Extended Designs Algebraically

Walter Guttmann

Department of Computer Science, University of Sheffield, UK
walter.guttmann@uni-ulm.de

Abstract

Extended designs distinguish non-terminating and aborting executions of sequential, non-deterministic programs. We show how to treat them algebraically based on techniques we have previously applied to total and general correctness approaches. In particular, we propose modifications to the definition of an extended design which make the theory more clear and simplify calculations, and an approximation order for recursion. We derive explicit formulas for operators on extended designs including non-deterministic choice, sequential composition, while-loops and full recursion. We show how to represent extended designs as designs or prescriptions over an extended state space. The new theory generalises our previous algebraic theory of general correctness by weakening its axioms. It also integrates with partial, total and general correctness into a common foundation which gives a unified semantics of while-programs. Program transformations derived using this semantics are valid in all four correctness approaches.

Keywords: domain operation, Egli-Milner order, fixpoint, general correctness, program semantics, recursion, semiring, while-program

1. Introduction

Models of total and general correctness are able to represent non-terminating executions of programs. Some approaches, including the Unifying Theories of Programming, use this facility also to signify the abortion of a program [22, Chapter 3]. Abortion means abnormal termination, for example, caused by expressions with undefined values such as integer division by zero.

Using the same representation for abortion and non-termination is adequate from a specification point of view where both are undesired. However, abortion and non-termination are different concepts from several other point of views as we argue in [14]. For example, the abortion of a program execution can be observed in contrast to non-termination. One way to distinguish them is by extending the value ranges of program variables with special values, say ∞ and $\bot$ for non-termination and abortion, respectively. Another way is by extending the state space with Boolean variables representing the respective observations; this is proposed in [19] using 'extended designs'. Different termination conventions are discussed and compared in [20].

Designs [22] and prescriptions [10] use the Boolean variable ok to represent normal termination of a program. Extended designs add the Boolean variable tm to distinguish non-termination from abortion. More precisely, ok and tm give the values before the execution of a program, while the values after execution are held in ok' and tm'.

The first purpose of the present paper is to develop the theory of extended designs. We do this at three levels as described in the following.

Section 2 is concerned with the definition of extended designs as given by [19] in terms of predicates. Previous works [29, 13, 18] represent designs and prescriptions as matrices with components according to the values of ok and ok'; this is briefly reviewed in Sections 2.1 and 2.2. In its conclusion [18] predicts the applicability of the method to further observations. Section 2.3 delivers by extending the matrices with components according to tm and tm' to obtain a representation of extended designs. Based on our findings we propose two changes to the definition of extended designs. Moreover, we show how to represent extended designs both as designs and as prescriptions.

For calculating the effect of operations on extended designs it is convenient to replace the matrix entries by elements of semirings. This second, more abstract level of the theory is the topic of Section 3. The effect of non-deterministic choice, sequential composition, domain, Kleene star and the omega operation on extended designs is given in Sections 3.1–3.3. We thus obtain a rich algebraic structure. In Section 3.4 we propose an approximation order for extended designs based on their representation as prescriptions.

To treat full recursion, we further abstract from the semiring matrices to their algebraic structure as derived in Section 3. This forms the third level of the theory, at which we operate in Section 4. We generalise our previous works [13, 15] by weakening their axioms, which are recapitulated in Section 4.1. The reduced axioms are still sufficient to represent the approximation order in Section 4.2 and the solution of recursion equations in Sections 4.3 and 4.4.

The second purpose of the present paper is to integrate extended designs with an algebraic theory of partial, total and general correctness.

This is based on our unified semantics of while-programs for partial, total and general correctness [15]. Section 5 adapts it to include iterations of extended designs with the semantics derived in Section 4.4. We generalise the program transformations obtained for our previous unified semantics to include extended designs. As a consequence, we obtain a proof of the normal form theorem for while-programs which is valid for relations, designs, prescriptions, extended designs and any other model satisfying the common axioms.

In Section 3 it is useful to switch the convention, whereby an extended design exhibits states with guaranteed termination, to the complementary one. Consequences of returning to the original convention are discussed in Appendix A.

2. The matrix approach

In this section, we recall the matrix approach to designs and prescriptions and apply it to extended designs. Two changes to the definition of extended designs are suggested this way. We also obtain representations of extended designs as designs and as prescriptions.

As in [22, 10, 19] specifications are given as predicates over program variables v and auxiliary variables such as ok and tm. The predicates relate the initial values v, ok and tm to the final values v', ok' and tm'.

2.1. Designs

Let P and R be predicates over program variables, that is, not referring to auxiliary variables. Then the design

$$P \vdash R \;\Leftrightarrow_{\mathrm{def}}\; ok \wedge P \Rightarrow ok' \wedge R$$

is a specification with the following intuitive meaning: if the assumption P holds, then every possible execution terminates and achieves the effect R [22]. It follows that R is relevant only if P is true. In this paper, we focus on the subclass of normal designs, which require P to depend only on the initial values v and not on the final values v' of the program variables [10]; by 'designs' we refer to normal designs in the following.

The auxiliary variable ok indicates whether the execution has been started, while ok' indicates whether it has terminated. Thus ok is true in the initial state, and in an intermediate state if the predecessor has terminated. No distinction is made between non-termination and abortion as the following two examples show. The endless loop, that is, the solution of the recursive specification $X = X$ is the design $\mathit{false} \vdash \mathit{false}$ whose assumption is false. A division by zero may be prevented by an appropriate assumption, too, as in $y \neq 0 \vdash x' = x/y \wedge y' = y$. The intention is that in a state with $y = 0$ the latter design aborts and the former design does not terminate. But in such a state neither assumption holds, whence the effects of the designs are irrelevant and there is no clue whether abortion or non-termination should happen.

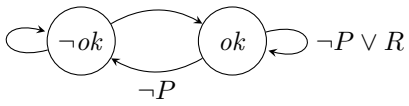
According to [29] the design $P \vdash R$ can be represented by a 2×2 matrix with entries given by instantiating the predicate with the possible values of ok and ok'. We annotate each row and column to clarify the corresponding values:

	$\neg ok'$	ok'
$\neg ok$	true	true
ok	$\neg P$	$\neg P \vee R$

The entry in row ok and column ok' shows that an execution might terminate also if P does not hold: nothing is said about executions in this case. Each entry in the top row is the same for every design; we call such entries 'structural'.

The benefits of the matrix representation will become evident in the remainder of this paper. In particular, operations on specifications such as non-deterministic choice, sequential composition and finite iteration reduce to familiar matrix operators and constructions.

The matrix of a design can also be regarded as specifying the transitions of a two-state automaton:



Edges with the predicate *true* are not labelled. The state ok represents termination; if $\neg P$ holds, the system may move to the state $\neg ok$ representing abortion or non-termination. Typical for designs, non-termination cannot be forced: there is always the possibility that the execution terminates, which is represented by the *true* edge leading back to the ok state.

This automaton gives an abstract view of a computation focused on its termination or non-termination, and disregarding intermediate states. A design describes a state transition of the automaton: the value of the auxiliary variable ok indicates the state before the transition and the value of ok' the state after the transition. Sequences of transitions correspond to the iteration of a computation discussed in Section 3.3.

2.2. Prescriptions

While designs are adequate for total correctness, for general correctness [23] a variant is needed which can represent terminating and non-terminating executions independently. The prescription

$$P \Vdash R \;\Leftrightarrow_{\text{def}}\; (tm \wedge P \Rightarrow tm') \wedge (tm' \Rightarrow R \wedge tm)$$

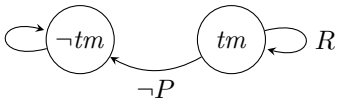
is a specification with the following intuitive meaning: if the assumption P holds, then every possible execution terminates; in any case, if an execution terminates, it achieves the effect R [10]. In the following, by 'prescriptions' we again refer to normal prescriptions, which require P to depend only on the initial values v of the program variables.

For prescriptions the auxiliary variable tm has the same role as ok for designs (and in fact was called ok originally, but in the context of extended designs this is no longer appropriate): tm indicates whether the execution has been started, while tm' indicates whether it has terminated. There is still no distinction between non-termination and abortion. Continuing the above examples, the endless loop is the prescription $false \Vdash false$ and division by zero may be prevented in $y \neq 0 \Vdash y \neq 0 \wedge x' = x/y \wedge y' = y$. In a state with $y = 0$ both prescriptions denote the same again.

The prescription $P \Vdash R$ can also be represented according to [29] by a 2×2 matrix, now with respect to the possible values of tm and tm':

	$\neg tm'$	tm'
$\neg tm$	*true*	*false*
tm	$\neg P$	R

It differs from the design matrix in the tm' column to the effect that non-terminating and terminating executions are now independent. In particular, one of the structural entries, which are again in the top row, is *false* for prescriptions. This is also reflected in the corresponding automaton:



Edges with the predicate *false* are omitted since the corresponding transitions cannot be taken. For prescriptions $\neg tm$ represents forced abortion or non-termination: once this state is reached it cannot be escaped. This is because any subsequent computation starts in the state where the current computation has finished, in that case $\neg tm$.

2.3. Extended designs

A generalisation is necessary to distinguish abortion and non-termination. To achieve this, extended designs are introduced by [19]. We first describe the original definition and restrictions imposed on extended designs. Afterwards we propose two modifications and give a matrix representation and representations in terms of designs and prescriptions. The extended design

$$P \vdash_{\mathrm{X}} R \;\Leftrightarrow_{\mathrm{def}}\; (ok \wedge tm \wedge P \Rightarrow ok' \wedge R) \wedge (\neg tm' \Rightarrow ok') \wedge (tm' \Rightarrow tm)$$

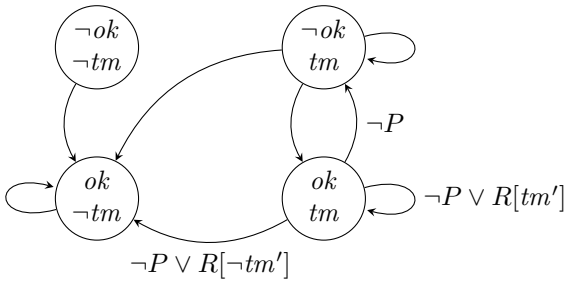
is a specification with the following intuitive meaning: if the assumption P holds, then every possible execution does not abort and achieves the effect R; the additional conjuncts are explained below. Here, the predicate P may refer to the initial values v of the program variables only, but not to the auxiliary variables. The predicate R may refer to v and v' and the final value tm', but not to ok, ok' and tm. The auxiliary variable tm indicates whether the execution terminates, and ok indicates whether it does not abort. (For designs ok indicates both and for prescriptions tm indicates both because abortion and non-termination are not distinguished.)

An extended design contains two additional conjuncts: $\neg tm' \Rightarrow ok'$ states that a non-terminating execution cannot abort, and $tm' \Rightarrow tm$ states that an execution can terminate only if it has started (its predecessor has terminated). Furthermore, the healthiness condition $(\forall v, v', tm' : P \wedge \neg tm' \Rightarrow (R \Leftrightarrow \forall v' : R))$ is imposed on an extended design by [19] to the effect that R may not constrain the final values of the program variables in the case of non-termination. The impact of this healthiness condition is made clear in Section 2.3.2.

We carry on the matrix representation from designs and prescriptions to extended designs. To this end, we instantiate the predicate $P \vdash_{\mathrm{X}} R$ by the possible values of ok, ok', tm and tm'. This results in the following 4×4 matrix:

	$\neg ok' \wedge \neg tm'$	$\neg ok' \wedge tm'$	$ok' \wedge \neg tm'$	$ok' \wedge tm'$
$\neg ok \wedge \neg tm$	*false*	*false*	*true*	*false*
$\neg ok \wedge tm$	*false*	*true*	*true*	*true*
$ok \wedge \neg tm$	*false*	*false*	*true*	*false*
$ok \wedge tm$	*false*	$\neg P$	$\neg P \vee R[\neg tm']$	$\neg P \vee R[tm']$

The predicate $R[tm'] =_{\mathrm{def}} \exists tm' : tm' \wedge R$ specialises R to the case where tm' holds, and similarly $R[\neg tm'] =_{\mathrm{def}} \exists tm' : \neg tm' \wedge R$ for the case $\neg tm'$. All entries except the second, third and fourth of the fourth row are structural. The corresponding four-state automaton is



Based on these representations we propose two changes to extended designs.

2.3.1. An invariant for extended designs

The first modification concerns the conjunct $\neg tm' \Rightarrow ok'$ every extended design satisfies. Observe that it refers only to the final values of (auxiliary) variables. Therefore, every predicate which is sequentially post-composed to an extended design can be assumed to satisfy the predicate $\neg tm \Rightarrow ok$, which is the relational converse of $\neg tm' \Rightarrow ok'$. In particular, we have

$$(P_1 \vdash_{\mathrm{X}} R_1)\,;\,(P_2 \vdash_{\mathrm{X}} R_2) \;\Leftrightarrow\; (P_1 \vdash_{\mathrm{X}} R_1)\,;\,((P_2 \vdash_{\mathrm{X}} R_2) \wedge (\neg tm \Rightarrow ok))$$

for all extended designs $P_1 \vdash_{\mathrm{X}} R_1$ and $P_2 \vdash_{\mathrm{X}} R_2$.

Adding the conjunct $\neg tm \Rightarrow ok$ to an extended design changes the structural entry in row one, column three from *true* to *false*. It is readily verified that this change has no effect on the three non-structural entries of the matrix whenever extended designs are composed by non-deterministic choice or sequential composition.

In fact, observe that the first row of the original matrix is a copy of the third row. This means that the corresponding states can be regarded as equivalent except for the very beginning of each computation. Hence if we take the two states with $\neg tm$ as equivalent initially, they are not distinguishable from the outside.

The intuition behind this modification is the same as that for adding the original $\neg tm' \Rightarrow ok'$. In the case of non-termination $\neg tm$, it is not reasonable to distinguish abortion $\neg ok$ from normal termination ok. We thus obtain the modified extended design

$$P \vdash'_{\mathrm{X}} R \;\Leftrightarrow_{\mathrm{def}}\; (ok \wedge tm \wedge P \Rightarrow ok' \wedge R) \wedge (\neg tm \Rightarrow ok) \wedge (\neg tm' \Rightarrow ok') \wedge (tm' \Rightarrow tm)\ .$$

In effect, $\neg tm \Rightarrow ok$ becomes an invariant valid at all steps during an execution, which completely excludes the state $\neg tm \wedge \neg ok$. The remaining three states are

ok	tm	intuitive meaning
false	*true*	abortion
true	*false*	non-termination
true	*true*	normal termination

2.3.2. Non-termination and termination

The second modification concerns the conjunct $ok \wedge tm \wedge P \Rightarrow ok' \wedge R$ of extended designs, and in particular the effect R. Observe that the entries three and four of row four of the extended design matrix are very similar. The only difference between $\neg P \vee R[\neg tm']$ and $\neg P \vee R[tm']$ is the use of $\neg tm'$ versus tm'. Yet these two predicates describe conceptually different facts. The first gives the conditions under which the execution may run forever, while the second gives the possible effects in case of normal termination. These two are just as different as the predicates P and R of a prescription and therefore should be denoted separately.

A closer look reveals that the healthiness condition $(\forall v, v', tm' : P \wedge \neg tm' \Rightarrow (R \Leftrightarrow \forall v' : R))$ in fact restricts the predicate R in the entry $\neg P \vee R[\neg tm']$ to be a condition, that is, to refer to the initial values v only. This is due to the term $(R \Leftrightarrow \forall v' : R)$ by which R relates v with every v' if it relates v with some v'. However, the healthiness condition has no effect on R in the other entry $\neg P \vee R[tm']$, where it may refer to the final values v' as well. The restriction only applies in the case of non-termination $\neg tm'$ [19]. Hence the impact of the healthiness condition is just as the restriction of prescriptions to normal prescriptions. This too suggests the separate denotation of the two predicates.

We therefore separate the predicate R of the original extended designs into two predicates Q and S. As P, the predicate Q may refer to the initial values v of the program variables only. The predicate S may refer to v and v'. Neither Q nor S may refer to the auxiliary variables. We obtain the further modified extended design now with three components

$$P|Q|S \;\Leftrightarrow_{\mathrm{def}}\; (ok \wedge tm \wedge P \Rightarrow ok' \wedge ((\neg tm' \wedge \neg Q) \vee (tm' \wedge S))) \wedge (\neg tm \Rightarrow ok) \wedge (\neg tm' \Rightarrow ok') \wedge (tm' \Rightarrow tm)\ .$$

We use $\neg Q$ instead of Q to stay in analogy with the predicate P, which also appears negated. Every extended design, modified as in Section 2.3.1, can be expressed this way by using $Q = \neg R[\neg tm']$ and $S = R[tm']$ since then

$$(\neg tm' \wedge \neg Q) \vee (tm' \wedge S) \;\Leftrightarrow\; (\neg tm' \wedge \exists tm' : \neg tm' \wedge R) \vee (tm' \wedge \exists tm' : tm' \wedge R) \;\Leftrightarrow\; (\neg tm' \wedge R) \vee (tm' \wedge R) \;\Leftrightarrow\; R$$

holds. Conversely, every three-component extended design can be expressed as a modified extended design simply by $R = (\neg tm' \wedge \neg Q) \vee (tm' \wedge S)$. This shows that our second modification does not change the class of extended designs; it just separates the concerns of non-terminating and terminating executions.

2.3.3. New extended designs

Observe that the predicate $(\neg tm' \wedge \neg Q) \vee (tm' \wedge S)$, which occurs as a part of our new extended design, is equivalent to $(Q \Rightarrow tm') \wedge (tm' \Rightarrow S)$. This is readily verified by case distinction on tm'. Thus we can reformulate according to

$$P|Q|S \;\Leftrightarrow\; (ok \wedge tm \wedge P \Rightarrow ok' \wedge (Q \Rightarrow tm') \wedge (tm' \Rightarrow S)) \wedge (\neg tm \Rightarrow ok) \wedge (\neg tm' \Rightarrow ok') \wedge (tm' \Rightarrow tm)\ .$$

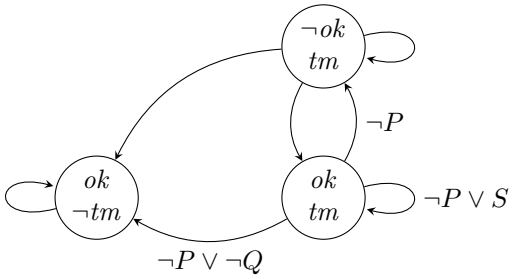
The intuition for the new extended design $P|Q|S$ is as follows:

* P characterises the states from which no execution aborts.
* Q characterises the states from which all executions terminate normally, provided they start in P.
* S characterises the effect of every execution which terminates normally, provided it starts in P.

The matrix representation of $P|Q|S$ is

	$\neg ok' \wedge \neg tm'$	$\neg ok' \wedge tm'$	$ok' \wedge \neg tm'$	$ok' \wedge tm'$
$\neg ok \wedge \neg tm$	*false*	*false*	*false*	*false*
$\neg ok \wedge tm$	*false*	*true*	*true*	*true*
$ok \wedge \neg tm$	*false*	*false*	*true*	*false*
$ok \wedge tm$	*false*	$\neg P$	$\neg P \vee \neg Q$	$\neg P \vee S$

As remarked in Section 2.3.1, the state $\neg ok \wedge \neg tm$ can be excluded from consideration, which is manifest in the *false* entries of the first row and column. The remaining 3×3 sub-matrix is the basis of our treatment of extended designs in Section 3. It corresponds to the following three-state automaton:



The top state represents abortion, the bottom left state non-termination and the bottom right state normal termination. Observe that non-termination is treated as for prescriptions: once this state is reached it cannot be escaped. On the other hand, abortion is treated as for designs: it signifies the absence of information about the execution, which may either abort or not terminate or terminate normally. The reason for this difference is the requirement of [19] that abortion cannot be demanded. Of course, this requirement can be debated, and the matrix representation suggests ways to give it up which will be explored in future work. In the present paper we maintain this restriction.

2.3.4. Extended designs as designs

The 3×3 sub-matrix representing our new extended designs has the structure of a design. This can be observed by sub-dividing between its first and second rows and columns:

	$\neg ok' \wedge tm'$	$ok' \wedge \neg tm'$	$ok' \wedge tm'$
$\neg ok \wedge tm$	*true*	*true*	*true*
$ok \wedge \neg tm$	*false*	*true*	*false*
$ok \wedge tm$	$\neg P$	$\neg P \vee \neg Q$	$\neg P \vee S$

We thus obtain a 2×2 matrix with entries that are themselves $\{1,2\} \times \{1,2\}$ matrices. The structural entries are *true* and the bottom left vector is a condition which implies the bottom right matrix if both

are taken as predicates. This is just the pattern of a design. Moreover, the bottom right 2×2 sub-matrix obviously has the structure of a prescription.

To turn this into a formal result, we reconsider our new extended designs as 4×4 matrices, but this time we replace the *false* entries of the first row and column by copies of the second row and column, respectively. The reason for this change is that the resulting matrix is a design as shown below. We obtain

	$\neg ok' \wedge \neg tm'$	$\neg ok' \wedge tm'$	$ok' \wedge \neg tm'$	$ok' \wedge tm'$
$\neg ok \wedge \neg tm$	*true*	*true*	*true*	*true*
$\neg ok \wedge tm$	*true*	*true*	*true*	*true*
$ok \wedge \neg tm$	*false*	*false*	*true*	*false*
$ok \wedge tm$	$\neg P$	$\neg P$	$\neg P \vee \neg Q$	$\neg P \vee S$

Because of the copying, the states $\neg ok \wedge \neg tm$ and $\neg ok \wedge tm$ of the corresponding automaton are equivalent. Hence the variant of extended designs based on this matrix, which is obtained by

$$P/Q/S \;\Leftrightarrow_{\mathrm{def}}\; (ok \wedge tm \wedge P \Rightarrow ok' \wedge (Q \Rightarrow tm') \wedge (tm' \Rightarrow S)) \wedge (ok \wedge \neg tm \Rightarrow ok' \wedge \neg tm') \;,$$

is isomorphic to extended designs as regards non-deterministic choice and sequential composition. But this variant is actually a design whose effect is a prescription:

$$\begin{array}{lcl} P/Q/S & \Leftrightarrow & (ok \wedge tm \wedge P \Rightarrow ok' \wedge (Q \Rightarrow tm') \wedge (tm' \Rightarrow S)) \wedge (ok \wedge \neg tm \Rightarrow ok' \wedge \neg tm') \\ & \Leftrightarrow & (ok \wedge tm \wedge P \Rightarrow ok' \wedge (tm \wedge Q \Rightarrow tm') \wedge (tm' \Rightarrow tm \wedge S)) \wedge \\ & & (ok \wedge \neg tm \Rightarrow ok' \wedge (tm \wedge Q \Rightarrow tm') \wedge (tm' \Rightarrow tm \wedge S)) \\ & \Leftrightarrow & (ok \wedge tm \wedge P) \vee (ok \wedge \neg tm) \Rightarrow ok' \wedge (Q \Vdash S) \\ & \Leftrightarrow & ok \wedge ((tm \wedge P) \vee \neg tm) \Rightarrow ok' \wedge (Q \Vdash S) \\ & \Leftrightarrow & \neg tm \vee P \vdash (Q \Vdash S) \;. \end{array}$$

The equivalences are readily confirmed by case distinction on tm and tm'. Observe that the assumption of the resulting design $\neg tm \vee P \vdash (Q \Vdash S)$ refers to the auxiliary variable tm; its effect refers to both tm and tm'.

This representation is not useful when it comes to recursion because for designs recursive equations are solved by the weakest fixpoint [22]. This is typical for total correctness approaches and gives *true* for the equation $X = X$. But $true = (false \vdash_{\mathrm{X}} false) = (false|false|false)$ is the extended design which has aborting executions from every state, not the one which loops forever.

2.3.5. Extended designs as prescriptions

To obtain a representation which works for recursion, we show that our extended designs also have the structure of prescriptions. To this end, we rearrange the rows and the columns of the matrix representation by exchanging ok with tm and ok' with tm'. In the 4×4 matrix of Section 2.3.3 this amounts to swapping the two middle rows and the two middle columns. The structural entries in the first row and column are not affected by this and remain *false*. By omitting them we are left with the 3×3 matrix of Section 2.3.4 with the first two rows swapped and the first two columns swapped:

	$ok' \wedge \neg tm'$	$\neg ok' \wedge tm'$	$ok' \wedge tm'$
$ok \wedge \neg tm$	*true*	*false*	*false*
$\neg ok \wedge tm$	*true*	*true*	*true*
$ok \wedge tm$	$\neg P \vee \neg Q$	$\neg P$	$\neg P \vee S$

The sub-division brings out a 2×2 matrix with the structure of a prescription: the structural entries in the top row are *true* and *false* (as a row vector), and the bottom left vector is a condition. Moreover, the bottom right 2×2 sub-matrix obviously has the structure of a design.

Analogously to Section 2.3.4 we rebuild the 4×4 matrix, but replace the *false* entries of the original first row and column by copies of the original third row and column, respectively. Again the reason for this

change is that the resulting matrix is a prescription as shown below. We obtain

	$\neg ok' \wedge \neg tm'$	$ok' \wedge \neg tm'$	$\neg ok' \wedge tm'$	$ok' \wedge tm'$
$\neg ok \wedge \neg tm$	$true$	$true$	$false$	$false$
$ok \wedge \neg tm$	$true$	$true$	$false$	$false$
$\neg ok \wedge tm$	$true$	$true$	$true$	$true$
$ok \wedge tm$	$\neg P \vee \neg Q$	$\neg P \vee \neg Q$	$\neg P$	$\neg P \vee S$

Because of the copying, the states $\neg ok \wedge \neg tm$ and $ok \wedge \neg tm$ of the corresponding automaton are equivalent. Hence the variant of extended designs based on this matrix, which is obtained by

$$P \backslash Q \backslash S \;\Leftrightarrow_{\text{def}}\; (ok \wedge tm \wedge P \Rightarrow (Q \Rightarrow tm') \wedge (tm' \Rightarrow ok' \wedge S)) \wedge (tm' \Rightarrow tm) \;,$$

is isomorphic to extended designs as regards non-deterministic choice and sequential composition. But this variant is actually a prescription whose effect is a design:

$$\begin{array}{rcl}
P \backslash Q \backslash S & \Leftrightarrow & (ok \wedge tm \wedge P \Rightarrow (Q \Rightarrow tm') \wedge (tm' \Rightarrow ok' \wedge S)) \wedge (tm' \Rightarrow tm) \\
& \Leftrightarrow & (ok \wedge tm \wedge P \Rightarrow (Q \Rightarrow tm')) \wedge (ok \wedge tm \wedge P \Rightarrow (tm' \Rightarrow ok' \wedge S)) \wedge (tm' \Rightarrow tm) \\
& \Leftrightarrow & (ok \wedge tm \wedge P \wedge Q \Rightarrow tm') \wedge (ok \wedge tm \wedge P \wedge tm' \Rightarrow ok' \wedge S) \wedge (tm' \Rightarrow tm) \\
& \Leftrightarrow & (tm \wedge ok \wedge P \wedge Q \Rightarrow tm') \wedge (tm' \Rightarrow (ok \wedge tm \wedge P \Rightarrow ok' \wedge S)) \wedge (tm' \Rightarrow tm) \\
& \Leftrightarrow & (tm \wedge ok \wedge P \wedge Q \Rightarrow tm') \wedge (tm' \Rightarrow tm \wedge (ok \wedge P \Rightarrow ok' \wedge S)) \\
& \Leftrightarrow & (tm \wedge ok \wedge P \wedge Q \Rightarrow tm') \wedge (tm' \Rightarrow tm \wedge (P \vdash S)) \\
& \Leftrightarrow & ok \wedge P \wedge Q \Vdash (P \vdash S) \;.
\end{array}$$

Observe that the assumption of the resulting prescription refers to the auxiliary variable ok; its effect refers to both ok and ok'. The condition P characterising the non-aborting states appears twice.

We use this representation to obtain the approximation order on extended designs. To this end, we recall the Egli-Milner order $\sqsubseteq$ for prescriptions [33, 32, 11]:

$$(P_1 \Vdash R_1) \sqsubseteq (P_2 \Vdash R_2) \;\Leftrightarrow\; \forall v, v' : (R_1 \Rightarrow R_2) \wedge (P_1 \Rightarrow P_2) \wedge (P_1 \wedge R_2 \Rightarrow R_1) \;.$$

We instantiate it with the prescription obtained above for an extended design, treating ok and ok' as ordinary variables:

$$\begin{array}{rcl}
 & & (P_1 \backslash Q_1 \backslash S_1) \sqsubseteq (P_2 \backslash Q_2 \backslash S_2) \\
& \Leftrightarrow & (ok \wedge P_1 \wedge Q_1 \Vdash (P_1 \vdash S_1)) \sqsubseteq (ok \wedge P_2 \wedge Q_2 \Vdash (P_2 \vdash S_2)) \\
& \Leftrightarrow & \forall v, v', ok, ok' : ((P_1 \vdash S_1) \Rightarrow (P_2 \vdash S_2)) \wedge (ok \wedge P_1 \wedge Q_1 \Rightarrow ok \wedge P_2 \wedge Q_2) \wedge \\
& & \qquad (ok \wedge P_1 \wedge Q_1 \wedge (P_2 \vdash S_2) \Rightarrow (P_1 \vdash S_1)) \\
& \Leftrightarrow & \forall v, v' : (P_2 \Rightarrow P_1) \wedge (P_2 \wedge S_1 \Rightarrow S_2) \wedge (\forall ok, ok' : ok \wedge P_1 \wedge Q_1 \Rightarrow ok \wedge P_2 \wedge Q_2) \wedge \\
& & \qquad (\forall ok, ok' : ok \wedge P_1 \wedge Q_1 \wedge (ok \wedge P_2 \Rightarrow ok' \wedge S_2) \Rightarrow (ok \wedge P_1 \Rightarrow ok' \wedge S_1)) \\
& \Leftrightarrow & \forall v, v' : (P_2 \Rightarrow P_1) \wedge (P_2 \wedge S_1 \Rightarrow S_2) \wedge (\forall ok, ok' : ok \wedge P_1 \wedge Q_1 \Rightarrow ok \wedge P_2 \wedge Q_2) \wedge \\
& & \qquad (\forall ok, ok' : ok \wedge P_1 \wedge Q_1 \wedge ok' \wedge S_2 \Rightarrow ok' \wedge S_1) \\
& \Leftrightarrow & \forall v, v' : (P_2 \Rightarrow P_1) \wedge (P_2 \wedge S_1 \Rightarrow S_2) \wedge (P_1 \wedge Q_1 \Rightarrow P_2 \wedge Q_2) \wedge (P_1 \wedge Q_1 \wedge S_2 \Rightarrow S_1) \;.
\end{array}$$

The final step reduces the universal quantification over ok and ok' to the case $ok = ok' = true$. In the third step we use the refinement relation on designs [22, Theorem 3.1.2]:

$$(\forall v, v', ok, ok' : (P_1 \vdash S_1) \Rightarrow (P_2 \vdash S_2)) \;\Leftrightarrow\; (\forall v, v' : (P_2 \Rightarrow P_1) \wedge (P_2 \wedge S_1 \Rightarrow S_2)) \;.$$

As a preparation for the next section we express the order in relational terms and give another, equivalent version. The relational $\cup$, $\cap$, $\overline{\ }$ and $\subseteq$ abstract from the predicate operations $\vee$, $\wedge$, $\neg$ and $\Rightarrow$. Hence

$$\begin{array}{rcl}
 & & (P_1 \backslash Q_1 \backslash S_1) \sqsubseteq (P_2 \backslash Q_2 \backslash S_2) \\
& \Leftrightarrow & (\overline{P_2} \subseteq \overline{P_1}) \wedge (\overline{P_2} \cap S_1 \subseteq S_2) \wedge (P_1 \cap Q_1 \subseteq P_2 \cap Q_2) \wedge (P_1 \cap Q_1 \cap S_2 \subseteq S_1) \\
& \Leftrightarrow & (\overline{P_1} \subseteq \overline{P_2}) \wedge (\overline{P_2} \cup \overline{Q_2} \subseteq \overline{P_1} \cup \overline{Q_1}) \wedge (\overline{P_1} \cup S_1 \subseteq \overline{P_2} \cup S_2 \subseteq \overline{P_1} \cup \overline{Q_1} \cup S_1) \;.
\end{array}$$

Although the obtained order is derived from the Egli-Milner order on prescriptions, it is not clear that it is actually adequate for extended designs. We revisit this issue in Section 3.4.

3. The matrix algebra of extended designs

In this section, we calculate the effect of non-deterministic choice, sequential composition, finite and infinite iteration on extended designs. The development is based on the matrix representation shown in Section 2.3.3. Following [29, 18], we abstract from the predicates used as matrix entries to elements of semirings. A suitable definition of an extended design is therefore

$$(p|q|r) \;=_{\mathrm{def}}\; \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ \overline{p} & \overline{p}+\overline{q} & \overline{p}+r \end{pmatrix}$$

for particular semiring elements p, q and r that characterise the non-aborting states, the terminating states and the state transitions. The structural entries $\top$ and 0 replace the predicates *true* and *false*.

This definition needs a complement operation at least for the components p and q, whence [29] uses Boolean semirings. But observe that the complement is only required due to the convention of extended designs by which p represents the non-aborting states and q represents the terminating states. Turning this convention around, by representing the aborting and the non-terminating states, not only simplifies but also generalises our development. We revisit the above definition in Appendix A.

3.1. Choice and sequential composition

A *bounded idempotent semiring* is an algebraic structure $(S, +, 0, \top, \cdot, 1)$ with the following axioms:

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

Here and for other structures, we add the qualifier '*without right zero*' if the axiom $x\cdot 0=0$ is omitted. The *natural order* is given by $x\le y \Leftrightarrow_{\mathrm{def}} x+y=y$. It follows that the operations $+$ and $\cdot$ are $\le$-isotone. An element $x\in S$ is a *vector* if $x\cdot\top=x$. The join operation $+$ has lower precedence than $\cdot$ which is frequently omitted by writing xy instead of $x\cdot y$.

In relational models of partial correctness, $+$ is set union (non-deterministic choice), $\cdot$ is relational (sequential) composition, $\le$ is subset (refinement), and 0, $\top$ and 1 are the empty, universal and identity relations, respectively. A vector relates a state either to all states or to no state and thus represents a set of states.

Let S be a bounded idempotent semiring and $p,q,r\in S$ such that p and q are vectors, $p\le q$ and $p\le r$. Then the *extended design* $\langle p|q|r\rangle$ is given by

$$\langle p|q|r\rangle \;=_{\mathrm{def}}\; \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p & q & r \end{pmatrix}.$$

The set of extended designs over S is

$$\mathrm{ED}(S) \;=_{\mathrm{def}}\; \{\langle p|q|r\rangle \mid p,q,r\in S \wedge p\top=p\le q=q\top\wedge p\le r\}\,.$$

As mentioned above, the vector p represents the set of states from which an aborting execution exists. The vector q represents those states from which a non-terminating execution exists. The element r represents the executions which terminate normally. The restrictions $p\le q$ and $p\le r$ express that in the aborting case the execution might also not terminate or terminate normally.

Operations on extended designs can now be reduced to matrix operations. The non-deterministic choice of two extended designs is obtained componentwise because

$$\begin{aligned} \langle p_1|q_1|r_1\rangle+\langle p_2|q_2|r_2\rangle &= \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p_1 & q_1 & r_1 \end{pmatrix}+\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p_2 & q_2 & r_2 \end{pmatrix}=\begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p_1+p_2 & q_1+q_2 & r_1+r_2 \end{pmatrix} \\ &= \langle p_1+p_2 \mid q_1+q_2 \mid r_1+r_2\rangle\,, \end{aligned}$$

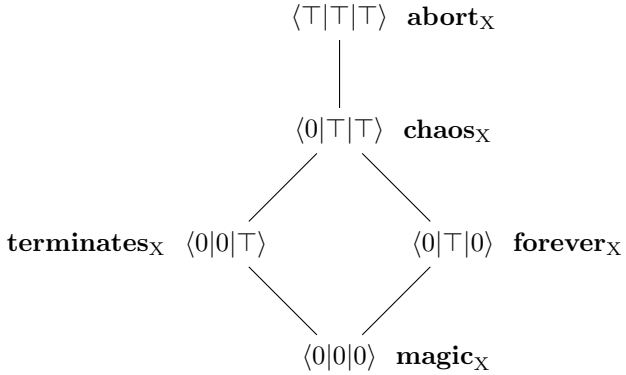
with the result being an extended design since $(p_1 + p_2)\top = p_1\top + p_2\top = p_1 + p_2$ and $p_1 + p_2 \leq r_1 + r_2$ and similarly for $q_1 + q_2$. This generalises to arbitrary choice by $\sum_{i\in I}\langle p_i|q_i|r_i\rangle = \langle \sum_{i\in I} p_i \mid \sum_{i\in I} q_i \mid \sum_{i\in I} r_i\rangle$ provided the occurring joins exist and $\cdot$ distributes over them. The equality of two extended designs is componentwise equality because

$$\langle p_1|q_1|r_1\rangle = \langle p_2|q_2|r_2\rangle \;\Leftrightarrow\; \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p_1 & q_1 & r_1 \end{pmatrix} = \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p_2 & q_2 & r_2 \end{pmatrix} \;\Leftrightarrow\; (p_1 = p_2) \wedge (q_1 = q_2) \wedge (r_1 = r_2)\ .$$

The natural order of extended designs is the componentwise order because

$$\langle p_1|q_1|r_1\rangle \leq \langle p_2|q_2|r_2\rangle \;\Leftrightarrow\; \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p_1 & q_1 & r_1 \end{pmatrix} \leq \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p_2 & q_2 & r_2 \end{pmatrix} \;\Leftrightarrow\; (p_1 \leq p_2) \wedge (q_1 \leq q_2) \wedge (r_1 \leq r_2)\ .$$

Hence the least extended design is $\langle 0|0|0\rangle$ and the greatest extended design is $\langle\top|\top|\top\rangle$. The following Hasse diagram shows them with the other extreme cases named by [19]:



For example, the extended design $\langle 0|\top|0\rangle$ represents the program which loops forever, and $\langle 0|0|\top\rangle$ specifies that all executions must terminate normally. The sequential composition of two extended designs is

$$\begin{aligned}
\langle p_1|q_1|r_1\rangle \cdot \langle p_2|q_2|r_2\rangle &= \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p_1 & q_1 & r_1 \end{pmatrix} \cdot \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p_2 & q_2 & r_2 \end{pmatrix} \\
&= \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p_1\top + q_1 0 + r_1 p_2 & p_1\top + q_1\top + r_1 q_2 & p_1\top + q_1 0 + r_1 r_2 \end{pmatrix} \\
&= \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p_1 + r_1 p_2 & q_1 + r_1 q_2 & p_1 + r_1 r_2 \end{pmatrix} = \langle p_1 + r_1 p_2 \mid q_1 + r_1 q_2 \mid p_1 + r_1 r_2\rangle\ ,
\end{aligned}$$

with the result being an extended design since $(p_1 + r_1 p_2)\top = p_1\top + r_1 p_2\top = p_1 + r_1 p_2$ and $p_1 + r_1 p_2 \leq p_1 + r_1 r_2$ and similarly for $q_1 + r_1 q_2$. It follows that the neutral extended design with respect to sequential composition is $\langle 0|0|1\rangle$. Furthermore,

$$\langle p_1|q_1|p_1\rangle \cdot \langle p_2|q_2|r_2\rangle = \langle p_1 + p_1 p_2 \mid q_1 + p_1 q_2 \mid p_1 + p_1 r_2\rangle = \langle p_1|q_1|p_1\rangle$$

since $p_1 p_2 \leq p_1\top = p_1$ and $p_1 q_2 \leq q_1 q_2 \leq q_1\top = q_1$ and $p_1 r_2 \leq p_1\top = p_1$. Hence any extended design of the form $\langle p|q|p\rangle$ is a left annihilator; this includes $\mathbf{magic}_{\mathrm{X}}$, $\mathbf{forever}_{\mathrm{X}}$ and $\mathbf{abort}_{\mathrm{X}}$. However, there is no right annihilator.

Properties such as associativity, commutativity and distributivity can be lifted from the underlying semiring to matrices. We therefore obtain the following algebraic structure for extended designs.

Theorem 1. *Let S be a bounded idempotent semiring. Then* $(\mathrm{ED}(S), +, \langle 0|0|0\rangle, \langle \top|\top|\top\rangle, \cdot, \langle 0|0|1\rangle)$ *is a bounded idempotent semiring without right zero.*

3.2. Domain

The domain $d(x)$ of a program x describes the set of states from which transitions under x are possible [6]. Axioms for the domain operation are conveniently expressed via the antidomain a, which describes the complementary set. An *antidomain semiring* (S, a) is an idempotent semiring S with an operation $a : S \to S$ that satisfies the following axioms [7]:

$$a(x) \cdot x = 0 \qquad a^2(x) + a(x) = 1 \qquad a(x \cdot a^2(y)) = a(x \cdot y)$$

These axioms also work for idempotent semirings without right zero. We obtain the *domain* operation by $d(x) =_{\mathrm{def}} a^2(x)$. It follows from the above axioms that the operation d is idempotent, whence the domain elements $d(S)$ are the fixpoints of d. They form a Boolean algebra $(d(S), +, 0, \cdot, 1, a)$ with join $+$, meet $\cdot$ and complement a.

The antidomain yields the complements necessary to terminate while-loops and to describe the conditional $\mathsf{if}\ p\ \mathsf{then}\ x\ \mathsf{else}\ y =_{\mathrm{def}} px + a(p)y$, where p is a domain element. Otherwise a domain semiring (S, d) axiomatised as in [7] suffices for the development carried out in this paper. In that case the domain elements form only a bounded distributive lattice $(d(S), +, 0, \cdot, 1)$.

The operation d is $\leq$-isotone and the operation a is $\leq$-antitone. Further consequences about the domain operation are

$$\begin{array}{cccc} d(x)x = x & d(xd(y)) = d(xy) & d(\top) = 1 & x \leq d(x)\top \\ a(d(x)) = a(x) & d(d(x)y) = d(x)d(y) & d(x+y) = d(x) + d(y) & xd(y)\top \leq d(xy)\top \end{array}$$

The laws using $\top$ hold in a bounded antidomain semiring.

The following result shows that the domain operation for extended designs is obtained by $d(\langle p|q|r\rangle) = \langle 0|0|d(q+r)\rangle$. Note that d and a are overloaded for S and $\mathrm{ED}(S)$.

Theorem 2. *Let S be a bounded antidomain semiring. Define* $a(\langle p|q|r\rangle) =_{\mathrm{def}} \langle 0|0|a(q+r)\rangle$ *for* $\langle p|q|r\rangle \in \mathrm{ED}(S)$. *Then* $(\mathrm{ED}(S), +, \langle 0|0|0\rangle, \cdot, \langle 0|0|1\rangle, a)$ *is an antidomain semiring without right zero.*

Proof. By Theorem 1 it remains to show the antidomain axioms. The domain operation $d(x) = a^2(x)$ is

$$d(\langle p|q|r\rangle) = a^2(\langle p|q|r\rangle) = a(\langle 0|0|a(q+r)\rangle) = \langle 0|0|a(0 + a(q+r))\rangle = \langle 0|0|a^2(q+r)\rangle = \langle 0|0|d(q+r)\rangle\ .$$

Observe that a on S satisfies $a(x+y)x \leq a(x)x = 0$ by being $\leq$-antitone. Hence a on $\mathrm{ED}(S)$ satisfies the first antidomain axiom $a(x)x = 0$ by

$$a(\langle p|q|r\rangle)\langle p|q|r\rangle = \langle 0|0|a(q+r)\rangle\langle p|q|r\rangle = \langle 0 + a(q+r)p \mid 0 + a(q+r)q \mid 0 + a(q+r)r\rangle = \langle 0|0|0\rangle$$

since $\langle 0|0|0\rangle$ is the additive identity of $\mathrm{ED}(S)$. The second axiom $d(x) + a(x) = 1$ follows by

$$d(\langle p|q|r\rangle) + a(\langle p|q|r\rangle) = \langle 0|0|d(q+r)\rangle + \langle 0|0|a(q+r)\rangle = \langle 0 + 0 \mid 0 + 0 \mid d(q+r) + a(q+r)\rangle = \langle 0|0|1\rangle$$

since $\langle 0|0|1\rangle$ is the multiplicative identity of $\mathrm{ED}(S)$. The third axiom $a(xd(y)) = a(xy)$ follows by

$$\begin{aligned} a(\langle p_1|q_1|r_1\rangle d(\langle p_2|q_2|r_2\rangle)) &= a(\langle p_1|q_1|r_1\rangle\langle 0|0|d(q_2+r_2)\rangle) = a(\langle p_1 + r_1 0 \mid q_1 + r_1 0 \mid p_1 + r_1 d(q_2+r_2)\rangle) \\ &= a(\langle p_1 \mid q_1 \mid p_1 + r_1 d(q_2+r_2)\rangle) = \langle 0|0|a(q_1 + p_1 + r_1 d(q_2+r_2))\rangle \\ &= \langle 0|0|a(q_1 + p_1 + r_1(q_2+r_2))\rangle = \langle 0|0|a(q_1 + r_1 q_2 + p_1 + r_1 r_2)\rangle \\ &= a(\langle p_1 + r_1 p_2 \mid q_1 + r_1 q_2 \mid p_1 + r_1 r_2\rangle) = a(\langle p_1|q_1|r_1\rangle\langle p_2|q_2|r_2\rangle) \end{aligned}$$

using the right zero axiom of S and the property $a(x + yd(z)) = a(x + yz)$ of S, which follows by applying a to $d(x + yd(z)) = d(x) + d(yd(z)) = d(x) + d(yz) = d(x + yz)$. □

3.3. Iteration

Following [18] we derive operations for finite and infinite iteration, that is, the Kleene star and the omega operation [24, 5]. We use the axioms of [30] which work for idempotent semirings with or without right zero:

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

An idempotent semiring that satisfies all of the above axioms not mentioning $^\omega$ is a *Kleene algebra*. If it satisfies also the axioms mentioning $^\omega$ it is an *omega algebra*.

As a consequence of the above axioms, the operations * and $^\omega$ are $\leq$-isotone and we have the following unfold, sliding and decomposition laws:

$$\begin{array}{lll} 1 + y^*y = y^* & x(yx)^* = (xy)^*x & (x+y)^* = x^*(yx^*)^* = (x^*y)^*x^* \\ 1 + yy^* = y^* & x(yx)^\omega = (xy)^\omega & (x+y)^\omega = (x^*y)^\omega + (x^*y)^*x^\omega \end{array}$$

The least and the greatest fixpoints of the function $\lambda x.yx + z$ are y^*z and $y^\omega + y^*z$, respectively. Every omega algebra is bounded by $\top = 1^\omega$. Moreover, $y^*0 \leq y^\omega 0$ and y^ω is a vector.

To obtain the Kleene star of an extended design, we apply the general construction of the star operation of a matrix. There are several versions in the literature; we use the one given in [12]:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^* =_{\text{def}} \begin{pmatrix} f^* & f^*bd^* \\ e^*ca^* & e^* \end{pmatrix},$$

where $f = a + bd^*c$ and $e = d + ca^*b$. To treat the 3×3 matrix of an extended design, we sub-divide it as in Section 2.3.4 into a 2×2 matrix with entries that are themselves $\{1,2\} \times \{1,2\}$ matrices. The above formula is then applied to the outer matrix as well as to the inner square matrices.

Let $p, q, r \in S$ such that $p\top = p \leq q = q\top$ and $p \leq r$, then

$$\begin{pmatrix} \top & 0 \\ q & r \end{pmatrix}^* = \begin{pmatrix} (\top + 0r^*q)^* & (\top + 0r^*q)^*0r^* \\ (r + q\top^*0)^*q\top^* & (r + q\top^*0)^* \end{pmatrix} = \begin{pmatrix} \top^* & 0 \\ r^*q\top^* & r^* \end{pmatrix} = \begin{pmatrix} \top & 0 \\ r^*q & r^* \end{pmatrix}$$

and

$$\begin{pmatrix} 0 \\ p \end{pmatrix} \top^* \begin{pmatrix} \top & \top \end{pmatrix} = \begin{pmatrix} 0 \\ p \end{pmatrix} \begin{pmatrix} \top & \top \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ p\top & p\top \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ p & p \end{pmatrix} \leq \begin{pmatrix} \top & 0 \\ q & r \end{pmatrix}.$$

Using $\cong$ to denote the sub-division steps, we therefore obtain

$$\begin{aligned} \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ p & q & r \end{pmatrix}^* &\cong \begin{pmatrix} \top & \begin{pmatrix} \top & \top \end{pmatrix} \\ \begin{pmatrix} 0 \\ p \end{pmatrix} & \begin{pmatrix} \top & 0 \\ q & r \end{pmatrix} \end{pmatrix}^* \\ &= \begin{pmatrix} \top & \top(\top \quad \top)\begin{pmatrix} \top & 0 \\ q & r \end{pmatrix}^* \\ \left(\begin{pmatrix} \top & 0 \\ q & r \end{pmatrix} + \begin{pmatrix} 0 \\ p \end{pmatrix}\top^*(\top \quad \top)\right)^*\begin{pmatrix} 0 \\ p \end{pmatrix}\top^* & \left(\begin{pmatrix} \top & 0 \\ q & r \end{pmatrix} + \begin{pmatrix} 0 \\ p \end{pmatrix}\top^*(\top \quad \top)\right)^* \end{pmatrix} \\ &= \begin{pmatrix} \top & (\top \quad \top)\begin{pmatrix} \top & 0 \\ r^*q & r^* \end{pmatrix} \\ \begin{pmatrix} \top & 0 \\ r^*q & r^* \end{pmatrix}\begin{pmatrix} 0 \\ p \end{pmatrix} & \begin{pmatrix} \top & 0 \\ r^*q & r^* \end{pmatrix} \end{pmatrix} \\ &= \begin{pmatrix} \top & \begin{pmatrix} \top & \top r^* \end{pmatrix} \\ \begin{pmatrix} 0 \\ r^*p \end{pmatrix} & \begin{pmatrix} \top & 0 \\ r^*q & r^* \end{pmatrix} \end{pmatrix} \cong \begin{pmatrix} \top & \top & \top \\ 0 & \top & 0 \\ r^*p & r^*q & r^* \end{pmatrix}. \end{aligned}$$

Observe that the assumptions $p\top = p \leq q = q\top$ and $p \leq r$ are satisfied for extended designs. Moreover, the resulting matrix is an extended design since $r^*p\top = r^*p \leq r^*q = r^*q\top$ and $r^*p \leq r^*r \leq r^*$. We thus obtain the Kleene star as follows.

Theorem 3. *Let S be a bounded Kleene algebra. Define $\langle p|q|r\rangle^* =_{\mathrm{def}} \langle r^*p \mid r^*q \mid r^*\rangle$ for $\langle p|q|r\rangle \in \mathrm{ED}(S)$. Then $(\mathrm{ED}(S), +, \langle 0|0|0\rangle, \cdot, \langle 0|0|1\rangle, ^*)$ is a Kleene algebra without right zero.*

A similar construction can be used to obtain the omega operation for a matrix [28]. When applied to an extended design, however, the resulting matrix does not have the structure of an extended design. The same problem is observed for prescriptions [13]. A systematic solution can be developed using typed omega algebras [17], according to which the right definition is

$$\langle p|q|r\rangle^\omega \;=_{\mathrm{def}}\; \langle r^\omega + r^*p \mid r^\omega + r^*q \mid r^\omega + r^*p\rangle$$

for $p, q, r \in S$ such that $p\top = p \leq q = q\top$ and $p \leq r$. We show that this operation satisfies the omega axioms. The unfold axiom follows by

$$\begin{aligned}\langle p|q|r\rangle\langle p|q|r\rangle^\omega &= \langle p|q|r\rangle\langle r^\omega + r^*p \mid r^\omega + r^*q \mid r^\omega + r^*p\rangle\\ &= \langle p + rr^\omega + rr^*p \mid q + rr^\omega + rr^*q \mid p + rr^\omega + rr^*p\rangle\\ &= \langle r^\omega + r^*p \mid r^\omega + r^*q \mid r^\omega + r^*p\rangle = \langle p|q|r\rangle^\omega\ .\end{aligned}$$

For the co-induction axiom assume

$$\langle x|y|z\rangle \leq \langle p|q|r\rangle\langle x|y|z\rangle + \langle u|v|w\rangle = \langle p + rx + u \mid q + ry + v \mid p + rz + w\rangle\ ,$$

which is equivalent to $x \leq p + rx + u$ and $y \leq q + ry + v$ and $z \leq p + rz + w$. Hence $x \leq r^\omega + r^*(p + u)$ and $y \leq r^\omega + r^*(q + v)$ and $z \leq r^\omega + r^*(p + w)$. Therefore,

$$\begin{aligned}\langle x|y|z\rangle &\leq \langle r^\omega + r^*(p + u) \mid r^\omega + r^*(q + v) \mid r^\omega + r^*(p + w)\rangle\\ &= \langle r^\omega + r^*p + r^*u \mid r^\omega + r^*q + r^*v \mid r^\omega + r^*p + r^*w\rangle\\ &= \langle r^\omega + r^*p \mid r^\omega + r^*q \mid r^\omega + r^*p\rangle + \langle r^*p \mid r^*q \mid r^*\rangle\langle u|v|w\rangle = \langle p|q|r\rangle^\omega + \langle p|q|r\rangle^*\langle u|v|w\rangle\ .\end{aligned}$$

We thus obtain the following result.

Theorem 4. *Let S be an omega algebra. Define $\langle p|q|r\rangle^\omega =_{\mathrm{def}} \langle r^\omega + r^*p \mid r^\omega + r^*q \mid r^\omega + r^*p\rangle$ for $\langle p|q|r\rangle \in \mathrm{ED}(S)$. Then $(\mathrm{ED}(S), +, \langle 0|0|0\rangle, \cdot, \langle 0|0|1\rangle, ^*, ^\omega)$ is an omega algebra without right zero.*

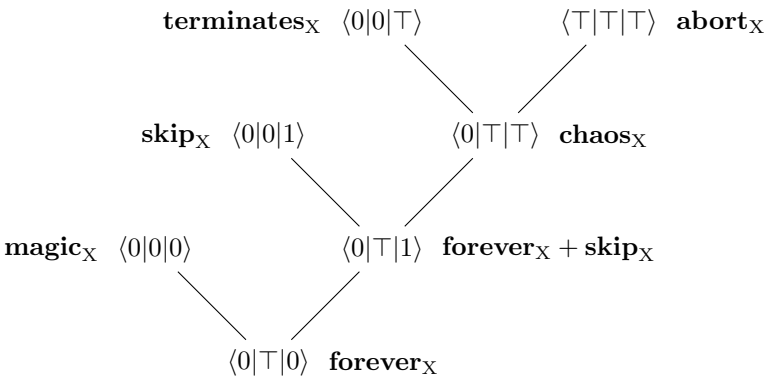
3.4. Approximation

In Section 2.3.5 we have prepared the approximation order for the switch to our convention as regards the components p and q of an extended design $\langle p|q|r\rangle$. By comparing the two matrices shown at the beginning of Section 3 we see that the components p, q and r of $\langle p|q|r\rangle$ correspond to $\overline{p}$, $\overline{p} + \overline{q}$ and $\overline{p} + r$ of the original $(p|q|r)$. Using this translation we obtain the *approximation order*

$$\langle p_1|q_1|r_1\rangle \sqsubseteq \langle p_2|q_2|r_2\rangle \;\Leftrightarrow_{\mathrm{def}}\; (p_1 \leq p_2) \wedge (q_2 \leq q_1) \wedge (r_1 \leq r_2 \leq q_1 + r_1)\ .$$

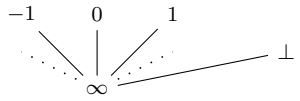
Since this order has no counterpart in the literature we argue for its adequacy.

Consider the extreme cases of extended designs along with the skip program $\langle 0|0|1\rangle$ as well as $\langle 0|\top|1\rangle$, a computation obtained by the non-deterministic choice between skip and the endless loop. The following Hasse diagram shows how these are related by $\sqsubseteq$:



The endless loop is the least element as expected since it should be the $\sqsubseteq$-least solution of the equation $X = X$. Ascending in the order, we can add executions (starting in a particular state) to an element as long as it contains the non-terminating execution (starting in the same state), and we can remove non-terminating executions. Once the component q of the extended design $\langle p|q|r\rangle$ is 0, whence all executions terminate, the element is maximal. The added executions may be normally terminating ones, in which case the component r increases. As soon as the aborting execution is added, we also reach a maximal element since it subsumes all other executions according to the requirement of [19] that abortion cannot be demanded.

To provide another view on the approximation order, we redraw this picture in terms of powerdomains, see [36]. The connection to our setting 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 $Int =_{\text{def}} \mathbb{Z} \cup \{\infty, \bot\}$ by adding two special elements ∞ and $\bot$. The element ∞ represents the non-terminating execution, and $\bot$ represents the aborting execution. The order $\preceq$ on Int is flat with ∞ as least element, that is, we have $x \preceq y \Leftrightarrow_{\text{def}} x = \infty \vee x = y$:



In particular, $\bot$ is treated as any other element except ∞. The Plotkin powerdomain of Int can be visualised as in Figure 1 on page 15 showing the sets without ∞ as maximal elements.

For extended designs, however, the aborting outcome absorbs all other outcomes in a similar way as the non-terminating outcome does in the Smyth powerdomain. Hence every set containing $\bot$ is identified with the set $\{\bot\}$. The resulting order is shown in Figure 2 on page 15, in which the set $\{\bot\}$ is above all sets containing ∞.

Let us finally discuss particular instances of $\langle p_1|q_1|r_1\rangle \sqsubseteq \langle p_2|q_2|r_2\rangle$:

- If $p_1 = \top$, and hence $q_1 = r_1 = \top$, the relation holds if and only if $p_2 = q_2 = r_2 = \top$. In this case $\langle p_1|q_1|r_1\rangle$ contains the aborting execution. Then $\langle p_2|q_2|r_2\rangle$ must also have the aborting execution. Non-terminating or normally terminating executions cannot be distinguished in this case.

- If $p_1 = 0$ and $q_1 = \top$, the relation holds if and only if $r_1 \leq r_2$. In this case $\langle p_1|q_1|r_1\rangle$ has no aborting execution, but the non-terminating one. Then $\langle p_2|q_2|r_2\rangle$ may add normally terminating executions ($r_1 \leq r_2$) while keeping ($q_2 = \top$) or removing ($p_2 = q_2 = 0$) the non-terminating execution. It may also add the aborting execution, which subsumes all normally terminating ones ($p_2 = q_2 = r_2 = \top$).

- If $p_1 = q_1 = 0$, the relation holds if and only if $p_2 = q_2 = 0$ and $r_1 = r_2$. In this case $\langle p_1|q_1|r_1\rangle$ has only normally terminating executions. Then $\langle p_2|q_2|r_2\rangle$ must equal $\langle p_1|q_1|r_1\rangle$.

More generally, this reasoning has to be applied to each starting state and the executions originating there.

4. The algebra of extended designs

We have seen that extended designs, represented as predicates or as matrices, have a rich structure. It features an antidomain semiring, a Kleene algebra and an omega algebra without right zero. In this section, we further investigate this algebraic structure by treating full recursion.

4.1. Partial, total and general correctness

We first recapitulate a few background ideas which lead to a unified semantics of partial, total and general correctness in our previous works [13, 15]. The subsequent algebraic treatment of extended designs generalises these investigations.

A brief characterisation of partial, total and general correctness is due. All of these approaches are able to represent non-determinism, that is, computations with several executions starting in the same state.

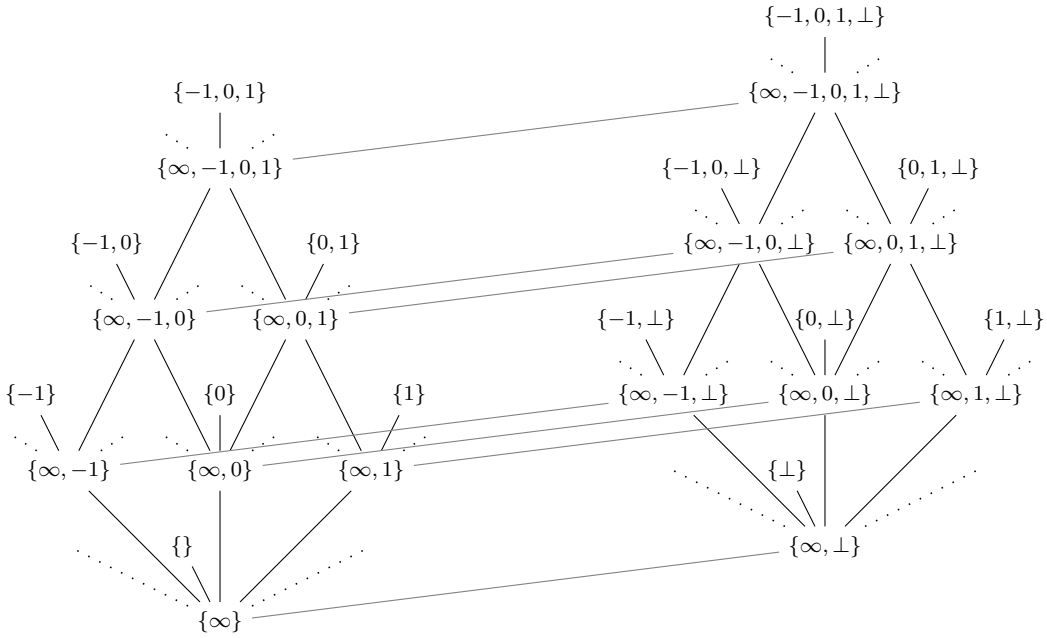


Figure 1: The Plotkin powerdomain of *Int*, based on [36, Figure 3]

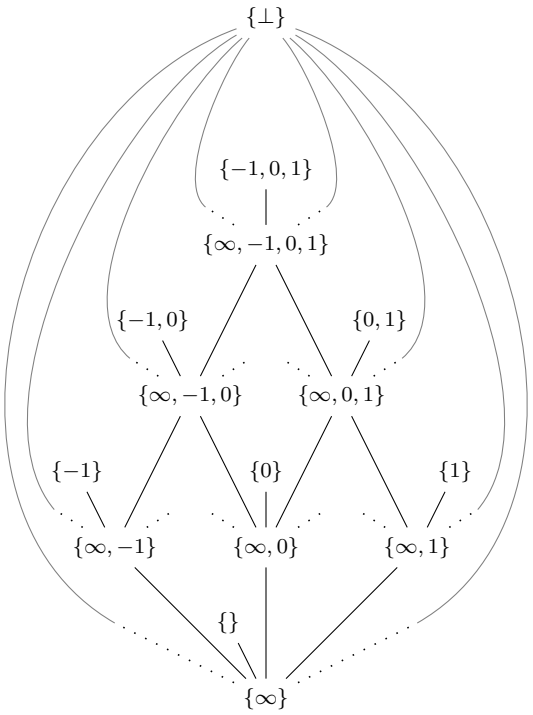


Figure 2: The powerdomain of *Int* as required for extended designs

However, non-deterministic choice is interpreted differently in each approach. Consider, for example, the computation $1 + \mathsf{L}$ which is the non-deterministic choice between the skip program 1 and the endless loop L.

In partial correctness [21, 8, 26, 6, 31] non-terminating executions are ignored in the presence of terminating ones, that is, $1 + \mathsf{L} = 1$ holds. Such a non-deterministic choice is angelic according to [36]; in fact $x + \mathsf{L} = x$ holds for every x, whence $\mathsf{L} \leq x$ or $\mathsf{L} = 0$. Recursion is thus modelled by least fixpoints with respect to the natural order $\leq$. Another observation for partial correctness is $\top 0 = 0$, whence $\mathsf{L} = \top 0$.

In total correctness [8, 22, 37, 4] terminating executions are ignored in the presence of non-terminating ones, that is, $1 + \mathsf{L} = \mathsf{L}$ holds. Such a non-deterministic choice is demonic; in fact $x + \mathsf{L} = \mathsf{L}$ holds for every x, whence $x \leq \mathsf{L}$ or $\mathsf{L} = \top$. Recursion is thus modelled by greatest fixpoints with respect to the natural order $\leq$. Another observation for total correctness is $\top 0 = \top$, whence again $\mathsf{L} = \top 0$.

In general correctness [1, 3, 34, 23, 2, 33, 9, 10, 32] terminating and non-terminating executions are independent, that is, $1 + \mathsf{L}$ is neither 1 nor L. The non-deterministic choice is erratic; L is neither 0 nor $\top$. Hence the Egli-Milner order is introduced and recursion is modelled by least fixpoints with respect to this order. Still one observes $\mathsf{L} = \top 0$ also in general correctness.

All three approaches have a common algebraic structure, namely a bounded antidomain semiring (and omega algebra) without right zero, which is also shared by extended designs. Partial correctness is obtained by adding the axiom $\top 0 = 0$. For total correctness the axiom $\top 0 = \top$ is added. For general correctness we have identified the axioms $d(x0)\top 0 \leq x$ and $d(\top 0) = 1$ in [15], and based on these axioms the Egli-Milner order $x \sqsubseteq y \Leftrightarrow x \leq y + x0 \wedge y \leq x + d(x0)\top$. It is then possible to express least fixpoints with respect to $\sqsubseteq$ in terms of least and greatest fixpoints with respect to $\leq$. This simplifies the solution of recursion equations and gives the semantics of iteration as a special case. In particular, the semantics of while-programs turns out to be the same in all three approaches which enables their unified treatment.

Our aim in the following is to generalise this development to extended designs. A change is required because neither partial, total nor general correctness describes aborting executions separately from non-terminating ones. For example, $\mathsf{L} \neq \top 0$ for extended designs since $\top 0$ contains aborting executions but L does not. We therefore use a modified set of axioms and a modified approximation order, and point out how they generalise our previous theory of general correctness.

4.2. Approximation

Throughout this section, let S be a bounded antidomain semiring without right zero. To represent the approximation order on S, we assume an element $\mathsf{L} \in S$ satisfying the following axioms:

$$\begin{array}{ll} (\mathsf{L1}) & d(x0)\mathsf{L} \leq x \\ (\mathsf{L2}) & d(\mathsf{L}0) = 1 \\ (\mathsf{L3}) & x\mathsf{L} \leq x0 + \mathsf{L} \end{array}$$

As is shown below, the element L abstractly represents the endless loop $\langle 0|\top|0\rangle$. The above axioms generalise those in our previous works [13, 15] about general correctness, where $\mathsf{L} = \top 0$ is assumed. Under this additional assumption (L3) vacuously holds and (L1) and (L2) reduce to the axioms used in those investigations. However, $\mathsf{L} = \top 0$ is not valid for extended designs (which actually satisfy $\top 0 = \top$, though this is no consequence of the above axioms). Intuitively, the operation $(\cdot 0)$ still cuts away all normally terminating executions of a program, but retains the aborting ones in addition to the non-terminating executions. The following lemma gives several consequences of the above axioms.

Lemma 5.

1. $\mathsf{L}x = \mathsf{L}$.
2. $d(x)\mathsf{L} \leq x\mathsf{L}$.
3. $x\mathsf{L} = x0 + d(x)\mathsf{L}$.
4. $xd(y)\mathsf{L} = x0 + d(xy)\mathsf{L}$.

Proof.

1. $\mathsf{L} = d(\mathsf{L}0)\mathsf{L} = d(\mathsf{L}0 \cdot 0)\mathsf{L} \leq \mathsf{L}0$ by ($\mathsf{L}2$) and ($\mathsf{L}1$). Hence $\mathsf{L}0 = \mathsf{L}$ and thus $\mathsf{L}x = \mathsf{L}0x = \mathsf{L}0 = \mathsf{L}$.
2. $d(x)\mathsf{L} = d(xd(\mathsf{L}0))\mathsf{L} = d(x\mathsf{L}0)\mathsf{L} \leq x\mathsf{L}$ by ($\mathsf{L}2$) and ($\mathsf{L}1$).
3. $x\mathsf{L} = d(x)x\mathsf{L} \leq d(x)(x0 + \mathsf{L}) = x0 + d(x)\mathsf{L}$ by ($\mathsf{L}3$) and $x0 + d(x)\mathsf{L} \leq x\mathsf{L}$ by part 2.
4. $xd(y)\mathsf{L} = xd(y)0 + d(xd(y))\mathsf{L} = x0 + d(xy)\mathsf{L}$ by part 3. □

Using L, we define the *approximation relation* on S by

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

Again this generalises [15] where $d(x0)\mathsf{L} = x0$ holds, which is not valid for extended designs. Intuitively, $d(x0)$ represents the states from which x has non-terminating executions. Hence the first part $x \leq y + d(x0)\mathsf{L}$ expresses that y has at least the aborting and terminating executions of x. The second part $y \leq x + d(x0)\top$ expresses that the infinite executions of y are at most those of x and, in combination with the first part, that y and x have the same aborting and terminating executions from states in $a(x0)$, that is, from which x is guaranteed to terminate. The specific axioms about L are used to establish basic properties of $\sqsubseteq$.

Theorem 6. *The relation $\sqsubseteq$ is a preorder. It is an order if and only if* ($\mathsf{L}1$) *holds. It has least element L if and only if* ($\mathsf{L}2$) *holds. The operation $+$ is $\sqsubseteq$-isotone. Assuming* ($\mathsf{L}1$) *and* ($\mathsf{L}2$)*, the operation $\cdot z$ is $\sqsubseteq$-isotone, and $z\cdot$ is $\sqsubseteq$-isotone if and only if* ($\mathsf{L}3$) *holds.*

Proof. Reflexivity of $\sqsubseteq$ is immediate. For transitivity assume $x \sqsubseteq y$ and $y \sqsubseteq z$, then

$$d(y0)\mathsf{L} \leq d((x + d(x0)\top)0)\mathsf{L} = d(x0)\mathsf{L} + d(x0)d(\top 0)\mathsf{L} = d(x0)\mathsf{L} \ ,$$

whence

$$x \leq y + d(x0)\mathsf{L} \leq z + d(y0)\mathsf{L} + d(x0)\mathsf{L} = z + d(x0)\mathsf{L}$$

and

$$z \leq y + d(y0)\top \leq x + d(x0)\top + d((x + d(x0)\top)0)\top = x + d(x0)\top + d(x0)d(\top 0)\top = x + d(x0)\top \ .$$

Therefore $x \sqsubseteq z$. For antisymmetry assume $x \sqsubseteq y$ and $y \sqsubseteq x$, then

$$x \leq y + d(x0)\mathsf{L} \leq y + d((y + d(y0)\top)0)\mathsf{L} = y + d(y0)\mathsf{L} + d(y0)d(\top 0)\mathsf{L} = y + d(y0)\mathsf{L} = y$$

by ($\mathsf{L}1$). Symmetrically $y \leq x$ is obtained, whence $x = y$. To show that antisymmetry of $\sqsubseteq$ implies ($\mathsf{L}1$), observe that $x + d(x0)\mathsf{L} \sqsubseteq x$ since

$$x + d(x0)\mathsf{L} \leq x + d(x0 + d(x0)\mathsf{L}0)\mathsf{L} = x + d((x + d(x0)\mathsf{L})0)\mathsf{L}$$

and $x \sqsubseteq x + d(x0)\mathsf{L}$ since $x + d(x0)\mathsf{L} \leq x + d(x0)\top$. Therefore $x + d(x0)\mathsf{L} = x$, which states ($\mathsf{L}1$).

The least element of $\sqsubseteq$ is L if and only if

$$\begin{array}{lll} (\forall y : \mathsf{L} \sqsubseteq y) & \Leftrightarrow & (\forall y : \mathsf{L} \leq y + d(\mathsf{L}0)\mathsf{L} \wedge y \leq \mathsf{L} + d(\mathsf{L}0)\top) \ \Leftrightarrow \ (\mathsf{L} \leq d(\mathsf{L}0)\mathsf{L} \wedge \top \leq \mathsf{L} + d(\mathsf{L}0)\top) \\ & \Leftrightarrow & (\mathsf{L} = d(\mathsf{L}0)\mathsf{L} \wedge \top = d(\mathsf{L}0)\top) \ \Leftrightarrow \ 1 = d(\mathsf{L}0) \ . \end{array}$$

The operation $+$ is $\sqsubseteq$-isotone since $x \sqsubseteq y$ implies

$$x + z \leq y + d(x0)\mathsf{L} + z \leq y + z + d((x + z)0)\mathsf{L}$$

and

$$y + z \leq x + d(x0)\top + z \leq x + z + d((x + z)0)\top \ ,$$

whence $x + z \sqsubseteq y + z$. The operation $\cdot$ is $\sqsubseteq$-isotone in its first argument since $x \sqsubseteq y$ implies

$$xz \leq (y + d(x0)\mathsf{L})z = yz + d(x0)\mathsf{L}z = yz + d(x0)\mathsf{L} \leq yz + d(xz0)\mathsf{L}$$

by Lemma 5.1 using (L1) and (L2), and

$$yz \leq (x + d(x0)\top)z = xz + d(x0)\top z \leq xz + d(xz0)\top \ ,$$

whence $xz \sqsubseteq yz$. The operation $\cdot$ is $\sqsubseteq$-isotone in its second argument since $x \sqsubseteq y$ implies

$$zx \leq z(y + d(x0)\mathsf{L}) = zy + zd(x0)\mathsf{L} = zy + z0 + d(zx0)\mathsf{L} = zy + d(zx0)\mathsf{L}$$

by Lemma 5.4 using (L3), and

$$zy \leq z(x + d(x0)\top) = zx + zd(x)0\top \leq zx + d(zx0)\top \ ,$$

whence $zx \sqsubseteq zy$. Furthermore, assuming (L2) the latter implies (L3) because

$$\mathsf{L} \sqsubseteq 0 \ \Rightarrow \ x\mathsf{L} \sqsubseteq x0 \ \Rightarrow \ x\mathsf{L} \leq x0 + d(x\mathsf{L}0)\mathsf{L} \leq x0 + \mathsf{L} \ .$$

□

As a consequence, L is uniquely determined by satisfying (L1) and (L2). We now show that extended designs are a model of the above theory. By Theorems 1 and 2 they form a bounded antidomain semiring without right zero. It therefore remains to prove (L1)–(L3). To this end, we need to assume $d(q)\top \leq q$ for every vector q in the underlying semiring, which is typical for relational models and asserts that q contains all executions starting in a state in the set represented by q.

Theorem 7. *Let S be a bounded antidomain semiring such that $d(q)\top \leq q$ for each vector $q \in S$. Then* (L1)–(L3) *hold in* $\mathrm{ED}(S)$ *with* $\mathsf{L} = \langle 0|\top|0\rangle$. *Moreover, $\sqsubseteq$ instantiates to the order of Section 3.4, that is,*

$$\langle p_1|q_1|r_1\rangle \sqsubseteq \langle p_2|q_2|r_2\rangle \ \Leftrightarrow \ (p_1 \leq p_2) \wedge (q_2 \leq q_1) \wedge (r_1 \leq r_2 \leq q_1 + r_1) \ .$$

Proof. Axiom (L1) holds since

$$\begin{aligned} d(\langle p|q|r\rangle\langle 0|0|0\rangle)\langle 0|\top|0\rangle &= d(\langle p + r0 \mid q + r0 \mid p + r0\rangle)\langle 0|\top|0\rangle = d(\langle p|q|p\rangle)\langle 0|\top|0\rangle = \langle 0|0|d(q + p)\rangle\langle 0|\top|0\rangle \\ &= \langle 0|0|d(q)\rangle\langle 0|\top|0\rangle = \langle 0 + d(q)0 \mid 0 + d(q)\top \mid 0 + d(q)0\rangle = \langle 0 \mid d(q)\top \mid 0\rangle \\ &\leq \langle p|q|r\rangle \ . \end{aligned}$$

Axiom (L2) holds since

$$d(\langle 0|\top|0\rangle\langle 0|0|0\rangle) = d(\langle 0|\top|0\rangle) = \langle 0|0|d(\top)\rangle = \langle 0|0|1\rangle \ .$$

Axiom (L3) holds since

$$\langle p|q|r\rangle\langle 0|\top|0\rangle = \langle p + r0 \mid q + r\top \mid p + r0\rangle \leq \langle p|\top|p\rangle = \langle p|q|p\rangle + \langle 0|\top|0\rangle = \langle p|q|r\rangle\langle 0|0|0\rangle + \langle 0|\top|0\rangle \ .$$

The approximation order $\langle p_1|q_1|r_1\rangle \sqsubseteq \langle p_2|q_2|r_2\rangle$ instantiates according to

$$\begin{aligned} \langle p_1|q_1|r_1\rangle &\leq \langle p_2|q_2|r_2\rangle + d(\langle p_1|q_1|r_1\rangle\langle 0|0|0\rangle)\langle 0|\top|0\rangle = \langle p_2|q_2|r_2\rangle + d(\langle p_1|q_1|p_1\rangle)\langle 0|\top|0\rangle \\ &= \langle p_2|q_2|r_2\rangle + \langle 0|0|q_1\rangle\langle 0|\top|0\rangle = \langle p_2|q_2|r_2\rangle + \langle 0|q_1|0\rangle = \langle p_2 \mid q_2 + q_1 \mid r_2\rangle \end{aligned}$$

and

$$\begin{aligned} \langle p_2|q_2|r_2\rangle &\leq \langle p_1|q_1|r_1\rangle + d(\langle p_1|q_1|r_1\rangle\langle 0|0|0\rangle)\langle\top|\top|\top\rangle = \langle p_1|q_1|r_1\rangle + \langle 0|0|q_1\rangle\langle\top|\top|\top\rangle \\ &= \langle p_1|q_1|r_1\rangle + \langle q_1|q_1|q_1\rangle = \langle p_1 + q_1 \mid q_1 + q_1 \mid r_1 + q_1\rangle = \langle q_1 \mid q_1 \mid q_1 + r_1\rangle \ . \end{aligned}$$

Together these two are equivalent to $(p_1 \leq p_2) \wedge (r_1 \leq r_2) \wedge (q_2 \leq q_1) \wedge (r_2 \leq q_1 + r_1)$. □

4.3. Recursion

As the solution of a recursion equation $x = f(x)$ with $f : S \to S$ we take the $\sqsubseteq$-least fixpoint ξf of f. Since the relation $\sqsubseteq$ is complex we show how to represent ξf in terms of the $\leq$-least and $\leq$-greatest fixpoints μf and νf. Formally, these fixpoints are elements of S satisfying the following properties:

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

Immediate consequences are $\mu f \leq \xi f \leq \nu f$ and $\xi f \sqsubseteq \mu f$ and $\xi f \sqsubseteq \nu f$.

The following result generalises and extends the representations shown in [13, 15] to the present setting. Its proof follows that of [16, Theorem 3]. The greatest lower bound of $x, y \in S$ with respect to $\sqsubseteq$ is denoted by $x \sqcap y$ provided it exists.

Theorem 8. *Let $f : S \to S$ be $\leq$- and $\sqsubseteq$-isotone and assume that μf and νf exist. Then the following are equivalent:*

1. *ξf exists.*
2. *$\xi f = \mu f \sqcap \nu f$.*
3. *$\xi f = \mu f + d(\nu f 0)\mathsf{L}$.*
4. *$\nu f \leq \mu f + d(\nu f 0)\top$.*
5. *$\mu f + d(\nu f 0)\mathsf{L} \sqsubseteq \nu f$.*
6. *$\mu f \sqcap \nu f = \mu f + d(\nu f 0)\mathsf{L}$.*
7. *$\mu f \sqcap \nu f \leq \nu f$.*

Proof. Abbreviate $\ell =_{\text{def}} \mu f + d(\nu f 0)\mathsf{L}$ and $m =_{\text{def}} \mu f \sqcap \nu f$. Observe that $\mu f \leq \ell \leq \nu f$ by ($\mathsf{L1}$) and

$$d(\ell 0) = d((\mu f + d(\nu f 0)\mathsf{L})0) = d(\mu f 0 + d(\nu f 0)\mathsf{L}0) = d(\mu f 0) + d(\nu f 0)d(\mathsf{L}0) = d(\mu f 0) + d(\nu f 0) = d(\nu f 0)$$

by ($\mathsf{L2}$). We first show that statements (4)–(7) are equivalent:

(4) $\Rightarrow$ (5): We get $\ell \sqsubseteq \nu f$ since $\ell \leq \nu f \leq \nu f + d(\ell 0)\mathsf{L}$ and $\nu f \leq \mu f + d(\nu f 0)\top \leq \ell + d(\ell 0)\top$.

(5) $\Rightarrow$ (6): For $\ell \sqsubseteq m$ it remains to show $\ell \sqsubseteq \mu f$, which follows from $\ell = \mu f + d(\ell 0)\mathsf{L}$ and $\mu f \leq \nu f \leq \ell + d(\ell 0)\top$. To obtain $\ell = m$, let $x \sqsubseteq \mu f$ and $x \sqsubseteq \nu f$. Then $x \leq \mu f + d(x0)\mathsf{L} \leq \ell + d(x0)\mathsf{L}$ and $\ell \leq \nu f \leq x + d(x0)\top$, whence $x \sqsubseteq \ell$.

(6) $\Rightarrow$ (7): This follows immediately because $\ell \leq \nu f$.

(7) $\Rightarrow$ (4): From $m \sqsubseteq \mu f$ we obtain $m \leq \mu f + d(m0)\mathsf{L}$. Therefore, $m \sqsubseteq \nu f$ implies $\nu f \leq m + d(m0)\top \leq \mu f + d(m0)\mathsf{L} + d(m0)\top = \mu f + d(m0)\top \leq \mu f + d(\nu f 0)\top$.

We next add statements (1)–(3) to this cycle:

(1) $\Rightarrow$ (2): Clearly $\xi f \sqsubseteq m$. To obtain $\xi f = m$, let $x \sqsubseteq \mu f$ and $x \sqsubseteq \nu f$. Then $x \leq \mu f + d(x0)\mathsf{L} \leq \xi f + d(x0)\mathsf{L}$ and $\xi f \leq \nu f \leq x + d(x0)\top$, whence $x \sqsubseteq \xi f$.

(2) $\Rightarrow$ (7): This follows immediately by $\xi f \leq \nu f$.

(7) $\Rightarrow$ (3): From $m \sqsubseteq \mu f$ we get $f(m) \sqsubseteq f(\mu f) = \mu f$ and $m \leq \mu f + d(m0)\mathsf{L} = f(\mu f) + d(m0)\mathsf{L} \leq f(m) + d(m0)\mathsf{L}$ since $\mu f \leq m$ by (6). From $m \sqsubseteq \nu f$ we get $f(m) \sqsubseteq f(\nu f) = \nu f$ and $f(m) \leq f(\nu f) = \nu f \leq m + d(m0)\top$ by (7). Hence $m \sqsubseteq f(m) \sqsubseteq m$, thus $f(m) = m$ by Theorem 6, and therefore $f(\ell) = \ell$ by (6).

To obtain $\xi f = \ell$, let $f(x) = x$. Then $\mu f \leq x \leq \nu f$, whence $\ell = \mu f + d(\ell 0)\mathsf{L} \leq x + d(\ell 0)\mathsf{L}$ and $x \leq \nu f \leq \ell + d(\ell 0)\top$ by (5). Thus $\ell \sqsubseteq x$.

(3) $\Rightarrow$ (1): This is clear. $\square$

4.4. Iteration

As in our previous works [13, 15, 16] we instantiate the full recursion result with while-loops. Note that the Boolean complement of a domain element p is given by its antidomain $a(p)$. Hence the semantics of the loop $\mathsf{while}\ p\ \mathsf{do}\ y$ is the $\sqsubseteq$-least fixpoint of the function $f(x) = pyx + a(p)$. To represent it compactly, we assume that S is additionally an omega algebra without right zero.

Corollary 9. *Let $y \in S$ and $p, q \in d(S)$ and $f(x) =_{\mathrm{def}} pyx + q$. Then $\xi f = d((py)^\omega)\mathsf{L} + (py)^* q$.*

Proof. We have $\mu f = (py)^* q$ and $\nu f = (py)^\omega + (py)^* q$. The function f is $\leq$-isotone and by Theorem 6 also $\sqsubseteq$-isotone. Thus ξf exists since Theorem 8.4 holds by

$$\begin{aligned} \nu f 0 &= ((py)^\omega + (py)^* q)0 = (py)^\omega 0 + (py)^* 0 = (py)^\omega 0 \ , \\ d(\nu f 0) &= d((py)^\omega 0) = d((py)^\omega \top 0) = d((py)^\omega d(\top 0)) = d((py)^\omega) \ , \\ \nu f &= (py)^\omega + \mu f \leq d((py)^\omega)\top + \mu f = d(\nu f 0)\top + \mu f \ , \end{aligned}$$

using (L2) in the second calculation. Hence Theorem 8.3 gives $\xi f = d(\nu f 0)\mathsf{L} + \mu f = d((py)^\omega)\mathsf{L} + (py)^* q$. □

We therefore have the explicit representation $\mathsf{while}\ p\ \mathsf{do}\ y = d((py)^\omega)\mathsf{L} + (py)^* a(p)$, which is also used for the unified semantics in Section 5. Introducing the operation $x^\circ =_{\mathrm{def}} d(x^\omega)\mathsf{L} + x^*$ to combine infinite and finite iteration we get $\mathsf{while}\ p\ \mathsf{do}\ y = (py)^\circ a(p)$ by Lemma 5.1. The operation $^\circ$ generalises our previous combined iteration $x^\omega 0 + x^*$ of [15]. It is clear that $^\circ$ is $\leq$-isotone, and the following result shows that it is $\sqsubseteq$-isotone, too.

Theorem 10. *Let $x, y \in S$ such that $x \sqsubseteq y$. Then $x^* \sqsubseteq y^*$ and $x^\omega \sqsubseteq y^\omega$ and $x^\circ \sqsubseteq y^\circ$.*

Proof. Let $x \sqsubseteq y$, whence $x \leq y + d(x0)\mathsf{L}$ and $y \leq x + d(x0)\top$. Then

$$y^* \leq (x + d(x0)\top)^* = x^*(d(x0)\top x^*)^* = x^*(d(x0)\top)^* = x^* + x^* d(x0)\top \leq x^* + d(x^* x0)\top \leq x^* + d(x^* 0)\top$$

and therefore

$$\begin{aligned} x^* &\leq (y + d(x0)\mathsf{L})^* = y^*(d(x0)\mathsf{L}y^*) = y^*(d(x0)\mathsf{L})^* = y^* + y^* d(x0)\mathsf{L} = y^* + y^* 0 + d(y^* x0)\mathsf{L} \\ &\leq y^* + d((x^* + d(x^* 0)\top)x0)\mathsf{L} = y^* + d(x^* x0)\mathsf{L} + d(x^* 0)d(\top x0)\mathsf{L} \leq y^* + d(x^* 0)\mathsf{L} \end{aligned}$$

by Lemmas 5.1 and 5.4. Both inequalities together amount to $x^* \sqsubseteq y^*$. Furthermore,

$$\begin{aligned} y^\omega &\leq (x + d(x0)\top)^\omega = (x^* d(x0)\top)^\omega + (x^* d(x0)\top)^* x^\omega = x^* d(x0)\top (x^* d(x0)\top)^\omega + x^\omega + x^* d(x0)\top x^\omega \\ &\leq x^\omega + x^* d(x0)\top \leq x^\omega + d(x^* x0)\top \leq x^\omega + d(x^* 0)\top \leq x^\omega + d(x^\omega 0)\top \end{aligned}$$

and

$$\begin{aligned} x^\omega &\leq (y + d(x0)\mathsf{L})^\omega = (y^* d(x0)\mathsf{L})^\omega + (y^* d(x0)\mathsf{L})^* y^\omega = y^* d(x0)\mathsf{L} + y^\omega = y^\omega + y^* 0 + d(y^* x0)\mathsf{L} \\ &\leq y^\omega + y^\omega 0 + d(x^* 0)\mathsf{L} \leq y^\omega + d(x^\omega 0)\mathsf{L} \end{aligned}$$

by Lemmas 5.1 and 5.4. Both inequalities together amount to $x^\omega \sqsubseteq y^\omega$. Finally,

$$d(y^\omega)\mathsf{L} \leq d(x^\omega + d(x^\omega 0)\top)\mathsf{L} = d(x^\omega)\mathsf{L} + d(x^\omega 0)d(\top)\mathsf{L} = d(x^\omega)\mathsf{L} \leq d(x^\omega)\mathsf{L} + d(d(x^\omega)\mathsf{L}0)\top$$

and

$$d(x^\omega)\mathsf{L} = d(x^\omega)d(\mathsf{L})\mathsf{L} = d(d(x^\omega)\mathsf{L})\mathsf{L} \leq d(y^\omega)\mathsf{L} + d(d(x^\omega)\mathsf{L}0)\mathsf{L}$$

by Lemma 5.1. Both inequalities together amount to $d(x^\omega)\mathsf{L} \sqsubseteq d(y^\omega)\mathsf{L}$. Hence $x^\circ \sqsubseteq y^\circ$ by Theorem 6. □

We finally instantiate Corollary 9 to extended designs according to Theorems 2, 4 and 7. Using the domain element b in the condition of the loop we obtain

$$\begin{aligned}
&\mathsf{while}\ \langle 0|0|b\rangle\ \mathsf{do}\ \langle p|q|r\rangle \\
&= d((\langle 0|0|b\rangle\langle p|q|r\rangle)^\omega)\langle 0|\top|0\rangle + (\langle 0|0|b\rangle\langle p|q|r\rangle)^* a(\langle 0|0|b\rangle) \\
&= d(\langle bp \mid bq \mid br\rangle^\omega)\langle 0|\top|0\rangle + \langle bp \mid bq \mid br\rangle^*\langle 0|0|a(b)\rangle \\
&= d(\langle (br)^\omega + (br)^*bp \mid (br)^\omega + (br)^*bq \mid (br)^\omega + (br)^*bp\rangle)\langle 0|\top|0\rangle + \langle (br)^*bp \mid (br)^*bq \mid (br)^*\rangle\langle 0|0|a(b)\rangle \\
&= \langle 0|0|d((br)^\omega + (br)^*bq)\rangle\langle 0|\top|0\rangle + \langle (br)^*bp \mid (br)^*bq \mid (br)^*bp + (br)^*a(b)\rangle \\
&= \langle 0 \mid d((br)^\omega + (br)^*bq)\top \mid 0\rangle + \langle (br)^*bp \mid (br)^*bq \mid (br)^*(bp + a(b))\rangle \\
&= \langle 0 \mid (br)^\omega + (br)^*bq \mid 0\rangle + \langle (br)^*bp \mid (br)^*bq \mid (br)^*(bp + a(b))\rangle \\
&= \langle (br)^*bp \mid (br)^\omega + (br)^*bq \mid (br)^*(bp + a(b))\rangle\ .
\end{aligned}$$

The first component $(br)^*bp$ shows that the loop aborts if a state in p is reached after a finite number of iterations. Non-termination occurs for two reasons: either the loop is iterated infinitely ($(br)^\omega$) or after finitely many iterations a state is reached from which the body does not terminate ($(br)^*bq$). Normal termination may be observed in the aborting case ($(br)^*bp$) or if the loop terminates after a finite number of iterations ($(br)^*a(b)$).

5. Unified semantics of while-programs

In this section, we generalise our unified treatment of while-programs in partial, total and general correctness [15] to include extended designs. Recall the semantics of while-programs derived in Corollary 9, namely $\mathsf{while}\ p\ \mathsf{do}\ y = d((py)^\omega)\mathsf{L} + (py)^*a(p)$. We first argue that it is valid not only for extended designs, but also for partial, total and general correctness models. In these models $\mathsf{L} = \top 0$ holds.

- In partial correctness additionally $\mathsf{L} = 0$, whence $d((py)^\omega)\mathsf{L}+(py)^*a(p) = (py)^*a(p) = \mu(\lambda x.pyx+a(p))$. But this $\leq$-least fixpoint is the semantics of while-loops in partial correctness.

- In total correctness additionally $\mathsf{L} = \top$, whence

  $$d((py)^\omega)\mathsf{L} + (py)^*a(p) = d((py)^\omega)\top + (py)^*a(p) = (py)^\omega + (py)^*a(p) = \nu(\lambda x.pyx + a(p))$$

  provided $d(q)\top \leq q$ for every vector q. The latter condition is satisfied, in particular, by relational models such as the Unifying Theories of Programming. The $\leq$-greatest fixpoint is the semantics of while-loops in total correctness.

- General correctness is a special case of the present setting as regards axioms and approximation order, which are obtained by instantiating $\mathsf{L} = \top 0$. It follows that the current semantics of while-loops is valid for general correctness. In this case $d((py)^\omega)\mathsf{L} + (py)^*a(p) = (py)^\omega 0 + (py)^*a(p)$.

Observe that (L2) does not hold in partial correctness models because 0 is a right annihilator there, whence $d(\mathsf{L}0) = d(0) = 0 \neq 1$. Hence the unified semantics must be based on a modified set of axioms about L, which hold in all models we aim at. We take the two properties derived in Lemmas 5.1 and 5.3:

$$\begin{array}{ll}
(\mathsf{L4}) & \mathsf{L}x = \mathsf{L} \\
(\mathsf{L5}) & x\mathsf{L} = x0 + d(x)\mathsf{L}
\end{array}$$

Due to (L4) we can use the combined iteration $^\circ$ to denote the semantics of loops. It is easy to see that (L5) implies (L1), (L3) and the remaining properties shown in Lemma 5:

$$d(x)\mathsf{L} \leq x\mathsf{L} \qquad d(x0)\mathsf{L} \leq x0\mathsf{L} = x0 \leq x \qquad x\mathsf{L} \leq x0 + \mathsf{L} \qquad xd(y)\mathsf{L} = x0 + d(xy)\mathsf{L}$$

By Theorem 7 and Lemma 5, (L4) and (L5) hold for extended designs. Again by specialisation, these two axioms hold in general correctness models as well. They immediately follow in partial correctness where

$\mathsf{L} = 0$. In total correctness ($\mathsf{L4}$) holds since $\top 0 = \mathsf{L} = \top$ is a left annihilator, and ($\mathsf{L5}$) follows again provided $d(q)\top \leq q$ for every vector q.

For the remainder of this section we therefore assume that S is an omega algebra without right zero, with an antidomain operation and an element L satisfying ($\mathsf{L4}$) and ($\mathsf{L5}$). By the results obtained in Sections 3 and 4 and in our previous works [18, 15], relations, designs, prescriptions and extended designs are particular models which satisfy these axioms. The semantics of while-programs in all of these models is

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

where $x^\circ = d(x^\omega)\mathsf{L} + x^*$. Results proved using this semantics and applying only the axioms of S hold in all four correctness approaches.

An example for such a result is the normal form theorem for while-programs. Algebraic proofs have been given for partial correctness [25], total correctness [35] and a unified setting that covers partial, total and general correctness [15]. We generalise the latter proof to extended designs in the remainder of this section. This is prepared by showing that our new combined iteration operator $^\circ$ satisfies the following sliding, unfold, decomposition, preservation and import laws.

Lemma 11. *Let $x, y \in S$ and $p \in d(S)$. Then*

1. $x(yx)^\circ = (xy)^\circ x$.
2. $x^\circ = 1 + xx^\circ = 1 + x^\circ x$.
3. $(x + y)^\circ = (x^*y)^\circ x^\circ = (x^\circ y)^\circ x^\circ = x^\circ (yx^\circ)^\circ$.
4. $yx \leq xy \Rightarrow yx^\circ \leq x^\circ y$.
5. $px \leq xp \Rightarrow px^\circ = p(px)^\circ$.

Proof.

1. By ($\mathsf{L5}$), sliding and ($\mathsf{L4}$) we have

$$\begin{aligned} x(yx)^\circ &= xd((yx)^\omega)\mathsf{L} + x(yx)^* = d(x(yx)^\omega)\mathsf{L} + x0 + x(yx)^* = d((xy)^\omega)\mathsf{L} + x(yx)^* \\ &= d((xy)^\omega)\mathsf{L}x + (xy)^*x = (xy)^\circ x \ . \end{aligned}$$

2. By ($\mathsf{L4}$) and unfold we obtain $1 + x^\circ x = 1 + d(x^\omega)\mathsf{L}x + x^*x = d(x^\omega)\mathsf{L} + x^* = x^\circ$. Moreover, $xx^\circ = x^\circ x$ by part 1.
3. By ($\mathsf{L4}$), ($\mathsf{L5}$) and decomposition we have

$$\begin{aligned} (x^*y)^\circ x^\circ &= d((x^*y)^\omega)\mathsf{L}x^\circ + (x^*y)^*x^\circ = d((x^*y)^\omega)\mathsf{L} + (x^*y)^*d(x^\omega)\mathsf{L} + (x^*y)^*x^* \\ &= d((x^*y)^\omega)\mathsf{L} + d((x^*y)^*x^\omega)\mathsf{L} + (x^*y)^*0 + (x^*y)^*x^* \\ &= d((x^*y)^\omega + (x^*y)^*x^\omega)\mathsf{L} + (x^*y)^*x^* = d((x+y)^\omega)\mathsf{L} + (x+y)^* = (x+y)^\circ \ . \end{aligned}$$

   Therefore,

$$\begin{aligned} (x^\circ y)^\circ x^\circ &= (d(x^\omega)\mathsf{L} + x^*y)^\circ x^\circ = ((x^*y)^*d(x^\omega)\mathsf{L})^\circ (x^*y)^\circ x^\circ = (x^*y)^\circ x^\circ + (x^*y)^*d(x^\omega)\mathsf{L} \\ &= (x^*y)^\circ x^\circ + (x^*y)^*0 + d((x^*y)^*x^\omega)\mathsf{L} = (x^*y)^\circ x^\circ + d((x^*y)^*x^\omega)\mathsf{L} = (x^*y)^\circ x^\circ \end{aligned}$$

   using ($\mathsf{L4}$), part 2, ($\mathsf{L5}$) and $(x^*y)^* \leq (x^*y)^\circ$ and $d((x^*y)^*x^\omega)\mathsf{L} \leq d((x+y)^\omega)\mathsf{L} \leq (x+y)^\circ = (x^*y)^\circ x^\circ$. The remaining equality follows by part 1.
4. Let $yx \leq xy$. Then $yx^* \leq x^*y$ and $yx^\omega \leq x^\omega$ [15, Lemma 8]. With ($\mathsf{L5}$) and ($\mathsf{L4}$) we have

$$yx^\circ = yd(x^\omega)\mathsf{L} + yx^* = d(yx^\omega)\mathsf{L} + y0 + yx^* \leq d(x^\omega)\mathsf{L} + y^*x \leq d(x^\omega)\mathsf{L}y + x^*y = x^\circ y \ .$$

5. Let $px \leq xp$. Then $px^* = p(px)^*$ and $px^\omega = p(px)^\omega$ [15, Lemma 9]. Hence

$$px^\circ = pd(x^\omega)\mathsf{L} + px^* = d(px^\omega)\mathsf{L} + px^* = d(p(px)^\omega)\mathsf{L} + px^* = pd((px)^\omega)\mathsf{L} + p(px)^* = p(px)^\circ \ .$$

□

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 $x \in S$ *preserves* a domain element $p \in d(S)$ if $px \leq xp$ and $a(p)x \leq xa(p)$. The element $s \in S$ *assigns* p *to* q if $s = s(pq + a(p)a(q))$ for domain elements $p, q \in d(S)$.

Theorem 12. *Every while-program, suitably augmented with assigning elements, is equivalent to a while-program in normal form under certain preservation assumptions.*

Proof. The proof of [15, Theorem 10] can be reused as it stands. It successively applies program transformations which move while-loops out of the different kinds of program constructs. These transformations rely only on the properties of $^\circ$ shown in Lemma 11 and the axioms making S an antidomain semiring without right zero. □

Thus the normal form theorem is valid for relations, designs, prescriptions, extended designs and any other model satisfying the common axioms. The ability to provide such general results and to reuse existing proofs is a benefit of the algebraic approach to unify programs.

6. Conclusion

The matrix representation developed in Sections 2 and 3 helps to establish a proper definition of extended designs, the effect of basic program constructs, and the laws they satisfy. Turning these laws into axioms of an algebraic structure helps to deal with the complex approximation order, to provide manageable representations for recursion and iteration, and to compare with other models of programs. A particular subset of the axioms is small enough to have models as diverse as relations, designs, prescriptions and extended designs, yet large enough to yield complex program transformations such as those bringing while-programs to their normal form.

Acknowledgement

I thank Steve Dunne, Ian Hayes and Georg Struth for helpful discussions and the anonymous referees for useful comments. This work was supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).

Appendix

A. The original matrix representation of extended designs

In this section, we return to the original convention of extended designs representing the non-aborting and the terminating states. Following [29], we let the matrix entries be elements of Boolean semirings. A further generalisation to the ideal condition semirings of [18] is possible, but not essential for the upcoming development.

A *Boolean semiring* is an algebraic structure $(S, +, 0, \curlywedge, \top, \cdot, 1, \bar{\ })$ such that $(S, +, 0, \cdot, 1)$ is a semiring and $(S, +, 0, \curlywedge, \top, \bar{\ })$ is a Boolean algebra. This can be obtained by extending the bounded idempotent semiring axioms of Section 3.1 with Huntington's axiom

$$x = \overline{\overline{x} + \overline{y}} + \overline{\overline{x} + y}$$

and defining $x \curlywedge y =_{\text{def}} \overline{\overline{x} + \overline{y}}$ [27]. The operation $\curlywedge$ has the same precedence as $+$.

Let S be a Boolean semiring and $p, q, r \in S$ such that $\overline{p}$ and $\overline{q}$ are vectors. Then the extended design $(p|q|r)$ is given by

$$(p|q|r) \;=_{\mathrm{def}}\; \langle \overline{p} \mid \overline{p} + \overline{q} \mid \overline{p} + r \rangle \; .$$

Conversely, we have $\langle p|q|r \rangle = \langle p \mid p + q \mid p + r \rangle = \langle \overline{\overline{p}} \mid \overline{\overline{p}} + \overline{\overline{q}} \mid \overline{\overline{p}} + r \rangle = (\overline{p}|\overline{q}|r)$ for each $\langle p|q|r \rangle \in \mathrm{ED}(S)$. This shows that the set of extended designs over S remains unchanged. On the other hand, as for designs, the representation in terms of the notation $(\cdot|\cdot|\cdot)$ is not unique. In particular, we have $(p|q|r) = (p \mid \overline{p} + q \mid \overline{p} + r) = (p \mid p \leftthreetimes q \mid p \leftthreetimes r)$ where each inequality of $p \leftthreetimes q \leq q \leq \overline{p} + q$ and $p \leftthreetimes r \leq r \leq \overline{p} + r$ may be strict.

In $(p|q|r)$ the vector p represents the set of states from which no aborting execution exists. The vector q represents those states from which no non-terminating execution exists, unless implicitly caused by an aborting execution. The element r represents the executions which terminate normally, including those implicitly caused by an aborting execution.

We now express the effect of non-deterministic choice, sequential composition, finite and infinite iteration using the original convention. The non-deterministic choice of two extended designs is

$$\begin{aligned}(p_1|q_1|r_1) + (p_2|q_2|r_2) &= \langle \overline{p_1} \mid \overline{p_1} + \overline{q_1} \mid \overline{p_1} + r_1 \rangle + \langle \overline{p_2} \mid \overline{p_2} + \overline{q_2} \mid \overline{p_2} + r_2 \rangle \\ &= \langle \overline{p_1} + \overline{p_2} \mid \overline{p_1} + \overline{q_1} + \overline{p_2} + \overline{q_2} \mid \overline{p_1} + r_1 + \overline{p_2} + r_2 \rangle \\ &= \langle \overline{p_1 \leftthreetimes p_2} \mid \overline{p_1 \leftthreetimes p_2} + \overline{q_1 \leftthreetimes q_2} \mid \overline{p_1 \leftthreetimes p_2} + r_1 + r_2 \rangle = (p_1 \leftthreetimes p_2 \mid q_1 \leftthreetimes q_2 \mid r_1 + r_2) \; .\end{aligned}$$

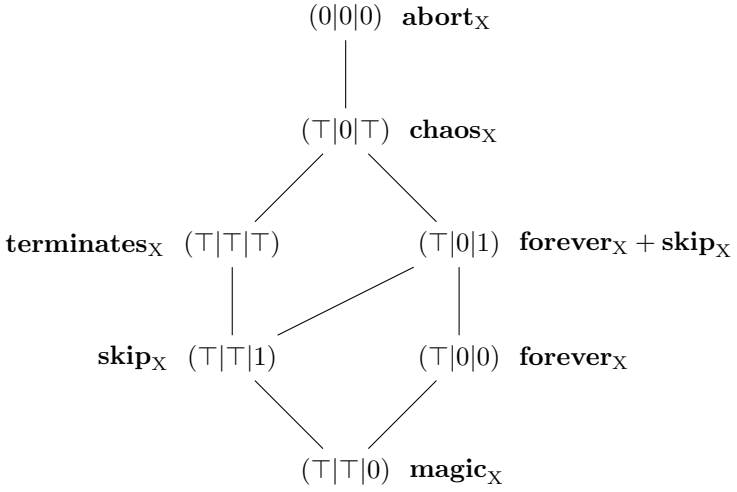
This generalises to arbitrary choice by $\sum_{i \in I} (p_i|q_i|r_i) = (\between_{i \in I} p_i \mid \between_{i \in I} q_i \mid \sum_{i \in I} r_i)$ provided the occurring meets and joins exist. The equality of two extended designs amounts to

$$\begin{aligned}(p_1|q_1|r_1) = (p_2|q_2|r_2) \quad &\Leftrightarrow \quad \langle \overline{p_1} \mid \overline{p_1} + \overline{q_1} \mid \overline{p_1} + r_1 \rangle = \langle \overline{p_2} \mid \overline{p_2} + \overline{q_2} \mid \overline{p_2} + r_2 \rangle \\ &\Leftrightarrow \quad (\overline{p_1} = \overline{p_2}) \wedge (\overline{p_1} + \overline{q_1} = \overline{p_2} + \overline{q_2}) \wedge (\overline{p_1} + r_1 = \overline{p_2} + r_2) \\ &\Leftrightarrow \quad (p_1 = p_2) \wedge (p_1 \leftthreetimes q_1 = p_2 \leftthreetimes q_2) \wedge (p_1 \leftthreetimes r_1 = p_2 \leftthreetimes r_2) \; .\end{aligned}$$

The natural order of extended designs is

$$\begin{aligned}(p_1|q_1|r_1) \leq (p_2|q_2|r_2) \quad &\Leftrightarrow \quad \langle \overline{p_1} \mid \overline{p_1} + \overline{q_1} \mid \overline{p_1} + r_1 \rangle \leq \langle \overline{p_2} \mid \overline{p_2} + \overline{q_2} \mid \overline{p_2} + r_2 \rangle \\ &\Leftrightarrow \quad (\overline{p_1} \leq \overline{p_2}) \wedge (\overline{p_1} + \overline{q_1} \leq \overline{p_2} + \overline{q_2}) \wedge (\overline{p_1} + r_1 \leq \overline{p_2} + r_2) \\ &\Leftrightarrow \quad (\overline{p_1} \leq \overline{p_2}) \wedge (\overline{q_1} \leq \overline{p_2} + \overline{q_2}) \wedge (r_1 \leq \overline{p_2} + r_2) \\ &\Leftrightarrow \quad (p_2 \leq p_1) \wedge (p_2 \leftthreetimes q_2 \leq q_1) \wedge (p_2 \leftthreetimes r_1 \leq r_2) \; .\end{aligned}$$

Hence the Hasse diagram of Section 3.1 extended by the skip program $(\top|\top|1)$ and $(\top|0|1)$ can be redrawn as



Since S is Boolean now we also have the greatest lower bound of two extended designs. It is obtained via the matrix representation, or by recalling

$$\begin{aligned}(p_1|q_1|r_1) \leq (p_2|q_2|r_2) \quad &\Leftrightarrow \quad (\overline{p_1} \leq \overline{p_2}) \wedge (\overline{p_1} + \overline{q_1} \leq \overline{p_2} + \overline{q_2}) \wedge (\overline{p_1} + r_1 \leq \overline{p_2} + r_2) \\ &\Leftrightarrow \quad (p_2 \leq p_1) \wedge (p_2 \leftthreetimes q_2 \leq p_1 \leftthreetimes q_1) \wedge (\overline{p_1} + r_1 \leq \overline{p_2} + r_2)\end{aligned}$$

from the calculation above, whence

$$(p_1|q_1|r_1) \curlywedge (p_2|q_2|r_2) = (p_1 + p_2 \mid (p_1 \curlywedge q_1) + (p_2 \curlywedge q_2) \mid (\overline{p_1} + r_1) \curlywedge (\overline{p_2} + r_2)) \; .$$

This generalises to arbitrary meets $\bigwedge_{i\in I}(p_i|q_i|r_i) = (\sum_{i\in I} p_i \mid \sum_{i\in I} p_i \curlywedge q_i \mid \bigwedge_{i\in I} \overline{p_i} + r_i)$ provided the occurring meets and joins exist and $\cdot$ distributes over the joins. On the other hand, a complement operation cannot be defined on extended designs. The sequential composition of two extended designs is

$$\begin{aligned}(p_1|q_1|r_1) \cdot (p_2|q_2|r_2) &= \langle \overline{p_1} \mid \overline{p_1} + \overline{q_1} \mid \overline{p_1} + r_1\rangle \cdot \langle \overline{p_2} \mid \overline{p_2} + \overline{q_2} \mid \overline{p_2} + r_2\rangle \\ &= \langle \overline{p_1} + (\overline{p_1} + r_1)\overline{p_2} \mid \overline{p_1} + \overline{q_1} + (\overline{p_1} + r_1)(\overline{p_2} + \overline{q_2}) \mid \overline{p_1} + (\overline{p_1} + r_1)(\overline{p_2} + r_2)\rangle \\ &= \langle \overline{p_1} + r_1\overline{p_2} \mid \overline{p_1} + r_1\overline{p_2} + \overline{q_1} + r_1\overline{q_2} \mid \overline{p_1} + r_1\overline{p_2} + r_1r_2\rangle \\ &= (p_1 \curlywedge \overline{r_1\overline{p_2}} \mid q_1 \curlywedge \overline{r_1\overline{q_2}} \mid r_1r_2)\end{aligned}$$

since $\overline{p_1}x \leq \overline{p_1}\top = \overline{p_1}$. We therefore obtain the following algebraic structure for extended designs over Boolean semirings.

Theorem 13. *Let S be a Boolean semiring. Then $(\mathrm{ED}(S), +, (\top|\top|0), \curlywedge, (0|0|0))$ is a bounded distributive lattice and $(\mathrm{ED}(S), +, (\top|\top|0), \cdot, (\top|\top|1))$ is an idempotent semiring without right zero.*

The antidomain operation for extended designs over a Boolean antidomain semiring is obtained as

$$a((p|q|r)) = a(\langle \overline{p} \mid \overline{p} + \overline{q} \mid \overline{p} + r\rangle) = \langle 0|0|a(\overline{p} + \overline{q} + \overline{p} + r)\rangle = \langle 0|0|a(\overline{p} + \overline{q} + r)\rangle = (\top|\top|a(\overline{p} + \overline{q} + r)) \; ,$$

whence the domain is $d((p|q|r)) = (\top|\top|d(\overline{p} + \overline{q} + r))$. The operation a in the underlying Boolean antidomain semiring S should not be confused with the general complement $\overline{\ }$ in S.

If S is a relation algebra, the domain operation can be defined explicitly by $d(x) = x\top \curlywedge 1$ and the antidomain is $a(x) = \overline{x\top} \curlywedge 1$. In our more general setting of a Boolean antidomain semiring we only have

$$x\top \curlywedge 1 = (d(x) + a(x))(x\top \curlywedge 1) = d(x)(x\top \curlywedge 1) + a(x)(x\top \curlywedge 1) \leq d(x)1 + a(x)x\top = d(x) + 0 = d(x) \; ,$$

but the converse $d(x) \leq x\top \curlywedge 1$ does not hold as a counterexample generated by Mace4 shows.

The Kleene star for extended designs over a Boolean Kleene algebra is

$$\begin{aligned}(p|q|r)^* &= \langle \overline{p} \mid \overline{p} + \overline{q} \mid \overline{p} + r\rangle^* = \langle (\overline{p} + r)^*\overline{p} \mid (\overline{p} + r)^*(\overline{p} + \overline{q}) \mid (\overline{p} + r)^*\rangle \\ &= \langle (\overline{p} + r)^*\overline{p} \mid (\overline{p} + r)^*\overline{p} + (\overline{p} + r)^*\overline{q} \mid (\overline{p} + r)^*\overline{p} + (\overline{p} + r)^*\rangle = (\overline{(\overline{p} + r)^*\overline{p}} \mid \overline{(\overline{p} + r)^*\overline{q}} \mid (\overline{p} + r)^*)\end{aligned}$$

since $(\overline{p}+r)^*\overline{p} \leq (\overline{p}+r)^*(\overline{p}+r) \leq (\overline{p}+r)^*$. The omega operation for extended designs over a Boolean omega algebra is

$$\begin{aligned}(p|q|r)^\omega &= \langle \overline{p} \mid \overline{p} + \overline{q} \mid \overline{p} + r\rangle^\omega = \langle (\overline{p} + r)^\omega + (\overline{p} + r)^*\overline{p} \mid (\overline{p} + r)^\omega + (\overline{p} + r)^*(\overline{p} + \overline{q}) \mid (\overline{p} + r)^\omega + (\overline{p} + r)^*\overline{p}\rangle \\ &= \langle r^\omega + (\overline{p} + r)^*\overline{p} \mid r^\omega + (\overline{p} + r)^*\overline{p} + (\overline{p} + r)^*\overline{q} \mid r^\omega + (\overline{p} + r)^*\overline{p} + 0\rangle \\ &= (\overline{r^\omega + (\overline{p} + r)^*\overline{p}} \mid \overline{(\overline{p} + r)^*\overline{q}} \mid 0)\end{aligned}$$

since $(\overline{p} + r)^\omega + (\overline{p} + r)^*\overline{p} = r^\omega + (\overline{p} + r)^*\overline{p}$ follows from

$$(\overline{p} + r)^\omega = (r^*\overline{p})^\omega + (r^*\overline{p})^*r^\omega = r^*\overline{p}(r^*\overline{p})^\omega + r^\omega + r^*\overline{p}(r^*\overline{p})^*r^\omega \leq r^*\overline{p}\top + r^\omega = r^\omega + r^*\overline{p} \leq r^\omega + (\overline{p} + r)^*\overline{p} \; .$$

As expected from Section 2.3.5, the approximation order amounts to

$$\begin{aligned}(p_1|q_1|r_1) \sqsubseteq (p_2|q_2|r_2) \quad &\Leftrightarrow \quad \langle \overline{p_1} \mid \overline{p_1} + \overline{q_1} \mid \overline{p_1} + r_1\rangle \sqsubseteq \langle \overline{p_2} \mid \overline{p_2} + \overline{q_2} \mid \overline{p_2} + r_2\rangle \\ &\Leftrightarrow \quad (\overline{p_1} \leq \overline{p_2}) \wedge (\overline{p_2} + \overline{q_2} \leq \overline{p_1} + \overline{q_1}) \wedge (\overline{p_1} + r_1 \leq \overline{p_2} + r_2 \leq \overline{p_1} + \overline{q_1} + \overline{p_1} + r_1) \\ &\Leftrightarrow \quad (\overline{p_1} \leq \overline{p_2}) \wedge (\overline{p_2} + \overline{q_2} \leq \overline{p_1} + \overline{q_1}) \wedge (r_1 \leq \overline{p_2} + r_2) \wedge (r_2 \leq \overline{p_1} + \overline{q_1} + r_1) \\ &\Leftrightarrow \quad (p_2 \leq p_1) \wedge (p_2 \curlywedge r_1 \leq r_2) \wedge (p_1 \curlywedge q_1 \leq p_2 \curlywedge q_2) \wedge (p_1 \curlywedge q_1 \curlywedge r_2 \leq r_1) \; .\end{aligned}$$

Using a domain element b, the semantics of while-loops under the original convention is

$$\begin{aligned}&\mathsf{while}\ (\top|\top|b)\ \mathsf{do}\ (p|q|r) \\ &= \mathsf{while}\ \langle 0|0|b\rangle\ \mathsf{do}\ \langle \overline{p} \mid \overline{p} + \overline{q} \mid \overline{p} + r\rangle \\ &= \langle (b(\overline{p} + r))^*b\overline{p} \mid (b(\overline{p} + r))^\omega + (b(\overline{p} + r))^*b(\overline{p} + \overline{q}) \mid (b(\overline{p} + r))^*(b\overline{p} + a(b))\rangle \\ &= \langle (b(\overline{p} + r))^*b\overline{p} \mid (br)^\omega + (b(\overline{p} + r))^*b(\overline{p} + \overline{q}) \mid (b(\overline{p} + r))^*(b\overline{p} + a(b))\rangle \\ &= (\overline{(b(\overline{p} + r))^*b\overline{p}} \mid \overline{(br)^\omega + (b(\overline{p} + r))^*b\overline{q}} \mid (b(\overline{p} + r))^*a(b))\end{aligned}$$

since $(b\overline{p} + br)^\omega \leq (br)^\omega + (b\overline{p} + br)^*b\overline{p}$ as above.

References

[1] 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.

[2] R. Berghammer and H. Zierer. Relational algebraic semantics of deterministic and nondeterministic programs. *Theoretical Computer Science*, 43:123–147, 1986.

[3] 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-Verlag, 1979.

[4] 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-Verlag, 2006.

[5] 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-Verlag, 2000.

[6] J. Desharnais, B. Möller, and G. Struth. Kleene algebra with domain. *ACM Transactions on Computational Logic*, 7(4):798–833, October 2006.

[7] J. Desharnais and G. Struth. Internal axioms for domain semirings. *Science of Computer Programming*, 76(3):181–203, March 2011.

[8] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[9] H. Doornbos. A relational model of programs without the restriction to Egli-Milner-monotone constructs. In E.-R. Olderog, editor, *Programming Concepts, Methods and Calculi*, pages 363–382. North-Holland Publishing Company, 1994.

[10] 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, July 2001.

[11] 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-Verlag, 2010.

[12] Z. Ésik and H. Leiß. Algebraically complete semirings and Greibach normal form. *Annals of Pure and Applied Logic*, 3(1–3):173–203, May 2005.

[13] 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-Verlag, 2009.

[14] 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-Verlag, 2010.

[15] 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-Verlag, 2010.

[16] 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-Verlag, 2010.

[17] 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-Verlag, 2011.

[18] W. Guttmann and B. Möller. Normal design algebra. *Journal of Logic and Algebraic Programming*, 79(2):144–173, February 2010.

[19] 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-Verlag, 2010.

[20] E. C. R. Hehner and A. J. Malton. Termination conventions and comparative semantics. *Acta Informatica*, 25(1):1–14, 1988.

[21] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580/583, October 1969.

[22] C. A. R. Hoare and J. He. *Unifying theories of programming*. Prentice Hall Europe, 1998.

[23] D. Jacobs and D. Gries. General correctness: A unification of partial and total correctness. *Acta Informatica*, 22(1):67–83, April 1985.

[24] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, May 1994.

[25] D. Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.

[26] D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Transactions on Computational Logic*, 1(1):60–76, July 2000.

[27] R. D. Maddux. Relation-algebraic semantics. *Theoretical Computer Science*, 160(1–2):1–85, June 1996.

[28] V. Mathieu and J. Desharnais. Verification of pushdown systems using omega algebra with domain. In W. MacCaull, M. Winter, and I. Düntsch, editors, *Relational Methods in Computer Science 2005*, volume 3929 of *Lecture Notes in Computer Science*, pages 188–199. Springer-Verlag, 2006.

[29] 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-Verlag, 2006.

[30] B. Möller. Kleene getting lazy. *Science of Computer Programming*, 65(2):195–214, March 2007.

[31] B. Möller and G. Struth. Algebras of modal operators and partial correctness. *Theoretical Computer Science*, 351(2):221–239, February 2006.
[32] 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-Verlag, 2006.
[33] G. Nelson. A generalization of Dijkstra's calculus. *ACM Transactions on Programming Languages and Systems*, 11(4):517–561, October 1989.
[34] D. Parnas. A generalized control structure and its formal definition. *Communications of the ACM*, 26(8):572–581, August 1983.
[35] K. Solin. Normal forms in total correctness for while programs and action systems. *Journal of Logic and Algebraic Programming*, 80(6):362–375, August 2011.
[36] H. Søndergaard and P. Sestoft. Non-determinism in functional languages. *The Computer Journal*, 35(5):514–523, October 1992.
[37] J. von Wright. Towards a refinement algebra. *Science of Computer Programming*, 51(1–2):23–45, May 2004.