ON THE ALGEBRAIC PROPERTIES OF PROGRAM INTEGRATION

by

Thomas Reps

Computer Sciences Technical Report #856

June 1989

ON THE ALGEBRAIC PROPERTIES OF PROGRAM INTEGRATION

THOMAS REPS
University of Wisconsin

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 recently proposed by Horwitz, Prins, and Reps 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. To determine this information, the algorithm employs a program representation that is similar, but not identical, to the *program dependence graphs* that have been used previously in vectorizing and parallelizing compilers.

This paper studies the algebraic properties of the program-integration operation. To do so, we first modify the integration algorithm by recasting it as an operation on a *Brouwerian algebra* constructed from sets of dependence graphs. (A Brouwerian algebra is a distributive lattice with an operation $a \div b$ characterized by $a \div b \subseteq c$ iff $a \subseteq b \cup c$.) In this algebra, the program-integration operation can be defined solely in terms of $\cup$, $\cap$, and $\div$. The bulk of the paper investigates this operation's algebraic properties; this investigation makes use of the rich set of algebraic laws that hold in Brouwerian algebras.

CR Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques – *programmer workbench*; D.2.6 [**Software Engineering**]: Programming Environments; D.2.7 [**Software Engineering**]: Distribution and Maintenance – *enhancement, restructuring, version control*; D.2.9 [**Software Engineering**]: Management – *programming teams, software configuration management*; F.4.m [**Mathematical Logic and Formal Languages**]: Miscellaneous; G.2.m [**Discrete Mathematics**]: Miscellaneous

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Brouwerian algebra, dependence graph, program slice, program integration

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. Given a program P and a set of variants of P—created, say, by modifying separate copies of P—the goal is to determine whether the modifications inter-

This work was supported in part by a David and Lucile Packard Fellowship for Science and Engineering, by the National Science Foundation under grant DCR-8552602, by the Defense Advanced Research Projects Agency, monitored by the Office of Naval Research under contract N00014-88-K-0590, as well as by grants from IBM, DEC, and Xerox.

Author's address: Computer Sciences Department, University of Wisconsin–Madison, 1210 W. Dayton St., Madison, WI 53706.

fere, and, if they do not, to create an *integrated* program that includes all changes as well as all features of P that are preserved in all variants [6, 7]. Opportunities for program integration arise in many situations:

(1) A system may be "customized" by a user while simultaneously being upgraded by a maintainer. When the next release of the system is sent to the user, he must integrate his customized version and the newly released version with respect to the original version of the system so as to incorporate both his customizations and the upgrades.

(2) Program integration also arises as systems are being created. Program development is usually a cooperative activity that involves multiple programmers. If a task can be decomposed into independent pieces, the different aspects of the task can be developed and tested independently by different programmers. However, if such a decomposition is not possible, the members of the programming team must work with multiple, separate copies of the source files, and the different versions of the files must ultimately be integrated to produce a common version.

(3) The program-integration problem also arises in a slightly different guise when a tree or dag of related versions of a program has been created (to support different machines or different operating systems, for instance), and the goal is to make the same enhancement or bug-fix to all of them. For example, if the change is made to the root version—by modifying the root manually—the process of installing the change in all other versions requires a succession of program integrations.

Anyone who has had to reconcile divergent lines of development will recognize these situations and appreciate the need for automatic assistance.

At present, the only available tools for integration implement an operation for merging files as strings of text. This approach has the advantage that the current tools are as applicable to merging documents, data files, and other text objects as they are to merging programs. However, these tools are necessarily of limited utility for integrating programs because the manner in which two programs are merged is not *safe*—one has no guarantees about the way the program that results from a purely *textual* merge behaves in relation to the behavior of the programs that are the arguments to the merge. For example, if one variant contains changes only on lines 5–10, while the other variant contains changes only on lines 15–20, a text-based program-integration algorithm would deem these changes to be interference-free; however, just because changes are made at different places in a program is no reason to believe that the changes are free of undesirable interactions. The merged program produced by such a tool must, therefore, be checked carefully for conflicts that might have been introduced by the merge.

The program-integration algorithm recently proposed by Horwitz, Prins, and Reps provides a way to create a *semantics-based* (*i.e.*, language-based) tool for automatic program integration [6, 7]. Changes in *behavior* rather than changes in text are detected, and are preserved in the integrated program. Although it is undecidable to determine whether a program modification actually leads to a change in program behavior, it is possible to determine a safe approximation by comparing each of the variants with the original program P. To determine this information, the algorithm employs a program representation that is similar, but not identical, to the *program dependence graphs* that have been used previously in vectorizing and parallelizing compilers. It makes use of an operation on these graphs called *program slicing* [11] (originally defined to aid in debugging [17]) to find potentially changed computations.

This paper concerns the algebraic properties of the program-integration operation. We actually study a version of the program-integration algorithm that is slightly different from that given in [6, 7]; in the version studied here, program integration is cast as an operation on a *Brouwerian algebra* constructed from sets of dependence graphs. A Brouwerian algebra [9] is a distributive lattice with an operation $a \div b$ characterized by $a \div b \subseteq c$ iff $a \subseteq b \cup c$ (see Section 3 and the Appendix). In this algebra, the program-

integration operation can be defined solely in terms of $\cup$, $\cap$, and $\dot{-}$. The bulk of the paper investigates this operation's algebraic properties; this investigation makes use of the rich set of algebraic laws that hold in Brouwerian algebras.

The program-integration problem studied in [6, 7] is a greatly simplified one, and the capabilities of the algorithm developed there are severely limited. In particular, the algorithm applies only to programs written in a simple language in which expressions contain scalar variables and constants, and the only statements are assignment statements, conditional statements, and while-loops. Current research focuses both on extending the set of language constructs to which the program-integration algorithm is applicable, as well as investigating some alternatives to using program dependence graphs and program slicing. However, our hope is that such extensions and modifications will share with the original integration algorithm some common set of algebraic properties.

We feel that Brouwerian algebras provide the right "abstract interface" for studying the common properties of different integration algorithms. The integration operation in a Brouwerian algebra is defined purely in terms of $\cup$, $\cap$, and $\dot{-}$, and thus has an analogue in all Brouwerian algebras. In Sections 5, 6, 7, and 8, the properties of this integration operation are studied using only algebraic identities and inequalities; consequently, the results obtained apply to all Brouwerian algebras. Thus, to show that a proposed program-integration algorithm shares these properties, one merely has to show (as we do in Section 4 for three variations on the program-integration algorithm from [6, 7]) that the algorithm can be cast as an integration operation in a Brouwerian algebra.

It is important to understand that, because the set of dependence graphs does *not* form a Brouwerian algebra, not all of the results established about integration in Brouwerian algebras carry over to the program-integration operation given in [6, 7]. However, as shown in Section 4, it is possible to generalize that algorithm slightly to construct a Brouwerian algebra from *sets* of dependence graphs. In this algebra (as well as all other Brouwerian algebras), the integration operation satisfies the properties demonstrated in the paper.

An additional contribution of this work is that it allows one of the restrictions that is part of the algorithm given in [6, 7] to be relaxed. We show how to eliminate the requirement that the editor used to create program variants from a given base program provide a tagging capability so that common components (*i.e.*, statements and predicates) can be identified in different versions. (The assumption is stated fully at the beginning of Section 2.2.) As discussed in Section 4, it is possible to construct dependence-graph algebras for which this restriction is no longer necessary; the elements of these algebras do not have tags on their components. Thus, in principle it is no longer necessary for program integration to be supported by a closed system; programs can be integrated, no matter what their origin.

The paper is organized into nine sections. Section 2 provides a brief overview of the program-integration algorithm that was described in [6, 7]. Section 2 also reviews the semantic properties of the program resulting from integration that were established in [15, 16]. Readers familiar with the algorithm from [6, 7] should skip directly to Section 3, which defines Brouwerian algebras and introduces some notation used throughout the rest of the paper. (To make the paper self-contained, the algebraic laws that hold for elements of Brouwerian algebras are discussed in an Appendix that appears at the end of the paper; thus, it may be helpful to read the Appendix in conjunction with Section 3. Throughout the body of the paper, when a law given in the Appendix is used to justify a step of a proof, it is referred to by the number given for the law in the Appendix.)

Section 4 defines a particular Brouwerian algebra constructed from sets of dependence graphs. Section 5 defines the operation to integrate elements of a Brouwerian algebra. For the dependence-graph algebra dis-

cussed in Section 4, this operation corresponds very closely to the operation for integrating programs by combining dependence graphs summarized in Section 2. Section 6 gives the proof of an associative law for the integration operation. It also defines a generalization of the integration operation to one that simultaneously integrates more than two variants with a given base element. Section 7 addresses the question of whether there is an integrand compatible with a given base b, integrand a, and result m. This problem is related to one of the applications of program integration, that of separating consecutive edits on some base program into individual edits on the base program. Section 8 concerns a similar question; it looks at the question of whether there is a base element that is compatible with given integrands a and b, and result m. Section 9 discusses the relationship of the work described in the paper to previous work.

2. OVERVIEW OF AN ALGORITHM FOR PROGRAM INTEGRATION

This section provides a brief overview of the algorithm that uses program dependence graphs to integrate programs. (Full details of the algorithm can be found in [6,7].) Given a program *Base* and two variants A and B, the program-integration algorithm determines whether the changes made to *Base* to produce A and B interfere; if there is no interference, the algorithm produces a merged program M that incorporates the changed behavior of A with respect to *Base*, the changed behavior of B with respect to *Base*, and the unchanged behavior common to *Base*, A, and B.

The algorithm applies to a simplified programming language with the following characteristics: expressions contain only scalar variables and constants; statements are either assignment statements, conditional statements, while loops, or a restricted kind of "output statement" called an *end statement*, which can only appear at the end of a program. An end statement names one or more of the variables used in the program. The variables named in the end statement are those whose final values are of interest to the programmer; when execution terminates, the final state is defined on only those variables in the end statement. Thus a program is of the form

program
  list-of-statements
end(id^*).

2.1. Program Dependence Graphs and Program Slicing

The program dependence graphs used by the program-integration algorithm are similar to those used previously for representing programs in vectorizing compilers [8,2]. Vertices represent the predicates and assignment statements of the program; in addition, there is a special vertex called the *entry vertex*, there are *initial-definition vertices* for each variable that may be used before being defined, and there are *final-use vertices* for each variable named in the **end** statement. There are two kinds of edges: *control dependence edges* and *data dependence edges*. The source of a control dependence edge is either the entry vertex or a predicate vertex and each edge is labeled either **true** or **false**. A control dependence edge $v \rightarrow_c w$ from vertex v to vertex w means (roughly) that during execution, whenever the predicate represented by v is evaluated and its value matches the label on the edge to w, then the program component represented by w will eventually be executed.

A data dependence edge from vertex v to vertex w means that the program's behavior might change if the relative order of the components represented by v and w were reversed. There are two kinds of data dependence edges, *flow dependence edges* and *def-order dependence edges*: A flow dependence edge $v \rightarrow_f w$ runs from a vertex v that represents an assignment to a variable x to a vertex w that represents a use of x reached by that assignment; a def-order edge $v \rightarrow_{do(u)} w$ runs between two vertices that represent assignments to x, both of which reach a common use u.

Example. Figure 1 shows an example program and its program dependence graph. The boldface arrows represent control dependence edges; dashed arrows represent def-order dependence edges; solid arrows represent loop-independent flow dependence edges; solid arrows with a hash mark represent loop-carried flow dependence edges.

For a vertex s of a program dependence graph G, the *slice* of G with respect to s, written as G/s, is a graph containing all vertices on which s has a transitive flow or control dependence (*i.e.*, all vertices that can reach s via flow or control edges): $V(G/s) = \{ w \in V(G) \mid w \longrightarrow^{*}_{c,f} s \}$. We extend the definition to a set of vertices $S = \bigcup_i s_i$ as follows: $V(G/S) = V(G/(\bigcup_i s_i)) = \bigcup_i V(G/s_i)$. It is useful to define $V(G/v) = \emptyset$ for any $v \notin V(G)$.

The edges in the graph G/S are essentially those in the subgraph of G induced by $V(G/S)$, with the exception that a def-order edge $v \longrightarrow_{do(u)} w$ is only included if, in addition to v and w, $V(G/S)$ also contains the vertex u that is directly flow dependent on the definitions at v and w. In terms of the three types of edges in a program dependence graph we have:

$$\begin{aligned} E(G/S) = \quad & \{ (v \longrightarrow_f w) \in E(G) \mid v, w \in V(G/S) \} \\ & \cup \{ (v \longrightarrow_c w) \in E(G) \mid v, w \in V(G/S) \} \\ & \cup \{ (v \longrightarrow_{do(u)} w) \in E(G) \mid u, v, w \in V(G/S) \} \end{aligned}$$

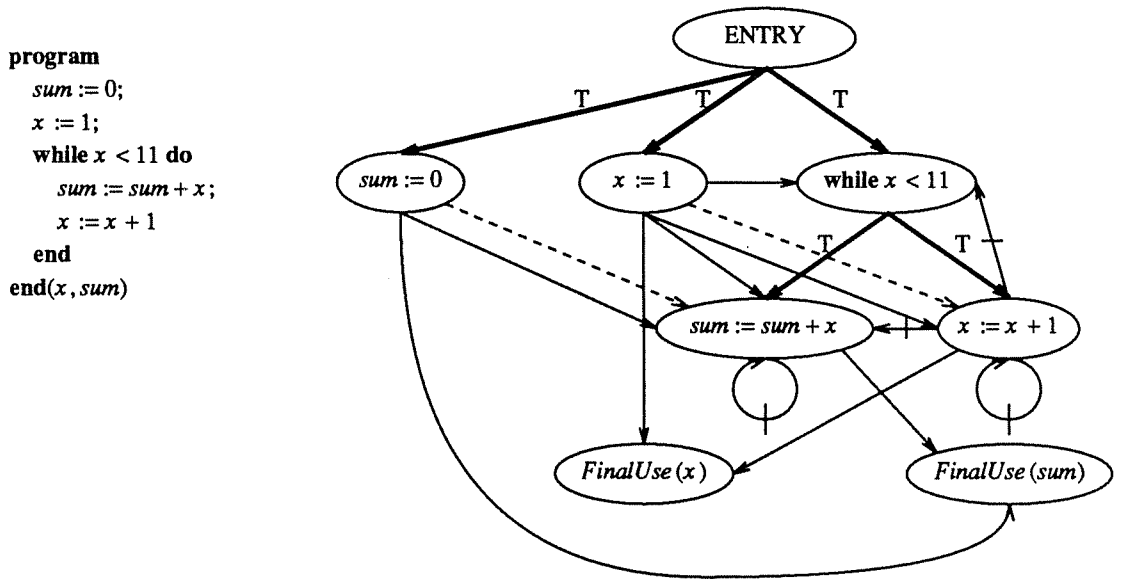

Figure 1. An example program, which sums the integers from 1 to 10 and leaves the result in the variable *sum*, and its program dependence graph. The boldface arrows represent control dependence edges, dashed arrows represent def-order dependence edges, solid arrows represent loop-independent flow dependence edges, and solid arrows with a hash mark represent loop-carried flow dependence edges.

Example. Figure 2 shows the graph that results from slicing the program dependence graph from Figure 1 with respect to the final-use vertex for x.

The significance of a slice is that it captures a portion of a program's behavior in the sense that, for any initial state on which the program halts, the program and the slice compute the same sequence of values for each element of the slice [15]. In our case a program point can be (1) an assignment statement, (2) a control predicate, or (3) a final use of a variable in an end statement. Because a statement or control predicate can be reached repeatedly in a program, by "computing the same sequence of values for each element of the slice" we mean: (1) for any assignment statement the same *sequence* of values are assigned to the target variable; (2) for a predicate the same *sequence* of boolean values are produced; and (3) for each final use the same value for the variable is produced.

THEOREM. (SLICING THEOREM [15]). *Let Q be a slice of program P with respect to a set of vertices. If σ is a state on which P halts, then for any state σ' that agrees with σ on all variables for which there are initial-definition vertices in G_Q: (1) Q halts on σ', (2) P and Q compute the same sequence of values at each program point of Q, and (3) the final states agree on all variables for which there are final-use vertices in G_Q.*

2.2. An Algorithm for Integrating Programs

The program-integration algorithm requires that a special program editor be used to create variants A and B from (copies of) *Base*. This editor is assumed to have the following properties:

```
program
  x := 1;
  while x < 11 do
    x := x + 1
  end
end(x)
```

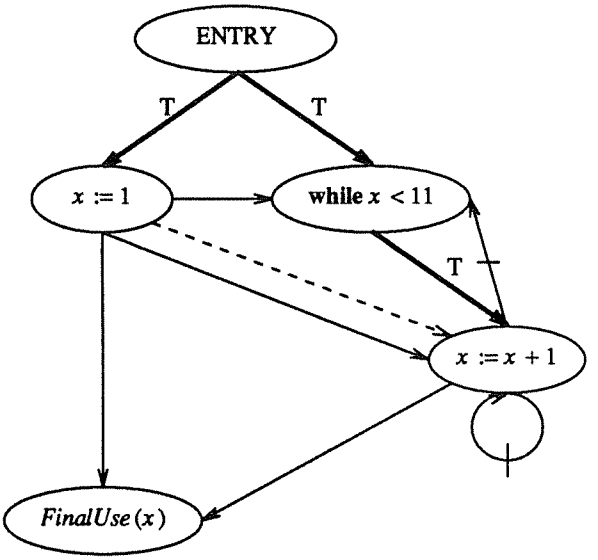

Figure 2. The graph that results from slicing the example from Figure 1 with respect to the final-use vertex for x, together with the one program to which it corresponds.

(1) The editor provides a tagging capability so that common components (*i.e.*, statements and predicates) can be identified in all three versions. Each component's tag is guaranteed to persist across different editing sessions and machines; tags are allocated by a single server, so that two different editors cannot allocate the same new tag.

(2) The operations on program components supported by the editor are insert, delete, and move. When editing a copy of *Base* to create a variant, a newly inserted component is given a previously unused tag; the tag of a component that is deleted is never re-used; a component that is moved from its original position in *Base* to a new position in the variant retains its tag from *Base*.

A tagging facility meeting these requirements can be supported by language-based editors, such as those that can be created by such systems as MENTOR [1], GANDALF [3, 10], and the Synthesizer Generator [13, 14].

These tags provide a means for identifying how the program-dependence-graph vertices in different versions correspond. It is these tags that are used to determine "identical" vertices when we perform operations using vertices from different program dependence graphs. For instance, when we speak below of "identical slices", where the slices are actually taken in different graphs (*e.g.*, $(G_{Base} / v) = (G_A / v)$), we mean that the slices are isomorphic under the mapping provided by the editor-supplied tags.

The first step of the program-integration algorithm determines slices that represent a safe approximation to the changed computation threads of A and B and the computation threads of *Base* preserved in both A and B; the second step combines these slices to form the merged graph G_M; the third step tests G_M for interference.

Step 1: Determining changed and preserved computation threads

If the slice of variant G_A at vertex v differs from the slice of G_{Base} at v, then G_A and G_{Base} can compute different values at v. In other words, vertex v is a site that potentially exhibits changed behavior in the two programs. Thus, we define the *affected points* of G_A with respect to G_{Base}, denoted by $AP_{A,Base}$, to be the subset of vertices of G_A whose slices in G_{Base} and G_A differ $AP_{A,Base} \triangleq \{ v \in V(G_A) \mid (G_{Base} / v) \neq (G_A / v) \}$. We define $AP_{B,Base}$ similarly. It follows that the slices $G_A / AP_{A,Base}$ and $G_B / AP_{B,Base}$ capture the respective computation threads of A and B that differ from *Base*.

We define the *preserved points* $PP_{Base,A,B}$ of G_{Base} as the subset of vertices of G_{Base} with identical slices in G_{Base}, G_A, and G_B:

$$PP_{Base,A,B} \triangleq \{ v \in V(G_{Base}) \mid (G_{Base} / v) = (G_A / v) = (G_B / v) \}.$$

Thus, the unchanged computation threads common to both A and B are captured by the slice $G_{Base} / PP_{Base,A,B}$.

Step 2: Forming the merged graph

The merged graph G_M is formed by taking the graph union of the slices that characterize the changed behavior of A, the changed behavior of B, and behavior of *Base* preserved in both A and B:

$$G_M \triangleq (G_A / AP_{A,Base}) \cup (G_B / AP_{B,Base}) \cup (G_{Base} / PP_{Base,A,B}).$$

Step 3: Testing for interference

There are two possible ways by which the graph G_M can fail to represent a satisfactory integrated program; we refer to them as "Type I interference" and "Type II interference." The criterion for Type I interference is based on a comparison of slices of G_A, G_B, and G_M. The slices $G_A / AP_{A,Base}$ and $G_B / AP_{B,Base}$ represent the changed computation threads of programs A and B with respect to *Base*. A and B interfere if G_M does not preserve these slices; that is, there is interference of this kind if either $(G_M / AP_{A,Base}) \neq (G_A / AP_{A,Base})$ or $(G_M / AP_{B,Base}) \neq (G_B / AP_{B,Base})$.

The final step of the integration method involves reconstituting a program from the merged program dependence graph. However, it is possible that there is no such program—the merged graph can be an infeasible program dependence graph; this is Type II interference. (The reader is referred to [7] for a discussion of reconstructing a program from the merged program dependence graph and the inherent difficulties of this problem.)

If neither kind of interference occurs, one of the programs that corresponds to the graph G_M is returned as the result of the integration operation.

The following theorem characterizes the execution behavior of the integrated program produced by the program-integration algorithm in terms of the behaviors of the base program and the two variants [15, 16].

THEOREM. (INTEGRATION THEOREM [15, 16]). *If A and B are two variants of Base for which integration succeeds (and produces program M), then for any initial state σ on which A, B, and Base all halt, (1) M halts on σ, (2) if x is a variable defined in the final state of A for which the final states of A and Base disagree, then the final state of M agrees with the final state of A on x, (3) if y is a variable defined in the final state of B for which the final states of B and Base disagree, then the final state of M agrees with the final state of B on y, and (4) if z is a variable on which the final states of A, B, and Base agree, then the final state of M agrees with the final state of Base on z.*

Restated less formally, M preserves the changed behaviors of both A and B (with respect to *Base*) as well as the unchanged behavior of all three.

3. TERMINOLOGY AND NOTATION

Definition. A *lattice* is an algebra $(L, \cup, \cap)$, where L is a set of elements that is closed under $\cup$ and $\cap$, and for all a, b, and c in L the following axioms are satisfied:

$$a \cup a = a \qquad a \cap a = a \qquad a \cup (a \cap b) = a$$
$$a \cup b = b \cup a \qquad a \cap b = b \cap a \qquad a \cap (a \cup b) = a$$
$$(a \cup b) \cup c = a \cup (b \cup c) \qquad (a \cap b) \cap c = a \cap (b \cap c)$$

The symbol $\subseteq$ is used to denote the partial order on the elements of L given by $a \subseteq b$ iff $a \cap b = a$ (or, equivalently, $a \subseteq b$ iff $a \cup b = b$). All lattices considered in this paper have a least element and a greatest element, which are denoted by $\varnothing$ and 1, respectively.

Definition. A *Brouwerian algebra* [9] is an algebra $(L, \cup, \cap, \div, 1)$ where

(i) $(L, \cup, \cap)$ is a lattice with greatest element 1.

(ii) L is closed under $\div$.

(iii) For all a, b, and c in L, $a \div b \subseteq c$ iff $a \subseteq b \cup c$.

It can be shown that L has a least element, given by $\varnothing = 1 \div 1$, and that L is a distributive lattice; that is, for all a, b, and c in L,

(iv) $a \cup (b \cap c) = (a \cup b) \cap (a \cup c)$.

(v) $a \cap (b \cup c) = (a \cap b) \cup (a \cap c)$.

Definition. A *double Brouwerian algebra* [9] is an algebra $(L, \cup, \cap, \dot{-}, \div, 1)$ where both $(L, \cup, \cap, \dot{-}, 1)$ and $(L, \cap, \cup, \div, 1 \dot{-} 1)$ are Brouwerian algebras. In particular,

(i) L is closed under $\div$.

(ii) For all a, b, and c in L, $a \div b \supseteq c$ iff $a \supseteq b \cap c$.

4. AN ALGEBRA OF DEPENDENCE GRAPHS

We now recast the integration algorithm as an operation on a Brouwerian algebra constructed from sets of dependence graphs.

Let G be the set of well-formed program dependence graphs, and let the relation symbol $\leq$ denote the relation "is-a-slice-of," (*i.e.*, $x \leq y$ means that graph x is a slice of graph y). Define set G_1, the set of all program dependence graphs that are single-point slices, to be

$G_1 \triangleq \{ g \in G \mid \exists x \in V(G)$ such that $(g / x) = g \}$.

Define set L, the set of all downwards-closed sets of single-point slices, to be

$L \triangleq \{ S \in P(G_1) \mid \forall s \in S, \forall x \in G_1$ if $x \leq s$ then $x \in S \}$,

where $P(G_1)$ denotes the power set of G_1. For all $x, y \in L$, define the operation $x \dot{-} y$ to be

$x \dot{-} y \triangleq \{ z \in G_1 \mid \exists p \in (x - y)$ such that $z \leq p \}$.

THEOREM 1. *$(L, \cup, \cap, \dot{-}, G_1)$ is a Brouwerian algebra.*

PROOF. Because the elements of L are downwards-closed sets of single-point slices, it is clear that G_1, which consists of all single-point slices, is a superset of any element of L. Suppose $s \in G_1$ and x is a single-point slice of s; because x—being a single-point slice—must also be a member of G_1, it follows that G_1 is itself downwards closed. L is closed under $\cup$ and $\cap$, and $(L, \cup, \cap)$ is a lattice ordered by set inclusion.

It remains to be shown that $\dot{-}$ has the properties required of a pseudo-difference; that is, we must show (1) L is closed under $\dot{-}$ and (2) for all $a, b, c \in L$, $a \dot{-} b \subseteq c$ iff $a \subseteq b \cup c$.

To show property (1), consider any two elements $a, b \in L$. From the definition of $\dot{-}$ we see that $a \dot{-} b$ is the downwards closure (under the "single-point slice" relation) of $a - b$, and hence $a \dot{-} b \in L$. (Note that $a - b$ denotes the set difference of a and b; $a - b$ is not necessarily downwards closed, and hence, in general, is not a member of L.)

Two show property (2), there are two cases to consider.

$\Rightarrow$ case: Assuming $a \dot{-} b \subseteq c$, we must show that $a \subseteq b \cup c$.

From the definition of $a \dot{-} b$, we know that $a - b \subseteq a \dot{-} b$.

$a - b \subseteq c$	by transitivity
$(a - b) \cup b \subseteq b \cup c$	
$a \cup b \subseteq b \cup c$	because $(a - b) \cup b = a \cup b$
$a \subseteq b \cup c$	because $a \subseteq a \cup b$

$\Leftarrow$ case: Assuming $a \subseteq b \cup c$, we must show that $a \dot{-} b \subseteq c$.

Let z be a member of $a \dot{-} b$; we will show that $z \in c$. From the definition of $\dot{-}$ we know that there exists a (single-point slice) $p \in a$ such that $p \notin b$ and $z \leq p$. By the downwards-closure property of elements of

L, because $p \in a$ we know that $z \in a$ as well. There are two cases to consider:

(i) Suppose $z \notin b$. By the assumption that $a \subseteq b \cup c$ and the fact that $z \in a$, we have $z \in b \cup c$. However, because $z \notin b$, we conclude that $z \in c$.

(ii) Suppose $z \in b$. Consider again the element p, where $p \in a$, $p \notin b$, and $z \leq p$. By the assumption that $a \subseteq b \cup c$ and the fact that $p \in a$, we have $p \in b \cup c$. However, because $p \notin b$, we have $p \in c$. But because $z \leq p$ and because c—being an element of L—is downwards closed, we conclude that $z \in c$.

□

The significance of this algebra is that the program-integration algorithm from [6,7] corresponds very closely to the operation on elements of a Brouwerian algebra defined in Section 5, which is defined solely in terms of $\cup$, $\cap$, and $\dot{-}$.

Recall that in Section 2.2, we stated a requirement that a special program editor be used to create the program variants from the base program. Our assumption about this editor was that it provides a tagging capability so that common components can be identified in all versions. Note, however, that this assumption is not used in the proof of Theorem 1; therefore, a second example of a Brouwerian algebra that can be used for integrating programs is the set of downwards-closed sets of *untagged* single-point slices.

It is also possible to work with the notion of a "slice of a program" rather than that of a "slice of a dependence graph." For example, in [15] a *program slice* is any program whose dependence graph is isomorphic to some (dependence-graph) slice of the original program. For example, when the program

```
program
  x := 0;
  y := 1;
  w := x;
  z := x + y
end
```

is sliced with respect to the statement $z := x + y$, the two possible results are:

```
program
  x := 0;
  y := 1;
  z := x + y
end
```

```
program
  y := 1;
  x := 0;
  z := x + y
end
```

Thus, we can construct a third example of a Brouwerian algebra that can be used for integrating programs by starting with the supposition "let G be the set of well-formed *programs*," rather than "let G be the set of well-formed *program dependence graphs*." In this case, lattice L is the set of downwards-closed sets of programs that correspond to single-point slices. (We refer collectively to all three possible definitions by the term *dependence-graph algebras*.)

When the operation defined in Section 5 to integrate elements of a Brouwerian algebra is used, the three different definitions of dependence-graph algebras correspond to three different variations on the program-integration algorithm given in [6, 7]. Although these variants produce somewhat different answers (both in terms of the final program that is the result of an integration, as well as in their notion of when integrands interfere), a successful integration that results from any of the algorithms satisfies the properties of the Integration Theorem, by essentially the same proof given in [15, 16].

Examples presented in the paper technically follow the last of the three approaches (using sets of program slices), although the points illustrated also apply to the other two, as well. For brevity, examples are presented using individual programs, rather than by listing the program's downwards-closed sets of single-point (program) slices. For instance, the program

```
program
  x := 0;
  y := x;
  z := y;
  w := x
end
```

stands for the following set of slices:

```
{  program   ,  program   ,  program   ,  program   }.
     x := 0       x := 0;      x := 0;      x := 0;
   end            y := x       y := x;      w := x
                  end          z := y     end
                               end
```

Note that the same set of slices also corresponds to the program

```
program
  x := 0;
  y := x;
  z := y;
  w := x;
  x := 0
end.
```

It is instructive to consider how the dependence-graph algebras differ from the (tagged) dependence graphs and operations on them used in Section 2 (which is a lower semi-lattice $(G, \leq)$). The differences are due to the fact that the operation of unioning two dependence graphs is not a join operation for $(G, \leq)$. For example, consider the union of programs a and b, shown below, in the case where $y := x$ has the same tag in both a and b.

```
   a                      b
program                program
  x := 0;                x := 1;
  y := x                 y := x
end                    end
```

The result of $a \cup b$ is an infeasible dependence graph with flow edges from $x := 0$ to $y := x$ and from $x := 1$ to $y := x$. Although both a and b are *subgraphs* of $a \cup b$, neither $a \leq a \cup b$ nor $b \leq a \cup b$ hold, as is required of a join.

In the dependence-graph algebras (both with and without tags) $a \cup b$ corresponds to the following set of slices:

```
a ∪ b  =  {  program   ,  program   ,  program   ,  program   }.
             x := 0       x := 0;      x := 1       x := 1;
           end            y := x     end            y := x
                          end                       end
```

Note that $a \subseteq a \cup b$ and $b \subseteq a \cup b$ both hold.

A Brouwerian algebra is similar, but not identical to a Boolean algebra; some of the properties that hold in a Boolean algebra do not hold in a Brouwerian algebra. (Consequently, applying one's intuition about Boolean algebras to Brouwerian algebras can be risky.)

Example. The laws for distributivity of $\div$ over $\cup$ and $\cap$ for Brouwerian algebras are somewhat different from the laws for distributing – over $\cup$ and $\cap$ in Boolean algebras. Two of the laws are the same:

(16) $(b \div a) \cup (c \div a) = (b \cup c) \div a$

(17) $(c \div a) \cup (c \div b) = c \div (a \cap b)$

However, the laws for distributing $\div$ through $\cap$ on the left and $\cup$ on the right are weaker:[1]

(29) $(a \cap b) \div c \subseteq (a \div c) \cap (b \div c)$

PROOF.

$$\begin{aligned}((a \cap b) \div c) \div ((a \div c) \cap (b \div c)) &= (((a \cap b) \div c) \div (a \div c)) \cup (((a \cap b) \div c) \div (b \div c)) \\ &\qquad \text{by (17)} \\ &= ((a \cap b) \div (c \cup (a \div c))) \cup ((a \cap b) \div (c \cup (b \div c))) \\ &\qquad \text{by (18)} \\ &= ((a \cap b) \div (a \cup c)) \cup ((a \cap b) \div (b \cup c)) \quad \text{by (14)} \\ &= \varnothing \cup \varnothing \quad \text{by (4)} \\ &= \varnothing\end{aligned}$$

Therefore, by (4), $(a \cap b) \div c \subseteq (a \div c) \cap (b \div c)$. □

(32) $c \div (a \cup b) \subseteq (c \div a) \cap (c \div b)$

PROOF.

$c \div (a \cup b) \subseteq c \div a$ by (11)
$c \div (a \cup b) \subseteq c \div b$ by (11)

Therefore, $c \div (a \cup b) \subseteq (c \div a) \cap (c \div b)$. □

We can show by means of examples that the inequalities in laws (29) and (32) are, at times, strict.

a	b	c	$(a \cap b) \div c$	$a \div c$	$b \div c$	$(a \div c) \cap (b \div c)$
program $x := 0;$ $y := x$ **end**	**program** $x := 0;$ $z := x$ **end**	**program** $x := 0$ **end**	$\varnothing$	**program** $x := 0;$ $y := x$ **end**	**program** $x := 0;$ $z := x$ **end**	**program** $x := 0$ **end**

In this example $(a \cap b) \div c \subset (a \div c) \cap (b \div c)$ because the (singleton) slice

program
  $x := 0$
end

occurs in both $a \div c$ and $b \div c$.

---

[1]Note, however, that law (18) does provide an identity that can be used to transform $c \div (a \cup b)$.

(18) $(c \div b) \div a = c \div (a \cup b) = (c \div a) \div b$

a	b	c	$c \div (a \cup b)$	$c \div a$	$c \div b$	$(c \div a) \cap (c \div b)$
program $x := 0;$ $y := x$ **end**	**program** $x := 0;$ $z := x$ **end**	**program** $x := 0;$ $y := x;$ $z := x$ **end**	$\emptyset$	**program** $x := 0;$ $z := x$ **end**	**program** $x := 0;$ $y := x$ **end**	**program** $x := 0$ **end**

In this example $c \div (a \cup b) \subset (c \div a) \cap (c \div b)$ because the (singleton) slice

program
  $x := 0$
end

occurs in both $c \div a$ and $c \div b$.

For all $x, y \in L$, define the operation $x \div y$ to be
$x \div y \triangleq \{ z \in G_1 \mid \forall p \in (y - x)\, p \not\leq z \}$.

THEOREM 2. *$(L, \cup, \cap, \div, \div, G_1)$ is a double Brouwerian algebra.*

PROOF. We must show that $(L, \cap, \cup, \div, \emptyset)$ is a Brouwerian algebra, which involves showing (1) L is closed under $\div$ and (2) for all $a, b, c \in L$, $a \div b \supseteq c$ iff $a \supseteq b \cap c$.

To show property (1), consider any two elements $a, b \in L$. From the definition of $\div$ we see that, for all $q \in a \div b$, if z is a single-point slice of q, z is also a member of $a \div b$, hence $a \div b \in L$.

Two show property (2), there are two cases to consider.

$\Rightarrow$ case: Assuming $a \div b \supseteq c$, we must show that $a \supseteq b \cap c$.

Let $\bar{b}$ be the complement of b with respect to G_1 (*i.e.*, $\bar{b} = G_1 - b$). From the definition of $a \div b$, we know that $\bar{b} \cup a \supseteq a \div b$.

$$\begin{aligned} \bar{b} \cup a &\supseteq a \div b \supseteq c \\ (\bar{b} \cup a) \cap b &\supseteq b \cap c \\ (\bar{b} \cap b) \cup (a \cap b) &\supseteq b \cap c \\ \emptyset \cup (a \cap b) &\supseteq b \cap c \\ a \cap b &\supseteq b \cap c \\ a &\supseteq b \cap c. \end{aligned}$$

$\Leftarrow$ case: Assuming $a \supseteq b \cap c$, we must show that $a \div b \supseteq c$.

Let z be a member of c; we will show that $z \in a \div b$. There are two cases to consider:

(1) Suppose $z \in b$. Because $z \in c$, $z \in b$, and $a \supseteq b \cap c$, we know $z \in a$. By the downwards-closure property of elements of L, $\forall q \in G_1$ such that $q \leq z$, $q \in a$. This means that $q \notin (b - a)$. Thus, $\exists p \in (b - a)$ such that $p \leq z$, or, equivalently, $\forall p \in (b - a)\, p \not\leq z$. Hence, we conclude that $z \in a \div b$.

(2) Suppose $z \notin b$. We first observe that $z \notin (b - a)$. Now suppose there exists a $p \in (b - a)$ such that $p \leq z$ (*). By the downwards-closure property of elements of L, because $p \leq z$ and $z \in c$, we know that $p \in c$. But since $p \in (b - a)$, we also have $p \in b$. Therefore $p \in b \cap c$, which means that $p \in a$ (because $a \supseteq b \cap c$). From $p \in a$ and $p \in b$, we conclude that $p \notin (b - a)$, which contradicts (*). Hence, $\exists p \in (b - a)$ such that $p \leq z$, or, equivalently, $\forall p \in (b - a)\, p \not\leq z$. Consequently, $z \in a \div b$.

□

In this paper, the only results that make use of the quotient operation are in Section 7.3 and Section 7.4.

5. INTEGRATION OF ELEMENTS OF A BROUWERIAN ALGEBRA

We now introduce a ternary operation on elements of a Brouwerian algebra that, for the the dependence-graph algebra discussed in Section 4, corresponds very closely to the ternary operation on dependence graphs used for program integration in [6, 7]. The integration operation on elements of a Brouwerian algebra, denoted by $a\,[b\,]c$, combines two elements a and c with respect to a third element b.

Definition. The *integration* of elements a and c with respect to element b is the element $a\,[b\,]c$ defined by $a\,[b\,]c \triangleq (a \div b) \cup (a \cap b \cap c) \cup (c \div b)$. If $a\,[b\,]c = m$, we refer to element b as the *base*, elements a and c as the *integrands*, and element m as the *result* of the integration.

Because the integration operation is defined solely in terms of $\cup$, $\cap$, and $\div$, it has an analogue in all Brouwerian algebras, not just the dependence-graph algebra from Section 4. Because we will study its properties strictly from an algebraic standpoint, our results apply to this operation in *all* Brouwerian algebras.

We now demonstrate some basic properties of the integration operation.

PROPOSITION. $a\,[a\,]b = b$.

PROOF.

$$\begin{aligned} a\,[a\,]b &= (a \div a) \cup (a \cap a \cap b) \cup (b \div a) \\ &= \varnothing \cup (a \cap b) \cup (b \div a) \\ &= b \qquad \text{by (26)} \end{aligned}$$

□

PROPOSITION. $a\,[b\,]a = a$.

PROOF.

$$\begin{aligned} a\,[b\,]a &= (a \div b) \cup (a \cap b \cap a) \cup (a \div b) \\ &= (a \div b) \cup (a \cap b) \\ &= a \qquad \text{by (26)} \end{aligned}$$

□

PROPOSITION. $a\,[\varnothing]b = a \cup b$.

PROOF.

$$\begin{aligned} a\,[\varnothing]b &= (a \div \varnothing) \cup (a \cap \varnothing \cap b) \cup (b \div \varnothing) \\ &= a \cup b \end{aligned}$$

□

PROPOSITION. $a\,[1]b = a \cap b$.

PROOF.

$$\begin{aligned} a\,[1]b &= (a \div 1) \cup (a \cap 1 \cap b) \cup (b \div 1) \\ &= \varnothing \cup (a \cap b) \cup \varnothing \\ &= a \cap b \end{aligned}$$

□

PROPOSITION. $a\,[b\,](x_1 \cup x_2) = a\,[b\,]x_1 \cup a\,[b\,]x_2$.

PROOF.

$$\begin{aligned} a[b](x_1 \cup x_2) &= (a \div b) \cup (a \cap b \cap (x_1 \cup x_2)) \cup ((x_1 \cup x_2) \div b) \\ &= (a \div b) \cup (a \cap b \cap x_1) \cup (a \cap b \cap x_2) \cup (x_1 \div b) \cup (x_2 \div b) && \text{by (16)} \\ &= a[b]x_1 \cup a[b]x_2 \end{aligned}$$

□

PROPOSITION. $a[b](x_1 \cap x_2) \subseteq a[b]x_1 \cap a[b]x_2$.

PROOF. Because $a \cap b \cap (x_1 \cap x_2) \subseteq a \cap b \cap x_1$ and $(x_1 \cap x_2) \div b \subseteq x_1 \div b$, we have

$$\begin{aligned} a[b](x_1 \cap x_2) &= (a \div b) \cup (a \cap b \cap (x_1 \cap x_2)) \cup ((x_1 \cap x_2) \div b) \\ &\subseteq (a \div b) \cup (a \cap b \cap x_1) \cup (x_1 \div b) \\ &= a[b]x_1 \end{aligned}$$

Therefore, $a[b](x_1 \cap x_2) \subseteq a[b]x_1$. Similarly, we have $a[b](x_1 \cap x_2) \subseteq a[b]x_2$. Consequently, $a[b](x_1 \cap x_2) \subseteq a[b]x_1 \cap a[b]x_2$. □

PROPOSITION. *$a[b]c$ is monotonic in a.*

PROOF. Suppose $a \subseteq a'$. We will show that $a[b]c \subseteq a'[b]c$.

$$\begin{aligned} a[b]c \div a'[b]c &= ((a \div b) \cup (a \cap b \cap c) \cup (c \div b)) \div ((a' \div b) \cup (a' \cap b \cap c) \cup (c \div b)) \\ &= ((a \div b) \div ((a' \div b) \cup (a' \cap b \cap c) \cup (c \div b))) \\ &\quad \cup ((a \cap b \cap c) \div ((a' \div b) \cup (a' \cap b \cap c) \cup (c \div b))) \\ &\quad \cup ((c \div b) \div ((a' \div b) \cup (a' \cap b \cap c) \cup (c \div b))) && \text{by (16)} \\ &= (((a \div b) \div (a' \div b)) \div ((a' \cap b \cap c) \cup (c \div b))) \\ &\quad \cup (((a \cap b \cap c) \div (a' \cap b \cap c)) \div ((a' \div b) \cup (c \div b))) \\ &\quad \cup (((c \div b) \div (c \div b)) \div ((a' \div b) \cup (a' \cap b \cap c))) && \text{by (18)} \\ &= \varnothing \cup \varnothing \cup \varnothing \\ &= \varnothing \end{aligned}$$

Therefore, by (4), $a[b]c \subseteq a'[b]c$. □

PROPOSITION. *$a[b]c$ is antimonotonic in b.*

PROOF. Suppose $b \subseteq b'$. We will show that $a[b']c \subseteq a[b]c$.

$$\begin{aligned} a[b']c \div a[b]c &= ((a \div b') \cup (a \cap b' \cap c) \cup (c \div b')) \div a[b]c \\ &= ((a \div b') \div a[b]c) \cup ((a \cap b' \cap c) \div a[b]c) \cup ((c \div b') \div a[b]c) \\ &\qquad && \text{by (16)} \\ &= ((a \div b') \div ((a \div b) \cup (a \cap b \cap c) \cup (c \div b))) \\ &\quad \cup ((a \cap b' \cap c) \div ((a \div b) \cup (a \cap b \cap c) \cup (c \div b))) \\ &\quad \cup ((c \div b') \div ((a \div b) \cup (a \cap b \cap c) \cup (c \div b))) \\ &= \varnothing \cup ((a \cap b' \cap c) \div ((a \div b) \cup (a \cap b \cap c) \cup (c \div b))) \cup \varnothing && \text{by (11) and (4)} \\ &= ((a \cap b' \cap c) \div (a \cap b \cap c)) \div ((a \cup c) \div b) && \text{by (18) and (16)} \\ &= (((a \cap b' \cap c) \div a) \cup ((a \cap b' \cap c) \div b) \cup ((a \cap b' \cap c) \div c)) \div ((a \cup c) \div b) \\ &\qquad && \text{by (17)} \\ &= (\varnothing \cup ((a \cap b' \cap c) \div b) \cup \varnothing) \div ((a \cup c) \div b) && \text{by (4)} \\ &= ((a \cap b' \cap c) \div b) \div ((a \cup c) \div b) \\ &= (a \cap b' \cap c) \div (b \cup ((a \cup c) \div b)) && \text{by (18)} \\ &= (a \cap b' \cap c) \div (a \cup b \cup c) && \text{by (14)} \\ &= \varnothing \end{aligned}$$

Therefore, by(4), $a[b']c \subseteq a[b]c$. □

PROPOSITION. $a[b]c \div b = (a \div b) \cup (c \div b)$.

PROOF.

$$\begin{aligned} a[b]c \div b &= ((a \div b) \cup (a \cap b \cap c) \cup (c \div b)) \div b \\ &= ((a \div b) \div b) \cup ((a \cap b \cap c) \div b) \cup ((c \div b) \div b) && \text{by (16)} \\ &= (a \div b) \cup \varnothing \cup (c \div b) && \text{by (4)} \\ &= (a \div b) \cup (c \div b) \end{aligned}$$

□

PROPOSITION. $a[b]c \div c = (a \div b) \div c$.

PROOF.

$$\begin{aligned} a[b]c \div c &= ((a \div b) \cup (a \cap b \cap c) \cup (c \div b)) \div c \\ &= ((a \div b) \div c) \cup ((a \cap b \cap c) \div c) \cup ((c \div b) \div c) && \text{by (16)} \\ &= ((a \div b) \div c) \cup \varnothing \cup \varnothing && \text{by (4)} \\ &= (a \div b) \div c \end{aligned}$$

□

6. ASSOCIATIVITY OF INTEGRATION

In this section, we prove an associative law for the integration operation. We also generalize the integration operation to simultaneously integrate more than two variants with a given base element, and show that a three-variant simultaneous integration can be done as a succession of two-variant integrations.

Definition. The *simultaneous integration* of elements $x_1, x_2, \cdots, x_n$ with respect to element b is the element $(x_1[b]x_2, \cdots, x_n)$ defined by

$$(x_1[b]x_2, \cdots, x_n) \triangleq (x_1 \div b) \cup (x_2 \div b) \cup \cdots \cup (x_n \div b) \cup (x_1 \cap x_2 \cap \cdots \cap x_n \cap b).$$

THEOREM 3 (ASSOCIATIVITY THEOREM).

$$(x[b]y)[b]z = x[b](y[b]z) = (x[b]z)[b]y = (x[b]y)[b](x[b]z) = (x[b]y)[x](x[b]z) = (x[b]y, z).$$

PROOF.

Part I. Show that $(x[b]y)[b]z = x[b](y[b]z) = (x[b]z)[b]y = (x[b]y, z)$.

$$\begin{aligned} (x[b]y)[b]z &= (x[b]y \div b) \cup (x[b]y \cap b \cap z) \cup (z \div b) \\ &= (x \div b) \cup (y \div b) \cup (((x \div b) \cup (x \cap b \cap y) \cup (y \div b)) \cap b \cap z) \cup (z \div b) \\ &= (x \div b) \cup (y \div b) \cup (z \div b) \cup (x \cap y \cap z \cap b) \\ &= (x[b]y, z) \end{aligned}$$

By analogous arguments, one can show $x[b](y[b]z) = (x[b]z)[b]y = (x[b]y, z)$.

Part II. Show that $(x[b]y)[b](x[b]z) = (x[b]y, z)$.

$$\begin{aligned} (x[b]y)[b](x[b]z) &= (x[b]y \div b) \cup (x[b]y \cap b \cap x[b]z) \cup (x[b]z \div b) \\ &= (x \div b) \cup (y \div b) \\ &\quad \cup (((x \div b) \cup (x \cap b \cap y) \cup (y \div b)) \cap b \cap ((x \div b) \cup (x \cap b \cap z) \cup (z \div b))) \\ &\quad \cup (x \div b) \cup (z \div b) \\ &= (x \div b) \cup (y \div b) \cup (z \div b) \cup (x \cap y \cap z \cap b) \\ &= (x[b]y, z) \end{aligned}$$

Part III. Show that $(x[b]y)[x](x[b]z) = (x[b]y, z)$.

$$
\begin{aligned}
(x[b]y)[x](x[b]z) &= (x[b]y \div x) \cup (x[b]y \cap x \cap x[b]z) \cup (x[b]z \div x) \\
&= ((y \div b) \div x) \\
&\quad \cup (((x \div b) \cup (x \cap b \cap y) \cup (y \div b)) \cap x \cap ((x \div b) \cup (x \cap b \cap z) \cup (z \div b))) \\
&\quad \cup ((z \div b) \div x) \\
&= ((y \div b) \div x) \\
&\quad \cup ((x \div b) \cap x) \cup ((x \div b) \cap (x \cap b \cap z)) \cup ((x \div b) \cap (z \div b)) \\
&\quad \cup ((x \div b) \cap b \cap y) \cup (x \cap b \cap z \cap y) \cup (x \cap b \cap y \cap (z \div b)) \\
&\quad \cup ((y \div b) \cap (x \div b)) \cup ((y \div b) \cap x \cap b \cap z) \cup ((y \div b) \cap (z \div b) \cap x) \\
&\quad \cup ((z \div b) \div x) \qquad (*)
\end{aligned}
$$

Note that each term in (*) is dominated by a term of $(x \div b) \cup (y \div b) \cup (z \div b) \cup (x \cap y \cap z \cap b)$; thus, $(x[b]y)[x](x[b]z) \subseteq (x[b]y, z)$.

However, by continuing from (*), we can also show that $(x[b]y)[x](x[b]z) \supseteq (x[b]y, z)$.

$$
\begin{aligned}
&= (x \div b) \\
&\quad \cup ((y \div b) \div x) \cup ((y \div b) \cap (x \div b)) \cup ((y \div b) \cap x \cap b \cap z) \cup ((y \div b) \cap (z \div b) \cap x) \\
&\quad \cup ((z \div b) \div x) \cup ((z \div b) \cap (x \div b)) \cup ((z \div b) \cap x \cap b \cap y) \cup ((z \div b) \cap (y \div b) \cap x) \\
&\quad \cup (x \cap b \cap z \cap y) \qquad \text{because } (x \div b) \cap x = x \div b \\
&= (x \div b) \\
&\quad \cup (((y \div b) \cup (z \div b)) \div x) \\
&\quad \cup (((y \div b) \cup (z \div b)) \cap (x \div b)) \\
&\quad \cup ((x \cap b) \cap (((y \div b) \cap z) \cup ((z \div b) \cap y))) \\
&\quad \cup (((y \div b) \cap (z \div b)) \cap x) \\
&\quad \cup (x \cap b \cap z \cap y) \qquad (**)
\end{aligned}
$$

Now consider the second and third terms of the expression in line (**). Their union is of the form $(a \div x) \cup (a \cap (x \div b))$, where $a = (y \div b) \cup (z \div b) = (y \cup z) \div b$.

$$
\begin{aligned}
(a \div x) \cup (a \cap (x \div b)) &\supseteq (a \div x) \cup ((a \cap x) \div b) && \text{by (28)} \\
&\supseteq ((a \div x) \div b) \cup ((a \cap x) \div b) && \text{by (13)} \\
&= ((a \div x) \cup (a \cap x)) \div b && \text{by (16)} \\
&= a \div b && \text{by (26)} \\
&= ((y \cup z) \div b) \div b \\
&= (y \cup z) \div b \\
&= (y \div b) \cup (z \div b)
\end{aligned}
$$

Substituting into (**), we have

$$
\begin{aligned}
(x[b]y)[x](x[b]z) &\supseteq (x \div b) \cup (y \div b) \cup (z \div b) \cup (x \cap y \cap z \cap b) \\
&= (x[b]y, z)
\end{aligned}
$$

We have shown $(x[b]y, z) \supseteq (x[b]y)[x](x[b]z) \supseteq (x[b]y, z)$; thus, $(x[b]y)[x](x[b]z) = (x[b]y, z)$. □

7. COMPATIBLE INTEGRANDS

Program integration deals with the problem of reconciling "competing" modifications to a base program. A different, but related, problem is that of separating *consecutive* edits to a base program into individual edits on the base program.

Example. Consider the case of two consecutive edits to base program O; let $O + \Delta A$ be the result of modifying O, and let $O + \Delta A + \Delta B$ be the result of modifying $O + \Delta A$. By "separating consecutive edits," we mean creating a program $O + \Delta B$ that includes the second modification but not the first. One way of formalizing this goal is to say that we are looking for an integrand $O + \Delta B$ that is compatible with base O, integrand $O + \Delta A$, and result $O + \Delta A + \Delta B$; that is, $O + \Delta B$ should satisfy the equation $(O + \Delta A)[O](O + \Delta B) = O + \Delta A + \Delta B$.

In this section the algebraic approach introduced in the previous sections is used to study this question. The question is posed as "When does there exist an integrand compatible with a given base b, integrand a, and result m?," or, equivalently, "When does there exist an element x such that $a\,[b\,]x = m$?" We show that, if they exist, the solutions to $a\,[b\,]x = m$ form a distributive lattice with a least element and a greatest element, and we give closed formulas for the least and greatest elements.

7.1. Existence of a Compatible Integrand

The theorem proven in this section provides a test for the existence of a compatible integrand. It shows that a solution to the equation $a\,[b\,]x = m$ exists if and only if m itself is a suitable integrand (*i.e.*, if and only if m itself has the property that $a\,[b\,]m = m$).

LEMMA. *$a\,[b\,]x = m$ iff $a \div (m \cup b) = \varnothing$ and $(m \div a) \div (m \div b) = \varnothing$.*

PROOF.

$\Leftarrow$ case: Assuming $a \div (m \cup b) = \varnothing$ and $(m \div a) \div (m \div b) = \varnothing$, we must show that there is a solution to $a\,[b\,]x = m$.

We will show that there is a solution to $a\,[b\,]x = m$ by showing that m itself is a solution (*i.e.*, $a\,[b\,]m = m$). The proof breaks into two parts: in part (i), we show that $a\,[b\,]m \subseteq m$; in part (ii), we show that $a\,[b\,]m \supseteq m$.

(i) We will show that $a\,[b\,]m \subseteq m$.
We start by considering the terms of $a\,[b\,]m$:
$a\,[b\,]m = (a \div b) \cup (a \cap b \cap m) \cup (m \div b)$.

(1) By the properties of $\cap$, we know $a \cap b \cap m \subseteq m$.

(2) Because $a \div (m \cup b) = \varnothing$ we know $a \subseteq m \cup b$, and consequently $a \div b \subseteq m$.

(3) Because $m \subseteq m \cup b$, we know $m \div b \subseteq m$.

Thus, $a\,[b\,]m \subseteq m \cup m \cup m = m$. (†)

(ii) We will show that $a\,[b\,]m \supseteq m$.
From the assumption $(m \div a) \div (m \div b) = \varnothing$, we know that $(m \div a) \subseteq (m \div b)$. (*)

$$\begin{aligned} m \div (a \cap b \cap m) &= (m \div a) \cup (m \div b) \cup (m \div m) && \text{by (17)} \\ &= (m \div a) \cup (m \div b) \cup \varnothing && \text{by (6)} \\ &= (m \div b) && \text{by (*)} \end{aligned}$$

We now start from $m \div b = m \div (a \cap b \cap m)$ and union both sides by $(a \cap b \cap m)$.

$$\begin{aligned} (m \div b) \cup (a \cap b \cap m) &= (m \div (a \cap b \cap m)) \cup (a \cap b \cap m) \\ &= m \cup (a \cap b \cap m) && \text{by (14)} \\ &= m && (**) \end{aligned}$$

$$\begin{aligned} a\,[b\,]m &= (a \div b) \cup (a \cap b \cap m) \cup (m \div b) \\ &\supseteq (a \cap b \cap m) \cup (m \div b) \\ &= m && \text{by (**)} \quad (‡) \end{aligned}$$

Combining (†) and (‡), we have $m \subseteq a\,[b\,]m \subseteq m$; hence, $a\,[b\,]m = m$.

$\Rightarrow$ case: Assuming $a\,[b\,]x = m$, we must s that (i) $a \div (m \cup b) = \varnothing$, and (ii) $(m \div a) \div (m \div b) = \varnothing$.

We consider each case in turn below.

(i) If $a[b]x = m$, then we have

$$\begin{aligned}
\varnothing &= m \div m \\
&= a[b]x \div m \\
&= ((a \div b) \cup (a \cap b \cap x) \cup (x \div b)) \div m \\
&= ((a \div b) \div m) \cup ((a \cap b \cap x) \div m) \cup ((x \div b) \div m) && \text{by (16)} \\
&\supseteq (a \div b) \div m \\
&= a \div (m \cup b) && \text{by (18)}
\end{aligned}$$

Therefore, $a \div (m \cup b) = \varnothing$.

(ii) If $a[b]x = m$, then

$$\begin{aligned}
(m \div a) \div (m \div b) &= (a[b]x \div a) \div (a[b]x \div b) \\
&= [((a \div b) \cup (a \cap b \cap x) \cup (x \div b)) \div a] \\
&\quad \div [((a \div b) \cup (a \cap b \cap x) \cup (x \div b)) \div b] \\
&= [((a \div b) \div a) \cup ((a \cap b \cap x) \div a) \cup ((x \div b) \div a)] \\
&\quad \div [((a \div b) \div b) \cup ((a \cap b \cap x) \div b) \cup ((x \div b) \div b)] \\
& && \text{by (16)}
\end{aligned}$$

It is possible to simplify five of the six terms in the last expression.

$(a \div b) \div a = (a \div a) \div b = \varnothing \div b = \varnothing$ by (18), (6), and (7)

$(a \cap b \cap x) \subseteq a$, so $(a \cap b \cap x) \div a = \varnothing$ by (4)

$(a \div b) \div b = a \div (b \cup b) = a \div b$ by (18)

$(a \cap b \cap x) \subseteq b$, so $(a \cap b \cap x) \div b = \varnothing$ by (4)

$(x \div b) \div b = x \div (b \cup b) = x \div b$ by (18)

Picking up the derivation, we have

$$\begin{aligned}
(m \div a) \div (m \div b) &= [\varnothing \cup \varnothing \cup ((x \div b) \div a)] \div [(a \div b) \cup \varnothing \cup (x \div b)] \\
&= (x \div (a \cup b)) \div ((a \div b) \cup (x \div b)) \\
&= x \div ((a \cup b) \cup (a \div b) \cup (x \div b)) && \text{by (18)} \\
&= x \div (a \cup b \cup (x \div b)) && \text{because, by (13), } a \div b \subseteq a \\
&= ((x \div b) \div (x \div b)) \div a && \text{by (18)} \\
&= \varnothing \div a && \text{by (6)} \\
&= \varnothing && \text{by (7)}
\end{aligned}$$

□

THEOREM 4. *$a[b]x = m$ iff $a[b]m = m$.*

PROOF.

⇐ case:

The proof of this case is immediate: To show that $a[b]x = m$ has a solution we merely choose x to be m.

⇒ case:

If $a[b]x = m$, then by the previous lemma, $a \div (m \cup b) = \varnothing$ and $(m \div a) \div (m \div b) = \varnothing$. As shown in the proof of the ⇐ case of the lemma, if $a \div (m \cup b) = \varnothing$ and $(m \div a) \div (m \div b) = \varnothing$, then $a[b]m = m$.

7.2. Existence of a Minimum Compatible Integrand

In this section, we give a closed formula for the least solution of the equation $a[b]x = m$ and show that it is, in fact, the least solution.

LEMMA. *If* $a\,[b\,]m = m$ *then* $m \div a = m \div (a \cup b)$.

PROOF. Because $a\,[b\,]m = m$, we know from Theorem 4 and the preceding Lemma that $a \div (m \cup b) = \varnothing$ and $(m \div a) \div (m \div b) = \varnothing$.

$$(m \div a) \div (m \div b) = \varnothing$$
$$m \div a \subseteq m \div b$$
$$m \div a = (m \div a) \div a \subseteq (m \div b) \div a = m \div (a \cup b) \qquad (*)$$

However, by (13) we know that

$$m \div a \supseteq (m \div a) \div b = m \div (a \cup b) \qquad (**)$$

Therefore, by (*) and (**), $m \div a = m \div (a \cup b)$. □

LEMMA. *If* $a\,[b\,]m = m$ *then* $(a \div b) \cup (a \cap b \cap m) \cup (m \div a) = m$.

PROOF. The proof breaks into two parts: in part (i), we show that $(a \div b) \cup (a \cap b \cap m) \cup (m \div a) \subseteq m$; in part (ii), we show that $(a \div b) \cup (a \cap b \cap m) \cup (m \div a) \supseteq m$.

(i) Because $a\,[b\,]m = m$, we know that $a \div (m \cup b) = \varnothing$. Consequently, $a \subseteq m \cup b$, or equivalently $a \div b \subseteq m$. Thus, $(a \div b) \cup (a \cap b \cap m) \cup (m \div a) \subseteq m \cup m \cup m = m$.

(ii) By (20), $(a \div b) \cup (m \div a) \supseteq m \div b$; thus, $(a \div b) \cup (a \cap b \cap m) \cup (m \div a) \supseteq (a \cap b \cap m) \cup (m \div b)$. From part (ii) of the $\Leftarrow$ case of the Lemma preceding Theorem 4, we know that from $(m \div a) \div (m \div b) = \varnothing$ we can deduce $(a \cap b \cap m) \cup (m \div b) = m$. Therefore, $(a \div b) \cup (a \cap b \cap m) \cup (m \div a) \supseteq m$. □

Definition. $x_{min} \triangleq (m \div a) \cup ((a \cap b \cap m) \div (a \div b))$.

THEOREM 5. *If* $a\,[b\,]m = m$ *then* x_{min} *is the minimum* x *such that* $a\,[b\,]x = m$.

PROOF.

Part I. Show that $a\,[b\,]x_{min} = m$.

$$a\,[b\,]x_{min} = (a \div b) \cup (a \cap b \cap ((m \div a) \cup ((a \cap b \cap m) \div (a \div b))))$$
$$\cup (((m \div a) \cup ((a \cap b \cap m) \div (a \div b))) \div b) \qquad (*)$$

First, we simplify the second term of (*).

$$\begin{aligned} a \cap b \cap ((m \div a) \cup ((a \cap b \cap m) \div (a \div b))) &= (a \cap b \cap (m \div a)) \cup (a \cap b \cap ((a \cap b \cap m) \div (a \div b))) \\ &= (a \cap b \cap (m \div a)) \cup ((a \cap b \cap m) \div (a \div b)) \\ &\qquad \text{because } a \cap b \supseteq a \cap b \cap m \end{aligned}$$

Next, we simplify the third term of (*).

$$\begin{aligned} ((m \div a) \cup ((a \cap b \cap m) \div (a \div b))) \div b &= ((m \div a) \div b) \cup (((a \cap b \cap m) \div (a \div b)) \div b) \\ &= (m \div (a \cup b)) \cup (((a \cap b \cap m) \div b) \div (a \div b)) \\ &= (m \div (a \cup b)) \cup (\varnothing \div (a \div b)) \\ &= (m \div (a \cup b)) \cup \varnothing \\ &= m \div a \qquad \text{by previous lemma} \end{aligned}$$

$$\begin{aligned} a\,[b\,]x_{min} &= (a \div b) \cup [(a \cap b \cap (m \div a)) \cup ((a \cap b \cap m) \div (a \div b))] \cup (m \div a) \\ &= (a \div b) \cup ((a \cap b \cap m) \div (a \div b)) \cup (m \div a) \cup (a \cap b \cap (m \div a)) \\ &= (a \div b) \cup (a \cap b \cap m) \cup (m \div a) \cup (a \cap b \cap (m \div a)) \qquad \text{by (14)} \end{aligned}$$

By a previous lemma, we know that $(a \div b) \cup (a \cap b \cap m) \cup (m \div a) = m$. Thus we can continue the derivation above as follows:

$$\begin{aligned} &= m \cup (a \cap b \cap (m \div a)) \\ &= m \end{aligned}$$

This completes the proof of Part I; we have shown that $a\,[b\,]x_{min} = m$.

Part II. Show that x_{min} is the minimum x such that $a\,[b\,]x = m$.

Suppose that x is an element such that $a\,[b\,]x = m$. We will demonstrate that $x \supseteq x_{min}$.

$$\begin{aligned}\varnothing &= a\,[b\,]x_{min} \div a\,[b\,]x \\ &= [(a \div b) \cup (a \cap b \cap x_{min}) \cup (x_{min} \div b)] \div [(a \div b) \cup (a \cap b \cap x) \cup (x \div b)] \\ &= [(a \div b) \div ((a \div b) \cup (a \cap b \cap x) \cup (x \div b))] \\ &\qquad \cup [(a \cap b \cap x_{min}) \div ((a \div b) \cup (a \cap b \cap x) \cup (x \div b))] \\ &\qquad \cup [(x_{min} \div b) \div ((a \div b) \cup (a \cap b \cap x) \cup (x \div b))] \qquad (*)\end{aligned}$$

Note that, by (4), the first term equals $\varnothing$. In addition, because the outermost connectives in (*) are joins, each of the remaining two terms must also equal $\varnothing$.

First, consider the third term of (*).

$$\begin{aligned}x_{min} \div b &\subseteq (a \div b) \cup (a \cap b \cap x) \cup (x \div b) \\ x_{min} &\subseteq (a \div b) \cup (a \cap b \cap x) \cup (x \div b) \cup b \\ &= (a \cup b) \cup (a \cap b \cap x) \cup (x \div b) \\ &= a \cup b \cup x \cup (a \cap b \cap x) \\ &= a \cup b \cup x\end{aligned}$$

Therefore, $x_{min} \div (a \cup b) \subseteq x$, or, expanding with the definition of x_{min},

$$((m \div a) \cup ((a \cap b \cap m) \div (a \div b))) \div (a \cup b) \subseteq x.$$

$$\begin{aligned}x &\supseteq ((m \div a) \div (a \cup b)) \cup (((a \cap b \cap m) \div (a \div b)) \div (a \cup b)) \\ &= (((m \div a) \div a) \div b) \cup (((a \cap b \cap m) \div (a \cup b)) \div (a \div b)) \\ &= ((m \div a) \div b) \cup (\varnothing \div (a \div b)) \\ &= (m \div (a \cup b)) \cup \varnothing \\ &= m \div a \qquad \text{by a previous lemma}\end{aligned}$$

Thus, $x \supseteq m \div a$. (†)

Now consider the second term of (*).

$$\begin{aligned}a \cap b \cap x_{min} &\subseteq (a \div b) \cup (a \cap b \cap x) \cup (x \div b) \\ (a \cap b \cap x_{min}) \div (a \div b) &\subseteq (a \cap b \cap x) \cup (x \div b) \subseteq x \\ x &\supseteq (a \cap b \cap x_{min}) \div (a \div b) \\ &= [a \cap b \cap ((m \div a) \cup ((a \cap b \cap m) \div (a \div b)))] \div (a \div b) \\ &= [(a \cap b \cap (m \div a)) \cup (a \cap b \cap ((a \cap b \cap m) \div (a \div b)))] \div (a \div b) \\ &= [(a \cap b \cap (m \div a)) \cup ((a \cap b \cap m) \div (a \div b))] \div (a \div b) \\ &\qquad \text{because } a \cap b \supseteq a \cap b \cap m \\ &= ((a \cap b \cap (m \div a)) \div (a \div b)) \cup (((a \cap b \cap m) \div (a \div b)) \div (a \div b)) \\ &= ((a \cap b \cap (m \div a)) \div (a \div b)) \cup ((a \cap b \cap m) \div (a \div b)) \\ &\supseteq (a \cap b \cap m) \div (a \div b) \qquad (‡)\end{aligned}$$

By (†), $x \supseteq m \div a$.

By (‡), $x \supseteq (a \cap b \cap m) \div (a \div b)$.

Therefore, $x \supseteq (m \div a) \cup ((a \cap b \cap m) \div (a \div b)) = x_{min}$.

□

7.3. Existence of a Maximum Compatible Integrand

In this section, we give a closed formula for the greatest solution of the equation $a\,[b\,]x = m$ and show that it is, in fact, the greatest solution. Note that our formula for the greatest solution makes use of the quotient operation, and thus holds only for double Brouwerian algebras.

Definition. $x_{max} \triangleq m \cup (b \cap (m \div a))$.

THEOREM 6. *If* $a[b]m = m$ *then* x_{max} *is the maximum* x *such that* $a[b]x = m$.

PROOF.

Part I. Show that $a[b]x_{max} = m$.

The proof breaks into two parts: in part (i), we show that $a[b]x_{max} \supseteq m$; in part (ii), we show that $a[b]x_{max} \subseteq m$.

(i)

$$\begin{aligned} a[b]x_{max} &= a[b](m \cup (b \cap (m \div a))) \\ &= a[b]m \cup a[b](b \cap (m \div a)) \\ &= m \cup a[b](b \cap (m \div a)) \\ &\supseteq m \end{aligned} \qquad (*)$$

(ii)

$$\begin{aligned} a[b]x_{max} &= a[b](m \cup (b \cap (m \div a))) \\ &= a[b]m \cup a[b](b \cap (m \div a)) \\ &= m \cup a[b](b \cap (m \div a)) \\ &\subseteq m \cup (a[b]b \cap a[b](m \div a)) \\ &= m \cup (a \cap a[b](m \div a)) \\ &= m \cup (a \cap ((a \div b) \cup (a \cap b \cap (m \div a)) \cup ((m \div a) \div b))) \\ &= m \cup (a \cap (a \div b)) \cup (a \cap a \cap b \cap (m \div a)) \cup (a \cap ((m \div a) \div b)) \end{aligned}$$

Because $a \div b \subseteq a$, the second term $(a \cap (a \div b))$ simplifies to $a \div b$. By (14′), $a \cap (m \div a) = a \cap m$, so the third term simplifies to $a \cap b \cap m$. However, $(a \div b) \cup (a \cap b \cap m) \subseteq a[b]m = m$, so the last line in the above derivation simplifies to $m \cup (a \cap ((m \div a) \div b))$.

Additional simplification is possible because, by (13), we have $(m \div a) \div b \subseteq m \div a$. Therefore, $a \cap ((m \div a) \div b) \subseteq m$, and $a[b]x_{max} \subseteq m$. (**)

Combining (*) and (**), we have $m \subseteq a[b]x_{max} \subseteq m$; hence, $a[b]x_{max} = m$. □

Part II. Show that x_{max} *is the maximum* x *such that* $a[b]x = m$.

Suppose that x is an element such that $a[b]x = m$. We will demonstrate that $x \subseteq x_{max}$.

$$\begin{aligned} m &= a[b]x \\ &= (a \div b) \cup (a \cap b \cap x) \cup (x \div b) \end{aligned}$$

$$\begin{aligned} x \div b &\subseteq m \\ x &\subseteq m \cup b \end{aligned} \qquad (*)$$

$$\begin{aligned} a \cap b \cap x &\subseteq m \\ x &\subseteq m \div (a \cap b) \end{aligned} \qquad (**)$$

Putting (*) and (**) together, we have

$$\begin{aligned} x &\subseteq (m \cup b) \cap (m \div (a \cap b)) \\ &= (m \cap (m \div (a \cap b))) \cup (b \cap (m \div (a \cap b))) \\ &= m \cup (b \cap (m \div (a \cap b))) && \text{by (15′)} \\ &= m \cup (b \cap ((b \cap m) \div (b \cap (a \cap b)))) && \text{by (24′)} \\ &= m \cup (b \cap ((b \cap m) \div (b \cap a))) \\ &= m \cup (b \cap (m \div a)) && \text{by (24′)} \\ &= x_{max} \end{aligned}$$

□

7.4. Properties of Solutions of $a\,[b\,]x = m$

LEMMA. *Solutions of* $a\,[b\,]x = m$ *are closed under* $\cup$.

PROOF. Let x_1 and x_2 be two solutions of $a\,[b\,]x = m$ (*i.e.*, $a\,[b\,]x_1 = m$ and $a\,[b\,]x_2 = m$).

$$\begin{aligned} a\,[b\,](x_1 \cup x_2) &= a\,[b\,]x_1 \cup a\,[b\,]x_2 \\ &= m \cup m \\ &= m \end{aligned}$$

□

LEMMA. *Solutions of* $a\,[b\,]x = m$ *are closed under* $\cap$.

PROOF. Let x_1 and x_2 be two solutions of $a\,[b\,]x = m$ (*i.e.*, $a\,[b\,]x_1 = m$ and $a\,[b\,]x_2 = m$). The proof breaks into two parts: in part (i), we show that $a\,[b\,](x_1 \cap x_2) \subseteq m$; in part (ii), we show that $a\,[b\,](x_1 \cap x_2) \supseteq m$.

(i)

$$\begin{aligned} a\,[b\,](x_1 \cap x_2) &\subseteq a\,[b\,]x_1 \cap a\,[b\,]x_2 \\ &= m \cap m \\ &= m \end{aligned} \qquad (*)$$

(ii) Because $a\,[b\,]x_1 = m$ and $a\,[b\,]x_2 = m$, we know that $x_1 \supseteq x_{min}$ and $x_2 \supseteq x_{min}$; therefore, $x_1 \cap x_2 \supseteq x_{min}$.

$$\begin{aligned} a\,[b\,](x_1 \cap x_2) &= (a \dot{-} b) \cup (a \cap b \cap (x_1 \cap x_2)) \cup ((x_1 \cap x_2) \dot{-} b) \\ &\supseteq (a \dot{-} b) \cup (a \cap b \cap x_{min}) \cup (x_{min} \dot{-} b) \\ &= m \end{aligned} \qquad (**)$$

Combining (*) and (**), we have $m \subseteq a\,[b\,](x_1 \cap x_2) \subseteq m$; hence, $a\,[b\,](x_1 \cap x_2) = m$. □

THEOREM 7. *Solutions of* $a\,[b\,]x = m$ *form a distributive lattice with least element* x_{min} *and greatest element* x_{max}.

PROOF. Immediate from the previous two lemmas, together with Theorem 5 and Theorem 6. □

7.5. Separating Consecutive Edits by Re-Rooting

In this section, we consider a different approach to the problem of separating consecutive edits to a base program into individual edits on the base program.

Example. Consider again the case of two consecutive edits to a base program O, where $O + \Delta A$ is the result of modifying O, $O + \Delta A + \Delta B$ is the result of modifying $O + \Delta A$, and we want to create a program $O + \Delta B$ that includes the second modification but not the first. Our approach is to re-root the development history $O \rightarrow O + \Delta A \rightarrow O + \Delta A + \Delta B$ so that $O + \Delta A$, rather than O, is the base program. Programs O and $O + \Delta A + \Delta B$ are treated as two variants of $O + \Delta A$. For instance, instead of treating the differences between O and $O + \Delta A$ as changes that were made to O to create $O + \Delta A$, they are now treated as changes made to $O + \Delta A$ to create O: when O is the base program, a statement s that occurs in $O + \Delta A$ but not in O is a "new" statement arising from an insertion; when $O + \Delta A$ is the base program, we treat the missing s in O as if a programmer had deleted s from $O + \Delta A$ to create O. (The status of variant $O + \Delta A + \Delta B$ is unchanged; it is still treated as a variant derived from $O + \Delta A$.) $O + \Delta B$ is created by integrating O and $O + \Delta A + \Delta B$ with respect to base program $O + \Delta A$, that is, by performing the integration $O\,[O + \Delta A\,](O + \Delta A + \Delta B)$.

In this section, our algebraic techniques are used to demonstrate that this approach is, in fact, reasonable. We show below that, although in general it does not produce an integrand compatible with base O, integrand $O + \Delta A$, and result $O + \Delta A + \Delta B$ (even when a compatible integrand does exist), the element E

produced, where $E = O\,[O + \Delta A\,](O + \Delta A + \Delta B)$, has the property $(O + \Delta A)[O\,]E \supseteq O + \Delta A + \Delta B$. This property shows that E captures everything that is different between $O + \Delta A + \Delta B$ and $O + \Delta A$ (as desired), plus more.

THEOREM 8. $a\,[b\,]m \subseteq a\,[b\,](m\,[a\,]b)$.

PROOF.

$$\begin{aligned}
a\,[b\,](m\,[a\,]b) &= (a \div b) \cup (a \cap b \cap m\,[a\,]b) \cup (m\,[a\,]b \div b) \\
&= (a \div b) \cup (a \cap b \cap ((m \div a) \cup (a \cap b \cap m) \cup (b \div a))) \cup ((m \div a) \div b) \\
&= (a \div b) \cup (a \cap b \cap (m \div a)) \cup (a \cap b \cap a \cap b \cap m) \cup (a \cap b \cap (b \div a)) \cup ((m \div a) \div b) \\
&= (a \div b) \cup (a \cap b \cap m) \cup (a \cap (b \div a)) \cup ((m \div a) \div b) \\
&= ((a \cup (m \div a)) \div b) \cup (a \cap b \cap m) \cup (a \cap (b \div a)) \\
&= ((m \cup a) \div b) \cup (a \cap b \cap m) \cup (a \cap (b \div a)) \\
&= (m \div a) \cup (m \div b) \cup (a \cap b \cap m) \cup (a \cap (b \div a)) \\
&\supseteq a\,[b\,]m
\end{aligned}$$

□

COROLLARY. *If* $a\,[b\,]m = m$, *then* $m \subseteq a\,[b\,](m\,[a\,]b)$.

PROOF. Immediate from Theorem 8. □

We can show by means of an example that the $\subseteq$ in the above corollary is, at times, strict.

Example. The programs shown below have the properties that $a\,[b\,]m = m$ and $m \subset a\,[b\,](m\,[a\,]b)$.

a	b	m	$a\,[b\,]m$	$m\,[a\,]b$	$a\,[b\,](m\,[a\,]b)$
program $x := 1;$ $y := x$ **end**	**program** $x := 1;$ $y := x;$ $z := y$ **end**	**program** $x := 1;$ $y := 2;$ $w := y$ **end**	**program** $x := 1;$ $y := 2;$ $w := y$ **end**	**program** $x := 1;$ $y := x;$ $z := y;$ $y := 2;$ $w := y$ **end**	**program** $x := 1;$ $y := x;$ $y := 2;$ $w := y$ **end**

Note that $m \subset a\,[b\,](m\,[a\,]b)$ due to the presence of the slice

program
$x := 1;$
$y := x$
end

in $a\,[b\,](m\,[a\,]b)$. Even though this slice is not a subslice of m, it does occur in a, b, and $m\,[a\,]b$; thus, it is part of $a \cap b \cap m\,[a\,]b$ and hence occurs in $a\,[b\,](m\,[a\,]b)$.

8. A COMPATIBLE BASE

We now turn to the question of whether there is a base element compatible with given integrands a and b and result element m; that is, we want to know "When does there exist an element x such that $a\,[x\,]b = m$?"

8.1. Existence of a Compatible Base

Note that, because $a\,[x\,]b$ is anti-monotonic in x, we have

$a \cap b = a\,[1]b \subseteq a\,[x\,]b = m = a\,[x\,]b \subseteq a\,[\varnothing]b = a \cup b$.

LEMMA. *$a[x]b = m$ has a solution for x iff $(a \div (a \div m)) \div (a \cap b) = m \div b$ and $(b \div (b \div m)) \div (b \cap a) = m \div a$.*

PROOF.

$\Rightarrow$ case:

Assuming that $a[x]b = m$ has a solution for x, we must show that $(a \div (a \div m)) \div (a \cap b) = m \div b$ and $(b \div (b \div m)) \div (b \cap a) = m \div a$. The proof breaks into two parts: in part (i), we show that $(a \div (a \div m)) \div (a \cap b) \subseteq m \div b$; in part (ii), we show that $(a \div (a \div m)) \div (a \cap b) \supseteq m \div b$. (The identical arguments with a and b interchanged show that $(b \div (b \div m)) \div (b \cap a) = m \div a$.)

(i)

$$\begin{aligned}(a \div (a \div m)) \div (a \cap b) &= ((a \div (a \div m)) \div a) \cup ((a \div (a \div m)) \div b) \\ &= \varnothing \cup ((a \div b) \div (a \div m)) \\ &= (a \div b) \div (a \div m) \\ &\subseteq m \div b \qquad \text{by (21)}\end{aligned}$$

(ii)

$$\begin{aligned}(a \div (a \div m)) \div (a \cap b) &= ((a \div (a \div m)) \div a) \cup ((a \div (a \div m)) \div b) \\ &= \varnothing \cup ((a \div b) \div (a \div m)) \\ &= (a \div b) \div (a \div m)\end{aligned}$$

$$\begin{aligned}(m \div b) \div ((a \div (a \div m)) \div (a \cap b)) &= (m \div b) \div ((a \div b) \div (a \div m)) \\ &= (a[x]b \div b) \div ((a \div b) \div (a \div m)) \\ &= ((a \div x) \div b) \div ((a \div b) \div (a \div m)) \\ &= ((a \div b) \div x) \div ((a \div b) \div (a \div m)) \\ &\subseteq (a \div m) \div x \qquad \text{by (21)} \\ &= (a \div x) \div m \\ &\subseteq ((a \div x) \div m) \cup ((a \cap x \cap b) \div m) \cup ((b \div x) \div m) \\ &= ((a \div x) \cup (a \cap x \cap b) \cup (b \div x)) \div m \\ &= a[x]b \div m \\ &= m \div m \\ &= \varnothing\end{aligned}$$

Therefore, $(a \div (a \div m)) \div (a \cap b) \supseteq m \div b$. (**)

Putting (*) and (**) together, we have $m \div b \supseteq (a \div (a \div m)) \div (a \cap b) \supseteq m \div b$, hence $(a \div (a \div m)) \div (a \cap b) = m \div b$.

$\Leftarrow$ case:

Assuming that $(a \div (a \div m)) \div (a \cap b) = m \div b$ and $(b \div (b \div m)) \div (b \cap a) = m \div a$, we must show that $a[x]b = m$ has a solution for x. We will show that a solution exists by showing that $(a \div m) \cup (b \div m)$ is a solution.

The proof breaks into two parts: in part (i), we show that $a[(a \div m) \cup (b \div m)]b \subseteq m$; in part (ii), we show that $a[(a \div m) \cup (b \div m)]b \supseteq m$.

(i)

$$\begin{aligned}a[(a \div m) \cup (b \div m)]b &= (a \div ((a \div m) \cup (b \div m))) \cup (a \cap ((a \div m) \cup (b \div m)) \cap b) \\ &\qquad \cup (b \div ((a \div m) \cup (b \div m))) \\ &= ((a \div (a \div m)) \div (b \div m)) \cup (a \cap b \cap (a \div m)) \\ &\qquad \cup (a \cap b \cap (b \div m)) \cup ((b \div (b \div m)) \div (a \div m)) \\ &\subseteq ((a \cap m) \div (b \div m)) \cup (a \cap (b \div m)) \qquad \text{by (30)} \\ &\qquad \cup ((b \cap m) \div (a \div m)) \cup (b \cap (a \div m))\end{aligned}$$

$$
\begin{aligned}
&\subseteq ((a \div (b \div m)) \cap (m \div (b \div m))) \cup (a \cap (b \div m)) && \text{by (29)}\\
&\quad \cup ((b \div (a \div m)) \cap (m \div (a \div m))) \cup (b \cap (a \div m))\\
&= (((a \div (b \div m)) \cup (a \cap (b \div m))) \cap ((m \div (b \div m)) \cup (a \cap (b \div m))))\\
&\quad \cup (((b \div (a \div m)) \cup (b \cap (a \div m))) \cap ((m \div (a \div m)) \cup (b \cap (a \div m))))\\
&= (a \cap ((m \div (b \div m)) \cup (a \cap (b \div m)))) && \text{by (26)}\\
&\quad \cup (b \cap ((m \div (a \div m)) \cup (b \cap (a \div m))))\\
&= ((a \cap (m \div (b \div m))) \cup (a \cap (b \div m)))\\
&\quad \cup ((b \cap (m \div (a \div m))) \cup (b \cap (a \div m)))\\
&= (a \cap ((m \div (b \div m)) \cup (b \div m))) \cup (b \cap ((m \div (a \div m)) \cup (a \div m)))\\
&= (a \cap (m \cup (b \div m))) \cup (b \cap (m \cup (a \div m))) && \text{by (14)}\\
&= (a \cap (m \cup b)) \cup (b \cap (m \cup a)) && \text{by (14)}\\
&= (a \cup b) \cap (a \cup (m \cup a)) \cap (b \cup (m \cup b)) \cap (m \cup a \cup b)\\
&= (a \cup b) \cap (a \cup m) \cap (b \cup m) && \text{because } m \subseteq a \cup b\\
&= (a \cup b) \cap ((a \cap b) \cup (m \cap b) \cup (a \cap m) \cup m)\\
&= (a \cup b) \cap m && \text{because } a \cap b \subseteq m\\
&= m && \text{because } m \subseteq a \cup b
\end{aligned}
$$

Therefore, $m \supseteq a[(a \div m) \cup (b \div m)]b$. (*)

(ii)

$$
\begin{aligned}
a[(a \div m) \cup (b \div m)]b &\supseteq a[a[m]b]b && \text{by the anti-monotonicity of } a[x]b\\
&= a[(a \div m) \cup (a \cap m \cap b) \cup (b \div m)]b\\
&= a[(a \div m) \cup (a \cap b) \cup (b \div m)]b && \text{because } a \cap b \subseteq m\\
&= (a \div ((a \div m) \cup (a \cap b) \cup (b \div m))) \cup (a \cap ((a \div m) \cup (a \cap b) \cup (b \div m)) \cap b)\\
&\quad \cup (b \div ((a \div m) \cup (a \cap b) \cup (b \div m)))\\
&= (((a \div (a \div m)) \div (a \cap b)) \div (b \div m)) \cup (a \cap b)\\
&\quad \cup (((b \div (b \div m)) \div (a \cap b)) \div (a \div m))\\
&= ((m \div b) \div (b \div m)) \cup (a \cap b) \cup ((m \div a) \div (a \div m)) && \text{by the assumptions}\\
&= (m \div b) \cup (a \cap b) \cup (m \div a) && \text{by (33)}\\
&= (m \div (a \cap b)) \cup (a \cap b) && \text{by (17)}\\
&= m
\end{aligned}
$$

Therefore, $a[(a \div m) \cup (b \div m)]b \supseteq m$. (**)

Putting (*) and (**) together, we have $m \supseteq a[(a \div m) \cup (b \div m)]b \supseteq m$, hence, $a[(a \div m) \cup (b \div m)]b = m$. □

8.2. Existence of a Minimum Compatible Base

In this section, we show that when the equation $a[x]b = m$ has a solution for x, $(a \div m) \cup (b \div m)$ is the least solution.

THEOREM 9. *If $a[x]b = m$ has a solution, then $(a \div m) \cup (b \div m)$ is the minimum x such that $a[x]b = m$.*

PROOF. By the proof of the previous lemma, if the equation $a[x]b = m$ has a solution for x, then $(a \div m) \cup (b \div m)$ is a solution to the equation. It remains to be shown that $(a \div m) \cup (b \div m)$ is the *least* solution.

In the following derivation, let x be any solution to the equation $a[x]b = m$.

$$
\begin{aligned}
\varnothing &= m \div m\\
&= a[x]b \div m\\
&= ((a \div x) \cup (a \cap x \cap b) \cup (b \div x)) \div m\\
&= ((a \div x) \div m) \cup ((a \cap x \cap b) \div m) \cup ((b \div x) \div m)\\
&= ((a \div m) \div x) \cup ((a \cap x \cap b) \div m) \cup ((b \div m) \div x)\\
&\supseteq ((a \div m) \div x) \cup ((b \div m) \div x)\\
&= ((a \div m) \cup (b \div m)) \div x
\end{aligned}
$$

Therefore, $((a \div m) \cup (b \div m)) \div x = \emptyset$, and hence $(a \div m) \cup (b \div m) \subseteq x$. □

8.3. Properties of Solutions of $a[x]b = m$

LEMMA. *Solutions of* $a[x]b = m$ *are closed under* $\cap$.

PROOF. Let x_1 and x_2 be two solutions of $a[x]b = m$ (*i.e.*, $a[x_1]b = m$ and $a[x_2]b = m$). The proof breaks into two parts: in part (i), we show that $a[x_1 \cap x_2]b \supseteq m$; in part (ii), we show that $a[x_1 \cap x_2]b \subseteq m$.

(i)

$$\begin{array}{ll} & x_1 \cap x_2 \subseteq x_1 \\ a[x_1 \cap x_2]b \supseteq a[x_1]b & \text{by the anti-monotonicity of } a[x]b \\ = m & (*) \end{array}$$

(ii)

$$\begin{aligned} a[x_1 \cap x_2]b &= (a \div (x_1 \cap x_2)) \cup (a \cap x_1 \cap x_2 \cap b) \cup (b \div (x_1 \cap x_2)) \\ &= (a \div x_1) \cup (a \div x_2) \cup (a \cap x_1 \cap x_2 \cap b) \cup (b \div x_1) \cup (b \div x_2) \\ &\subseteq (a \div x_1) \cup (a \cap x_1 \cap b) \cup (b \div x_1) \cup (a \div x_2) \cup (a \cap x_2 \cap b) \cup (b \div x_2) \\ &= a[x_1]b \cup a[x_2]b \\ &= m \cup m \\ &= m \qquad (**) \end{aligned}$$

Putting (*) and (**) together, we have $m \subseteq a[x_1 \cap x_2]b \subseteq m$; hence, $a[x_1 \cap x_2]b = m$. □

9. RELATION TO PREVIOUS WORK

In unpublished work, Susan Horwitz and I found proofs of several algebraic properties of the program-integration algorithm from [6, 7]. The results given in this paper consist of the analogues for Brouwerian algebras of these earlier results, together with a number of new results. However, the method of proof used in this paper is much different than the proof techniques used to establish our earlier results, which involved complicated arguments—with many sub-cases and argument by *reductio ad absurdum*—about operations on dependence graphs. In contrast, the proofs given in this paper are *strictly algebraic* in nature, making use of the rich set of algebraic laws that hold in Brouwerian algebras.

The work described here was motivated by the desire to find a simpler way to prove properties about program integration. In this, I feel it succeeds—proofs in Brouwerian algebra are less complicated than direct proofs about dependence graphs. It also provides an abstraction for studying common properties of variations on or extensions to the program-integration algorithm. The integration operation in a Brouwerian algebra is defined purely in terms of $\cup$, $\cap$, and $\div$, and thus has an analogue in all Brouwerian algebras. Thus, to show that a proposed program-integration algorithm shares these properties, one merely has to show that the algorithm can be cast as an integration operation in a Brouwerian algebra.

It is important to understand that, because the set of dependence graphs does not form a Brouwerian algebra, not all of the results established about integration in Brouwerian algebras carry over to the program-integration operation of [6, 7]. However, it is possible to generalize that algorithm slightly by constructing a Brouwerian algebra from *sets* of dependence graphs. As shown in Section 4, the downwards-closed sets of single-point slices form a (double) Brouwerian algebra. The operation $a[b]c$ in this algebra is very close to the integration operation of [6, 7]. In this algebra (as well as all other Brouwerian algebras), the operation $a[b]c$ satisfies the properties demonstrated in the paper.

One of the virtues of this generalization of the integration algorithm is that it removes a restriction that is part of the original algorithm. In the original algorithm, it is assumed that a specialized program editor—providing a tagging capability so that common components can be identified in different versions—is used

to create the program variants from the base program. As discussed in Section 4, it is possible to construct dependence-graph algebras for which this restriction is no longer necessary; the elements of these algebras do not have tags on their components. Thus, in principle it is no longer necessary for program integration to be supported by a closed system; programs can be integrated, no matter what their origin.

Some of the properties of $a[b]c$ in Boolean algebras are discussed in [4,5]. The notation $a[b]c$ that has been used here for the integration operation in Brouwerian algebras is taken from that paper.

APPENDIX: ALGEBRAIC LAWS FOR BROUWERIAN ALGEBRAS

This appendix covers algebraic laws that hold for Brouwerian algebras. The material presented here is meant to make the paper self-contained; further information about Brouwerian algebras can be found in [9].

Fundamental Properties of $\div$

(a) $a \div b \subseteq c$ iff $a \subseteq b \cup c$

(b) $a \div b = \min\{ z \mid a \subseteq b \cup z \}$

Algebraic Properties of $\div$

The following properties of Brouwerian algebras are taken from [12] (pp. 59-60); the numbering for laws (4) – (24) follows that given in [12]. (In [12], the laws are expressed in dual form, using a "pseudo-complement" operator, rather than in the form given below, where pseudo-difference is used.)

(4) $b \div a = \emptyset$ iff $a \supseteq b$

(5) $a = b$ iff $b \div a = \emptyset = a \div b$

(6) $a \div a = \emptyset$

(7) $\emptyset \div a = \emptyset$

(8) $b \div \emptyset = b$

(9) $(a \div a) \cup b = b$

(10) $a \cup (b \div a) \supseteq b$

PROOF.

$b \div a = b \div a$

$b \div a \subseteq b \div a$

$b \subseteq (b \div a) \cup a$ by (a)

□

(11) If $a_1 \supseteq a_2$ then $b \div a_2 \supseteq b \div a_1$

PROOF.

$$a_1 \cup (b \div a_2) \underset{\supseteq b}{\overset{a_1 \supseteq a_2}{\supseteq}} a_2 \cup (b \div a_2) \quad \text{by (10)}$$

$a_1 \cup (b \div a_2) \supseteq b$, hence, by fundamental property (a), $b \div a_2 \supseteq b \div a_1$. □

(12) If $b_1 \supseteq b_2$ then $b_1 \div a \supseteq b_2 \div a$

(13) $b \supseteq b \div a$

PROOF. $b \subseteq b \cup a$, hence, by fundamental property (a), $b \div a \subseteq b$. □

(14) $a \cup (b \div a) = a \cup b$

(15) $(b \div a) \cup b = b$

(16) $(b \div a) \cup (c \div a) = (b \cup c) \div a$

(17) $(c \div a) \cup (c \div b) = c \div (a \cap b)$

(18) $(c \div b) \div a = c \div (a \cup b) = (c \div a) \div b$

(19) $a \div c \supseteq (b \div c) \div ((b \div a) \div c)$

(20) $(b \div a) \cup (c \div b) \supseteq c \div a$

(21) $(b \div a) \supseteq (c \div a) \div (c \div b)$

(22) $a \supseteq (a \cup b) \div b$

(23) $(c \div b) \div a \supseteq (c \div a) \div (b \div a)$

PROOF.

$$\begin{aligned} a \cup (b \div a) \cup ((c \div b) \div a) &= a \cup b \cup (c \div b) && \text{by (14)} \\ &= a \cup b \cup c && \text{by (14)} \\ &\supseteq c \end{aligned}$$

$c \subseteq a \cup (b \div a) \cup ((c \div b) \div a)$

$c \div a \subseteq (b \div a) \cup ((c \div b) \div a)$ by (a)

$(c \div a) \div (b \div a) \subseteq ((c \div b) \div a)$ by (a)

□

(24) $c \cup ((c \cup b) \div (c \cup a)) = c \cup (b \div a)$

PROOF.

$$\begin{aligned} c \cup ((c \cup b) \div (c \cup a)) &= c \cup ((c \div (c \cup a)) \cup (b \div (c \cup a))) && \text{by (16)} \\ &= c \cup (\varnothing \cup (b \div (c \cup a))) && \text{by (4)} \\ &= c \cup (b \div (c \cup a)) \\ &= c \cup ((b \div a) \div c) && \text{by (18)} \\ &= c \cup (b \div a) && \text{by (14)} \end{aligned}$$

□

Additional Algebraic Properties of $\div$

(25) $(a \div b) \div (a \cap b) = a \div b$

PROOF.

$$\begin{aligned} (a \div b) \div (a \cap b) &= ((a \div b) \div a) \cup ((a \div b) \div b) && \text{by (17)} \\ &= ((a \div a) \div b) \cup (a \div (b \cup b)) && \text{by (18)} \\ &= (\varnothing \div b) \cup (a \div b) && \text{by (6)} \\ &= \varnothing \cup (a \div b) && \text{by (7)} \\ &= a \div b \end{aligned}$$

□

(26) $(b \div a) \cup (b \cap a) = b$

PROOF.

$$\begin{aligned} (b \div a) \cup (b \cap a) &= ((b \div a) \cup b) \cap ((b \div a) \cup a) \\ &= b \cap (a \cup b) \\ &= b \end{aligned}$$

□

(27) $(c \div b) \div a = (c \div a) \div (b \div a)$ (This is a stronger version of (23).)

PROOF.

(i) $(c \div b) \div a \supseteq (c \div a) \div (b \div a)$ by (23) (*)
(ii) $b \supseteq b \div a$ by (13)
$(c \div a) \div (b \div a) \supseteq (c \div a) \div b$ by (11)
$= (c \div b) \div a$ by (18) (**)

Putting (*) and (**) together, we have $(c \div b) \div a \supseteq (c \div a) \div (b \div a) \supseteq (c \div b) \div a$; therefore, $(c \div b) \div a = (c \div a) \div (b \div a)$. □

(28) $(a \cap b) \div c \subseteq a \cap (b \div c)$

PROOF.

$$\begin{aligned}((a \cap b) \div c) \div (a \cap (b \div c)) &= (((a \cap b) \div c) \div a) \cup (((a \cap b) \div c) \div (b \div c)) \\ &= (((a \cap b) \div a) \div c) \cup ((a \cap b) \div (c \cup (b \div c))) \\ &= (\varnothing \div c) \cup ((a \cap b) \div (b \cup c)) \\ &= \varnothing \cup \varnothing \\ &= \varnothing\end{aligned}$$

Therefore, $(a \cap b) \div c \subseteq a \cap (b \div c)$. □

(29) $(a \cap b) \div c \subseteq (a \div c) \cap (b \div c)$

PROOF.

$$\begin{aligned}((a \cap b) \div c) \div ((a \div c) \cap (b \div c)) &= (((a \cap b) \div c) \div (a \div c)) \cup (((a \cap b) \div c) \div (b \div c)) \\ &= ((a \cap b) \div (c \cup (a \div c))) \cup ((a \cap b) \div (c \cup (b \div c))) \\ &= ((a \cap b) \div (a \cup c)) \cup ((a \cap b) \div (b \cup c)) \\ &= \varnothing \cup \varnothing \\ &= \varnothing\end{aligned}$$

□

(30) $b \div (b \div a) \subseteq a \cap b$

PROOF.

$b = (a \cap b) \cup (b \div a)$ by (26)
$b \subseteq (a \cap b) \cup (b \div a)$
$b \div (b \div a) \subseteq a \cap b$ by (a)

□

(31) If $a \div b \subseteq b$, then $a \subseteq b$

PROOF. $a \div b \subseteq b$, therefore $a \subseteq b \cup b = b$. □

(32) $c \div (a \cup b) \subseteq (c \div a) \cap (c \div b)$

PROOF.

$c \div (a \cup b) \subseteq (c \div a)$
$c \div (a \cup b) \subseteq (c \div b)$

Therefore, $c \div (a \cup b) \subseteq (c \div a) \cap (c \div b)$. □

(33) $(a \div b) \div (b \div a) = a \div b$

PROOF.

(i) $(a \div b) \div (b \div a) \subseteq a \div b$ by (13)
(ii) $(a \div b) \div (b \div a) \supseteq (a \div b) \div b$ by (11) and (13)
$= a \div b$ by (18)

Therefore, $(a \div b) \div (b \div a) = a \div b$. □

(34) $(a \cup b) \div (a \cap b) = (a \div b) \cup (b \div a)$

PROOF.

$$\begin{aligned}(a \cup b) \div (a \cap b) &= (a \div (a \cap b)) \cup (b \div (a \cap b)) && \text{by (16)}\\ &= (a \div a) \cup (a \div b) \cup (b \div a) \cup (b \div b) && \text{by (17)}\\ &= (a \div b) \cup (b \div a) && \text{by (6)}\end{aligned}$$

□

Fundamental Properties of $\div$

(a′) $a \div b \supseteq c$ iff $a \supseteq b \cap c$

(b′) $a \div b = \max\{ z \mid a \supseteq b \cap z \}$

Algebraic Properties of $\div$

(4′) $b \div a = 1$ iff $a \subseteq b$

(5′) $a = b$ iff $b \div a = 1 = a \div b$

(6′) $a \div a = 1$

(7′) $1 \div a = 1$

(8′) $b \div 1 = b$

(9′) $(a \div a) \cap b = b$

(10′) $a \cap (b \div a) \subseteq b$

(11′) If $a_1 \subseteq a_2$ then $b \div a_2 \subseteq b \div a_1$

(12′) If $b_1 \subseteq b_2$ then $b_1 \div a \subseteq b_2 \div a$

(13′) $b \subseteq b \div a$

(14′) $a \cap (b \div a) = a \cap b$

(15′) $(b \div a) \cap b = b$

(16′) $(b \div a) \cap (c \div a) = (b \cap c) \div a$

(17′) $(c \div a) \cap (c \div b) = c \div (a \cup b)$

(18′) $(c \div b) \div a = c \div (a \cap b) = (c \div a) \div b$

(19′) $a \div c \subseteq (b \div c) \div ((b \div a) \div c)$

(20′) $(b \div a) \cap (c \div b) \subseteq c \div a$

(21′) $(b \div a) \subseteq (c \div a) \div (c \div b)$

(22′) $a \subseteq (a \cap b) \div b$

(23′) $(c \div b) \div a \subseteq (c \div a) \div (b \div a)$

(24′) $c \cap ((c \cap b) \div (c \cap a)) = c \cap (b \div a)$

Relationship Between $\div$ and $\div$

PROPOSITION. $(a \div b) \cap (b \div a) = a \cap (b \div a)$.

PROOF. The proof splits into two cases: in part (i), we show that $(a \div b) \cap (b \div a) \subseteq a \cap (b \div a)$; in part (ii), we show that $(a \div b) \cap (b \div a) \supseteq a \cap (b \div a)$.

(i) We will show that $(a \div b) \cap (b \dot{-} a) \subseteq a \cap (b \dot{-} a)$.
$(a \div b) \cap (b \dot{-} a) \subseteq a \div b$, so $((a \div b) \cap (b \dot{-} a)) \cap b \subseteq a$. (*)
$(a \div b) \cap (b \dot{-} a) \subseteq b \dot{-} a \subseteq b$, therefore, $((a \div b) \cap (b \dot{-} a)) \cap b = (a \div b) \cap (b \dot{-} a)$. (**)
Substituting (**) into (*), we have
$(a \div b) \cap (b \dot{-} a) \subseteq a$
Therefore, $(a \div b) \cap (b \dot{-} a) \subseteq a \cap (b \dot{-} a)$. (†)

(ii) We will show that $(a \div b) \cap (b \dot{-} a) \supseteq a \cap (b \dot{-} a)$ by showing that (a) $b \dot{-} a \supseteq a \cap (b \dot{-} a)$, and (b) $a \div b \supseteq a \cap (b \dot{-} a)$.
(a) follows immediately from the properties of $\cap$. To show (b), we have

$$\begin{aligned} a &\supseteq a \\ &\supseteq a \cap (b \dot{-} a) \\ &\supseteq b \cap a \cap (b \dot{-} a) \end{aligned}$$

Therefore, $a \div b \supseteq a \cap (b \dot{-} a)$.
Thus, $(a \div b) \cap (b \dot{-} a) \supseteq a \cap (b \dot{-} a)$. (‡)

Combining (†) and (‡), we have $(a \div b) \cap (b \dot{-} a) = a \cap (b \dot{-} a)$. □

PROPOSITION. $b \cup ((b \div a) \dot{-} a) = b \div a$.

PROOF. The proof splits into two cases: in part (i), we show that $b \cup ((b \div a) \dot{-} a) \subseteq b \div a$; in part (ii), we show that $b \cup ((b \div a) \dot{-} a) \supseteq b \div a$.

(i)

$b \subseteq b \div a$ by (13′)
$(b \div a) \dot{-} a \subseteq b \div a$ by (13)

Therefore, $b \cup ((b \div a) \dot{-} a) \subseteq b \div a$. (*)

(ii)

$b \supseteq a \cap (b \div a)$ by (10′)
$\supseteq (b \div a) \dot{-} ((b \div a) \dot{-} a)$ by (30)

Therefore, $b \cup ((b \div a) \dot{-} a) \supseteq b \div a$. (**)

Putting (*) and (**) together, we have $b \div a \supseteq b \cup ((b \div a) \dot{-} a) \supseteq b \div a$. Therefore, $b \cup ((b \div a) \dot{-} a) = b \div a$. □

COROLLARY. $(b \div a) \dot{-} b \subseteq (b \div a) \dot{-} a$.

PROOF.

$b \div a = b \cup ((b \div a) \dot{-} a)$ by the previous proposition
$b \div a \subseteq b \cup ((b \div a) \dot{-} a)$
$(b \div a) \dot{-} b \subseteq (b \div a) \dot{-} a$ by (a)

□

ACKNOWLEDGEMENTS

Tony Hoare suggested using the notation $a\,[b]\,c$ for the integration operation and pointed out the existence of [4,5], which discusses properties of $a\,[b]\,c$ in Boolean algebras. Susan Horwitz provided many comments and helpful suggestions.

REFERENCES

1. Donzeau-Gouge, V., Huet, G., Kahn, G., and Lang, B., "Programming environments based on structured editors: The MENTOR experience," pp. 128-140 in *Interactive Programming Environments*, ed. D. Barstow, E. Sandewall, and H. Shrobe,McGraw-Hill, New York, NY (1984).

2. Ferrante, J., Ottenstein, K., and Warren, J., "The program dependence graph and its use in optimization," *ACM Transactions on Programming Languages and Systems* **9**(3) pp. 319-349 (July 1987).

3. Habermann, A. N. and Notkin, D., "Gandalf: Software development environments," *IEEE Transactions on Software Engineering* **SE-12**(12) pp. 1117-1127 (December 1986).

4. Hoare, C.A.R., "A couple of novelties in the propositional calculus," *Zeitschr. f. math. Logik und Grundlagen d. Math.* **31** pp. 173-178 (1985).

5. Hoare, C.A.R., "A couple of novelties in the propositional calculus," pp. 325-331 in *Essays in Computing Science*, ed. C.B. Jones,Prentice-Hall, New York, NY (1989).

6. Horwitz, S., Prins, J., and Reps, T., "Integrating non-interfering versions of programs," pp. 133-145 in *Conference Record of the Fifteenth ACM Symposium on Principles of Programming Languages*, (San Diego, CA, January 13-15, 1988), ACM, New York, NY (1988).

7. Horwitz, S., Prins, J., and Reps, T., "Integrating non-interfering versions of programs," To appear in *ACM Trans. Program. Lang. Syst.*, (1989).

8. Kuck, D.J., Kuhn, R.H., Leasure, B., Padua, D.A., and Wolfe, M., "Dependence graphs and compiler optimizations," pp. 207-218 in *Conference Record of the Eighth ACM Symposium on Principles of Programming Languages*, (Williamsburg, VA, January 26-28, 1981), ACM, New York, NY (1981).

9. McKinsey, J.C.C. and Tarski, A., "On closed elements in closure algebras," *Annals of Mathematics* **47**(1) pp. 122-162 (January 1946).

10. Notkin, D., Ellison, R.J., Staudt, B.J., Kaiser, G.E., Kant, E., Habermann, A.N., Ambriola, V., and Montangero, C., Special issue on the GANDALF project, *Journal of Systems and Software* **5**(2)(May 1985).

11. Ottenstein, K.J. and Ottenstein, L.M., "The program dependence graph in a software development environment," *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, (Pittsburgh, PA, Apr. 23-25, 1984), *ACM SIGPLAN Notices* **19**(5) pp. 177-184 (May 1984).

12. Rasiowa, H. and Sikorski, R., *The Mathematics of Metamathematics*, Polish Scientific Publishers, Warsaw (1963).

13. Reps, T. and Teitelbaum, T., "The Synthesizer Generator," *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, (Pittsburgh, PA, Apr. 23-25, 1984), *ACM SIGPLAN Notices* **19**(5) pp. 42-48 (May 1984).

14. Reps, T. and Teitelbaum, T., *The Synthesizer Generator: A System for Constructing Language-Based Editors*, Springer-Verlag, New York, NY (1988).

15. Reps, T. and Yang, W., "The semantics of program slicing," TR-777, Computer Sciences Department, University of Wisconsin, Madison, WI (June 1988).

16. Reps, T. and Yang, W., "The semantics of program slicing and program integration," pp. 360-374 in *Proceedings of the Colloquium on Current Issues in Programming Languages*, (Barcelona, Spain, March 13-17, 1989), *Lecture Notes in Computer Science*, Vol. 352, Springer-Verlag, New York, NY (1989).

17. Weiser, M., "Program slicing," *IEEE Transactions on Software Engineering* **SE-10**(4) pp. 352-357 (July 1984).