Algebras for Iteration, Infinite Executions and Correctness of Sequential Computations

Habilitationsschrift

Fachgebiet Informatik
Fakultät für Ingenieurwissenschaften und Informatik
Universität Ulm

Dr. rer. nat. Walter Guttmann

2015

Abstract

We study models of state-based non-deterministic sequential computations. They differ in the kinds of computation they can represent and the precision they achieve. We systematically explore existing models, propose new models and investigate their connections.

We propose algebras that describe iteration for strict and non-strict computations. They enable a unified treatment of various computation models which greatly differ in the fixpoints used to represent iteration. Our axioms are general enough to capture the semantics of while-loops in all of these models, yet powerful enough to derive complex results including program transformations and refinement theorems.

We propose algebras that describe the infinite executions of a computation. In these algebras we define a unified approximation order, which applies to a wide variety of computation models. We thus obtain a unified semantics of recursion and results that connect fixpoints in the approximation and refinement orders. These results simplify reasoning about recursion and seamlessly match with our algebras for iteration when specialised to this case.

We propose algebras that describe preconditions and the effect of while-programs under postconditions. Based on these we unify correctness statements in two dimensions: one statement applies in various computation models to various correctness claims. Despite their generality and the weakness of the underlying axioms we can give a sound and relatively complete correctness calculus. It also covers computation models in which choices are made by two interacting agents.

The overarching algebraic method is to identify key aspects of computations, to describe these aspects by operations of algebras and to capture their properties by axioms. Theorems derived from these axioms express program transformations, correctness statements and refinement laws. We strive towards weaker axioms so as to capture more computation models. Reasoning about programs and specifications amounts to reasoning in the algebras, which can be supported by automated theorem proving technology. We extensively apply such tools to structure our results and to ensure their correctness.

Acknowledgements

The reported work was carried out while I was academic assistant at Ulm University (Germany), a visiting researcher at the University of Sheffield (UK) supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD), and a lecturer at the University of Canterbury (New Zealand). I thank these organisations for their support.

I thank Professor Helmuth Partsch for providing the freedom to carry out this research in Ulm. I thank Professor Georg Struth for being my host in Sheffield, for collaborating with me and for showing me Isabelle/HOL. I thank Professor Bernhard Möller, Dr Peter Höfner and numerous other researchers I have met at RelMiCS/AKA, RAMiCS, MPC and UTP conferences and IFIP WG 2.1 meetings for helpful discussions. I thank the anonymous referees of these conferences and various journals where the reported work has been published for their constructive feedback. I thank Steve Dunne for repeatedly supplying ideas which wanted to be expressed algebraically; solving these challenges has contributed to the foundations of the present work.

Walter Guttmann

Contents

Chapter 1

Introduction

Our daily lives increasingly depend on the effective operation of ever more complex software-intensive systems. Failures – for example, in telecommunication or energy supply – can be costly, disruptive, unacceptable or even dangerous. Methods, formalisms and tools need to be devised for the reliable construction of reliable systems. Software testing can detect many errors, but provides only partial assurance as the number of possible test cases is typically unbounded. Especially for critical systems it is important to go further by using mathematical methods that are based on computation models and give guarantees about the absence of errors. Software engineering needs to be underpinned by a firm understanding of computation models to ensure that the developed systems meet the required quality standards.

A computation model is a mathematical description of what happens when a program is run on a computer. Such models facilitate the construction of correct programs by mathematical calculation, in addition to the experimental method of testing programs. Models differ in the kinds of computation they can represent and the precision they achieve. Many computation models have been proposed, but they have not been explored systematically and their connections are not understood.

The present work proposes the use of algebra for giving structure to this diversity of models, for unifying existing approaches and for laying open their connections with an eye to discovering new models. Outcomes of this research are new computation models, new algebraic descriptions of computation models, new algebras that instantiate to several computation models, and characterisations of computation models according to which axioms they satisfy.

Our general approach is to investigate different computation models and the operations they support for constructing programs and specifications. We study the properties of these operations and introduce algebraic structures to describe them. Elements of the carriers of the algebras represent programs and specifications; they can be composed using program constructs built from the operations of the algebras; axioms capture properties of the operations; derived theorems express program transformations, correctness statements and refinement laws relating programs and specifications.

In this work we look at models of state-based non-deterministic sequential computations. Computations may be strict or non-strict and may support two kinds of non-deterministic choice representing the interaction of agents. We are interested in which kinds of correctness statement the models support, which kinds of execution – finite, infinite, aborting – they can represent and how they describe approximation which is the key ingredient for the semantics of recursion and iteration. Our aim is to give a unifying treatment of these aspects across different computation models. An immediate benefit is that any result – once established in the common framework – automatically holds in several computation models. Beyond that, the unification gives insight into how the models are related and where they differ essentially.

1.1 Structure of this work and contributions

For reasoning about programs it is necessary to specify their semantics. Approaches to program semantics are frequently classified into 'operational', 'denotational' and 'axiomatic'. Operational semantics is concerned with the stepwise execution of a program; denotational semantics describes the overall behaviour; axiomatic semantics deals with the properties of programs and program constructs. In this spectrum the present work is denotational in Chapter 2 and axiomatic in Chapters 3–5.

We work at different levels of abstraction, which helps to bridge the gap between a program's behaviour as executed on a physical machine and high-level properties, say, of iterations. The most concrete level is used in Chapter 2. It presents a selection of computation models based on relations, matrices of relations, and multirelations. Relations are the common theme because the computations we treat are state-based and non-deterministic. Moreover, relations are well understood mathematically and can be easily visualised as Boolean matrices or state transition graphs.

Chapter 2 discusses the various constants, operations, orders and properties that are treated algebraically in subsequent chapters, and shows how they are instantiated in different computation models. It also compares the models based on how they combine relations to represent computations. This comparison and systematic investigation leads to the discovery of new computation models that generalise previously existing ones.

A more abstract view is taken in Chapters 3–5 with the introduction of algebras to describe program constructs and their properties. The connection between the abstraction levels is that the computation models of Chapter 2 instantiate the algebras of the subsequent chapters. Any result derived from the axioms of an algebraic structure holds in all computation models that instantiate it. Not all models instantiate all structures and insight is gained by looking at the underlying reasons.

Chapter 3 discusses iteration, that is, the repeated sequential execution of a computation. Algebras for iteration in specific models have been investigated before; our novelty is the unified treatment of models which widely differ in the kinds of computation they support and how they represent iteration. The basic ideas are to weaken axioms and to generalise operations, but the difficulty is how to do this without giving up too many properties that can be derived. We can retain complex program transformations and refinement laws, including Kozen's while-program normal form theorem, Back's atomicity refinement theorem and Cohen's separation theorems [77, 6, 21]. These are results hitherto known from particular models and henceforth recognised to hold in many different computation models.

It is well known that iteration is the special case of recursion where a single recursive call is at the end of the body of the recursive program – often called tail-recursion. The aim of Chapter 4 is to apply the unifying treatment also to general recursion. Because the semantics of recursion is given by least fixpoints in a suitable approximation order that depends on the model, this amounts to unifying approximation orders. The idea here is to introduce an operation that describes the infinite executions of a computation and to define the approximation order in terms of this operation and the usual refinement order. We use a characteristic Galois connection of the operation to derive it for various computation models. Our main result reduces least fixpoints in the approximation order to least and greatest fixpoints in the refinement order. Because the refinement order – in concrete models, the subset order – is much simpler than the approximation order, this is useful for calculating with recursive programs. We give examples of such program reasoning and generalise the unfold-fold method of program construction to different computation models.

By instantiating the general results to the special case of tail-recursion we obtain an operation that satisfies the axioms for iteration given in Chapter 3. In this sense, our treatment of recursion is at an intermediate abstraction level between the relational models and the algebras for iteration.

Chapter 5 discusses correctness statements and generalises Hoare triples and Dijkstra's weakest preconditions in several ways [66, 28]. A correctness statement expresses that a

computation may have only certain kinds of execution – for example, that all executions must terminate in states satisfying a given property. Our treatment unifies both different kinds of correctness statement and different models they apply to. We achieve this by working in very general algebras requiring only a bare minimum of axioms to represent preconditions and the effect of while-programs. Surprisingly a sound and relatively complete correctness calculus can be given even in such a minimalistic setting. This generalises previous results to a wide variety of correctness statements and computation models including multirelations. We show examples applying the calculus to prove the correctness of multirelational and relational computations as well as to introduce while-loops in program refinement.

To ease the instantiation of the correctness algebras of Chapter 5 to concrete models we introduce algebras with modal operators at an intermediate abstraction level. A wide range of algebras – and thereby models – satisfy our correctness axioms: some previously known from the literature, some newly introduced in Chapters 3 and 4, and some even in many different ways.

In summary the contributions described in the present work are as follows:

* Chapter 2 systematically investigates models for state-based non-deterministic sequential computations. This results in changes to existing models, new models with more precision, approximation orders for existing and new models, and new kinds of correctness statement with more precision.

* Chapter 3 introduces new algebras which uniformly describe iteration in models of strict and non-strict computations. Existing results including complex program transformations are generalised to these algebras and hence to many models.

* Chapter 4 introduces new algebras which uniformly describe the infinite executions of computations in different models. They result in a unified approximation order and a method to calculate the semantics of recursions given by least fixpoints in this order.

* Chapter 5 introduces new algebras which uniformly describe correctness statements of varying precision for different computation models. Existing correctness calculi are generalised to these algebras and hence to many models.

Together these contributions demonstrate that algebras are a useful unifying tool, make a case for approaching the semantics of programs simultaneously at several abstraction levels, and show the benefits of weakening axioms as far as possible beyond what is needed for the current application.

1.2 Formalisation using Isabelle

Correctness of most results obtained in this work is assured by theorem proving technology. To this end we have implemented the algebras of Chapters 3–5 in Isabelle/HOL [93]. This has greatly assisted our work in two ways. First, Isabelle offers structuring and modularisation mechanisms necessary for working with large theories. In particular, we have built a hierarchy of algebras in which theorems are inherited; this avoids repeating proofs of similar results for different algebras. Second, while Isabelle is an interactive theorem prover, it has been integrated with external automated theorem provers, SMT solvers and counterexample generators [13, 96, 12]. Results obtained with the help of external tools can be trusted because proofs are reconstructed internally. The present work shows that the use of algebras for unifying semantics combines well with interactive and automated theorem proving.

Due to their size the resulting theories are given in a separate technical report [54]. They are structured into 40 Isabelle/HOL theory files with a total of more than 16000 lines containing more than 3000 proved or refuted facts. In particular, they contain proofs of all theorems in Chapters 3–5 except Theorems 33–36. A comment of the form '— Theorem n' precedes results in the theory files which contribute to Theorem n. Corresponding proofs are omitted in the following chapters.

1.3 Publications

This work is a revised and consolidated synthesis of selected parts of papers written by the author, which explore the algebraic approach to unifying computation models and have been published in conference proceedings and in journals [38, 41, 42, 44, 43, 57, 47, 46, 45, 48, 49, 52, 51, 50, 53].

Among the bigger omissions from these papers are typed omega algebras [44, 46] which are used to derive the omega operation for matrix-based computation models, the investigation of fixpoints in [43] which relate the approximation order to the median operation of lattice theory, a justification of changes to the computation model of extended designs [49], and more recent work on multirelations and their algebraic properties [52].

Each of Chapters 2–5 in the present work has a final section that briefly summarises the author's contributions and mentions the relevant papers. Related work by other authors is referred to throughout the text.

Chapter 2

Relational, matrix-based and multirelational computation models

In this chapter we describe several computation models that will be the subject of algebraic treatment in the remainder of this work. All models are state-based and feature non-determinism. They record only the input/output behaviour of a sequential computation but not the intermediate states.

Consider the set of states a computation can be in; for example, a state could be given by the values of program variables. Each state might have several possible successor states; an execution of the computation leads to one of these. Depending on the model, the involved non-deterministic choice can be characterised as angelic, demonic or erratic [16, 4]. Deterministic computations have exactly one successor per state; zero or more than one successors are useful for specification purposes.

Because the models do not represent intermediate states, we identify an *execution* with a single possible input/output behaviour of a computation. For example, this could be a pair of start and end states or the observation that the execution does not terminate when started in a particular state. We identify a *computation* with a set of possible input/output behaviours. Thus a computation is made up of a number of executions; non-determinism means there are several executions starting in the same state. A *finite* execution is one that terminates normally; an *infinite* execution does not terminate; an *aborting* execution fails due to an error such as division by zero.

All models presented in the following feature several operations acting on computations or states. The operations can be used to define program and specification constructs. They include non-deterministic choice, conjunction, sequential composition, various forms of finite and infinite iteration, preconditions, pre-post specifications, and an operation to describe the states from which infinite executions exist. Computations are related by two partial orders: refinement and approximation. Refinement amounts to the reduction of non-determinism, which is used for developing programs from specifications. 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 zeros of the operations, and least or greatest elements in the orders. Correctness statements relate computations and pre-/post-states. The properties of these constants, operations and relations are stated algebraically in subsequent chapters; this chapter describes how they instantiate in the various models.

The models are presented roughly in order of increasing expressiveness; the later models can typically 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 recursion and iteration. For example, the endless loop is an identity of non-deterministic choice in model M1, a zero in model M2 and neither in models M3–M8. To illustrate this, we compare for models M1–M5 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.

In Section 2.1 we describe the relational computation model M1 which represents only finite executions and is the basis for the following, more detailed models. The matrix-based models M2–M5 of Section 2.2 represent infinite and aborting executions, but only record whether such executions are present or absent from each starting state. The models M6–M8 of Section 2.3 relax this constraint. In Section 2.4 we present the relational and matrix-based models M9–M12 of non-strict computations, which can produce defined outputs from undefined inputs. Section 2.5 describes the multirelational computation model M13, which represents the interaction of two agents. The following chapters unify various aspects of these computation models briefly overviewed in Section 2.6. We conclude with a summary of our publications and their contributions relevant to this chapter in Section 2.7.

2.1 A relational computation model

Here and in the following sections, let A be the set of possible states of a computation. In model M1, a computation is a relation over the state space 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 which starts in state x and ends in state x'. More than one x' may be related to a given x, which amounts to non-determinism. For example, using $A = \mathbb{N}$, the relation S_1 given by

$$\begin{array}{rcl} 0 & \mapsto & \{0,3\} \\ 1 & \mapsto & \{2,4\} \\ 2 & \mapsto & \{4,5\} \\ 3 & \mapsto & \{6\} \\ 4 & \mapsto & \{7,8\} \\ & \vdots & \end{array}$$

models a computation that either adds 3 to its input or multiplies it by 2. Non-deterministic choice, conjunction and sequential composition are given by set union $\cup$, set intersection $\cap$ and relational composition, respectively. The composition QR of relations Q and R is

$$QR = \{(x, z) \in A \times A \mid \exists y \in A : (x, y) \in Q \wedge (y, z) \in R\}$$

It has higher precedence than union and intersection. See [107, 103, 102] for further details about relations.

The models of Chapter 2 generalise to heterogeneous relations, which are subsets of the Cartesian product $A \times B$ where A and B may be different. This signifies a change in the state space which is useful, for example, to introduce new variables in computations. Operations on relations then have to satisfy additional typing constraints. In the following we focus on the homogeneous case $A = B$.

All executions are finite in model M1; it represents neither infinite nor aborting executions. Accordingly, the set of states from which infinite executions exist is empty for each computation. However, the absence of finite executions can be interpreted as non-termination, in which case the endless loop is the empty relation $\mathsf{O} = \emptyset$. Because O is an identity of set union, choice is angelic with respect to non-termination: the non-deterministic choice between the endless loop and any relation R is just R.

The refinement relation is given by the subset order $\subseteq$. In model M1, also the approximation order $\sqsubseteq$ is the subset order, that is,

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

In all our computation models, recursions are solved by least fixpoints in the approximation order. Hence in model M1 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 operation * [22, 76]. Kleene algebras capture * as the least fixpoint of an affine function by unfold and induction axioms, and provide further properties useful for reasoning about programs. More details about operations for iteration follow in Chapter 3.

Specific constants in computation model M1 are the empty relation O, the identity relation $\mathsf{I} = \{(x, x) \mid x \in A\}$ and the universal relation $\mathsf{T} = A \times A$. They play their usual roles as identities, zeros, least and greatest elements. In particular, we obtain the $\sqsubseteq$-least element $\mathsf{L} = \mathsf{O}$ and $n(R) = \mathsf{O}$ for each R as there are no infinite executions in this model. More details about the operation n follow in Chapter 4.

Angelic choice renders the model suitable for partial correctness [66, 28, 103, 78, 23, 87]. For sets of states p and q, the Hoare triple $p\{R\}q$ expresses that every execution of R which starts in a state in p and terminates, does so in a state in q. Because the statement has to be proved for *every* execution of R, choice is demonic with respect to finite executions.

The Hoare triple $p\{R\}q$ is algebraically formalised by $p \cdot R \cdot q' \leq 0$ using tests p and q [78]. Tests represent subsets of the state space A and act as filters in a sequential composition. For example, in $p \cdot R$ the executions of R are restricted to those whose starting state is in the set described by p. In the relational model M1, tests are subsets of the identity relation I. The operation $'$ complements tests relative to I, the operation $\cdot$ is relational composition, $\leq$ is subset and 0 is the empty relation. According to the inequality $p \cdot R \cdot q' \leq 0$ there is no execution in R which starts in p and ends in a state not in q. More details about tests and correctness statements follow in Section 3.2.3 and in Chapter 5.

The Hoare triple can be expressed as $p \leq |R]q$ using a modal box operator [87]. This expresses that if a state is in the subset represented by p, all executions of R from that state lead to a state in the subset represented by q. Thus box represents the weakest liberal precondition in this computation model; it signifies the universal quantification that takes into account every execution of R. The box operator is defined as $|R]q = d(R \cdot q')'$ in terms of the domain operation d that describes the set of states from which a computation has any executions [23]. In the relational model M1, domain is given by $d(R) = R\mathsf{T} \cap \mathsf{I}$. A relation of the form $R\mathsf{T}$ is a *vector*, that is, it relates every state either to all states or to none, and therefore corresponds to a set of states. Intersection with I represents this set as a test. Composition with T turns a test into a vector representing the same states. More details about domain and box follow in Section 5.2.

2.2 Matrix-based models of strict computations

In this section we extend the relational computation model M1 to matrix-based models M2–M5 that take infinite and aborting executions into account. Because these models are more detailed they can represent more precise correctness statements.

2.2.1 Total correctness

Computation model M2 extends model M1 to represent the presence or absence of infinite executions. To this end, a component is added that represents the set of states from which infinite executions exist. Together with the finite executions this is conveniently described by a matrix [85]. Thus a computation is a 2×2 matrix whose entries are relations over the state space A. The matrix has the form

$$\begin{pmatrix} \mathsf{T} & \mathsf{T} \\ Q & R \end{pmatrix}$$

where $Q \subseteq R$ and Q is a vector. The vector Q represents the set of states from which there are infinite executions. The relation R represents the finite executions of the computation

in states where infinite executions do not exist, similarly to model M1. Additionally, states where infinite executions do exist are related by R to all states because $Q \subseteq R$. This means that the presence or absence of finite executions cannot be distinguished in the presence of infinite ones. Both entries in the top row are fixed to the universal relation T so as to propagate the infinite executions in the entry Q appropriately through sequential composition. Subsequent models feature matrices with other combinations of O, I and T entries providing a characteristic constant structure for each model.

For example, consider the relation S_2 on $A = \mathbb{N}$ given by

$$\begin{array}{ccc} 0 & \mapsto & \mathbb{N} \\ 1 & \mapsto & \emptyset \\ 2 & \mapsto & \mathbb{N} \\ 3 & \mapsto & \emptyset \\ 4 & \mapsto & \mathbb{N} \\ & \vdots & \end{array}$$

representing the set of states $\{0, 2, 4, \ldots\}$ as a vector. Then the matrix

$$\begin{pmatrix} \mathsf{T} & \mathsf{T} \\ S_2 & S_1 \cup S_2 \end{pmatrix}$$

models a computation that either adds 3 to its input x or multiplies it by 2, when x is odd, but need not terminate when x is even. Thus S_2 describes the states from which termination is not guaranteed.

The matrices in model M2 correspond to the 'designs' of the Unifying Theories of Programming [70, 85, 56]. Aborting executions are not represented or they are identified with infinite executions. Because finite executions are ignored in the presence of infinite ones, the model is suitable for total correctness. Non-deterministic choice and conjunction are obtained by componentwise union and intersection, respectively. Sequential composition is given by the matrix product, where union and relational composition play the roles of addition and multiplication, respectively. For example, the entry $Q_1 \cup R_1 Q_2$ in

$$\begin{pmatrix} \mathsf{T} & \mathsf{T} \\ Q_1 & R_1 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{T} & \mathsf{T} \\ Q_2 & R_2 \end{pmatrix} = \begin{pmatrix} \mathsf{T} & \mathsf{T} \\ Q_1 \cup R_1 Q_2 & Q_1 \cup R_1 R_2 \end{pmatrix}$$

shows that the sequential composition of two computations does not terminate if the first computation does not terminate or it leads to a state from which the second computation does not terminate. Refinement $\leq$ is given by the componentwise subset order. Approximation $\sqsubseteq$ is given by its converse, that is, the componentwise superset order:

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

Hence recursions are solved by greatest fixpoints in the componentwise subset order and iteration is obtained by a combination of the Kleene star and the omega operation $^\omega$ [21]. Omega algebras capture $^\omega$ as the greatest fixpoint of an affine function by unfold and induction axioms; more details follow in Chapter 3.

Specific constants are obtained by setting the entries Q and R of the 2×2 matrix as follows:

- $Q = \mathsf{O}$ and $R = \mathsf{O}$ yields the computation with no executions. It is the identity of non-deterministic choice, the zero of conjunction, a left zero of sequential composition, the least element in the refinement order and the greatest in the approximation order.
- $Q = \mathsf{O}$ and $R = \mathsf{I}$ yields the computation that does not change the state. It is the identity of sequential composition.

- $Q = \mathsf{O}$ and $R = \mathsf{T}$ yields the computation with all finite executions.
- $Q = \mathsf{T}$ and $R = \mathsf{T}$ yields the computation with all executions. In this model, it is the same as the computation with all infinite executions L. It is the zero of non-deterministic choice, the identity of conjunction, a left zero 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:

$$\mathsf{L} = \begin{pmatrix} \mathsf{T} & \mathsf{T} \\ \mathsf{T} & \mathsf{T} \end{pmatrix} \qquad n\begin{pmatrix} \mathsf{T} & \mathsf{T} \\ Q & R \end{pmatrix} = \begin{pmatrix} \mathsf{T} & \mathsf{T} \\ \mathsf{O} & Q \cap \mathsf{I} \end{pmatrix}$$

Note that $n(x)$ is a test, which is an element below the identity of sequential composition in the refinement order. The intersection $Q \cap \mathsf{I}$ converts the vector Q to a relation below the identity relation; it represents the states from which x has infinite executions. A test in model M2 is a 2×2 matrix in which the bottom-left entry is O and the bottom-right entry is a relational test. To obtain the complement $'$ of a test in model M2, the complement $'$ of the bottom-right test entry is taken.

Because L is its zero, non-deterministic choice is demonic with respect to non-termination. This renders the model suitable for total correctness [28, 70, 108, 55, 85, 19, 56]. The Hoare triple $p\{R\}q$ now expresses that every execution of R which starts in p terminates in q. Again this is formalised by $p \cdot R \cdot q' \leq 0$ [108], where p and q are tests, the operation $\cdot$ is the matrix product, 0 is the computation with no executions and $\leq$ is the refinement order. Using the matrix representations

$$p = \begin{pmatrix} \mathsf{T} & \mathsf{T} \\ \mathsf{O} & R_1 \end{pmatrix} \qquad R = \begin{pmatrix} \mathsf{T} & \mathsf{T} \\ Q & R_2 \end{pmatrix} \qquad q = \begin{pmatrix} \mathsf{T} & \mathsf{T} \\ \mathsf{O} & R_3 \end{pmatrix}$$

this correctness statement elaborates as

$$\begin{pmatrix} \mathsf{T} & \mathsf{T} \\ \mathsf{O} & R_1 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{T} & \mathsf{T} \\ Q_2 & R_2 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{T} & \mathsf{T} \\ \mathsf{O} & R_3' \end{pmatrix} = \begin{pmatrix} \mathsf{T} & \mathsf{T} \\ R_1Q_2 & R_1Q_2 \cup R_1R_2R_3' \end{pmatrix} \leq \begin{pmatrix} \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{O} \end{pmatrix}$$

which is equivalent to $R_1Q_2 \subseteq \mathsf{O} \wedge R_1R_2R_3' \subseteq \mathsf{O}$. We contrast $\subseteq$ on the components with its lifted counterpart $\leq$. The first term $R_1Q_2 \subseteq \mathsf{O}$ expresses that all executions starting in p terminate normally. The second term $R_1R_2R_3' \subseteq \mathsf{O}$ claims partial correctness: no execution starting in p terminates in q'.

However, partial correctness cannot be claimed alone. In particular, the 'weak correctness' statement $p \cdot R = p \cdot R \cdot q$ of [108] reduces to $R_1R_2R_3' \subseteq R_1Q_2$. This expresses that no execution starting in a state in p terminates in q', provided all executions starting in the same state terminate (whence $R_1Q_2 = \mathsf{O}$).

2.2.2 General correctness

Computation model M3 represents finite and infinite executions independently and thereby removes the restriction imposed by model M2. This is achieved by modifying the matrix structure. Model M3 uses 2×2 matrices of the form

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} \\ Q & R \end{pmatrix}$$

where Q is a vector. Now the top-right entry is O instead of T and the restriction $Q \subseteq R$ is abandoned. Otherwise the interpretations of Q and R remain as in model M2. 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 [29, 85]. 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 erratic non-determinism and general correctness [7, 17, 75, 9, 92, 79, 29, 88, 86, 38]. Non-deterministic choice, conjunction and refinement are represented as in model M2. Sequential composition is again given by the matrix product; due to the different structure this now elaborates as

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} \\ Q_1 & R_1 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ Q_2 & R_2 \end{pmatrix} = \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ Q_1 \cup R_1 Q_2 & R_1 R_2 \end{pmatrix}$$

Approximation, however, is given by the Egli-Milner order $\sqsubseteq$, according to which infinite executions may be removed and finite executions may be added only in the presence of infinite executions starting in the same state [32]:

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} \\ Q_1 & R_1 \end{pmatrix} \sqsubseteq \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ Q_2 & R_2 \end{pmatrix} \;\Leftrightarrow\; Q_2 \subseteq Q_1 \;\wedge\; R_1 \subseteq R_2 \subseteq R_1 \cup Q_1$$

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

* $Q = \mathsf{O}$ and $R = \mathsf{O}$ yields the computation with no executions. It is the identity of non-deterministic choice, the zero of conjunction, a left zero of sequential composition and the least element in the refinement order.
* $Q = \mathsf{O}$ and $R = \mathsf{I}$ yields the computation that does not change the state. It is the identity of sequential composition.
* $Q = \mathsf{O}$ and $R = \mathsf{T}$ yields the computation with all finite executions.
* $Q = \mathsf{T}$ and $R = \mathsf{O}$ yields the computation with all infinite executions L. It is a left zero of sequential composition and the least element in the approximation order.
* $Q = \mathsf{T}$ and $R = \mathsf{T}$ yields the computation with all executions. It is the zero 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:

$$\mathsf{L} = \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} \end{pmatrix} \qquad n\begin{pmatrix} \mathsf{T} & \mathsf{O} \\ Q & R \end{pmatrix} = \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ \mathsf{O} & Q \cap \mathsf{I} \end{pmatrix}$$

Computation model M3 supports both partial- and total-correctness statements, and is typically called 'general correctness' [75]. First, observe that

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} \\ \mathsf{O} & R_1 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ Q_2 & R_2 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ \mathsf{O} & R_3' \end{pmatrix} = \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ R_1 Q_2 & R_1 R_2 R_3' \end{pmatrix} \leq \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} \end{pmatrix}$$

is equivalent to the partial-correctness statement $R_1 R_2 R_3' \subseteq \mathsf{O}$. The matrix on the right-hand side of the inequality is L. On the level of computations the partial-correctness statement is thus formalised by $p \cdot R \cdot q' \leq \mathsf{L}$ [38]. This is equivalent to $p \cdot R = p \cdot R \cdot q$ in the current model. Second, set $R_3' = \mathsf{O}$ and observe that

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} \\ \mathsf{O} & R_1 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ Q_2 & R_2 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} \end{pmatrix} = \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ R_1 Q_2 & \mathsf{O} \end{pmatrix} \leq \begin{pmatrix} \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} \end{pmatrix}$$

is equivalent to $R_1 Q_2 \subseteq \mathsf{O}$, which expresses that all executions starting in p terminate normally. On the level of programs this is formalised by $p \cdot R \cdot 0 \leq 0$.

The conjunction of the two inequalities amounts to a total-correctness statement, but it is not required to use the same precondition p in both inequalities. This makes it possible to make statements about termination independently from statements about finite executions.

2.2.3 Extended designs

Computation model M4 extends models M1–M3 to represent the presence or absence of aborting executions. It uses 3×3 matrices of the form

$$\begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ P & Q & R \end{pmatrix}$$

where P and Q are vectors and $P \subseteq Q$ and $P \subseteq R$. The extra column stores the vector P that 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; this is similar to the restriction of model M2 which ignores finite executions in the presence of infinite ones.

The matrices of model M4 correspond to 'extended designs' [61, 49]. Non-deterministic choice, conjunction and refinement are as in models M2–M3; sequential composition is again the matrix product. The approximation order in this model is a variation of the Egli-Milner order:

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

An algebraic method to derive this order and approximation orders for other computation models will be given in Section 4.3.2. Specific constants are obtained by setting P, Q and R as follows:

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

In computation model M4, the $\sqsubseteq$-least element L and the operation n are:

$$\mathsf{L} = \begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \end{pmatrix} \qquad n\begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ P & Q & R \end{pmatrix} = \begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & Q \cap \mathsf{I} \end{pmatrix}$$

Several kinds of correctness statement are possible in this model. For the first, observe that

$$\begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & R_1 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ P_2 & Q_2 & R_2 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & R_3' \end{pmatrix} = \begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ R_1P_2 & R_1Q_2 & R_1P_2 \cup R_1R_2R_3' \end{pmatrix}$$

Because $P_2 \subseteq Q_2$ implies $R_1P_2 \subseteq R_1Q_2$,

$$\begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ R_1P_2 & R_1Q_2 & R_1P_2 \cup R_1R_2R_3' \end{pmatrix} \leq \begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & \mathsf{O} \end{pmatrix}$$

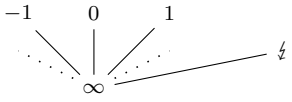
is equivalent to $R_1Q_2 \subseteq \mathsf{O} \wedge R_1R_2R_3' \subseteq \mathsf{O}$. Abstracting from the matrices to computations p, R and q' it follows that $p \cdot R \cdot q' \leq 0$ formalises a total-correctness statement, namely that all executions starting in p terminate in q. In particular, no infinite executions start in p and therefore also no aborting ones. Another correctness statement is obtained by

$$\begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ R_1P_2 & R_1Q_2 & R_1P_2 \cup R_1R_2R_3' \end{pmatrix} \leq \begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \end{pmatrix}$$

which is equivalent to $R_1P_2 \subseteq \mathsf{O} \wedge R_1R_2R_3' \subseteq \mathsf{O}$. The matrix on the right-hand side of the inequality is L. Hence the statement is formalised by $p \cdot R \cdot q' \leq \mathsf{L}$ and expresses that no aborting executions start in p and all finite executions starting there end in q. It states nothing about the infinite executions.

We obtain two further statements by setting $R_3' = \mathsf{O}$ again. First, $p \cdot R \cdot 0 \leq 0$ expresses that no infinite and hence no aborting executions start in p. Second, $p \cdot R \cdot 0 \leq \mathsf{L}$ means that no aborting executions start in p. It is thus possible to make statements about aborting and infinite executions, but partial correctness cannot be claimed alone. In particular, $p \cdot R = p \cdot R \cdot q$ reduces to $R_1R_2R_3' \subseteq R_1P_2$, which expresses that no execution starting in a state in p terminates in q', provided no execution starting in the same state aborts.

To provide another view of the approximation order in model M4, we illustrate the connection to powerdomains [106]. It is given by considering a program with a single variable and reflecting on the possible outcomes for a fixed starting state. Assume that the value range of the variable is $\mathbb{Z}$ from which we obtain the domain $Int = \mathbb{Z} \cup \{\infty, \lightning\}$ by adding two special elements ∞ and $\lightning$. The element ∞ represents the infinite execution, and $\lightning$ represents the aborting execution. The order $\preceq$ on Int is flat with ∞ as least element, that is, we have $x \preceq y \Leftrightarrow x = \infty \vee x = y$:



In particular, $\lightning$ is treated like any other element except ∞. The Plotkin powerdomain of Int can be visualised as in Figure 1 on the next page showing the sets without ∞ as maximal elements.

For extended designs, however, the aborting outcome absorbs all other outcomes in a similar way as the infinite outcome does in the Smyth powerdomain. Hence every set containing $\lightning$ is identified with the set $\{\lightning\}$. The resulting order is shown in Figure 2 on the next page, in which the set $\{\lightning\}$ is above all sets containing ∞.

2.2.4 Independent finite, infinite and aborting executions

In model M5, aborting, finite and infinite executions are represented independently [45]. This is achieved by modifying the structure of the 3×3 matrices of model M4. Computation model M5 uses 3×3 matrices of the form

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ P & Q & R \end{pmatrix}$$

where P and Q are vectors. Now the second and third entries in the top row are O instead of T and the restrictions $P \subseteq Q$ and $P \subseteq R$ are abandoned. Otherwise the interpretations

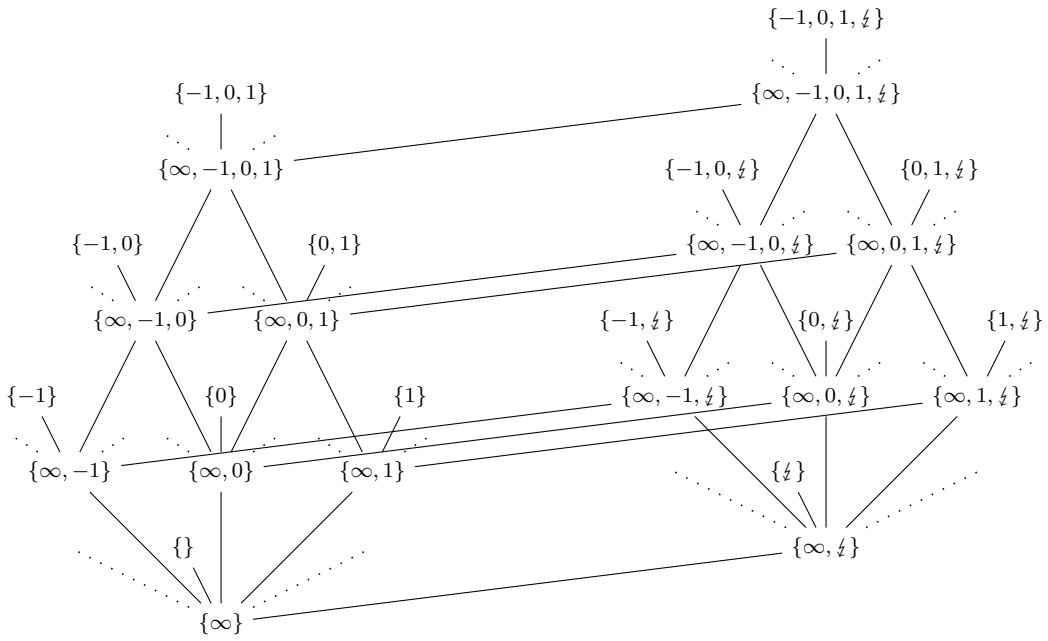


Figure 1: The Plotkin powerdomain of *Int*, based on [106, Figure 3]

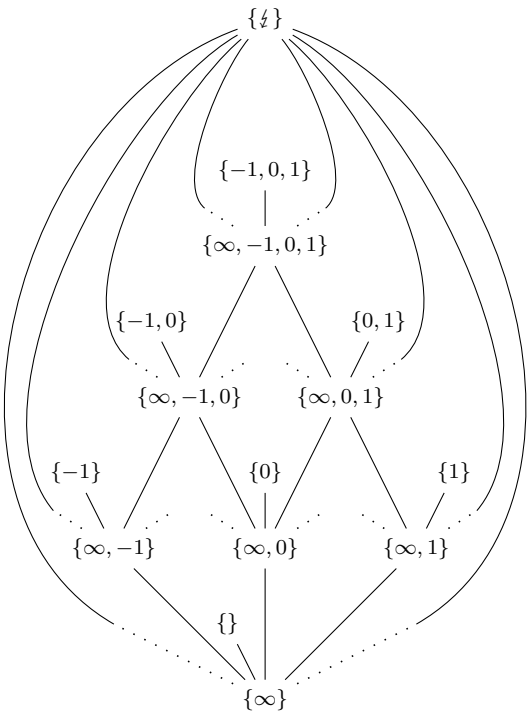


Figure 2: The powerdomain of *Int* as required for extended designs

of P, Q and R remain as in model M4. 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 and refinement are as in models M2–M4; sequential composition is the matrix product. Another variation of the Egli-Milner order is used for approximation:

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ P_1 & Q_1 & R_1 \end{pmatrix} \sqsubseteq \begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ P_2 & Q_2 & R_2 \end{pmatrix} \;\Leftrightarrow\; \begin{array}{l} Q_2 \subseteq Q_1 \;\wedge \\ P_1 \subseteq P_2 \subseteq P_1 \cup Q_1 \;\wedge \\ R_1 \subseteq R_2 \subseteq R_1 \cup Q_1 \end{array}$$

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

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

In model M5, we obtain the $\sqsubseteq$-least element L and the operation n as follows:

$$\mathsf{L} = \begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \end{pmatrix} \qquad n\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ P & Q & R \end{pmatrix} = \begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & Q \cap \mathsf{I} \end{pmatrix}$$

Statements about finite, infinite and aborting executions can also be made independently; see [59] for a derivation from different execution methods based on computation trees. Observe that

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & R_1 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ P_2 & Q_2 & R_2 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & R_3' \end{pmatrix} = \begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ R_1P_2 & R_1Q_2 & R_1R_2R_3' \end{pmatrix}$$

An inequality may be formed with either 0, L, A or $\mathsf{L} + \mathsf{A}$ on the right-hand side, where $+$ represents non-deterministic choice and the computations 0, L and A are the matrices

$$0 = \begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & \mathsf{O} \end{pmatrix} \qquad \mathsf{L} = \begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \end{pmatrix} \qquad \mathsf{A} = \begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} & \mathsf{O} \end{pmatrix}$$

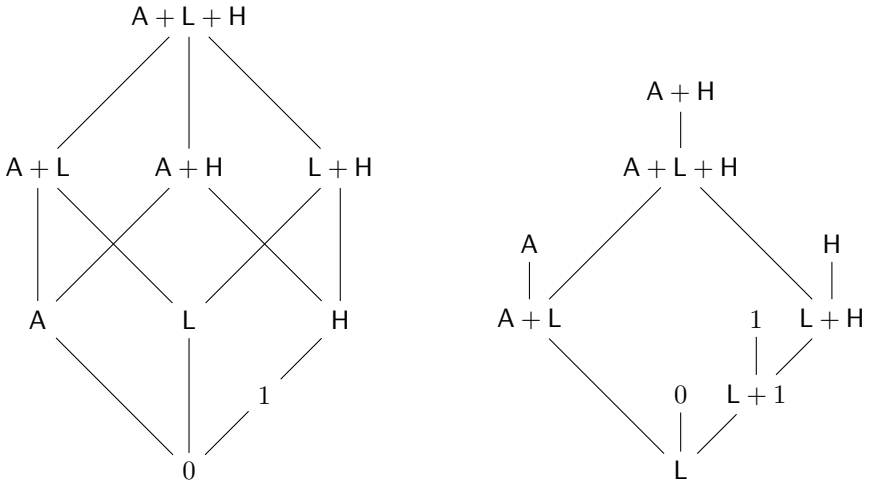


Figure 3: Refinement order (left) and approximation order (right) in model M5

The computation A contains all aborting executions. Using $p \cdot R \cdot q'$ or $p \cdot R \cdot 0$ on the left-hand side, we obtain the following seven kinds of correctness statement:

$$
\begin{array}{lllllllll}
(7) & p \cdot R \cdot q' \leq 0 & \Leftrightarrow & R_1 P_2 \subseteq \mathsf{O} & \wedge & R_1 Q_2 \subseteq \mathsf{O} & \wedge & R_1 R_2 R_3' \subseteq \mathsf{O} \\
(6) & p \cdot R \cdot q' \leq \mathsf{A} & \Leftrightarrow & & & R_1 Q_2 \subseteq \mathsf{O} & \wedge & R_1 R_2 R_3' \subseteq \mathsf{O} \\
(5) & p \cdot R \cdot q' \leq \mathsf{L} & \Leftrightarrow & R_1 P_2 \subseteq \mathsf{O} & \wedge & & & R_1 R_2 R_3' \subseteq \mathsf{O} \\
(4) & p \cdot R \cdot q' \leq \mathsf{L} + \mathsf{A} & \Leftrightarrow & & & & & R_1 R_2 R_3' \subseteq \mathsf{O} \\
(3) & p \cdot R \cdot 0 \leq 0 & \Leftrightarrow & R_1 P_2 \subseteq \mathsf{O} & \wedge & R_1 Q_2 \subseteq \mathsf{O} & & \\
(2) & p \cdot R \cdot 0 \leq \mathsf{A} & \Leftrightarrow & & & R_1 Q_2 \subseteq \mathsf{O} & & \\
(1) & p \cdot R \cdot 0 \leq \mathsf{L} & \Leftrightarrow & R_1 P_2 \subseteq \mathsf{O} & & & &
\end{array}
$$

Of particular interest are statements (1), (2) and (4) since the other statements are obtained as their conjunctions. Statement (1) expresses the absence of aborting executions from states in p, and statement (2) expresses the absence of infinite executions. Statement (4) is partial correctness and equivalent to $p \cdot R = p \cdot R \cdot q$ in model M5. For another example, the total-correctness statement (6) expresses the absence of infinite executions in addition to partial correctness.

The Hasse diagrams in Figure 3 above show how the constants of this model are related by the refinement order and the approximation order; the computations 1 and H are

$$
1 = \begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & \mathsf{I} \end{pmatrix} \qquad \mathsf{H} = \begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & \mathsf{T} \end{pmatrix}
$$

The matrix H represents the computation with all finite executions. Note that $\mathsf{A}+\mathsf{L}+\mathsf{H} = \top$ in model M5, where $\top$ is the greatest element in the refinement order $\leq$.

2.2.5 Summary

Figure 4 on the next page provides an overview of the specific constants, the element L, the operation n and the approximation order $\sqsubseteq$ in models M1–M5. Infinite executions are ignored in model M1; aborting executions are ignored in models M1–M3.

In models M1–M5, the computation L with all infinite executions is not only the $\sqsubseteq$-least element, but also a left zero of sequential composition. This property characterises strict

executions	M1	M2	M3	M4	M5
none	O	$\begin{pmatrix}\mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{O}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{O}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{O} & \mathsf{O}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{O} & \mathsf{O}\end{pmatrix}$
no state change	I	$\begin{pmatrix}\mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{I}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{I}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{O} & \mathsf{I}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{O} & \mathsf{I}\end{pmatrix}$
all finite	T	$\begin{pmatrix}\mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{O} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{O} & \mathsf{T}\end{pmatrix}$
all infinite L	O	$\begin{pmatrix}\mathsf{T} & \mathsf{T}\\ \mathsf{T} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O}\\ \mathsf{T} & \mathsf{O}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\end{pmatrix}$
all infinite and finite	T	$\begin{pmatrix}\mathsf{T} & \mathsf{T}\\ \mathsf{T} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O}\\ \mathsf{T} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{T}\end{pmatrix}$
all aborting	O	$\begin{pmatrix}\mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{O}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{O}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{T} & \mathsf{T} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{T} & \mathsf{O} & \mathsf{O}\end{pmatrix}$
all aborting and finite	T	$\begin{pmatrix}\mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{T} & \mathsf{T} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{T} & \mathsf{O} & \mathsf{T}\end{pmatrix}$
all aborting and infinite	O	$\begin{pmatrix}\mathsf{T} & \mathsf{T}\\ \mathsf{T} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O}\\ \mathsf{T} & \mathsf{O}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{T} & \mathsf{T} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{T} & \mathsf{T} & \mathsf{O}\end{pmatrix}$
all aborting, infinite and finite	T	$\begin{pmatrix}\mathsf{T} & \mathsf{T}\\ \mathsf{T} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O}\\ \mathsf{T} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{T} & \mathsf{T} & \mathsf{T}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{T} & \mathsf{T} & \mathsf{T}\end{pmatrix}$
x_i	R_i	$\begin{pmatrix}\mathsf{T} & \mathsf{T}\\ Q_i & R_i\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O}\\ Q_i & R_i\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ P_i & Q_i & R_i\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ P_i & Q_i & R_i\end{pmatrix}$
$n(x_i)$	O	$\begin{pmatrix}\mathsf{T} & \mathsf{T}\\ \mathsf{O} & Q_i \cap \mathsf{I}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O}\\ \mathsf{O} & Q_i \cap \mathsf{I}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{T} & \mathsf{T}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{O} & Q_i \cap \mathsf{I}\end{pmatrix}$	$\begin{pmatrix}\mathsf{T} & \mathsf{O} & \mathsf{O}\\ \mathsf{O} & \mathsf{T} & \mathsf{O}\\ \mathsf{O} & \mathsf{O} & Q_i \cap \mathsf{I}\end{pmatrix}$
$x_1 \sqsubseteq x_2$	$R_1 \subseteq R_2$	$Q_2 \subseteq Q_1 \wedge$ $R_2 \subseteq R_1$	$Q_2 \subseteq Q_1 \wedge$ $R_1 \subseteq R_2 \wedge$ $R_2 \subseteq R_1 \cup Q_1$	$Q_2 \subseteq Q_1 \wedge$ $P_1 \subseteq P_2 \wedge$ $R_1 \subseteq R_2 \wedge$ $R_2 \subseteq R_1 \cup Q_1$	$Q_2 \subseteq Q_1 \wedge$ $P_1 \subseteq P_2 \wedge$ $P_2 \subseteq P_1 \cup Q_1 \wedge$ $R_1 \subseteq R_2 \wedge$ $R_2 \subseteq R_1 \cup Q_1$
restrictions for x_i	–	$Q_i \subseteq R_i$	–	$P_i \subseteq Q_i \wedge$ $P_i \subseteq R_i$	–

Figure 4: Specific computations, the element L, the operation n and the approximation order $\sqsubseteq$ in models M1–M5

computations. Intuitively, it states that no computations are performed after an endless loop. In Section 2.4 we will look at models M9–M12 of non-strict computations, which do not satisfy this property.

Models M1–M5 vary in the precision with which they can describe finite, infinite and aborting executions. These models support various kinds of correctness statement limited by their precision. Many statements take the form $p \cdot R \cdot q' \leq \mathsf{Z}$ for a constant Z depending on the statement and the model. The computation Z is either 0, L, A or $\mathsf{L} + \mathsf{A}$ and captures the executions that are ignored by the correctness statement. The test p is the precondition, and the test q' is the complement of the postcondition or 0 if nothing is claimed about finite executions. As shown in Section 5.2 also this more general statement can be expressed as $p \leq |R]q$ using a relativised modal box operator. Depending on the computation model, the box operator represents the weakest liberal precondition, the weakest precondition or other kinds of precondition, which avoid aborting executions in addition to finite or infinite ones. A unifying algebraic treatment of these preconditions and correctness statements is given in Section 5.1.

2.3 Matrix-based models with unrestricted infinite or aborting executions

Whenever matrix-based models M2–M5 represent infinite or aborting executions, they only record if such executions are present or absent from each starting state. In particular, it is not possible to provide additional information about such executions, for example, the last state before an execution aborts. The restriction is effected by requiring some matrix entries to be vectors; in this section we generalise these to arbitrary relations. When computations are viewed as assumption/commitment specifications, this means that assumptions can also refer to post-states, not just to pre-states. This generalisation provides a basis for more detailed models that involve time [64, 61, 62, 30], though we do not address these in the present work.

Two computation models that lift the restriction have been described in [30]. They are discussed in Sections 2.3.1 and 2.3.2. We make a further step of generalisation with the model presented in Section 2.3.3.

2.3.1 Conscriptions

Computation model M6 generalises model M3 by lifting the restriction to vectors. The matrices in model M6 correspond to the 'conscriptions' of [30] and have the form

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q & R \end{pmatrix}$$

Now the top-left entry is I instead of T and the restriction that Q has to be a vector is abandoned. The relation Q represents the infinite executions (the complement of the assumption) and the relation R represents the finite executions (the commitment). Nondeterministic choice, conjunction, sequential composition and refinement are as in model M3. The computation which does not change the state and the endless loop are

$$1 = \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} \end{pmatrix} \qquad \mathsf{L} = \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} \end{pmatrix}$$

The computation 1 is an identity of sequential composition and L is a left zero.

To define the semantics of recursion we need a suitable approximation order $\sqsubseteq$. Because the endless loop L has to be the $\sqsubseteq$-least element, the refinement order cannot be used for approximation.

In the following we discuss two attempts to define an approximation order for computation model M6 and the reasons why they fail. The first attempt is to take the approximation

order of model M3 [38, 31]. Computations in model M3 correspond to a subset of the computations in model M6, namely those where the component Q is a vector. It stands to reason that the approximation order for model M6 specialises to the approximation order for model M3 when restricted to this subset. The approximation order $\sqsubseteq_1$ in model M6 would accordingly be defined as

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q_1 & R_1 \end{pmatrix} \sqsubseteq_1 \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q_2 & R_2 \end{pmatrix} \;\Leftrightarrow\; Q_2 \subseteq Q_1 \;\wedge\; R_1 \subseteq R_2 \subseteq R_1 \cup Q_1$$

The intuition underlying $\sqsubseteq_1$ is that in states with infinite executions, finite executions can be added but only such that have the same output. A problem with $\sqsubseteq_1$ is that sequential composition from the right is not $\sqsubseteq_1$-isotone. Namely,

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{I} & \mathsf{O} \end{pmatrix} \sqsubseteq_1 \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} \end{pmatrix}$$

but

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{I} & \mathsf{O} \end{pmatrix} \cdot \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{T} \end{pmatrix} = \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{I} & \mathsf{O} \end{pmatrix} \not\sqsubseteq_1 \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{T} \end{pmatrix} = \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} \end{pmatrix} \cdot \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{T} \end{pmatrix}$$

The second attempt to define an approximation order converts computations in model M6 to computations in model M3 and takes their order:

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q_1 & R_1 \end{pmatrix} \sqsubseteq_2 \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q_2 & R_2 \end{pmatrix} \;\Leftrightarrow\; \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q_1\mathsf{T} & R_1 \end{pmatrix} \sqsubseteq_1 \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q_2\mathsf{T} & R_2 \end{pmatrix}$$

The Q-components are converted to vectors by composing them with T. Hence the resulting computation has an infinite execution from state x if the original computation has any infinite execution starting in x. The intuition underlying $\sqsubseteq_2$ is that in states with any infinite executions, any finite execution can be added. A problem with $\sqsubseteq_2$ is that it is not antisymmetric. Namely,

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{I} & \mathsf{O} \end{pmatrix} \sqsubseteq_2 \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} \end{pmatrix} \qquad \text{and} \qquad \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} \end{pmatrix} \sqsubseteq_2 \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{I} & \mathsf{O} \end{pmatrix}$$

More generally, it is inconsistent to assume all of the following four properties of an approximation relation $\sqsubseteq$ for model M6:

1. $\sqsubseteq$ is a partial order,
2. sequential composition from the right is $\sqsubseteq$-isotone,
3. $\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} \end{pmatrix} \sqsubseteq \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} \end{pmatrix}$,
4. $\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{I} & \mathsf{O} \end{pmatrix} \sqsubseteq \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} \end{pmatrix}$.

This is because they would imply

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{I} & \mathsf{O} \end{pmatrix} = \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{I} & \mathsf{O} \end{pmatrix} \cdot \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} \end{pmatrix} \sqsubseteq \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} \end{pmatrix} \cdot \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} \end{pmatrix} = \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} \end{pmatrix}$$

and

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} \end{pmatrix} = \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} \end{pmatrix} \cdot \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{I} & \mathsf{O} \end{pmatrix} \sqsubseteq \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} \end{pmatrix} \cdot \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{I} & \mathsf{O} \end{pmatrix} = \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{I} & \mathsf{O} \end{pmatrix}$$

and therefore

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{I} & \mathsf{O} \end{pmatrix} = \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} \end{pmatrix}$$

A suitable approximation order for model M6 can be obtained using the algebraic method presented in Section 4.3.2. It will satisfy the first three properties of the above list, but not the last one. Along with the approximation order we will derive the operation n for models M6–M8.

2.3.2 Extended conscriptions

Computation model M6 represents infinite and finite executions independently, but has no notion of aborting executions. This is improved in model M7, which represents aborting, infinite and finite executions independently. In this model, there is no restriction on aborting executions, but the restriction to vectors still applies for infinite executions. Hence model M7 combines aspects of three computation models:

* It covers aborting executions in addition to finite and infinite executions as model M4.
* It represents aborting, infinite and finite executions independently as model M5.
* Aborting executions can refer to post-states as infinite executions do in model M6.

Computations in model M7 correspond to the 'extended conscriptions' of [30] and have the form

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ P & Q & R \end{pmatrix}$$

where Q is a vector, that is, $Q\mathsf{T} = Q$. The relation P represents the aborting executions, Q represents the states from which infinite executions exist and R represents the finite executions. Non-deterministic choice, conjunction, sequential composition and refinement are as in model M5. The computation which does not change the state and the endless loop are

$$1 = \begin{pmatrix} \mathsf{I} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & \mathsf{I} \end{pmatrix} \qquad \mathsf{L} = \begin{pmatrix} \mathsf{I} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \end{pmatrix}$$

The computation 1 is an identity of sequential composition and L is a left zero. The algebraic method of Section 4.3.2 can be used to derive an approximation order for computation model M7.

2.3.3 Unrestricted infinite and aborting executions

A comparison of the computation models M2–M7 suggests a further generalisation of model M7 by eliminating the restriction placed on infinite executions. This is done in a similar way as for model M6 and for the aborting executions of model M7. Thus a computation in model M8 is a 3×3 matrix of the following form:

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} & \mathsf{O} \\ P & Q & R \end{pmatrix}$$

There are no restrictions on P, Q or R. The relation P represents the aborting executions, Q represents the infinite executions and R represents the finite executions. Non-deterministic choice, conjunction, sequential composition and refinement are as in model M5. The computation which does not change the state and the endless loop are

$$1 = \begin{pmatrix} \mathsf{I} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & \mathsf{I} \end{pmatrix} \qquad \mathsf{L} = \begin{pmatrix} \mathsf{I} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \end{pmatrix}$$

The computation 1 is an identity of sequential composition and L is a left zero. The approximation order for model M8 will be derived in Section 4.3.2.

Model M8 is the most precise among the models M1–M8: it can represent finite, infinite and aborting executions independently and without any restrictions. Models M1–M7 appear

as substructures of model M8 by applying the following restrictions to computations:

$$
\begin{array}{lllllllll}
\text{M7:} & & & Q = Q\mathsf{T} & & & & & \\
\text{M6:} & P = \mathsf{O} & & & & & & & \\
\text{M5:} & P = P\mathsf{T} & \wedge & Q = Q\mathsf{T} & & & & & \\
\text{M4:} & P = P\mathsf{T} & \wedge & Q = Q\mathsf{T} & \wedge & P \subseteq Q & \wedge & P \subseteq R \\
\text{M3:} & P = \mathsf{O} & \wedge & Q = Q\mathsf{T} & & & & & \\
\text{M2:} & P = \mathsf{O} & \wedge & Q = Q\mathsf{T} & \wedge & Q \subseteq R & & & \\
\text{M1:} & P = \mathsf{O} & \wedge & Q = \mathsf{O} & & & & &
\end{array}
$$

Other restrictions lead to further computation models which can be represented by matrices. For example,

- $P = P\mathsf{T}$ requires that aborting executions do not refer to post-states;
- $P = P\mathsf{T} \wedge Q = Q\mathsf{T} \wedge Q \subseteq P \wedge Q \subseteq R$ requires that aborting and infinite executions do not refer to post-states and that in the presence of infinite executions, neither aborting nor finite executions can be distinguished.

Further combinations are possible, but in each case it has to be verified that the subset is closed under operations such as sequential composition.

Note that less precise models are not rendered superfluous. First, the ability to represent more computations comes at the price of increased complexity, for example, more complex approximation orders. Second, models such as M1, M2 and M8 use different approximation orders. Our algebraic treatment helps to develop a common theory to overcome these issues.

2.4 Relational and matrix-based models of non-strict computations

Models M1–M8 describe *strict* computations, which cannot produce a defined output from an undefined input. This matches the conventional execution of imperative programs. For example, if $S = (\mathsf{while\ true\ do}\ x := x)$ is the endless loop or $S = (x := 1/0)$ aborts, then $S\,;\, R = S$ for every program R as the execution of the sequential composition $S\,;\, R$ never reaches R. However, there are also models of non-strict computations, which can recover from undefined input [39]. In such models, for example, $S\,;\,(x := 2) = (x := 2)$ holds for either of the above definitions of S, assuming the state contains the single variable x. As elaborated in [39], this makes it possible to construct and compute with infinite data structures.

In particular, the endless loop L is not a left zero of sequential composition for *non-strict* computations. Intuitively, such computations can recover from undefined inputs: for example, $\mathsf{L}\,;\,(x := 2)$ will produce the output 2 and similarly for sequential compositions with other constant assignments.

In the following we describe models M9–M12 of non-strict computations. We first discuss the models M9–M10, which are simplified for computations having a single variable with an elementary type. They show the connection to the matrix-based models of strict computations. Relational models M11–M12 generalise to several variables and complex data types.

2.4.1 A matrix-based model with a flat state space

In model M9, a computation is a 3×3 matrix whose entries are relations over the state space A. The matrix has the form

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

where the relations P and Q are vectors, W and Z are relational converses of vectors and $U, V, X, Y \in \{\mathsf{O}, \mathsf{T}\}$. The converse of a relation R is given by $R^{\mathsf{c}} = \{(y, x) \mid (x, y) \in R\}$. Hence the converse of a vector is a relation in which every state is related to the same set of states. The first, second and third rows of the matrix indicate the transitions if the preceding execution is aborting, infinite or finite, respectively. In each of these cases, the first, second and third columns of the matrix 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 $U = \mathsf{T}$ and $V = W = \mathsf{O}$, the present computation aborts, too, which propagates the error as typical for strict computations. A non-strict computation, however, might have $U = V = \mathsf{O}$ 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 $X = Z = \mathsf{O}$ and $Y = \mathsf{T}$. A non-strict computation might recover with $X = Y = \mathsf{O}$ and particular finite executions in Z. Finally, the entries P, Q and R retain their meaning from computation model M5. They describe the executions if the preceding execution terminates normally.

Non-deterministic choice, conjunction, sequential composition, refinement and approximation are adapted from model M2, that is, are componentwise union, componentwise intersection, matrix product, componentwise subset and componentwise superset, respectively. Hence recursions are solved by greatest fixpoints in the componentwise subset order. For the following examples of computations in model M9, 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

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & \mathsf{I} \end{pmatrix}$$

* Let $S_3 = \{(x, x') \mid x' = x + 1\}$. The assignment $x := x + 1$ is the (strict) computation

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & S_3 \end{pmatrix}$$

* Let $S_4 = \{(x, x') \mid x' = 2\}$. The strict version of the assignment $x := 2$ is

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{O} & S_4 \end{pmatrix}$$

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

$$\begin{pmatrix} \mathsf{O} & \mathsf{O} & S_4 \\ \mathsf{O} & \mathsf{O} & S_4 \\ \mathsf{O} & \mathsf{O} & S_4 \end{pmatrix}$$

* The following computation $\preccurlyeq$ effects a closure; it will be used in Section 2.4.2:

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{O} & \mathsf{I} \end{pmatrix}$$

∗ The computation with all aborting executions is

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} & \mathsf{O} \end{pmatrix}$$

∗ The computation with all infinite executions is

$$\begin{pmatrix} \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \\ \mathsf{O} & \mathsf{T} & \mathsf{O} \end{pmatrix}$$

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

$$\begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{T} & \mathsf{T} & \mathsf{T} \end{pmatrix}$$

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 work; we just note that the matrices with $U = Y = \mathsf{T}$ and $V = W = X = Z = \mathsf{O}$ are the strict computations of model M5, which uses a variant of the Egli-Milner order.

2.4.2 A matrix-based total-correctness model with a flat state space

Using the componentwise superset order for approximation requires a different interpretation of infinite executions. Namely, the semantics of the endless loop 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 M2 and technically achieved by requiring the matrices to be closed with respect to sequential composition with the computation $\preccurlyeq$ given in Section 2.4.1. We therefore eliminate the matrices that are not closed in this way.

Model M10 uses a subset of the matrices of model M9 with the same operations. The matrices still have the form

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

where the relations P and Q are vectors, W and Z are relational converses of vectors and $U, V, X, Y \in \{\mathsf{O}, \mathsf{T}\}$. 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 left specify that both the first row and the third row of a matrix are componentwise below its second row. The six conditions on the right 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 in the componentwise subset order. The closed versions of the examples of Section 2.4.1 are:

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

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{O} & \mathsf{I} \end{pmatrix}$$

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

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{O} & S_3 \end{pmatrix}$$

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

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{O} & \mathsf{O} & S_4 \end{pmatrix}$$

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

$$\begin{pmatrix} \mathsf{O} & \mathsf{O} & S_4 \\ \mathsf{O} & \mathsf{O} & S_4 \\ \mathsf{O} & \mathsf{O} & S_4 \end{pmatrix}$$

* The computation with all aborting executions is

$$\begin{pmatrix} \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} & \mathsf{O} \\ \mathsf{T} & \mathsf{O} & \mathsf{O} \end{pmatrix}$$

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

$$\begin{pmatrix} \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{T} & \mathsf{T} & \mathsf{T} \\ \mathsf{T} & \mathsf{T} & \mathsf{T} \end{pmatrix}$$

2.4.3 A relational model with complex data types

We now work towards generalising models M9–M10 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 $U, V, X, Y \in \{\mathsf{O}, \mathsf{T}\}$. Because of these restrictions, each 3×3 matrix of relations over A 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 model M9. 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 [34].

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 need not be 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 [39].

We can thus generalise the simplified models M9–M10 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 work.

Model M11 generalises model M9; a computation is a relation over the extended state space A'. Using A' instead of A allows us to represent infinite and aborting executions in contrast to model M1. Non-deterministic choice, conjunction, sequential composition and refinement are as in model M1. Approximation is given by the superset order, whence recursions are solved by greatest fixpoints in the subset order. The relations O, I and T play their usual roles as identities, zeros, least and greatest elements. It follows that $\mathsf{L} = \mathsf{T}$ in this model. The operation n for models M11–M12 will be derived in Section 4.3.3. We translate the examples of computations given in Section 2.4.1, which use a single variable $x \in A' = \mathbb{N} \cup \{\infty, \lightning\}$.

* The (strict) computation that does not change the state is the identity relation I.
* The assignment $x := x + 1$ is the (strict) computation $\{(\lightning, \lightning), (\infty, \infty)\} \cup S_3$.
* The strict version of the assignment $x := 2$ is $\{(\lightning, \lightning), (\infty, \infty)\} \cup S_4$.
* The non-strict version of the assignment $x := 2$ is $\{(x, 2) \mid x \in A'\}$.
* Being a relation over A', the partial order $\preccurlyeq = \{(\infty, x') \mid x' \in A'\} \cup \mathsf{I}$ 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 T.

2.4.4 A relational total-correctness model with complex data types

Regarding the interpretation of infinite executions, the remarks made for model M9 apply accordingly to model M11. Computations which are not closed with respect to sequential composition with $\preccurlyeq$ are therefore eliminated.

In model M12, 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 model M11. The relation $\preccurlyeq$ replaces I as the identity of sequential composition. The $\preccurlyeq$-closed versions of the examples of Section 2.4.3 are:

* The (strict) computation that does not change the state is $\preccurlyeq$.
* The assignment $x := x + 1$ is $\{(\lightning, \lightning)\} \cup \{(\infty, x') \mid x' \in A'\} \cup S_3$.
* The strict version of the assignment $x := 2$ is $\{(\lightning, \lightning)\} \cup \{(\infty, x') \mid x' \in A'\} \cup S_4$.
* 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 $\mathsf{L} = \mathsf{T}$, which is also the computation with all executions.

It is shown in [39] 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.

2.5 A multirelational computation model

In models M1–M12, non-deterministic choices are made by a single agent. These choices are demonic in the sense that all executions must be taken into account to guarantee correctness. We now look at the multirelational model M13 which considers two interacting agents. This is useful for modelling contracts between agents, interaction, games and protocols [95, 4, 80].

Model M13 can represent computations that involve two kinds of non-determinism associated with players in a game [95] and sometimes called angelic and demonic [4, 101]. The underlying intuition is that there are two players: the 'angel' who makes a choice, and the 'demon' who subsequently makes a choice based on the angel's selection. To achieve this, relations and their matrix-based extensions are replaced with up-closed multirelations or isotone predicate transformers. The latter two structures are isomorphic [4, 100, 65]; we use up-closed multirelations in the following. An early application of up-closed multirelations is found in [3] for describing contact and topology.

A relation is a subset of the Cartesian product $A \times A$ for a set A. In contrast, a *multirelation* [100] is a subset of the Cartesian product $A \times 2^A$ where 2^A is the powerset of A. Thus it maps an element of A to a set of subsets of A. A multirelation R is *up-closed* if $(x, X) \in R \wedge X \subseteq Y \Rightarrow (x, Y) \in R$ for each $x \in A$ and $X, Y \subseteq A$. Thus, if an element of A is related to a set X, it must be related to all supersets of X. The *dual* R^{d} of a multirelation R is given by $(x, X) \in R^{\mathsf{d}} \Leftrightarrow (x, A \setminus X) \notin R$. A generalisation to heterogeneous multirelations, that is, subsets of $A \times 2^B$ for sets A and B can be done similarly to the relational case.

A multirelation models a computation as follows; see also [101]. Consider a state $x \in A$ and the set of subsets Xs it is mapped to. The outer set structure of Xs represents an angelic choice: the 'angel' chooses a set $X \in Xs$. The inner set structure of Xs represents a demonic choice: the 'demon' subsequently chooses an element $x' \in X$ which is the next state. The choices are angelic and demonic as regards finite executions.

For example, consider the multirelation S_5 for $A = \{0, 1, 2, 3, 4\}$ given by

$$\begin{array}{lcl} 0 & \mapsto & \{\{1,2\},\{1,3,4\}\} \\ 1 & \mapsto & \{\{1\},\{2\}\} \\ 2 & \mapsto & \{\{1,2\}\} \\ 3 & \mapsto & \{\emptyset\} \\ 4 & \mapsto & \emptyset \end{array}$$

It describes the following computation. In state 0 an angelic choice between two sets $\{1, 2\}$ and $\{1, 3, 4\}$ is made. If the angel chooses $\{1, 3, 4\}$ the demon chooses which of 1, 3 and 4 is the next state. If the angel chooses $\{1, 2\}$ the demon chooses one of 1 and 2 as the next state. State 1 has a purely angelic choice between states 1 and 2, because each inner set is a singleton set in which the demon's choice is fixed. State 2 has a purely demonic choice between states 1 and 2, because the outer set is a singleton set in which the angel's choice is fixed. In state 3 the computation fails to progress since the demon cannot choose from the empty set. In the game interpretation this means that the angel wins; in terms of refinement this means that any specification is satisfied. In state 4 the computation fails to progress since the angel cannot choose from the empty set. In the game interpretation this means that the demon wins; in terms of refinement this means that no specification is satisfied.

The multirelation S_5 is not up-closed, but can be extended to an up-closed multirelation S_6 by adding the required supersets as shown in

$$\begin{array}{lcl} 0 & \mapsto & \{1,2\}^{\uparrow} \cup \{1,3,4\}^{\uparrow} \\ 1 & \mapsto & \{1\}^{\uparrow} \cup \{2\}^{\uparrow} \\ 2 & \mapsto & \{1,2\}^{\uparrow} \\ 3 & \mapsto & 2^A \\ 4 & \mapsto & \emptyset \end{array}$$

using the upward closure $X^{\uparrow} = \{Y \mid X \subseteq Y \subseteq A\}$.

Adding a superset Y of a set X to which x is related does not change the computational interpretation. This just increases the angelic choice by options which are not interesting for the angel because they subsequently allow more choices for the demon. For example, in S_6 the state 0 is related to $\{1,2,3\}$ as a result of the upward closure, but angelic choice prefers $\{1,2\}$ so as to restrict demonic choice as much as possible.

A consequence of using up-closed multirelations is a nice interplay between the outer and the inner set structures. Forming the union of up-closed multirelations simultaneously increases angelic choice and decreases demonic choice, while intersection simultaneously decreases angelic choice and increases demonic choice. Union and intersection thus provide angelic and demonic choice, respectively, at the level of computations. Moreover, these operations are duals of each other. Up-closed multirelations form a bounded distributive lattice using union and intersection as the lattice operations, the empty multirelation O as the least element and the universal multirelation T as the greatest. In general, they do not form a Boolean algebra. See Section 3.1.1 for definitions of these structures.

Consider multirelations $R \subseteq A \times 2^A$ and $S \subseteq A \times 2^A$. Their sequential composition $R\,;\,S$ contains (x,Z) if and only if there is a set $Y \subseteq A$ such that $(x,Y) \in R$ and $(y,Z) \in S$ for each $y \in Y$ [95, 100]. This involves both existential and universal quantification signifying the angelic and demonic choices that take place. Up-closed multirelations form a monoid using sequential composition and the set membership multirelation $\mathsf{E} = \{(x,X) \mid x \in X\}$ as identity. Both the empty multirelation and the universal multirelation are left zeros of sequential composition, but neither is a right zero. Up-closed multirelations satisfy the following distributivity equalities and inequalities:

$$(R \cup S)\,;\,T = (R\,;\,T) \cup (S\,;\,T) \qquad (R\,;\,S) \cup (R\,;\,T) \subseteq R\,;\,(S \cup T)$$
$$(R \cap S)\,;\,T = (R\,;\,T) \cap (S\,;\,T) \qquad (R\,;\,S) \cap (R\,;\,T) \supseteq R\,;\,(S \cap T)$$

Sequential composition from the left in general does not distribute over union or intersection. This is in contrast to relational and matrix-based computation models, in which sequential composition from both sides distributes over union. However, union, intersection and sequential composition are $\subseteq$-isotone. Therefore, up-closed multirelations form an idempotent left semiring [86].

To represent sets of states, we generalise vectors, tests and the domain operation to up-closed multirelations. For vectors, we reuse the relational definition: a multirelation R is a vector if each state is related either to all sets of states or to none, algebraically $R = R\,;\,\mathsf{T}$. In the relational case, a test is a subset of the identity relation. Because of the upward closure, we cannot simply take subsets of the set membership multirelation. Instead a multirelation is a test if it is the intersection of a vector with E. This means that a state x is related either to no set or to all sets containing x. As in the case of relations, the vectors form a Boolean algebra, and the tests form a Boolean algebra in which intersection and sequential composition coincide. Our definition of tests corresponds to the assertions of [97], which have a dual notion of assumptions. The difference is that, if an assertion relates a state to no set, the corresponding assumption relates that state to all sets. The domain of a multirelation R is given by the test $d(R) = (R\,;\,\mathsf{T}) \cap \mathsf{E}$. This operation satisfies the axioms for domain in weaker forms of semirings given by [86, 26] and thus a modal diamond operator can be defined as $|R\rangle q = d(R\,;\,q)$.

Because sequential composition of multirelations already involves universal quantification, correctness statements are formalised differently from relational models. Given tests p and q and an up-closed multirelation R, the Hoare triple $p\{R\}q$ is equivalent to $p \subseteq (R\,;\,q\,;\,\mathsf{T}) \cap \mathsf{E}$ if we translate the definition given in [97]. Using the domain operation, this is $p \subseteq d(R\,;\,q)$ or equivalently $p \leq |R\rangle q$ where $\leq$ is the refinement order $\subseteq$ of multirelations. Taken point-wise, this amounts to 'angelic correctness' of [80], whose dual 'demonic correctness' is expressed by $p \leq |R]q$. Hence both diamond and box are useful for expressing correctness of multirelations. This should be contrasted with the relational and matrix-based models, where only $p \leq |R]q$ is used and diamond represents the preimage operator.

To unify relational, matrix-based and multirelational computation models, it is therefore helpful to have a precondition operator that instantiates to both the modal diamond and the modal box operators. This is pursued in Chapter 5. In particular, we no longer assume that the precondition operator distributes over intersection in its second argument. Neither $|S_7\rangle(p\ ;\ q) = |S_7\rangle p\ ;\ |S_7\rangle q$ nor $|S_8](p\ ;\ q) = |S_8]p\ ;\ |S_8]q$ holds for the up-closed multirelations S_7, S_8, p, q where the state space is $A = \{0, 1\}$ and

$$S_7 = \begin{pmatrix} 0 & \mapsto & \{0\}^\uparrow \cup \{1\}^\uparrow \\ 1 & \mapsto & \emptyset \end{pmatrix} \qquad p = \begin{pmatrix} 0 & \mapsto & \{0\}^\uparrow \\ 1 & \mapsto & \emptyset \end{pmatrix}$$

$$S_8 = \begin{pmatrix} 0 & \mapsto & \{\{0,1\}\} \\ 1 & \mapsto & \emptyset \end{pmatrix} \qquad q = \begin{pmatrix} 0 & \mapsto & \emptyset \\ 1 & \mapsto & \{1\}^\uparrow \end{pmatrix}$$

Modal box and diamond operators can be interchanged by using the dual of a multirelation with an alternative computational interpretation, where the outer set structure describes demonic choice and the inner set structure describes angelic choice [20]. Then union represents demonic choice and intersection represents angelic choice.

In the multirelational model, iteration can be represented by least or greatest fixpoints. However, the resulting operations fail to satisfy some simulation properties used for unifying relational and matrix-based models. For example, consider the up-closed multirelation S_9 over state space $A = \{0, 1, 2\}$ given by

$$\begin{array}{lcl} 0 & \mapsto & \{\{0,1,2\}\} \\ 1 & \mapsto & 2^A \\ 2 & \mapsto & \emptyset \end{array}$$

Neither the isolation property $S_9^\omega = (S_9^\omega\ ;\ \emptyset) \cup S_9^*$ nor $S_9\ ;\ S_9^* \subseteq S_9^*\ ;\ S_9$ nor $S_9\ ;\ S_9^\omega \subseteq S_9^\omega\ ;\ S_9$ holds, where S_9^* and S_9^ω denote the least and the greatest fixpoint of $\lambda X.(S_9\ ;\ X) \cup \mathsf{E}$, respectively.

To summarise, we have the following differences between model M13 and models M1–M12, which entail considerable generalisations of the algebraic structures:

* sequential composition no longer distributes from the left over union;
* preconditions no longer come just as modal box operators, but also as modal diamond operators;
* preconditions no longer distribute over intersection in their test argument;
* iteration no longer satisfies certain simulation properties.

2.6 Overview

The presented relational, matrix-based and multirelational models exemplify the range of computation models that we wish to treat uniformly. They have

* different fixpoints to describe iteration, which satisfy different properties: this is addressed in Chapter 3;
* different approximation orders and therefore different fixpoints to describe recursion: this is addressed in Chapter 4;
* different kinds of correctness statement: this is addressed in Chapter 5.

We use algebraic methods to achieve a unifying treatment.

2.7 Publications

Extended designs were introduced in [61]; changes to this model resulting in the matrix-based model M4 and its approximation order were proposed in [49]. Model M5 was proposed in [45]; the various kinds of correctness statement were described in [47]. Model M8 was proposed in [51].

Models M9–M12 were investigated in [48, 53]. They are based on non-strict computations proposed in [35, 37, 36, 40, 39].

A relation-algebraic investigation of model M13 and an extension that represents infinite executions independently of finite executions appeared in [52].

Chapter 3

Iteration

The computation models described in Chapter 2 use different approximation orders to define the semantics of recursion. We will look at this in more detail in Chapter 4, but here we note that they also use different kinds of iteration, which is a special case of recursion for the repeated sequential execution of a computation. To illustrate this, consider the while-loop $X = \mathsf{while}\ P\ \mathsf{do}\ R$ with condition P and body R. By unrolling the loop we obtain the equality

$$\mathsf{while}\ P\ \mathsf{do}\ R = \mathsf{if}\ P\ \mathsf{then}\ (R\ ;\ \mathsf{while}\ P\ \mathsf{do}\ R)\ \mathsf{else}\ \mathsf{skip}$$

where skip is the computation that does not change the state. Therefore

$$X = \mathsf{if}\ P\ \mathsf{then}\ (R\ ;\ X)\ \mathsf{else}\ \mathsf{skip}$$

and hence $X = f(X)$ where $f(X) = \mathsf{if}\ P\ \mathsf{then}\ (R\ ;\ X)\ \mathsf{else}\ \mathsf{skip}$. We will see in Section 3.2.4 that the conditional statement in the definition of f can be represented by $(P \cdot R \cdot X) + P'$ in a semiring, where $\cdot$ represents sequential composition, $+$ represents non-deterministic choice and P' is the complement of the test P. Abstracting further from the constants in this term, we are interested in the affine function $f(X) = (Y \cdot X) + Z$ for constants Y and Z. Because the while-loop satisfies $X = f(X)$, its semantics is a fixpoint of this function.

Different computation models use different fixpoints of this affine function. In particular, model M1 uses the least fixpoint in the refinement order, models M2 and M9–M12 use the greatest fixpoint, and for model M13 we are interested in both extremal fixpoints. Models M3–M8 use neither the least nor the greatest fixpoint in the refinement order, but the least fixpoint in a dedicated approximation order. Thus, the semantics of iteration differs greatly between the models.

The purpose of this chapter is to develop a unifying theory of iteration. To this end we propose algebraic structures which are general enough to cover many computation models and expressive enough to yield useful results. A benefit of this algebraic approach is that any result once proved in a general algebra holds in many models without the need of separate proofs. This includes complex separation, refinement and program transformation theorems along with a plethora of results useful for their development. Because first-order axioms are used for the algebras, they are well supported by automated theorem provers. We have implemented our results in Isabelle/HOL [93] and make heavy use of its integrated automatic theorem provers and SMT solvers [96, 12]. Benefits of this approach are further detailed in [58]. Proofs that appear in the Isabelle theories [54] are omitted in the following.

Section 3.1 gives basic algebraic structures common to models M1–M12. In particular, Kleene algebras and omega algebras provide unfold and induction axioms for the least and greatest fixpoints of affine functions. Section 3.2 proposes an algebra for iterations which applies to models M1–M8 of strict computations. The key idea is to replace induction axioms, which yield extremal fixpoints, by simulation axioms which apply to other fixpoints as well. Section 3.3 generalises this to an algebra which covers iteration in models M1–M12,

including the models M9–M12 of non-strict computations. The key idea here is to generalise the unary iteration operation to a binary operation by appending a continuation. Section 3.4 further generalises to cover models M1–M13, including the multirelational model M13. The key idea to cover this model is to abandon the simulation axioms and rely only on the equational properties of iteration.

We do not instantiate the algebraic structures to the computation models here, because the instances follow from general results we establish in Chapter 4. Instances will be given in Section 4.3.

3.1 Basic algebras

In this section we axiomatise the operations of non-deterministic choice, conjunction and sequential composition and two kinds of iteration featured by many computation models. To this end we use lattices, semirings, Kleene algebras and omega algebras [11, 63, 76, 21].

3.1.1 Monoids, semilattices, lattices and Boolean algebras

A *monoid* is an algebraic structure $(S, +, 0)$ satisfying the axioms

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

Hence $+$ is an associative operation with identity 0.

A *bounded join-semilattice* is a commutative idempotent monoid, that is, a monoid $(S, +, 0)$ satisfying the axioms

$$\begin{aligned} x + y &= y + x \\ x + x &= x \end{aligned}$$

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

A *bounded distributive lattice* $(S, +, \curlywedge, 0, \top)$ adds to a bounded join-semilattice $(S, +, 0)$ 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. The operations $+$ and $\curlywedge$ have the same precedence. A bounded distributive lattice S is *complete* if every subset of S has a least upper bound.

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

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

Formal languages and relations are two examples of Boolean algebras. More generally, the powerset 2^A of any set A forms a Boolean algebra $(2^A, \cup, \cap, {}^{-}, \emptyset, A)$.

3.1.2 Semirings

An *idempotent left semiring* $(S, +, \cdot, 0, 1)$ combines a bounded join-semilattice $(S, +, 0)$ and a monoid $(S, \cdot, 1)$ with the following distribution and left-zero axioms [86]:

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

The operation $\cdot$ is $\leq$-isotone and distributes over $+$ from the right. The precedence of $\cdot$ is higher than that of $+$. We often abbreviate $x \cdot y$ as xy.

An *idempotent left-zero semiring* is an idempotent left semiring $(S, +, \cdot, 0, 1)$ satisfying the left distribution axiom

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$

In particular, $x \cdot 0 = 0$ is not an axiom; the right-zero property is omitted as it does not hold in many computation models. Thus idempotent left-zero semirings are rings with a left zero in which $+$ is idempotent instead of having an inverse.

An *idempotent semiring* is an idempotent left-zero semiring $(S, +, \cdot, 0, 1)$ satisfying the right-zero axiom

$$x \cdot 0 = 0$$

An idempotent (left/left-zero) semiring is *bounded* if it has a $\leq$-greatest element $\top$ satisfying $x + \top = \top$. A *vector* is an element x such that $x \cdot \top = x$.

Formal languages and relations are two examples of idempotent semirings. When they are used as computation models the operation $+$ represents non-deterministic choice, the operation $\cdot$ sequential composition, 0 the computation with no executions, 1 the computation which does not change the state, $\top$ the computation with all executions, and $\leq$ the refinement relation.

3.1.3 Fixpoints, Kleene algebras and omega algebras

Let S be a set 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:

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

The existence of μf and νf may depend on additional properties of f or S from which this definition abstracts as in [25]. Nevertheless a fixpoint calculus including diagonal, exchange, fusion, rolling and square rules in the style of [1] can be developed. We abbreviate $\mu(\lambda x.f(x))$ by $\mu x.f(x)$ and $\nu(\lambda x.f(x))$ by $\nu x.f(x)$.

Some algebraic structures use axioms that correspond to the $\leq$-least prefixpoint $\hat{\mu} f$ or the $\leq$-greatest postfixpoint $\hat{\nu} f$:

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

If f is $\leq$-isotone and $\hat{\mu} f$ exists, then μf exists and $\hat{\mu} f = \mu f$ [25, Theorem 4.2]. A similar remark holds for $\hat{\nu} f$ and νf.

In computation models, fixpoints are used to solve recursions which are characterised by equations of the form $x = f(x)$. In this equation, $f(x)$ represents the body of the recursive program; occurrences of x in the body correspond to recursive calls of the program x being defined. The semantics of the recursion $x = f(x)$ is the least fixpoint of the function f in a suitable order. Several models of Chapter 2 use the semilattice order or its converse, whence the semantics of recursion is given by μf or νf. Some models require different fixpoints, which will be introduced as least fixpoints in a dedicated approximation order in Section 4.2.2.

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 the $\leq$-least and $\leq$-greatest fixpoints of affine functions.

A *left Kleene algebra* $(S, +, \cdot, ^*, 0, 1)$ adds to an idempotent left semiring $(S, +, \cdot, 0, 1)$ an operation * with the following left unfold and left induction axioms [76, 86]:

$$1 + yy^* \leq y^* \qquad z + yx \leq x \Rightarrow y^*z \leq x$$

It follows that $y^*z = \mu x.yx + z$. The operation * is $\leq$-isotone. The precedence of unary operations such as * is higher than that of binary operations.

A *left-zero Kleene algebra* $(S, +, \cdot, ^*, 0, 1)$ is a left Kleene algebra $(S, +, \cdot, ^*, 0, 1)$ and an idempotent left-zero semiring $(S, +, \cdot, 0, 1)$ satisfying the following right unfold and right induction axioms [76]:

$$1 + y^*y \leq y^* \qquad\qquad z + xy \leq x \Rightarrow zy^* \leq x$$

It follows that $zy^* = \mu x.xy + z$.

A *Kleene algebra* $(S, +, \cdot, ^*, 0, 1)$ is a left-zero Kleene algebra with a right zero, that is, one whose reduct $(S, +, \cdot, 0, 1)$ is an idempotent semiring. A (left/left-zero) Kleene algebra is *bounded* if it has a $\leq$-greatest element $\top$.

A *(left/left-zero) omega algebra* $(S, +, \cdot, ^*, ^\omega, 0, 1)$ further adds to a (left/left-zero) Kleene algebra $(S, +, \cdot, ^*, 0, 1)$ an operation $^\omega$ with the following unfold and induction axioms [21, 86]:

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

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

In some computation models the operation * represents finite iteration and the operation $^\omega$ infinite iteration. For relations, the operation * is the reflexive transitive closure and $^\omega$ yields a vector that represents the states from which infinite transition sequences exist.

3.2 Iteration for strict computations

In this section we expand semirings by an operation that describes iteration in computation models M1–M8 of Chapter 2. The models differ in their treatment of finite, infinite and aborting executions, covering partial, total and general correctness and extensions thereof. Our axioms are general enough to capture the semantics of while-loops in all of these models, yet powerful enough to derive complex results including program transformations and refinement theorems. Besides finding the right balance, the difficulty of giving suitable axioms comes from the fact that in different models iteration is captured by either $\leq$-least fixpoints, $\leq$-greatest fixpoints or various combinations of the two. We therefore cannot use the induction axioms of Kleene algebras and omega algebras, but replace them by simulation properties.

3.2.1 Iterings

The equational properties of the fixpoint operation, hence in particular iterations, are thoroughly investigated in [14]. The authors define *Conway semirings* which are semirings expanded by an operation * satisfying the sumstar axiom $(x + y)^* = (x^*y)^*x^*$ and the productstar axiom $(xy)^* = 1 + x(yx)^*y$ of Conway [22]. The latter is equivalent to the conjunction of sliding $x(yx)^* = (xy)^*x$ and either the left unfold equality $1 + xx^* = x^*$ or the right unfold equality $1 + x^*x = x^*$.

Because we aim for models which vary in the fixpoints used for iteration, we cannot settle for the $\leq$-least fixpoint or any other particular fixpoint. This rules out the use of the induction axioms of Kleene algebras. But we can take from that setting Conway's suggestion of using simulation axioms instead [22]. Our new iteration generalises the Kleene star * and is denoted by $^\circ$ to avoid confusion.

A well-known simulation property in Kleene algebras is $zx \leq yz \Rightarrow zx^\circ \leq y^\circ z$. It arises by setting $w = 0$ in the 'iteration theorem' $zx \leq yz + w \Rightarrow zx^\circ \leq y^\circ(z + wx^\circ)$ of omega algebras [21]. We further generalise this by weakening the antecedent to $zx \leq yy^\circ z + w$. The resulting, first simulation axiom is $zx \leq yy^\circ z + w \Rightarrow zx^\circ \leq y^\circ(z + wx^\circ)$.

The dual simulation property $xz \leq zy \Rightarrow x^\circ z \leq zy^\circ$ holds in Kleene algebras, but it implies $1^\circ 0 \leq 0$ which fails in other target models. First of all, we therefore weaken its

consequent to $x^\circ z \leq zy^\circ + x^\circ 0$. Two generalisations make the outcome nearly symmetric to the first axiom; the resulting, second simulation axiom is $xz \leq zy^\circ + w \Rightarrow x^\circ z \leq (z + x^\circ w)y^\circ$. The symmetric antecedent $zx \leq y^\circ z + w$ cannot be used in the first axiom because this would imply $1^\circ \leq 0^\circ$ which again fails in some target models.

Additionally to these two simulation axioms we adopt the sumstar and productstar equations. Thus an *itering* $(S, +, \cdot, {}^\circ, 0, 1)$ is an idempotent left-zero semiring $(S, +, \cdot, 0, 1)$ expanded by an operation $^\circ$ satisfying the four axioms

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

Derived properties of the operation $^\circ$ are shown in the following result. This collection and subsequent ones form a reference to guide the axiomatisation and facilitate program reasoning as in Sections 3.2.2 and 3.2.4.

Theorem 1. *Let S be an itering and $x, y, z \in S$. Then $^\circ$ is $\leq$-isotone and*

1. $0^\circ = 1$
2. $(x0)^\circ = 1 + x0$
3. $1 \leq x^\circ$
4. $x \leq x^\circ$
5. $x \leq xx^\circ$
6. $xx^\circ \leq x^\circ$
7. $xx^\circ = x^\circ x$
8. $x^\circ = 1 + xx^\circ$
9. $x^\circ = 1 + x^\circ x$
10. $x^\circ = (x^\circ x)^\circ$
11. $x^\circ = x^\circ x^\circ$
12. $x^\circ = 1 + x + x^\circ x^\circ$
13. $x^\circ \leq x^{\circ\circ}$
14. $x^{\circ\circ} = x^\circ 1^\circ$
15. $x^{\circ\circ} = 1^\circ x^\circ$
16. $x^{\circ\circ} = (1 + x)^\circ$
17. $x^{\circ\circ} = x^{\circ\circ\circ}$
18. $1^\circ = 1^{\circ\circ}$
19. $(xy)^\circ \leq (x + y)^\circ$
20. $x^\circ y^\circ \leq (x + y)^\circ$
21. $(x + y)^\circ \leq (x^\circ y^\circ)^\circ$
22. $(x^\circ y^\circ)^\circ = (y^\circ x^\circ)^\circ$
23. $(x^\circ y^\circ)^\circ = x^\circ(y^\circ x^\circ)^\circ$
24. $(yx^\circ)^\circ = y^\circ + y^\circ yxx^\circ(yx^\circ)^\circ$
25. $(yx^\circ)^\circ = (yy^\circ x^\circ)^\circ$
26. $x(yx)^\circ = (xy)^\circ x$

Moreover $y^\circ z$ is a fixpoint of $\lambda x.yx + z$ and zy° is a fixpoint of $\lambda x.xy + z$. Finally, if S has a $\leq$-greatest element $\top$, it satisfies $\top^\circ = \top$.

Counterexamples generated by Isabelle's Nitpick [13] witness that none of the inequalities in Theorem 1 can be strengthened to an equation and that the induction axioms of Kleene algebras do not follow. In particular, $0^\circ = 1 \neq 1^\circ = 0^{\circ\circ}$ in some target models.

The following result gives instances of iterings which cover several models of strict computations. Instance 1 covers iteration in model M1, instances 2–4 cover model M2, instance 5 covers model M4, instance 6 covers models M3–M5 and instances 7–8 cover models M1–M8. The element L represents the endless loop.

Theorem 2. *Iterings have the following instances:*

1. *Every left-zero Kleene algebra is an itering using $x^\circ = x^*$.*
2. *Every left-zero omega algebra is an itering using $x^\circ = x^\omega 0 + x^*$.*
3. *Every left-zero omega algebra with $\top x = \top$ is an itering using $x^\circ = x^\omega + x^*$.*

4. *Every demonic refinement algebra [108] is an itering using* $x^\circ = x^\omega$.

5. *Extended designs [61, 49] form an itering using* $x^\circ = d(x^\omega)\mathsf{L} + x^*$. *See Section 5.2.1 for the axioms of the domain operation* d.

6. *Every n-semiring that is a left-zero omega algebra is an itering using* $x^\circ = n(x^\omega)\mathsf{L}+x^*$. *See Section 4.1.1 for the axioms of n-semirings.*

7. *Every n-omega algebra with* $\mathsf{L}x = \mathsf{L}$ *is an itering using* $x^\circ = n(x^\omega)\mathsf{L} + x^*$. *See Section 4.2.4 for the axioms of n-omega algebras.*

8. *Every binary itering with* $(x \star y)z = x \star (yz)$ *is an itering using* $x^\circ = x \star 1$. *See Section 3.3.1 for the axioms of binary iterings.*

3.2.2 Applications: separation and refinement

Various results which have been used for program development in specific structures can be generalised to iterings and further improved by making weaker assumptions. The following result generalises two separation theorems of omega algebras [21]. In particular, the assumption is satisfied if $yx \leq xy$. The weaker bounds xy° and $xx^\circ(1 + y)$ include infinite iterations in some models.

Theorem 3. *Let* S *be an itering and* $x, y \in S$ *such that* $yx \leq xy^\circ$ *or* $yx \leq xx^\circ(1+y)$. *Then* $y^\circ x^\circ \leq x^\circ y^\circ = (x + y)^\circ$.

The next result is Back's atomicity refinement theorem. Our formulation is adapted from [108]. The continuity assumption of [6] is expressed by $r^\circ q \leq qr^\circ$ in iterings.

Theorem 4. *Let* S *be an itering and* $a, b, l, r, q, s \in S$ *such that*

$$\begin{array}{lllll} s = sq & rb \leq br & rl \leq lr & bl \leq lb & r^\circ q \leq qr^\circ \\ a = qa & qb = 0 & al \leq la & ql \leq lq & q \leq 1 \end{array}$$

Then $s(a + b + r + l)^\circ q \leq s(ab^\circ q + r + l)^\circ$.

3.2.3 Tests

For defining the semantics of while-programs, we need to represent conditions. In semirings this can be done by tests, which are elements ≤ 1. For an arbitrary set S, we introduce tests by means of two operations $\cdot$ and $'$ with axioms making the image $S' = \{x' \mid x \in S\}$ a Boolean algebra with greatest lower bound $\cdot$ and complement $'$ [57, 47, 45]. The effect of the operations $\cdot$ and $'$ on elements of $S \setminus S'$ is of no concern.

Any axiomatisation of Boolean algebras can be applied to S'. For concision we use Huntington's axioms [73], which lead to the following definition. A *test algebra* is a structure $(S, \cdot, ')$ satisfying the axioms

$$\begin{array}{ll} x'(y'z') = (x'y')z' & x' = (x''y')'(x''y'')' \\ x'y' = y'x' & x'y' = (x'y')'' \end{array}$$

The last axiom states that S' is closed under the operation $\cdot$ and the remaining ones are associativity, commutativity and Huntington's special axiom. Then $(S', +, \cdot, ', 0, 1)$ is a Boolean algebra with the order $x' \leq y' \Leftrightarrow x'y' = x'$, least upper bound $x' + y' = (x''y'')'$, greatest lower bound $\cdot$, complement $'$, least element $0 = x'x''$ for any x, and greatest element $1 = 0'$. The extension $(S, +, \cdot, ', 0, 1)$ thus obtained is also called a test algebra; elements of S' are *tests*. The equation $x = x''$ holds if and only if x is a test.

A benefit of this axiomatisation is that it avoids introducing a separate sort for Boolean elements as, for example, in Kleene algebras with tests [77] without imposing additional constraints as, for example, in antidomain semirings [27].

For representing conditional statements and while-loops it is necessary to take the complement of tests. We do not make the whole set S a Boolean algebra because some computation models are not closed under complements, in particular models M2, M4, M10, M12 and M13 of Chapter 2.

A *test itering* is an algebraic structure $(S, +, \cdot, {}^\circ, {}', 0, 1)$ whose reduct $(S, +, \cdot, {}^\circ, 0, 1)$ is an itering and whose reduct $(S, +, \cdot, {}', 0, 1)$ is a test algebra. The following result shows that tests which are preserved by an element x are preserved by and can be imported into iterations of x. This propagates information about the current state which can be used for further simplifications. The assumption $px \leq xp$ is equivalent to $px = pxp$ if p is a test; recall that px restricts the executions of x to those that satisfy the condition p.

Theorem 5. *Let S be a test itering and let $x \in S$ and $p \in S'$ such that $px \leq xp$. Then $px^\circ = px^\circ p = p(px)^\circ$.*

3.2.4 Applications: transformation of while-programs

In test iterings we define the semantics of while-programs by generalising from Kleene algebras with tests [77]:

$$\begin{aligned} x \,;\, y &= xy \\ \mathsf{if}\ p\ \mathsf{then}\ x\ \mathsf{else}\ y &= px + p'y \\ \mathsf{if}\ p\ \mathsf{then}\ x &= px + p' \\ \mathsf{while}\ p\ \mathsf{do}\ x &= (px)^\circ p' \end{aligned}$$

A while-program is in *normal form* if it has the form x ; $\mathsf{while}\ p\ \mathsf{do}\ y$ with while-free x and y. An element x *preserves* the test p if both $px \leq xp$ and $p'x \leq xp'$ hold. An element x *assigns p to q* if $x = x(pq + p'q')$ holds.

The following result generalises a split/merge loop theorem of Back and von Wright [6] to test iterings. While the original proof takes two pages of calculation, the Isabelle proof boils down to only two calls to the SMT solver Z3.

Theorem 6. *Let S be a test itering and let $x, y \in S$ and $p, q \in S'$ such that $p'y \leq yp'$. Then* $\mathsf{while}\ p + q\ \mathsf{do}\ (\mathsf{if}\ p\ \mathsf{then}\ x\ \mathsf{else}\ y) = (\mathsf{while}\ p\ \mathsf{do}\ x)\ ;\ (\mathsf{while}\ q\ \mathsf{do}\ y)$.

The next result is Kozen's algebraic version of the while-program normal form theorem [77]. The original proof uses Kleene algebras and works in the partial-correctness setting of model M1. It was adapted for total-correctness settings such as model M2 in [105]; see also the extension to probabilistic demonic refinement algebras in [99]. Our generalisation to iterings uniformly applies to computation models M1–M8. The proof uses the following program transformations to move while-programs out of each kind of program construct and hence into normal form.

Theorem 7. *Let S be a test itering and let $x_1, x_2, y_1, y_2, z_1, z_2 \in S$ and $p, q, r_1, r_2 \in S'$.*

1. *Let z_1 assign p to q and let x_1, x_2, y_1, y_2 preserve q. Then*

$$\begin{aligned} & z_1 \,;\, \mathsf{if}\ p\ \mathsf{then}\ (x_1 \,;\, \mathsf{while}\ r_1\ \mathsf{do}\ y_1)\ \mathsf{else}\ (x_2 \,;\, \mathsf{while}\ r_2\ \mathsf{do}\ y_2) \\ = {} & z_1 \,;\, (\mathsf{if}\ q\ \mathsf{then}\ x_1\ \mathsf{else}\ x_2) \,;\, \mathsf{while}\ qr_1 + q'r_2\ \mathsf{do}\ (\mathsf{if}\ q\ \mathsf{then}\ y_1\ \mathsf{else}\ y_2) \end{aligned}$$

2. *Let z_1 assign p to q and let x_1, y_1 preserve q. Then*

$$\begin{aligned} & z_1 \,;\, \mathsf{while}\ p\ \mathsf{do}\ (x_1 \,;\, \mathsf{while}\ r_1\ \mathsf{do}\ y_1) \\ = {} & z_1 \,;\, (\mathsf{if}\ q\ \mathsf{then}\ x_1) \,;\, \mathsf{while}\ q(p + r_1)\ \mathsf{do}\ (\mathsf{if}\ r_1\ \mathsf{then}\ y_1\ \mathsf{else}\ x_1) \end{aligned}$$

3. *Let z_1 assign r_1 to q and let z_2 assign q to p and let $z_1 z_2 = z_2 z_1$. Let x_2, y_2, z_2 preserve q and let y_1, z_1, x_2, y_2 preserve p. Then*

$$\begin{aligned} & x_1 \,;\, z_1 \,;\, z_2 \,;\, (\mathsf{while}\ r_1\ \mathsf{do}\ y_1 \,;\, z_1) \,;\, x_2 \,;\, (\mathsf{while}\ r_2\ \mathsf{do}\ y_2) \\ = {} & x_1 \,;\, z_1 \,;\, z_2 \,;\, (\mathsf{if}\ q\ \mathsf{then}\ (y_1 \,;\, z_1 \,;\, \mathsf{if}\ q'\ \mathsf{then}\ x_2)\ \mathsf{else}\ x_2) \,; \\ & \mathsf{while}\ q + r_2\ \mathsf{do}\ (\mathsf{if}\ q\ \mathsf{then}\ (y_1 \,;\, z_1 \,;\, \mathsf{if}\ q'\ \mathsf{then}\ x_2)\ \mathsf{else}\ y_2) \end{aligned}$$

By repeatedly applying these program transformations it can be shown that every while-program, suitably augmented with assigning elements, is equivalent to a while-program in normal form under certain preservation assumptions.

The transformations explicitly state the source and target programs, the positions where assigning elements are inserted in them and the preservation assumptions. Also included is the commutativity assumption $z_1 z_2 = z_2 z_1$ for the assigning elements, which is not obvious in previous proofs.

3.3 Iteration for strict and non-strict computations

In this section, we extend the algebraic treatment of iterations to non-strict computations exemplified by the models M9–M12 of Section 2.4. Because they are also based on relations, we can reuse fundamental algebras to develop a common theory of strict and non-strict computations, which covers models M1–M12.

Due to non-strictness, the iteration underlying loops cannot be described by a unary operation [48]. To see this, consider the semantics of the endless loop while true do skip in iterings, which is $1^\circ 0$. In models M9–M12 of non-strict computations 0 is a right zero of composition, so $1^\circ 0 = 0$. But in the same models, recursion is solved by $\leq$-greatest fixpoints, so the semantics of the endless loop is the $\leq$-greatest fixpoint of the identity function, which is $\top$. This would lead to the contradiction $0 = \top$. Hence loops cannot be represented in the form $y^\circ z$ in this model, no matter how the unary operation $^\circ$ is defined.

3.3.1 Binary iterings

We therefore propose a binary operation which adds a continuation to the unary itering operation. Our axioms generalise the binary iteration operation $y \star z = y^\omega + y^* z$ of omega algebras [21]. This operation cannot be used without change as it does not describe iteration in several models of strict computations. After introducing the new operation, we derive properties of it which hold for both strict and non-strict computations.

A *binary itering* $(S, +, \cdot, \star, 0, 1)$ is an idempotent left-zero semiring $(S, +, \cdot, 0, 1)$ expanded with a binary operation $\star$ satisfying the following axioms:

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

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

The precedence of $\star$ is the same as that of $+$. These axioms generalise the itering axioms by appropriately composing to an iteration of the form y° a continuation z, which results in $y \star z$. 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 [21].

To understand the computational meaning of the two simulation axioms they can be seen as generalising the basic simulation laws

$$\begin{array}{l} zx \leq yz \Rightarrow z(x \star v) \leq y \star (zv) \\ xz \leq zy \Rightarrow x \star (zv) \leq z(y \star v) + (x \star 0) \end{array}$$

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 the induction axioms which characterise the $\leq$-least and $\leq$-greatest fixpoints of linear functions. As in iterings, 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 4.2.2. For model M3 such properties are shown in [43].

In models of strict computations, full associativity $(x \star y)z = x \star (yz)$ holds and the binary itering axioms specialise to the itering axioms by setting $x \star y = x^\circ y$ and setting some continuations to 1. However, full associativity does not hold in models M9–M12 of non-strict computations 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 these models. 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. We give an example in Section 3.3.3.

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. Instances 1 and 5 cover iteration in models M1–M8, instances 2–3 cover models M2 and M9–M12 and instance 4 covers models M1–M12.

Theorem 8. *Binary iterings have the following instances:*

1. *Every itering is an extended binary itering using $x \star y = x^\circ y$.*
2. *Every left-zero omega algebra is a binary itering using $x \star y = x^\omega + x^*y$.*
3. *Every left-zero omega algebra with the additional axiom $x \leq x\top x\top$ is an extended binary itering using $x \star y = x^\omega + x^*y$.*
4. *Every n-omega algebra is an extended binary itering using $x \star y = n(x^\omega)\mathsf{L} + x^*y$. See Section 4.2.4 for the axioms of n-omega algebras.*
5. *Every binary itering with $(x \star y)z = x \star (yz)$ is an extended binary itering.*

The property $x \leq x\top x\top$ can equivalently be stated in each of the following forms in left-zero 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 computation models M1–M12.

Theorem 9. *Let S be a binary itering and $w, x, y, z \in S$. Then the following properties 1–48 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 algebras [76].

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. $x \leq z \star 1 \wedge y \leq z \star w \Rightarrow xy \leq z \star w$

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.

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 Theorem 9 appear in [21]. In extended binary iterings, and therefore in computation models M1–M12, we can add the following properties.

Theorem 10. *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 algebras.

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 [81] witness.

3.3.2 Applications: separation, transformation and refinement

We show how the separation and refinement theorems of Section 3.2.2 generalise to binary iterings.

Theorem 11. *Let S be a binary itering and $p, x, y, z \in S$. Then the following properties hold.*

1. $yx \leq x \Rightarrow y \star x \leq x + (y \star 0)$
2. $yx \leq xy \Rightarrow (xy) \star z \leq x \star (y \star z)$
3. $yx \leq xy \Rightarrow y \star (x \star z) \leq x \star (y \star z)$
4. $yx \leq xy \Rightarrow (x + y) \star z = x \star (y \star z)$

Properties 3 and 4 correspond to basic simulation and separation laws of omega algebras.

5. $yx \leq x(y \star 1) \Rightarrow y \star (x \star z) \leq x \star (y \star z) = (x + y) \star z$
6. $yx \leq x(x \star (1 + y)) \Rightarrow y \star (x \star z) \leq x \star (y \star z) = (x + y) \star z$

Properties 5 and 6 sharpen the simulation and separation laws and generalise theorems of [21].

7. $y(x \star 1) \leq x \star (y \star 1) \Leftrightarrow y \star (x \star 1) \leq x \star (y \star 1)$
8. $p \leq pp \wedge p \leq 1 \wedge px \leq xp \Rightarrow p(x \star y) = p((px) \star y) = p(x \star (py))$

Property 8 can be used to import and preserve tests in iterations, which is useful for program transformations; see also Theorem 5.

The next result applies Theorems 9 and 10 to derive Back's atomicity refinement theorem [6, 108]. Because we generalise it to extended binary iterings, it is valid in models M1–M12 of strict and non-strict and computations. Whether it holds in binary iterings is unknown.

Theorem 12. *Let S be an extended binary itering and $b, l, q, r, s, x, z \in S$ such that*

$s = sq$	$rb \leq br$	$rl \leq lr$	$bl \leq lb$	$r \star q \leq q(r \star 1)$
$x = qx$	$qb = 0$	$xl \leq lx$	$ql \leq lq$	$q \leq 1$

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

3.3.3 Specific properties

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 13. *The following properties 1–6 hold in the instance of Theorem 8.1 – extended by $\top$ for the last two – but not in the instances of Theorems 8.2 and 8.3.*

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 instances of Theorems 8.2 and 8.3, but not in the instance of Theorem 8.1 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 instance of Theorem 8.3, but neither in the instance of Theorem 8.2 nor in the instance of Theorem 8.1 extended by $\top$.

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

In particular, we have the following four variants of the sumstar property, which coincide models M1–M8:

1. $(x + y) \star z = (x \star y) \star (x \star z)$ (binary itering axiom)
2. $(x + y) \star z = ((x \star 1)y) \star (x \star z)$ (Theorem 10.9)
3. $(x + y) \star z = (x \star y) \star ((x \star 1)z)$ (Theorem 10.10)
4. $(x + y) \star z = ((x \star 1)y) \star ((x \star 1)z)$ (Theorem 13.4)

In contrast to the first three, however, the last property does not hold in models M9–M12. This exemplifies the difficulty in generalising from iterings to binary iterings.

3.4 Iteration for multirelational models

In this section we extend the algebraic treatment of iterations to the multirelational computation model M13. Iterings and binary iterings replace the induction axioms of Kleene algebras by weaker simulation axioms so as to cover different relational and matrix-based computation models. However, even some of the simulation properties fail in multirelational models; see the counterexamples in Section 2.5.

A basic algebraic difference between relational and multirelational models is that the latter do not satisfy the left distribution axiom $x(y+z) = xy+xz$. However, composition is $\leq$-isotone in both arguments even for multirelations. We therefore work in idempotent left semirings instead of idempotent left-zero semirings.

Because the simulation properties fail, we rely on the equational properties of iteration [14, 15]. Recall that a Conway semiring is a semiring expanded by an operation * satisfying the sumstar and productstar axioms. In the following we generalise this to reflect the absence of left distribution in idempotent left semirings.

A *left Conway semiring* $(S, +, \cdot, ^\circ, 0, 1)$ is an idempotent left semiring $(S, +, \cdot, 0, 1)$ expanded by an operation $^\circ$ satisfying the axioms

$$\begin{aligned} (x + y)^\circ &= x^\circ(yx^\circ)^\circ \\ 1 + xx^\circ &= x^\circ \\ (xy)^\circ x &\leq x(yx)^\circ \end{aligned}$$

The axioms are the slided version of the sumstar equation, the left unfold equation and one inequality of sliding. The following result shows that many properties of iterings already hold in left Conway semirings.

Theorem 14. *Let S be a left Conway semiring and $x, y, z \in S$. Then $^\circ$ is $\leq$-isotone and*

1. $0^\circ = 1$
2. $(x0)^\circ = 1 + x0$
3. $1 \leq x^\circ$
4. $x \leq x^\circ$
5. $x \leq x^\circ x$
6. $x^\circ x \leq x x^\circ$
7. $x x^\circ \leq x^\circ$
8. $1 + x^\circ x \leq x^\circ$
9. $x^\circ = (x x^\circ)^\circ$
10. $x^\circ = (x^\circ x)^\circ$
11. $x^\circ = x^\circ x^\circ$
12. $x^\circ = 1 + x + x^\circ x^\circ$
13. $x^\circ \leq x^{\circ\circ}$
14. $x^\circ 1^\circ \leq x^{\circ\circ}$
15. $1^\circ x^\circ \leq x^{\circ\circ}$
16. $x^{\circ\circ} = (1 + x)^\circ$
17. $x^{\circ\circ} = x^{\circ\circ\circ}$
18. $1^\circ = 1^{\circ\circ}$
19. $1 + x(yx)^\circ y \leq (xy)^\circ$
20. $(xy)^\circ \leq (x^\circ y)^\circ x^\circ$
21. $(x^\circ y)^\circ x^\circ \leq (x + y)^\circ$
22. $(x + y)^\circ = x^\circ(1 + y(x + y)^\circ)$
23. $x^\circ y^\circ \leq (x + y)^\circ$
24. $(x + y)^\circ \leq (x^\circ y^\circ)^\circ$
25. $(x^\circ y^\circ)^\circ = (y^\circ x^\circ)^\circ$
26. $(x^\circ y^\circ)^\circ = x^\circ(y^\circ x^\circ)^\circ$
27. $(x^\circ y^\circ)^\circ = (x^\circ y^\circ)^\circ x^\circ$
28. $(yx^\circ)^\circ = (yy^\circ x^\circ)^\circ$

Moreover $y^\circ z$ is a fixpoint of $\lambda x.yx + z$ and zy° is a prefixpoint of $\lambda x.xy + z$.

Counterexamples generated by Nitpick witness that none of the following properties follow in left Conway semirings:

$$\begin{array}{rl} x^\circ = 1 + x^\circ x & 1^\circ x^\circ \leq x^\circ 1^\circ = x^{\circ\circ} \\ (xy)^\circ = 1 + x(yx)^\circ y & x^\circ 1^\circ \leq 1^\circ x^\circ = x^{\circ\circ} \\ x(yx)^\circ = (xy)^\circ x & (x + y)^\circ = (x^\circ y)^\circ x^\circ \\ x x^\circ = x^\circ x & (x + y)^\circ = x^\circ + x^\circ y(x + y)^\circ \end{array}$$

These properties hold in models M1–M8, but have to be omitted as we generalise to include model M13. Other properties, such as $1^\circ = 1$, fail already in matrix-based models where $^\circ$ involves infinite iteration.

In addition to being a left Conway semiring, computation models M1–M8 and M13 are left Kleene algebras. The operations * and $^\circ$ may be identical as in model M1 or different as in models M2–M8. Model M13 can be instantiated as a left Conway semiring in several ways; the operation $^\circ$ can be chosen as * but also differently. This will be shown in Section 5.3; moreover Section 5.2.3 extends left Conway semirings by modal operators for correctness reasoning.

In the setting of left Conway semirings we do not look at models M9–M12 of non-strict computations because they do not represent iteration by a unary operation as shown in Section 3.3. Models M1–M13 can be captured by generalising left Conway semirings to a binary operation, but this is not further discussed in the present work.

3.5 Publications

Properties of iteration in computation models describing partial, total and general correctness were investigated in [41, 43, 49]. Iterings as a unifying algebraic structure were proposed in [45]. Instances of iterings were given in [45, 51].

Test algebras were proposed in [57, 47, 45]. Generalisations of Kozen's while-program normal form theorem were given in [41, 45, 49].

Binary iterings for unifying strict and non-strict computations were proposed in [48]. Instances of binary iterings were given in [48, 51, 53].

Left Conway semirings were proposed in [50], where also instances were given.

Chapter 4

Recursion

In Chapter 3 we gave axioms for operations that describe iteration in the various computation models of Chapter 2. In semiring-based structures these operations allow us to form fixpoints of the affine function $\lambda x.yx + z$. Different computation models use different fixpoints of this function; the axioms of our algebras state unifying properties of iteration which hold in several models.

The purpose of the present chapter is to generalise the theory to arbitrary recursions. We therefore look at fixpoints of more general functions than the above affine function. As customary, we describe the particular fixpoint that gives the semantics of recursion as the least fixpoint in a dedicated approximation order $\sqsubseteq$. Different computation models use different approximation orders: the semilattice order $\leq$ for model M1, its converse $\geq$ for models M2 and M9–M12, and variants of the Egli-Milner order for models M3–M8.

To unify these disparate orders we propose algebras with an operation n such that $n(x)$ describes the infinite executions of the computation x. A constant L in these algebras represents the computation with all infinite executions. We then express the approximation order $\sqsubseteq$ in terms of the operation n, the constant L and the semilattice order $\leq$. Our main results – Theorems 17, 27 and 28 – show how to reduce $\sqsubseteq$-least fixpoints to a combination of $\leq$-least and $\leq$-greatest fixpoints. This is helpful because $\leq$ is much simpler than $\sqsubseteq$. In particular, it allows us to represent the semantics of iteration in terms of the Kleene star and omega operations. Moreover, we show that the resulting operation gives rise to a binary itering as defined in Section 3.3.1, so we inherit all properties shown there. The above holds uniformly in computation models M1–M12.

Section 4.1 carries out these ideas in a setting that applies to models M3–M5 and M7. We propose n-semirings, define $\sqsubseteq$ in these structures, calculate $\sqsubseteq$-least fixpoints in terms of $\leq$-least and $\leq$-greatest fixpoints, and instantiate these results to iteration. Moreover, we show how n-semirings give rise to tests and modal operators, and apply these for reasoning about programs. Section 4.2 generalises this approach to cover models M1–M12. We propose n-algebras with weaker axioms and revise the approximation order. Section 4.3 exemplifies how to instantiate n-algebras for the computation models M8, M11 and M12. It also shows how to use the characteristic property of the operation n to derive an approximation order for these models.

4.1 Recursion for strict computations

In this section we axiomatise an operation that captures the infinite executions of a computation in the models M3–M5 and M7 of Chapter 2. Based on the new operation we define a common approximation order for these 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 itering axioms of Section 3.2.1. We also show how the operation induces tests so our results

extend to the test iterings of Section 3.2.3. Based on tests we introduce modal operators with applications in program reasoning.

4.1.1 Axioms for the infinite executions

We first describe the infinite executions of a computation and the endless loop. An *n-semiring* $(S, +, \cdot, n, 0, 1, \mathsf{L}, \top)$ is a bounded idempotent left-zero semiring $(S, +, \cdot, 0, 1, \top)$ expanded by an operation $n : S \to S$ and a constant L satisfying the axioms

$$\begin{array}{lrlr} (i1) & n(0) = 0 & (i5) & n(x) = n(x0)n(x) \\ (i2) & n(\top) = 1 & (i6) & x\,n(y)\mathsf{L} = x0 + n(xy)\mathsf{L} \\ (i3) & n(x+y) = n(x) + n(y) & (i7) & x \leq x0 + n(x\mathsf{L})\top \\ (i4) & n(n(x)y) = n(x)n(y) & & \end{array}$$

The constant L represents the endless loop, that is, the computation that has only infinite executions. The element $n(x)$ describes – as a test – the set of states from which the computation x has infinite executions. Recall that tests are elements ≤ 1 and 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.

The n-semiring axioms hold in models M2–M5 and M7 and have the following rationale. Axioms $(i1)$ and $(i2)$ express that the computation 0 has no infinite executions and that the computation $\top$ has infinite executions starting from each state, respectively. By axiom $(i3)$, the infinite executions of a non-deterministic choice between two computations are given as the union of the individual infinite executions. Axiom $(i4)$ expresses that the infinite executions of a computation y restricted to starting states $n(x)$ are given by intersecting the infinite executions of y with the set $n(x)$. Recall that the operation $\cdot$ implements the greatest lower bound of tests, which is the intersection of the represented sets.

By axioms $(i1)$–$(i4)$ the image $n(S) = \{n(x) \mid x \in S\}$ of the operation n is closed under 0, 1, $+$ and $\cdot$. Moreover n is $\leq$-isotone since it is additive, whence $n(x) \leq 1$ and $n(x)0 = 0$ hold. Therefore $(n(S), +, \cdot, 0, 1)$ is a bounded idempotent semiring.

By adding axiom $(i5)$ we obtain $n(x)n(x) = n(x)$. Hence $(n(S), +, \cdot, 0, 1)$ is a bounded distributive lattice by Theorem II.10 in Birkhoff's book [11]. Moreover $n(1) = 0$ follows.

A consequence of adding axiom $(i6)$ is $n(x)\mathsf{L} \leq x$. The element $n(x)\mathsf{L}$ contains the infinite executions of x; see the characterisation below. By adding axiom $(i7)$ we also obtain $n(\mathsf{L}) = 1$. It follows that there is a 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. Its significance is that $n(y)$ is the greatest test that, sequentially composed with L, is below y. This is the characteristic property of the infinite executions of y. These and further consequences of n-semirings are summarised in the following result.

Theorem 15. *Let S be an n-semiring and $x, y \in S$. Then $(n(S), +, \cdot, 0, 1)$ is a bounded idempotent semiring and a bounded distributive lattice. Moreover n is $\leq$-isotone and*

1. $n(1) = 0$
2. $n(\mathsf{L}) = 1$
3. $n(x) = n(x0)$
4. $n(x\mathsf{L}) = n(x\top)$
5. $x0 + n(x\mathsf{L})\mathsf{L} = x\mathsf{L}$
6. $x\mathsf{L} \leq x0 + \mathsf{L}$
7. $\mathsf{L}x = \mathsf{L}$
8. $n(xy) = n(x\,n(y)\mathsf{L})$
9. $n(x\,n(y)) = n(x)$
10. $n(x) \leq n(y) \Leftrightarrow n(x)\mathsf{L} \leq y$

Property 7 intuitively states that anything which is supposed to happen 'after' an infinite execution is ignored. Axioms $(i6)$ and $(i7)$ are also used to establish that the operation $\cdot$ is isotone with respect to the approximation order we introduce in Section 4.1.2. Counterexamples generated by Nitpick witness that each of the axioms $(i1)$–$(i7)$ is independent of the others and the underlying semiring axioms.

The operation n facilitates two tasks: to access the infinite executions of a computation and to represent tests. For models M2–M4 this can be done by using the domain operation instead [38, 41, 42, 49]. However, domain semirings are not sufficient to describe the computations of models M5 and M7, which have independent aborting, finite and infinite executions. This is because sequential composition, non-deterministic choice and domain cannot distinguish between aborting and infinite executions, but it is necessary to access the infinite executions of a computation to define the approximation order used for recursion. The necessary information happens to be available in model M4 because there aborting executions entail infinite ones. So, while domain could still be used to induce tests, another operation has to be added for the infinite executions. In the following we mostly focus on the infinite executions, but we will come back to tests in Section 4.1.4.

4.1.2 Approximation and recursion

The approximation relation $\sqsubseteq$ on computations, which generalises the Egli-Milner order, is defined as follows:

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

To obtain an intuition for this definition, consider a computation x that has infinite executions starting in every state. Then $n(x) = 1$ and $x \sqsubseteq y$ reduces to $x \leq y + \mathsf{L}$, meaning that y must have at least the aborting and finite executions of x and may have any infinite executions. On the other hand, if x has no infinite executions, then $n(x) = 0$ and $x \sqsubseteq y$ reduces to $x = y$: no executions may be added or removed. The following result shows that the relation $\sqsubseteq$ is an order and that it is preserved by many operations.

Theorem 16. *Let S be an n-semiring.*

1. *The relation $\sqsubseteq$ is a partial order on S with least element L.*
2. *The operations $+$ and $\cdot$ 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 left-zero omega algebra, the operation $^\omega$ is $\sqsubseteq$-isotone.*

The approximation order is suitable for defining the semantics of recursion in models M3–M5 and M7. Hence the results about $\sqsubseteq$ derived below also hold in these models. Different approximation orders are required for partial- and total-correctness models, namely $\leq$ and $\geq$, respectively. A unified treatment of these approximation orders and recursion which covers models M1–M12 is given in Section 4.2.

Let $f : S \to S$ be a function on an n-semiring S. Provided it exists, the $\sqsubseteq$-*least fixpoint* of f is denoted by κf:

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

We abbreviate $\kappa(\lambda x.f(x))$ by $\kappa x.f(x)$. The semantics of the recursion specified by $f(x) = x$ is κf, that is, the least fixpoint of f in the approximation order $\sqsubseteq$.

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

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

The following result gives conditions on the existence of $\sqsubseteq$-least fixpoints and shows how to calculate them.

Theorem 17. *Let S be an n-semiring and let $f : S \to S$ be $\leq$- and $\sqsubseteq$-isotone such 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 $\kappa f = \mu f + n(\nu f)\mathsf{L}$.*
4. *$\nu f \leq \mu f + n(\nu f)\top$.*
5. *$\mu f + n(\nu f)\mathsf{L} \sqsubseteq \nu f$.*
6. *$\mu f \sqcap \nu f$ exists and $\mu f \sqcap \nu f = \mu f + n(\nu f)\mathsf{L}$.*
7. *$\mu f \sqcap \nu f$ exists and $\mu f \sqcap \nu f \leq \nu f$.*

Condition 3 reduces the calculation of κf to that of μf and νf, which are often easier to obtain as the semilattice order $\leq$ is less complex than the approximation order $\sqsubseteq$. It is typically inferred by establishing condition 4 that characterises the existence of κf in terms of μf and νf.

4.1.3 Iteration

We now look at the special case of iteration introduced in Chapter 3. Recall that the while-loop $\mathsf{while}\ p\ \mathsf{do}\ r$ is a solution to its unfolding equation $x = prx + p'$ using the complement p' of the condition p, which is a test. We solve the more general equation $x = yx + z$ by calculating the $\sqsubseteq$-least fixpoint κf of the function $f : S \to S$ given by $f(x) = yx + z$ for constants $y, z \in S$. To instantiate Theorem 17 for while-loops an additional assumption is needed that captures the interaction of * and $^\omega$ with n.

Theorem 18. *Let S be an n-semiring and a left-zero omega algebra with $x^\omega \leq x^* n(x^\omega)\top$ for each $x \in S$. Let $y, z \in S$ and $f(x) = yx + z$. Then $\kappa f = n(y^\omega)\mathsf{L} + y^* z$.*

The proof uses that f is $\leq$- and $\sqsubseteq$-isotone, $\mu f = y^* z$ and $\nu f = y^\omega + y^* z$. It is unknown whether the additional assumption $x^\omega \leq x^* n(x^\omega)\top$ is independent of the axioms of n-semirings and left-zero omega algebras.

Suggested by the previous result we let $y^\circ = n(y^\omega)\mathsf{L} + y^*$ and obtain $\kappa f = y^\circ z$. The operation $^\circ$ so defined satisfies the itering axioms of Section 3.2.1. We thus obtain all consequences of iterings also in the present setting. The additional assumption of Theorem 18 is not required here, just to derive the semantics of iteration as a special case of recursion.

Theorem 19. *Let S be an n-semiring and a left-zero omega algebra. Then S is an itering using $x^\circ = n(x^\omega)\mathsf{L} + x^*$.*

Further consequences about the interaction of * and $^\omega$ with n are summarised in the following result.

Theorem 20. *Let S be an n-semiring and a left-zero omega algebra. Let $x, y, z \in S$. Then*

1. $n(x) \leq n(x^*)$
2. $n(x^*) \leq n(x^\omega)$
3. $n(x^\omega) = n(x^\omega y)$
4. $x^* n(x)\mathsf{L} = x^* 0$
5. $x^* n(x^\omega)\top = x^* 0 + n(x^\omega)\top$
6. $n(x) \leq n(z+yx) \Rightarrow n(x) \leq n(y^\omega + y^* z)$

The last implication is similar to the induction axiom of omega algebras. Whether the dual $n(z + yx) \leq n(x) \Rightarrow n(y^* z) \leq n(x)$ follows is unknown. This is reversed in omega algebras with domain, where $d(z + yx) \leq d(x) \Rightarrow d(y^* z) \leq d(x)$ holds [23] but the dual $d(x) \leq d(z + yx) \Rightarrow d(x) \leq d(y^\omega + y^* z)$ has not been derived; see the additional axiom for the divergence operation [24]. The reason for this asymmetry is the characterisation of $n(x)$ as a greatest test and that of $d(x)$ as a least test satisfying certain properties.

4.1.4 Boolean tests

The axioms of n-semirings induce a set of tests $n(S)$ which is a bounded distributive lattice. To make it a Boolean algebra, so as to inherit our general results about while-programs from Sections 3.2.3 and 3.2.4, we add the Boolean complement $\bar{n}$ of n. This is achieved by introducing the operation $\bar{n} : S \to S$ such that $\bar{n}(x) + n(x) = 1$ and $\bar{n}(x)n(x) = 0$. Hence $\bar{n}(x)$ represents the set of states from which the computation x has no infinite executions.

After adding the complementing axioms to those of n, it is possible to reduce their number, similarly to the case of (anti)domain [27]. Thus an *$\bar{n}$-semiring* $(S, +, \cdot, \bar{n}, n, 0, 1, \mathsf{L}, \top)$ is a bounded idempotent left-zero semiring $(S, +, \cdot, 0, 1, \top)$ expanded by operations $\bar{n}, n : S \to S$ and a constant L satisfying the axioms

$$\begin{array}{rr} \bar{n}(x) + n(x) = 1 & \bar{n}(x) = \bar{n}(x0) \\ \bar{n}(x + y) = \bar{n}(x)\bar{n}(y) & x\,n(y)\mathsf{L} = x0 + n(xy)\mathsf{L} \\ \bar{n}(\bar{n}(x)y) = n(x) + \bar{n}(y) & \bar{n}(x\mathsf{L})x \leq x0 \end{array}$$

The following result shows that $\bar{n}$-semirings provide the intended structure.

Theorem 21. *Let S be an $\bar{n}$-semiring and $x \in S$. Then S is an n-semiring and a test algebra with complement $x' = \bar{n}(x\mathsf{L})$. In particular, $n(S) = \bar{n}(S)$ is a Boolean algebra, $\bar{n}(x)n(x) = 0$ and $\bar{n}$ is $\leq$-antitone. Moreover $n(x) = \bar{n}(\bar{n}(x)\mathsf{L})$ and $\bar{n}(x) = n(\bar{n}(x)\mathsf{L})$.*

By the following consequence, while-programs can be defined as in Section 3.2.4 and all results shown there also hold in the present setting.

Theorem 22. *Let S be an $\bar{n}$-semiring and a left-zero omega algebra. Then S is a test itering using $x^\circ = n(x^\omega)\mathsf{L} + x^*$ and $x' = \bar{n}(x\mathsf{L})$.*

4.1.5 Modal operators

For the domain operation d it is possible to define modal diamond and box operators [87]. Given an element x and a test p, the diamond operator yields $d(xp)$. In model M1 this captures the set of states from which there is an execution of x to a state in p, that is, the preimage of p under x. Dually, the box operator yields the states from which all executions of x go to p, and hence corresponds to the weakest liberal precondition. In other computation models, box describes the weakest precondition or variants thereof [88, 47].

We introduce modal diamond and box operators in $\bar{n}$-semirings. They are defined by

$$\begin{array}{l} |x\rangle y = n(xy\mathsf{L}) \\ |x]y = \bar{n}(x\,\bar{n}(y\mathsf{L})\mathsf{L}) \end{array}$$

Typically the second argument of these operators is a test p. Then $|x\rangle p$ yields the set of states from which x has infinite executions or finite executions terminating in p. Moreover $|x]p$ yields the set of states from which all finite executions of x terminate in p and there are no infinite executions. Diamond and box satisfy the following distribution, duality, induction and unfold properties.

Theorem 23. *Let S be an $\bar{n}$-semiring and $x, y, z \in S$ and $p, q \in n(S)$. Then*

1. $|x + y\rangle z = |x\rangle z + |y\rangle z$
2. $|x\rangle(y + z) = |x\rangle y + |x\rangle z$
3. $|xy\rangle z = |x\rangle|y\rangle z = |x\rangle(yz)$
4. $|x\rangle y = (|x]y')'$
5. $|x + y]z = |x]z \cdot |y]z$
6. $|x](pq) = |x]p \cdot |x]q$
7. $|xy]z = |x]|y]z$
8. $|x]y = (|x\rangle y')'$

using $y' = \bar{n}(y\mathsf{L})$ from the induced test algebra. If S is a left-zero omega algebra,

9. $p \leq |x\rangle p + q \Rightarrow p \leq |x^\omega + x^*\rangle q$
10. $|x^\omega + x^*\rangle p = p + |p'x\rangle|x^\omega + x^*\rangle p$
11. $|x^*\rangle p = p + |p'x\rangle|x^*\rangle p$
12. $|x^\omega + x^*\rangle p = p + |x^\omega + x^*\rangle(p' \cdot |x\rangle p)$
13. $|x]p \cdot q \leq p \Rightarrow |x^\omega + x^*]q \leq p$
14. $|x^\omega + x^*]p = p \cdot |px]|x^\omega + x^*]p$
15. $|x^*]p = p \cdot |px]|x^*]p$
16. $|x^\omega + x^*]p = p \cdot |x^\omega + x^*](p' + |x]p)$

The last property is a version of Segerberg's induction axiom for propositional dynamic logic, but with infinite iterations [104]. Segerberg's formula in turn is based on an axiom of tense logic attributed to Lemmon [98].

4.1.6 Applications: program reasoning

We give several applications that show how to reason about programs using the modal operators. They are instances of the following general result.

Theorem 24. *Let S be an $\bar{n}$-semiring and a left-zero omega algebra. Let $x \in S$ and $p, q, r \in n(S)$. Then*

1. $|x^\circ\rangle p = |(p'x)^\circ\rangle p = |\mathsf{while}\ p'\ \mathsf{do}\ x\rangle p$,
2. $qp\mathsf{L} \leq xp\mathsf{L} \ \wedge\ p \leq q + r \ \Rightarrow\ p \leq |\mathsf{while}\ q\ \mathsf{do}\ x\rangle r$,

using $x^\circ = n(x^\omega)\mathsf{L} + x^*$ and $p' = \bar{n}(p\mathsf{L})$ and while-programs from the induced test itering.

The first property has the following interpretation. Its left-hand side $|x^\circ\rangle p$ describes a non-deterministic iteration to reach a state in p. This can be optimised to the deterministic loop $|\mathsf{while}\ p'\ \mathsf{do}\ x\rangle p$ which stops as soon as p is reached.

One instance of the second property is given by the following program x and conditions p, q, r with two integer variables a and b:

$$\begin{array}{ll} x = (a := a/b) & q = (a \geq 1) \\ p = (b \geq 1) & r = (a < 1) = q' \end{array}$$

Then $qp\mathsf{L} \leq p\mathsf{L} \leq pxp\mathsf{L} \leq xp\mathsf{L}$ holds because $p\mathsf{L}$ has only infinite executions starting in a state with $b \geq 1$, and so has $pxp\mathsf{L}$ since the assignment $a := a/b$ terminates and does not change the value of b. Moreover clearly $p \leq 1 = q + q' = q + r$. By Theorem 24 the execution of the program $\mathsf{while}\ a \geq 1\ \mathsf{do}\ a := a/b$ in a state with $b \geq 1$ does not terminate or terminates in a state satisfying $a < 1$. Because the loop is deterministic, this implies that it does not abort.

Another instance of the second property uses $p = (a \geq 1 \wedge b = 1)$ and $r = 0$ while x and q remain as above. Then $qp\mathsf{L} \leq p\mathsf{L} = pxp\mathsf{L} \leq xp\mathsf{L}$ because in a state with $b = 1$ the assignment $a := a/b$ has no effect. Moreover clearly $p \leq q = q + r$. By Theorem 24 the execution of the program $\mathsf{while}\ a \geq 1\ \mathsf{do}\ a := a/b$ in a state with $b = 1$ does not terminate.

4.2 Recursion for strict and non-strict computations

In this section we generalise the operation n so as to capture the infinite executions of both strict and non-strict computations. Based on this operation we define a common approximation order for models M1–M12 and thereby a unified semantics of recursion. The special case of iteration satisfies the extended binary itering axioms of Section 3.3.1.

4.2.1 Infinite executions

We start again by axiomatising 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. A revision of the n-semiring axioms given in Section 4.1.1 is necessary because we aim to capture model computation models.

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

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

We refer to the elements of the image $n(S) = \{n(x) \mid x \in S\}$ as tests although n-algebras do not provide a complement operation for these. An extension to Boolean tests can be done similarly to Section 4.1.4. Except for $(n10)$, these axioms follow from the ones of n-semirings extended by a meet operation. The following remarks describe the underlying intuition, their use in the subsequent development and the relation to n-semiring axioms and axioms in previous works.

1. Axiom $(n1)$ weakens the distributivity axiom $(i3)$ of n-semirings, $n(x)+n(y) = n(x+y)$, which no longer holds. This weakening is necessary to capture models M9 and M11; in the remaining models n distributes over $+$. Axiom $(n1)$ also implies that tests are closed under $+$.

2. Axiom $(n2)$ is the same as axiom $(i4)$ of n-semirings; it 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 Chapter 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 $\cdot$.

3. Axiom $(n3)$ implies that composition of tests is idempotent by setting $y = 0$. The axiom $(i5)$ of n-semirings, $n(x) = n(x0)n(x)$, which was previously used to this effect, and its consequence $n(x) = n(x0)$ do not hold in models M9–M12. Another consequence of $(n3)$ is that n is $\leq$-isotone; this must be axiomatised because distributivity over $+$ no longer holds.

4. Axiom $(n4)$ is needed to establish $\mathsf{L}x \leq \mathsf{L}$. This is a key weakening of the previous property $\mathsf{L}x = \mathsf{L}$ that does not hold for non-strict computations. Hoare [68] formulates this characteristic strictness property as $\top x = \top$, which is the same in models M2 and M9–M12 where $\mathsf{L} = \top$, but different in other models. An axiom related to $(n4)$ but with the domain operation instead of n is used in [48] for a different purpose.

5. 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 in Section 4.2.2.

6. 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. Axiom $(i2)$ of n-semirings, $n(\top) = 1$, and the property $n(\mathsf{L}) = 1$ do not hold in model M1. While axiom $(i1)$ of n-semirings, $n(0) = 0$, holds in models M1–M12, it does not follow from the current axioms. Nevertheless $n(0)$ 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 for while-programs, which would include 0 and 1 for false and true, respectively.

7. Axiom ($n7$) is a component of the Galois connection $n(x)\mathsf{L} \leq y \Leftrightarrow n(x) \leq n(y)$, which is characteristic for the operation n. Hence all infinite executions restricted to a starting state in $n(x)$ are contained in x.

8. Axiom ($n8$) is used, for example, to show that $\cdot$ is isotone with respect to the approximation order. Models M2–M12 satisfy $n(\mathsf{L}) = 1$, while $n(\mathsf{L}) = 0$ holds in model M1. 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 of Section 4.2.2. A similar axiom with the domain operation instead of n is used in [48] to this end.

9. 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 in n-semirings are ($i6$) and ($i7$).

10. Axiom ($n10$) generalises the previous property $x\top \curlywedge \mathsf{L} \leq x\mathsf{L}$ of [48], 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 and Nitpick 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 25. *Let S be an n-algebra and $x, y \in S$. Then $(n(S), +, \cdot, n(0), n(\top))$ is an idempotent semiring 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:

$$\begin{array}{rrr} n(0) = 0 & x\,n(y)\mathsf{L} = x0 + n(xy)\mathsf{L} & n(\mathsf{L})x\top \leq n(x\top \curlywedge \mathsf{L})\top \\ n(1) = 0 & x \leq x0 + n(x\mathsf{L})\top & n(x\top \curlywedge \mathsf{L})\top \leq n(\mathsf{L})x\top \\ n(\mathsf{L}) = 1 & n(xy) \leq n(x\,n(y)\top) & x0 \curlywedge \mathsf{L} \leq n(x\mathsf{L})\mathsf{L} \\ n(\top) = 1 & n(\mathsf{L})x \leq n(x\top)\top & n(x\mathsf{L})\mathsf{L} \leq n(x)\mathsf{L} \\ n(x) = n(x0) & x \curlywedge n(y)\top \leq n(y)x & n(x)\mathsf{L} \leq (x \curlywedge \mathsf{L})0 \\ n(x) + n(y) = n(x + y) & x \curlywedge n(y)\top \leq n(\mathsf{L})x & (x \curlywedge \mathsf{L})0 \leq n(x)\mathsf{L} \end{array}$$

4.2.2 Approximation and recursion

All computation models of Chapter 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 [42, 45, 48] so as to capture models M1–M12. The intuition underlying this definition is as follows. If $n(\mathsf{L}) = 0$, then $\mathsf{L} = n(\mathsf{L})\mathsf{L} = 0$ by Theorem 25.26 and $x \sqsubseteq y$ reduces to $x \leq y$. This captures model M1; 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 26. *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 left-zero Kleene algebra, the operation * is $\sqsubseteq$-isotone.*

We reuse the definitions of κf and $\sqcap$ given in Section 4.1.2 with respect to the generalised approximation order. The following characterisations of κf generalise to the present setting of n-algebras, which covers models M1–M12.

Theorem 27. *Let S be an n-algebra and let $f : S \to S$ be $\leq$- and $\sqsubseteq$-isotone such 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. Further characterisations can be generalised to n-algebras as shown in the following result.

Theorem 28. *Let S be an n-algebra and let $f : S \to S$ be $\leq$- and $\sqsubseteq$-isotone such that μf and νf exist. Then the following are equivalent and imply the statements of Theorem 27:*

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 27.

4.2.3 Application: correctness of unfold-fold

As an example showing the usefulness of the representation granted by Theorem 27, we describe the unfold-fold method [18], which can be used to develop recursive programs from specifications. We are concerned with a generalisation given by [8] that allows the reduction of non-determinism in addition to meaning-preserving transformations. Its essence is captured in our algebraic setting as follows:

1. Start with a specification $x_0 \in S$.
2. Successively apply meaning-preserving or refining transformations (such as unfold and fold) to obtain a sequence of specifications $x_0, x_1, x_2, \ldots, x_n \in S$ where each step maintains or reduces non-determinism, that is, $x_i \geq x_{i+1}$.
3. Reach a specification x_n which is given in terms of the original x_0, that is, $x_n = f(x_0)$.
4. Turn it into the recursive program κf.

In summary, we have $f(x_0) = x_n \leq x_{n-1} \leq \ldots \leq x_1 \leq x_0$, whence x_0 is a prefixpoint of f. If f is $\leq$-isotone and has a $\leq$-least prefixpoint $\hat{\mu} f$, it coincides with the $\leq$-least fixpoint μf and we obtain $\mu f \leq x_0$. This means that the method is valid in the partial-correctness model M1, where recursions are solved by $\leq$-least fixpoints. The validity in other computation models, however, amounts to $\kappa f \leq x_0$, which states that the recursively defined result κf implements the original specification x_0. It is not clear that this holds, since x_0 is not necessarily a fixpoint of f, and even if $f(x_0) = x_0$ held, we could only conclude $\kappa f \sqsubseteq x_0$.

A proof of the unfold-fold method for general correctness is given by [8] in a functional setting. We algebraically state and prove their result and thereby generalise it to other computation models. Validity of unfold-fold in a total-correctness model is addressed by [33, Theorem 4.5] in relation algebras.

Theorem 29. *Let S be an n-algebra and let $f : S \to S$ be $\leq$- and $\sqsubseteq$-isotone such that $\hat{\mu} f$, νf and κf exist. Then*

1. *$f(x) \leq x \;\Rightarrow\; \kappa f \leq x + \mathsf{L}$.*
2. *$f(x) \leq x \;\wedge\; \kappa f \curlywedge \mathsf{L} \leq x \curlywedge \mathsf{L} \;\Rightarrow\; \kappa f \leq x$.*

Part 1 corresponds to [8, Theorem 4.4] and part 2 to [8, Corollary 4.5]; we note that our proof needs no induction. Intuitively, $\kappa f \curlywedge \mathsf{L} \leq x \curlywedge \mathsf{L}$ states that whenever x terminates, so does κf.

Note that the main correctness claim $\kappa f \leq x$ combines the refinement order $\leq$ with the least fixpoint κf in the approximation order $\sqsubseteq$. Another example for such a combination of the two orders is the general-correctness loop refinement rule [38, Theorem 11].

4.2.4 Iteration

Iteration is dealt with by instantiating the results of Section 4.2.2 for the affine function $f(x) = yx + z$. Note that Theorems 27 and 28 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 a left-zero 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.

11. Axiom $(n11)$ is adapted from Theorem 18 and captures the interaction of the operations * 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.

12. Axiom $(n12)$ modifies the property $x\top \leq x\top x\top$ of Section 3.3.1. While the latter holds in relation algebras, a counterexample generated by Nitpick witnesses that it does not hold in n-omega algebras. Models M1–M8 satisfy $\mathsf{L}x = \mathsf{L}$ and the models M9–M12 of non-strict computations are relations or isomorphic to relations with $\mathsf{L} = \top$, whence $(n12)$ holds for both kinds.

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

Theorem 30. *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 27 and 28 to obtain the $\sqsubseteq$-least fixpoint of an affine function. It also shows that every n-omega algebra forms an extended binary itering.

Theorem 31. *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 Section 3.3 hold in models M1–M12. This includes various simulation and separation laws generalised from omega algebras and Back's atomicity refinement theorem. 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 models M1–M12 we obtain additional guarantees, for example, about the absence of aborting executions.

4.3 Instances for computation models

We exemplify how to instantiate the algebraic structures introduced in this chapter to our computation models. To this end, we give instances of n-omega algebras for computation model M8 in Section 4.3.2 and for models M11–M12 in Section 4.3.3. Instantiation to the other models and of n-semirings to models M3–M5 and M7 works similarly.

4.3.1 Relation algebras and residuals

Because models M11–M12 are relational and the matrices of model M8 contain relations we work in relation algebras. A *relation algebra* $(S, +, \curlywedge, \cdot, \overline{\ }, {}^{\mathsf{c}}, 0, 1, \top)$ [107, 79] adds to a Boolean algebra $(S, +, \curlywedge, \overline{\ }, 0, \top)$ a composition operation $\cdot$, a converse operation ${}^{\mathsf{c}}$ and a constant 1 with the following axioms:

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

It follows that the reduct $(S, +, \cdot, 0, 1)$ of every relation algebra is an idempotent semiring. Relations over a set A form a relation algebra with the operations set union, set intersection, relational composition, set complement, relational converse and the constants O, I and T.

In relation algebras, the *left residual* is defined by $x/y = \overline{\overline{x}y^{\mathsf{c}}}$. It provides a weak inverse of composition as characterised by the Galois connection in part 1 of the following theorem. We include further properties of left residuals used for proving the instances below. Composition has higher precedence than /, which has higher precedence than + and $\curlywedge$.

Theorem 32. *Let $(S, +, \curlywedge, \cdot, \overline{\ }, {}^{\mathsf{c}}, 0, 1, \top)$ be a relation algebra with left residual $x/y = \overline{\overline{x}y^{\mathsf{c}}}$. Let $x, y, z \in S$. Then*

1. $xy \leq z \;\Leftrightarrow\; x \leq z/y$
2. $\lambda x.x/y$ *is* $\leq$*-isotone and* $\lambda y.x/y$ *is* $\leq$*-antitone*
3. $(x/y)/z = x/zy$
4. $x(y/z) \leq xy/z$
5. $(x \curlywedge y)/z = x/z \curlywedge y/z$
6. $(x + y)/z = x + y/z$ *if* x *is a vector*
7. $z/\top$ *is a vector*
8. $z/\top \leq z$
9. $\top/z = \top$
10. $(x/\top \curlywedge y)z = x/\top \curlywedge yz$

4.3.2 Instance for strict computations

Recall from Section 2.3.3 that a computation in model M8 is a 3×3 matrix of relations over the state space A. The matrix has the following form:

$$(P|Q|R) = \begin{pmatrix} \mathsf{I} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} & \mathsf{O} \\ P & Q & R \end{pmatrix}$$

The relation P represents the aborting executions, Q represents the infinite executions and R represents the finite executions. Let $M_8 = \{(P|Q|R) \mid P, Q, R \subseteq A \times A\}$ be the set of such matrices. Basic operations on M_8 are defined as follows.

* Non-deterministic choice is given by the componentwise union of the involved matrices:

$$(P_1|Q_1|R_1) + (P_2|Q_2|R_2) = (P_1 \cup P_2|Q_1 \cup Q_2|R_1 \cup R_2)$$

* Conjunction is given by the componentwise intersection of the involved matrices:

$$(P_1|Q_1|R_1) \curlywedge (P_2|Q_2|R_2) = (P_1 \cap P_2|Q_1 \cap Q_2|R_1 \cap R_2)$$

* Sequential composition is given by the matrix product, where union and relational composition replace addition and multiplication. This elaborates as follows:

$$(P_1|Q_1|R_1) \cdot (P_2|Q_2|R_2) = (P_1 \cup R_1P_2|Q_1 \cup R_1Q_2|R_1R_2)$$

* The refinement order is the componentwise set inclusion order:

$$(P_1|Q_1|R_1) \leq (P_2|Q_2|R_2) \;\Leftrightarrow\; P_1 \subseteq P_2 \;\wedge\; Q_1 \subseteq Q_2 \;\wedge\; R_1 \subseteq R_2$$

* The computation with no executions is

$$0 = (\mathsf{O}|\mathsf{O}|\mathsf{O})$$

  The computation 0 is an identity of non-deterministic choice, a zero of conjunction, a left zero of sequential composition and the least element in the refinement order.

* The computation with all executions is

$$\top = (\mathsf{T}|\mathsf{T}|\mathsf{T})$$

  The computation $\top$ is a zero of non-deterministic choice, an identity of conjunction and the greatest element in the refinement order.

* The computation that does not change the state is

$$1 = (\mathsf{O}|\mathsf{O}|\mathsf{I})$$

  The computation 1 is an identity of sequential composition.

* The computation with all infinite executions is

$$\mathsf{L} = (\mathsf{O}|\mathsf{T}|\mathsf{O})$$

  The computation L is a left zero of sequential composition and the least element in the approximation order derived below.

These operations give the following basic structure for computation model M8. We omit the proof which is obtained by simple matrix calculations.

Theorem 33. *$(M_8, +, \cdot, 0, 1)$ is an idempotent left-zero semiring and $(M_8, +, \curlywedge, 0, \top)$ is a bounded distributive lattice with order $\leq$.*

For defining the semantics of recursion an approximation order is needed. The refinement order cannot be used for this purpose because L is not its least element. To obtain a suitable approximation order we instantiate n-algebras. A test in M_8 is an element ≤ 1, that is, a computation of the form

$$(\mathsf{O}|\mathsf{O}|R)$$

where $R \subseteq \mathsf{I}$. We now derive the operation n, which maps computations to tests, from its characteristic Galois connection:

$$n(x)\mathsf{L} \leq y \;\Leftrightarrow\; n(x) \leq n(y)$$

Thus $n(y)$ is the greatest test whose composition with L is below y. We use this Galois connection to obtain a definition of n in M_8. Because the result of n is a test, we assume that n has the general form

$$n(P|Q|R) = (\mathsf{O}|\mathsf{O}|f(P,Q,R))$$

using a function f that maps its argument relations P, Q and R to a relation below the identity I, that is, $f(P,Q,R) \subseteq \mathsf{I}$. By the Galois connection and Theorem 32.1,

$$\begin{array}{ll} & f(P_1,Q_1,R_1) \subseteq f(P_2,Q_2,R_2) \\ \Leftrightarrow & (\mathsf{O}|\mathsf{O}|f(P_1,Q_1,R_1)) \leq (\mathsf{O}|\mathsf{O}|f(P_2,Q_2,R_2)) \\ \Leftrightarrow & n(P_1|Q_1|R_1) \leq n(P_2|Q_2|R_2) \\ \Leftrightarrow & n(P_1|Q_1|R_1)\mathsf{L} \leq (P_2|Q_2|R_2) \\ \Leftrightarrow & (\mathsf{O}|\mathsf{O}|f(P_1,Q_1,R_1))(\mathsf{O}|\mathsf{T}|\mathsf{O}) \leq (P_2|Q_2|R_2) \\ \Leftrightarrow & (\mathsf{O}|f(P_1,Q_1,R_1)\mathsf{T}|\mathsf{O}) \leq (P_2|Q_2|R_2) \\ \Leftrightarrow & f(P_1,Q_1,R_1)\mathsf{T} \subseteq Q_2 \\ \Leftrightarrow & f(P_1,Q_1,R_1) \subseteq Q_2/\mathsf{T} \\ \Leftrightarrow & f(P_1,Q_1,R_1) \subseteq Q_2/\mathsf{T} \cap \mathsf{I} \end{array}$$

The above calculation suggests the definition $f(P,Q,R) = Q/\mathsf{T} \cap \mathsf{I}$. A simple rearrangement of the calculation shows that this satisfies the Galois connection for n. We therefore define

$$n(P|Q|R) = (\mathsf{O}|\mathsf{O}|Q/\mathsf{T} \cap \mathsf{I})$$

The following result shows that n satisfies the axioms of n-algebras given in Section 4.2.1.

Theorem 34. $(M_8, +, \curlywedge, \cdot, n, 0, 1, \mathsf{L}, \top)$ *is an n-algebra.*

Proof.

$(n1)$ Using Theorems 32.10, 32.7 and 32.6,

$$\begin{array}{ll} & n(n(P_1|Q_1|R_1)(\mathsf{T}|\mathsf{T}|\mathsf{T}) + (P_2|Q_2|R_2)) \\ = & n((\mathsf{O}|\mathsf{O}|Q_1/\mathsf{T} \cap \mathsf{I})(\mathsf{T}|\mathsf{T}|\mathsf{T}) + (P_2|Q_2|R_2)) \\ = & n(((Q_1/\mathsf{T} \cap \mathsf{I})\mathsf{T}|(Q_1/\mathsf{T} \cap \mathsf{I})\mathsf{T}|(Q_1/\mathsf{T} \cap \mathsf{I})\mathsf{T}) + (P_2|Q_2|R_2)) \\ = & n((Q_1/\mathsf{T}|Q_1/\mathsf{T}|Q_1/\mathsf{T}) + (P_2|Q_2|R_2)) \\ = & n(Q_1/\mathsf{T} \cup P_2|Q_1/\mathsf{T} \cup Q_2|Q_1/\mathsf{T} \cup R_2) \\ = & (\mathsf{O}|\mathsf{O}|(Q_1/\mathsf{T} \cup Q_2)/\mathsf{T} \cap \mathsf{I}) \\ = & (\mathsf{O}|\mathsf{O}|(Q_1/\mathsf{T} \cup Q_2/\mathsf{T}) \cap \mathsf{I}) \\ = & (\mathsf{O}|\mathsf{O}|(Q_1/\mathsf{T} \cap \mathsf{I}) \cup (Q_2/\mathsf{T} \cap \mathsf{I})) \\ = & (\mathsf{O}|\mathsf{O}|Q_1/\mathsf{T} \cap \mathsf{I}) + (\mathsf{O}|\mathsf{O}|Q_2/\mathsf{T} \cap \mathsf{I}) \\ = & n(P_1|Q_1|R_1) + n(P_2|Q_2|R_2) \end{array}$$

$(n2)$ Using Theorems 32.10, 32.5 and 32.3,

$$\begin{array}{ll} & n(n(P_1|Q_1|R_1)(P_2|Q_2|R_2)) \\ = & n((\mathsf{O}|\mathsf{O}|Q_1/\mathsf{T} \cap \mathsf{I})(P_2|Q_2|R_2)) \\ = & n((Q_1/\mathsf{T} \cap \mathsf{I})P_2|(Q_1/\mathsf{T} \cap \mathsf{I})Q_2|(Q_1/\mathsf{T} \cap \mathsf{I})R_2) \\ = & n(Q_1/\mathsf{T} \cap P_2|Q_1/\mathsf{T} \cap Q_2|Q_1/\mathsf{T} \cap R_2) \\ = & (\mathsf{O}|\mathsf{O}|(Q_1/\mathsf{T} \cap Q_2)/\mathsf{T} \cap \mathsf{I}) \\ = & (\mathsf{O}|\mathsf{O}|(Q_1/\mathsf{T})/\mathsf{T} \cap Q_2/\mathsf{T} \cap \mathsf{I}) \\ = & (\mathsf{O}|\mathsf{O}|Q_1/\mathsf{T}\mathsf{T} \cap Q_2/\mathsf{T} \cap \mathsf{I}) \\ = & (\mathsf{O}|\mathsf{O}|Q_1/\mathsf{T} \cap Q_2/\mathsf{T} \cap \mathsf{I}) \\ = & (\mathsf{O}|\mathsf{O}|(Q_1/\mathsf{T} \cap \mathsf{I})(Q_2/\mathsf{T} \cap \mathsf{I})) \\ = & (\mathsf{O}|\mathsf{O}|Q_1/\mathsf{T} \cap \mathsf{I})(\mathsf{O}|\mathsf{O}|Q_2/\mathsf{T} \cap \mathsf{I}) \\ = & n(P_1|Q_1|R_1)n(P_2|Q_2|R_2) \end{array}$$

($n3$) Using Theorems 32.10 and 32.2,

$$\begin{aligned}
&n(P_1|Q_1|R_1)n((P_1|Q_1|R_1)+(P_2|Q_2|R_2))\\
&= n(P_1|Q_1|R_1)n(P_1\cup P_2|Q_1\cup Q_2|R_1\cup R_2)\\
&= (\mathsf{O}|\mathsf{O}|Q_1/\mathsf{T}\cap\mathsf{I})(\mathsf{O}|\mathsf{O}|(Q_1\cup Q_2)/\mathsf{T}\cap\mathsf{I})\\
&= (\mathsf{O}|\mathsf{O}|(Q_1/\mathsf{T}\cap\mathsf{I})((Q_1\cup Q_2)/\mathsf{T}\cap\mathsf{I}))\\
&= (\mathsf{O}|\mathsf{O}|Q_1/\mathsf{T}\cap(Q_1\cup Q_2)/\mathsf{T}\cap\mathsf{I})\\
&= (\mathsf{O}|\mathsf{O}|Q_1/\mathsf{T}\cap\mathsf{I})\\
&= n(P_1|Q_1|R_1)
\end{aligned}$$

($n4$) In M_8 we have $\mathsf{L}x=\mathsf{L}$ for any $x\in S$ because $(\mathsf{O}|\mathsf{T}|\mathsf{O})(P|Q|R)=(\mathsf{O}|\mathsf{T}|\mathsf{O})$. Moreover, using Theorem 32.9,

$$n(\mathsf{L})=n(\mathsf{O}|\mathsf{T}|\mathsf{O})=(\mathsf{O}|\mathsf{O}|\mathsf{T}/\mathsf{T}\cap\mathsf{I})=(\mathsf{O}|\mathsf{O}|\mathsf{T}\cap\mathsf{I})=(\mathsf{O}|\mathsf{O}|\mathsf{I})=1$$

Hence the claim follows by simple lattice and semiring calculations.

($n5$) Using Theorems 32.10, 32.6 and 32.8,

$$\begin{aligned}
&(P|Q|R)(\mathsf{O}|\mathsf{O}|\mathsf{O})+n((P|Q|R)(\mathsf{O}|\mathsf{T}|\mathsf{O}))(\mathsf{O}|\mathsf{T}|\mathsf{O})\\
&= (P|Q|\mathsf{O})+n(P|Q\cup R\mathsf{T}|\mathsf{O})(\mathsf{O}|\mathsf{T}|\mathsf{O})\\
&= (P|Q|\mathsf{O})+(\mathsf{O}|\mathsf{O}|(Q\cup R\mathsf{T})/\mathsf{T}\cap\mathsf{I})(\mathsf{O}|\mathsf{T}|\mathsf{O})\\
&= (P|Q|\mathsf{O})+(\mathsf{O}|((Q\cup R\mathsf{T})/\mathsf{T}\cap\mathsf{I})\mathsf{T}|\mathsf{O})\\
&= (P|Q|\mathsf{O})+(\mathsf{O}|(Q\cup R\mathsf{T})/\mathsf{T}|\mathsf{O})\\
&= (P|Q|\mathsf{O})+(\mathsf{O}|Q/\mathsf{T}\cup R\mathsf{T}|\mathsf{O})\\
&= (P|Q\cup Q/\mathsf{T}\cup R\mathsf{T}|\mathsf{O})\\
&= (P|Q\cup R\mathsf{T}|\mathsf{O})\\
&= (P|Q|R)(\mathsf{O}|\mathsf{T}|\mathsf{O})
\end{aligned}$$

($n6$) The claim follows since $n(x)\leq 1$ and $n(\mathsf{L})=1$ as shown above.

($n7$) Using Theorems 32.10 and 32.8,

$$\begin{aligned}
&n(P|Q|R)(\mathsf{O}|\mathsf{T}|\mathsf{O})\\
&= (\mathsf{O}|\mathsf{O}|Q/\mathsf{T}\cap\mathsf{I})(\mathsf{O}|\mathsf{T}|\mathsf{O})\\
&= (\mathsf{O}|(Q/\mathsf{T}\cap\mathsf{I})\mathsf{T}|\mathsf{O})\\
&= (\mathsf{O}|Q/\mathsf{T}|\mathsf{O})\\
&\leq (P|Q|R)
\end{aligned}$$

($n8$) The claim follows since $n(\mathsf{L})=1$ as shown above.

($n9$) Using Theorems 32.10, 32.4 and 32.2,

$$\begin{aligned}
&(P_1|Q_1|R_1)n(P_2|Q_2|R_2)(\mathsf{T}|\mathsf{T}|\mathsf{T})\\
&= (P_1|Q_1|R_1)(\mathsf{O}|\mathsf{O}|Q_2/\mathsf{T}\cap\mathsf{I})(\mathsf{T}|\mathsf{T}|\mathsf{T})\\
&= (P_1|Q_1|R_1)((Q_2/\mathsf{T}\cap\mathsf{I})\mathsf{T}|(Q_2/\mathsf{T}\cap\mathsf{I})\mathsf{T}|(Q_2/\mathsf{T}\cap\mathsf{I})\mathsf{T})\\
&= (P_1|Q_1|R_1)(Q_2/\mathsf{T}|Q_2/\mathsf{T}|Q_2/\mathsf{T})\\
&= (P_1\cup R_1(Q_2/\mathsf{T})|Q_1\cup R_1(Q_2/\mathsf{T})|R_1(Q_2/\mathsf{T}))\\
&= (P_1|Q_1|\mathsf{O})+(R_1(Q_2/\mathsf{T})|R_1(Q_2/\mathsf{T})|R_1(Q_2/\mathsf{T}))\\
&\leq (P_1|Q_1|\mathsf{O})+(R_1Q_2/\mathsf{T}|R_1Q_2/\mathsf{T}|R_1Q_2/\mathsf{T})\\
&= (P_1|Q_1|\mathsf{O})+((R_1Q_2/\mathsf{T}\cap\mathsf{I})\mathsf{T}|(R_1Q_2/\mathsf{T}\cap\mathsf{I})\mathsf{T}|(R_1Q_2/\mathsf{T}\cap\mathsf{I})\mathsf{T})\\
&= (P_1|Q_1|\mathsf{O})+(\mathsf{O}|\mathsf{O}|R_1Q_2/\mathsf{T}\cap\mathsf{I})(\mathsf{T}|\mathsf{T}|\mathsf{T})\\
&\leq (P_1|Q_1|\mathsf{O})+(\mathsf{O}|\mathsf{O}|(Q_1\cup R_1Q_2)/\mathsf{T}\cap\mathsf{I})(\mathsf{T}|\mathsf{T}|\mathsf{T})\\
&= (P_1|Q_1|\mathsf{O})+n(P_1\cup R_1P_2|Q_1\cup R_1Q_2|R_1R_2)(\mathsf{T}|\mathsf{T}|\mathsf{T})\\
&= (P_1|Q_1|R_1)(\mathsf{O}|\mathsf{O}|\mathsf{O})+n((P_1|Q_1|R_1)(P_2|Q_2|R_2))(\mathsf{T}|\mathsf{T}|\mathsf{T})
\end{aligned}$$

$(n10)$ Since $\mathsf{L}y = \mathsf{L}$ as shown above,

$$\begin{array}{ll} & (P_1|Q_1|R_1)(\mathsf{T}|\mathsf{T}|\mathsf{T})(P_2|Q_2|R_2) \curlywedge (\mathsf{O}|\mathsf{T}|\mathsf{O}) \\ = & (P_1 \cup R_1\mathsf{T}|Q_1 \cup R_1\mathsf{T}|R_1\mathsf{T})(P_2|Q_2|R_2) \curlywedge (\mathsf{O}|\mathsf{T}|\mathsf{O}) \\ = & (P_1 \cup R_1\mathsf{T} \cup R_1\mathsf{T}P_2|Q_1 \cup R_1\mathsf{T} \cup R_1\mathsf{T}Q_2|R_1\mathsf{T}R_2) \curlywedge (\mathsf{O}|\mathsf{T}|\mathsf{O}) \\ = & (\mathsf{O}|Q_1 \cup R_1\mathsf{T}|\mathsf{O}) \\ \leq & (P_1|Q_1 \cup R_1\mathsf{T}|\mathsf{O}) \\ = & (P_1|Q_1|R_1)(\mathsf{O}|\mathsf{T}|\mathsf{O}) \\ = & (P_1|Q_1|R_1)(\mathsf{O}|\mathsf{T}|\mathsf{O})(P_2|Q_2|R_2) \end{array} \qquad \square$$

We thus automatically obtain the following approximation order $\sqsubseteq$ given in Section 4.2.2:

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

Because $n(\mathsf{L}) = 1$ in M_8, this simplifies to

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

The order elaborates as follows. Using Theorems 32.10 and 32.8,

$$\begin{array}{ll} & (P_1|Q_1|R_1) + n(P_1|Q_1|R_1)(\mathsf{T}|\mathsf{T}|\mathsf{T}) \\ = & (P_1|Q_1|R_1) + (\mathsf{O}|\mathsf{O}|Q_1/\mathsf{T} \cap \mathsf{I})(\mathsf{T}|\mathsf{T}|\mathsf{T}) \\ = & (P_1|Q_1|R_1) + ((Q_1/\mathsf{T} \cap \mathsf{I})\mathsf{T}|(Q_1/\mathsf{T} \cap \mathsf{I})\mathsf{T}|(Q_1/\mathsf{T} \cap \mathsf{I})\mathsf{T}) \\ = & (P_1|Q_1|R_1) + (Q_1/\mathsf{T}|Q_1/\mathsf{T}|Q_1/\mathsf{T}) \\ = & (P_1 \cup Q_1/\mathsf{T}|Q_1 \cup Q_1/\mathsf{T}|R_1 \cup Q_1/\mathsf{T}) \\ = & (P_1 \cup Q_1/\mathsf{T}|Q_1|R_1 \cup Q_1/\mathsf{T}) \end{array}$$

Hence,

$$\begin{array}{lll} & (P_1|Q_1|R_1) \sqsubseteq (P_2|Q_2|R_2) & \\ \Leftrightarrow & (P_1|Q_1|R_1) \leq (P_2|Q_2|R_2) + (\mathsf{O}|\mathsf{T}|\mathsf{O}) \;\wedge & \\ & (P_2|Q_2|R_2) \leq (P_1|Q_1|R_1) + n(P_1|Q_1|R_1)(\mathsf{T}|\mathsf{T}|\mathsf{T}) & \\ \Leftrightarrow & (P_1|Q_1|R_1) \leq (P_2|\mathsf{T}|R_2) \;\wedge & \\ & (P_2|Q_2|R_2) \leq (P_1 \cup Q_1/\mathsf{T}|Q_1|R_1 \cup Q_1/\mathsf{T}) & \\ \Leftrightarrow & P_1 \subseteq P_2 \;\wedge\; R_1 \subseteq R_2 \;\wedge & \\ & P_2 \subseteq P_1 \cup Q_1/\mathsf{T} \;\wedge\; Q_2 \subseteq Q_1 \;\wedge\; R_2 \subseteq R_1 \cup Q_1/\mathsf{T} & \\ \Leftrightarrow & P_1 \subseteq P_2 \subseteq P_1 \cup Q_1/\mathsf{T} \;\wedge\; Q_2 \subseteq Q_1 \;\wedge\; R_1 \subseteq R_2 \subseteq R_1 \cup Q_1/\mathsf{T} & \end{array}$$

The relation Q_1/T is a vector that represents the states where all infinite executions of the computation $(P_1|Q_1|R_1)$ are present. The intuition underlying the approximation order is that in states with all infinite executions, any execution can be added. In states where at least one infinite execution is missing, no executions can be added. Only infinite executions can be removed.

For instantiating further results concerning iteration, we show that the computations of model M8 form an n-omega algebra. To this end, we first derive the Kleene star and omega operations. Conway's automata-based matrix construction [22] yields the Kleene star of a 2×2 matrix:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^* = \begin{pmatrix} e^* & a^*bf^* \\ d^*ce^* & f^* \end{pmatrix} \qquad \text{where} \qquad \begin{pmatrix} e \\ f \end{pmatrix} = \begin{pmatrix} a \cup bd^*c \\ d \cup ca^*b \end{pmatrix}$$

In model M8, the matrix entries are relations whose Kleene star is the reflexive-transitive closure. Because the computations are 3×3 matrices the construction has to be applied twice. First,

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q & R \end{pmatrix}^* = \begin{pmatrix} \mathsf{I}^* & \mathsf{I}^*\mathsf{O}R^* \\ R^*Q\mathsf{I}^* & R^* \end{pmatrix} = \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ R^*Q & R^* \end{pmatrix} \quad \text{using} \quad \begin{pmatrix} e \\ f \end{pmatrix} = \begin{pmatrix} \mathsf{I} \cup \mathsf{O}R^*Q \\ R \cup Q\mathsf{I}^*\mathsf{O} \end{pmatrix} = \begin{pmatrix} \mathsf{I} \\ R \end{pmatrix}$$

Second,

$$\begin{pmatrix} \mathsf{I} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} & \mathsf{O} \\ P & Q & R \end{pmatrix}^* = \begin{pmatrix} \mathsf{I}^* & \mathsf{I}^*(\mathsf{O}\ \ \mathsf{O})\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q & R \end{pmatrix}^* \\ \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q & R \end{pmatrix}^*\begin{pmatrix} \mathsf{O} \\ P \end{pmatrix}\mathsf{I}^* & \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q & R \end{pmatrix}^* \end{pmatrix}$$

$$= \begin{pmatrix} \mathsf{I} & (\mathsf{O}\ \ \mathsf{O}) \\ \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ R^*Q & R^* \end{pmatrix}\begin{pmatrix} \mathsf{O} \\ P \end{pmatrix} & \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ R^*Q & R^* \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \mathsf{I} & \mathsf{O} & \mathsf{O} \\ \mathsf{O} & \mathsf{I} & \mathsf{O} \\ R^*P & R^*Q & R^* \end{pmatrix}$$

using

$$\begin{pmatrix} e \\ f \end{pmatrix} = \begin{pmatrix} \mathsf{I} \cup (\mathsf{O}\ \ \mathsf{O})\begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q & R \end{pmatrix}^*\begin{pmatrix} \mathsf{O} \\ P \end{pmatrix} \\ \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q & R \end{pmatrix} \cup \begin{pmatrix} \mathsf{O} \\ P \end{pmatrix}\mathsf{I}^*(\mathsf{O}\ \ \mathsf{O}) \end{pmatrix} = \begin{pmatrix} \mathsf{I} \\ \begin{pmatrix} \mathsf{I} & \mathsf{O} \\ Q & R \end{pmatrix} \end{pmatrix}$$

It follows that

$$(P|Q|R)^* = (R^*P|R^*Q|R^*)$$

The standard automata-based construction does not work for the omega operation as the resulting matrix does not have the form of the matrices in model M8. This issue can be solved in the framework of typed omega algebras as detailed in [44]. The resulting operation is

$$(P|Q|R)^\omega = (R^\omega \cup R^*P|R^\omega \cup R^*Q|R^\omega)$$

The operation $^\omega$ on relations describes the states from which infinite transition paths exist. The following result shows that $^\omega$ on computations in model M8 satisfies the axioms of left-zero omega algebras given in Section 3.1.3 and that n satisfies the axioms of n-omega algebras given in Section 4.2.4.

Theorem 35. $(M_8, +, \curlywedge, \cdot, n, ^*, ^\omega, 0, 1, \mathsf{L}, \top)$ *is an n-omega algebra.*

Proof. The Kleene star has been derived above. We first show the left-zero omega algebra axioms and then the n-omega algebra axioms.

- The omega unfold axiom is obtained by

$$\begin{aligned} & (P|Q|R)(P|Q|R)^\omega \\ = {} & (P|Q|R)(R^\omega \cup R^*P|R^\omega \cup R^*Q|R^\omega) \\ = {} & (P \cup R(R^\omega \cup R^*P)|Q \cup R(R^\omega \cup R^*Q)|RR^\omega) \\ = {} & (RR^\omega \cup RR^*P \cup P|RR^\omega \cup RR^*Q \cup Q|RR^\omega) \\ = {} & (R^\omega \cup (RR^* \cup \mathsf{I})P|R^\omega \cup (RR^* \cup \mathsf{I})Q|R^\omega) \\ = {} & (R^\omega \cup R^*P|R^\omega \cup R^*Q|R^\omega) \\ = {} & (P|Q|R)^\omega \end{aligned}$$

  using the omega unfold axiom on the underlying relations.

- For the omega induction axiom, assume

$$(P_1|Q_1|R_1) \leq (P_2|Q_2|R_2)(P_1|Q_1|R_1) + (P_3|Q_3|R_3)$$

  This implies

$$\begin{aligned} & (P_1|Q_1|R_1) \\ \leq {} & (P_2|Q_2|R_2)(P_1|Q_1|R_1) + (P_3|Q_3|R_3) \\ = {} & (P_2 \cup R_2P_1|Q_2 \cup R_2Q_1|R_2R_1) + (P_3|Q_3|R_3) \\ = {} & (R_2P_1 \cup P_2 \cup P_3|R_2Q_1 \cup Q_2 \cup Q_3|R_2R_1 \cup R_3) \end{aligned}$$

whence

$$\begin{aligned} P_1 &\subseteq R_2P_1 \cup P_2 \cup P_3 \\ Q_1 &\subseteq R_2Q_1 \cup Q_2 \cup Q_3 \\ R_1 &\subseteq R_2R_1 \cup R_3 \end{aligned}$$

By the omega induction axiom on the underlying relations,

$$\begin{aligned} P_1 &\subseteq R_2^\omega \cup R_2^*(P_2 \cup P_3) \\ Q_1 &\subseteq R_2^\omega \cup R_2^*(Q_2 \cup Q_3) \\ R_1 &\subseteq R_2^\omega \cup R_2^*R_3 \end{aligned}$$

Therefore

$$\begin{aligned} &(P_1|Q_1|R_1) \\ &\leq (R_2^\omega \cup R_2^*(P_2 \cup P_3)|R_2^\omega \cup R_2^*(Q_2 \cup Q_3)|R_2^\omega \cup R_2^*R_3) \\ &= (R_2^\omega \cup R_2^*P_2 \cup R_2^*P_3|R_2^\omega \cup R_2^*Q_2 \cup R_2^*Q_3|R_2^\omega \cup R_2^*R_3) \\ &= (R_2^\omega \cup R_2^*P_2|R_2^\omega \cup R_2^*Q_2|R_2^\omega) + (R_2^*P_2 \cup R_2^*P_3|R_2^*Q_2 \cup R_2^*Q_3|R_2^*R_3) \\ &= (P_2|Q_2|R_2)^\omega + (R_2^*P_2|R_2^*Q_2|R_2^*)(P_3|Q_3|R_3) \\ &= (P_2|Q_2|R_2)^\omega + (P_2|Q_2|R_2)^*(P_3|Q_3|R_3) \end{aligned}$$

($n11$) Note that $n(\mathsf{L}) = 1$ by Theorem 34. The claim follows since, using Theorems 32.6 and 32.10,

$$\begin{aligned} &(P|Q|R)^\omega \\ &= (R^\omega \cup R^*P|R^\omega \cup R^*Q|R^\omega) \\ &= (R^*P \cup R^*R^\omega|R^*Q \cup R^*R^\omega|R^*R^\omega) \\ &= (R^*P|R^*Q|R^*)(R^\omega|R^\omega|R^\omega) \\ &= (P|Q|R)^*(R^\omega|R^\omega|R^\omega) \\ &\leq (P|Q|R)^*(R^\omega \cup R^*Q/\mathsf{T}|R^\omega \cup R^*Q/\mathsf{T}|R^\omega \cup R^*Q/\mathsf{T}) \\ &= (P|Q|R)^*((R^\omega \cup R^*Q)/\mathsf{T}|(R^\omega \cup R^*Q)/\mathsf{T}|(R^\omega \cup R^*Q)/\mathsf{T}) \\ &= (P|Q|R)^*(\mathsf{O}|\mathsf{O}|(R^\omega \cup R^*Q)/\mathsf{T} \cap \mathsf{I})(\mathsf{T}|\mathsf{T}|\mathsf{T}) \\ &= (P|Q|R)^*n(R^\omega \cup R^*P|R^\omega \cup R^*Q|R^\omega)(\mathsf{T}|\mathsf{T}|\mathsf{T}) \\ &= (P|Q|R)^*n((P|Q|R)^\omega)(\mathsf{T}|\mathsf{T}|\mathsf{T}) \end{aligned}$$

($n12$) The claim follows since $\mathsf{L}x = \mathsf{L}$ for any $x \in S$ by Theorem 34. □

The following result elaborates the binary and unary itering operations in model M8 according to Theorems 31 and 2.8.

Theorem 36. $(M_8, +, \cdot, \star, 0, 1)$ *is an extended binary itering and* $(M_8, +, \cdot, {}^\circ, 0, 1)$ *is an itering, where*

$$\begin{aligned} (P_1|Q_1|R_1) \star (P_2|Q_2|R_2) &= (R_1^*(P_1 \cup P_2)|R_1^\omega \cup R_1^*(Q_1 \cup Q_2)|R_1^*R_2) \\ (P|Q|R)^\circ &= (R^*P|R^\omega \cup R^*Q|R^*) \end{aligned}$$

Proof. By Theorem 31.3, using Theorems 32.10, 32.6 and 32.8,

$$\begin{aligned} &(P_1|Q_1|R_1) \star (P_2|Q_2|R_2) \\ &= n((P_1|Q_1|R_1)^\omega)(\mathsf{O}|\mathsf{T}|\mathsf{O}) + (P_1|Q_1|R_1)^*(P_2|Q_2|R_2) \\ &= n(R_1^\omega \cup R_1^*P_1|R_1^\omega \cup R_1^*Q_1|R_1^\omega)(\mathsf{O}|\mathsf{T}|\mathsf{O}) + (R_1^*P_1|R_1^*Q_1|R_1^*)(P_2|Q_2|R_2) \\ &= (\mathsf{O}|\mathsf{O}|(R_1^\omega \cup R_1^*Q_1)/\mathsf{T} \cap \mathsf{I})(\mathsf{O}|\mathsf{T}|\mathsf{O}) + (R_1^*P_1 \cup R_1^*P_2|R_1^*Q_1 \cup R_1^*Q_2|R_1^*R_2) \\ &= (\mathsf{O}|((R_1^\omega \cup R_1^*Q_1)/\mathsf{T} \cap \mathsf{I})\mathsf{T}|\mathsf{O}) + (R_1^*(P_1 \cup P_2)|R_1^*Q_1 \cup R_1^*Q_2|R_1^*R_2) \\ &= (R_1^*(P_1 \cup P_2)|(R_1^\omega \cup R_1^*Q_1)/\mathsf{T} \cup R_1^*Q_1 \cup R_1^*Q_2|R_1^*R_2) \\ &= (R_1^*(P_1 \cup P_2)|R_1^\omega \cup R_1^*Q_1/\mathsf{T} \cup R_1^*Q_1 \cup R_1^*Q_2|R_1^*R_2) \\ &= (R_1^*(P_1 \cup P_2)|R_1^\omega \cup R_1^*(Q_1 \cup Q_2)|R_1^*R_2) \end{aligned}$$

Elements of M_8 satisfy the property $(x \star y)z = x \star (yz)$:

$$\begin{aligned}
&((P_1|Q_1|R_1) \star (P_2|Q_2|R_2))(P_3|Q_3|R_3)\\
&= (R_1^*(P_1 \cup P_2)|R_1^\omega \cup R_1^*(Q_1 \cup Q_2)|R_1^*R_2)(P_3|Q_3|R_3)\\
&= (R_1^*(P_1 \cup P_2) \cup R_1^*R_2P_3|R_1^\omega \cup R_1^*(Q_1 \cup Q_2) \cup R_1^*R_2Q_3|R_1^*R_2R_3)\\
&= (R_1^*(P_1 \cup P_2 \cup R_2P_3)|R_1^\omega \cup R_1^*(Q_1 \cup Q_2 \cup R_2Q_3)|R_1^*R_2R_3)\\
&= (P_1|Q_1|R_1) \star (P_2 \cup R_2P_3|Q_2 \cup R_2Q_3|R_2R_3)\\
&= (P_1|Q_1|R_1) \star ((P_2|Q_2|R_2)(P_3|Q_3|R_3))
\end{aligned}$$

As a consequence Theorem 2.8 yields the itering instance $x^\circ = x \star 1$, which elaborates as stated above. □

4.3.3 Instances for non-strict computations

We instantiate n-omega algebras for the relational models M11–M12 of non-strict computations. Recall that every relation algebra is a bounded distributive lattice and an idempotent semiring. To turn a relation algebra into an n-algebra, it therefore remains to define the operation n and the constant L. For the latter we observe that L is the universal relation in models M11–M12, so we use $\mathsf{L} = \top$. We derive the operation n again from its characteristic Galois connection. Since relation algebras have left residuals we use Theorem 32.1 in

$$\begin{aligned}
& n(x) \leq n(y)\\
\Leftrightarrow\ & n(x)\mathsf{L} \leq y\\
\Leftrightarrow\ & n(x)\top \leq y\\
\Leftrightarrow\ & n(x) \leq y/\top\\
\Leftrightarrow\ & n(x) \leq y/\top \curlywedge 1
\end{aligned}$$

The last step holds because n maps relations to tests, which are elements below 1. This suggests the definition $n(x) = x/\top \curlywedge 1$ and we obtain the following result.

Theorem 37. *Let S be a relation algebra. Then S is an n-algebra using $n(x) = x/\top \curlywedge 1$ and $\mathsf{L} = \top$.*

The intuition underlying this definition of n is that the 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 assuming completeness of the underlying lattice to ensure the involved fixpoints exist or – as in the following result – including the axioms of omega algebras.

Theorem 38. *Let S be a relation algebra and a left-zero omega algebra. Then S is an n-omega algebra using $n(x) = x/\top \curlywedge 1$ and $\mathsf{L} = \top$.*

As a consequence model M11 is an n-omega algebra. Moreover it can be shown that $n(R)$ is $\preccurlyeq$-closed if R is $\preccurlyeq$-closed, whence model M12 is another instance.

4.4 Publications

An algebraic definition of the approximation order for a general-correctness model was investigated in [38]. It is based on axioms for the domain operation, the constant L and a constant that represents the computation with all finite executions. This enables the representation of least fixpoints in the approximation order in terms of least and greatest fixpoints in the semilattice order.

A unified approximation order for partial-, total- and general-correctness models was proposed in [41], again using the domain operation. It was investigated and generalised to further models in [42, 43, 48, 49].

Extending to models which independently represent finite, infinite and aborting executions brought along n-semirings in [45]. For generalising to models of non-strict computations n-algebras were proposed in [53]. Further models of n-algebras were given in [51].

Connections between least and greatest fixpoints in the semilattice and approximation orders were investigated in [43]. Connections to the median operation of lattices were investigated in [43, 52].

Chapter 5

Correctness

The previous chapters were concerned with the semantics of recursion and its special case, iteration. In particular, the algebraic approach gives a semantics of while-programs which can be used for program development using transformation and refinement as exemplified in Sections 3.2.2, 3.2.4, 3.3.2 and 4.2.3. Program reasoning is also illustrated in Section 4.1.6. As shown in Section 4.3 the algebras are unifying across a variety of computation models.

In this chapter we extend the algebraic approach to correctness statements and their calculi. Correctness statements similar to Hoare triples claim that a computation has only restricted kinds of execution, for example, that there are no aborting executions or that all finite executions end in a given set of states. Clearly such guarantees can be given only in a computation model which is precise enough to talk about the kinds of execution involved. But in general there are several computation models capable of expressing a particular statement. Vice versa, a given model might support different kinds of correctness statement as discussed in Chapter 2. Our algebraic approach is unifying in both dimensions: one statement applies in various models to various correctness claims.

Section 5.1 gives a propositional correctness calculus for the unified correctness statements and show its soundness and relative completeness. It is based on algebras which describe the semantics of while-programs via their effect on postconditions, thereby generalising the wp and wlp operators of [28]. The calculus, too, unifies different computation models and correctness statements. A technical innovation is that the loop variant or bound function is captured by a sequence of tests. We furthermore extend our unifying approach to pre-post specifications and loop refinement rules useful for program construction.

In Section 5.2 we take a step towards concrete models by instantiating the algebras of Section 5.1 in modal semirings. They are based on the domain operation, but relativised so as to capture the various correctness statements. This simplifies instantiation to the concrete models M1–M8 and M13 of Chapter 2 which is performed in Section 5.3.

In particular, our algebras also cover the multirelational computation model M13. As shown throughout the chapter, this entails considerable generalisations in the algebraic structures, changes to the correctness calculus and complications in the proof of relative completeness.

5.1 Algebras for correctness reasoning

In this section we develop an algebraic theory of correctness suitable for relational, matrix-based and multirelational computation models. We give an axiomatic description of pre-conditions and while-programs, which we use to obtain a correctness calculus. All results uniformly apply to various computation models and various correctness statements in each model as discussed in Chapter 2.

5.1.1 Preconditions

Preconditions are represented by tests as introduced in Section 3.2.3. We first discuss completeness of tests and chains. A test algebra $(S, +, \cdot, ', 0, 1)$ is *complete* if the image S' of S under $'$ is a complete Boolean algebra, that is, every set of tests $T \subseteq S'$ has a least upper bound or supremum $\sum T$ in S':

$$\sum T \in S' \qquad \forall x \in T : x \leq \sum T \qquad \forall y \in S' : (\forall x \in T : x \leq y) \Rightarrow \sum T \leq y$$

It follows that the meet operation $\cdot$ on S' distributes over suprema of tests and that every set of tests $T \subseteq S'$ has a greatest lower bound or infimum $\prod T$:

$$\prod T \in S' \qquad \forall x \in T : \prod T \leq x \qquad \forall y \in S' : (\forall x \in T : y \leq x) \Rightarrow y \leq \prod T$$

An *ascending chain* with respect to a partial order $\leq$ is a sequence x_i such that $x_i \leq x_{i+1}$ for each $i \in \mathbb{N}$, while $x_i \geq x_{i+1}$ is required for a *descending chain*.

A *precondition algebra* $(S, +, \cdot, «, ', 0, 1)$ is a test algebra $(S, +, \cdot, ', 0, 1)$ expanded with a binary operation « satisfying the axioms

$$\begin{array}{ll} x«q = (x«q)'' & xy«q = x«y«q \\ q \leq 1«q & x«pq \leq x«q \end{array}$$

for $x, y \in S$ and $p, q \in S'$. We assume that « associates to the right and has the same precedence as $+$, which is lower than that of $\cdot$.

The first axiom states that the result of « is a test, making « an operation which takes an element and a test and yields a test. This is the reason why « is right-associative. The remaining axioms express the effect of « on 1, the sequential composition of elements and the conjunction of postconditions. They are weaker than our previous axioms of [57, 47] in two respects. First, distributivity $x«pq = (x«p)(x«q)$ is replaced with $x«pq \leq x«q$ which expresses that $\lambda q.x«q$ is $\leq$-isotone: this accommodates the modal diamond operator in addition to the modal box operator as shown in Section 5.2.2. Second, previous axioms about the precondition of tests implied $q = 1«q$; these are replaced with $q \leq 1«q$ which suffices for the following development.

In many computation models, the precondition $x«q$ represents the set of states from which execution of x is guaranteed to establish postcondition q. Properties of « are recorded in the following result.

Theorem 39. *Let S be a precondition algebra and $x, y \in S$ and $p, q, r \in S'$. Then*

1. *$\lambda q.x«q$ is $\leq$-isotone*
2. *$x«pq \leq (x«p)(x«q)$*
3. *$xy«1 \leq x«1$*
4. *$x«q \leq x«1$*
5. *$x«1 \leq 1$*
6. *$p \leq x«q \wedge q \leq y«r \Rightarrow p \leq xy«r$*

For example, the last property amounts to soundness of the rule for sequential composition in the correctness calculus of Section 5.1.3.

5.1.2 While-programs

To describe the semantics of while-programs we specify their effect on postconditions similarly to [28]. This is complementary to the explicit definition of $\star$ in n-omega algebras given by Theorem 31.3 and the axiomatisation of $\star$ in binary iterings, both of which work directly with computations.

A *while-algebra* $(S, \lhd\rhd, +, \cdot, «, \star, ', 0, 1)$ is a precondition algebra $(S, +, \cdot, «, ', 0, 1)$ expanded with a ternary operation $\lhd\rhd$ and a binary operation $\star$ satisfying the axioms

$$\begin{array}{l} (x \lhd p \rhd y)«q = p(x«q) + p'(y«q) \\ (p \star x)«q = p(x«(p \star x)«q) + p'q \\ (p \star x)«q = (p \star x)«p'q \end{array}$$

for $x, y \in S$ and $p, q \in S'$. The element $x \lhd p \rhd y$ represents the conditional statement if p then x else y and the corresponding axiom characterises the two branches under a postcondition; see [67, 69, 74] for more comprehensive axiomatisations. The element $p \star x$ represents the while-loop while p do x and the second axiom describes its fixpoint unfolding, again under a postcondition. The third axiom describes that at the end of a loop its condition is false. All three axioms are equations of tests. Some of their consequences are recorded in the following result.

Theorem 40. *Let S be a while-algebra and $x, y \in S$ and $p, q, r \in S'$. Then*

1. $pq \leq x\llangle r \wedge p'q \leq y\llangle r \Rightarrow q \leq (x \lhd p \rhd y)\llangle r$
2. $p((x \lhd p \rhd y)\llangle q) = p(x\llangle q)$
3. $p'((x \lhd p \rhd y)\llangle q) = p'(y\llangle q)$
4. $p((p \star x)\llangle q) = p(x\llangle (p \star x)\llangle q)$
5. $p'((p \star x)\llangle q) = p'q$
6. $(p \star x)\llangle q \leq (x(p \star x) \lhd p \rhd 1)\llangle q$
7. $(p \star x)\llangle q \leq (x \lhd p \rhd 1)\llangle (p \star x)\llangle q$
8. $p' \leq (p \star x)\llangle p'$
9. $(p \star x)\llangle p' \leq (p \star x)\llangle 1$
10. $q \leq (p \star x)\llangle 1 \Leftrightarrow pq \leq (p \star x)\llangle 1$

For example, the first property amounts to soundness of the rule for conditionals in the correctness calculus of Section 5.1.3. The next four properties show how to reduce conditionals and while-loops if their conditions are known to hold or not to hold.

In Section 5.1.3, the test $\ell = (1 \star 1)\llangle 1$ helps us to treat claims of total correctness and claims that do not involve termination in a uniform way. The element $1 \star 1$ represents the endless loop while true do skip. It establishes the postcondition true if and only if the infinite executions are ignored. Statements which do not involve termination are thus obtained in instances with $\ell = 1$, whereas instances with $\ell = 0$ yield total correctness. In particular, a convenient way to obtain partial correctness is to add the axiom $x\llangle 1 = 1$, a characteristic property of wlp [28].

Consider a while-algebra S, a subset $A \subseteq S$ of atomic programs and a subset $T \subseteq S'$ of atomic tests. We assume that $1 \in A$ and $0 \in T$, that is, skip is an atomic program and false is an atomic test. There are no further requirements on A and T; in concrete models they typically contain basic statements – such as assignments – and basic conditions.

Test expressions are constructed from atomic tests by the operations $'$ for negation and $\cdot$ for conjunction. Hence they are tests and closed under 0, 1, finite sums and finite products. *While-programs* are constructed from atomic programs and test expressions by the operations $\cdot$ for sequential composition, $\lhd\rhd$ for conditionals and $\star$ for while-loops. Hence they are closed under 1 and finite products. *Precondition expressions* are test expressions extended by preconditions; they are constructed from test expressions and while-programs by the operations $'$ for negation, $\cdot$ for conjunction and $\llangle$ for preconditions. Hence they are tests and closed under 0, 1, ℓ, finite sums and finite products.

5.1.3 Correctness calculus

In the following we introduce correctness statements and a sound and relatively complete correctness calculus. To this end, a *correctness algebra* S is a while-algebra whose underlying test algebra is complete and which satisfies the additional axioms

$$pq \leq x\llangle q \Rightarrow \ell q \leq (p \star x)\llangle q \qquad (x\llangle q)\ell \leq x\llangle q\ell$$
$$p(x\llangle q) \leq q \Rightarrow (p \star x)\llangle q \leq q + \ell \qquad y\llangle \sum t_i = \sum y\llangle t_i$$

for $x, y \in S$ and $p, q, t_i \in S'$ such that y is a while-program and t_i is an ascending chain of tests. The first axiom is sufficient to prove soundness of the correctness calculus below, while the remaining three suffice for proving relative completeness. The axiom $(x\llangle q)\ell \leq x\llangle q\ell$ imports the constant ℓ into the postcondition and clearly holds for $\ell = 0$ or $\ell = 1$. The axiom $y\llangle \sum t_i = \sum y\llangle t_i$ states that the function $\lambda q.y\llangle q$ is continuous if y is a while-program. For

up-closed multirelations, this follows if y has bounded non-determinism as we will explain in Section 5.2.4.

A *correctness statement* $p\{x\}q$ is composed of a while-program x and precondition expressions p and q. The statement $p\{x\}q$ is *valid* if and only if $p \leq x\mathbin{«} q$. Its meaning depends on the model and the interpretation of the precondition operation «. In some models, $p \leq x\mathbin{«} q$ amounts to partial correctness, that is, all finite executions of x starting in p establish the postcondition q. In other models, $p \leq x\mathbin{«} q$ amounts to total correctness, which additionally requires that all executions of x starting in p are finite. In yet other models, $p \leq x\mathbin{«} q$ requires that no execution of x starting in p aborts.

To *derive* correctness statements, we use a calculus with the following rules, for atomic program z, while-programs x and y, test expression p, precondition expressions q, r, s and t, and tests t_i:

$$\text{(atom)}\quad \frac{}{z\mathbin{«} q\{z\}q}$$

$$\text{(seq)}\quad \frac{q\{x\}r \qquad r\{y\}s}{q\{xy\}s}$$

$$\text{(cond)}\quad \frac{pq\{x\}r \qquad p'q\{y\}r}{q\{x \lhd p \rhd y\}r}$$

$$\text{(while)}\quad \frac{q \leq t_{<\infty} \qquad t_0pq\{x\}\ell q \qquad \forall n > 0 : t_npq\{x\}t_{<n}q}{q\{p \star x\}p'q}$$

$$\text{(cons)}\quad \frac{q \leq r \qquad r\{x\}s \qquad s \leq t}{q\{x\}t}$$

Rule (while) is abstracted from the Hoare calculus for total correctness [2], but a change has to be made to accommodate multirelational models. For $n \in \mathbb{N} \cup \{\infty\}$, the test $t_{<n}$ is defined by $t_{<n} = \sum_{0 \leq i < n} t_i$. If $\ell = 0$, the sequence of tests t_i describes the bound function; each test t_n represents a set of states from which the loop terminates after at most n iterations. The inequality $q \leq t_{<\infty}$ expresses that the bound is non-negative while the invariant q holds. By $t_npq\{x\}t_{<n}q$ every iteration decreases the bound and preserves the loop invariant q at the same time. These two tasks cannot be separated as in previous calculi [2, 47] because « is not distributive so as to capture multirelational models. The triple $t_0pq\{x\}\ell q$ makes sure that the iteration terminates. If $\ell = 1$, the premises simplify to $pq\{x\}q$ by setting $t_i = q$, which expresses that the loop invariant q is preserved by the loop body for partial-correctness statements. Rule (while) concludes that the invariant is preserved by the while-loop.

A related rule for while-programs appears in [4]. It can be used for showing total correctness, but not partial correctness. In that rule, invariant and bound function are conflated, for which the authors quote practical reasons. Our investigation suggests that this is necessary due to the multirelational model.

The following result shows that our calculus is sound and complete. As usual, completeness is relative to having all true inequalities $p \leq q$ available in the calculus. We sketch the proof and provide some technical details because the support for multirelations substantially increases its complexity.

Theorem 41. *In a correctness algebra, the above calculus is sound and complete, that is, all valid and only valid correctness statements can be derived.*

Proof sketch. The additional difficulty arises because decreasing the bound and preserving the loop invariant are not separated. Technically, the triple $t_npq\{x\}t_{<n}q$ implies but is not

equivalent to the conjunction of $t_n pq\{x\}t_{<n}$ and $t_n pq\{x\}q$. Namely, in terms of preconditions we have

$$t_n pq \leq x \ll t_{<n} q \leq (x \ll t_{<n})(x \ll q)$$

by Theorem 39.2, but the latter inequality is not an equality in general.

Because of the inequality, the triple $t_n pq\{x\}t_{<n}q$ implies both $t_n pq\{x\}t_{<n}$ and $t_n pq\{x\}q$. This means that the soundness part is not affected and can be adapted from our earlier setting, which did not take into account multirelational models [47]. First, $pq \leq x \ll q$ is proved from the premises of rule (while). Then $t_n q \leq (p \star x) \ll p'q$ and $t_{<n} q \leq (p \star x) \ll p'q$ are proved simultaneously by induction on n. The first axiom of correctness algebras is used in the base case.

Completeness is proved by induction over while-programs and the difficulty arises for while-loops, in which case the triple $w\{p \star x\}p'w$ must be derived, where $w = (p \star x) \ll q$. This is done by applying rule (while) of the calculus, for which we invent the following bound function given by the sequence t_n:

$$\begin{aligned} t_n &= f^n(p' + (x \ll w\ell)) \\ f(r) &= p' + (x \ll wr) \end{aligned}$$

Its main feature is that the invariant w is wired into each unfolding of the function f, because it can no longer be established separately from decreasing the bound. This makes it relatively easy to establish the triples required for applying rule (while), but complicates proof of the remaining assumption $w \leq t_{<\infty}$. For this, we invent a second bound function given by the sequence s_n:

$$\begin{aligned} s_n &= g^n(p'q + p(x \ll w\ell)) \\ g(r) &= p'q + p(x \ll wr) \end{aligned}$$

Again the invariant w is wired into each unfolding of the function g. Because $s_n \leq t_n$ for each $n \in \mathbb{N}$, it remains to show $w \leq s_{<\infty}$. For this, we observe that $w\ell \leq s_0 \leq s_{<\infty}$ by unfolding w and importing ℓ into the postcondition, whence it suffices to show $w \leq s_{<\infty} + \ell$. By a consequence of two correctness algebra axioms, this reduces to $p(x \ll s_{<\infty}) + p'q \leq s_{<\infty}$ and hence to $p(x \ll s_{<\infty}) \leq s_{<\infty}$. Now s_n is an ascending chain by induction, again using $w\ell \leq s_0$. We therefore apply continuity of $\ll$ and are left with $p(x \ll s_i) \leq s_{<\infty}$ for each $i \in \mathbb{N}$. But this follows since $s_i \leq w$ holds, which implies

$$p(x \ll s_i) = p(x \ll ws_i) \leq p'q + p(x \ll ws_i) = g(s_i) = s_{i+1} \leq s_{<\infty}$$

Note that $t_i \leq w$ does not hold, so this argument cannot be made using the bound function t_n instead of s_n. Moreover, the bound function s_n cannot be used instead of t_n to establish the triples required for applying rule (while). Thus the main idea and difficult part is to carefully craft two bound functions $s_n \leq t_n$ such that s_n applies to one assumption of rule (while) and t_n applies to the remaining two assumptions. □

Our calculus unifies and generalises previous algebraic calculi for partial, total and general correctness [78, 87, 88, 38, 41, 57, 47]. In particular, it applies to further computation models, including multirelations.

Some axioms of precondition algebras and while-algebras are necessary to prove soundness and completeness of the above calculus. Instantiating $r = y \ll s$ and $q = x \ll r = x \ll y \ll s$ in rule (seq) implies $x \ll y \ll s \leq xy \ll s$ by Theorem 41. Likewise, instantiating $q = r = x \ll s$ in rule (cons) shows $s \leq t \Rightarrow x \ll s \leq x \ll t$ and hence $x \ll st \leq x \ll t$. Furthermore, if the triple $q\{1\}q$ is assumed to be valid, as expected if 1 represents the skip computation, we obtain $q \leq 1 \ll q$. Instantiating $q = p(x \ll r) + p'(y \ll r)$ in rule (cond) implies $p(x \ll r) + p'(y \ll r) \leq (x \lhd p \rhd y) \ll r$. Whether the remaining inequalities are necessary is an open question.

5.1.4 Applications: games and integer division

As an example for applying the calculus in a multirelational setting we revisit a simplified version of the Nim game [5, 4, 80]. A pile of matches is given, from which two players take turns in removing either one or two. The last player to remove a match loses. The number of remaining matches is the state of the game and given by the value of the variable $v \in \mathbb{N}$. The first player's move P_1, the second player's move P_2 and their composition $R = P_1 \,;\, P_2$ are described by the following three multirelations:

P_1			P_2			R		
0	$\mapsto$	$2^{\mathbb{N}}$	0	$\mapsto$	$\emptyset$	0	$\mapsto$	$2^{\mathbb{N}}$
1	$\mapsto$	$\{0\}^\uparrow$	1	$\mapsto$	$\{0\}^\uparrow$	1	$\mapsto$	$\emptyset$
2	$\mapsto$	$\{0\}^\uparrow \cup \{1\}^\uparrow$	2	$\mapsto$	$\{0,1\}^\uparrow$	2	$\mapsto$	$\{0\}^\uparrow$
3	$\mapsto$	$\{1\}^\uparrow \cup \{2\}^\uparrow$	3	$\mapsto$	$\{1,2\}^\uparrow$	3	$\mapsto$	$\{0\}^\uparrow$
4	$\mapsto$	$\{2\}^\uparrow \cup \{3\}^\uparrow$	4	$\mapsto$	$\{2,3\}^\uparrow$	4	$\mapsto$	$\{0,1\}^\uparrow \cup \{1,2\}^\uparrow$
5	$\mapsto$	$\{3\}^\uparrow \cup \{4\}^\uparrow$	5	$\mapsto$	$\{3,4\}^\uparrow$	5	$\mapsto$	$\{1,2\}^\uparrow \cup \{2,3\}^\uparrow$
$\vdots$			$\vdots$			$\vdots$		

The image $2^{\mathbb{N}}$ means that the first player wins, while $\emptyset$ means that the second player wins. The game is specified by the while-loop $\mathsf{while\ true\ do}\ R$, which is algebraically represented by the element $1 \star R$. We show that a win for the first player can be guaranteed from a state v if $v \bmod 3 \neq 1$. Let the multirelation q describe these states as a test:

q		
0	$\mapsto$	$\{0\}^\uparrow$
1	$\mapsto$	$\emptyset$
2	$\mapsto$	$\{2\}^\uparrow$
3	$\mapsto$	$\{3\}^\uparrow$
4	$\mapsto$	$\emptyset$
5	$\mapsto$	$\{5\}^\uparrow$
$\vdots$		

Consider the sequence of tests $t_n = (v = n)$ for which $t_{<\infty} = 1$. Then $t_n q\{R\} t_{<n} q$ holds for every $n > 0$, that is, q is an invariant of R and the bound t is decreased by R at the same time. Moreover, $t_0 q\{R\} 0$ holds since R maps state 0 to $2^{\mathbb{N}}$. We apply the total-correctness instance of rule (while), where $\ell = 0$ and $p = 1$ and $x = R$. The conclusion $q\{1 \star R\}0$ shows that the first player can guarantee a win by establishing postcondition false from starting states in q, whence the images of $1 \star R$ in these states must be $2^{\mathbb{N}}$.

In our second example we prove correctness of a program for integer division along the lines of [2], namely

$$(q, r := 0, x) \,;\, \mathsf{while}\ (r \geq y)\ \mathsf{do}\ (q, r := q + 1, r - y)$$

with four variables q, r, x, y ranging over $\mathbb{N}$. It computes the quotient q and the remainder r of the division of x by y. Using tests p_1, p_2, p_3, t_n and assignments z_1, z_2 defined by

$$\begin{array}{lll} p_1 = (y > 0) & p_3 = (r \geq y) & z_1 = (q, r := 0, x) \\ p_2 = (x = q \times y + r) & t_n = (r = n) & z_2 = (q, r := q + 1, r - y) \end{array}$$

the program is abstractly expressed as $z_1(p_3 \star z_2)$ and the following correctness statements hold:

1. $p_1\{z_1\}p_1 p_2$ since z_1 does not affect y and $r = 0 \times y + r$ holds,
2. $t_0 p_3 p_1\{z_2\}\ell p_1 p_2$ since $r = 0$ and $r \geq y > 0$ imply that the precondition is false, and

3. $t_n p_3 p_1 p_2 \{z_2\} t_{<n} p_1 p_2$ for each $n > 0$ since $n = r \geq y > 0$ implies $r - y = n - y < n$, and z_2 does not affect y, and $x = q \times y + r$ implies $x = (q+1) \times y + (r - y)$; more precisely, the postcondition $t_{n-y} p_1 p_2$ is established.

Furthermore $t_{<\infty} = 1$, whence we derive

$$\dfrac{p_1\{z_1\}p_1p_2 \qquad \dfrac{p_1p_2 \leq t_{<\infty} \qquad \dfrac{t_0p_3p_1\{z_2\}\ell p_1p_2}{t_0p_3p_1p_2\{z_2\}\ell p_1p_2} \qquad \forall n > 0 : t_np_3p_1p_2\{z_2\}t_{<n}p_1p_2}{p_1p_2\{p_3 \star z_2\}p_3'p_1p_2}}{\dfrac{p_1\{z_1(p_3 \star z_2)\}p_3'p_1p_2}{p_1\{z_1(p_3 \star z_2)\}p_3'p_2}}$$

using the loop invariant p_1p_2 and the bound function t_n. Hence the precondition $y > 0$ suffices to establish the postcondition $x = q \times y + r$ and $r < y$, by which q is the quotient and r is the remainder of the division of x by y.

At the same time, this derivation establishes total correctness: the program terminates when started in a state with $y > 0$. Moreover, it establishes that the program does not abort when started in such a state. These consequences hold because the assumed correctness statements and the derivation are valid in all computation models of Chapter 2 and for any $\mathsf{Z} \in \{0, \mathsf{L}, \mathsf{A}, \mathsf{L} + \mathsf{A}\} \setminus \{\top\}$, where the constant Z contains the executions that are ignored by a correctness statement as described in Section 5.2.1.

5.1.5 Pre-post specifications

Consider the correctness statement $p\{x\}q$. For given x and q, the test $x \mathbin{«} q$ is the greatest precondition that suffices to establish the postcondition q; all tests p with $p \leq x \mathbin{«} q$ are sufficient, too. Another viewpoint is obtained for given p and q: the pre-post specification $p \dashv q$ is the greatest computation for which p suffices to establish q; all computations x with $x \leq p \dashv q$ satisfy $p \leq x \mathbin{«} q$ as well [84, 90, 89, 108]. Pre-post specifications can therefore be introduced by a Galois connection.

A *pre-post algebra* is an algebraic structure $(S, +, \cdot, «, \dashv, ', 0, 1, \top)$ such that the reduct $(S, +, \cdot, 0, 1, \top)$ is a bounded idempotent left semiring, the reduct $(S, +, \cdot, «, ', 0, 1)$ is a precondition algebra, and the operation $\dashv$ satisfies

$$x \leq p \dashv q \;\Leftrightarrow\; p \leq x « q$$

for $x \in S$ and $p, q \in S'$. This axiom is an order-reversing Galois connection between S and the set of tests S'. The precedence of $\dashv$ is the same as that of $«$. The following result captures properties of pre-post specifications.

Theorem 42. *Let S be a pre-post algebra and $x, y \in S$ and $p, q, r, s \in S'$. Then*

1. $\lambda x.x « q$ *is* $\leq$*-antitone*
2. $\lambda p.p \dashv q$ *is* $\leq$*-antitone*
3. $\lambda q.p \dashv q$ *is* $\leq$*-isotone*
4. $(x + y) « q = (x « q) \cdot (y « q)$
5. $x \leq (x « q) \dashv q$
6. $p \leq (p \dashv q) « q$
7. $(1 \dashv q) « q = 1 \leq p \dashv p$
8. $0 \dashv q = \top$
9. $q \leq r \Rightarrow (p \dashv q)(r \dashv s) \leq p \dashv s$
10. $(p \dashv q)(q \dashv r) \leq p \dashv r$
11. $(p \dashv p)(p \dashv q) = p \dashv q$
12. $(p \dashv q)(q \dashv q) = p \dashv q$
13. $(p \dashv p)(p \dashv p) = p \dashv p$
14. $x « 1 = 1 \Leftrightarrow x \leq 1 \dashv 1$

5.1.6 Application: introduction of while-loops

The following result applies pre-post specifications to introduce while-loops in program refinement. It applies to models M1–M8 and M13 of Chapter 2.

Theorem 43. *Let S be a pre-post algebra and a correctness algebra. Let $x \in S$ and $p, q, r, t_i \in S'$. Then*

1. $q \le t_{<\infty} \;\wedge\; x \le t_0 pq{\dashv}\ell q \;\wedge\; (\forall n > 0 : x \le t_n pq{\dashv}t_{<n} q) \;\Rightarrow\; p \star x \le q{\dashv}p'q$
2. $r \le t_{<\infty} \;\wedge\; (\forall n \in \mathbb{N} : x \le t_n p{\dashv}t_{<n}) \;\Rightarrow\; p \star x \le r{\dashv}1$

If $\ell = 1$, then

3. $x \le pq{\dashv}q \;\Rightarrow\; p \star x \le q{\dashv}p'q$

Theorem 43.1 introduces a while-loop by refining a pre-post specification. It is a translation of the correctness rule of while-loops. We describe two instances in computation model M5. First, Theorem 43.3 is a partial-correctness rule obtained by setting $t_n = 1$ for each $n \in \mathbb{N}$ and $\mathsf{Z} = \top 0$, whence $\ell = 1$ by Theorem 47.18. Second, Theorem 43.2 is a total-correctness rule, which follows by setting $q = t_{<\infty}$ and $\mathsf{Z} = 0$, whence $\ell = 0$ in model M5.

Both instances are obtained in the same model, with different values of Z, and therefore apply to the same while-loop $p \star x$. This achieves a separation of the invariant q and the termination condition r as advocated by [38, 31]. Moreover, by using $\mathsf{Z} = \mathsf{L}$ a further separation can be obtained to specify the states from which the execution of the loop does not abort independently of q and r.

5.2 Modal semirings

Correctness statements in the computation models of Chapter 2 have the form $p \cdot R \cdot q' \le \mathsf{Z}$ for a constant Z. The constant Z captures executions that are to be ignored in the correctness statement. In models M1–M2 we have $\mathsf{Z} = 0$; statements of this special form $p \cdot R \cdot q' \le 0$ are well known in semirings with tests [78] or with a domain operation [23, 87]. In this section we generalise domain semirings to be able to encode statements of the form $p \cdot R \cdot q' \le \mathsf{Z}$ for various values of Z.

In Section 5.2.1 we relativise the domain operation so as to ignore executions contained in Z. Semirings extended by a domain operation are also called modal semirings [23, 87] because they allow the definition of modal diamond and box operators. In Section 5.2.2 we show that the relativised modal semirings instantiate the various algebras for correctness introduced in Section 5.1. In Section 5.2.3 we further extend modal semirings by a general iteration operator to define the semantics of while-programs for relational, matrix-based and multirelational computation models in a uniform way.

5.2.1 Relative domain

In relational computation models the domain $d(x)$ of the computation x is a test representing the set of states from which x has executions. Its Boolean complement, the antidomain $a(x)$, represents the set of states from which x has no executions. In contrast to the operation n of Chapter 4, the domain operation captures all executions, not just the infinite ones. The operations d and a satisfy the characteristic properties

$$\begin{array}{lcl} x \le d(y) \cdot x & \Leftrightarrow & d(x) \le d(y) \\ a(y) \cdot x \le 0 & \Leftrightarrow & a(y) \le a(x) \end{array}$$

which are at the centre of our generalisation. By Boolean algebra, $d(x) \le d(y)$ holds if and only if $a(y) \le a(x)$ does. According to the first equivalence, $d(x)$ is the least test p such

that $x \leq p \cdot x$, that is, all executions of x start in p. According to the second equivalence, $a(x)$ is the greatest test p such that $p \cdot x \leq 0$, that is, x has no executions starting in p. We generalise these properties as follows, taking the operations d and a relative to an element Z which captures executions that are ignored:

$$\begin{array}{rcl} x \leq d(y) \cdot x + \mathsf{Z} & \Leftrightarrow & d(x) \leq d(y) \\ a(y) \cdot x \leq \mathsf{Z} & \Leftrightarrow & a(y) \leq a(x) \end{array}$$

Hence $d(x)$ is the least test p such that all executions of x start in p, except those executions that are in Z. This means that the executions of x that are in Z are ignored in the calculation of the domain. Similarly, $a(x)$ is the greatest test p such that x has no executions starting in p, except perhaps executions in Z. Setting $\mathsf{Z} = 0$ gives the characterisations of the usual domain and antidomain operations.

Because left distributivity fails for up-closed multirelations, we work in idempotent left semirings. When instantiating an idempotent left semiring with multirelations, the operation $+$ may represent angelic or demonic choice as we will show in Section 5.3. We use an algebra with just one choice operation because it should also capture models M1–M8, which offer only a single kind of choice. This is consistent, for example, with general refinement algebras [108]. The algebras of monotonic Boolean transformers [97] are an example with two choice operations.

A *relative domain semiring* is an algebraic structure $(S, +, \cdot, d, 0, 1, \mathsf{Z})$ such that the reduct $(S, +, \cdot, 0, 1)$ is an idempotent left semiring and the axioms

$$\begin{array}{lll} d(\mathsf{Z}) = 0 & d(x + y) = d(x) + d(y) & x \leq d(x)x + \mathsf{Z} \\ d(x) \leq 1 & d(d(x)y) = d(x)d(y) & d(xy) = d(x\, d(y)) \end{array}$$

are satisfied. Counterexamples generated by Nitpick or Mace4 show that none of these axioms follows from the remaining ones and the semiring axioms. Setting $\mathsf{Z} = 0$ gives the domain semiring axioms of [27], in which case $d(d(x)y) = d(x)d(y)$ follows from the remaining axioms. The following result records properties of the domain operation.

Theorem 44. *Let $(S, +, \cdot, d, 0, 1, \mathsf{Z})$ be a relative domain semiring and let $x, y \in S$. Then $(d(S), +, \cdot, 0, d(1))$ is a bounded distributive lattice with least element 0 and greatest element $d(1)$ and the following properties hold:*

1. *d is $\leq$-isotone*
2. $d(0) = 0$
3. $d(d(x)) = d(x)$
4. $d(xy) \leq d(x)$
5. $d(x) \leq d(1)$
6. $\mathsf{Z}x \leq \mathsf{Z}$
7. $x + \mathsf{Z} = d(x)x + \mathsf{Z}$
8. $d(x) = 0 \Leftrightarrow x \leq \mathsf{Z}$
9. $xy \leq \mathsf{Z} \Leftrightarrow x\, d(y) \leq \mathsf{Z}$
10. $d(x)y \leq z \Leftrightarrow d(x)y \leq d(x)z$
11. $d(x)y \leq y\, d(z) \Leftrightarrow d(x)y = d(x)y\, d(z)$
12. $x \leq d(y)x + \mathsf{Z} \Leftrightarrow d(x) \leq d(y)$

From property 7 we obtain $1 + \mathsf{Z} = d(1) + \mathsf{Z}$, but $d(1) = 1$ does not hold in general. Property 8 shows that precisely the computations below Z have an empty relative domain as all their executions are ignored. Property 12 is the characteristic property of relative domain mentioned above.

Each of the computation models M1–M8 of Chapter 2 is a relative domain semiring, where Z is any one of the values 0, L, A or $\mathsf{L} + \mathsf{A}$ available in the model. Here L represents the computation with all infinite executions and A represents the computation which has every aborting execution. In these models the relative domain $d(x)$ is given by first omitting the executions of x that are in Z and then taking the usual domain. Moreover, model M13 is a relative domain semiring where $\mathsf{Z} = 0$.

A *relative antidomain semiring* is a structure $(S, +, \cdot, a, d, 0, 1, \mathsf{Z})$ such that the reduct $(S, +, \cdot, 0, 1)$ is an idempotent left semiring, $d(x) = a(a(x))$ and the axioms

$$\begin{array}{lll} a(\mathsf{Z}) = 1 & a(x+y) = a(x)a(y) & a(x)x \leq \mathsf{Z} \\ a(x)d(x) = 0 & a(d(x)y) = a(x) + a(y) & a(xy) = a(x\,d(y)) \end{array}$$

are satisfied. Counterexamples generated by Nitpick or Mace4 show that none of these axioms follows from the remaining ones and the semiring axioms. Setting $\mathsf{Z} = 0$ gives axioms which are equivalent to the antidomain axioms of Boolean domain semirings [27]. The following result shows, in particular, that tests and their Boolean complements can be represented by domain elements $d(x)$ and their antidomain $a(x)$.

Theorem 45. *Let $(S, +, \cdot, a, d, 0, 1, \mathsf{Z})$ be a relative antidomain semiring and $x, y, z \in S$. Then*

1. *$(S, +, \cdot, a, 0, 1)$ is a test algebra with $S' = d(S)$*
2. *$(S, +, \cdot, d, 0, 1, \mathsf{Z})$ is a relative domain semiring*
3. *$(a(S), +, \cdot, a, 0, 1)$ is a Boolean algebra with complement a*
4. *$y\,a(z) \leq a(x)y \Leftrightarrow y\,a(z) = a(x)y\,a(z) \Leftrightarrow d(x)y\,a(z) = 0$*

and

5. *$a(S) = d(S)$*
6. *$d(a(x)) = a(d(x)) = a(x)$*
7. *$a(x) + d(x) = 1$*
8. *$a(x) \leq a(xy)$*
9. *$a(x) = 1 \Leftrightarrow x \leq \mathsf{Z}$*
10. *$a(y)x \leq \mathsf{Z} \Leftrightarrow a(y) \leq a(x)$*

The last property is the characteristic property of antidomain mentioned above. A property that does not follow is left distributivity for domain elements $d(x)(y+z) = d(x)y+d(x)z$. For example, it would imply the shunting rule $d(x)y \leq z \Leftrightarrow y \leq z + a(x)\top$ if the greatest element $\top$ exists. Because left distributivity of tests holds for up-closed multirelations, it could be added as an axiom without affecting the validity of the models of Chapter 2.

Using the Boolean complement of the relative domain, each of the models M1–M8 forms a relative antidomain semiring where Z is any of the values 0, L, A or $\mathsf{L} + \mathsf{A}$ available in the model and different from $\top$. Moreover, model M13 is a relative antidomain domain semiring where $\mathsf{Z} = 0$.

In Chapter 2 we have shown that many correctness statements in relational and matrix-based models take the form $p \cdot R \cdot q' \leq \mathsf{Z}$ using tests p and q. In a relative antidomain semiring such statements take the form $d(x)y\,a(z) \leq \mathsf{Z}$ where $d(x)$, y and $d(z)$ play the roles of p, R and q, respectively. By Theorem 45.10 this is equivalent to $d(x) \leq a(y\,a(z))$. On the other hand, correctness statements for up-closed multirelations take the form $d(x) \leq d(y\,d(z))$. In the following we show that these different statements can be unified.

5.2.2 Relative modal operators

Preconditions such as those in correctness statements can be expressed by modal diamond and box operators. These are defined in terms of the domain and antidomain operations. In this section we generalise them to relative modal operators.

In a relative domain semiring the binary *diamond* operator is defined by $|x\rangle y = d(xy)$, which is the same as $d(x\,d(y))$. This means that its second argument is effectively a test p, and $|x\rangle p$ represents the set of states from which x has an execution that is not in Z and leads to a state in p. The diamond operator satisfies many properties known from the unrelativised setting as the following result shows.

Theorem 46. *Let S be a relative domain semiring and $x, y, z \in S$ and $p, q \in d(S)$. Then*

1. $\lambda y.|x\rangle y$ *and* $\lambda x.|x\rangle y$ *are* $\leq$*-isotone*
2. $|x + y\rangle z = |x\rangle z + |y\rangle z$
3. $|x\rangle y + |x\rangle z \leq |x\rangle(y + z)$
4. $|xy\rangle z = |x\rangle(yz) = |x\rangle|y\rangle z$
5. $|x\rangle(pq) \leq |x\rangle p \cdot |x\rangle q$
6. $|px\rangle y = p|x\rangle y = p|px\rangle y$
7. $|xp\rangle q = |xp\rangle(pq)$
8. $p \leq |x\rangle y \Leftrightarrow p \leq |px\rangle y$
9. $pq \leq |x\rangle y \Leftrightarrow pq \leq |qx\rangle y$
10. $|p\rangle q = pq$
11. $|p\rangle 0 = |0\rangle y = 0$
12. $|p\rangle 1 = |p\rangle p = |1\rangle p = p$
13. $p|x\rangle q \leq \mathsf{Z} \Leftrightarrow pxq \leq \mathsf{Z}$
14. $|x\rangle q \leq p \Leftrightarrow xq \leq px + \mathsf{Z}$

For example, the last two properties show how to eliminate the diamond operator on the left-hand side of an inequality, while $|px\rangle y = p|x\rangle y = p|px\rangle y$ shows that tests can be imported to and exported from diamonds. Because left distributivity does not hold in idempotent left semirings, we no longer have $|x\rangle(y + z) = |x\rangle y + |x\rangle z$.

In a relative antidomain semiring the dual *box* operator is defined by $|x]y = a(x\, a(y))$. Again its second argument is effectively a test p, and $|x]p$ represents the set of states from which all executions of x that are not in Z lead to p. Also the box operator satisfies many properties known from the unrelativised setting. Moreover, as shown in the following result, both box and diamond can express preconditions.

Theorem 47. *Let S be a relative antidomain semiring and $x, y, z \in S$ and $p, q \in d(S)$. Then S is a precondition algebra with $x\mathbin{«} q = |x]q$ and a precondition algebra with $x\mathbin{«} q = |x\rangle q$ and*

1. $|x]y = a(|x\rangle a(y))$
2. $|x\rangle y = a(|x]a(y))$
3. $\lambda y.|x]y$ *is* $\leq$*-isotone*
4. $\lambda x.|x]y$ *is* $\leq$*-antitone*
5. $|x + y]z = |x]z \cdot |y]z$
6. $|xy]z = |x]|y]z$
7. $|x](pq) \leq |x]p \cdot |x]q$
8. $|px]y = a(p) + |x]y$
9. $p|x]y = p|px]y$
10. $|xp]q = |xp](pq)$
11. $p \leq |x]y \Leftrightarrow p \leq |px]y$
12. $pq \leq |x]y \Leftrightarrow pq \leq |qx]y$
13. $|p]q = a(p) + q$
14. $|p]0 = a(p)$
15. $|p]1 = |p]p = |0]y = 1$
16. $|1]q = q \leq |p]q$
17. $(|x]y)x\, a(y) \leq \mathsf{Z}$
18. $|x]1 = 1 \Leftrightarrow x0 \leq \mathsf{Z}$
19. $|x\rangle q \leq p \Leftrightarrow a(p)xq \leq \mathsf{Z}$
20. $p \leq |x]q \Leftrightarrow px\, a(q) \leq \mathsf{Z}$

Property 9 shows how to import a test into a box. For another example, property 17 states that in a state in $|x]y$, where all executions that are not in Z lead to $d(y)$, there are no executions to $a(y)$ except perhaps those in Z. The last two properties give ways to eliminate diamond and box. The last property applies to correctness statements. Again because idempotent left semirings lack left distributivity, we no longer have $|x](pq) = |x]p \cdot |x]q$. We also lose the demodalisation property $p \leq |x]q \Rightarrow px \leq xq + \mathsf{Z}$. As explained in Chapter 2, the correctness statement for relational and matrix-based models is formalised by $p \leq |x]q$, and the correctness statement for multirelational models is formalised by $p \leq |x\rangle q$.

It is known that the box operator corresponds to wlp in model M1 [87] and to wp in model M3 [88] and in model M2 [85]. These cases are captured by using $\mathsf{Z} = 0$ in our setting.

We furthermore find that

- with $\mathsf{Z} = 0$ box corresponds to a variant of wp, which avoids aborting executions in addition to infinite ones, in models M4–M5,
- with $\mathsf{Z} = \mathsf{L}$ box corresponds to wlp in model M3, and to a variant of wlp, which avoids aborting executions, in models M4–M5,
- with $\mathsf{Z} = \mathsf{A}$ box corresponds to wp in model M5,
- with $\mathsf{Z} = \mathsf{L} + \mathsf{A}$ box corresponds to wlp in model M5.

Similar variants of wp are observed in [59, 60] and related to different execution methods without a unified treatment; see [91] for variants of wlp.

A relative antidomain semiring is *complete* if its test algebra according to Theorem 45.1 is complete and $a(\sum x_i) = \prod a(x_i)$ for every ascending chain x_i and $a(\prod y_i) = \sum a(y_i)$ for every descending chain y_i. It follows that d distributes over suprema of ascending chains and infima of descending chains.

5.2.3 Iteration

We now extend relative modal semirings by iteration operations. A *modal Conway semiring* is a structure $(S, +, \cdot, a, d, {}^\circ, 0, 1, \mathsf{Z})$ such that the reduct $(S, +, \cdot, a, d, 0, 1, \mathsf{Z})$ is a relative antidomain semiring and the reduct $(S, +, \cdot, {}^\circ, 0, 1)$ is a left Conway semiring as introduced in Section 3.4. A *modal Conway algebra* is a structure $(S, +, \cdot, a, d, {}^*, {}^\circ, 0, 1, \mathsf{Z})$ such that the reduct $(S, +, \cdot, a, d, {}^\circ, 0, 1, \mathsf{Z})$ is a modal Conway semiring and the reduct $(S, +, \cdot, {}^*, 0, 1)$ is a bounded left Kleene algebra as defined in Section 3.1.3.

We give several multirelational instances of these structures in Section 5.3. As the following result shows, the operations $\lhd\rhd$ and $\star$ of while-algebras can be defined in modal Conway semirings, giving a unified semantics of while-programs.

Theorem 48. *Let S be a modal Conway semiring. Then S is a while-algebra with*

1. $x \lhd p \rhd y = px + a(p)y$ *and* $p \star x = (px)^\circ a(p)$ *and* $x \mathbin{«} p = |x\rangle p$, *or*
2. $x \lhd p \rhd y = px + a(p)y$ *and* $p \star x = (px)^\circ a(p)$ *and* $x \mathbin{«} p = |x]p$.

These definitions use $x, y \in S$ and $p \in d(S)$.

5.2.4 Correctness

The following result shows how to obtain a sound and complete correctness calculus in modal Conway algebras subjected to additional conditions.

Theorem 49. *Let S be a modal Conway algebra – which is a test algebra, a precondition algebra and a while-algebra according to Theorems 45, 47 and 48 – with a complete relative antidomain semiring, $\ell \in \{0, 1\}$ and $x(y + \mathsf{Z}) \leq xy + \mathsf{Z}$ for each $x, y \in S$. Then the following hold.*

1. *Let $x^\circ = x^*$ for each $x \in S$ and $y \sum t_i = \sum y t_i$ for every while-program y and ascending chain of tests t_i. Then S is a correctness algebra using $x \mathbin{«} q = |x\rangle q$.*
2. *Let $\ell x^\circ \leq x^*$ and $p|x]q \leq q \Rightarrow |x^\circ]p \leq q + \ell$ for each $x \in S$ and $p, q \in d(S)$. Let $y \prod t_i = \prod y t_i$ for every while-program y and descending chain of tests t_i. Then S is a correctness algebra using $x \mathbin{«} q = |x]q$.*

In both cases, Theorem 41 yields a sound and complete correctness calculus.

The property $x(y+\mathsf{Z}) \leq xy+\mathsf{Z}$ holds for multirelational models where $\mathsf{Z}=0$ and for relational and matrix-based models which satisfy left distributivity and $x\mathsf{Z} \leq x0+\mathsf{Z}$ [47]. The property $p|x]q \leq q \Rightarrow |x^\circ]p \leq q+\ell$ is similar to the induction axiom for the convergence operation of [88]. For up-closed multirelations, the continuity property $y\sum t_i = \sum yt_i$ holds if the demonic choices in y are finite [94]; see [82] for continuity in probabilistic Kleene algebras. A related argument shows that $y\prod t_i = \prod yt_i$ holds for an up-closed multirelation y if the angelic choices in y are finite.

5.3 Instances

In this section we instantiate the algebras of Section 5.2 to demonstrate the range of computation models covered by our theory of correctness. We first recall the algebras of monotonic Boolean transformers [97] that capture up-closed multirelations and play a prominent role below.

An *algebra of monotonic Boolean transformers* is a structure $(S, \sqcup, \sqcap, \cdot, ^{\mathsf{d}}, ^{\omega}, \bot, \top, 1)$ such that the reduct $(S, \sqcup, \sqcap, \bot, \top)$ is a bounded distributive lattice, the reduct $(S, \cdot, 1)$ is a monoid and the axioms

$$\begin{array}{lll} (x \sqcap y)\cdot z = (x\cdot z)\sqcap(y\cdot z) & x^{\mathsf{dd}} = x & (x\cdot\top)\sqcap(x^{\mathsf{d}}\cdot\bot) = \bot \\ \top\cdot x = \top & (x\cdot y)^{\mathsf{d}} = x^{\mathsf{d}}\cdot y^{\mathsf{d}} & x\cdot x^{\omega}\sqcap 1 = x^{\omega} \\ x\leq y \Rightarrow z\cdot x\leq z\cdot y & x\leq y \Leftrightarrow y^{\mathsf{d}}\leq x^{\mathsf{d}} & x\cdot z\sqcap y\leq z \Rightarrow x^{\omega}\cdot y\leq z \end{array}$$

are satisfied, where $x \leq y \Leftrightarrow x \sqcup y = y \Leftrightarrow x \sqcap y = x$ is the lattice order. It follows that both $(S, \sqcup, \cdot, \bot, 1)$ and $(S, \sqcap, \cdot, \top, 1)$ are idempotent left semirings. An *assertion* is an element $p \in S$ such that $p = p\cdot\top\sqcap 1$. An *assumption* is the dual p^{d} of an assertion p. The operations $\sqcup$ and $\sqcap$ have the same precedence, which is lower than that of $\cdot$. An algebra of monotonic Boolean transformers is *complete* if its underlying bounded distributive lattice is complete and $\sqcap$ distributes over arbitrary suprema and $\sqcup$ distributes over arbitrary infima.

In multirelational models, $\sqcup$ and $\sqcap$ are angelic and demonic choice, $\bot$ and $\top$ are the empty and the universal multirelations, $\cdot$ is multirelational composition, 1 is the set membership multirelation E, and $^{\mathsf{d}}$ is the dual of multirelations. Assertions represent tests.

As a consequence of the above axioms we obtain the $\leq$-least fixpoint $x^{\omega}\cdot y = \mu z.x\cdot z\sqcap y$. Algebras of monotonic Boolean transformers can be extended by the weak iteration operator * of [108] with the axioms

$$x\cdot x^*\sqcap 1 = x^* \qquad z\leq x\cdot z\sqcap y \Rightarrow z\leq x^*\cdot y$$

Then we obtain the $\leq$-greatest fixpoint $x^*\cdot y = \nu z.x\cdot z\sqcap y$. Note that * and $^{\omega}$ denote other fixpoints for monotonic Boolean transformers than for Kleene algebras and omega algebras, which use the function $\lambda z.x\cdot z\sqcup y$ and the reverse order.

Algebras of monotonic Boolean transformers instantiate left Conway semirings as shown in the following result along with numerous other instances that cover computation models M1–M8 and M13 of Chapter 2.

Theorem 50. *Left Conway semirings have the following instances:*

1. *Every left Kleene algebra is a left Conway semiring using $x^\circ = x^*$. In particular, so is every probabilistic Kleene algebra [83].*

2. *Every left omega algebra is a left Conway semiring using $x^\circ = x^*(x^\omega 0+1)$. In particular, every left omega algebra with $\top x = \top$ is a left Conway semiring using $x^\circ = x^*(x^\omega+1)$.*

3. *Every general refinement algebra [108] is a left Conway semiring using $x^\circ = x^\omega$. In particular, so is every demonic refinement algebra.*

4. *Model M4 of Section 2.2.3 forms a left Conway semiring using* $x^\circ = d(x^\omega)\mathsf{L} + x^*$, *where* d *is the domain operation.*

5. *Model M5 of Section 2.2.4 forms a left Conway semiring using* $x^\circ = n(x^\omega)\mathsf{L} + x^*$, *where* $n(x)$ *captures the infinite executions of* x *as a test as described in Section 4.1.1.*

6. *Every itering as defined in Section 3.2.1 is a left Conway semiring.*

7. *Every extended binary itering as defined in Section 3.3.1 is a left Conway semiring using* $x^\circ = x \star 1$.

8. *Every algebra of monotonic Boolean transformers is a left Conway semiring in two ways:*

   (a) *using* $x^\circ = x^\omega$ *and* $x + y = x \sqcap y$ *and* $0 = \top$, *and*

   (b) *using* $x^\circ = x^{\mathsf{d}\omega\mathsf{d}}$ *and* $x + y = x \sqcup y$ *and* $0 = \bot$.

9. *Every algebra of monotonic Boolean transformers extended by the weak iteration operator* * *of [108] is a left Conway semiring in two further ways:*

   (a) *using* $x^\circ = x^*$ *and* $x + y = x \sqcap y$ *and* $0 = \top$, *and*

   (b) *using* $x^\circ = x^{\mathsf{d}*\mathsf{d}}$ *and* $x + y = x \sqcup y$ *and* $0 = \bot$.

Up-closed multirelations form a general refinement algebra and an algebra of monotonic Boolean transformers. Hence they form a left Conway semiring in four different ways as stated in parts 8 and 9 of Theorem 50.

Idempotent left semirings can be extended with domain and antidomain operations as shown in [86, 26]. This generalises to our relative domain and antidomain operations, which can extend each of the above models. Because of their duality, up-closed multirelations actually form relative antidomain semirings in two different ways as shown by the following result.

Theorem 51. *Every algebra of monotonic Boolean transformers is a relative antidomain semiring in two ways:*

1. *using* $a(x) = x^{\mathsf{d}}\top \sqcup 1$ *and* $d(x) = x\bot \sqcup 1$ *and* $x + y = x \sqcap y$ *and* $0 = \mathsf{Z} = \top$, *and*

2. *using* $a(x) = x^{\mathsf{d}}\bot \sqcap 1$ *and* $d(x) = x\top \sqcap 1$ *and* $x + y = x \sqcup y$ *and* $0 = \mathsf{Z} = \bot$.

In the second instance, tests are assertions, while they are assumptions in the first instance. Given a relative antidomain semiring, we can choose either diamond or box to represent preconditions by Theorem 47.

Altogether we obtain eight different instances in which up-closed multirelations form a modal Conway semiring. We describe these instances in terms of monotonic Boolean transformers. The options are summarised as follows:

- choose operations or their duals,
- choose $\leq$-least or $\leq$-greatest fixpoints for loops,
- choose diamond or box for preconditions.

All eight instances have the same operations for sequential composition $\cdot$ and 1. Depending on the choice of operations or their duals we obtain the instances

$+$	0	$\leq$	Z	$d(x)$	$\lvert x\rangle y$	$a(x)$	$\lvert x]y$	$x \lhd p \rhd y$
$\sqcup$	$\bot$	$\leq$	$\bot$	$x\top \sqcap 1$	$xy\top \sqcap 1$	$\neg x$	$\neg(x\neg y)$	$px \sqcup \neg py$
$\sqcap$	$\top$	$\geq$	$\top$	$x\bot \sqcup 1$	$xy\bot \sqcup 1$	$\sim x$	$\sim(x\sim y)$	$px \sqcap \sim py$

where $\neg x = x^{\mathsf{d}} \bot \sqcap 1$ complements assertions and $\sim x = x^{\mathsf{d}} \top \sqcup 1$ complements assumptions. In each of these two cases, we can choose the $\leq$-least or the $\leq$-greatest fixpoint for loops, which yields

$+$	x°	$p \star x$
$\sqcup$	$x^{\mathsf{d}*\mathsf{d}} = \mu y.xy \sqcup 1$	$(px)^{\mathsf{d}*\mathsf{d}}\neg p = \mu y.pxy \sqcup \neg p$
$\sqcup$	$x^{\mathsf{d}\omega\mathsf{d}} = \nu y.xy \sqcup 1$	$(px)^{\mathsf{d}\omega\mathsf{d}}\neg p = \nu y.pxy \sqcup \neg p$
$\sqcap$	$x^\omega = \mu y.xy \sqcap 1$	$(px)^\omega \sim p = \mu y.pxy \sqcap \sim p$
$\sqcap$	$x^* = \nu y.xy \sqcap 1$	$(px)^* \sim p = \nu y.pxy \sqcap \sim p$

Finally, in each of these four cases, we can choose the diamond or the box operator for preconditions:

$+$	$x \lhd p \rhd y$	$p \star x$	$x \ll q$	ℓ
$\sqcup$	$px \sqcup \neg py$	$(px)^{\mathsf{d}*\mathsf{d}}\neg p$	$xq\top \sqcap 1$	$\bot$
$\sqcup$	$px \sqcup \neg py$	$(px)^{\mathsf{d}*\mathsf{d}}\neg p$	$x^{\mathsf{d}}q\top \sqcap 1$	1
$\sqcup$	$px \sqcup \neg py$	$(px)^{\mathsf{d}\omega\mathsf{d}}\neg p$	$xq\top \sqcap 1$	1
$\sqcup$	$px \sqcup \neg py$	$(px)^{\mathsf{d}\omega\mathsf{d}}\neg p$	$x^{\mathsf{d}}q\top \sqcap 1$	$\bot$

$+$	$x \lhd p \rhd y$	$p \star x$	$x \ll q$	ℓ
$\sqcap$	$px \sqcap \sim py$	$(px)^\omega \sim p$	$xq\bot \sqcup 1$	1
$\sqcap$	$px \sqcap \sim py$	$(px)^\omega \sim p$	$x^{\mathsf{d}}q\bot \sqcup 1$	$\top$
$\sqcap$	$px \sqcap \sim py$	$(px)^* \sim p$	$xq\bot \sqcup 1$	$\top$
$\sqcap$	$px \sqcap \sim py$	$(px)^* \sim p$	$x^{\mathsf{d}}q\bot \sqcup 1$	1

The instances are collected in the following result, which shows that we obtain a sound and complete correctness calculus for four of them. We obtain at least soundness for the remaining four, with completeness being an open problem.

Theorem 52. *Every complete algebra of monotonic Boolean transformers is a modal Conway algebra with a precondition operation in eight ways:*

$+$	x^*	x°	$x \ll q$	*sound*	*complete*
$\sqcup$	$x^{\mathsf{d}*\mathsf{d}}$	$x^{\mathsf{d}*\mathsf{d}}$	$\lvert x\rangle q$	✓	✓
$\sqcup$	$x^{\mathsf{d}*\mathsf{d}}$	$x^{\mathsf{d}*\mathsf{d}}$	$\lvert x] q$	✓	
$\sqcup$	$x^{\mathsf{d}*\mathsf{d}}$	$x^{\mathsf{d}\omega\mathsf{d}}$	$\lvert x\rangle q$	✓	
$\sqcup$	$x^{\mathsf{d}*\mathsf{d}}$	$x^{\mathsf{d}\omega\mathsf{d}}$	$\lvert x] q$	✓	✓

$+$	x^*	x°	$x \ll q$	*sound*	*complete*
$\sqcap$	x^*	x^ω	$\lvert x\rangle q$	✓	
$\sqcap$	x^*	x^ω	$\lvert x] q$	✓	✓
$\sqcap$	x^*	x^*	$\lvert x\rangle q$	✓	✓
$\sqcap$	x^*	x^*	$\lvert x] q$	✓	

Instances marked sound and complete satisfy the conditions of Theorem 49 yielding a sound and complete correctness calculus. Instances marked sound satisfy conditions that yield a sound correctness calculus.

For constructing while-programs, these instances use the set of all assertions/assumptions as atomic tests and the set of all continuous/co-continuous elements as atomic programs. These are described in the following.

Let S be a complete algebra of monotonic Boolean transformers. A subset $T \subseteq S$ is *directed* if it is not empty and every pair of elements in T has an upper bound in T. Dually, lower bounds are required for *co-directed* sets. We call an element $x \in S$ *continuous* if $x \sum_{y \in T} y = \sum_{y \in T} xy$ for every directed set $T \subseteq S$. Dually, $x \in S$ is *co-continuous* if $x \prod_{y \in T} y = \prod_{y \in T} xy$ for every co-directed set $T \subseteq S$. Continuity and co-continuity are preserved as the following result shows.

Theorem 53. *Let S be a complete algebra of monotonic Boolean transformers. Let $x, y \in S$ be continuous, let $u, v \in S$ be co-continuous and let p be an assertion or an assumption. Then*

1. *p and 1 and xy and $x \sqcup y$ and $x \sqcap y$ and $x^{\mathsf{d}*\mathsf{d}}$ and x^ω and u^{d} are continuous;*
2. *p and 1 and uv and $u \sqcup v$ and $u \sqcap v$ and $u^{\mathsf{d}\omega\mathsf{d}}$ and u^* and x^{d} are co-continuous.*

5.4 Publications

Axioms for preconditions and while-programs and a sound and relatively complete correctness calculus for partial-correctness statements in model M1 were given in [57]. They were

generalised to total-correctness statements and other kinds of correctness statement that can be expressed in models M1–M5 in [47]. A further generalisation to cover the multirelational computation model M13 was carried out in [50].

Relative (anti)domain semirings and the associated relative modal operators were proposed in [47], which also treats pre-post specifications. The combination of modal semirings with left Conway semirings to cover model M13 was proposed in [50], which also instantiates these structures to the algebras of monotonic Boolean transformers of [97].

Chapter 6

Conclusion

In this work we have investigated several models of state-based non-deterministic sequential computations and introduced a number of algebras that unify various aspects of these models. The models cover strict and non-strict computations, support various combinations of finite, infinite and aborting executions and up to two kinds of non-deterministic choice. The algebras describe operations such as non-deterministic choice, sequential composition, various kinds of iteration, the domain and the infinite executions of a computation, tests, preconditions and pre-post specifications. Program constructs, correctness statements and refinement and approximation orders are defined in terms of these operations.

Program transformations and refinements can be stated as equations or conditional equations in the algebras and inferred from the axioms manually or with various degrees of automation. Because of the unifying approach such results – once established – hold for a variety of computation models with widely different interpretations. They provide additional guarantees about programs.

On a more fundamental level the algebraic approach gives insights into how computation models are related. The difference between two models can sometimes be expressed in a few characteristic axioms that hold in one model but fail in the other. Moreover the axioms provide a structure for organising the hierarchy of algebras that arises when different computation models are covered. We strive to prove results in algebras with weaker axioms so as to cover more models.

Our work has profited in both quality and quantity from the use of tools, in particular, Isabelle's interactive and automated theorem proving technology. The system provides a level of assurance which goes beyond that provided by careful manual calculations. The effort needed to create a proof varies. Some results can be automatically proved or refuted easily before a human would even get an intuition for whether they might hold or not hold; this greatly encourages exploring, for example, variants of axioms. On the other hand, some proofs need to be broken down tediously into small steps below a level at which a human prover operates who intuitively applies properties such as associativity, commutativity, idempotence and transitivity in lattices. This in turn is offset by the structuring mechanisms without which theory development does not scale well.

The algebras in this work are based on state-based non-deterministic sequential computations. We did not consider many aspects found in other computation models, for example, timing, probabilities, concurrency and communication. We also did not consider many other computation models, for example, those underlying functional programming, logic programming, hybrid systems and quantum computing. This is a deliberate restriction due to the author's opinion that we do not know enough even about fundamental models of sequential computations. It does not mean that algebraic techniques are limited to these; for example, see [10, 70, 83, 72, 71]. Algebras will continue to be applied in understanding, structuring and relating computation models.

References

[1] C. J. Aarts, R. C. Backhouse, E. A. Boiten, H. Doornbos, N. van Gasteren, R. van Geldrop, P. F. Hoogendijk, E. Voermans, and J. van der Woude. Fixed-point calculus. *Information Processing Letters*, 53(3):131–136, 1995.

[2] K. R. Apt, F. S. de Boer, and E.-R. Olderog. *Verification of Sequential and Concurrent Programs*. Springer, third edition, 2009.

[3] G. Aumann. Kontakt-Relationen. *Sitzungsberichte der Bayerischen Akademie der Wissenschaften, Mathematisch-Naturwissenschaftliche Klasse*, pages 67–77, 1970.

[4] R.-J. Back and J. von Wright. *Refinement Calculus*. Springer, New York, 1998.

[5] R. J. R. Back and J. von Wright. Games and winning strategies. *Information Processing Letters*, 53(3):165–172, 1995.

[6] R. J. R. Back and J. von Wright. Reasoning algebraically about loops. *Acta Informatica*, 36(4):295–334, 1999.

[7] 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.

[8] R. Berghammer, H. Ehler, and B. Möller. On the refinement of non-deterministic recursive routines by transformations. In M. Broy and C. B. Jones, editors, *Programming Concepts and Methods*, pages 53–71. North-Holland Publishing Company, 1990.

[9] R. Berghammer and H. Zierer. Relational algebraic semantics of deterministic and nondeterministic programs. *Theoretical Computer Science*, 43:123–147, 1986.

[10] R. Bird and O. de Moor. *Algebra of Programming*. Prentice Hall, 1997.

[11] G. Birkhoff. *Lattice Theory*, volume XXV of *Colloquium Publications*. American Mathematical Society, third edition, 1967.

[12] 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.

[13] 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.

[14] S. L. Bloom and Z. Ésik. *Iteration Theories: The Equational Logic of Iterative Processes*. Springer, 1993.

[15] S. L. Bloom and Z. Ésik. Matrix and matricial iteration theories, part I. *Journal of Computer and System Sciences*, 46(3):381–408, 1993.

[16] M. Broy. A theory for nondeterminism, parallelism, communication, and concurrency. *Theoretical Computer Science*, 45:1–61, 1986.

[17] M. Broy, R. Gnatz, and M. Wirsing. Semantics of nondeterministic and noncontinuous constructs. In F. L. Bauer and M. Broy, editors, *Program Construction*, volume 69 of *Lecture Notes in Computer Science*, pages 553–592. Springer, 1979.

[18] R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, 1977.

[19] 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.

[20] A. Cavalcanti, J. Woodcock, and S. Dunne. Angelic nondeterminism in the unifying theories of programming. *Formal Aspects of Computing*, 18(3):288–307, 2006.

[21] 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.

[22] J. H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.

[23] J. Desharnais, B. Möller, and G. Struth. Kleene algebra with domain. *ACM Transactions on Computational Logic*, 7(4):798–833, 2006.

[24] J. Desharnais, B. Möller, and G. Struth. Algebraic notions of termination. *Logical Methods in Computer Science*, 7(1:1):1–29, 2011.

[25] J. Desharnais, B. Möller, and F. Tchier. Kleene under a modal demonic star. *Journal of Logic and Algebraic Programming*, 66(2):127–160, 2006.

[26] J. Desharnais and G. Struth. Domain axioms for a family of near-semirings. In J. Meseguer and G. Roşu, editors, *Algebraic Methodology and Software Technology*, volume 5140 of *Lecture Notes in Computer Science*, pages 330–345. Springer, 2008.

[27] J. Desharnais and G. Struth. Internal axioms for domain semirings. *Science of Computer Programming*, 76(3):181–203, 2011.

[28] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[29] 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.

[30] S. Dunne. Conscriptions: A new relational model for sequential computations. In B. Wolff, M.-C. Gaudel, and A. Feliachi, editors, *Unifying Theories of Programming, Fourth International Symposium, UTP 2012*, volume 7681 of *Lecture Notes in Computer Science*, pages 144–163. Springer, 2013.

[31] S. E. Dunne, I. J. Hayes, and A. J. Galloway. Reasoning about loops in total and general correctness. In A. Butterfield, editor, *Unifying Theories of Programming, Second International Symposium, UTP 2008*, volume 5713 of *Lecture Notes in Computer Science*, pages 62–81. Springer, 2010.

[32] H. Egli. A mathematical model for non-deterministic computations. Technical report, Forschungsinstitut für Mathematik ETH Zürich, 1975.

[33] T. F. Gritzner and R. Berghammer. A relation algebraic model of robust correctness. *Theoretical Computer Science*, 159(2):245–270, 1996.

[34] 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.

[35] W. Guttmann. *Algebraic Foundations of the Unifying Theories of Programming*. Dissertation, Universität Ulm, 2007.

[36] W. Guttmann. Algebraic foundations of the Unifying Theories of Programming. In D. Wagner et al., editors, *Ausgezeichnete Informatikdissertationen 2007*, volume D-8 of *Lecture Notes in Informatics*, pages 141–150. Gesellschaft für Informatik, 2008.

[37] W. Guttmann. Lazy relations. In R. Berghammer, B. Möller, and G. Struth, editors, *Relations and Kleene Algebra in Computer Science*, volume 4988 of *Lecture Notes in Computer Science*, pages 138–154. Springer, 2008.

[38] 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.

[39] W. Guttmann. Imperative abstractions for functional actions. *Journal of Logic and Algebraic Programming*, 79(8):768–793, 2010.

[40] W. Guttmann. Lazy UTP. In A. Butterfield, editor, *Unifying Theories of Programming, Second International Symposium, UTP 2008*, volume 5713 of *Lecture Notes in Computer Science*, pages 82–101. Springer, 2010.

[41] 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.

[42] 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.

[43] W. Guttmann. Fixpoints for general correctness. *Journal of Logic and Algebraic Programming*, 80(6):248–265, 2011.

[44] 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.

[45] W. Guttmann. Algebras for iteration and infinite computations. *Acta Informatica*, 49(5):343–359, 2012.

[46] W. Guttmann. Typing theorems of omega algebra. *Journal of Logic and Algebraic Programming*, 81(6):643–659, 2012.

[47] W. Guttmann. Unifying correctness statements. In J. Gibbons and P. Nogueira, editors, *Mathematics of Program Construction*, volume 7342 of *Lecture Notes in Computer Science*, pages 198–219. Springer, 2012.

[48] 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.

[49] W. Guttmann. Extended designs algebraically. *Science of Computer Programming*, 78(11):2064–2085, 2013.

[50] W. Guttmann. Algebras for correctness of sequential computations. *Science of Computer Programming*, 85(Part B):224–240, 2014.

[51] W. Guttmann. Extended conscriptions algebraically. In P. Höfner, P. Jipsen, W. Kahl, and M. E. Müller, editors, *Relational and Algebraic Methods in Computer Science*, volume 8428 of *Lecture Notes in Computer Science*, pages 139–156. Springer, 2014.

[52] W. Guttmann. Multirelations with infinite computations. *Journal of Logical and Algebraic Methods in Programming*, 83(2):194–211, 2014.

[53] W. Guttmann. Infinite executions of lazy and strict computations. *Journal of Logical and Algebraic Methods in Programming*, 84(3):326–340, 2015.

[54] W. Guttmann. Isabelle/HOL theories of algebras for iteration, infinite executions and correctness of sequential computations. Technical Report TR-COSC 02/15, Department of Computer Science and Software Engineering, University of Canterbury, 2015. The document is available at http://www.cosc.canterbury.ac.nz/research/reports/TechReps/abstracts/1502_abs.html.

[55] W. Guttmann and B. Möller. Modal design algebra. In S. Dunne and W. Stoddart, editors, *Unifying Theories of Programming*, volume 4010 of *Lecture Notes in Computer Science*, pages 236–256. Springer, 2006.

[56] W. Guttmann and B. Möller. Normal design algebra. *Journal of Logic and Algebraic Programming*, 79(2):144–173, 2010.

[57] W. Guttmann, G. Struth, and T. Weber. Automating algebraic methods in Isabelle. In S. Qin and Z. Qiu, editors, *Formal Methods and Software Engineering*, volume 6991 of *Lecture Notes in Computer Science*, pages 617–632. Springer, 2011.

[58] W. Guttmann, G. Struth, and T. Weber. A repository for Tarski-Kleene algebras. In P. Höfner, A. McIver, and G. Struth, editors, *Automated Theory Engineering*, volume 760 of *CEUR Workshop Proceedings*, pages 30–39, 2011.

[59] D. Harel. *First-Order Dynamic Logic*, volume 68 of *Lecture Notes in Computer Science*. Springer, 1979.

[60] D. Harel. On the total correctness of nondeterministic programs. *Theoretical Computer Science*, 13(2):175–192, 1981.

[61] 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.

[62] I. J. Hayes, S. E. Dunne, and L. A. Meinicke. Linking Unifying Theories of Program refinement. *Science of Computer Programming*, 78(11):2086–2107, 2013.

[63] U. Hebisch and H. J. Weinert. *Halbringe*. Teubner, 1993.

[64] E. C. R. Hehner. Termination is timing. In J. L. A. van de Snepscheut, editor, *Mathematics of Program Construction*, volume 375 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 1989.

[65] W. H. Hesselink. Multirelations are predicate transformers. The document is available at http://www.cs.rug.nl/~wim/pub/whh318.pdf, 2004.

[66] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580/583, 1969.

[67] C. A. R. Hoare. A couple of novelties in the propositional calculus. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 31(9–12):173–178, 1985.

[68] C. A. R. Hoare. Theories of programming: Top-down and bottom-up and meeting in the middle. In J. M. Wing, J. Woodcock, and J. Davies, editors, *FM'99: Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 1999.

[69] C. A. R. Hoare, I. J. Hayes, J. He, C. C. Morgan, A. W. Roscoe, J. W. Sanders, I. H. Sorensen, J. M. Spivey, and B. A. Sufrin. Laws of programming. *Communications of the ACM*, 30(8):672–686, 1987.

[70] C. A. R. Hoare and J. He. *Unifying theories of programming.* Prentice Hall Europe, 1998.

[71] C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene Algebra and its foundations. *Journal of Logic and Algebraic Programming*, 80(6):266–296, 2011.

[72] P. Höfner and B. Möller. An algebra of hybrid systems. *Journal of Logic and Algebraic Programming*, 78(2):74–97, 2009.

[73] E. V. Huntington. Boolean algebra. A correction. *Transactions of the American Mathematical Society*, 35(2):557–558, 1933.

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

[75] D. Jacobs and D. Gries. General correctness: A unification of partial and total correctness. *Acta Informatica*, 22(1):67–83, 1985.

[76] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.

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

[78] D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Transactions on Computational Logic*, 1(1):60–76, 2000.

[79] R. D. Maddux. Relation-algebraic semantics. *Theoretical Computer Science*, 160(1–2):1–85, 1996.

[80] C. E. Martin, S. A. Curtis, and I. Rewitzky. Modelling angelic and demonic nondeterminism with multirelations. *Science of Computer Programming*, 65(2):140–158, 2007.

[81] 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/.

[82] A. McIver, T. M. Rabehaja, and G. Struth. On probabilistic Kleene algebras, automata and simulations. In H. de Swart, editor, *Relational and Algebraic Methods in Computer Science*, volume 6663 of *Lecture Notes in Computer Science*, pages 264–279. Springer, 2011.

[83] A. K. McIver and T. Weber. Towards automated proof support for probabilistic distributed systems. In G. Sutcliffe and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3835 of *Lecture Notes in Computer Science*, pages 534–548. Springer, 2005.

[84] L. Meertens. Abstracto 84: The next generation. In A. L. Martin and J. L. Elshoff, editors, *ACM '79: Proceedings of the 1979 annual conference*, pages 33–39. ACM Press, 1979.

[85] 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.

[86] B. Möller. Kleene getting lazy. *Science of Computer Programming*, 65(2):195–214, 2007.

[87] B. Möller and G. Struth. Algebras of modal operators and partial correctness. *Theoretical Computer Science*, 351(2):221–239, 2006.

[88] 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.

[89] C. Morgan. The specification statement. *ACM Transactions on Programming Languages and Systems*, 10(3):403–419, 1988.

[90] J. M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming*, 9(3):287–306, 1987.

[91] J. M. Morris. Varieties of weakest liberal preconditions. *Information Processing Letters*, 25(3):207–210, 1987.

[92] G. Nelson. A generalization of Dijkstra's calculus. *ACM Transactions on Programming Languages and Systems*, 11(4):517–561, 1989.

[93] 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.

[94] K. Nishizawa, N. Tsumagari, and H. Furusawa. The cube of Kleene algebras and the triangular prism of multirelations. 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 276–290. Springer, 2009.

[95] R. Parikh. Propositional logics of programs: new directions. In M. Karpinski, editor, *Foundations of Computation Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 1983.

[96] 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.

[97] V. Preoteasa. Algebra of monotonic Boolean transformers. In A. Simao and C. Morgan, editors, *Formal Methods: Foundations and Applications*, volume 7021 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 2011.

[98] A. N. Prior. *Past, Present and Future*. Clarendon Press, 1967.

[99] T. M. Rabehaja and J. W. Sanders. Refinement algebra with explicit probabilism. In W.-N. Chin and S. Qin, editors, *Third IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 63–70. IEEE Computer Society, 2009.

[100] I. Rewitzky. Binary multirelations. In H. de Swart, E. Orłowska, G. Schmidt, and M. Roubens, editors, *Theory and Applications of Relational Structures as Knowledge Instruments*, volume 2929 of *Lecture Notes in Computer Science*, pages 256–271. Springer, 2003.

[101] I. Rewitzky and C. Brink. Monotone predicate transformers as up-closed multirelations. In R. Schmidt, editor, *Relations and Kleene Algebra in Computer Science*, volume 4136 of *Lecture Notes in Computer Science*, pages 311–327. Springer, 2006.

[102] G. Schmidt. *Relational Mathematics*. Cambridge University Press, 2011.

[103] G. Schmidt and T. Ströhlein. *Relationen und Graphen*. Springer, 1989.

[104] K. Segerberg. A completeness theorem in the modal logic of programs. In T. Traczyk, editor, *Universal Algebra and Applications*, volume 9 of *Banach Center Publications*, pages 31–46. Institute of Mathematics, Polish Academy of Sciences, 1982.

[105] K. Solin. A while program normal form theorem in total correctness. 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 322–336. Springer, 2009.

[106] H. Søndergaard and P. Sestoft. Non-determinism in functional languages. *The Computer Journal*, 35(5):514–523, 1992.

[107] A. Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73–89, 1941.

[108] J. von Wright. Towards a refinement algebra. *Science of Computer Programming*, 51(1–2):23–45, 2004.