Unifying Correctness Statements

Walter Guttmann

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

Abstract. Partial, total and general correctness and further models of sequential computations differ in their treatment of finite, infinite and aborting executions. Algebras structure this diversity of models to avoid the repeated development of similar theories and to clarify their range of application. We introduce algebras that uniformly describe correctness statements, correctness calculi, pre-post specifications and loop refinement rules in five kinds of computation models. This extends previous work that unifies iteration, recursion and program transformations for some of these models. Our new description includes a relativised domain operation, which ignores parts of a computation, and represents bound functions for claims of termination by sequences of tests. We verify all results in Isabelle heavily using its automated theorem provers.

1 Introduction

Sequential computations have many different models with varying degrees of precision as regards their ability to distinguish finite, infinite and aborting executions (which terminate due to an error; 'finite' means 'normally terminating'). Partial correctness models [23, 10, 29, 33] ignore infinite and aborting executions, general correctness models [2, 4, 27, 3, 38, 11, 34, 32, 13] represent but do not distinguish infinite and aborting executions, and total correctness models [10, 25, 39, 7, 31, 18] ignore finite executions in the presence of infinite and aborting ones. Yet other models ignore finite and infinite executions when aborting ones are present [22, 17] or represent finite, infinite and aborting executions independently [16].

Having a variety of models is useful: better precision is not always desired as it entails more details (which might be unnecessary or distracting for some applications) and typically a more complex theory (which might hinder comprehension or automation). However, this diversity of models should be structured to avoid the repeated development of similar theories for similar models, to improve our understanding of their connections and characterising properties and to encourage a systematic exploration with an eye to discovering new models.

Algebra provides the required structure. Key aspects of the models, such as the semantics of iteration or the infinite executions of a computation, are described by operations and axioms which are general enough to capture various computation models, yet powerful enough for the derivation of results known from particular models. These results are then recognised to hold in all models satisfying the common axioms. Examples include complex program transformations, separation rules and refinement laws, for which one common proof

establishes validity across several models [14, 17, 16]. Individual models are characterised by adding specific axioms to the common ones. Moreover, the axioms are suited to support by automated theorem provers and SMT solvers [19, 16].

The present paper extends this unifying approach to correctness statements and their calculi. Correctness statements similar to Hoare triples claim that a computation has only restricted kinds of executions, 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 executions involved. But in general there are several computation models capable of expressing a particular statement. At the level of concrete models, we therefore distinguish between the computation models and the kinds of correctness statements each supports. At the algebraic level, our approach is unifying in both dimensions: one statement applies in various models to various correctness claims.

We give a propositional correctness calculus for the unified correctness statements and show its soundness and completeness. The latter presumes boundedly non-deterministic programs. The calculus, too, unifies different computation models and correctness claims. We furthermore extend our unifying approach to pre-post specifications and loop refinement rules useful for program construction. Innovations which facilitate our development are a relativised domain operation and the representation of bound functions by sequences of tests.

Thus the contributions of the present paper are as follows:

- A generalisation of domain semirings [8, 9]: the new operation uniformly describes the domain of a part of a computation, such as its aborting, infinite or finite executions or combinations thereof. This is a form of relativisation: the operation gives the domain up to certain executions which are ignored. Technically, an element Z is singled out and the domain axioms are relaxed so as to ignore parts of elements contained in Z. Most of the theory of domain and antidomain semirings is relativised this way, including modal semirings with their diamond and box operators.
- An algebraic description of termination arguments known, for example, from the total correctness while-loop rule of the Hoare calculus. This is done by capturing the loop variant or bound function by a sequence of tests.
- An extension of algebraic accounts of correctness statements and their calculi [29, 33, 34, 13, 14, 19], which unifies existing propositional Hoare calculi in two ways. First, it applies to several computation models, which vary in their ability to describe aborting and infinite executions. Second, for each model, it applies to several kinds of correctness statements, which vary in their claims about aborting, infinite and finite executions or combinations thereof, up to the precision allowed by the model. In particular, this uniformly describes claims of partial, total and general correctness.
- An algebraic description of pre-post specifications and loop refinement rules based on the above correctness statements. This generalises previous works [39, 12–14] again by uniformly applying to several computation models and several kinds of correctness claims.

Together they achieve the main contribution: a framework that unifies correctness reasoning for various computation models and correctness claims.

All results are verified in Isabelle making heavy use of its integrated automated theorem provers. The proofs can be found in the theory files, which are available at http://www.uni-ulm.de/en/in/pm/staff/guttmann/algebra/.

We deal with computation models at various levels of abstraction. Concrete models appear in the literature, for example, in terms of relations over extended state spaces, predicates or predicate transformers. In Section 2 we describe a number of these models in a uniform setting, namely matrices of relations, and discuss the different kinds of correctness statements they feature. Abstracting from the matrix representation, in Section 3 we introduce relative domain semirings, which allow us to express the correctness statements in a uniform way. At the same level we express iteration in Section 4. Finally, in Section 5 we axiomatise properties of preconditions and while-programs, based on which we introduce the unified correctness calculus, pre-post specifications and loop refinement laws.

2 Models

This section gives an overview of the models and various kinds of correctness statements which will be algebraically captured in the remainder of this paper. We distinguish between the computation models and the correctness statements applicable for each model. The models vary according to precision as regards their ability to describe infinite and aborting executions in addition to finite ones. Each model may support different kinds of correctness statements about the executions of a computation limited by its precision.

All of the following models describe sequential, non-deterministic computations and are based on relations over a state space given by the possible values of program variables. The program

$$R = (\mathsf{while}\ x \leq 1\ \mathsf{do}\ x := x/x)$$

is our running example. Its variable x has values in $\mathbb{N}$ and x/x is integer division. Execution of R leaves x unchanged unless $x = 1$, in which case the execution does not terminate, or $x = 0$, in which case it aborts due to division by zero. However, not all computation models can represent aborting or infinite executions.

2.1 Partial Correctness

In the first model, the program R is a binary relation over the state space $\mathbb{N}$. A pair (x, x') in R specifies that there is an execution of R which starts in state x and terminates in x'. Hence

$$R = \{(x, x) \mid x \geq 2\}$$

comprises the finite executions of the program. Because it is deterministic, R is a partial function; in general there may be more than one final state x' related to a

single initial state x. This simple model has no provision for representing aborting or infinite executions. The partiality of R indicates 'missing' finite executions, but is otherwise unrelated to the presence of aborting or infinite executions; see also [38].

Therefore the typical correctness claim in this model is about partial correctness. For conditions or 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 normally, does so in a state in q. This triple claims nothing about infinite or aborting executions.

The Hoare triple $p\{R\}q$ is algebraically formalised by $p \cdot R \cdot q' \leq 0$ using tests p and q [29]. Tests correspond to subsets of the identity relation and act as filters in a sequential composition. The test q' is the complement of q, the operation $\cdot$ is relational composition, $\leq$ is subset and 0 is the empty relation. According to the inequality there is no execution in R which starts in p and ends in a state not in q.

2.2 Total Correctness

The second model augments the above relational model to represent executions that do not terminate normally [31, 18]. It is an abstraction of the 'designs' of the Unifying Theories of Programming [25]. The program R is a 2×2 matrix whose entries are relations over $\mathbb{N}$, namely

$$R = \begin{pmatrix} \top & \top \\ W & X \end{pmatrix}, \qquad \begin{array}{l} W = \{(x, x') \mid x \leq 1\} \\ X = \{(x, x) \mid x \geq 2\} \cup W \end{array}$$

For every program in this model, both entries in the top row are the universal relation $\top = \mathbb{N} \times \mathbb{N}$. They are chosen so as to appropriately propagate the information captured in W through a sequential composition. Subsequent models feature matrices with other combinations of 0 and $\top$ entries providing a characteristic, constant structure for each model.

The entry W represents the states from which executions exist that do not terminate normally. It is a vector, that is, a relation in which every state is related either to all states or to none, and therefore corresponds to a set of states. No distinction is made between infinite and aborting executions: $(x, x') \in W$ means that there is an execution starting in x which does not terminate normally.

The entry X represents the finite executions of the program with the proviso $W \subseteq X$. This requirement leads to a demonic non-deterministic choice: for example, the endless loop is the matrix with four $\top$ entries, which is an annihilator of the non-deterministic choice given by componentwise union. Because of this, computations with both finite and infinite or aborting executions starting in the same state cannot be represented properly. For example, consider $R + \mathsf{skip}$ using the non-deterministic choice $+$ and the program skip that does not change the state. The matrix representation of skip has the empty relation as W and the identity relation as X. In the initial state $x = 1$ there is both an infinite and a finite execution in the computation $R + \mathsf{skip}$, but in the current model $R = R + \mathsf{skip}$. Thus the finite execution of skip is ignored in the presence of R.

This model supports total correctness claims. The Hoare triple $p\{R\}q$ now expresses that every execution of R which starts in p terminates normally in q. Again this is formalised by $p \cdot R \cdot q' \leq 0$ [39]. The order $\leq$ is the subset relation lifted componentwise to matrices. The test p is represented just as a program by a 2×2 matrix of the above form, using $W = 0$ and a subset U of the identity relation as X; a similar representation is used for the test q'. The computation 0 with no executions is represented by a matrix with $W = X = 0$. The relational operations are lifted to matrices in the standard way, so that $p \cdot R \cdot q' \leq 0$ elaborates as

$$\begin{pmatrix} \top & \top \\ 0 & U \end{pmatrix} \cdot \begin{pmatrix} \top & \top \\ W & X \end{pmatrix} \cdot \begin{pmatrix} \top & \top \\ 0 & Y' \end{pmatrix} = \begin{pmatrix} \top & \top \\ UW & UW + UXY' \end{pmatrix} \leq \begin{pmatrix} \top & \top \\ 0 & 0 \end{pmatrix}$$

which is equivalent to $UW \subseteq 0 \wedge UXY' \subseteq 0$. As here, we frequently omit the operator $\cdot$ for relational composition, and we contrast $\subseteq$ on the components with its lifted counterpart $\leq$. The first term $UW \subseteq 0$ expresses that all executions starting in p terminate normally. The second term $UXY' \subseteq 0$ claims partial correctness: no execution starting in p terminates in q'. Their conjunction holds, for example, using $U = \{(x, x) \mid 2 \leq x \leq 5\}$ and $Y = \{(x, x) \mid x \leq 5\}$.

However, partial correctness cannot be claimed alone. In particular, the 'weak correctness' claim $p \cdot R = p \cdot R \cdot q$ of [39] reduces to $UXY' \subseteq UW$. This expresses that no execution starting in a state in p terminates in q', provided all executions starting in the same state terminate normally (whence $UW = 0$).

2.3 General Correctness

The third model removes the restriction imposed by the previous one, so that finite executions can be represented independently of executions which do not terminate normally [31, 13]. It is an abstraction of the 'prescriptions' of the Unifying Theories of Programming [11]. The independence is achieved by modifying the structure of the 2×2 matrices. The program R becomes

$$R = \begin{pmatrix} \top & 0 \\ W & X \end{pmatrix}, \qquad \begin{matrix} W = \{(x, x') \mid x \leq 1\} \\ X = \{(x, x) \mid x \geq 2\} \end{matrix}$$

Now the top-right entry is the empty relation 0 instead of $\top$ and the restriction $W \subseteq X$ is abandoned. Otherwise the interpretations of W and X remain as in the total correctness model. In the current model, $R \neq R + \mathsf{skip}$ and $R + \mathsf{skip}$ has precisely the two expected executions starting in the state $x = 1$. Still there is no distinction between infinite and aborting executions.

This model supports both partial and total correctness claims, and is typically called 'general correctness' [27]. First, observe that

$$\begin{pmatrix} \top & 0 \\ 0 & U \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ W & X \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & Y' \end{pmatrix} = \begin{pmatrix} \top & 0 \\ UW & UXY' \end{pmatrix} \leq \begin{pmatrix} \top & 0 \\ \top & 0 \end{pmatrix}$$

is equivalent to the partial correctness claim $UXY' \subseteq 0$. The matrix on the right-hand side of the inequality represents the program loop which has only

infinite executions. On the level of programs the partial correctness claim is thus formalised by $p \cdot R \cdot q' \leq \mathsf{loop}$ [13]. This is equivalent to $p \cdot R = p \cdot R \cdot q$ in the current model. Second, set $Y' = 0$ and observe that

$$\begin{pmatrix} \top & 0 \\ 0 & U \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ W & X \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} \top & 0 \\ UW & 0 \end{pmatrix} \leq \begin{pmatrix} \top & 0 \\ 0 & 0 \end{pmatrix}$$

is equivalent to $UW \subseteq 0$, 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 claim, but it is not required to use the same precondition p in both inequalities. This makes it possible to state claims about termination independently from claims about finite executions. For example, for the computation $R + \mathsf{skip}$ the partial correctness claim holds using $U = Y = \{(x, x) \mid x \leq 5\}$ and the claim about normal termination holds using $U = \{(x, x) \mid x \geq 2\}$.

2.4 Extended Designs

The fourth model called 'extended designs' distinguishes aborting and infinite executions [22, 17]. In this model the program R is the 3×3 matrix

$$R = \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ V & W & X \end{pmatrix}, \qquad \begin{aligned} V &= \{(x, x') \mid x = 0\} \\ W &= \{(x, x') \mid x = 1\} \cup V \\ X &= \{(x, x) \mid x \geq 2\} \cup V \end{aligned}$$

The extra row/column stores the main extension with respect to the previously discussed models, namely the entry V which is a vector similar to W. The vector V represents the states from which aborting executions exist, while W analogously represents the infinite executions. Like designs, extended designs make the restrictions $V \subseteq W$ and $V \subseteq X$. Because of them, in the presence of an aborting execution there is no way to distinguish finite or infinite executions. For example, consider the program $S = (\mathsf{if}\ x = 1\ \mathsf{then}\ \mathsf{loop}\ \mathsf{else}\ \mathsf{skip})$. In the current model $R = R + S$ and the finite execution of S in the state $x = 0$ is ignored in the presence of R.

Several kinds of correctness claims are possible in this model. For the first, observe that $V \subseteq W$ implies $UV \subseteq UW$ and therefore

$$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & 0 & U \end{pmatrix} \cdot \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ V & W & X \end{pmatrix} \cdot \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & 0 & Y' \end{pmatrix} = \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ UV & UW & UV{+}UXY' \end{pmatrix} \leq \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

is equivalent to $UW \subseteq 0 \wedge UXY' \subseteq 0$. Hence $p \cdot R \cdot q' \leq 0$ formalises a total correctness claim, 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 claim is obtained by

$$\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & 0 & U \end{pmatrix} \cdot \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ V & W & X \end{pmatrix} \cdot \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & 0 & Y' \end{pmatrix} = \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ UV & UW & UV{+}UXY' \end{pmatrix} \leq \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ 0 & \top & 0 \end{pmatrix}$$

which is equivalent to $UV \subseteq 0 \wedge UXY' \subseteq 0$. The matrix on the right-hand side of the inequality represents loop in this model. Hence the claim is formalised by $p \cdot R \cdot q' \leq \mathsf{loop}$ 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.

Two further claims are obtained by setting $Y' = 0$ again. First, $p \cdot R \cdot 0 \leq 0$ expresses that no infinite and therefore no aborting executions start in p. Second, $p \cdot R \cdot 0 \leq \mathsf{loop}$ expresses 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 $UXY' \subseteq UV$, which expresses that no execution starting in a state in p terminates in q', provided no execution starting in the same state aborts.

2.5 Finite, Infinite and Aborting Executions

The fifth model treats finite, infinite and aborting executions independently [16]. The independence is achieved by modifying the structure of the 3×3 matrices of extended designs. The program R becomes

$$R = \begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ V & W & X \end{pmatrix}, \qquad \begin{aligned} V &= \{(x, x') \mid x = 0\} \\ W &= \{(x, x') \mid x = 1\} \\ X &= \{(x, x) \mid x \geq 2\} \end{aligned}$$

Now the second and third entries in the top row are the empty relation 0 instead of $\top$ and the restrictions $V \subseteq W$ and $V \subseteq X$ are abandoned. Otherwise the interpretations of V, W and X remain as for extended designs. In the current model, $R \neq R + S$ using $S = (\mathsf{if}\ x = 1\ \mathsf{then}\ \mathsf{loop}\ \mathsf{else}\ \mathsf{skip})$ again. Moreover, the computation $R + S$ has an aborting and a finite execution starting in the state $x = 0$, and the computation $R + S + \mathsf{loop}$ additionally has an infinite one. Thus finite, infinite and aborting executions may occur independently.

Claims about finite, infinite and aborting executions can be stated independently, too (see [20] for a derivation from different execution methods based on computation trees). Observe that

$$\begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & 0 & U \end{pmatrix} \cdot \begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ V & W & X \end{pmatrix} \cdot \begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & 0 & Y' \end{pmatrix} = \begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ UV & UW & UXY' \end{pmatrix}.$$

An inequality may be formed with either 0, loop, abort or $\mathsf{loop} + \mathsf{abort}$ on the right-hand side, where the computations 0, loop and abort are represented by the matrices

$$0 = \begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad \mathsf{loop} = \begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ 0 & \top & 0 \end{pmatrix}, \qquad \mathsf{abort} = \begin{pmatrix} \top & 0 & 0 \\ 0 & \top & 0 \\ \top & 0 & 0 \end{pmatrix}.$$

Using $p \cdot R \cdot q'$ or $p \cdot R \cdot 0$ on the left-hand side, we obtain the following seven kinds of claims:

$$\begin{array}{lll}
(7) & p \cdot R \cdot q' \leq 0 & \Leftrightarrow UV \subseteq 0 \wedge UW \subseteq 0 \wedge UXY' \subseteq 0 \\
(6) & p \cdot R \cdot q' \leq \mathsf{abort} & \Leftrightarrow \quad UW \subseteq 0 \wedge UXY' \subseteq 0 \\
(5) & p \cdot R \cdot q' \leq \mathsf{loop} & \Leftrightarrow UV \subseteq 0 \wedge \quad UXY' \subseteq 0 \\
(4) & p \cdot R \cdot q' \leq \mathsf{loop} + \mathsf{abort} & \Leftrightarrow \quad UXY' \subseteq 0 \\
(3) & p \cdot R \cdot 0 \leq 0 & \Leftrightarrow UV \subseteq 0 \wedge UW \subseteq 0 \\
(2) & p \cdot R \cdot 0 \leq \mathsf{abort} & \Leftrightarrow \quad UW \subseteq 0 \\
(1) & p \cdot R \cdot 0 \leq \mathsf{loop} & \Leftrightarrow UV \subseteq 0
\end{array}$$

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

2.6 Summary

We have discussed five computation models that vary in the precision with which they can describe finite, infinite and aborting executions. These models support various kinds of correctness claims limited by their precision. Most claims take the form $p \cdot R \cdot q' \leq \mathsf{Z}$ for a constant Z depending on the model. The computation Z is either 0, loop, abort or $\mathsf{loop} + \mathsf{abort}$. 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.

In the following we give a uniform algebraic description of all of these models and all of these correctness statements.

3 Relative Domain Semirings

Correctness statements in our models have the form $p \cdot R \cdot q' \leq \mathsf{Z}$ for a constant Z. In the partial and total correctness models $\mathsf{Z} = 0$ holds, and claims of this special form $p \cdot R \cdot q' \leq 0$ are well known in semirings with tests [29] or with a domain operation [8, 33]. In this section we generalise domain semirings to be able to encode claims of the form $p \cdot R \cdot q' \leq \mathsf{Z}$ for various values of Z.

3.1 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. These operations satisfy the characteristic properties

$$\begin{array}{l}
x \leq d(y) \cdot x \Leftrightarrow d(x) \leq d(y) \\
a(y) \cdot x \leq 0 \Leftrightarrow a(y) \leq a(x)
\end{array}$$

which are at the centre of our generalisation. By Boolean algebra $d(x) \leq d(y)$ holds if and only if $a(y) \leq a(x)$ 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 to has no executions starting in p. We generalise these properties as follows:

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

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.

In the following we axiomatise the domain and antidomain operations relative to an element Z, which captures executions that are to be ignored. First, an idempotent semiring without right zero – simply called *semiring* in the remainder of this paper – is an algebraic structure $(S, +, \cdot, 0, 1)$ satisfying the axioms

$$\begin{array}{lll} x + (y + z) = (x + y) + z & x(y + z) = xy + xz & x(yz) = (xy)z \\ x + y = y + x & (x + y)z = xz + yz & 1x = x \\ x + x = x & 0x = 0 & x1 = x \\ 0 + x = x & & \end{array}$$

where $x \cdot y$ is conventionally abbreviated as xy. In particular, the operation $+$ is idempotent and $x0 = 0$ is not an axiom. The *semilattice order* $x \leq y \Leftrightarrow x+y = y$ has least element 0, least upper bound $+$ and isotone operations $+$ and $\cdot$. A semiring is *bounded* if it has a greatest element $\top$ satisfying $x + \top = \top$.

In computation models, the operation $+$ represents non-deterministic choice, the operation $\cdot$ sequential composition, 0 the computation with no executions, 1 the program which does not change the state, $\top$ the computation with all possible executions, and $\leq$ the refinement relation. All computation models of Section 2 are semirings, and most of them do not satisfy the law $x0 = 0$.

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

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

are satisfied. Counterexamples generated by 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 [9], in which case $d(d(x)y) = d(x)d(y)$ follows from the remaining axioms.

Theorem 1. *Let S be a relative domain semiring and $x, y \in S$. Then*

- *$(d(S), +, \cdot, 0, d(1))$ is a bounded distributive lattice,*

– d *is isotone,*
– $d(0) = 0$,
– $d(d(x)) = d(x)$,
– $d(xy) \leq d(x) \leq d(1)$,
– $\mathsf{Z}x \leq \mathsf{Z}$,
– $x + \mathsf{Z} = d(x)x + \mathsf{Z}$,
– $d(x) = 0 \Leftrightarrow x \leq \mathsf{Z}$,
– $xy \leq \mathsf{Z} \Leftrightarrow xd(y) \leq \mathsf{Z}$,
– $x \leq d(y)x + \mathsf{Z} \Leftrightarrow d(x) \leq d(y)$.

In particular, $1 + \mathsf{Z} = d(1) + \mathsf{Z}$ holds, but $d(1) = 1$ does not hold in general. Each computation model of Section 2 is a relative domain semiring where Z is any one of the values 0, loop, abort or $\mathsf{loop} + \mathsf{abort}$ available in the model. 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. The element Z cannot be chosen arbitrarily; for example, $\mathsf{Z} = 1$ implies $d(x) \leq d(1) = d(\mathsf{Z}) = 0$ and therefore $x \leq d(x)x + \mathsf{Z} = 1$ which does not hold in any model given in Section 2.

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

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

are satisfied. Counterexamples generated by 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 [9].

Theorem 2. *Let S be a relative antidomain semiring and $x, y, z \in S$. Then*

– $(S, +, \cdot, d, 0, 1, \mathsf{Z})$ *is a relative domain semiring,*
– $(a(S), +, \cdot, a, 0, 1)$ *is a Boolean algebra with complement a,*
– $a(S) = d(S)$,
– $d(a(x)) = a(d(x)) = a(x)$,
– $a(x) \leq a(xy)$,
– $a(x) = 1 \Leftrightarrow x \leq \mathsf{Z}$,
– $d(x)y \leq z \Leftrightarrow d(x)y \leq d(x)z$,
– $x \leq y + \mathsf{Z} \Leftrightarrow x \leq d(x)y + \mathsf{Z}$,
– $ya(z) \leq a(x)y \Leftrightarrow ya(z) = a(x)ya(z) \Leftrightarrow d(x)ya(z) = 0$,
– $d(x)y \leq yd(z) \Leftrightarrow d(x)y = d(x)yd(z) \Leftrightarrow d(x)ya(z) = d(x)y0 \Leftrightarrow d(x)ya(z) \leq y0$,
– $a(y)x \leq \mathsf{Z} \Leftrightarrow a(y) \leq a(x)$.

In particular, a is antitone, $a(1) = 0$ and $d(1) = a(0) = 1$. Using the Boolean complement of the relative domain, each computation model of Section 2 is a relative antidomain semiring where Z is any of the values 0, loop, abort or loop+abort available in the model and different from $\top$. We can therefore represent tests and their Boolean complements by domain elements $d(x)$ and their antidomain $a(x)$. The correctness claim $p \cdot R \cdot q' \leq \mathsf{Z}$ is thus expressed as $d(x)ya(z) \leq \mathsf{Z}$ in a relative antidomain semiring. By Theorem 2 this is equivalent to $d(x) \leq a(ya(z))$.

3.2 Relative Modal Operators

Domain and antidomain semirings give rise to modal diamond and box operators. We generalise them to relative (anti)domain semirings.

In a relative domain semiring the binary *diamond* operator is defined by $|x\rangle y = d(xy)$, which is the same as $d(xd(y))$. This means that its second argument is effectively a test p, and $|x\rangle p$ represents the states from which there is an execution of x that is not in Z and that terminates in p if it terminates normally. In particular, $|x\rangle p$ contains the starting states of all infinite and aborting executions of x if $\mathsf{Z} = 0$, but these executions are filtered out if $\mathsf{Z} = \mathsf{loop} + \mathsf{abort}$. The diamond operator satisfies many properties known from the unrelativised setting.

Theorem 3. *Let S be a relative domain semiring and $x, y, z \in S$ and $p, q \in d(S)$. Then*

- $|\cdot\rangle\cdot$ *is isotone,*
- $|x + y\rangle z = |x\rangle z + |y\rangle z$,
- $|x\rangle y + z = |x\rangle y + |x\rangle z$,
- $|xy\rangle z = |x\rangle(yz) = |x\rangle|y\rangle z$,
- $|x\rangle(pq) = |x\rangle p \cdot |x\rangle q$,
- $|px\rangle y = p|x\rangle y$,
- $p|x\rangle q \leq \mathsf{Z} \Leftrightarrow pxq \leq \mathsf{Z}$,
- $|x\rangle q \leq p \Leftrightarrow xq \leq px + \mathsf{Z}$.

In a relative antidomain semiring the dual *box* operator is defined by $|x]y = a(xa(y))$. Again its second argument is effectively a test p, and $|x]p$ represents the states from which all executions are in Z or terminate in p. Also the box operator satisfies many properties known from the unrelativised setting.

Theorem 4. *Let S be a relative antidomain semiring and $x, y, z \in S$ and $p, q \in d(S)$. Then*

- $|x]y = a(|x\rangle a(y))$,
- $|x\rangle y = a(|x]a(y))$,
- $|x]\cdot$ *is isotone,*
- $|\cdot]x$ *is antitone,*
- $|x + y]z = |x]z \cdot |y]z$,
- $|xy]z = |x]|y]z$,
- $|x](pq) = |x]p \cdot |x]q$,
- $|px]y = a(p) + |x]y$,
- $|x\rangle q \leq p \Leftrightarrow a(p)xq \leq \mathsf{Z}$,
- $p \leq |x]q \Leftrightarrow pxa(q) \leq \mathsf{Z} \Rightarrow px \leq xq + \mathsf{Z}$.

Consequently, the correctness claim $p \cdot x \cdot q' \leq \mathsf{Z}$ with tests p and q is formalised by $p \leq |x]q$.

It is known that the box operator corresponds to wlp in partial correctness models [33] and to wp in general correctness [34] and total correctness models [31]. 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, for extended designs and the model of Section 2.5,
– with $\mathsf{Z} = \mathsf{loop}$ box corresponds to wlp in general correctness models, and to a variant of wlp, which avoids aborting executions, for extended designs and the model of Section 2.5,
– with $\mathsf{Z} = \mathsf{abort}$ box corresponds to wp in the model of Section 2.5,
– with $\mathsf{Z} = \mathsf{loop} + \mathsf{abort}$ box corresponds to wlp in the model of Section 2.5.

Similar variants of wp are observed in [20, 21] and related to different execution methods without a unified treatment; see [37] for variants of wlp.

4 Iteration

In this section we give axioms for operations that describe iteration in various computation models. They facilitate a unified semantics of while-programs as we show in Section 5.3.

A *Kleene algebra* [28] is a semiring expanded by an operation * satisfying the axioms

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

It follows that y^*z is the least fixpoint of $\lambda x.yx + z$ and that zy^* is the least fixpoint of $\lambda x.xy + z$. The Kleene star describes finite iteration, but is not appropriate for models with infinite executions which require other fixpoints. We therefore use the following, more general structure.

An *itering* [16] is a semiring expanded by an operation $^\circ$ satisfying the axioms

$$(x + y)^\circ = (x^\circ y)^\circ x^\circ \qquad zx \leq yy^\circ z + w \Rightarrow zx^\circ \leq y^\circ(z + wx^\circ)$$
$$(xy)^\circ = 1 + x(yx)^\circ y \qquad xz \leq zy^\circ + w \Rightarrow x^\circ z \leq (z + x^\circ w)y^\circ$$

The equations are the sumstar and productstar axioms of [6]. The other two axioms generalise simulation properties such as $zx \leq yz \Rightarrow zx^\circ \leq y^\circ z$, which is known in Kleene algebra and omega algebra [5]. Its dual $xz \leq zy \Rightarrow x^\circ z \leq zy^\circ$ holds in Kleene algebras, but not in other target models, whence we weaken its consequent. Properties of the operation $^\circ$ are shown in the following result.

Theorem 5. *Let S be an itering and $x, y, z \in S$. Then $^\circ$ is isotone and*

– $0^\circ = 1 \leq (x0)^\circ = 1 + x0 \leq x^\circ$,
– $x^\circ = x^\circ x^\circ = (x^\circ x)^\circ = 1 + xx^\circ = 1 + x^\circ x$,
– $x \leq xx^\circ = x^\circ x \leq x^\circ$,
– $x^\circ \leq x^\circ 1^\circ = 1^\circ x^\circ = (1 + x)^\circ = x^{\circ\circ} = x^{\circ\circ\circ}$,
– $x^\circ y^\circ \leq (x + y)^\circ \leq (x^\circ y^\circ)^\circ = (y^\circ x^\circ)^\circ = x^\circ(y^\circ x^\circ)^\circ$,
– $(yx^\circ)^\circ = y^\circ + y^\circ yxx^\circ(yx^\circ)^\circ = (yy^\circ x^\circ)^\circ$,
– $x(yx)^\circ = (xy)^\circ x$.

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

The following result gives six models of iterings which cover all computation models of Section 2 [16].

Theorem 6. *Iterings have the following models:*

1. *Every Kleene algebra is an itering using* $x^\circ = x^*$.
2. *Every omega algebra [5] is an itering using* $x^\circ = x^\omega 0 + x^*$.
3. *Every omega algebra with* $\top x = \top$ *is an itering using* $x^\circ = x^\omega + x^*$.
4. *Every demonic refinement algebra [39] is an itering using* $x^\circ = x^\omega$.
5. *Extended designs [22, 17] form an itering using* $x^\circ = d(x^\omega)\mathsf{loop} + x^*$.
6. *The model of Section 2.5 forms an itering using* $x^\circ = n(x^\omega)\mathsf{loop} + x^*$, *where* $n(x)$ *captures the infinite executions of* x *as a test [16].*

A *modal itering* 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, the reduct $(S, +, \cdot, {}^*, 0, 1)$ is a Kleene algebra and the reduct $(S, +, \cdot, {}^\circ, 0, 1)$ is an itering. The operations * and $^\circ$ may be identical as in Kleene algebras or different as in the other models.

5 Correctness Statements

The box operator of Section 3.2 expresses various kinds of preconditions depending on the model and the value of the constant Z. In this section we give an axiomatic description of such preconditions suitable, in particular, for total correctness claims. This extends our previous work on preconditions for partial correctness [19]. We then use the preconditions to obtain a correctness calculus, pre-post specifications and loop refinement rules, all of which uniformly apply to the computation models and correctness statements of Section 2.

5.1 Tests

Preconditions are represented as tests, which we introduce first. A *test algebra* [19, 16] is a structure $(S, \cdot, ')$ satisfying the axioms

$$x'(y'z') = (x'y')z' \qquad x' = (x''y')'(x''y'')'$$
$$x'y' = y'x' \qquad x'y' = (x'y')''$$

They are derived from Huntington's axioms and make $S' = \{x' \mid x \in S\}$ a Boolean algebra with meet $\cdot$, complement $'$, order $x' \leq y' \Leftrightarrow x'y' = x'$, least element $0 = x'x''$ for any x, and greatest element $1 = 0'$. The operation $x' + y' = (x''y'')'$ is the join in S'. The extension $(S, +, \cdot, ', 0, 1)$ is also called a test algebra; elements of S' are *tests*. This axiomatisation imposes fewer constraints than antidomain semirings, which induce tests as the following result shows, without introducing a separate sort for tests.

Theorem 7. *Let* S *be a relative antidomain semiring. Then* $(S, +, \cdot, a, 0, 1)$ *is a test algebra with* $S' = d(S)$.

A test algebra is *complete* if S' is a complete Boolean algebra. Then every set of tests has a supremum in S' and the meet operation $\cdot$ on S' distributes over suprema.

5.2 Preconditions

A *precondition algebra* $(S, \cdot, \mathbin{«}, ')$ is a test algebra $(S, \cdot, ')$ expanded with a binary operation « satisfying the axioms

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

for $x, y \in S$ and $p, q \in S'$.

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. The axiom $p«q = (pq')' = p' + q$ reduces the precondition of tests to an implication; it is slightly stronger than our original in [19]. The remaining axioms express the effect of « on the sequential composition of elements and the conjunction of postconditions.

Theorem 8. *Let S be a precondition algebra and $x, y \in S$ and $p, q, r \in S'$. Then*

- *$x«\cdot$ is isotone,*
- *$p(x«q) = p(px«q)$,*
- *$px«q = p' + (x«q)$,*
- *$xp«q = xp«pq$,*
- *$p(p«q) = pq$,*
- *$p'(p«q) = p'$,*
- *$0«q = 1$,*
- *$1«q = q$,*
- *$xy«1 \leq x«1$,*
- *$x«q \leq x«1 \leq 1$,*
- *$p \leq x«q \wedge q \leq y«r \Rightarrow p \leq xy«r$.*

As the following result shows, the box operator expresses preconditions. Thus wp, wlp and their variants discussed in Section 3.2 are instances of «.

Theorem 9. *Let S be a relative antidomain semiring and $x, y \in S$ and $p, q \in d(S)$. Then S is a precondition algebra with $x«q = |x]q$. Moreover,*

- *$(x + y)«q = (x«q) \cdot (y«q)$,*
- *$\cdot«q$ is antitone,*
- *$(x«q)x + \mathsf{Z} = (x«q)xq + \mathsf{Z}$,*
- *$(x«q)xq' \leq \mathsf{Z}$,*
- *$p \leq x«q \Leftrightarrow pxq' \leq \mathsf{Z}$,*
- *$x«1 = 1 \Leftrightarrow x0 \leq \mathsf{Z}$.*

In a complete precondition algebra S, the *progressively bounded states* of an element $x \in S$ are given by $b(x) = \sup\{x^n«0 \mid n \in \mathbb{N}\}$. The test $b(x)$ describes the states which have an upper bound on the lengths of the emerging x-transition paths. This means that for every state in $b(x)$ there is a bound n such that x can be iterated at most n times starting from the state; the bound n may depend on the state.

This should be contrasted with the progressively finite states characterised, for example, by the convergence operation of [34]. These require the absence of infinite transition paths without giving bounds on the lengths of finite paths, and coincide with the progressively bounded states for deterministic programs.

5.3 While-Programs

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

$$(x \lhd p \rhd y)\ll q = p(x\ll q) + p'(y\ll q)$$
$$(p \star x)\ll q = (x(p \star x) \lhd p \rhd 1)\ll q$$

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 [24, 26] for more comprehensive axiomatisations. The element $p \star x$ represents the while loop while p do x and the corresponding axiom describes its fixpoint unfolding, again under a postcondition. Both axioms are equations of tests, weakening our original axioms in [19]. As we show below, they hold in a wide range of computation models.

Theorem 10. *Let S be a while algebra and $x, y \in S$ and $p, q, r \in S'$. Then*

- $p((x \lhd p \rhd y)\ll q) = p(x\ll q)$,
- $p'((x \lhd p \rhd y)\ll q) = p'(y\ll q)$,
- $pq \leq x\ll r \wedge p'q \leq y\ll r \Rightarrow q \leq (x \lhd p \rhd y)\ll r$,
- $p((p \star x)\ll q) = p(x\ll(p \star x)q)$,
- $p'((p \star x)\ll q) = p'q$,
- $p' \leq (p \star x)\ll p' \leq (p \star x)\ll 1$,
- $q \leq (p \star x)\ll 1 \Leftrightarrow pq \leq (p \star x)\ll 1$.

In Section 5.4, the test $\ell = (1 \star 1)\ll 1$ helps us to treat total correctness claims and claims which 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. Claims 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 $x\ll 1 = 1$, a characteristic property of wlp [10].

As the following result shows, the operations $\lhd\rhd$ and $\star$ can be defined in modal iterings, hence in all models given in Section 2. This gives a unified semantics of while-programs.

Theorem 11. *Let S be a modal itering and $x, y \in S$ and $p \in d(S)$. Then S is a while algebra with $x \lhd p \rhd y = px + a(p)y$ and $p \star x = (px)^\circ a(p)$. In particular, $\ell = a(1^\circ 0)$.*

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. *Assertions* are test expressions extended by preconditions; they are constructed from test expressions and while-programs by the operations $'$ for negation, $\cdot$ for conjunction and $\ll$ for preconditions. Hence they are tests and closed under 0, 1, ℓ, finite sums and finite products.

5.4 Correctness Calculus

A *correctness algebra* is a complete while algebra satisfying the additional axiom

$$pq \leq x \ll q \Rightarrow q\ell \leq (p \star x) \ll p'q$$

for $x \in S$ and $p, q \in S'$. It expresses soundness of the partial correctness rule for while loops in the correctness calculus.

A *correctness statement* $p\{x\}q$ is composed of a while-program x and two assertions p and q. The statement $p\{x\}q$ is *valid* if and only if $p \leq x \ll q$. Its meaning depends on the model and the interpretation of the precondition operation $\ll$. In some models, $p \leq x \ll q$ amounts to partial correctness, that is, all finite executions of x starting in p establish the postcondition q. In other models, $p \leq x \ll q$ amounts to total correctness, which additionally requires that all executions of x starting in p are finite. In yet other models, $p \leq x \ll q$ 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, assertions q, r, s and t, and tests t_i:

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

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

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

$$\text{(while)} \quad \frac{pq\{x\}q \quad q \leq \ell + t_{<\infty} \quad \forall n \in \mathbb{N} : t_n pq\{x\}\ell + t_{<n}}{q\{p \star x\}p'q}$$

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

The rule for while loops is abstracted from the Hoare calculus for total correctness [1]. The test $t_{<n}$ is defined by $t_{<n} = \sup\{t_i \mid 0 \leq i < n\}$ for $n \in \mathbb{N} \cup \{\infty\}$. 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 $q \leq \ell + t_{<\infty}$ expresses that the bound is non-negative while the invariant q holds, and $t_n pq\{x\}\ell + t_{<n}$ expresses that every iteration decreases the

bound. If $\ell = 1$, the premises simplify to $pq\{x\}q$, which expresses that the loop invariant q is preserved by the loop body. The rule concludes that the invariant is preserved by the while loop.

Theorem 12. *The calculus is sound, that is, only valid correctness statements can be derived. Let $(p \star x)\text{«}1 \leq \ell + b(px)$ for every test expression p and while-program x. Then the calculus is complete, that is, every valid correctness statement can be derived.*

The condition for completeness is satisfied if the body px of a while loop is boundedly non-deterministic in total correctness models. Namely, $(p \star x)\text{«}1$ contains the states from which the loop $p \star x$ has only finite executions, and they have to be among the progressively bounded states $b(px)$ if $\ell = 0$.

As usual, completeness is relative to having all true inequalities $p \leq q$ available in the calculus. The bound function $t_i = (px)^i\text{«}0$ is used in the completeness proof. Termination after a given number of iterations is described by domain elements in [18].

Our calculus unifies and generalises previous algebraic calculi for partial, total and general correctness [29, 33, 34, 13, 14, 19]. In particular, it applies to further computation models and facilitates total correctness claims by algebraically representing the bound function.

Example 13. We prove correctness of a program for integer division along the lines of [1], 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:

- $p_1\{z_1\}p_1p_2$ since z_1 does not affect y and $r = 0 \times y + r$ holds,
- $p_3p_1p_2\{z_2\}p_1p_2$ since z_2 does not affect y and $x = q \times y + r$ implies $x = (q + 1) \times y + (r - y)$, and
- $t_np_1p_3\{z_2\}t_{<n}$ for each $n \in \mathbb{N}$ since $n = r \geq y > 0$ implies $r - y = n - y < n$; more precisely, t_{n-y} is established.

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

$$\dfrac{p_1\{z_1\}p_1p_2 \qquad \dfrac{p_3p_1p_2\{z_2\}p_1p_2 \quad p_1p_2 \leq \ell + t_{<\infty} \quad \dfrac{\forall n \in \mathbb{N} : t_np_1p_3\{z_2\}t_{<n}}{\forall n \in \mathbb{N} : t_np_3p_1p_2\{z_2\}\ell + t_{<n}}}{p_1p_2\{p_3 \star z_2\}p_3'p_1p_2}}{\dfrac{p_1\{z_1(p_3 \star z_2)\}p_3'p_1p_2}{p_1\{z_1(p_3 \star z_2)\}p_3'p_2}}$$

using the loop invariant $p_1 p_2$ 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 our computation models and for any $\mathsf{Z} \in \{0, \mathsf{loop}, \mathsf{abort}, \mathsf{loop} + \mathsf{abort}\} \setminus \{\top\}$.

The following result shows how to define correctness statements in modal iterings subjected to two additional axioms.

Theorem 14. *Let S be a modal itering – which is a test algebra, a precondition algebra and a while algebra according to Theorems 7, 9 and 11 – such that the test algebra is complete and*

$$x\mathsf{Z} \leq x0 + \mathsf{Z} \qquad |x^*]y \leq |\ell x^\circ]y$$

for each $x, y \in S$. Then S is a correctness algebra. Moreover, the induction laws

$$p \leq |x]p \Rightarrow p \leq |x^*]p$$
$$p \leq |x]p \Rightarrow \ell p \leq |x^\circ]p$$

hold for $x \in S$ and $p \in d(S)$.

In particular, $p\{x\}q$ is valid if and only if $p \leq |x]q$. The axiom $x\mathsf{Z} \leq x0 + \mathsf{Z}$ separates the executions of x in the composition $x\mathsf{Z}$. All finite executions of x reach Z; this part is subsumed by Z. The executions of x which do not reach Z because they are infinite or abort are subsumed by $x0$. The axiom $|x^*]p \leq |\ell x^\circ]p$ expresses that for claims not involving termination, whence $\ell = 1$, iteration reduces to finite iteration as infinite executions are ignored.

5.5 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 $p \dashv q$ [30, 36, 35, 39] 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. 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 semiring, the reduct $(S, +, \cdot, «, ', 0, 1)$ is a precondition algebra, and the operation $\dashv$ satisfies

$$x \leq p \dashv q \Leftrightarrow p \leq x \mathbin{«} q$$

for $x \in S$ and $p, q \in S'$. This axiom is an order-reversing Galois connection between S and S'.

Theorem 15. *Let S be a pre-post algebra and $x, y \in S$ and $p, q, r, s \in S'$. Then*

- $\cdot\mathbin{«} q$ *is antitone,*
- $\cdot\mathbin{⊣} q$ *is antitone,*
- $p\mathbin{⊣}\cdot$ *is isotone,*
- $(x + y)\mathbin{«} q = (x\mathbin{«} q) \cdot (y\mathbin{«} q)$,
- $pq\mathbin{⊣} r = (p\mathbin{⊣} r) + (q\mathbin{⊣} r)$,
- $p\mathbin{⊣}(q + r) = (p\mathbin{⊣} q) + (p\mathbin{⊣} r)$,
- $x \leq (x\mathbin{«} q)\mathbin{⊣} q$,
- $p \leq (p\mathbin{⊣} q)\mathbin{«} q$,
- $(p\mathbin{⊣} q)r = (p\mathbin{⊣} qr)r = (p\mathbin{⊣}(q + r'))r$,
- $r(p\mathbin{⊣} q) = r(rp\mathbin{⊣} q) = r((r' + p)\mathbin{⊣} q)$,
- $p\mathbin{⊣} q = (1\mathbin{⊣} q) + p'\top$,
- $p(p\mathbin{⊣} q) = p(1\mathbin{⊣} q)$,
- $p'(p\mathbin{⊣} q) = p'\top$,
- $(1\mathbin{⊣} q)\mathbin{«} q = 1 \leq p\mathbin{⊣} p$,
- $0\mathbin{⊣} q = \top$,
- $q \leq r \Rightarrow (p\mathbin{⊣} q)(r\mathbin{⊣} s) \leq p\mathbin{⊣} s$,
- $(p\mathbin{⊣} q)(q\mathbin{⊣} r) \leq p\mathbin{⊣} r$,
- $(p\mathbin{⊣} p)(p\mathbin{⊣} q) = (p\mathbin{⊣} q)(q\mathbin{⊣} q) = p\mathbin{⊣} q$,
- $(p\mathbin{⊣} p)(p\mathbin{⊣} p) = p\mathbin{⊣} p$,
- $x\mathbin{«} 1 = 1 \Leftrightarrow x \leq 1\mathbin{⊣} 1$,
- $x \leq pq\mathbin{⊣} r \Leftrightarrow px \leq q\mathbin{⊣} r$.

Example 16. In a structure which is both a pre-post algebra and a correctness algebra, such as every model in Section 2, the correctness rule for while loops translates as

$$x \leq pq\mathbin{⊣} q \;\wedge\; q \leq \ell + t_{<\infty} \;\wedge\; (\forall n \in \mathbb{N} : x \leq t_n pq\mathbin{⊣} \ell + t_{<n}) \;\Rightarrow\; p \star x \leq q\mathbin{⊣} p'q\ .$$

This rule introduces a while loop by refining a pre-post specification. We give two instances in the computation model of Section 2.5. The first instance is a partial correctness rule obtained by setting $\mathsf{Z} = \top 0$, whence $\ell = 1$ by Theorem 9 and therefore

$$x \leq pq\mathbin{⊣} q \;\Rightarrow\; p \star x \leq q\mathbin{⊣} p'q$$

because $pq\mathbin{⊣} q \leq t_n pq\mathbin{⊣} 1$ by Theorem 15. The second instance is a total correctness rule obtained by setting $\mathsf{Z} = 0$, whence $\ell = 0$. Using $q = t_{<\infty}$, a consequence of the rule is

$$r \leq t_{<\infty} \;\wedge\; (\forall n \in \mathbb{N} : x \leq t_n p\mathbin{⊣} t_{<n}) \;\Rightarrow\; p \star x \leq r\mathbin{⊣} 1\ .$$

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 [13, 12]. Moreover, by using $\mathsf{Z} = \mathsf{loop}$ 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.

The following result shows how to define pre-post specifications in antidomain semirings subjected to two additional axioms taken from [15]. They are equivalent to $x0 \leq y \Leftrightarrow x \leq y + \mathsf{H}$ and introduce the element H representing the program havoc, which is the greatest program that has only finite executions.

Theorem 17. *Let S be a bounded relative antidomain semiring, which is a precondition algebra according to Theorem 9. Let $\mathsf{H} \in S$ such that*

$$\mathsf{H}0 = 0 \qquad x \leq x0 + \mathsf{H}$$

for each $x \in S$. Then S is a pre-post algebra with $p \dashv q = \mathsf{Z} + a(p)\top + \mathsf{H}q$.

6 Conclusion

Five computation models with varying representations of finite, infinite and aborting executions support similar correctness statements. These are captured uniformly by a relativisation of domain semirings and, more generally, by an algebra for preconditions. It facilitates the definition of correctness claims, their calculus and pre-post specifications in a unified way for various models and correctness statements.

Future work concerns a question raised by a referee, namely whether the approach can be extended to general refinement algebra [39] which models dual non-determinism.

Acknowledgement. I thank Jeremy Gibbons and the anonymous referees for providing helpful comments.

References

1. Apt, K.R., de Boer, F.S., Olderog, E.R.: Verification of Sequential and Concurrent Programs. Springer, third edn. (2009)
2. de Bakker, J.W.: Semantics and termination of nondeterministic recursive programs. In: Michaelson, S., Milner, R. (eds.) Automata, Languages and Programming: Third International Colloquium. pp. 435–477. Edinburgh University Press (1976)
3. Berghammer, R., Zierer, H.: Relational algebraic semantics of deterministic and nondeterministic programs. Theor. Comput. Sci. 43, 123–147 (1986)
4. Broy, M., Gnatz, R., Wirsing, M.: Semantics of nondeterministic and noncontinuous constructs. In: Bauer, F.L., Broy, M. (eds.) Program Construction. LNCS, vol. 69, pp. 553–592. Springer (1979)
5. Cohen, E.: Separation and reduction. In: Backhouse, R., Oliveira, J.N. (eds.) MPC 2000. LNCS, vol. 1837, pp. 45–59. Springer (2000)
6. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall (1971)
7. De Carufel, J.L., Desharnais, J.: Demonic algebra with domain. In: Schmidt, R. (ed.) RelMiCS/AKA 2006. LNCS, vol. 4136, pp. 120–134. Springer (2006)
8. Desharnais, J., Möller, B., Struth, G.: Kleene algebra with domain. ACM Transactions on Computational Logic 7(4), 798–833 (2006)

9. Desharnais, J., Struth, G.: Internal axioms for domain semirings. Sci. Comput. Program. 76(3), 181–203 (2011)
10. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall (1976)
11. Dunne, S.: Recasting Hoare and He's Unifying Theory of Programs in the context of general correctness. In: Butterfield, A., Strong, G., Pahl, C. (eds.) 5th Irish Workshop on Formal Methods. Electronic Workshops in Computing, The British Computer Society (2001)
12. Dunne, S.E., Hayes, I.J., Galloway, A.J.: Reasoning about loops in total and general correctness. In: Butterfield, A. (ed.) UTP 2008. LNCS, vol. 5713, pp. 62–81. Springer (2010)
13. Guttmann, W.: General correctness algebra. In: Berghammer, R., Jaoua, A.M., Möller, B. (eds.) RelMiCS/AKA 2009. LNCS, vol. 5827, pp. 150–165. Springer (2009)
14. Guttmann, W.: Partial, total and general correctness. In: Bolduc, C., Desharnais, J., Ktari, B. (eds.) MPC 2010. LNCS, vol. 6120, pp. 157–177. Springer (2010)
15. Guttmann, W.: Unifying recursion in partial, total and general correctness. In: Qin, S. (ed.) UTP 2010. LNCS, vol. 6445, pp. 207–225. Springer (2010)
16. Guttmann, W.: Algebras for iteration and infinite computations (submitted, 2011)
17. Guttmann, W.: Extended designs algebraically. Sci. Comput. Program. (to appear, 2012)
18. Guttmann, W., Möller, B.: Normal design algebra. Journal of Logic and Algebraic Programming 79(2), 144–173 (2010)
19. Guttmann, W., Struth, G., Weber, T.: Automating algebraic methods in Isabelle. In: Qin, S., Qiu, Z. (eds.) ICFEM 2011. LNCS, vol. 6991, pp. 617–632. Springer (2011)
20. Harel, D.: First-Order Dynamic Logic, LNCS, vol. 68. Springer (1979)
21. Harel, D.: On the total correctness of nondeterministic programs. Theor. Comput. Sci. 13(2), 175–192 (1981)
22. Hayes, I.J., Dunne, S.E., Meinicke, L.: Unifying theories of programming that distinguish nontermination and abort. In: Bolduc, C., Desharnais, J., Ktari, B. (eds.) MPC 2010. LNCS, vol. 6120, pp. 178–194. Springer (2010)
23. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM 12(10), 576–580/583 (1969)
24. Hoare, C.A.R., Hayes, I.J., He, J., Morgan, C.C., Roscoe, A.W., Sanders, J.W., Sorensen, I.H., Spivey, J.M., Sufrin, B.A.: Laws of programming. Commun. ACM 30(8), 672–686 (1987)
25. Hoare, C.A.R., He, J.: Unifying theories of programming. Prentice Hall Europe (1998)
26. Jackson, M., Stokes, T.: Semigroups with if-then-else and halting programs. International Journal of Algebra and Computation 19(7), 937–961 (2009)
27. Jacobs, D., Gries, D.: General correctness: A unification of partial and total correctness. Acta Inf. 22(1), 67–83 (1985)
28. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation 110(2), 366–390 (1994)
29. Kozen, D.: On Hoare logic and Kleene algebra with tests. ACM Transactions on Computational Logic 1(1), 60–76 (2000)
30. Meertens, L.: Abstracto 84: The next generation. In: Martin, A.L., Elshoff, J.L. (eds.) ACM '79: Proceedings of the 1979 annual conference. pp. 33–39. ACM Press (1979)
31. Möller, B.: The linear algebra of UTP. In: Uustalu, T. (ed.) MPC 2006. LNCS, vol. 4014, pp. 338–358. Springer (2006)

32. Möller, B.: Kleene getting lazy. Sci. Comput. Program. 65(2), 195–214 (2007)
33. Möller, B., Struth, G.: Algebras of modal operators and partial correctness. Theor. Comput. Sci. 351(2), 221–239 (2006)
34. Möller, B., Struth, G.: WP is WLP. In: MacCaull, W., Winter, M., Düntsch, I. (eds.) RelMiCS 2005. LNCS, vol. 3929, pp. 200–211. Springer (2006)
35. Morgan, C.: The specification statement. ACM Trans. Progr. Lang. Syst. 10(3), 403–419 (1988)
36. Morris, J.M.: A theoretical basis for stepwise refinement and the programming calculus. Sci. Comput. Program. 9(3), 287–306 (1987)
37. Morris, J.M.: Varieties of weakest liberal preconditions. Inf. Process. Lett. 25(3), 207–210 (1987)
38. Nelson, G.: A generalization of Dijkstra's calculus. ACM Trans. Progr. Lang. Syst. 11(4), 517–561 (1989)
39. von Wright, J.: Towards a refinement algebra. Sci. Comput. Program. 51(1–2), 23–45 (2004)