Infinite executions of lazy and strict computations

Walter Guttmann

Department of Computer Science and Software Engineering, University of Canterbury, New Zealand
walter.guttmann@canterbury.ac.nz

Abstract

We give axioms for an operation that describes the states from which a computation has infinite executions in several relational and matrix-based models. The models cover non-strict and strict computations which represent finite, infinite and aborting executions with varying precision. Based on the operation we provide an approximation order for a unified description of recursion. Least fixpoints in the approximation order are reduced to least and greatest fixpoints in the underlying semilattice order. We specialise this to a unified description of iteration which satisfies the axioms of a binary operation introduced in previous work. Previous consequences therefore generalise to all discussed computation models in a uniform way. All algebraic results are verified in Isabelle using its integrated automated theorem provers and SMT solvers.

Keywords: algebraic structures, axiomatic semantics, infinite executions, iteration, lazy execution, non-strict computations, program semantics, relations, sequential computations

1. Introduction

For reasoning about programs it is necessary to specify their semantics. This can be achieved using, for example, operational semantics for the stepwise execution of a program, denotational semantics for the overall behaviour, or axiomatic semantics for the properties of program constructs. While mathematically precise, the semantics will abstract from many details of a program's behaviour as executed on a physical machine. It is therefore not surprising that different approaches exist which describe the semantics of programs with varying precision.

Axiomatic semantics facilitate a unified treatment of the resulting diversity of models, especially if the axioms are wrapped up in algebraic structures [16, 17, 19, 20, 21]. Programs and specifications are elements of the carriers of the algebras; they can be composed using program constructs that are operations of the algebras; their properties arise as axioms and derived theorems. Computation models are characterised by the operations they support and the axioms these operations satisfy. The resulting hierarchy of algebras structures the models, points out connections and helps to avoid the repeated development of similar theories.

In previous works we have developed algebras for unifying various kinds of relational and matrix-based computation models. One approach covers models which differ in the precision with which they describe finite, infinite and aborting executions [19]. Another approach covers both strict and non-strict computations [20]. While some models feature in both approaches, some are only covered by one approach and not the other. The goal of the present paper is to generalise the algebras in order to unify all computation models described in these works.

Computation models treated in [19] describe finite, infinite and aborting executions with varying precision. They include

- partial-correctness models [24, 11, 30, 10, 35], which describe finite executions but neither infinite nor aborting ones,
- total-correctness models [11, 25, 42, 33, 7, 22], which ignore finite executions in the presence of infinite ones but do not distinguish aborting ones,

- general-correctness models [2, 26, 3, 37, 12, 36, 34, 14], which independently describe finite and infinite executions but do not distinguish aborting ones,
- extended designs [23, 21], which ignore finite and infinite executions in the presence of aborting ones, and
- a model which independently describes finite, infinite and aborting executions.

These models are unified by axiomatising two unary operations. The first describes iteration; its axioms capture simulation properties that generalise properties of least and greatest fixpoints. The second operation describes the set of states from which a computation has infinite executions. All models covered by this approach have strict computations; moreover the treatment of recursion in [19] does not account for partial- and total-correctness models.

The approach in [20] describes non-strict computations in addition to some of the above models of strict computations. Non-strict computations can produce defined outputs from undefined inputs, which makes it possible to compute with infinite data structures [15]. Again, two operations are used to unify the models. The first is a binary operation which generalises the above-mentioned unary iteration operation by adding a continuation. The second operation is the domain operation [10] that describes the set of states from which a computation has any executions. Because it does not distinguish between infinite and aborting executions, the domain operation cannot be used to describe recursion in models with independent finite, infinite and aborting executions.

For unifying all of these computation models, we combine and generalise the algebras developed in both previous works. We take the operation from [19] for infinite executions and generalise it so as to apply to models of non-strict computations in addition to strict ones. We generalise the approximation order that is used to describe recursion so that it applies to all of the above models. Instantiating recursion, we derive a unified semantics of iteration that is valid in all these models. We show that the resulting semantics of iteration satisfies the axioms of the binary operation from [20]. The resulting algebras instantiate to all of the above computation models.

In Section 2 we detail the above-mentioned relational and matrix-based models of strict and non-strict computations to which the subsequent development applies. Basic algebraic structures for sequential, non-deterministic computations are recalled in Section 3. We present the unified description of infinite executions, approximation, recursion and iteration in Section 4. We conclude that section with a relation-algebraic representation of the infinite executions of non-strict computations.

Specific contributions of the present paper are:

- A matrix-based representation of non-strict computations that shows their connection to matrix-based models of strict computations; see Section 2.2.
- New axioms for an operation that describes the states from which a computation has infinite executions; see Section 4.1. This is the first axiomatisation of such an operation that covers models of non-strict computations. Moreover, the operation does not have to distribute over non-deterministic choice.
- A generalisation of the Egli-Milner approximation order which unifies further computation models; see Section 4.2. Previous representations of least fixpoints generalise accordingly. Furthermore, a new representation is obtained in terms of lattice operations and the above operation of infinite executions.
- A new instance of binary iterings [20] in terms of the above operation which covers all our computation models; see Section 4.3. Numerous properties of binary iterings therefore apply to these models in a uniform way.
- A relation-algebraic instance of the above operation for models of non-strict computations; see Section 4.4. This is the first representation in terms of relation algebra.

All algebraic results are verified in Isabelle/HOL [38], making heavy use of its integrated automated theorem provers and SMT solvers [39, 5]. The proofs are omitted and can be found in the theory files, which are available at http://www.csse.canterbury.ac.nz/walter.guttmann/algebra/.

2. Relational and matrix-based models of strict and non-strict computations

In this section we describe several computation models that will be the subject of algebraic treatment in the remainder of this paper. The models originate in previous works as referred to in the introduction and below. All models are state-based and feature non-determinism. Let A be the set of states a computation can be in; for example, a state could be given by the values of program variables. Each state may have several possible successor states; an execution of the computation leads to one of these. Depending on the model, the involved choice can be characterised as angelic, demonic or erratic. Deterministic computations have exactly one successor per state; zero or more successors are useful for specification purposes.

All models feature a number of operations acting on computations, which may be used to define program and specification constructs. They include non-deterministic choice, conjunction, sequential composition, various forms of finite and infinite iteration, and an operation to describe the states from which infinite executions exist. Computations are related by two partial orders: refinement and approximation. Recursion is defined by least fixpoints in the approximation order; of particular interest is the semantics of iteration given by the least fixpoints of affine functions. Specific constant computations are identities or annihilators of the operations, and least or greatest elements in the orders. The properties of these constants, operations and relations are stated algebraically in Sections 3 and 4; in the following we describe how they instantiate in the various models.

The models are presented in order of increasing expressiveness; the later models can represent computations with more kinds of execution or fewer constraints than those in the earlier models. However, the models differ essentially as regards the approximation order and hence the semantics of iteration. For example, the endless loop is an identity of non-deterministic choice in the first model, an annihilator in the second model and neither in the third model. To illustrate this, we describe for each model the approximation order $\sqsubseteq$, its least element L which represents the computation with all infinite executions, and the operation n such that $n(x)$ describes the set of states from which the computation x has infinite executions.

2.1. Models of strict computations

In the first model, a computation is a relation over A, that is, a subset R of the Cartesian product $A \times A$. A pair $(x, x') \in R$ means that there is an execution of R starting in state x and ending in state x'. All executions are finite; there are neither infinite nor aborting executions, which renders the model suitable for partial correctness. Accordingly, the set of states from which infinite executions exist is empty for each computation. Non-deterministic choice, conjunction and sequential composition are given by union, intersection and relational composition, respectively. Refinement and approximation are both given by the subset order, that is,

$$R_1 \sqsubseteq R_2 \Leftrightarrow R_1 \subseteq R_2$$

Hence recursions are solved by least fixpoints in the subset order and iteration is given by the reflexive transitive closure of a relation, which is the Kleene star [9, 27]. Specific constants are the empty relation $0 = \emptyset$, the identity relation $1 = \{(x, x) \mid x \in A\}$ and the universal relation $\top = A \times A$. They play their usual roles as identities, annihilators, least and greatest elements. In particular, we obtain the $\sqsubseteq$-least element $\mathsf{L} = 0$ and $n(R) = 0$ as there are no infinite executions in this model.

The second model extends the first model to represent the presence or absence of infinite executions. A computation is a 2×2 matrix whose entries are relations over A. The matrix has the form

$$\begin{pmatrix} \top & \top \\ Q & R \end{pmatrix}$$

where $Q \subseteq R$ and Q is a vector, that is, a relation in which each state is related either to all states or to none. The vector Q represents the set of states from which there are infinite executions. The relation R represents the finite executions in states where infinite executions do not exist, similarly to the first model. Such matrices correspond to the 'designs' of the Unifying Theories of Programming [25, 33, 22]; their particular form is chosen so as to propagate the infinite executions through sequential composition. Aborting executions are not represented or they are identified with infinite executions. Finite executions

are ignored in the presence of infinite ones, which renders the model suitable for total correctness. Non-deterministic choice and conjunction are obtained componentwise. Sequential composition amounts to the matrix product where union and relational composition act as addition and multiplication, respectively. Refinement is given by the componentwise subset order, and approximation by its converse, that is, the componentwise superset order:

$$\begin{pmatrix} \top & \top \\ Q_1 & R_1 \end{pmatrix} \sqsubseteq \begin{pmatrix} \top & \top \\ Q_2 & R_2 \end{pmatrix} \;\Leftrightarrow\; Q_2 \subseteq Q_1 \wedge R_2 \subseteq R_1$$

Hence recursions are solved by greatest fixpoints in the componentwise subset order and iteration is obtained by the omega operation [8]. Specific constants are obtained by setting Q and R as follows:

- $Q = 0$ and $R = 0$ yields the computation with no executions. It is the identity of non-deterministic choice, the annihilator of conjunction, a left-annihilator of sequential composition, the least element in the refinement order and the greatest in the approximation order.
- $Q = 0$ and $R = 1$ yields the computation that does not change the state. It is the identity of sequential composition.
- $Q = 0$ and $R = \top$ yields the computation with all finite executions.
- $Q = \top$ and $R = \top$ yields the computation with all executions. In this model, it is the same as the computation with all infinite executions L. It is the annihilator of non-deterministic choice, the identity of conjunction, a left-annihilator of sequential composition, the greatest element in the refinement order and the least in the approximation order.

In particular, we obtain the $\sqsubseteq$-least element L and the operation n as follows:

$$\mathsf{L} = \begin{pmatrix} \top & \top \\ \top & \top \end{pmatrix} \qquad n\begin{pmatrix} \top & \top \\ Q & R \end{pmatrix} = \begin{pmatrix} \top & \top \\ 0 & Q \cap 1 \end{pmatrix}$$

Note that $n(x)$ is a test, which is an element below the identity of sequential composition in the refinement order. The intersection $Q \cap 1$ converts the vector Q to a relation below the identity relation; it represents the states from which x has infinite executions.

The third model represents finite and infinite executions independently. It uses 2×2 matrices of the form

$$\begin{pmatrix} \top & 0 \\ Q & R \end{pmatrix}$$

where Q is a vector. Again, Q represents the set of states from which there are infinite executions and R represents the finite executions. These matrices correspond to the 'prescriptions' of the Unifying Theories of Programming [12, 33]. Aborting executions are not represented or they are identified with infinite executions, but finite and infinite executions are independent, which renders the model suitable for general correctness [26]. Non-deterministic choice, conjunction, sequential composition and refinement are as in the second model. Approximation, however, is given by the Egli-Milner order $\sqsubseteq$:

$$\begin{pmatrix} \top & 0 \\ Q_1 & R_1 \end{pmatrix} \sqsubseteq \begin{pmatrix} \top & 0 \\ Q_2 & R_2 \end{pmatrix} \;\Leftrightarrow\; Q_2 \subseteq Q_1 \wedge R_1 \subseteq R_2 \subseteq R_1 \cup Q_1$$

Recursions are solved by least fixpoints in this order. Specific constants are obtained by setting Q and R as follows:

- $Q = 0$ and $R = 0$ yields the computation with no executions. It is the identity of non-deterministic choice, the annihilator of conjunction, a left-annihilator of sequential composition and the least element in the refinement order.

* $Q = 0$ and $R = 1$ yields the computation that does not change the state. It is the identity of sequential composition.
* $Q = 0$ and $R = \top$ yields the computation with all finite executions.
* $Q = \top$ and $R = 0$ yields the computation with all infinite executions L. It is a left-annihilator of sequential composition and the least element in the approximation order.
* $Q = \top$ and $R = \top$ yields the computation with all executions. It is the annihilator of non-deterministic choice, the identity of conjunction and the greatest element in the refinement order.

The $\sqsubseteq$-least element L and the operation n are as follows:

$$\mathsf{L} = \begin{pmatrix} \top & 0 \\ \top & 0 \end{pmatrix} \qquad n\begin{pmatrix} \top & 0 \\ Q & R \end{pmatrix} = \begin{pmatrix} \top & 0 \\ 0 & Q \cap 1 \end{pmatrix}$$

The fourth model extends the previous models to represent the presence or absence of aborting executions. It uses 3×3 matrices of the form

$$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ P & Q & R \end{pmatrix}$$

where P and Q are vectors and $P \subseteq Q$ and $P \subseteq R$. The vector P represents the set of states from which there are aborting executions. In their presence, the infinite executions Q and the finite executions R are ignored. These matrices correspond to 'extended designs' [23, 21]. Non-deterministic choice, conjunction, sequential composition and refinement are as in the second model. The approximation order in this model is a variation of the Egli-Milner order:

$$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ P_1 & Q_1 & R_1 \end{pmatrix} \sqsubseteq \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ P_2 & Q_2 & R_2 \end{pmatrix} \;\Leftrightarrow\; Q_2 \subseteq Q_1 \wedge P_1 \subseteq P_2 \wedge R_1 \subseteq R_2 \subseteq R_1 \cup Q_1$$

Specific constants are obtained by setting P, Q and R as follows:

* $P = 0$ and $Q = 0$ and $R = 0$ yields the computation with no executions. It is the identity of non-deterministic choice, the annihilator of conjunction, a left-annihilator of sequential composition and the least element in the refinement order.
* $P = 0$ and $Q = 0$ and $R = 1$ yields the computation that does not change the state. It is the identity of sequential composition.
* $P = 0$ and $Q = 0$ and $R = \top$ yields the computation with all finite executions.
* $P = 0$ and $Q = \top$ and $R = 0$ yields the computation with all infinite executions L. It is a left-annihilator of sequential composition and the least element in the approximation order.
* $P = 0$ and $Q = \top$ and $R = \top$ yields the computation with all finite and all infinite executions.
* $P = \top$ and $Q = \top$ and $R = \top$ yields the computation with all executions. In this model, it is the same as the computation with all aborting executions. It is the annihilator of non-deterministic choice, the identity of conjunction, a left-annihilator of sequential composition and the greatest element in the refinement order.

In this model, the $\sqsubseteq$-least element L and the operation n are:

$$\mathsf{L} = \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & \top & 0 \end{pmatrix} \qquad n\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ P & Q & R \end{pmatrix} = \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & 0 & Q \cap 1 \end{pmatrix}$$

In the fifth model, aborting, finite and infinite executions are independent [19]. It uses 3×3 matrices of the form

$$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ P & Q & R \end{pmatrix}$$

where P and Q are vectors. Again, P represents the states from which there are aborting executions, Q represents the set of states from which there are infinite executions and R represents the finite executions. Non-deterministic choice, conjunction, sequential composition and refinement are as in the second model. Another variation of the Egli-Milner order is used for approximation:

$$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ P_1 & Q_1 & R_1 \end{pmatrix} \sqsubseteq \begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ P_2 & Q_2 & R_2 \end{pmatrix} \;\Leftrightarrow\; Q_2 \subseteq Q_1 \wedge P_1 \subseteq P_2 \subseteq P_1 \cup Q_1 \wedge R_1 \subseteq R_2 \subseteq R_1 \cup Q_1$$

Specific constants are obtained by setting P, Q and R as follows:

* $P = 0$ and $Q = 0$ and $R = 0$ yields the computation with no executions. It is the identity of non-deterministic choice, the annihilator of conjunction, a left-annihilator of sequential composition and the least element in the refinement order.
* $P = 0$ and $Q = 0$ and $R = 1$ yields the computation that does not change the state. It is the identity of sequential composition.
* $P = 0$ and $Q = 0$ and $R = \top$ yields the computation with all finite executions.
* $P = 0$ and $Q = \top$ and $R = 0$ yields the computation with all infinite executions L. It is a left-annihilator of sequential composition and the least element in the approximation order.
* $P = 0$ and $Q = \top$ and $R = \top$ yields the computation with all finite and all infinite executions.
* $P = \top$ and $Q = 0$ and $R = 0$ yields the computation with all aborting executions. It is a left-annihilator of sequential composition.
* $P = \top$ and $Q = 0$ and $R = \top$ yields the computation with all aborting and all finite executions.
* $P = \top$ and $Q = \top$ and $R = 0$ yields the computation with all aborting and all infinite executions. It is a left-annihilator of sequential composition.
* $P = \top$ and $Q = \top$ and $R = \top$ yields the computation with all executions. It is the annihilator of non-deterministic choice, the identity of conjunction and the greatest element in the refinement order.

In the fifth model, we obtain the $\sqsubseteq$-least element L and the operation n as follows:

$$\mathsf{L} = \begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & \top & 0 \end{pmatrix} \qquad n\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ P & Q & R \end{pmatrix} = \begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & 0 & Q \cap 1 \end{pmatrix}$$

Figure 2.1 provides an overview of the specific constants, the element L, the operation n and the approximation order $\sqsubseteq$ in the above models. Infinite executions are ignored in the first model; aborting executions are ignored in the first three models.

In the above models, the computation L with all infinite executions is not only the $\sqsubseteq$-least element, but also a left annihilator of sequential composition. This property characterises strict computations. Intuitively, it states that no computations are performed after an endless loop.

model	first	second	third	fourth	fifth
no executions	0	$\begin{pmatrix} \top & \top \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \top & 0 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & 0 & 0 \end{pmatrix}$
no state change	1	$\begin{pmatrix} \top & \top \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \top & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & 0 & 1 \end{pmatrix}$
all finite executions	$\top$	$\begin{pmatrix} \top & \top \\ 0 & \top \end{pmatrix}$	$\begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}$	$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & 0 & \top \end{pmatrix}$	$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & 0 & \top \end{pmatrix}$
all infinite executions L	0	$\begin{pmatrix} \top & \top \\ \top & \top \end{pmatrix}$	$\begin{pmatrix} \top & 0 \\ \top & 0 \end{pmatrix}$	$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & \top & 0 \end{pmatrix}$	$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & \top & 0 \end{pmatrix}$
all infinite and finite executions	$\top$	$\begin{pmatrix} \top & \top \\ \top & \top \end{pmatrix}$	$\begin{pmatrix} \top & 0 \\ \top & \top \end{pmatrix}$	$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & \top & \top \end{pmatrix}$	$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & \top & \top \end{pmatrix}$
all aborting executions	0	$\begin{pmatrix} \top & \top \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \top & 0 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ \top & \top & \top \end{pmatrix}$	$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ \top & 0 & 0 \end{pmatrix}$
all aborting and finite executions	$\top$	$\begin{pmatrix} \top & \top \\ 0 & \top \end{pmatrix}$	$\begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}$	$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ \top & \top & \top \end{pmatrix}$	$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ \top & 0 & \top \end{pmatrix}$
all aborting and infinite executions	0	$\begin{pmatrix} \top & \top \\ \top & \top \end{pmatrix}$	$\begin{pmatrix} \top & 0 \\ \top & 0 \end{pmatrix}$	$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ \top & \top & \top \end{pmatrix}$	$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ \top & \top & 0 \end{pmatrix}$
all aborting, infinite and finite executions	$\top$	$\begin{pmatrix} \top & \top \\ \top & \top \end{pmatrix}$	$\begin{pmatrix} \top & 0 \\ \top & \top \end{pmatrix}$	$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ \top & \top & \top \end{pmatrix}$	$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ \top & \top & \top \end{pmatrix}$
x_i	R_i	$\begin{pmatrix} \top & \top \\ Q_i & R_i \end{pmatrix}$	$\begin{pmatrix} \top & 0 \\ Q_i & R_i \end{pmatrix}$	$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ P_i & Q_i & R_i \end{pmatrix}$	$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ P_i & Q_i & R_i \end{pmatrix}$
$n(x_i)$	0	$\begin{pmatrix} \top & \top \\ 0 & Q_i \cap 1 \end{pmatrix}$	$\begin{pmatrix} \top & 0 \\ 0 & Q_i \cap 1 \end{pmatrix}$	$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & 0 & Q_i \cap 1 \end{pmatrix}$	$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & 0 & Q_i \cap 1 \end{pmatrix}$
$x_1 \sqsubseteq x_2$	$R_1 \subseteq R_2$	$Q_2 \subseteq Q_1 \wedge$ $R_2 \subseteq R_1$	$Q_2 \subseteq Q_1 \wedge$ $R_1 \subseteq R_2 \wedge$ $R_2 \subseteq R_1 \cup Q_1$	$Q_2 \subseteq Q_1 \wedge$ $P_1 \subseteq P_2 \wedge$ $R_1 \subseteq R_2 \wedge$ $R_2 \subseteq R_1 \cup Q_1$	$Q_2 \subseteq Q_1 \wedge$ $P_1 \subseteq P_2 \wedge$ $P_2 \subseteq P_1 \cup Q_1 \wedge$ $R_1 \subseteq R_2 \wedge$ $R_2 \subseteq R_1 \cup Q_1$
restrictions for x_i	$-$	$Q_i \subseteq R_i$	$-$	$P_i \subseteq Q_i \wedge P_i \subseteq R_i$	$-$

Figure 1: Specific computations, the element L, the operation n and the approximation order $\sqsubseteq$ in the models of Section 2.1

2.2. Models of non-strict computations

For non-strict computations, L is not a left annihilator of sequential composition. Intuitively, such computations can recover from undefined inputs. For example, if a program's state is given by the value of a single integer variable x, the sequential composition of L with the constant assignment $x := 2$ is just $x := 2$. Such a computation model is described in [15]. We first present a simplified version for computations having a single variable with an elementary type. This shows the connection to the matrix-based models of strict computations.

In the sixth model, a computation is a 3×3 matrix whose entries are relations over A. The matrix has the form

$$\begin{pmatrix} U & V & W \\ X & Y & Z \\ P & Q & R \end{pmatrix}$$

where P and Q are vectors, W and Z are relational converses of vectors and $U, V, X, Y \in \{0, \top\}$. The converse of a vector is a relation in which each state is related to the same set of states. The first, second and third rows indicate the transitions if the preceding execution is aborting, infinite or finite, respectively. In each of these cases, the first, second and third columns indicate the aborting, infinite or finite executions of the present computation, respectively.

Intuitively, the entries U, V and W describe the possible executions when the computation is started in an 'aborting' state, that is, when the preceding execution aborts due to an error. If $U = \top$ and $V = W = 0$, the present computation aborts, too, which propagates the error as typical for strict computations. A non-strict computation, however, might have $U = V = 0$ and W containing particular finite executions to the effect that the computation recovers from abortion. A similar remark holds for the entries X, Y and Z with respect to an 'infinite' state, that is, when the preceding execution does not terminate. A strict computation will propagate this using $X = Z = 0$ and $Y = \top$. A non-strict computation might recover with $X = Y = 0$ and particular finite executions in Z.

Non-deterministic choice, conjunction, sequential composition, refinement and approximation are as in the second model. In particular, the approximation order $\sqsubseteq$ is the componentwise superset order. Hence recursions are solved by greatest fixpoints in the componentwise subset order. For the following examples of computations in this model, we assume that the state space $A = \mathbb{N}$ is given by the variable $x \in \mathbb{N}$.

* The (strict) computation that does not change the state is

$$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

* Let $R_1 = \{(x, x') \mid x' = x + 1\}$. The assignment $x := x + 1$ is the (strict) computation

$$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & 0 & R_1 \end{pmatrix}$$

* Let $R_2 = \{(x, x') \mid x' = 2\}$. The strict version of the assignment $x := 2$ is

$$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & 0 & R_2 \end{pmatrix}$$

* A constant assignment, such as the previous one, can be executed lazily without executing preceding computations which may abort or may not terminate. Accordingly, the non-strict version of the assignment $x := 2$ is

$$\begin{pmatrix} 0 & 0 & R_2 \\ 0 & 0 & R_2 \\ 0 & 0 & R_2 \end{pmatrix}$$

* The following computation $\preccurlyeq$ effects a closure; it will be used below:

$$\begin{pmatrix} \top & 0 & 0 \\ \top & \top & \top \\ 0 & 0 & 1 \end{pmatrix}$$

* The computation with all aborting executions is

$$\begin{pmatrix} \top & 0 & 0 \\ \top & 0 & 0 \\ \top & 0 & 0 \end{pmatrix}$$

* The computation with all infinite executions is

$$\begin{pmatrix} 0 & \top & 0 \\ 0 & \top & 0 \\ 0 & \top & 0 \end{pmatrix}$$

* The computation with all executions is the $\sqsubseteq$-least element L, namely

$$\begin{pmatrix} \top & \top & \top \\ \top & \top & \top \\ \top & \top & \top \end{pmatrix}$$

The operation n in this model and in the following models of non-strict computations will be derived in Section 4.4. The computation with all infinite executions might be an alternative for L with a different approximation order. This will not be explored in the present paper; we just note that the matrices with $U = Y = \top$ and $V = W = X = Z = 0$ are the strict computations of the fifth model, which use a variant of the Egli-Milner order.

Using the componentwise superset order for approximation requires a different interpretation of infinite executions. Namely, the semantics of the endless loop $\mathsf{while\ true\ do\ skip}$ is the least fixpoint of the identity function in this order, which is L. But the endless loop has neither finite nor aborting executions, which suggests that these should be ignored in the presence of infinite executions. This is similar to the total-correctness model and technically achieved by requiring the matrices to be closed with respect to sequential composition with the matrix $\preccurlyeq$ as described below. In the following model we eliminate the matrices that are not closed in this way.

The seventh model uses a subset of the matrices of the sixth model with the same operations. The matrices still have the form

$$\begin{pmatrix} U & V & W \\ X & Y & Z \\ P & Q & R \end{pmatrix}$$

where P and Q are vectors, W and Z are relational converses of vectors and $U, V, X, Y \in \{0, \top\}$. Additionally, we require the following conditions:

$$\begin{array}{llll} U \subseteq X & P \subseteq X & V \subseteq U & V \subseteq W \\ V \subseteq Y & Q \subseteq Y & Y \subseteq X & Y \subseteq Z \\ W \subseteq Z & R \subseteq Z & Q \subseteq P & Q \subseteq R \end{array}$$

The six conditions on the right specify that both the first row and the third row of a matrix are componentwise below its second row. The six conditions on the left specify that the second column is componentwise below both the first column and the third column. A matrix that does not satisfy these conditions can be turned into one that does by sequentially composing the matrix $\preccurlyeq$ on both sides; this is a closure operation, that is, isotone, idempotent and increasing. The closed versions of the above examples are:

* The (strict) computation that does not change the state is $\preccurlyeq$, that is,

$$\begin{pmatrix} \top & 0 & 0 \\ \top & \top & \top \\ 0 & 0 & 1 \end{pmatrix}$$

* The assignment $x := x + 1$ is the (strict) computation

$$\begin{pmatrix} \top & 0 & 0 \\ \top & \top & \top \\ 0 & 0 & R_1 \end{pmatrix}$$

* The strict version of the assignment $x := 2$ is

$$\begin{pmatrix} \top & 0 & 0 \\ \top & \top & \top \\ 0 & 0 & R_2 \end{pmatrix}$$

* The non-strict version of the assignment $x := 2$ is

$$\begin{pmatrix} 0 & 0 & R_2 \\ 0 & 0 & R_2 \\ 0 & 0 & R_2 \end{pmatrix}$$

* The computation with all aborting executions is

$$\begin{pmatrix} \top & 0 & 0 \\ \top & 0 & 0 \\ \top & 0 & 0 \end{pmatrix}$$

* The computation with all infinite executions L is the same as the computation with all executions, namely,

$$\begin{pmatrix} \top & \top & \top \\ \top & \top & \top \\ \top & \top & \top \end{pmatrix}$$

We now work towards generalising the two preceding models to computations with several variables and complex data types. Recall that in each 3×3 matrix, P and Q are vectors, W and Z are relational converses of vectors and $U, V, X, Y \in \{0, \top\}$. Because of these restrictions, each 3×3 matrix can be represented as a single relation over the extended state space $A \cup \{\infty, \lightning\}$. The special values ∞ and $\lightning$ represent the results of infinite and aborting executions, respectively. The value ∞ as the resulting state signifies that the execution of the program does not terminate. Similarly, the value $\lightning$ means that the execution aborts. Relations over this extended state space are isomorphic to the matrices of the sixth model. The element $\lightning$ corresponds to the first row and column of a 3×3 matrix, while ∞ corresponds to the second row and column. A similar isomorphism exists between representations of non-termination by a special value or an auxiliary variable in the Unifying Theories of Programming [13].

In the present setting, the representation over the extended state space has an advantage: it can be extended to complex data types. The matrix-based representation treats all finite executions equally; special care is taken only for aborting and infinite executions by adding two extra rows and columns. When computing with infinite data structures, however, the necessary sum, product, function and recursive types are sets with a partial order $\preccurlyeq$ which is not flat. For example, the first and the third elements of the list $1{:}\infty{:}3{:}\infty$ are defined, but access to the second element or the tail after the third element results in an infinite execution. It is not obvious how to represent such nested infinite executions by matrices whose elements

are relations over A. However, they can be represented by relations over an extended state space with a non-flat partial order $\preccurlyeq$ as elaborated in [15].

We can thus generalise the above simplified models of non-strict computations to several variables and complex data types. To this end, we assume that an extended state space A' has been constructed and partially ordered by $\preccurlyeq$. In the special case of a single variable with an elementary type, $A' = A \cup \{\infty, \lightning\}$ and the partial order is flat with ∞ as least element. The detailed construction for complex data types plays no role in the present paper.

The eighth model generalises the sixth model; a computation is a relation over A'. Using A' instead of A allows us to represent infinite and aborting executions in contrast to the first model of Section 2.1. Non-deterministic choice, conjunction, sequential composition and refinement are as in the first model. Approximation is given by the superset order, whence recursions are solved by greatest fixpoints in the subset order. The relations 0, 1 and $\top$ play their usual roles as identities, annihilators, least and greatest elements. It follows that $\mathsf{L} = \top$ in this model. The operation n will be derived in Section 4.4. We translate the above examples of computations with a single variable $x \in A' = \mathbb{N} \cup \{\infty, \lightning\}$.

* The (strict) computation that does not change the state is the identity relation 1.
* The assignment $x := x+1$ is the (strict) computation $\{(\lightning, \lightning), (\infty, \infty)\} \cup \{(x, x') \mid x \in \mathbb{N} \wedge x' = x+1\}$.
* The strict version of the assignment $x := 2$ is $\{(\lightning, \lightning), (\infty, \infty)\} \cup \{(x, 2) \mid x \in \mathbb{N}\}$.
* The non-strict version of the assignment $x := 2$ is $\{(x, 2) \mid x \in A'\}$.
* Being a relation over A', the partial order $\preccurlyeq = \{(\infty, x') \mid x' \in A'\} \cup 1$ is also a computation.
* The computation with all aborting executions is $\{(x, \lightning) \mid x \in A'\}$.
* The computation with all infinite executions is $\{(x, \infty) \mid x \in A'\}$.
* The computation with all executions is $\top$.

Regarding the interpretation of infinite executions, the remarks made for the sixth model apply accordingly. Computations which are not closed with respect to sequential composition with $\preccurlyeq$ are eliminated in the following model.

In the ninth model, a computation is a relation R over A' that satisfies $\preccurlyeq \,;\, R = R = R \,;\, \preccurlyeq$. We call such relations $\preccurlyeq$-closed. The operations on $\preccurlyeq$-closed relations remain as in the eighth model. The relation $\preccurlyeq$ replaces 1 as the identity of sequential composition. The $\preccurlyeq$-closed versions of the above examples are:

* The (strict) computation that does not change the state is $\preccurlyeq$.
* The assignment $x := x + 1$ is $\{(\lightning, \lightning)\} \cup \{(\infty, x') \mid x' \in A'\} \cup \{(x, x') \mid x \in \mathbb{N} \wedge x' = x + 1\}$.
* The strict version of the assignment $x := 2$ is $\{(\lightning, \lightning)\} \cup \{(\infty, x') \mid x' \in A'\} \cup \{(x, 2) \mid x \in \mathbb{N}\}$.
* The non-strict version of the assignment $x := 2$ is $\{(x, 2) \mid x \in A'\}$.
* The computation with all aborting executions is $\{(x, \lightning) \mid x \in A'\}$.
* The computation with all infinite executions is $\mathsf{L} = \top$, which is also the computation with all executions.

It is shown in [15] that $\preccurlyeq$-closed relations form a complete lattice and that they are closed under program and specification constructs such as assignments, non-deterministic choice, sequential composition, conjunction, conditional and recursion. Furthermore, they are closed under the Kleene star and omega operations.

3. Basic algebraic structures for sequential computations

In this section we axiomatise the operations of non-deterministic choice, conjunction and sequential composition and various forms of iteration featured by the computation models of Section 2. To this end we use lattices and variants of semirings, Kleene algebras and omega algebras [4, 34, 27, 8].

3.1. Semilattices, lattices and semirings

A *bounded join-semilattice* is an algebraic structure $(S, +, 0)$ satisfying the axioms

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

The *semilattice order* $x \leq y \Leftrightarrow x + y = y$ has least element 0 and least upper bound $+$. The operation $+$ is $\leq$-isotone.

A *bounded distributive lattice* $(S, +, \curlywedge, 0, \top)$ adds to a bounded join-semilattice a dual bounded meet-semilattice $(S, \curlywedge, \top)$ as well as distribution and absorption axioms:

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

The semilattice order has the alternative characterisation $x \leq y \Leftrightarrow x \curlywedge y = x$, greatest element $\top$ and greatest lower bound $\curlywedge$. The operation $\curlywedge$ is $\leq$-isotone.

An idempotent semiring $(S, +, \cdot, 0, 1)$ without right annihilator – simply called *semiring* in the remainder of this paper – adds to a bounded join-semilattice a monoid $(S, \cdot, 1)$ as well as distribution axioms and a left annihilation axiom:

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

In particular, $x \cdot 0 = 0$ is not an axiom. The operation $\cdot$ is $\leq$-isotone. We abbreviate $x \cdot y$ as xy.

All computation models of Section 2 are bounded distributive lattices and semirings. In those models the operation $+$ represents non-deterministic choice, the operation $\curlywedge$ conjunction, the operation $\cdot$ sequential composition, 0 the computation with no executions, 1 the computation that does not change the state, $\top$ the computation with all executions, and $\leq$ the refinement relation.

3.2. Fixpoints, Kleene algebras, omega algebras and iterings

Let S be partially ordered by $\leq$ and let $f : S \to S$ be a function on S. The element $x \in S$ is a *fixpoint* of f if $f(x) = x$. Provided they exist, the $\leq$-least and $\leq$-greatest fixpoints of f are denoted by μf and νf, respectively:

$$\begin{aligned} f(\mu f) &= \mu f & \qquad f(x) = x \Rightarrow \mu f \leq x \\ f(\nu f) &= \nu f & \qquad f(x) = x \Rightarrow \nu f \geq x \end{aligned}$$

The existence of μf and νf may depend on additional properties of f or S from which we abstract. We abbreviate $\mu(\lambda x.f(x))$ by $\mu x.f(x)$ and $\nu(\lambda x.f(x))$ by $\nu x.f(x)$.

Several computation models of Section 2 use $\leq$-least and $\leq$-greatest fixpoints as the semantics of recursion. Some models require different fixpoints, which will be introduced as least fixpoints in a dedicated approximation order in Section 4.2.

We are particularly interested in a special case of recursion: fixpoints of the affine function $\lambda x.yx + z$ describe iteration. They are useful, for example, to define the semantics of while-loops. The following algebras capture various fixpoints of affine functions.

A *Kleene algebra* $(S, +, \cdot, ^*, 0, 1)$ adds to a semiring an operation * with the following unfold and induction axioms [27]:

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

It follows that $y^*z = \mu x.yx + z$ and $zy^* = \mu x.xy + z$. The operation * is $\leq$-isotone.

An *omega algebra* $(S, +, \cdot, ^*, ^\omega, 0, 1)$ adds to a Kleene algebra an operation $^\omega$ with the following unfold and induction axioms [8, 34]:

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

It follows that $y^\omega + y^*z = \nu x.yx + z$ and that $\top = 1^\omega$ is the $\leq$-greatest element. The operation $^\omega$ is $\leq$-isotone.

Yielding $\leq$-least and $\leq$-greatest fixpoints, respectively, Kleene star and omega describe iteration in several computation models of Section 2. For models that require different fixpoints of $\lambda x.yx + z$, we use the following generalisations of Kleene algebras.

An *itering* $(S, +, \cdot, ^\circ, 0, 1)$ adds to a semiring an operation $^\circ$ with the sumstar and productstar equations of [9] and two simulation axioms [19]:

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

It follows that $y^\circ z$ is a fixpoint of $\lambda x.yx + z$ and that zy° is a fixpoint of $\lambda x.xy + z$. The operation $^\circ$ is $\leq$-isotone.

Every Kleene algebra is an itering using $x^\circ = x^*$. Every omega algebra is an itering using $x^\circ = x^\omega 0 + x^*$. Further instances and consequences of iterings are presented in [19]; they cover the models of strict computations of Section 2.1.

A *binary itering* $(S, +, \cdot, \star, 0, 1)$ adds to a semiring a binary operation $\star$ with the following axioms [20]:

$$\begin{array}{ll} (x + y) \star z = (x \star y) \star (x \star z) & zx \leq y(y \star z) + w \Rightarrow z(x \star v) \leq y \star (zv + w(x \star v)) \\ (xy) \star z = z + x((yx) \star (yz)) & xz \leq z(y \star 1) + w \Rightarrow x \star (zv) \leq z(y \star v) + (x \star (w(y \star v))) \\ x \star (y + z) = (x \star y) + (x \star z) & (x \star y)z \leq x \star (yz) \end{array}$$

It follows that $y \star z$ is a fixpoint of $\lambda x.yx + z$. The operation $\star$ is $\leq$-isotone. An *extended binary itering* is a binary itering with the additional axiom

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

The element $y \star z$ corresponds to iterating y an unspecified number of times, followed by a single occurrence of z. This may involve an infinite number of iterations of y.

Every itering is an extended binary itering using $x \star y = x^\circ y$. Every omega algebra is a binary itering using $x \star y = x^\omega + x^*y$. Together, these instances cover all computation models of Section 2; in Section 4.3 we express $\star$ uniformly in all of these models. Further consequences of binary iterings are presented in [20].

The binary operation $\star$ generalises the unary itering operation $^\circ$ by composing a continuation. This is necessary to describe iteration in the models of non-strict computations of Section 2.2 in addition to the models of strict computations of Section 2.1. For the latter, the unary operation is sufficient.

4. Unifying recursion for models of strict and non-strict computations

In this section we axiomatise an operation that captures the infinite executions of a computation in each model of Section 2. Based on this operation we define a common approximation order for those models. We then derive a unified semantics of recursion as least fixpoints in this order. We show that the special case of iteration satisfies the extended binary itering axioms of Section 3.2. The resulting algebras instantiate to all computation models of Section 2.

4.1. Infinite executions

We first axiomatise the constant L, which represents the computation with all infinite executions, and the operation n such that $n(x)$ describes the set of states from which the computation x has infinite executions. Sets of states are represented as tests, which are particular elements ≤ 1. Tests act as filters in sequential compositions: in the sequential composition px of a test p and a computation x, the executions of x are restricted to those whose starting state is in the set described by p.

Tests are frequently used to describe conditions in conditional statements and while-loops [28]. In such a context, they usually form a Boolean algebra. The following axioms focus on the infinite executions; axioms may be added to deal with conditions as in [19], but this is not necessary for the present paper.

An *n-algebra* $(S, +, \curlywedge, \cdot, n, 0, 1, \mathsf{L}, \top)$ expands a semiring $(S, +, \cdot, 0, 1)$ and a bounded distributive lattice $(S, +, \curlywedge, 0, \top)$ by an operation $n : S \to S$ and a constant L satisfying the following axioms:

$$
\begin{array}{lrlrl}
(n1) & n(x)+n(y) & = n(n(x)\top + y) & (n6) & n(x) \leq n(\mathsf{L}) \curlywedge 1 \\
(n2) & n(x)n(y) & = n(n(x)y) & (n7) & n(x)\mathsf{L} \leq x \\
(n3) & n(x)n(x+y) & = n(x) & (n8) & n(\mathsf{L})x \leq x\, n(\mathsf{L}) \\
(n4) & n(\mathsf{L})x & = (x \curlywedge \mathsf{L}) + n(\mathsf{L}0)x & (n9) & x\, n(y)\top \leq x0 + n(xy)\top \\
(n5) & x\mathsf{L} & = x0 + n(x\mathsf{L})\mathsf{L} & (n10) & x\top y \curlywedge \mathsf{L} \leq x\mathsf{L}y
\end{array}
$$

A *test* is an element of the image $n(S) = \{n(x) \mid x \in S\}$. Except for $(n10)$, these axioms follow from the ones given in [19] extended by a meet operation. The following remarks describe the underlying intuition, their use in the subsequent development and the relation to previous works, in particular, to the axioms and properties of n given in [19].

- Axiom $(n1)$ weakens the previous distributivity property $n(x) + n(y) = n(x + y)$, which no longer holds. This weakening is necessary to capture the sixth and the eighth model of Section 2.2; in the remaining models n distributes over $+$. Axiom $(n1)$ also implies that tests are closed under $+$.

- Axiom $(n2)$ states that the infinite executions of a computation y restricted to starting states in $n(x)$ are obtained by restricting the infinite executions of y with $n(x)$. In all models of Section 2, the operation $\cdot$ on tests is the intersection of the represented sets; whether $n(x)n(y) = n(x) \curlywedge n(y)$ follows from the axioms is unknown. Axiom $(n2)$ also implies that tests are closed under $\cdot$.

- Axiom $(n3)$ implies that composition of tests is idempotent by setting $y = 0$. The property $n(x) = n(x0)n(x)$, which was previously used to this effect, and its consequence $n(x) = n(x0)$ do not hold in models of non-strict computations. Another consequence of $(n3)$ is that n is $\leq$-isotone; this must be axiomatised because distributivity over $+$ no longer holds.

- Axiom $(n4)$ is needed to establish $\mathsf{L}x \leq \mathsf{L}$, which weakens the previous axiom $\mathsf{L}x = \mathsf{L}$ that does not hold for non-strict computations. A related axiom with the domain operation instead of n is used in [20] for a different purpose.

- Axiom $(n5)$ expresses that a computation followed by an infinite execution comprises infinite executions – contained in $n(x\mathsf{L})\mathsf{L}$ – and aborting executions – contained in $x0$. Executions are split this way, for example, when showing that $\cdot$ is isotone with respect to the approximation order introduced below.

- Axiom $(n6)$ states that no computation can have more infinite executions than L and that every test is ≤ 1 since $\curlywedge$ is the $\leq$-greatest lower bound. Previous properties $n(\top) = n(\mathsf{L}) = 1$ do not hold in the first model of Section 2.1. While $n(0) = 0$ holds in all our models, it does not follow from the current axioms, although $n(0)$ is the $\leq$-least test since n is $\leq$-isotone. The set of tests may therefore be situated strictly between 0 and 1. This demonstrates that n is not primarily intended to induce a set of conditions, which would include 0 and 1 for false and true, respectively.

- Axiom $(n7)$ is a component of the Galois connection $n(x)\mathsf{L} \leq y \Leftrightarrow n(x) \leq n(y)$ between $n(S)$ and S with lower adjoint $\lambda p.p\mathsf{L}$ and upper adjoint n. Hence $n(y)$ is the $\leq$-greatest test by which L can be restricted so that the result is below y. This is the characterising property of the infinite executions of y observed in [19].

* Axiom ($n8$) is used, for example, to show that $\cdot$ is isotone with respect to the approximation order. All computation models of Section 2 satisfy $n(\mathsf{L}) = 1$, except the first model, which satisfies $n(\mathsf{L}) = 0$. Axiom ($n8$) is the technical means to extend our treatment to this particular model; see also the occurrence of $n(\mathsf{L})$ in the approximation order below. A similar axiom with the domain operation instead of n is used in [20] to this end.

* Axiom ($n9$) is a splitting property similar to ($n5$) and used for similar purposes. The difference to ($n5$) is that x is not followed by L, but by $\top$ possibly restricted with $n(y)$. Related splitting axioms are used in [19].

* Axiom ($n10$) generalises the previous property $x\top \curlywedge \mathsf{L} \leq x\mathsf{L}$, which states that the infinite executions of $x\top$ are already contained in $x\mathsf{L}$. It is used, for example, to show antisymmetry of the approximation order.

Counterexamples generated by Mace4 [32] and Nitpick [6] witness that each of the axioms ($n1$), ($n3$)–($n10$) is independent of the others and the underlying semiring and lattice axioms. For ($n2$) this is unknown. Further consequences of n-algebras are recorded in the following result.

Theorem 1. *Let S be an n-algebra and $x, y \in S$. Then $(n(S), +, \cdot, n(0), n(\top))$ is a semiring with right annihilator $n(0)$ and a bounded distributive lattice with meet $\cdot$. Moreover, n is $\leq$-isotone and the following properties hold:*

1. $n(x)n(y) = n(y)n(x)$
2. $n(x)n(x) = n(x)$
3. $n(x)n(y) \leq n(x)$
4. $n(x)n(y) \leq n(y)$
5. $n(x) \leq n(x + y)$
6. $n(x) \leq 1$
7. $n(x)0 = 0$
8. $n(x)n(0) = n(0)$
9. $n(x) \leq x + n(x0)$
10. $n(x + n(x)\top) = n(x)$
11. $n(n(x)\mathsf{L}) = n(x)$
12. $n(x)n(\mathsf{L}) = n(x)$
13. $n(x) \leq n(\mathsf{L})$
14. $n(x) \leq n(x\mathsf{L})$
15. $n(x)\mathsf{L} \leq x\mathsf{L}$
16. $n(0)\mathsf{L} = 0$
17. $n(\mathsf{L}) = n(\top)$
18. $n(x\top) = n(x\mathsf{L})$
19. $n(x)\top = n(x)\mathsf{L} + n(x0)\top$
20. $n(x\, n(y)\mathsf{L}) \leq n(xy)$
21. $x\, n(y)\top \leq xy + n(xy)\top$
22. $n(x)\top y \leq xy + n(xy)\top$
23. $x\, n(y)\mathsf{L} = x0 + n(x\, n(y)\mathsf{L})\mathsf{L}$
24. $x\, n(y)\mathsf{L} \leq x0 + n(xy)\mathsf{L}$
25. $n(\mathsf{L})x \leq x0 + n(x\mathsf{L})\top$
26. $n(\mathsf{L})\mathsf{L} = \mathsf{L}n(\mathsf{L}) = \mathsf{L}$
27. $\mathsf{L}\mathsf{L} = \mathsf{L}\top = \mathsf{L}\top\mathsf{L} = \mathsf{L}$
28. $\mathsf{L}x \leq \mathsf{L}$
29. $x\mathsf{L} \leq x0 + \mathsf{L}$
30. $x\top \curlywedge \mathsf{L} \leq x\mathsf{L}$
31. $x\top y \curlywedge \mathsf{L} = x\mathsf{L}y \curlywedge \mathsf{L}$
32. $x\top y \curlywedge \mathsf{L} \leq x0 + \mathsf{L}y$
33. $(x \curlywedge \mathsf{L})0 \leq x0 \curlywedge \mathsf{L}$
34. $n(x) = n(x \curlywedge \mathsf{L}) = (n(x) \curlywedge \mathsf{L}) + n(x0)$
35. $n(x)\mathsf{L} \leq x \curlywedge \mathsf{L} \leq n(\mathsf{L})x$
36. $n(x) \curlywedge \mathsf{L} \leq (n(x) \curlywedge \mathsf{L})\top \leq n(x)\mathsf{L} \leq x$
37. $x \leq y \Leftrightarrow x \leq y + \mathsf{L} \wedge n(\mathsf{L})x \leq y + n(y)\top$
38. $x \leq y \Leftrightarrow x \leq y + \mathsf{L} \wedge x \leq y + n(y)\top$
39. $n(y)x \leq x\, n(y) \Leftrightarrow n(y)x = n(y)x\, n(y)$
40. $n(x) \leq n(y) \Leftrightarrow n(x)\mathsf{L} \leq y$

Counterexamples generated by Nitpick witness that none of the following properties follow from the axioms of n-algebras:

* $n(0) = 0$
* $n(1) = 0$
* $n(\mathsf{L}) = 1$
* $n(\top) = 1$
* $n(x) = n(x0)$
* $n(x) + n(y) = n(x + y)$
* $x\, n(y)\mathsf{L} = x0 + n(xy)\mathsf{L}$
* $x \leq x0 + n(x\mathsf{L})\top$
* $n(xy) \leq n(x\, n(y)\top)$
* $n(\mathsf{L})x \leq n(x\top)\top$
* $x \curlywedge n(y)\top \leq n(y)x$
* $x \curlywedge n(y)\top \leq n(\mathsf{L})x$
* $n(\mathsf{L})x\top \leq n(x\top \curlywedge \mathsf{L})\top$
* $n(x\top \curlywedge \mathsf{L})\top \leq n(\mathsf{L})x\top$
* $x0 \curlywedge \mathsf{L} \leq n(x\mathsf{L})\mathsf{L}$
* $n(x\mathsf{L})\mathsf{L} \leq n(x)\mathsf{L}$
* $n(x)\mathsf{L} \leq (x \curlywedge \mathsf{L})0$
* $(x \curlywedge \mathsf{L})0 \leq n(x)\mathsf{L}$

4.2. Approximation and recursion

All computation models of Section 2 define the semantics of a recursive specification $x = f(x)$ as the least fixpoint of the function f in a particular approximation order. The approximation order varies among the models. For a unified treatment in n-algebras we use the following approximation order:

$$x \sqsubseteq y \;\Leftrightarrow\; x \leq y + \mathsf{L} \wedge n(\mathsf{L})y \leq x + n(x)\top$$

It combines the orders of [17, 19, 20] so as to capture all models of Section 2. The intuition underlying this definition is as follows. If $n(\mathsf{L}) = 0$, then $\mathsf{L} = n(\mathsf{L})\mathsf{L} = 0$ by Theorem 1.26 and $x \sqsubseteq y$ reduces to $x \leq y$. This captures the first model of Section 2.1; in the other models $n(\mathsf{L}) = 1$, whence $x \sqsubseteq y$ reduces to $x \leq y + \mathsf{L} \wedge y \leq x + n(x)\top$. The part $x \leq y + \mathsf{L}$ states that executions may be added and infinite executions may be removed when improving the approximation from x to y. The part $y \leq x + n(x)\top$ expresses that in states where x has no infinite executions, no executions may be added when going from x to y, whence y must have the same executions as x.

Because of the new axiomatisation, the following results require new proofs.

Theorem 2. *Let S be an n-algebra.*

1. *The relation $\sqsubseteq$ is a partial order with least element L.*
2. *The operations $+$ and $\cdot$ and $\lambda x.x \curlywedge \mathsf{L}$ and $\lambda x.n(x)\mathsf{L}$ are $\sqsubseteq$-isotone.*
3. *If S is an itering, the operation $^\circ$ is $\sqsubseteq$-isotone.*
4. *If S is a Kleene algebra, the operation * is $\sqsubseteq$-isotone.*

Provided it exists, the $\sqsubseteq$-greatest lower bound of $x, y \in S$ in an n-algebra S is denoted by $x \sqcap y$:

$$x \sqcap y \sqsubseteq x \qquad x \sqcap y \sqsubseteq y \qquad z \sqsubseteq x \wedge z \sqsubseteq y \Rightarrow z \sqsubseteq x \sqcap y$$

The existence of $\sqcap$ may depend on additional properties of S from which we abstract. Provided it exists, the $\sqsubseteq$-least fixpoint of a function $f : S \to S$ is denoted by κf:

$$f(\kappa f) = \kappa f \qquad f(x) = x \Rightarrow \kappa f \sqsubseteq x$$

We abbreviate $\kappa(\lambda x.f(x))$ by $\kappa x.f(x)$. The following characterisations of κf are generalised from [19, 20] to the present setting of n-algebras, which covers all models of Section 2.

Theorem 3. *Let S be an n-algebra, 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. *$n(\mathsf{L})\nu f \leq (\nu f \curlywedge \mathsf{L}) + \mu f + n(\nu f)\top$.*
5. *$n(\mathsf{L})\nu f \leq (\nu f \curlywedge \mathsf{L}) + \mu f + n((\nu f \curlywedge \mathsf{L}) + \mu f)\top$.*
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 $\mu f \sqcap \nu f \leq \nu f$.*

Condition 4 of this theorem characterises the existence of κf in terms of μf and νf. Condition 3 shows how to obtain κf from μf and νf. This simplifies calculations as $\leq$ is less complex than $\sqsubseteq$. Further characterisations can be generalised to n-algebras as shown in the following result.

Theorem 4. *Let S be an n-algebra, 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 3:*

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

A counterexample generated by Nitpick witnesses that condition 2 of this theorem is strictly stronger than condition 4 of Theorem 3.

4.3. Iteration

Iteration is dealt with by instantiating the above results for the affine function $f(x) = yx + z$. Note that Theorems 3 and 4 reduce κf to μf and νf, which can be expressed using the Kleene star and omega operations. We therefore introduce the following structure.

An *n-omega algebra* $(S, +, \curlywedge, \cdot, n, {}^*, {}^\omega, 0, 1, \mathsf{L}, \top)$ is an n-algebra $(S, +, \curlywedge, \cdot, n, 0, 1, \mathsf{L}, \top)$ and an omega algebra $(S, +, \cdot, {}^*, {}^\omega, 0, 1)$ that satisfies the following axioms:

$$(n11) \quad n(\mathsf{L})x^\omega \leq x^* n(x^\omega)\top \qquad\qquad (n12) \quad x\mathsf{L} \leq x\mathsf{L}x\mathsf{L}$$

The underlying intuition is as follows.

- Axiom $(n11)$ is taken from [19] and captures the interaction of * and $^\omega$ with n. It states that whenever x can be infinitely iterated, x can be iterated a finite number of times so that afterwards x^ω has an infinite execution.
- Axiom $(n12)$ modifies the property $x\top \leq x\top x\top$ of [20]. The latter has been used as an alternative to adding the 'Tarski rule' of relation algebra, according to which $x = 0$ or $\top x \top = \top$ holds for every x [40]. A referee has noted that $x\top \leq x\top x\top$ already holds in relation algebras without the Tarski rule; a counterexample generated by Nitpick witnesses that it does not hold in n-omega algebras. The strict computations of Section 2.1 satisfy $\mathsf{L}x = \mathsf{L}$ and the models of non-strict computations of Section 2.2 are relations with $\mathsf{L} = \top$, whence $(n12)$ holds for both kinds.

Consequences of n-omega algebras are recorded in the following result.

Theorem 5. *Let S be an n-omega algebra and $x, y, z \in S$. Then the following properties hold:*

1. $\mathsf{L}x^* = \mathsf{L}$
2. $(x\mathsf{L})^* = 1 + x\mathsf{L}$
3. $(x\mathsf{L})^\omega = x\mathsf{L} = x\mathsf{L}x\mathsf{L}$
4. $(x\mathsf{L})^* y \leq y + x\mathsf{L}$
5. $(x\mathsf{L} + y)^* = y^* + y^* x\mathsf{L}$
6. $(x\mathsf{L} + y)^\omega = y^\omega + y^* x\mathsf{L}$
7. $n(x) \leq n(x^\omega)$
8. $n(y^\omega + y^* z) = n(y^\omega) + n(y^* z)$
9. $x^* + n(x^\omega)\mathsf{L} = x^* + x^* n(x^\omega)\mathsf{L}$
10. $x^* + n(x^\omega)\mathsf{L} = x^* + x\, n(x^\omega)\mathsf{L}$
11. $yx^* + n(yx^\omega)\mathsf{L} = yx^* + y\, n(x^\omega)\mathsf{L}$
12. $x^*0 + n(x^\omega)\mathsf{L} = x^*0 + x^* n(x^\omega)\mathsf{L}$
13. $xx^*0 + n(x^\omega)\mathsf{L} = xx^*0 + x\, n(x^\omega)\mathsf{L}$
14. $yx^*0 + n(yx^\omega)\mathsf{L} = yx^*0 + y\, n(x^\omega)\mathsf{L}$
15. $n(\mathsf{L})x^\omega \leq x^*0 + n(x^\omega)\top$
16. $n(\mathsf{L})(y^\omega + y^* z) \leq y^* z + n(y^\omega + y^* z)\top$

The following result instantiates Theorems 3 and 4 to obtain the $\sqsubseteq$-least fixpoint of an affine function. It also shows that every n-omega algebra forms an extended binary itering.

Theorem 6. *Let S be an n-omega algebra, let $x, y, z \in S$ and let $f(x) = yx + z$.*

1. *The $\sqsubseteq$-least fixpoint of f is $\kappa f = (y^\omega \curlywedge \mathsf{L}) + y^* z = n(y^\omega)\mathsf{L} + y^* z$.*
2. *The operations $^\omega$ and $\lambda y.(\kappa x.yx + z)$ and $\lambda z.(\kappa x.yx + z)$ are $\sqsubseteq$-isotone.*
3. *S is an extended binary itering using $x \star y = n(x^\omega)\mathsf{L} + x^* y$.*

According to the last statement, all properties of extended binary iterings shown in [20] hold in all computation models of Section 2. This includes various simulation and separation laws generalised from omega algebras and Back's atomicity refinement theorem [1, 42]. These results have been applied for program development and were originally proved for computation models that do not distinguish aborting executions. Because they hold in all computation models of Section 2 we obtain additional guarantees, for example, about the absence of aborting executions.

4.4. Instance for non-strict computations

We conclude by showing how to instantiate n-algebras for the models of non-strict computations given in Section 2.2. Because these models are relational, we work in relation algebras [41, 31].

A *Boolean algebra* $(S, +, \curlywedge, \overline{\ }, 0, \top)$ adds to a bounded distributive lattice a complement operation $\overline{\ }$ with the following axioms:

$$x \curlywedge \overline{x} = 0 \qquad\qquad x + \overline{x} = \top$$

A *relation algebra* $(S, +, \curlywedge, \cdot, \overline{\ }, \breve{\ }, 0, 1, \top)$ adds to a Boolean algebra a composition operation $\cdot$, a converse operation $\breve{\ }$ and a constant 1 with the following axioms:

$$\begin{array}{cc} x \cdot (y \cdot z) = (x \cdot y) \cdot z & x\breve{\ }\breve{\ } = x \\ (x + y) \cdot z = (x \cdot z) + (y \cdot z) & (x + y)\breve{\ } = x\breve{\ } + y\breve{\ } \\ x \cdot 1 = x & (xy)\breve{\ } = y\breve{\ }x\breve{\ } \\ & x\breve{\ }\overline{xy} \leq \overline{y} \end{array}$$

It follows that the reduct $(S, +, \cdot, 0, 1)$ of every relation algebra is a semiring with right annihilator 0. To turn a relation algebra into an n-algebra, it therefore remains to define the operation n and the constant L. This is shown in the following result.

Theorem 7. *Let S be a relation algebra. Then S is an n-algebra using $n(x) = \overline{\overline{x}\top} \curlywedge 1$ and $\mathsf{L} = \top$.*

The intuition underlying this definition of n is that our non-strict computations have an infinite execution from a state if and only if that state is related to all states.

To turn a relation algebra into an n-omega algebra, the Kleene star and omega operations have to be added, too. This can be done by including the axioms of Kleene algebra and omega algebra or assuming completeness of the underlying lattice, as in the following result, to ensure the involved fixpoints exist.

Theorem 8. *Let S be a relation algebra with a complete lattice. Then S is an n-omega algebra using $x^* = \mu y.xy + 1$ and $x^\omega = \nu y.xy$ and $n(x) = \overline{\overline{x}\top} \curlywedge 1$ and $\mathsf{L} = \top$.*

As a consequence the eighth model of Section 2.2 is an n-omega algebra. Moreover it can be shown that $n(R)$ is $\preccurlyeq$-closed if R is $\preccurlyeq$-closed, whence the ninth model is another instance. Because these models generalise the sixth and the seventh models, they are instances, too.

5. Conclusion

We have presented a unified description of infinite executions, approximation, recursion and iteration that covers several relational and matrix-based models of strict and non-strict computations. The models include strict computations that distinguish finite, infinite and aborting executions and non-strict computations that can be executed lazily. This unifies the previous approaches of [19, 20]. Because iteration satisfies the axioms of extended binary iterings, consequences such as Back's atomicity refinement theorem [1, 42] hold in the unified setting.

The unification is based on an operation that captures the set of states from which infinite executions exist. While the sets are represented by test elements, they do not need to be conflated with conditions as used, for example, for conditionals and while-loops.

An interesting observation is that a simplified form of non-strict computations can be represented by matrices. This opens up the possibility of further models inspired by the matrices that are used for strict computations. Vice versa, it is open what the more general, relational representation of non-strict computations implies for the strict case. A referee has suggested to represent the non-strict computations of Section 2.2 by matrices containing relations of different types; this is similar to matrices over typed Kleene and omega algebras [29, 18].

The development in this paper has again benefited from the structuring mechanisms provided by Isabelle and the automation support provided by the integrated automated theorem provers, SMT solvers and counterexample generators.

Acknowledgements

I thank the anonymous referees for their valuable comments and suggestions.

References

[1] R. J. R. Back and J. von Wright. Reasoning algebraically about loops. *Acta Informatica*, 36(4):295–334, 1999.
[2] J. W. de Bakker. Semantics and termination of nondeterministic recursive programs. In S. Michaelson and R. Milner, editors, *Automata, Languages and Programming: Third International Colloquium*, pages 435–477. Edinburgh University Press, 1976.
[3] R. Berghammer and H. Zierer. Relational algebraic semantics of deterministic and nondeterministic programs. *Theoretical Computer Science*, 43:123–147, 1986.
[4] G. Birkhoff. *Lattice Theory*, volume XXV of *Colloquium Publications*. American Mathematical Society, third edition, 1967.
[5] J. C. Blanchette, S. Böhme, and L. C. Paulson. Extending Sledgehammer with SMT solvers. In N. Bjørner and V. Sofronie-Stokkermans, editors, *Automated Deduction: CADE-23*, volume 6803 of *Lecture Notes in Computer Science*, pages 116–130. Springer, 2011.
[6] J. C. Blanchette and T. Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In M. Kaufmann and L. C. Paulson, editors, *Interactive Theorem Proving*, volume 6172 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2010.
[7] J.-L. De Carufel and J. Desharnais. Demonic algebra with domain. In R. Schmidt, editor, *Relations and Kleene Algebra in Computer Science*, volume 4136 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 2006.
[8] E. Cohen. Separation and reduction. In R. Backhouse and J. N. Oliveira, editors, *Mathematics of Program Construction*, volume 1837 of *Lecture Notes in Computer Science*, pages 45–59. Springer, 2000.
[9] J. H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.
[10] J. Desharnais, B. Möller, and G. Struth. Kleene algebra with domain. *ACM Transactions on Computational Logic*, 7(4):798–833, 2006.
[11] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.
[12] S. Dunne. Recasting Hoare and He's Unifying Theory of Programs in the context of general correctness. In A. Butterfield, G. Strong, and C. Pahl, editors, *5th Irish Workshop on Formal Methods*, Electronic Workshops in Computing. The British Computer Society, 2001.
[13] W. Guttmann. Non-termination in Unifying Theories of Programming. In W. MacCaull, M. Winter, and I. Düntsch, editors, *Relational Methods in Computer Science 2005*, volume 3929 of *Lecture Notes in Computer Science*, pages 108–120. Springer, 2006.
[14] W. Guttmann. General correctness algebra. In R. Berghammer, A. M. Jaoua, and B. Möller, editors, *Relations and Kleene Algebra in Computer Science*, volume 5827 of *Lecture Notes in Computer Science*, pages 150–165. Springer, 2009.
[15] W. Guttmann. Imperative abstractions for functional actions. *Journal of Logic and Algebraic Programming*, 79(8):768–793, 2010.
[16] W. Guttmann. Partial, total and general correctness. In C. Bolduc, J. Desharnais, and B. Ktari, editors, *Mathematics of Program Construction*, volume 6120 of *Lecture Notes in Computer Science*, pages 157–177. Springer, 2010.
[17] W. Guttmann. Unifying recursion in partial, total and general correctness. In S. Qin, editor, *Unifying Theories of Programming, Third International Symposium, UTP 2010*, volume 6445 of *Lecture Notes in Computer Science*, pages 207–225. Springer, 2010.
[18] W. Guttmann. Towards a typed omega algebra. In H. de Swart, editor, *Relational and Algebraic Methods in Computer Science*, volume 6663 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 2011.
[19] W. Guttmann. Algebras for iteration and infinite computations. *Acta Informatica*, 49(5):343–359, 2012.
[20] W. Guttmann. Unifying lazy and strict computations. In W. Kahl and T. G. Griffin, editors, *Relational and Algebraic Methods in Computer Science*, volume 7560 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2012.
[21] W. Guttmann. Extended designs algebraically. *Science of Computer Programming*, 78(11):2064–2085, 2013.
[22] W. Guttmann and B. Möller. Normal design algebra. *Journal of Logic and Algebraic Programming*, 79(2):144–173, 2010.
[23] I. J. Hayes, S. E. Dunne, and L. Meinicke. Unifying theories of programming that distinguish nontermination and abort. In C. Bolduc, J. Desharnais, and B. Ktari, editors, *Mathematics of Program Construction*, volume 6120 of *Lecture Notes in Computer Science*, pages 178–194. Springer, 2010.
[24] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580/583, 1969.
[25] C. A. R. Hoare and J. He. *Unifying theories of programming*. Prentice Hall Europe, 1998.
[26] D. Jacobs and D. Gries. General correctness: A unification of partial and total correctness. *Acta Informatica*, 22(1):67–83, 1985.
[27] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.
[28] D. Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, 1997.
[29] D. Kozen. Typed Kleene algebra. Technical Report TR98-1669, Cornell University, 1998.
[30] D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Transactions on Computational Logic*, 1(1):60–76, 2000.
[31] R. D. Maddux. Relation-algebraic semantics. *Theoretical Computer Science*, 160(1–2):1–85, 1996.

[32] W. McCune. Mace4 reference manual and guide. Technical Memorandum ANL/MCS-TM-264, Mathematics and Computer Science Division, Argonne National Laboratory, 2003. Mace4 is available at http://www.cs.unm.edu/~mccune/mace4/.
[33] B. Möller. The linear algebra of UTP. In T. Uustalu, editor, *Mathematics of Program Construction*, volume 4014 of *Lecture Notes in Computer Science*, pages 338–358. Springer, 2006.
[34] B. Möller. Kleene getting lazy. *Science of Computer Programming*, 65(2):195–214, 2007.
[35] B. Möller and G. Struth. Algebras of modal operators and partial correctness. *Theoretical Computer Science*, 351(2):221–239, 2006.
[36] B. Möller and G. Struth. WP is WLP. In W. MacCaull, M. Winter, and I. Düntsch, editors, *Relational Methods in Computer Science 2005*, volume 3929 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2006.
[37] G. Nelson. A generalization of Dijkstra's calculus. *ACM Transactions on Programming Languages and Systems*, 11(4):517–561, 1989.
[38] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.
[39] L. C. Paulson and J. C. Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In G. Sutcliffe, E. Ternovska, and S. Schulz, editors, *Proceedings of the 8th International Workshop on the Implementation of Logics*, pages 3–13, 2010.
[40] G. Schmidt and T. Ströhlein. *Relationen und Graphen*. Springer, 1989.
[41] A. Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73–89, 1941.
[42] J. von Wright. Towards a refinement algebra. *Science of Computer Programming*, 51(1–2):23–45, 2004.