{f}_b)t\\ (b\mathtt{f}_a\mathtt{f}_b)s &= (b\mathtt{f}_a\mathtt{f}_b)t.\end{aligned}$$

Now, consider $a\mathtt{f}_a\mathtt{f}_b \vee b\mathtt{f}_a\mathtt{f}_b$. By distributivity of (D6'), this equals to $(a \vee b)\mathtt{f}_a\mathtt{f}_b$, which is in F because all of $(a \vee b)$, $\mathtt{f}_a$ and $\mathtt{f}_b$ are in F.

However, (D7) implies that $\big((a \vee b)\mathtt{f}_a\mathtt{f}_b\big)s = \big((a \vee b)\mathtt{f}_a\mathtt{f}_b\big)t$, and this contradicts our assumption of F being (s,t)-separating.

Therefore, given $a \vee b \in F$, at least one of a and b must be in F, so F is indeed a prime filter. $\square$

3.4 The Embedding

Let F be a prime filter of $D(S)$. By Definition 3.3, this F induces a determinative pair, which then gives rise to a semigroup homomorphism from S into $\mathcal{P}(X_F)$ for some set X_F. Define this homomorphism to take every $x \in S$ to ψ_x^F, where

$$\forall \overline{z} \in X_F \;:\; \overline{z}\psi_x^F := \begin{cases} \overline{zx} \text{ if } D(zx) \in F; \\ \text{undefined otherwise.} \end{cases}$$

Then, define Ψ_x to be $\bigcup_F \psi_x^F$, the union of all ψ_x^F where F ranges through all the different prime filters.

An intuitive picture of this potentially obscure concept is the following. Given a prime filter F, every $x \in S$ is taken to ψ_x^F, which is a set of maplets on a set of blocks X_F. Given another prime filter G, every $x \in S$ is taken to ψ_x^G, which is another set of maplets on another set of blocks X_G.

By "pasting" the functions $x \mapsto \psi_x^F$ and $x \mapsto \psi_x^G$ together, the resulting function is one that takes every $x \in S$ to $\psi_x^F \cup \psi_x^G$, which is itself a set of maplets on the combined set of blocks $X_F \cup X_G$.

This combined set of maplets on $X_F \cup X_G$ remains a valid function because:

1. each of ψ_x^F and ψ_x^G is itself a function;

2. X_F and X_G are disjoint.

When we need to show that a statement holds for Ψ_x, it is enough to show that the same statement holds for ψ_x^G, where G is any arbitrary prime filter of $D(S)$.

Finally, set F to range over all the prime filters of $D(S)$, define our ultimate embedding candidate $\Phi : S \to \bigcup_F \mathcal{P}(X_F)$ by letting it map every $x \in S$ to Ψ_x.

Theorem 3.10 *Φ is a homomorphism of a semigroup with $\vee$, D, 1 and 0.*

Proof. The following need to be shown. For all $x, y \in S$:

(i) $x\Phi \cdot y\Phi = (xy)\Phi$;

(ii) $x\Phi \vee y\Phi = \big(x \vee y\big)\Phi$;

(iii) $D(x\Phi) = \big(D(x)\big)\Phi$;

(iv) 1Φ is the full identity function on the base set $\bigcup_F \mathcal{P}(X_F)$;

(v) 0Φ is the empty function on the base set.

The statements **(i)**, **(iv)** and **(v)** are true because Φ is made of smaller detpair mappings, each of which is a homomorphism as justified by the discussion after Definition 3.3. We proceed to prove the remaining points.

(iii) To prove that $\psi^F_{D(x)} = D(\psi^F_x)$, firstly their domains must be equal. This is true because

$$\begin{aligned}\overline{z} \in dom\big(\psi^F_{D(x)}\big) &\iff D\big(zD(x)\big) \in F \\ &\iff D(zx) \in F \\ &\iff \overline{z} \in dom\big(\psi^F_x\big) = dom\big(D(\psi^F_x)\big).\end{aligned}$$

Then, note that $D(\psi^F_x)$ is an identity restriction; what does $\psi^F_{D(x)}$ do as a function? Let $\psi^F_{D(x)}$ be defined at $\overline{z} \in X_F$; then $\overline{z}\psi^F_{D(x)} = \overline{zD(x)}$ and $D(zx) \in F$. Now, focusing on the relationship between $\overline{z}$ and $\overline{zD(x)}$,

$$\begin{aligned}D(zx) \cdot z &= zD(x) && \text{(R5)} \\ &= D\big(zD(x)\big) \cdot zD(x) && \text{(R1)} \\ &= D(zx) \cdot zD(x). && \text{(R9)}\end{aligned}$$

The above reasoning shows that $z \ \theta_F \ zD(x)$, so $\overline{z} = \overline{zD(x)}$. As $\overline{z}\psi^F_{D(x)} = \overline{zD(x)} = \overline{z}$, $\psi^F_{D(x)}$ is an identity restriction.

Therefore, since $\psi^F_{D(x)}$ and $D(\psi^F_x)$ have the same domain and are both identity restrictions, these two functions are indeed equal.

(ii) Firstly, recall that $x \vee y$ is defined to be $D(x) \vee D(y)$ when x and y are arbitrary elements in S. Hence, let a denote $D(x)$ and b denote $D(y)$, where both $a, b \in D(S)$.

Then, the left-hand side of (ii) can be rewritten as

$$\begin{aligned} x\Phi \vee y\Phi &= D\big(x\Phi\big) \vee D\big(y\Phi\big) \\ &= \big(D(x)\big)\Phi \vee \big(D(y)\big)\Phi && \text{(item (iii) above)} \\ &= a\Phi \vee b\Phi. \end{aligned}$$

Similarly, the right-hand side of (ii) can be rewritten as

$$\begin{aligned} (x \vee y)\Phi &= \big(D(x) \vee D(y)\big)\Phi \\ &= (a \vee b)\Phi. \end{aligned}$$

Therefore, in order to prove **(ii)**, we must show that $\forall a, b \in D(S)$: $a\Phi \vee b\Phi = (a \vee b)\Phi$, which in turn requires us to prove $\psi^F_{a\vee b} = \psi^F_a \cup \psi^F_b$.

Since both sides of this latest statement are identity restrictions, it is enough to show that the domains of both sides are equal.

$$\begin{aligned} \overline{z} \in dom\big(\psi^F_{a\vee b}\big) &\iff D\big(z(a \vee b)\big) \in F \\ &\iff D\big(za\big) \vee D\big(zb\big) \in F. && \text{(D4)} \end{aligned}$$

Here, F is a prime filter, so $D\big(za\big) \cup D\big(zb\big) \in F$ means that either $D\big(za\big) \in F$ or $D\big(zb\big) \in F$. Therefore, either $\overline{z} \in dom\big(\psi^F_a\big)$ or $\overline{z} \in dom\big(\psi^F_b\big)$; in other words,

$$\begin{aligned} \overline{z} &\in dom\big(\psi^F_a\big) \cup dom\big(\psi^F_b\big) \\ &= dom\big(\psi^F_a \cup \psi^F_b\big). \end{aligned}$$

The above argument works in the reverse direction too, so we can now conclude that $dom(\psi^F_{a\vee b}) = dom(\psi^F_a \cup \psi^F_b)$, and hence $\psi^F_{a\vee b} = \psi^F_a \cup \psi^F_b$. □

Theorem 3.11 Φ *is injective.*

Proof. Let $s, t \in S$ with $s \not\leq t$ (and hence $s \neq t$). Also, let F be a prime filter which separates s and t. Use this F to define ψ^F_x where $x \in S$. It will be shown that for this F, $\psi^F_s \neq \psi^F_t$, and hence $s\Phi \neq t\Phi$.

Firstly, $D(s) \in F$ by Definition 3.7, and $D\big(D(s) \cdot s\big) = D(s) \in F$, so ψ^F_s is defined at $\overline{D(s)}$, and hence $\psi^F_s\big(\overline{D(s)}\big) = \overline{s}$.

Now, we show that ψ^F_t is not equal to ψ^F_s.

Case 1: Assume $D(t) \in F$, and note that

$$\begin{aligned} D\big(D(s) \cdot t\big) &= D\big(D(s) \cdot D(t)\big) && \text{(R9)} \\ &= D(s) \cdot D(t) && \text{(R6)} \\ &\in F, \end{aligned}$$

so ψ_t^F is defined at $\overline{D(s)}$. Then, are $\psi_s^F\big(\overline{D(s)}\big)$ and $\psi_t^F\big(\overline{D(s)}\big)$ equal? Assume (for contradiction) that they are, and this leads to $\overline{s} = \overline{D(s)t}$. By the definition of our congruence θ_F, there would be an $\mathtt{e} \in F$ which satisfies $\mathtt{e} \cdot s = \mathtt{e} \cdot D(s)t$. However, $\mathtt{e} \cdot s = \mathtt{e} \cdot D(s)s$, so we have $\mathtt{e}D(s)s = \mathtt{e}D(s)t$, where $\mathtt{e}D(s) \in F$ because both $\mathtt{e}$ and $D(s)$ are in F. This would contradict the premise that F is an (s,t)-separating filter, so it must be the case that $\psi_s^F \neq \psi_t^F$ as they disagree on $\overline{D(s)}$.

Case 2: Assume $D(t) \notin F$.
The property of filters implies that any element "smaller" than $D(t)$ cannot be in F. We know from the Case 1 that $D\big(D(s)t\big) = D(s)D(t) \leq D(t)$, so $D\big(D(s)t\big)$ cannot be in F, and hence ψ_t^F is undefined at $\overline{D(s)}$. Therefore, $\psi_s^F \neq \psi_t^F$.

In conclusion, as ψ_x^F is a subset of Ψ_x for all $x \in S$, the above has shown that $\psi_s^F \neq \psi_t^F$, which implies $s\Phi \neq t\Phi$. Therefore, Φ is injective. □

Combining Theorems 3.10 and 3.11 gives us the following.

Corollary 3.12 *The above map Φ is an embedding from the algebra of programs (*i.e. *a semigroup with $\vee$, D, 1 and 0) into a same algebra of partial functions (*i.e. *$\mathcal{P}(X)$ for some set X).*

This final theorem is the culmination of this section, and it asserts that there is a representation of the algebra of programs. On this important note, this section concludes.

4 Modeling the Tests

This section brings the tests into the picture.

A possibly non-halting test is modeled by a partial predicate, which is a partial function that maps from the state set to the truth set $\{T, F\}$. The undefined part represents the places where the test does not halt.

We use the representation of programs Φ from Section 3 to construct an embedding from the algebra of tests K into an algebra of disjoint pairs.

To express each test as a disjoint pair, we must identify its true and false parts. The true part is $\alpha_T := \alpha[1,0]$, which is an element of $D(S)$ by Axiom (T5). Similarly, the false part of α is $\alpha_F := \alpha[0,1]$. These true and false parts satisfy the following axioms. For all $\alpha, \beta \in K$:

(T1) $(\alpha \vee \beta)[1,0] = \alpha[1,0] \vee \beta[1,0]$
$(\alpha \wedge \beta)[1,0] = \alpha[1,0] \cdot \beta[1,0]$

(T2) $(\alpha \vee \beta)[0,1] = \alpha[0,1] \cdot \beta[0,1]$
$(\alpha \wedge \beta)[0,1] = \alpha[0,1] \vee \beta[0,1]$

(T3) $(\neg\alpha)[s,t] = \alpha[t,s]$ for all $s, t \in S$

(T4) $\alpha[1,0] \cdot \alpha[0,1] = 0$

(T5) $\alpha[1,0], \alpha[0,1] \in D(S)$

(T6) $\alpha[1,0] = \beta[1,0]$ and $\alpha[0,1] = \beta[0,1]$ imply that $\alpha = \beta$

These axioms are all sound under our functional interpretation: (T1) to (T3) follow from the discussion at the end of Section 2.3, (T4) ensures that $\alpha[1,0]$ and $\alpha[0,1]$ do form a disjoint pair, (T5) asserts that these parts are indeed domain elements, and (T6) enables us to uniquely identify a test by its two parts.

Define $\Gamma : K \to D(S)^\wedge$ as follows[1] :

$$\forall \alpha \in K : \alpha\Gamma := \big\langle \alpha_T\Phi, \alpha_F\Phi \big\rangle.$$

Axiom (T4) guarantees that the right-hand side is indeed a disjoint pair.

Theorem 4.1 *Γ is an embedding.*

Proof. Firstly, we show that $(\alpha \wedge \beta)\Gamma = \alpha\Gamma \wedge \beta\Gamma$. The left-hand side of this equation is $\big\langle \big((\alpha\wedge\beta)_T\big)\Phi, \big((\alpha\wedge\beta)_F\big)\Phi \big\rangle$, and the right-hand side is $\big\langle (\alpha_T)\Phi, (\alpha_F)\Phi \big\rangle \wedge \big\langle (\beta_T)\Phi, (\beta_F)\Phi \big\rangle = \big\langle (\alpha_T)\Phi \cdot (\beta_T)\Phi, (\alpha_F)\Phi \cup (\beta_F)\Phi \big\rangle$. In order to show that the two sides are the same, we prove that both of their components are equal.

For the first component, we show $(\alpha_T)\Phi \cdot (\beta_T)\Phi = \big((\alpha\wedge\beta)_T\big)\Phi$, which amounts to $\psi^G_{\alpha_T} \cap \psi^G_{\beta_T} = \psi^G_{(\alpha\wedge\beta)_T}$, where G is any prime filter in $D(S)$. Suppose that $\overline{x} \in dom\big(\psi^G_{\alpha_T} \cap \psi^G_{\beta_T}\big)$. This is equivalent to

$$\begin{aligned} \overline{x} \in dom\big(\psi^G_{\alpha_T}\big) \text{ and } \overline{x} \in dom\big(\psi^G_{\alpha_T}\big) &\iff D(x\alpha_T) \in G \text{ and } D(x\beta_T) \in G \\ &\iff D(x\alpha_T) \cdot D(x\beta_T) \in G. \end{aligned}$$

[1]We use the letter Γ to stand for "guards", which is another name for "tests". We choose not to use the Greek "Tau" to avoid confusion with the truth value T.

But, $D(x\alpha_T) \cdot D(x\beta_T) = D(x\alpha_T\beta_T) = D\big(x \cdot (\alpha \wedge \beta)_T\big)$, so

$$\begin{aligned} D(x\alpha_T) \cdot D(x\beta_T) \in G &\iff D\big(x \cdot (\alpha \wedge \beta)_T\big) \in G \\ &\iff \overline{x} \in \mathit{dom}\big(\psi^G_{\alpha\wedge\beta}\big). \end{aligned}$$

The above reasoning established that $\mathit{dom}\big(\psi^G_{\alpha_T} \cap \psi^G_{\beta_T}\big) = \mathit{dom}\big(\psi^G_{(\alpha\wedge\beta)_T}\big)$, and since these functions are all identity restrictions, it follows that $\psi^G_{\alpha_T} \cap \psi^G_{\beta_T} = \psi^G_{(\alpha\wedge\beta)_T}$.

On the other hand, for the second component, we show $\psi^G_{\alpha_F} \cup \psi^G_{\beta_F} = \psi^G_{(\alpha\vee\beta)_F}$. Suppose that $\overline{x} \in \mathit{dom}\big(\psi^G_{(\alpha\vee\beta)_F}\big)$, which is equivalent to $D\big(x \cdot (\alpha \vee \beta)_F\big) \in G$.

$$\begin{aligned} D\big(x \cdot (\alpha \vee \beta)_F\big) &= D\big(x \cdot (\alpha_F \vee \beta_F)\big) && \text{(T2)} \\ &= D(x\alpha_F) \vee D(x\beta_F), && \text{(D5)} \end{aligned}$$

so $D(x\alpha_F) \vee D(x\beta_F) \in G$. Since G is prime, either $D(x\alpha_F) \in G$ or $D(x\beta_F) \in G$, which means that $\overline{x} \in \mathit{dom}\big(\psi^G_{\alpha_F}\big) \cup \mathit{dom}\big(\psi^G_{\beta_F}\big)$. The above showed that $\mathit{dom}\big(\psi^G_{(\alpha\wedge\beta)_F}\big) = \mathit{dom}\big(\psi^G_{\alpha_F}\big) \cup \mathit{dom}\big(\psi^G_{\beta_F}\big)$, and as these functions are all identity restrictions, it follows that $\psi^G_{\alpha_F} \cup \psi^G_{\beta_F} = \psi^G_{(\alpha\wedge\beta)_F}$.

Now that the case of $(\alpha \wedge \beta)\Gamma$ is done, the dual case of $(\alpha \vee \beta)\Gamma$ can be established in a similar way.

For the negation $(\neg\alpha)\Gamma$, note that $(\neg\alpha)\Gamma = \big\langle \psi^G_{(\neg\alpha)_T}, \psi^G_{(\neg\alpha)_F} \big\rangle$ and $\neg(\alpha\Gamma) = \big\langle \psi^G_{\alpha_F}, \psi^G_{\alpha_T} \big\rangle$, so we need to show $\psi^G_{\alpha_F} = \psi^G_{(\neg\alpha)_T}$. Again, as both of these are identity restrictions, it is enough to consider their domains. Starting from the right-hand side, $\overline{x} \in \mathit{dom}(\psi^G_{(\neg\alpha)_T})$ if and only if $D\big(x \cdot (\neg\alpha)_T\big) \in G$. However, $x \cdot (\neg\alpha)_T = x \cdot \alpha_F$, so $D\big(x \cdot \alpha_F\big) \in G$, which is equivalent to $\overline{x} \in \mathit{dom}(\psi^G_{\alpha_F})$. Therefore, it is indeed true that $\psi^G_{\alpha_F} = \psi^G_{(\neg\alpha)_T}$.

As for the constants T and F:

$$\begin{aligned} T\Phi &= \big\langle 1\Phi, 0\Phi \big\rangle = \big\langle 1, 0 \big\rangle \\ F\Phi &= \big\langle 0\Phi, 1\Phi \big\rangle = \big\langle 0, 1 \big\rangle, \end{aligned}$$

so these are correctly represented too.

Finally, we prove that the test homomorphism Γ is injective. Let $\alpha, \beta \in K$ such that $\alpha\Gamma = \beta\Gamma$. Then, $\big\langle \alpha_T\Phi, \alpha_F\Phi \big\rangle = \big\langle \beta_T\Phi, \beta_F\Phi \big\rangle$. By the definition of equality of disjoint pairs (Section 2.3), the above equation is equivalent to $\alpha_T\Phi = \beta_T\Phi$ and $\alpha_F\Phi = \beta_F\Phi$. These latest equations only involve programs, and since Φ is a program embedding, it follows that $\alpha_T = \beta_T$ and $\alpha_F = \beta_F$. Now, by axiom (T6), we can conclude that α and β are equal, and hence Φ is a test embedding. □

5 Modeling `if-then-else`

The previous two sections established mappings which correctly represent the programs and the tests. Now, this section shows that these mappings correctly represent `if-then-else` expressions too, once further axioms are added. Each `if-then-else` expression is itself a program, so the relevant mapping to use here is Φ, and Φ applied to $\alpha[s,t]$ is defined to be

$$\alpha[s,t]\Phi := \big(\alpha[1,0]s\big)\Phi \cup \big(\alpha[0,1]t\big)\Phi.$$

This definition is clearly valid, as the partial function $\alpha[s,t]$ is one that behaves like s when α is true, and acts like t when α is false.

The further axioms are the following. For all $\alpha, \beta \in K$ and $s, t \in S$:

(G1) $\alpha[1,0] \cdot s = \alpha[1,0] \cdot \alpha[s,t]$
$\alpha[0,1] \cdot t = \alpha[0,1] \cdot \alpha[s,t]$

(G2) $D\big(\alpha[s,t]\big) = \big(\alpha[1,0] \cdot D(s)\big) \vee \big(\alpha[0,1] \cdot D(t)\big)$

The soundness of these axioms is evident. Turning to completeness, we shall make use of the fact that Φ and Γ represent programs and tests, as well as elements of the form $\alpha_T = \alpha[1,0]$ and $\alpha_F = \alpha[0,1]$, correctly (Corollary 3.12 and Theorem 4.1). We leverage these facts to show that arbitrary elements of the form $\alpha[s,t]$ are also correctly represented.

Now the first equation of (G1) says that, under this correct representation, $\alpha[s,t]$ is the same as s when α is true, while the second says that $\alpha[s,t]$ is the same as t when α is false. In short, (G1) states the behavior of $\alpha[s,t]$ at places where $\alpha_T \cdot s$ and $\alpha_F \cdot t$ are defined. However, based only on (G1), when neither $\alpha_T \cdot s$ nor $\alpha_F \cdot t$ is defined, $\alpha[s,t]$ may still be defined. So the purpose of (G2) is to exclude this possibility by asserting that the domain of $\alpha[s,t]$ is exactly the union of $\alpha_T \cdot s$ and $\alpha_F \cdot t$. Together then, (G1) and (G2) force $\alpha[s,t]$ to be exactly what we want it to be. So the completeness of (G1) and (G2) follows.

6 Modeling H

It remains to give axioms for H. We argue in similar fashion as for the `if-then-else` operation. It is easy to see that the following two laws for H are sound.

(H1) $\big(H(p)\big)[1,0] = D(p)$

(H2) $\big(H(p)\big)[0,1] = 0$

Conversely, given that $H(p)$ is a test, and all else is correctly represented by earlier results, the first laws above force $H(p)$ to be true where p is defined, false otherwise, and undefined nowhere, exactly as required.

7 Final Comments

Pulling together what has gone before, we arrive at Theorem 2.2. Note that this theorem can be expressed without reference to D at all, since $D(p) = \big(H(p)\big)[1,0]$ so all mention of D can be eliminated, and for similar reasons domain joins can be eliminated. Once this is done, we are left with a sound and complete finite set of first-order axioms involving only the operations we began with, namely composition of programs and the distinguished programs 1 and 0, the connectives on tests and the distinguished tests T and F, the halting operation H, and the `if-then-else` operation. Let us call a two-sorted algebra (S, K) with semigroup type S and test type K having this signature and satisfying all the needed laws an *algebra of programs with parallel tests*. It follows that we have described up to isomorphism the concrete algebras of programs with parallel tests (those embeddable in $(\mathcal{P}(X), \mathcal{P}Pred(X))$) as abstract algebras of programs with parallel tests.

We note that our signature may also be augmented to include the *equality test*, considered in some detail in [9] and [11]. Given two programs, the equality test is a partial predicate which is true when the two programs are both defined and equal, false when both are defined but not equal, and undefined when either program is undefined. Under our framework, this operation (and its axioms) turns out to be exactly the same as the *weak comparison* operation considered in [11].

Finally, as noted above, the logic we use for tests in terms of T, F, U is Kleene's logical system K_3 as in [14]. It follows that the algebra of tests K in an algebra of programs with parallel tests (S, K) is a Kleene algebra in the sense of [18]: all the laws for Kleene algebras must follow from our axioms. It would be interesting to attempt to formulate the ideas of this paper using an abstract Kleene algebra of three-valued tests in terms of which the `if-then-else` operation are defined. In the current work, we leverage the fact that $\alpha[1,0]$ and $\alpha[0,1]$ together provide complete information about the test α via Law (T6), but in a signature lacking one or both of $0, 1$, this would not be possible, necessitating a more direct approach to dealing with the tests.

References

[1] John McCarthy. A basis for a mathematical theory of computation. In *Computer programming and formal systems*, pages 33–70. North-Holland, 1963.

[2] Stephen L. Bloom and Ralph Tindell. Varieties of "if-then-else". *SIAM Journal of Computing*, 12(4):677–707, 1983.

[3] Irène Guessarian and José Meseguer. On the axiomatization of "if-then-else". *SIAM Journal of Computing*, 16(2):332–357, 1987.

[4] Don Pigozzi. Equality-test and if-then-else algebras: axiomatization and specification. *SIAM Journal of Computing*, 20(4):766–805, 1991.

[5] Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, 1997.

[6] Jules Desharnais, Bernhard Möller, and Georg Struth. Kleene algebra with domain. *ACM Transactions on Computational Logic*, 7(4):798–833, 2006.

[7] George M. Bergman. Actions of boolean rings on sets. *Algebra Universalis*, 28(2):153–187, 1991.

[8] Ernest G. Manes. Adas and the equational theory of if-then-else. *Algebra Universalis*, 30(3):373–394, 1993.

[9] Marcel Jackson and Timothy Stokes. Semigroups with if-then-else and halting programs. *International Journal of Algebra and Computation*, 19(7):937–961, 2009.

[10] Marcel Jackson and Timothy Stokes. Modal restriction semigroups: towards an algebra of functions. *International Journal of Algebra and Computation*, 21(7):1053–1095, 2011.

[11] Marcel Jackson and Timothy Stokes. Monoids with tests and the algebra of possibly non-halting programs. *Journal of Logical and Algebraic Methods in Programming*, 84(2):259–275, 2015.

[12] Gayatri Panicker, K. V. Krishna, and Purandar Bhaduri. Axiomatization of `if-then-else` over possibly non-halting programs and tests. *International Journal of Algebra and Computation*, 27(3):273–297, 2017.

[13] Gayatri Panicker, K. V. Krishna, and Purandar Bhaduri. Monoids of non-halting programs with tests. *Algebra Universalis*, 79(8):published online: https://doi.org/10.1007/s00012–018–0490–3, 2018.

[14] Stephen Cole Kleene. On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3(4):150–155, 1938.

[15] Valentin S. Trokhimenko. Menger's function systems [Russian]. *Izv. Vysš. Učebn. Zaved. Matematika*, 11(138):71–78, 1973.

[16] Marcel Jackson and Timothy Stokes. An invitation to C-semigroups. *Semigroup Forum*, 62(2):279–310, 2001.

[17] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2002.

[18] Roberto Cignoli. Injective de morgan and kleene algebras. *Proceedings of the American Mathematical Society*, 47(2):269–277468, 1975.

Department of Mathematics and Statistics, The University of Waikato,
Private Bag 3105, Hamilton, New Zealand 3240
tim.stokes@waikato.ac.nz

Department of Computer Science, The University of Auckland,
Private Bag 92019, Auckland, New Zealand 1142
rsu050@aucklanduni.ac.nz