Modification Algebras

G. Ramalingam and Thomas Reps
University of Wisconsin, Madison, WI 53706, USA

Abstract

Given a program base and two of its variants a and b, the problem of program integration is to determine whether the modifications made to base to produce a and b have undesirable semantic interactions, and if there is no such interference, to produce a merged program that incorporates the modifications made in developing either of the variants. Program-modifications can be formalized as functions from the set of programs to itself. This paper develops an algebra of program-modifications, and uses it to establish various results concerning the problem of program-integration and some variants of the problem.

1. Introduction

The need to integrate several versions of a program into a common one arises frequently, but it is a tedious and time consuming task to merge programs by hand. The program-integration algorithm proposed by Horwitz, Prins, and Reps [1] —referred to hereafter as the HPR algorithm—provides a way to create a semantics-based tool for integrating a base program with two or more variants. The integration algorithm is based on the assumption that any change in the behavior, rather than the text, of a program variant is significant and must be preserved in the merged program. An integration system based on this algorithm will determine whether the variants incorporate interfering changes, and, if they do not, create an integrated program that includes all changes as well as all features of the base program that are preserved in all variants. Various applications have been envisioned for a semantics-based integration tool [1].

This paper is a continuation of two earlier studies of the algebraic properties of the program-integration operation. (A simple example of an algebraic property — which a specific integration algorithm might or might not have — is that of associativity. In this context associativity means: “If three variants of a given base are to be integrated by a pair of two-variant integrations, the same result is produced no matter which two variants are integrated first.”) One earlier work that studied the algebraic properties of program integration ([6]) formalized the HPR integration algorithm as an operation in a Brouwerian algebra [2]. Recently, a new algebraic structure in which integration can be formalized, called fm-algebra, was introduced [4]. In fm-algebra, the notion of

This work was supported in part by an IBM graduate fellowship, by a David and Lucile Packard Fellowship for Science and Engineering, by the National Science Foundation under grants DCR-8552602 and CCR-9100424, by the Defense Advanced Research Projects Agency, monitored by the Office of Naval Research under contract N00014-88-K-0590, as well as by a grant from the Digital Equipment Corporation.
integration derives from the concepts of a program-modification and an operation for combining modifications. Thus, while the earlier work reported in [6] concerned a homogeneous algebra of programs, the later approach concerns a heterogeneous algebra of programs and program-modifications. Both these frameworks are reviewed in Section 2.

The aim of this paper is to develop the algebraic structure of the domain of program-modifications further, in an attempt to explore the relationships between the problem of program-integration and various related problems, and the relationships between different solutions to these problems. In Section 3, we investigate one particular model of \( fm \)-algebra that represents the HPR algorithm. We also define several auxiliary operators that have a simple intuitive meaning, many of them being defined in terms of function composition alone. We study the properties of these operators and use them in Section 4 to achieve our goal: determining the relationships between different solutions to program-integration and related problems. Due to space constraints, we do not include all the results obtained along these lines. A more complete exposition can be found in [3].

The potential benefits of the theory outlined are actually three-fold: (1) It allows one to understand the fundamental algebraic properties of integration—laws that express the “essence of integration.” Such laws allow one to reason formally about the integration operation; (2) It provides knowledge that is useful for designing alternative integration algorithms whose power and scope are beyond the capabilities of current algorithms; (3) Because such a theory formalizes certain operations that are more primitive than the integration operation, an implementation of these primitive operations can form the basis for a more powerful program-manipulation system than one based on just the integration operation.

2. Previous Work

2.1. The Brouwerian algebraic framework

Reps [6] shows that the set of all programs written in a simple language may be embedded in an algebraic structure called double Brouwerian algebra, and establishes the correspondence between the HPR program-integration algorithm and a ternary operation of this double Brouwerian algebra. We review these concepts below.

**Definition 2.1.** A double Brouwerian algebra [2] is an algebra \((P, \sqcup, \sqcap, \bot, \top)\) where

(i) \((P, \sqcup, \sqcap, \bot, \top)\) is a lattice with join operator \(\sqcup\), meet operator \(\sqcap\), least element \(\bot\), and greatest element \(\top\). Let \(\sqsubseteq\) denote the corresponding partial order.

(ii) For all \(a, b,\) and \(c\) in \(P\), \((a \div b) \sqsupseteq c\) iff \(a \sqsubseteq (b \sqcap c)\).

(iii) For all \(a, b,\) and \(c\) in \(P\), \((a \div b) \sqsubseteq c\) iff \(a \sqsubseteq (b \sqcap c)\).

**Definition 2.2.** The integration operator of a Brouwerian algebra is the ternary operator defined as follows:

\[
a [\text{base }] b \triangleq (a \div b) \sqcap (a \sqcap \text{base}) \sqcup (b \div \text{base})
\]

We now introduce the notion of a partial double Brouwerian algebra, which will prove to be useful later on.
Definition 2.3. A partial algebra \((T, \sqcup_1, \sqcap_1, \neg_1, \div_1)\) is said to be a partial double Brouwerian algebra (abbreviated PDBA) if it can be extended to a double Brouwerian algebra \((B, \sqcup_2, \sqcap_2, \neg_2, \div_2, \perp)\) such that \(T\) is a downwards-closed subset of \(B\), and each of the operations \(\sqcup_1, \sqcap_1, \neg_1, \) and \(\div_1\) is the restriction of the corresponding operations \(\sqcup_2, \sqcap_2, \neg_2, \) and \(\div_2\) to \(T\). (A subset \(X\) of \(B\) is said to be downwards-closed if \(y \sqsubseteq_2 x\) and \(x \in X\) imply that \(y \in X\), where \(\sqsubseteq_2\) denotes the partial ordering of \(B\).)

Note that a partial double Brouwerian algebra satisfies the various algebraic properties of double Brouwerian algebra as long as all the relevant expressions are defined. The reader is referred to [2, 5-7] for a discussion of the algebraic laws that hold in Brouwerian algebras. The following proposition follows directly from the previous definition.

Proposition 2.4. A PDBA is consistently complete: two elements of a PDBA have a least upper bound iff they have an upper bound.

2.2. The fm-algebraic framework

The goal of program-integration is to merge or combine changes made to some program. The concept of a “change made to a program” or a program-modification was formalized in [4] and made use of in providing an alternative definition of integration. We briefly review this formalism.

Definition 2.5. A functional-modification algebra (abbreviated fm-algebra) is an algebra \((P, M, \Delta, +)\) where \(M \subseteq P \rightarrow P\), and \(\Delta\) and \(+\) are operations with the following functionalities:

\[ \Delta : P \times P \rightarrow M \]
\[ + : M \times M \rightarrow M. \]

For our purposes, we may interpret the components of an fm-algebra as follows. \(P\) denotes the set of programs; \(M\) denotes a set of allowable program-modification operations: \(m(\text{base})\) is the program obtained by performing modification \(m\) on program \(\text{base}\); \(\Delta(a, \text{base})\) yields the program-modification performed on \(\text{base}\) to obtain \(a\); operation \(m_1 + m_2\) combines two modifications \(m_1\) and \(m_2\) to give a new modification. Of course, there need not be a unique modification that yields \(a\) when applied to \(\text{base}\). The precise meaning of \(\Delta(a, \text{base})\) will be discussed in the next section.

Definition 2.6. The 2-variant integration operator \(\_\[\_\]\_\): \(P \times P \times P \rightarrow P\) of an fm-algebra is defined as follows:

\[ a[\text{base}]b = \text{def} (\Delta(a, \text{base}) + \Delta(b, \text{base})) (\text{base}). \]

Given a double Brouwerian algebra \((P, \sqcup, \sqcap, \neg, \div, \perp, \top)\), an fm-algebra \((P, M, \Delta, +)\) may be defined as in [4] so that the \(\_\[\_\]\_\) operator of the fm-algebra is the same as the \(\_\[\_\]\_\) operator of the Brouwerian algebra. The goal of this paper is an extended study of this particular model. The tools we use are certain other operators that we define on the set of modifications. Sections 3 and 4 give a brief overview of our results.
3. An algebra of modifications

Let $\mathcal{P} = (P, \sqcap, \sqcup, \ast, \circ, \top)$ be a double Brouwerian algebra. The elements of $P$ will be referred to as programs. The letters $x, y, z, a, b, c$ and base will typically denote elements of $P$. The set of program-modifications of interest, $M$, is given by the following definition.

**Definition 3.1.** $M = \{ \lambda z. (z \sqcap y) \sqcup x | x, y \in P \}$.

The word modification will denote an element of $M$. The symbols $m$ and $m_i$ will typically denote elements of $M$. Informally, we may think of modification $\lambda z. (z \sqcap y) \sqcup x = (\lambda z. z \sqcup x) \circ (\lambda z. z \sqcap y)$ as consisting of two components: $\lambda z. z \sqcup x$, which represents an addition of new program components, and $\lambda z. z \sqcap y$, which represents deletion of certain program components.

**Definition 3.2.** Let $\overline{\mathcal{P}} = (\overline{P}, \sqcap, \sqcup, \ast, \circ, \top)$ denote the dual of $\mathcal{P}$. Let $\mathcal{P} \times \overline{\mathcal{P}}$ denote the product algebra of $\mathcal{P}$ and $\overline{\mathcal{P}}$. Both $\mathcal{P}$ and $\mathcal{P} \times \overline{\mathcal{P}}$ are double Brouwerian algebras.

**Definition 3.3.** $S$, a subset of $P \times P$, is defined as follows:

$$S = \{ (x, y) \in P \times P | x \sqsubseteq y \}$$

Let $S$ denote the corresponding partial double Brouwerian algebra induced by $S$ (as a set of elements of $\mathcal{P} \times \overline{\mathcal{P}}$).

**Definition 3.4.** A function $\rho$ from $S$ to $M$ is defined as follows:

$$\rho((x,y)) = \lambda z. (z \sqcap y) \sqcup x$$

**Proposition 3.5.** $\rho$ is a bijection from $S$ to $M$.

**Proof.** It can be verified easily that $\rho^{-1}(m) = (m(\bot), m(\top))$. □

We use the bijection $\rho$ to represent modifications as ordered pairs. We will abbreviate $\rho((x,y))$ as $\langle x, y \rangle$. In particular, any reference to a modification $\langle x, y \rangle$ automatically implies that $x \sqsubseteq y$.

Since a Brouwerian algebra is a distributive lattice, $\lambda z. (z \sqcap y) \sqcup x = \lambda z. (z \sqcup x) \sqcap y$ whenever $x \sqsubseteq y$. Let $\circ$ denote function composition. Thus, $m_1 \circ m_2$ denotes the function $\lambda x. m_1(m_2(x))$. The following properties are easily verified.

**Proposition 3.6.**

(a) $M$ is closed with respect to $\circ$. In particular,

$$\langle x_1, y_1 \rangle \circ \langle x_2, y_2 \rangle = \langle x_1 \sqcup (x_2 \sqcap y_1), y_1 \sqcap (y_2 \sqcup x_1) \rangle.$$

(b) Let $I$ denote the modification $\langle \bot, \top \rangle$. Then, $(M, \circ, I)$ is a monoid: i.e., $\circ$ is associative, and $I$ is the identity with respect to $\circ$.

**Proposition 3.7.** $m_1 \circ m_2 \circ m_1 = m_1 \circ m_2$ for all $m_1, m_2$ in $M$.

**Corollary 3.8.** $\circ$ is idempotent in $M$.

**Definition 3.9.** Define a binary relation $\leq$ on $M$ as follows:

$$m_1 \leq m_2 = \text{def} m_1 \circ m_2 = m_2 = m_2 \circ m_1$$

The relation $\leq$ is intended to capture the notion of subsumption among modifications. Thus, modification $m_1$ is subsumed by (or “is contained in”, “is smaller than”) modification $m_2$ if performing $m_1$ and $m_2$ in either order does nothing more than performing $m_2$. The following proposition establishes some simple properties of this relation.
Proposition 3.10.

(a) \( m_1 \leq m_2 \) iff \( m_1 \circ m_2 = m_2 \) iff \( \exists m, m_2 = m_1 \circ m \).

(b) \( \leq \) is a partial order, with \( I \) as the least element.

(c) \( m_1 \circ m_2 \geq m_1 \).

We will often make use of Proposition 3.10(a), without specifically referring to it, in showing that \( m_1 \leq m_2 \) for some particular \( m_1 \) and \( m_2 \).

Proposition 3.11. \( \rho \) (see Definition 3.4) is a poset isomorphism from \((M, \leq)\) to \( S \).

It follows from the above proposition that \( M \) is itself a partial double Brouwerian algebra with respect to the partial order \( \leq \). We denote the corresponding meet and (partial) join operators \( \sqcap \) and \( \sqcup \), and the pseudo-difference and (partial) quotient operators \( \vdash \) and \( \div \). The above proposition implies that \( <x_1, y_1> \leq <x_2, y_2> \) iff \( (x_1 \sqcap x_2) \) and \( (y_1 \sqcup y_2) \). (Note that the ordering on the respective second components is the dual of the ordering on the respective first components.) Hence, for instance, \( <x_1, y_1> \sqcap <x_2, y_2> \) is \( <x_1 \sqcap x_2, y_1 \sqcup y_2> \).

Now, if \( m_1 \leq m_2 \), then we may think of modification \( m_1 \) as being a “part” of modification \( m_2 \). We know that the meet of any two modifications \( m_1 \) and \( m_2 \) exists, since \( M \) is a PDBA. We may think of \( m_1 \sqcap m_2 \) as the modification that is “common” to the two modifications \( m_1 \) and \( m_2 \). However, two modifications may or may not have a least upper bound. We first formalize a notion of conflict between modifications, and prove that to establish certain results concerning the least upper bound of two modifications.

Definition 3.12. Modifications \( m_1 \) and \( m_2 \) are said to conflict or be incompatible with each other if \( m_1 \circ m_2 \neq m_2 \circ m_1 \). Otherwise, they are said to commute or be compatible with each other.

Proposition 3.13.

(a) \( m_1 \) and \( m_2 \) are compatible iff \( m_1 \) and \( m_2 \) have a least upper bound iff \( m_1 \) and \( m_2 \) have an upper bound, in which case \( m_1 \sqcap m_2 = m_1 \circ m_2 \).

(b) If \( m_3 \leq m_1 \) and \( m_4 \leq m_2 \) and if \( m_1 \) and \( m_2 \) are compatible then \( m_3 \) and \( m_4 \) are compatible.

(c) If \( m_1 \geq m_2 \) then \( m \circ m_1 \geq m \circ m_2 \).

Proof.

(a) We first show that if \( m_1 \) and \( m_2 \) are compatible, then \( m_1 \) and \( m_2 \) have an upper bound. Since \( m_1 \) and \( m_2 \) are compatible, we have, by definition, \( m_1 \circ m_2 = m_2 \circ m_1 \).

We know from Proposition 3.10(d) that \( m_1 \circ m_2 \geq m_1 \). Similarly, \( m_1 \circ m_2 = m_2 \circ m_1 \geq m_2 \). Thus, \( m_1 \circ m_2 \) is an upper bound of \( m_1 \) and \( m_2 \).

Since \( M \) is a PDBA, it is consistently complete (see Proposition 2.4). Hence, if \( m_1 \) and \( m_2 \) have an upper bound, then they must have a least upper bound.

We now show that if \( m_1 \) and \( m_2 \) have a least upper bound, they commute. Let \( m_1 \) be \( \rho((x_1, y_1)) \) and let \( m_2 \) be \( \rho((x_2, y_2)) \). Since \( \rho \) is a poset isomorphism between \( M \) and \( S \), \( m_1 \) and \( m_2 \) have a least upper bound in \( M \) iff \( (x_1, y_1) \) and \( (x_2, y_2) \) have a least upper bound in \( S \). But \( (x_1, y_1) \) and \( (x_2, y_2) \) have a least upper bound in \( S \) iff \( x_1 \sqcap x_2 \sqcup y_1 \sqcup y_2 \). Thus, if \( m_1 \sqcap m_2 \) exists then \( x_1 \sqcap x_2 \sqcup y_1 \sqcup y_2 \). Hence, \( m_1 \circ m_2 = <x_1 \sqcap (x_2 \sqcup y_1), y_1 \sqcup (y_2 \sqcup x_1)> = <x_1 \sqcap x_2, y_1 \sqcup y_2> = m_1 \sqcap m_2 \). Similarly, \( m_2 \circ m_1 = <x_1 \sqcap y_2, y_1 \sqcup x_1> = m_2 \circ m_1 \). Therefore, \( m_1 \) and \( m_2 \) commute. The result follows.
(b) If \( m_1 \) and \( m_2 \) commute, then they have an upper bound, which is also an upper bound for \( m_3 \) and \( m_4 \). Hence, \( m_3 \) and \( m_4 \) commute.

(c) Assume that \( m_1 \geq m_2 \). Then, \((m \circ m_2) \circ (m \circ m_1) = m \circ m_2 \circ m_1 = m \circ m_1\). Hence, \( m \circ m_1 \geq m \circ m_2 \). □

**Definition 3.14.** For any \( a \) and \( b \) in \( \mathbb{P} \), let \( M_{a,b} \) denote the set of modifications that map \( b \) to \( a \). Thus,

\[
M_{a,b} \triangleq \{ m \in M \mid m(b) = a \}.
\]

**Proposition 3.15.** \( M_{a,b} \) has a least element with respect to \( \preceq \), namely \( < a \div b, a \div b > \), and a greatest element with respect to \( \preceq \), namely \( < a, a > \).

**Proof.** \( < x, y > \in M_{a,b} \iff (b \sqcap y) \sqcap x = a \) (and \( (b \sqcap x) \sqcap y = a \))

\[
\iff b \sqcap x \sqsubseteq a \text{ and } b \sqcap y \sqsubseteq a \text{ and } x \sqsubseteq a \text{ and } y \sqsubseteq a
\]

\[
\iff < x, y > \geq < a \div b, a \div b > \text{ and } < x, y > \leq < a, a >
\]

Hence, \( < a \div b, a \div b > \) is the least element of \( M_{a,b} \), and \( < a, a > \) is the greatest element of \( M_{a,b} \). □

**Definition 3.16.** Let \( \Delta(a,b) \) denote the least element (with respect to \( \preceq \)) of \( M_{a,b} \).

\[
\Delta(a,b) \triangleq \min_{\preceq}(M_{a,b}) = < a \div b, a \div b >.
\]

The above definition chooses \( \Delta(a,b) \) to be the least modification that changes \( b \) to \( a \).

**Proposition 3.17.**

(a) \( \Delta(a,b)(b) = a \). \hspace{1cm} (b) \( \Delta(m(a),a) \leq m \).

(c) \( \Delta(a,a) = 1 \). \hspace{1cm} (d) \( \Delta(a,b) = < a \div b, a \div b > = < a, a > \div < b, b > \).

(e) \( \Delta(m(b),b) = m \div < b,b > \).

**Proof.** (a) and (b) follow directly from the definition of \( \Delta \). (c) is a special case of (b). (d) follows from Proposition 3.15. As for (e), let \( a \) denote \( m(b) \). Then, from Proposition 3.15, \( \Delta(a,b) \leq m \leq < a,a > \). Hence, \( \Delta(a,b) \div < b,b > \leq m \div < b,b > \leq < a,a > \div < b,b > \). That is, \( \Delta(a,b) \leq m \div < b,b > \leq \Delta(a,b) \). The result follows. □

Now we define a binary operation \( \sqcup_c \) such that \( m_1 \sqcup_c m_2 \) may be interpreted as that part of modification \( m_1 \) that is compatible with modification \( m_2 \).

**Definition 3.18.** \( m_1 \sqcup_c m_2 \triangleq \min_{\preceq}(m_1 \circ m_2) \sqcap m_1 \).

In terms of Brouwerian algebraic operators, the above definition yields

\[
<x_1, y_1 > \sqcup_c < x_2, y_2 > = < x_1 \sqcap y_2, y_1 \sqcup x_2 >.
\]

**Proposition 3.19.**

(a) \( m_1 \sqcup_c m_2 = \max \{ m \leq m_1 \mid m \text{ is compatible with } m_2 \} \).

(b) \( m_1 \sqcup_c m_2 \) and \( m_2 \sqcup_c m_1 \) are compatible with each other.

**Proof.**

(a) Let \( A \) denote the set \( \{ m \leq m_1 \mid m \text{ commutes with } m_2 \} \). We first show that \( m_1 \sqcup_c m_2 \in A \). \( m_1 \sqcup_c m_2 \leq m_2 \circ m_1 \), from the definition of \( \sqcup_c \). \( m_2 \leq m_2 \circ m_1 \), from Proposition 3.10(c). Since \( m_1 \sqcup_c m_2 \) and \( m_2 \) have an upper bound (namely, \( m_2 \circ m_1 \)) it follows from Proposition 3.13(a) that \( m_1 \sqcup_c m_2 \) is compatible with \( m_2 \). \( m_1 \sqcup_c m_2 \leq m_1 \) from the definition of \( \sqcup_c \). Hence, \( m_1 \sqcup_c m_2 \in A \).

Now consider any \( m \in A \). Since \( m \leq m_1 \), we get \( m_2 \circ m \leq m_2 \circ m_1 \), from Proposition 3.13(c). Hence, \( (m_2 \circ m) \sqcup m \leq (m_2 \circ m_1) \sqcup m \leq (m_2 \circ m_1) \sqcup m_1 = m_1 \sqcup_c m_2 \). But
(m_2 \circ m) \sqcap m \text{ is } m, \text{ since } m_2 \circ m \sqsubseteq m \text{ whenever } m_2 \text{ and } m \text{ commute. Hence, } m \leq m_1 \sqcap m_2. \text{ The result follows.}

(b) \quad m_1 \mid_c m_2 \text{ commutes with } m_2, \text{ from (a). } m_2 \mid_c m_1 \leq m_2, \text{ again using (a). Hence, } m_1 \mid_c m_2 \text{ commutes with } m_2 \mid_c m_1, \text{ from Proposition 3.13(b).} \quad \Box

**Proposition 3.20.**

(a) \quad m_1 \circ m_2 = (m_1 \dashv m_2) \circ m_2.

(b) \quad m_1 \circ m_2 = (m_2 \mid m_1) \circ m_1 = (m_2 \mid m_1) \sqcap m_1.

(c) \quad \text{If } m_1 \circ m = m_2, \text{ then } m \geq m_2 \dashv m_1.

**Proof.**

(a) \quad \text{Since } M \text{ is a PDBA, } m_1 = (m_1 \dashv m_2) \sqcap (m_1 \sqcap m_2) = (m_1 \dashv m_2) \circ (m_1 \sqcap m_2). \text{ Therefore, } m_1 \circ m_2 = (m_1 \dashv m_2) \circ (m_1 \sqcap m_2) = (m_1 \dashv m_2) \circ m_2, \text{ since } m_1 \sqcap m_2 \leq m_2.

(b) \quad \text{It can be shown that } (m_1 \circ m_2) \dashv m_2 \leq m_1. \text{ Consequently, } m_1 \circ m_2 = (m_1 \circ m_2) \sqcap m_1 = ((m_1 \circ m_2) \sqcap m_2) \sqcap ((m_1 \circ m_2) \dashv m_2) \sqcap m_1 = (m_2 \mid m_1) \sqcap m_1.

(c) \quad \text{From (b) } m_1 \circ m = (m_1 \mid m_1) \sqcap m_1. \text{ It follows immediately that if } (m_1 \mid m_1) \sqcap m_1 = m_2, \text{ then } m \geq m_1 \mid m_1 \geq m_2 \dashv m_1. \quad \Box

Now we consider an operation + that “combines” modifications.

**Definition 3.21.** \(<x_1,y_1> + <x_2,y_2> =_{\text{def}} <x_1 \sqcup x_2, (y_1 \sqcup y_2) \sqcap x_1 \sqcap x_2> .

**Proposition 3.22.** \((M, +, I)\) is a join semi-lattice with \(I\) as the least element. Thus,

(a) \quad m + m = m

(b) \quad (m_1 + m_2) + m_3 = m_1 + (m_2 + m_3)

(c) \quad m_1 + m_2 = m_2 + m_1

(d) \quad m + I = m = I + m

**Proof.** \text{These follow directly from the definition of } +. \quad \Box

However, note that \((M, +, I)\) is not a join semi-lattice with respect to the partial order \(\leq\) we have used so far. The relationship between + and \(\leq\) will be clearer soon.

**Proposition 3.23.**

(a) \quad \text{If } m_1 \text{ and } m_2 \text{ commute then } m_1 + m_2 = m_1 \circ m_2 = m_1 \sqcap m_2.

(b) \quad (m_1 \mid_c m_2) \sqcap (m_2 \mid_c m_1) \leq m_1 + m_2.

**Proof.**

(a) \quad \text{Let } m_1 = <x_1, y_1> \text{ and } m_2 = <x_2, y_2>. \text{ If } m_1 \text{ and } m_2 \text{ are compatible then they have an upper bound, and hence } x_1 \sqcup x_2 \sqsubseteq y_1 \sqcup y_2. \text{ Hence, } m_1 + m_2 = <x_1 \sqcup x_2, y_1 \sqcup y_2> = m_1 \circ m_2 = m_1 \sqcap m_2.

(b) \quad \text{Let } m_1 = <x_1, y_1> \text{ and } m_2 = <x_2, y_2>. \text{ Note that } m_1 \mid_c m_2 = <x_1 \sqcup y_2, y_1 \sqcup x_2>, \text{ and } (m_1 \mid_c m_2) \sqcap (m_2 \mid_c m_1) = <x_1 \sqcup x_2) \sqcap (y_1 \sqcup y_2), (y_1 \sqcup y_2) \sqcap (x_1 \sqcup x_2)> . \text{ Hence, the result follows.} \quad \Box

As may be observed from the previous proposition, whenever \(m_1 \sqcap m_2 \) exists, \(m_1 + m_2 \) is the same as \(m_1 \sqcap m_2 \) and \(m_1 \circ m_2 \). Thus, + may be thought of as an appropriate totalization of the partial operator \(\sqcap\). If \(m_1 \) and \(m_2 \) conflict, we may think of \(m_1 + m_2 \) as being obtained by resolving the conflict in favour of the more important modification. Now we define a binary operator \(\mid_o\) so that \(m_1 \mid_o m_2 \) represents that part of modification \(m_1 \) that is not “overruled” by modification \(m_2 \), and an associated operator \(\dashv_o\).

**Definition 3.24.** Define the two binary operators \(|_o\) and \(\dashv_o\) as follows:

\[
m_1 \mid_o m_2 =_{\text{def}} (m_1 + m_2) \sqcap m_1
\]

\[
m_1 \dashv_o m_2 =_{\text{def}} (m_1 \mid_o m_2) \dashv m_2
\]
It can be verified that the definition of $|_o$ simplifies to $\langle x_1, y_1 \rangle |_o \langle x_2, y_2 \rangle = \langle x_1, y_1 \sqcap x_2 \rangle$.

**Proposition 3.25.**

(a) $m_1 |_o m_2 \leq m_1 |_o m_2 \leq m_1$

(b) $m_1 |_o m_2$ and $m_2 |_o m_1$ commute.

(c) $m_1 + m_2 = (m_1 |_o m_2) \sqcap (m_2 |_o m_1)$

(d) $m_1 |_o m_2 = (m_1 \sqcup o m_2) \sqcap (m_1 \sqcap m_2)$

(e) $m_1 + m_2 = (m_1 \sqcup o m_2) \sqcap (m_2 \sqcap o m_1)$

(f) $m_1 + m_2 = (m_1 |_o m_2) \sqcap m_2 = (m_2 |_o m_1) \sqcap m_1$

(g) $m_1 + m_2 = (m_1 \sqcup o m_2) \sqcap m_2 = (m_2 \sqcap o m_1) \sqcap m_1$

**Proof.**

(a) $m_1 |_o m_2 \leq m_1$ from the definition of $|_o$. It follows from Proposition 3.23(b) that $m_1 |_o m_2 \leq m_1 + m_2$. Since $m_1 |_o m_2 \leq m_1$ also, it follows that $m_1 |_o m_2 \leq (m_1 + m_2) \sqcap m_1 = m_1 |_o m_2$.

(b) From the definition of $|_o$, it follows that $m_1 + m_2$ is an upper bound for both $m_1 |_o m_2$ and $m_2 |_o m_1$. Hence, it follows from Proposition 3.13(a) that $m_1 |_o m_2$ and $m_2 |_o m_1$ commute.

(c) It can be verified that $m_1 + m_2 = ((m_1 + m_2) \sqcup m_1) \sqcap ((m_1 + m_2) \sqcap m_1)$.

(d) This follows from the Brouwerian identity $x = (x \sqcap y) \sqcup (x \sqcup y)$. Note that $(m_1 |_o m_2) \sqcap m_2 = m_1 \sqcap m_2$.

(e) This follows from (c) and (d) above.

(f) Let $m_1$ be $\langle x_1, y_1 \rangle$ and let $m_2$ be $\langle x_2, y_2 \rangle$. Then, $(m_1 |_o m_2) \circ m_2 = \langle x_1, y_1 \sqcap x_2 \rangle \circ \langle x_2, y_2 \rangle = \langle x_1 \sqcup x_2, (y_1 \sqcap x_2) \sqcup (y_2 \sqcup x_1) \rangle = m_1 + m_2$ (since $x_1 \sqsubseteq y_1$ and $x_2 \sqsubseteq y_2$).

(g) This follows from (f) and Proposition 3.20(a). $\square$

**Definition 3.26.** Modification $m_1$ is said to **overrule part of** modification $m_2$ if $m_2 + m_1 \nless m_2$.

The above definition, in Brouwerian algebraic terms, says that $\langle x_1, y_1 \rangle$ does not overrule part of $\langle x_2, y_2 \rangle$ iff $x_1 \sqsupseteq y_2$.

**Proposition 3.27.**

(a) $m_1$ overrules part of $m_2$ iff $m_2 |_o m_1 \neq m_2$.

(b) $m_1 |_o m_2 |_o m_2 = m_1 |_o m_2$

(c) $m_1 |_o m_2$ and $m_1 \sqcup o m_2$ are monotonic in $m_1$ and anti-monotonic in $m_2$.

(d) $m_1 |_o m_2 = \max \{ m \leq m_1 | m_2 \text{ does not overrule part of } m \}$.

(e) If $m_1 \leq m_2$ and $m$ overrules part of $m_1$ then $m$ overrules part of $m_2$.

**Proof.**

(a) It follows directly that $m_1 + m_2 \geq m_2$ iff $(m_1 + m_2) \sqcup m_2 = m_2$.

(b) This follows easily since $\langle x_1, y_1 \rangle |_o \langle x_2, y_2 \rangle = \langle x_1, y_1 \sqcap x_2 \rangle$.

(c) This too follows easily since $\langle x_1, y_1 \rangle |_o \langle x_2, y_2 \rangle = \langle x_1, y_1 \sqcup x_2 \rangle$.

(d) Let $A$ denote the set $\{ m \leq m_1 | m_2 \text{ does not overrule part of } m \}$. From (a) and (b) it follows that $m_1 |_o m_2 \in A$. Consider any $m \in A$. Then, $m \leq m_1$. Hence, from (c), $m |_o m_2 \leq m_1 |_o m_2$. But $m = m |_o m_2$ since $m \in A$ (from a). Hence, $m \leq m_1 |_o m_2$. The result follows.
(e) Assume that \( <x_1,y_1> \leq <x_2,y_2> \). If \( <x,y> \) does not overrule part of \( <x_2,y_2> \), then \( x \sqsubseteq y_2 \). Hence, \( x \sqsubseteq y_1 \). Consequently, \( <x,y> \) does not overrule part of \( <x_1,y_1> \). \( \square \)

4. Applications

We look at some applications of the various properties derived above. Let \( \text{interp} \) denote the integration operator of the \( fm \)-algebra \( (P,M,\Delta,+) \) defined in the previous section. The following properties make it clearer as to what exactly program-integration does. The properties will also be of use in subsequent applications.

**Proposition 4.1.**

(a) \( c[[a]]b = (\Delta(c,a) \circ \Delta(b,a)) \sqcup (\Delta(b,a) \circ \Delta(c,a)) \) (a).

(b) \( c[[a]]b = (\Delta(c,a) \circ \Delta(b,a)) \sqcup (\Delta(c,a) \circ \Delta(b,a)) \sqcup (\Delta(b,a) \circ \Delta(c,a)) \) (b).

(c) \( c[[a]]b = (\Delta(c,a) \circ \Delta(b,a) \circ \Delta(c,a)) = (\Delta(c,a) \circ \Delta(b,a)) \circ \Delta(c,a) \)

(d) If \( \Delta(c,a) \) and \( \Delta(b,a) \) commute then \( c[[a]]b = \Delta(c,a)(b) = \Delta(b,a)(c) \).

(e) \( \Delta(c[[a]]b,a) \leq \Delta(c,a) + \Delta(b,a) \).

(f) If \( \Delta(c,a) \) and \( \Delta(b,a) \) commute then \( \Delta(c[[a]]b,a) = \Delta(c,a) + \Delta(b,a) \).

(g) \( \Delta(c[[a]]b,a) \leq \Delta(b,a) \circ \Delta(c,a) \).

(h) \( \Delta(c[[a]]b,a) = (\Delta(c,a) \circ \Delta(b,a)) \circ \Delta(c,a) \).

**Proof.**

(a)-(d) These follow immediately by rewriting \( \Delta(c,a) + \Delta(b,a) \) in the definition of \( \text{interp} \) using Propositions 3.25 and 3.23.

(e) This follows from the definition of \( c[[a]]b \) and Proposition 3.17(b).

(f) It follows from 3.17(e) that \( \Delta(c[[a]]b,a) = (\Delta(c,a) + \Delta(b,a)) \circ \Delta(c,a) \), which simplifies to \( \Delta(c,a) + \Delta(b,a) \) when \( \Delta(c,a) \) and \( \Delta(b,a) \) are compatible.

(g) This follows from (d) and Proposition 3.17(b).

(h) This follows from (c) and Proposition 3.17(e). \( \square \)

**Proposition 4.2.** Let \( m_1 \) and \( m_2 \) be compatible modifications. Then, \( (m_1 \sqcup m_2)(a) = m_1(a) \circ m_2(a) \).

**Proof.** Let \( c = m_1(a) \) and \( b = m_2(a) \). Now, \( \Delta(c,a) \leq m_1 \) and \( \Delta(b,a) \leq m_2 \). Thus, \( \Delta(c,a) \) and \( \Delta(b,a) \) are themselves compatible with each other, and with \( m_1 \) and \( m_2 \) too. Then, \( (m_1 \sqcup m_2)(a) = (m_1 \circ m_2)(a) = m_1(m_2(a)) = m_1(b) = m_1(\Delta(b,a)(a)) = \Delta(b,a)(m_1(a)) = \Delta(b,a)(c) = c[[a]]b \). The last step follows from Proposition 4.1(d). \( \square \)

The problem of separating consecutive edits on some program into individual edits on the same program is as follows: Given programs \( a, d, \) and \( c \), where \( d \) was obtained by modifying \( a \) and \( c \) was obtained by modifying \( d \), we seek a program \( b \) that includes the second modification but not the first. A solution previously proposed [6] was that the program we seek is a solution \( x \) to the equation \( d[[a]]x = c \). Another solution, suggested by our interpretation of \( \Delta \) as capturing “program modifications”, is that \( \Delta(c,d)(a) \) is the program we seek. We consider the relationship between these solutions below.

**Theorem 4.3.** If \( \Delta(c,d) \) and \( \Delta(d,a) \) do not conflict then the equation \( d[[a]]x = c \) has a solution for \( x \).
**Proof.** We show that \( b = \Delta(c,d)(a) \) satisfies the given equation. Now, \( \Delta(b,a) \leq \Delta(c,d) \) (Proposition 3.17(b)). Since \( \Delta(c,d) \) and \( \Delta(d,a) \) are compatible, it follows that \( \Delta(b,a) \) and \( \Delta(d,a) \) are also compatible (Proposition 3.13(b)). Hence, from Proposition 4.1(d), we get \( d[[a]]b = \Delta(d,a)(b) = \Delta(d,a)(\Delta(c,d)(a)) = \Delta(c,d)(\Delta(d,a)(a)) = c. \square \)

**Theorem 4.4.** If the equation \( d[[a]]x = c \) has a solution for \( x \) then \( b = \Delta(c,d)(a) \) is a solution of that equation. Further, \( \Delta(c,d) \) is the least modification of \( a \) necessary to produce a solution of that equation (and thus \( b \) is the “least-modified” solution of that equation) in the following sense: if \( m(a) \) is any solution to the equation, then \( m \geq \Delta(c,d) = \Delta(b,a) \).

**Proof.** Let \( x \) be such that \( d[[a]]x = c \). Let \( b \) denote \( \Delta(c,d)(a) \). We have \( \Delta(b,a) = \Delta(c,d) \cdot <a,a>, \) from Proposition 3.17(e), and \( \Delta(c,d) = (\Delta(x,a)|_o \Delta(d,a)) \cdot <d,d>, \) from Proposition 4.1(h). Using the properties of Brouwerian algebra we can show that \( \Delta(c,d) \cdot <a,a> = \Delta(c,d) \), and hence that \( \Delta(b,a) = \Delta(c,d). \)

Now, \( \Delta(c,d) \leq \Delta(x,a)|_o \Delta(d,a) \) (Proposition 4.1(h)). Further, \( \Delta(x,a)|_o \Delta(d,a) \) is not overruled by \( \Delta(d,a) \). Hence, \( \Delta(c,d) \) is not overruled by \( \Delta(d,a) \). Thus, \( \Delta(c,d)|_o \Delta(d,a) = \Delta(c,d) \). Then, from Proposition 4.1(c), we get \( d[[a]]b = (\Delta(b,a)|_o \Delta(d,a))(d) = (\Delta(c,d)|_o \Delta(d,a))(d) = \Delta(c,d)(d) = c. \) Hence, \( b \) also satisfies the equation \( d[[a]]x = c \). It also follows from above that \( \Delta(c,d) \leq \Delta(x,a) \) for any \( x \) that satisfies \( d[[a]]x = c. \) \square

An apparent similarity between the concept of program integration and that of pushout in category theory gives rise to the following question: is program integration simply a pushout in an appropriately constructed category? We define below a particular category, one in which the objects denote programs, and arrows denote program-modifications.

**Definition 4.5.** Define a category \( C \) as follows: The set of objects is \( P \). For every \( a \in P \), and \( m \in M \), there is an arrow \([a,m,m(a)]\) with domain \( a \) and codomain \( m(a) \). Thus, every arrow \([a,m,m(a)]\) is associated with a program modification \( m \). And, if no confusion is likely, the arrow will be denoted just \( m \). The operation \( \circ \) is defined to be function composition. Thus, \([b,m_2,c] \circ [a,m_1,b] = [a,m_2 \circ m_1,c]\).

**Proposition 4.6.** The category \( C \) is skeletal: no two distinct objects are isomorphic.

**Proof.** Let \( a \) and \( b \) be isomorphic objects. Thus, there exist arrows \([a,f,b]\) and \([b,g,a]\) such that \( f \circ g = I \) and \( g \circ f = I \). But, \( I = f \circ g \) implies that \( I \geq f \). Hence, \( I = f \), since \( I \) is the least element of \( M \). Hence, \( b = f(a) = a. \) \square

**Theorem 4.7.** The pushout of diagram \( 1(a) \) exists iff \( m_1 \) and \( m_2 \) commute and \( m_1 \sqcap m_2 = I \), in which case the pushout is the diagram \( 1(b) \). In particular, the pushout, when it exists, does yield the integrated program.

**Proof.** In this proof, we will show only the following: if the pushout of diagram \( 1(a) \) exists then \( m_1 \) and \( m_2 \) must commute and the pushout must be the diagram \( 1(c) \). See [3] for the complete proof. Note that if \( m_1 \sqcap m_2 = I \) then \( m_1 \sqcup m_2 = (m_1 \sqcup m_2) \sqcap (m_1 \sqcap m_2) = m_1 \), and similarly, \( m_2 \sqcup m_1 = m_2. \) Under these conditions, diagram \( 1(c) \) reduces to diagram \( 1(b) \).
Assume that diagram 1(d) is the pushout of diagram 1(a). Then, for any \( m_5 \) and \( m_6 \) that make diagram 1(e) commute, there exists a unique \( m \) that makes diagram 1(f) commute. Thus, \( m \circ m_3 = m_5 \). Hence, \( m \leq m_5 \) from Proposition 3.10(c). Similarly, \( m_5 \leq m_6 \).

Further, since \( m \circ m_3 = m_5 \), Proposition 3.20(c) yields \( m_3 \geq m_5 \). \( \vdash m \geq m_5 \vdash m_6 \). Now choose \( m_5 \) to be \( m_1 \) and \( m_6 \) to be \( m_2 \vdash m_1 \) — this choice makes diagram 1(e) commute (see Proposition 3.20(b)). Hence, from the previous inequality, we derive that \( m_3 \geq m_5 \geq m_6 \geq m_1 \vdash (m_2 \vdash m_1) \geq m_1 \vdash m_2 \). Similarly, we derive that \( m_4 \geq m_2 \vdash m_1 \).

Now, \( m_3 \) and \( m_4 \) commute, since \( m_4 \circ m_1 = m_3 \circ m_2 \) is an upper bound for \( m_3 \) and \( m_4 \). Hence, \( m_1 \vdash m_2 \) and \( m_2 \vdash m_1 \) commute (Proposition 3.13(b)). Hence, \( m_1 \) and \( m_2 \) commute. (Note that \( m_1 \vdash m_2 \)) commutes with both \( m_2 \vdash m_1 \) and \( m_2 \vdash m_1 \), and hence with \( (m_2 \vdash m_1) \circ (m_2 \vdash m_1) \), which is \( m_2 \). Similarly, since \( m_2 \) commutes with both \( m_1 \vdash m_2 \) and \( m_1 \vdash m_2 \), it commutes with \( (m_1 \vdash m_2) \circ (m_1 \vdash m_2) \), which is \( m_1 \).

Now, choose \( m_5 \) to be \( m_1 \vdash m_2 \) and \( m_6 \) to be \( m_2 \vdash m_1 \). Since \( m_1 \) and \( m_2 \) commute, this choice makes diagram 1(e) commute (see Proposition 3.20(a)). As observed earlier, \( m \) must be such that \( m \leq m_5 = m_1 \vdash m_2 \leq m_3 \). Thus, \( m_3 = m \circ m_3 = m_5 = m_1 \vdash m_2 \). Similarly, \( m_4 = m_2 \vdash m_1 \).

It follows that \( d = (m_3 \circ m_2)(a) = ((m_1 \vdash m_2) \circ m_2)(a) = (m_1 \circ m_2)(a) = (m_1 \vdash m_2)(a) = m_1(a) \vdash m_2(a) = c \vdash [a] b \). \( \square \)

The above theorem establishes certain similarities between the concepts of program-integration and pushout. However, program integration is a “weaker” concept than pushout in that it is always defined (i.e., for any three programs) unlike the pushout. For example, if \( m_1 \) and \( m_2 \) do not commute, then the pushout of diagram 1(a)
does not exist, although $c[[a]]b$ does exist.

5. Conclusion

We have studied the algebraic structure of (a particular model of) the domain of program modifications. The central concept developed is that of a partial order among modifications, defined in terms of function composition. The set of modifications forms a partial double Brouwerian algebra with respect to this partial ordering. The concept of compatibility among modifications was defined (in terms of commutativity with respect to function composition), and was shown to be related to the existence of the least upper bound of the given modifications.

The $fm$-algebraic operators $\Delta$ and $+$ were then considered, and various of their properties, in terms of the above concepts, were established. Various other related operators were defined, and their properties were studied.

Finally, these properties were utilized to establish various results concerning program-integration, separating consecutive edits, and the relationship between the concepts of program-integration and pushout.

REFERENCES
