Unifying Lazy and Strict Computations

Walter Guttmann

Institut für Programmiermethodik und Compilerbau, Universität Ulm
walter.guttmann@uni-ulm.de

Abstract. Non-strict sequential computations describe imperative programs that can be executed lazily and support infinite data structures. Based on a relational model of such computations we investigate their algebraic properties. We show that they share many laws with conventional, strict computations. We develop a common theory generalising previous algebraic descriptions of strict computation models including partial, total and general correctness and extensions thereof. Due to non-strictness, the iteration underlying loops cannot be described by a unary operation. We propose axioms that generalise the binary operation known from omega algebra, and derive properties of this new operation which hold for both strict and non-strict computations. All algebraic results are verified in Isabelle using its integrated automated theorem provers.

1 Introduction

Previous works show that various sequential computation models can be unified by devising algebraic structures whose operations satisfy a common set of axioms [9, 10, 13, 12, 14]. This unified treatment covers partial-, total- and general-correctness models as well as extensions of them. It provides a common approximation order, a unified semantics of recursion and iteration, and common preconditions, correctness calculi and pre-post specifications. Particular results proved in the unified setting include complex program transformations, refinement laws and separation theorems, such as Kozen's while-program normalisation and Back's atomicity refinement [20, 2].

So far, only models of strict computations have been considered for this unifying algebraic approach. By 'strict' we mean that a computation cannot produce a defined output from an undefined input. This matches the conventional execution of imperative programs: for example, if $A = \mathsf{while}\ \mathit{true}\ \mathsf{do}\ \mathsf{skip}$ is the endless loop or $A = (x := {}^{1}\!/_{0})$ aborts, then $A\ ;\ P = A$ for every program P. However, there are also models of non-strict computations, which can recover from undefined input [8]. In such models, for example, $A\ ;\ (x := 2) = (x := 2)$ holds for either of the above definitions of A, assuming the state contains the single variable x. As elaborated in [8], this makes it possible to construct and compute with infinite data structures.

So far, the investigation of such non-strict computations has been concerned with a relational model [8] and an operational semantics [21]. This paper extends the unifying algebraic approach to cover non-strict computation models

in addition to strict ones. We therefore provide axioms and obtain results which are valid across this wide range of models.

Section 2 recalls the relational model of non-strict computations. In Section 3 we recall the basic algebraic structures that describe sequential, nondeterministic computations. The key observation is that many axioms are valid in both strict and non-strict settings. Section 4 contributes new axioms which uniformly describe the endless loop in these models. They generalise previous unifying algebraic descriptions. In Section 5 we derive the unified semantics of recursion and loops from these new axioms. Due to the weaker setting, the underlying iteration can no longer be described by a unary operation. Section 6 therefore generalises the binary operation known from omega algebra [3]. We contribute axioms for this operation, which hold for both strict and non-strict computations, and establish a collection of its properties. They are applied to derive Back's atomicity refinement theorem, which is proved for the first time in a non-strict setting.

All algebraic results are verified in Isabelle [25] heavily using its integrated automated theorem provers. The proofs are omitted and can be found in the theory files available at http://www.uni-ulm.de/en/in/pm/staff/guttmann/algebra/.

2 A Relational Model of Non-Strict Computations

Our previous work [8] describes in full detail a relational model of non-strict, sequential computations. We recall this model as far as it is necessary for our subsequent algebraic description. For simplicity we assume a homogeneous setting in which variables cannot be added to or removed from the state. The addition of a typing discipline which gives heterogeneous algebras with partial operations is orthogonal to our present aims.

2.1 States

The state of a sequential computation that models an imperative program is given by the values of its variables. Let the variables be $x_1, x_2, \ldots$ which we abbreviate as $\vec{x}$. Associate with each variable x_i its type or range D_i, which is the set of values the variable can take. Each set D_i contains two special elements ∞ and $\lightning$ with the following intuitive meaning:

- If x_i has the value ∞ and this value is needed, the execution of the program does not terminate.
- If x_i has the value $\lightning$ and this value is needed, the execution aborts.

Thus ∞ and $\lightning$ represent the results of non-terminating and undefined computations, respectively. Each set D_i is partially ordered by $\preccurlyeq$ such that ∞ is the least element. For elementary types $\preccurlyeq$ is flat, treating $\lightning$ like any other value different from ∞. The full theory [8] imposes additional structure on D_i to facilitate the construction of sum, product, function and recursive types.

Let $D_I = \prod_{i \in I} D_i$ denote the Cartesian product of the ranges of the variables x_i with $i \in I$ for an index set I. A state is an element $\vec{x}_I \in D_I$ where I indicates the variables comprising the state. The index set I is constant in our homogeneous setting. The partial order $\preccurlyeq$ is lifted componentwise to states.

2.2 Statements

Statements transform states into new states. We use x_i and x_i' to denote the values of the variable x_i before and after execution of a statement. A computation is modelled as a homogeneous relation $R \subseteq D_I \times D_I$ on the state space D_I. An element $(\vec{x}, \vec{x}') \in R$ intuitively means that the execution of R with input $\vec{x}$ may yield the output $\vec{x}'$. Several output values for the same input indicate non-determinism. Note that the componentwise lifted partial order $\preccurlyeq$ is a relation on states.

Computations may be specified by set comprehensions like $\{(\vec{x}, \vec{x}') \mid x_1'{=}x_2\}$ or $\{(\vec{x}, \vec{x}') \mid x_2 = 7\}$. We abbreviate such a comprehension by its constituent predicate, that is, $x_1'{=}x_2$ or $x_2{=}7$ for the preceding examples.

Programming constructs include the following ones:

- The program skip is modelled by the relation $\preccurlyeq$ in order to enforce an upper closure on the image of each state. This is typical for total-correctness approaches [7, 17].
- The assignment $\vec{x} \leftarrow \vec{e}$ is the relation $\preccurlyeq \;;\; (\vec{x}' = \vec{e}) \;;\; \preccurlyeq$ which is the usual assignment composed with $\preccurlyeq$ to obtain upper closure. It follows that $\vec{x}{\leftarrow}\vec{\infty}$ is the universal relation.
- Sequential composition of computations P and Q is their relational composition $P \,;\, Q$.
- Non-deterministic choice between computations P and Q is their union $P \cup Q$.
- The conditional $\mathsf{if}\ b\ \mathsf{then}\ P\ \mathsf{else}\ Q$ is the relation

$$(b{=}\infty \cap \vec{x}{\leftarrow}\vec{\infty}) \cup (b{=}\lightning \cap \vec{x}{\leftarrow}\vec{\lightning}) \cup (b{=}\mathit{true} \cap P) \cup (b{=}\mathit{false} \cap Q)\ .$$

  It distinguishes the four possible values of the condition b in the current state. An outcome of ∞ or $\lightning$ is propagated to the whole state.
- The recursive specification $P = f(P)$ is solved as the greatest fixpoint νf of the function f with respect to the refinement order $\subseteq$. This, too, is typical for total correctness: for example, it follows that the endless loop is the universal relation, which absorbs any computation in a non-deterministic choice and suitably equals $\vec{x}{\leftarrow}\vec{\infty}$.

Expressions in assignments and conditionals are assumed to be $\preccurlyeq$-continuous. Several examples illustrate that computations in this model are non-strict:

- $(x_1, x_2{\leftarrow}\lightning, 2) \;;\; (x_1{\leftarrow}x_2) = (x_1, x_2{\leftarrow}2, 2)$,
- $(\vec{x}{\leftarrow}\vec{\infty}) \;;\; (x_1, x_2{\leftarrow}2, 2) = (x_1, x_2, \vec{x}_{3..}{\leftarrow}2, 2, \vec{\infty})$,
- $(\vec{x}{\leftarrow}\vec{\infty}) \;;\; (\vec{x}{\leftarrow}\vec{e}) = (\vec{x}{\leftarrow}\vec{e})$ if all expressions $\vec{e}$ are constant.

Such computations are similar to Haskell's state transformers [22]. Our relational semantics suits sequential computations better than the λ-calculus, allows non-determinism and distinguishes non-termination from undefinedness.

The following consequences are shown in [8].

Theorem 1. *Consider the above programming constructs.*

1. *Every relation P composed of those constructs satisfies $\preccurlyeq ; P = P = P ; \preccurlyeq$.*
2. *Relations composed of those constructs are total.*
3. *Functions composed of those constructs are $\subseteq$-isotone.*
4. *Functions composed of those constructs without the non-deterministic choice are $\bigcap$-continuous.*

With Theorem 1.1 there are two ways to obtain a monoid structure with respect to sequential composition:

1. For the set of all relations, the identity relation is a neutral element.
2. For the set of relations $\{P \mid \preccurlyeq ; P = P = P ; \preccurlyeq\}$, which includes all those composed of the above programming constructs, the skip program $\preccurlyeq$ is a neutral element.

We treat the first structure in the present paper as this choice simplifies the representation of the conditional. An investigation of the second monoid structure is postponed.

In the remainder of this paper we provide an algebraic description of this computation model as well as other, strict ones. Basic and compound programs are represented by elements of algebraic structures. Operations of these structures represent some of the above programming constructs, the laws of which are specified as axioms:

- Section 3 axiomatises the operations $\cdot$ and $+$ for sequential composition and non-deterministic choice, which yields the refinement order $\leq$.
- Section 5 introduces a general approximation order $\sqsubseteq$, which instantiates to $\geq$ in this model, and axiomatises the $\sqsubseteq$-least fixpoint κf, which instantiates to the $\leq$-greatest fixpoint νf for recursion.

Conditional statements are not represented as above by intersection with condition relations, but by using test elements which act as filters in sequential composition. As usual, while-loops appear as special cases of recursion.

3 Basic Algebraic Structures for Sequential Computations

The model of non-strict computations presented in Section 2 is based on relations and relational operations. We therefore inherit many properties known from relation algebras [27, 23] and similar structures. The same is true for a number of strict computation models including partial-, total- and general-correctness models and various extensions which differ in their treatment of finite, infinite and aborting executions as described in [12, 14].

3.1 Lattice, Semiring and Domain

In particular, both strict and non-strict computations form an algebraic structure $(S, +, \curlywedge, \cdot, 0, 1, \top)$ such that $(S, +, \curlywedge, 0, \top)$ is a bounded distributive lattice and $(S, +, \cdot, 0, 1)$ is a semiring without the right zero law. Their common basis is a bounded join-semilattice, which is a structure $(S, +, 0)$ satisfying the axioms

$$\begin{array}{cc} x + (y + z) = (x + y) + z & x + x = x \\ x + y = y + x & 0 + x = x \end{array}$$

The semilattice order $x \leq y \Leftrightarrow x + y = y$ has least element 0 and least upper bound $+$. A bounded distributive lattice $(S, +, \curlywedge, 0, \top)$ adds a dual bounded meet-semilattice $(S, \curlywedge, \top)$ along with distribution and absorption axioms:

$$\begin{array}{cc} x \curlywedge (y \curlywedge z) = (x \curlywedge y) \curlywedge z & x \curlywedge x = x \\ x \curlywedge y = y \curlywedge x & \top \curlywedge x = x \\ x + (y \curlywedge z) = (x + y) \curlywedge (x + z) & x + (x \curlywedge y) = x \\ x \curlywedge (y + z) = (x \curlywedge y) + (x \curlywedge z) & x \curlywedge (x + y) = x \end{array}$$

The semilattice order has another characterisation $x \leq y \Leftrightarrow x \curlywedge y = x$, the greatest element $\top$ and the greatest lower bound $\curlywedge$. An idempotent semiring $(S, +, \cdot, 0, 1)$ without the right zero law – simply called *semiring* in the remainder of this paper – adds to a bounded join-semilattice $(S, +, 0)$ a monoid $(S, \cdot, 1)$, distribution axioms and a left annihilation axiom:

$$\begin{array}{cc} 1 \cdot x = x & x \cdot (y + z) = (x \cdot y) + (x \cdot z) \\ x \cdot 1 = x & (x + y) \cdot z = (x \cdot z) + (y \cdot z) \\ x \cdot (y \cdot z) = (x \cdot y) \cdot z & 0 \cdot x = 0 \end{array}$$

We conventionally abbreviate $x \cdot y$ as xy. Note that $x0 = 0$ is not an axiom, because in many computation models this law does not hold (although it does for our non-strict computations). The operations $+$, $\curlywedge$ and $\cdot$ are $\leq$-isotone.

In computation models, the operations $+$, $\curlywedge$ and $\cdot$ are instantiated by the non-deterministic choice, intersection and sequential composition. The constants 0, 1 and $\top$ correspond to the empty, identity and universal relations (in some models adapted to satisfy certain healthiness conditions). The order $\leq$ is the subset order and $x \leq y$ expresses that x refines y.

Strict and non-strict models furthermore support the operation d which describes the domain $d(x)$ of a computation x, that is, the initial states from which execution of x is enabled. Semirings with domain have been investigated in [5]; the following axioms for d are taken from [6]:

$$\begin{array}{ccc} x = d(x)x & d(x) \leq 1 & d(xy) = d(xd(y)) \\ & d(0) = 0 & d(x + y) = d(x) + d(y) \end{array}$$

It follows that $(d(S), +, \cdot, 0, 1)$ is a bounded distributive lattice in which the operations $\cdot$ and $\curlywedge$ coincide. We also obtain the Galois connection $d(x) \leq d(y) \Leftrightarrow x \leq d(y)\top$ given by [1]. The explicit characterisation $d(x) = x\top \curlywedge 1$ is not a consequence of the axioms and, actually, it does not hold in some of the strict models. Elements of $d(S)$ represent tests in computation models. Boolean complements of tests can be added but are not required in the present paper.

3.2 Kleene Algebra, Omega Algebra and Itering

A Kleene algebra $(S, +, \cdot, ^*, 0, 1)$ extends a semiring $(S, +, \cdot, 0, 1)$ with an operation * satisfying the following unfold and induction axioms [19]:

$$\begin{array}{ll} 1 + yy^* \leq y^* & z + yx \leq x \Rightarrow y^*z \leq x \\ 1 + y^*y \leq y^* & z + xy \leq x \Rightarrow zy^* \leq x \end{array}$$

It follows that y^*z is the least fixpoint of $\lambda x.yx + z$ and that zy^* is the least fixpoint of $\lambda x.xy + z$. An omega algebra $(S, +, \cdot, ^*, ^\omega, 0, 1)$ extends a Kleene algebra with an operation $^\omega$ satisfying the following axioms [3, 24]:

$$yy^\omega = y^\omega \qquad x \leq yx + z \Rightarrow x \leq y^\omega + y^*z$$

It follows that $y^\omega + y^*z$ is the greatest fixpoint of $\lambda x.yx + z$ and that $\top = 1^\omega$ is the greatest element. For models which require other fixpoints we use the following generalisation of Kleene algebras. An *itering* $(S, +, \cdot, ^\circ, 0, 1)$ extends a semiring with an operation $^\circ$ satisfying the sumstar and productstar equations of [4] and two simulation axioms introduced in [13]:

$$\begin{array}{ll} (x + y)^\circ = (x^\circ y)^\circ x^\circ & zx \leq yy^\circ z + w \Rightarrow zx^\circ \leq y^\circ(z + wx^\circ) \\ (xy)^\circ = 1 + x(yx)^\circ y & xz \leq zy^\circ + w \Rightarrow x^\circ z \leq (z + x^\circ w)y^\circ \end{array}$$

The following result gives five instances of iterings which cover several strict computation models [13]. The element L represents the endless loop as detailed in the Section 4.

Theorem 2. *Iterings have the following instances:*

1. *Every Kleene algebra is an itering using $x^\circ = x^*$.*
2. *Every omega algebra is an itering using $x^\circ = x^\omega 0 + x^*$.*
3. *Every omega algebra with $\top x = \top$ is an itering using $x^\circ = x^\omega + x^*$.*
4. *Every demonic refinement algebra [28] is an itering using $x^\circ = x^\omega$.*
5. *Extended designs [15, 14] form an itering using $x^\circ = d(x^\omega)\mathsf{L} + x^*$.*

However, in Section 6 we shall see that the non-strict model of Section 2 is not a useful itering. Also the properties of L known from the strict models need to be generalised to hold in the non-strict model. Thus the topic of the remainder of this paper is to develop a theory which captures L, recursion and iteration in all models of Theorem 2 and in the non-strict model.

4 Infinite Executions

Many models of sequential computations, in particular our non-strict model of Section 2, can represent infinite executions in addition to finite ones. In some of these models, for example, general-correctness models, this requires the use of an approximation order $\sqsubseteq$ for recursion which is different from the refinement order $\leq$ and typically based on the Egli-Milner order. To express the approximation

order, which we do in Section 5, we have to represent the computation which contains precisely all infinite executions, that is, the endless loop L.

In this section and in the following one we work in a structure S that is a bounded distributive lattice and a semiring and has a domain operation.

The element $\mathsf{L} \in S$ satisfies rather different properties in different computation models; for example, $\mathsf{L} = 0$ in partial-correctness models, $\mathsf{L} = \top$ in total-correctness models and in our non-strict model, and neither holds in general correctness and for extended designs. We therefore carefully choose the following axioms for L so that they hold in all of these models:

$$
\begin{array}{ll}
(\mathsf{L1}) & x\mathsf{L} = x0 + d(x)\mathsf{L} \\
(\mathsf{L2}) & d(\mathsf{L})x \leq xd(\mathsf{L}) \\
(\mathsf{L3}) & d(\mathsf{L})\top \leq \mathsf{L} + d(\mathsf{L}0)\top \\
(\mathsf{L4}) & \mathsf{L}\top \leq \mathsf{L} \\
(\mathsf{L5}) & x0 \curlywedge \mathsf{L} \leq (x \curlywedge \mathsf{L})0
\end{array}
$$

We reuse axioms ($\mathsf{L1}$) and ($\mathsf{L2}$) from our previous unifying treatments of strict computation models [10, 14]. The axioms ($\mathsf{L3}$) and ($\mathsf{L4}$) generalise the property $\mathsf{L}x = \mathsf{L}$ which holds in strict models, but not in our non-strict model. Finally, axiom ($\mathsf{L5}$) is a property of algebras for hybrid systems [18] which satisfy $\mathsf{L} = \top 0$, but the latter holds neither for extended designs nor in our non-strict model.

The following remarks describe the intuition underlying the axioms and their use in the subsequent development.

- Axiom ($\mathsf{L1}$) expresses that intermediate states cannot be observed in infinite executions, which is typical for relational models. A computation x followed by an infinite execution essentially amounts to infinite executions starting from the initial states of x – contained in $d(x)\mathsf{L}$ – up to aborting executions in strict models – contained in $x0$.
- Axiom ($\mathsf{L2}$) is used to show that sequential composition, finite and infinite iteration are $\sqsubseteq$-isotone. All of our models satisfy either $d(\mathsf{L}) = 0$ or $d(\mathsf{L}) = 1$.
- Axiom ($\mathsf{L3}$) is equivalent to the requirement that L is the least element of the approximation order $\sqsubseteq$ which we use for defining the semantics of recursion in Section 5. It is furthermore used to derive the semantics of iteration as a special case of recursion. It can equivalently be stated as $d(\mathsf{L})x \leq \mathsf{L} + d(\mathsf{L}0)x$.
- Axiom ($\mathsf{L4}$) can equivalently be stated as $\mathsf{L}x \leq \mathsf{L}$. This is a key weakening of the previously used axiom $\mathsf{L}x = \mathsf{L}$, which is now recognised to characterise strict computations. The intuitive meaning of $\mathsf{L}x = \mathsf{L}$ is that anything which is supposed to happen 'after' an infinite execution is ignored. Hoare [16] formulates this characteristic strictness property as $\top x = \top$, which is the same in models where $\mathsf{L} = \top$, such as the designs of the Unifying Theories of Programming [17], but different in other models. We use axiom ($\mathsf{L4}$) in particular to show that sequential composition, finite and infinite iteration are $\sqsubseteq$-isotone.
- Axiom ($\mathsf{L5}$) is used in Section 5 to simplify the characterisation of solutions to recursive specifications. Intuitively, in strict models the operation $\cdot 0$ cuts away the finite executions of a computation (unless constraints of the model

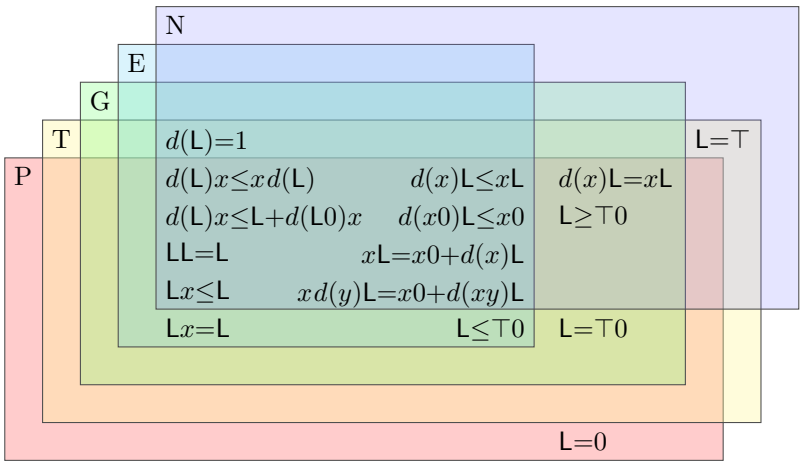
force their presence), and $\curlywedge\mathsf{L}$ keeps only the infinite executions. The axiom's impact is that these two operations commute.

Each of the axioms (L1)–(L3) is independent from the remaining axioms as counterexamples generated by Nitpick or Mace4 witness, but the dependence is unknown for (L4) and (L5). Axioms (L3)–(L5) can be strengthened to equalities as shown by the following lemma, along with further consequences of the above axioms.

Lemma 3.

1. $x0 \curlywedge \mathsf{L} = (x \curlywedge \mathsf{L})0 = d(x0)\mathsf{L}$
2. $\mathsf{L}x \leq \mathsf{L} = \mathsf{LL} = \mathsf{L}\top = \mathsf{L}\top\mathsf{L} = \mathsf{L}d(\mathsf{L})$
3. $d(\mathsf{L})x = (x \curlywedge \mathsf{L}) + d(\mathsf{L}0)x$
4. $d(x\top 0)\mathsf{L} \leq d(x\mathsf{L})\mathsf{L} = d(x)\mathsf{L} = x\top \curlywedge \mathsf{L} \leq x\mathsf{L} \leq x0 + \mathsf{L}$
5. $xd(y)\mathsf{L} = x0 + d(xy)\mathsf{L}$
6. $x \leq y \Leftrightarrow x \leq y + \mathsf{L} \wedge x \leq y + d(y0)\top$
7. $x = x\top \Rightarrow d(\mathsf{L}0)x \leq d((x \curlywedge \mathsf{L})0)\top$

The following Venn diagram shows for each of several properties of L in which computation models it holds (P/T/G/E/N = partial correctness/total correctness/general correctness/extended designs/non-strict):



Only the properties that hold in all models are suitable for our unifying approach, while the others are useful for specialised theories.

5 Recursion

In many models of sequential computations the semantics of the recursive specification $x = f(x)$ is defined as the least fixpoint of the function f with respect to a given approximation order. However, quite different approximation orders are used in the individual models, for example, $\leq$ in partial correctness, $\geq$ in total

correctness and in our non-strict model, and variants of the Egli-Milner order in general correctness and for extended designs. We reuse the following approximation relation $\sqsubseteq$ from our previous unifying treatment of strict computation models [10]:

$$x \sqsubseteq y \Leftrightarrow x \leq y + \mathsf{L} \wedge d(\mathsf{L})y \leq x + d(x0)\top \ .$$

Intuitively, the part $x \leq y + \mathsf{L}$ states that executions may be added and infinite executions may be removed, and the part $d(\mathsf{L})y \leq x + d(x0)\top$ states that this may happen only if x has infinite executions. However, this consideration is somewhat simplified and we must verify that $\sqsubseteq$ instantiates to the correct approximation order for our strict and non-strict models. This is shown in [10] for partial-, total- and general-correctness models; for extended designs this follows by a calculation similar to the one in [14]; finally, our non-strict model has $\mathsf{L} = \top$ and $d(\mathsf{L}) = 1$ and $x0 = 0$, whence $x \sqsubseteq y$ reduces to $x \geq y$ as required.

Because of the weaker axioms (L1)–(L5), new proofs are needed for the following results. The first shows that $\sqsubseteq$ is indeed a partial order.

Theorem 4. *The relation $\sqsubseteq$ is a partial order with least element L. The operations $+$ and $\cdot$ and $\curlywedge\mathsf{L}$ are $\sqsubseteq$-isotone.*

In each of our models, the semantics of the recursion $x = f(x)$ is the $\sqsubseteq$-least fixpoint κf of the function $f : S \to S$. Additionally, let μf denote the $\leq$-least fixpoint of f and νf the $\leq$-greatest one, as specified by the following properties:

$$\begin{array}{ll} f(\kappa f) = \kappa f & f(x) = x \Rightarrow \kappa f \sqsubseteq x \\ f(\mu f) = \mu f & f(x) = x \Rightarrow \mu f \leq x \\ f(\nu f) = \nu f & f(x) = x \Rightarrow \nu f \geq x \end{array}$$

These laws hold if the respective fixpoint exists, which typically requires additional properties of the function f or the structure S, from which we abstract.

Let $x \sqcap y$ denote the $\sqsubseteq$-greatest lower bound of x and y, provided it exists. We can then generalise our previous characterisations of κf [10, 14] to the present setting which covers extended designs and our non-strict model in addition to partial-, total- and general-correctness models.

Theorem 5. *Let $f : S \to S$ be $\leq$- and $\sqsubseteq$-isotone, and assume that μf and νf exist. Then the following are equivalent:*

1. *κf exists.*
2. *κf and $\mu f \sqcap \nu f$ exist and $\kappa f = \mu f \sqcap \nu f$.*
3. *κf exists and $\kappa f = (\nu f \curlywedge \mathsf{L}) + \mu f$.*
4. *$d(\mathsf{L})\nu f \leq (\nu f \curlywedge \mathsf{L}) + \mu f + d(\nu f 0)\top$.*
5. *$d(\mathsf{L})\nu f \leq (\nu f \curlywedge \mathsf{L}) + \mu f + d(((\nu f \curlywedge \mathsf{L}) + \mu f)0)\top$.*
6. *$(\nu f \curlywedge \mathsf{L}) + \mu f \sqsubseteq \nu f$.*
7. *$\mu f \sqcap \nu f$ exists and $\mu f \sqcap \nu f = (\nu f \curlywedge \mathsf{L}) + \mu f$.*
8. *$\mu f \sqcap \nu f$ exists and $\mu f \sqcap \nu f \leq \nu f$.*

To obtain the semantics of a recursion more easily, this theorem reduces the calculation of κf to that of μf and νf, which are extremal fixpoints with respect to $\leq$ instead of the more complex $\sqsubseteq$. The characterisation (4) can be included in the above theorem due to axiom (L5), without which only the more involved characterisation (5) is available. They are helpful because they do not use $\sqsubseteq$ or κf. By requiring the stronger property (2) of the following result, we can generalise the additional characterisations given in [10, 14] to the present setting, too.

Corollary 6. *Let $f : S \to S$ be $\leq$- and $\sqsubseteq$-isotone, and assume that μf and νf exist. Then the following are equivalent and imply the statements of Theorem 5:*

1. *κf exists and $\kappa f = d(\nu f 0)\mathsf{L} + \mu f$.*
2. *$d(\mathsf{L})\nu f \leq \mu f + d(\nu f 0)\top$.*
3. *$d(\nu f 0)\mathsf{L} + \mu f \sqsubseteq \nu f$.*
4. *$\mu f \sqcap \nu f$ exists and $\mu f \sqcap \nu f = d(\nu f 0)\mathsf{L} + \mu f$.*

We instantiate these results about general recursion to the special case of iteration using the Kleene star and omega operations of Section 3.2. The loop $\mathsf{while}\ p\ \mathsf{do}\ w$ is characterised by the unfolding equation

$$\mathsf{while}\ p\ \mathsf{do}\ w = \mathsf{if}\ p\ \mathsf{then}\ (w\ ;\ \mathsf{while}\ p\ \mathsf{do}\ w)\ \mathsf{else}\ \mathsf{skip}\ .$$

Representing the conditional in terms of the non-deterministic choice, this recursion has the form $x = yx + z$, where y and z depend on the condition p and the body w of the loop. The following result shows how to obtain the $\sqsubseteq$-least fixpoint of such a linear characteristic function. It assumes that S is also an omega algebra.

Corollary 7. *Let $y, z \in S$ and $f : S \to S$ with $f(x) = yx + z$. Then $\kappa f = (y^\omega \curlywedge \mathsf{L}) + y^*z = d(y^\omega)\mathsf{L} + y^*z$.*

Finally, we obtain that the various iteration operations are $\sqsubseteq$-isotone, whence by Theorem 4 also the while-loop construct is $\sqsubseteq$-isotone. For this, S has to be also a Kleene algebra, an omega algebra or an itering, respectively.

Theorem 8. *Let $x, y \in S$ such that $x \sqsubseteq y$. Then $x^* \sqsubseteq y^*$ and $x^\omega \sqsubseteq y^\omega$ and $x^\circ \sqsubseteq y^\circ$.*

6 A Binary Operation for Iteration

In strict computation models that form an itering, the iteration underlying loops can be described in terms of a unary operation. Namely, $\mathsf{L}z = \mathsf{L}$ holds in such models, whence for $f(x) = yx + z$ we obtain

$$\kappa f = d(y^\omega)\mathsf{L} + y^*z = d(y^\omega)\mathsf{L}z + y^*z = (d(y^\omega)\mathsf{L} + y^*)z = y^\circ z$$

using the unary operation $y^\circ = d(y^\omega)\mathsf{L} + y^*$ which specialises to the itering operations of Theorem 2 in partial-, total- and general-correctness models [14].

However, this is not the case in our non-strict computation model. In particular, in this model

$$\kappa f = \nu f = y^\omega + y^* z \neq (y^\omega + y^*)z$$

in general, as can be seen by setting $z = 0$ and observing that $x0 = 0$ for every x. The instance $z = 0$ arises for infinite loops such as $\mathsf{while}\ \mathit{true}\ \mathsf{do\ skip}$. The semantics of this loop is $\top$ and not 0 in the non-strict model. Thus loops cannot be represented in the form $y^\circ z$ in this model, no matter how the unary operation $^\circ$ is defined.

In the non-strict model, this problem is solved by using the binary iteration operation of omega algebra, namely $y \star z = y^\omega + y^* z$ [3]. In fact, the non-strict model of Section 2 is an omega algebra in which $x0 = 0$ holds for every x – in contrast to the strict models of total and general correctness as well as extended designs, which do not satisfy this right annihilation property.

On the other hand, omega algebra's binary operation $y \star z = y^\omega + y^* z$ does not describe iteration in several strict computation models, because in general it differs from $y^\circ z = d(y^\omega)\mathsf{L} + y^* z$. A unified description of iteration for strict and non-strict models therefore requires a more general binary operation $\star$, like the one we introduce next.

A *binary itering* $(S, +, \cdot, \star, 0, 1)$ is a semiring $(S, +, \cdot, 0, 1)$ extended with a binary operation $\star$ satisfying the following axioms:

$$\begin{array}{rr} (x + y) \star z = (x \star y) \star (x \star z) & x \star (y + z) = (x \star y) + (x \star z) \\ (xy) \star z = z + x((yx) \star (yz)) & (x \star y)z \leq x \star (yz) \end{array}$$

$$\begin{array}{c} zx \leq y(y \star z) + w \Rightarrow z(x \star v) \leq y \star (zv + w(x \star v)) \\ xz \leq z(y \star 1) + w \Rightarrow x \star (zv) \leq z(y \star v) + (x \star (w(y \star v))) \end{array}$$

These axioms generalise the itering axioms by appropriately composing to an iteration of the form y° a continuation z. The distributivity axiom $x \star (y + z) = (x \star y) + (x \star z)$ and the semi-associativity axiom $(x \star y)z \leq x \star (yz)$ have to be added here, while for the unary operation they follow from the corresponding properties of $\cdot$. The sumstar equation and the first simulation axiom generalise theorems of [3].

To understand the computational meaning of the two simulation axioms they can be seen as generalising the basic simulation laws $zx \leq yz \Rightarrow z(x \star v) \leq y \star (zv)$ and $xz \leq zy \Rightarrow x \star (zv) \leq z(y \star v) + (x \star 0)$, where $x \star 0$ is needed since $\star$ may capture infinite iterations of x. These are similar to simulation laws known in Kleene and omega algebras, where they follow from even simpler induction axioms that characterise the $\leq$-least and $\leq$-greatest fixpoints of linear functions. Because $\star$ is intended for iteration in several computation models that require different fixpoints, we cannot use those induction axioms. However, we might expect a characterisation as the $\sqsubseteq$-least fixpoint of a linear function using a unified approximation order $\sqsubseteq$ such as the one in Section 5. For general-correctness models such properties are shown in [11].

Note that full associativity $(x \star y)z = x \star (yz)$ does not hold in our non-strict model as witnessed by setting $x = 1$ and $z = 0$:

$$(1 \star y)0 = 0 \neq \top = \top + 0 = 1^\omega + 1^* 0 = 1 \star 0 = 1 \star (y0)$$

since binary iteration is $y \star z = y^\omega + y^* z$ in this model. It is therefore not obvious how to generalise the axioms and other formulas from iterings to binary iterings. For example, $y^\circ z$ could be translated to $y \star z$ or to $(y \star 1)z$, and similar options are available for each occurrence of $^\circ$ in a formula. In particular for the axioms, these choices have a critical impact: certain combinations might yield a formula that fails in some target computation models, while another choice might yield a formula too weak to derive a useful theory.

An *extended binary itering* is a binary itering which satisfies the additional axiom

$$w(x \star (yz)) \leq (w(x \star y)) \star (w(x \star y)z) \; .$$

In the special case $w = 1$, it is a substitute for associativity by replacing $x \star (yz)$ with $(x \star y)z$ at the expense of iterating $x \star y$.

The following result shows that binary iterings indeed capture both the non-strict and the strict models.

Theorem 9. *Binary iterings have the following models:*

1. *Every itering is an extended binary itering using $x \star y = x^\circ y$.*
2. *Every omega algebra is an itering using $x \star y = x^\omega + x^* y$.*
3. *Every omega algebra with the additional axiom $x \leq x\top x\top$ is an extended binary itering using $x \star y = x^\omega + x^* y$.*

Part 1 covers the five strict computation models of Theorem 2, including partial, total and general correctness as well as extended designs. Parts 2 and 3 cover our non-strict model of Section 2. In particular, this model satisfies the property $x \leq x\top x\top$. It is a weakening of the 'Tarski rule' of relation algebra, according to which $x = 0$ or $\top x \top = \top$ holds for every x [26], and can equivalently be stated in each of the following forms in omega algebras:

$$\begin{array}{lll} x\top = x\top x\top & x\top = (x\top)^\omega & xy^\omega = (xy^\omega)^\omega \\ x\top \leq x\top x\top & x\top \leq (x\top)^\omega & xy^\omega \leq (xy^\omega)^\omega \\ x \leq x\top x\top & x \leq (x\top)^\omega & \end{array}$$

It implies the law $x^{\omega\omega} = x^\omega$, but not vice versa.

The following result shows a selection of properties which hold in binary iterings and therefore in all of the above computation models. This collection and subsequent ones form a reference to guide the axiomatisation and facilitate program reasoning as in Corollary 13.

Theorem 10. *Let S be a binary itering and $p, w, x, y, z \in S$. Then the following properties 1–56 hold.*

1. $0 \star x = x$
2. $x \leq x \star 1$
3. $y \leq x \star y$
4. $xy \leq x \star y$
5. $x(x \star y) = x \star (xy)$
6. $x(x \star y) \leq x \star y$
7. $(x \star 1)y \leq x \star y$
8. $(xx) \star y \leq x \star y$
9. $x \star x \leq x \star 1$
10. $x \star (x \star y) = x \star y$

11. $(x \star x) \star y = x \star y$
12. $(x(x \star 1)) \star y = x \star y$
13. $(x \star 1)(y \star 1) = x \star (y \star 1)$
14. $1 \star (x \star y) = (x \star 1) \star y$
15. $x \star (1 \star y) = (x \star 1) \star y$
16. $((x \star 1) \star 1) \star y = (x \star 1) \star y$
17. $x \star y = y + x(x \star y)$
18. $x \star y = y + (x \star (xy))$
19. $y + xy + (x \star (x \star y)) = x \star y$
20. $(1 + x) \star y = (x \star 1) \star y$
21. $(x0) \star y = x0 + y$
22. $x + y \leq x \star (y \star 1)$
23. $x \star (x + y) \leq x \star (1 + y)$
24. $(xx) \star ((x + 1)y) \leq x \star y$

For example, property 5 exchanges $\cdot$ with $\star$ and property 10 shows that iteration is transitive. Properties 17 and 18 are unfold laws for the operation $\star$.

25. $(xy) \star (xz) = x((yx) \star z)$
26. $(x \star (y \star 1)) \star z = (y \star (x \star 1)) \star z$
27. $(x \star (y \star 1)) \star z = x \star ((y \star (x \star 1)) \star z)$
28. $(y(x \star 1)) \star z = (y(y \star (x \star 1))) \star z$
29. $x \star (y(z \star 1)) = (x \star y)(z \star 1)$
30. $(x + y) \star z = x \star (y \star ((x + y) \star z))$
31. $(x + y) \star z = (x + y) \star (x \star (y \star z))$
32. $(x + y) \star z \leq (x \star (y \star 1)) \star z$
33. $(x + y0) \star z = x \star (y0 + z)$
34. $x \star z \leq (x + y) \star z$
35. $(xy) \star z \leq (x + y) \star z$
36. $x \star (y \star z) \leq (x + y) \star z$
37. $x \star (y \star z) \leq ((x \star y) \star z) + (x \star z)$
38. $x \star ((y(x \star 1)) \star z) \leq (x + y) \star z$
39. $x \star ((y(x \star 1)) \star z) \leq ((x \star 1)y) \star (x \star z)$
40. $(w(x \star 1)) \star (yz) \leq (x \star w) \star ((x \star y)z)$

Property 25 corresponds to the sliding law of Kleene algebra [19].

41. $x \leq y \Rightarrow x \star z \leq y \star z$
42. $y \leq z \Rightarrow x \star y \leq x \star z$
43. $x \leq y \Rightarrow x \star (y \star z) = y \star z$
44. $x \leq y \Rightarrow y \star (x \star z) = y \star z$
45. $1 \leq x \Rightarrow x(x \star y) = x \star y$
46. $1 \leq z \Rightarrow x \star (yz) = (x \star y)z$
47. $x \leq z \star y \wedge y \leq z \star w \Rightarrow x \leq z \star w$
48. $x \leq z \star 1 \wedge y \leq z \star w \Rightarrow xy \leq z \star w$
49. $yx \leq x \Rightarrow y \star x \leq x + (y \star 0)$
50. $yx \leq xy \Rightarrow (xy) \star z \leq x \star (y \star z)$
51. $yx \leq xy \Rightarrow y \star (x \star z) \leq x \star (y \star z)$
52. $yx \leq xy \Rightarrow (x + y) \star z = x \star (y \star z)$

Properties 41 and 42 state that $\star$ is $\leq$-isotone. Property 46 shows that $\star$ and $\cdot$ associate if the continuation z is above 1. Properties 51 and 52 correspond to basic simulation and separation laws of omega algebra.

53. $yx \leq x(y \star 1) \Rightarrow y \star (x \star z) \leq x \star (y \star z) = (x + y) \star z$
54. $yx \leq x(x \star (1 + y)) \Rightarrow y \star (x \star z) \leq x \star (y \star z) = (x + y) \star z$
55. $y(x \star 1) \leq x \star (y \star 1) \Leftrightarrow y \star (x \star 1) \leq x \star (y \star 1)$
56. $p \leq pp \wedge p \leq 1 \wedge px \leq xp \Rightarrow p(x \star y) = p((px) \star y) = p(x \star (py))$

Properties 53 and 54 sharpen the simulation and separation laws. Property 56 is useful to import and preserve tests in iterations (which can be introduced, for example, using the domain operation of Section 3.1).

It follows that $y \star z$ is a fixpoint of $\lambda x.yx + z$ and that $z(y \star 1)$ is a prefixpoint of $\lambda x.xy + z$. Moreover, if a binary itering has a greatest element $\top$, it satisfies $x \star \top = \top = \top(x \star 1)$.

Properties 10, 17 and 36 of the preceding theorem appear in [3], and properties 53 and 54 generalise theorems therein.

In extended binary iterings, and therefore in all of our computation models, we can add the following properties.

Theorem 11. *Let S be an extended binary itering and $w, x, y, z \in S$. Then the following properties 1–15 hold.*

1. $y((x+y)\star z) \leq (y(x\star 1))\star z$
2. $w(x\star(yz)) \leq (w(x\star y))\star z$
3. $w((x\star(yw))\star z) = w(((x\star y)w)\star z)$
4. $(x\star w)\star(x\star(yz)) = (x\star w)\star((x\star y)z)$
5. $(w(x\star y))\star z = z+w((x+yw)\star(yz))$
6. $x\star((y(x\star 1))\star z) = y\star((x(y\star 1))\star z)$
7. $x\star 0 = 0 \Rightarrow (x\star y)z = x\star(yz)$
8. $(x+y)\star z = x\star((y(x\star 1))\star z)$
9. $(x+y)\star z = ((x\star 1)y)\star(x\star z)$
10. $(x+y)\star z = (x\star y)\star((x\star 1)z)$
11. $(x(y\star 0))\star 0 = x(y\star 0)$
12. $(x\star w)\star(x\star 0) = (x\star w)\star 0$

Property 7 gives another condition under which $\star$ and $\cdot$ associate. Property 8 is the slided version of the sumstar law of Kleene algebra.

13. $w((x\star(yw))\star(x\star(yz))) = w(((x\star y)w)\star((x\star y)z))$
14. $(y(x\star 1))\star z = (y\star z) + (y\star(yx(x\star((y(x\star 1))\star z))))$
15. $x\star((x\star w)\star((x\star y)z)) = (x\star w)\star((x\star y)z)$

It is unknown whether properties 6–9 of the preceding theorem hold in binary iterings. All the other properties do not follow in binary iterings as counterexamples generated by Nitpick or Mace4 witness.

On the other hand, there are properties which are characteristic for the strict or non-strict settings and therefore not suitable for a unifying theory.

Theorem 12. *The following properties 1–6 hold in the model of Theorem 9.1 – extended by $\top$ for the last two – but not in the models of Theorems 9.2 and 9.3.*

1. $(x\star y)z = x\star(yz)$
2. $(x\star 1)y = x\star y$
3. $(x\star 1)x = x\star x$
4. $(x+y)\star z = ((x\star 1)y)\star((x\star 1)z)$
5. $(x\top)\star y = y + x\top y$
6. $\top\star y = \top y$

The following properties 7–12 hold in the models of Theorems 9.2 and 9.3, but not in the model of Theorem 9.1 extended by $\top$.

7. $1\star x = \top$
8. $\top\star x = \top$
9. $x(1\star y) \leq 1\star x$
10. $x = yx \Rightarrow x \leq y\star 1$
11. $x = z + yx \Rightarrow x \leq y\star z$
12. $x \leq z + yx \Rightarrow x \leq y\star z$

The following properties 13–14 hold in the model of Theorem 9.3, but neither in the model of Theorem 9.2 nor in the model of Theorem 9.1 extended by $\top$.

13. $(x\top)\star z = z + x\top$
14. $x\top = x\top x\top$

We thus have the following four variants of the sumstar property, which coincide in our strict models:

- $(x + y) \star z = (x \star y) \star (x \star z)$ (binary itering axiom)
- $(x + y) \star z = ((x \star 1)y) \star (x \star z)$ (Theorem 11.9)
- $(x + y) \star z = (x \star y) \star ((x \star 1)z)$ (Theorem 11.10)
- $(x + y) \star z = ((x \star 1)y) \star ((x \star 1)z)$ (Theorem 12.4)

In contrast to the first three, however, Theorem 12.4 does not hold in the non-strict model. This exemplifies the difficulty in generalising from iterings to binary iterings.

Our final result applies Theorems 10 and 11 to derive Back's atomicity refinement theorem [2, 28]. Because we generalise it to extended binary iterings, it is valid in our non-strict and in several strict computation models. Whether it holds in binary iterings is unknown.

Corollary 13. *Let S be an extended binary itering and $b, l, q, r, s, x, z \in S$ such that*

$$\begin{array}{lllll} s = sq & rb \leq br & rl \leq lr & bl \leq lb & r \star q \leq q(r \star 1) \\ x = qx & qb = 0 & xl \leq lx & ql \leq lq & q \leq 1 \end{array}$$

Then

$$s((x + b + r + l) \star (qz)) \leq s((x(b \star q) + r + l) \star z) \; .$$

7 Conclusion

Strict and non-strict computation models can be unified algebraically. This includes a common approximation order, a common semantics of recursion and an operation describing iteration. Based on this unified treatment common refinement results can be derived.

An issue for future work is how to choose the set of computations that are represented by the algebraic description. For the strict models, only computations satisfying certain healthiness conditions are included. As a consequence, for example, the unit of sequential composition is not the identity relation but modified so as to satisfy the healthiness conditions. For the non-strict model, all computations are included in this paper. It remains to be investigated whether they can be restricted according to Theorem 1.

Acknowledgement. I thank the anonymous referees for valuable comments.

References

1. Aarts, C.J.: Galois connections presented calculationally. Master's thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology (1992)
2. Back, R.J.R., von Wright, J.: Reasoning algebraically about loops. Acta Inf. 36(4), 295–334 (1999)
3. Cohen, E.: Separation and reduction. In: Backhouse, R., Oliveira, J.N. (eds.) MPC 2000. LNCS, vol. 1837, pp. 45–59. Springer (2000)

4. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall (1971)
5. Desharnais, J., Möller, B., Struth, G.: Kleene algebra with domain. ACM Transactions on Computational Logic 7(4), 798–833 (2006)
6. Desharnais, J., Struth, G.: Internal axioms for domain semirings. Sci. Comput. Program. 76(3), 181–203 (2011)
7. Gritzner, T.F., Berghammer, R.: A relation algebraic model of robust correctness. Theor. Comput. Sci. 159(2), 245–270 (1996)
8. Guttmann, W.: Imperative abstractions for functional actions. Journal of Logic and Algebraic Programming 79(8), 768–793 (2010)
9. Guttmann, W.: Partial, total and general correctness. In: Bolduc, C., Desharnais, J., Ktari, B. (eds.) MPC 2010. LNCS, vol. 6120, pp. 157–177. Springer (2010)
10. Guttmann, W.: Unifying recursion in partial, total and general correctness. In: Qin, S. (ed.) UTP 2010. LNCS, vol. 6445, pp. 207–225. Springer (2010)
11. Guttmann, W.: Fixpoints for general correctness. Journal of Logic and Algebraic Programming 80(6), 248–265 (2011)
12. Guttmann, W.: Unifying correctness statements. In: Gibbons, J., Nogueira, P. (eds.) Mathematics of Program Construction. LNCS, vol. 7342, pp. 198–219. Springer (2012)
13. Guttmann, W.: Algebras for iteration and infinite computations. Acta Inf. (to appear 2012)
14. Guttmann, W.: Extended designs algebraically. Sci. Comput. Program. (to appear 2012)
15. Hayes, I.J., Dunne, S.E., Meinicke, L.: Unifying theories of programming that distinguish nontermination and abort. In: Bolduc, C., Desharnais, J., Ktari, B. (eds.) MPC 2010. LNCS, vol. 6120, pp. 178–194. Springer (2010)
16. Hoare, C.A.R.: Theories of programming: Top-down and bottom-up and meeting in the middle. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) FM 1999. LNCS, vol. 1708, pp. 1–27. Springer (1999)
17. Hoare, C.A.R., He, J.: Unifying theories of programming. Prentice Hall Europe (1998)
18. Höfner, P., Möller, B.: An algebra of hybrid systems. Journal of Logic and Algebraic Programming 78(2), 74–97 (2009)
19. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation 110(2), 366–390 (1994)
20. Kozen, D.: Kleene algebra with tests. ACM Trans. Progr. Lang. Syst. 19(3), 427–443 (1997)
21. Lai, A.Y.C.: Operational Semantics and Lazy Execution. Forthcoming PhD thesis, University of Toronto (expected 2012)
22. Launchbury, J., Peyton Jones, S.: State in Haskell. Lisp and Symbolic Computation 8(4), 293–341 (1995)
23. Maddux, R.D.: Relation-algebraic semantics. Theor. Comput. Sci. 160(1–2), 1–85 (1996)
24. Möller, B.: Kleene getting lazy. Sci. Comput. Program. 65(2), 195–214 (2007)
25. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
26. Schmidt, G., Ströhlein, T.: Relationen und Graphen. Springer (1989)
27. Tarski, A.: On the calculus of relations. The Journal of Symbolic Logic 6(3), 73–89 (1941)
28. von Wright, J.: Towards a refinement algebra. Sci. Comput. Program. 51(1–2), 23–45 (2004)