Partial, Total and General Correctness

Walter Guttmann

Institut für Programmiermethodik und Compilerbau
Universität Ulm, 89069 Ulm, Germany
walter.guttmann@uni-ulm.de

Abstract. We identify weak semirings, which drop the right annihilation axiom $a0 = 0$, as a common foundation for partial, total and general correctness. It is known how to extend weak semirings by operations for finite and infinite iteration and domain. We use the resulting weak omega algebras with domain to define a semantics of while-programs which is valid in all three correctness approaches. The unified, algebraic semantics yields program transformations at once for partial, total and general correctness. We thus give a proof of the normal form theorem for while-programs, which is a new result for general correctness and extends to programs with non-deterministic choice.
By adding specific axioms to the common ones, we obtain partial, total or general correctness as a specialisation. We continue our previous investigation of axioms for general correctness. In particular, we show that a subset of these axioms is sufficient to derive a useful theory, which includes the Egli-Milner order, full recursion, correctness statements and a correctness calculus. We also show that this subset is necessary.

1 Introduction

Partial, total and general correctness are three approaches to the semantics of programs distinguished by [25]. Leaving the characterisation to Section 3, at this point we just give a sample of the available literature: For example, partial correctness is supported by Hoare logic [21], weakest liberal preconditions [12] and Kleene algebra with tests [28]; total correctness is supported by weakest preconditions [12], the Unifying Theories of Programming [22], demonic refinement algebra [37] and demonic algebra [5]; general correctness is supported by the works [2, 4, 25, 3, 34, 14, 32, 17].

Despite various links between partial, total and general correctness, the approaches differ essentially by giving distinct semantics to programs and distinct laws about programs. As a consequence, a particular program transformation may be applicable in one approach, but not in the others. Even if a transformation rule is valid in all three approaches, this has to be proved separately for each of them.

Here comes into play the algebraic approach of identifying basic laws that programs satisfy, collecting them as axioms of algebraic structures, and taking programs as elements of these algebras. The first benefit is that reasoning carried out at the algebraic level is valid in any concrete model that satisfies the axioms; typically there are several in the literature. The second benefit is that reasoning can be supported by automated theorem provers and counterexample generators such as Prover9/Mace4, since the axioms are usually first order conditional equations.

It turns out that partial, total and general correctness have a fairly large set of common axioms, certainly large enough to prove complex program transformations. Such a transformation will then be valid in all three approaches, since each of them satisfies the common axioms, and possibly further ones. So the third benefit is that reasoning has to be performed only once, provided the transformation is stated so that it meaningfully talks about programs in each particular approach. At least for while-programs this can be achieved, as this paper shows.

In Section 2, we present the common set of axioms, namely weak omega algebra with domain.

In Section 3, we discuss which further axioms have to be added to obtain partial, total or general correctness. We particularly focus on general correctness, since it is developed less well than the others. The main contribution here is a reduction of our previous axioms [17] to the minimum necessary to represent the Egli-Milner order, which is used for the semantics of loops and recursion.

In Section 4, we show that this minimal axiomatisation is still sufficient to develop a useful theory of general correctness, which includes full recursion, correctness statements and a correctness calculus. Some results require new proofs due to the reduced set of axioms. Although carried out in a general correctness setting, the development provides a semantics of while-loops that also suits partial and total correctness.

This is established in Section 5, where we return to the common axioms. We use the unified semantics to state and prove transformations that bring while-programs to a normal form. This result is known for partial correctness [27] and has recently been extended to total correctness [35], but both proofs use a program semantics and axioms specific to the respective correctness approach. Our proof avoids this and thus gives a transformation valid in all three approaches; in particular, this is new for general correctness. Additionally, we extend the result to programs with non-deterministic choice.

2 Common Axioms

In this section, we present the axioms which are common to partial, total and general correctness. They are grouped in algebraic structures which are fundamental to many branches of computer science, not only program semantics.

Former investigations have shown that these structures underly various correctness approaches. For a detailed discussion and concrete models we refer to [28, 31, 9] for partial correctness, to [37, 19, 30, 5, 20] for total correctness, and to [32, 17] for general correctness.

2.1 Choice and Composition

A *weak semiring* is a structure $(S, +, 0, \cdot, 1)$ such that $(S, +, 0)$ is a commutative monoid, $(S, \cdot, 1)$ is a monoid, the operation $\cdot$ distributes over $+$ in both arguments and 0 is a left annihilator, that is, $0 \cdot a = 0$. We assume $0 \neq 1$, otherwise S would be trivial. A *semiring* is a weak semiring in which 0 is also a right annihilator, that is, $a \cdot 0 = 0$. We frequently abbreviate $a \cdot b$ with ab.

A weak semiring is *idempotent* if $+$ is, that is, if $a + a = a$. In an idempotent weak semiring the relation $a \leq b \Leftrightarrow_{\mathrm{def}} a + b = b$ is a partial order, called the *natural order* on S [9]. The operations $+$ and $\cdot$ are isotone and $a + b$ is the join of a and b with respect to the natural order. An idempotent weak semiring is *bounded* if it has a greatest element $\top$, such that $a \leq \top$.

Our notation reflects the intended relational model, where elements of S represent the state transition relation or input/output behaviour of programs. In this model, the operations $+$ and $\cdot$ are non-deterministic choice and sequential composition, respectively. Moreover, 0, 1 and $\top$ are the empty, identity and universal relations, respectively, and $\leq$ is the subset order, so that $0 \leq 1 \leq \top$ holds, for example. Hence $a \leq b$ states that a is a refinement of b. To avoid confusion, it should be kept in mind that other approaches in the literature use different conventions, such as the reverse order in [37].

2.2 Finite and Infinite Iteration

A *(weak) Kleene algebra* [26, 29] is a structure $(S, ^*)$ such that S is an idempotent (weak) semiring and the operation star * satisfies the unfold and induction laws

$$\begin{array}{ll} 1 + a \cdot a^* \leq a^* & \qquad b + a \cdot c \leq c \Rightarrow a^* \cdot b \leq c \\ 1 + a^* \cdot a \leq a^* & \qquad b + c \cdot a \leq c \Rightarrow b \cdot a^* \leq c \end{array}$$

for $a, b, c \in S$. As a consequence, a^*b is the least fixpoint of $\lambda x.ax + b$ and * is isotone with respect to $\leq$. We also have the sliding law $a(ba)^* = (ab)^*a$ and the decomposition laws $(a + b)^* = a^*(ba^*)^* = (a^*b)^*a^*$.

A *(weak) omega algebra* [6, 29] is a structure $(S, ^\omega)$ such that S is a (weak) Kleene algebra and the operation omega $^\omega$ satisfies the unfold and co-induction laws

$$a^\omega = a \cdot a^\omega \qquad\qquad c \leq a \cdot c + b \Rightarrow c \leq a^\omega + a^* \cdot b$$

for $a, b, c \in S$. As a consequence, $a^\omega + a^*b$ is the greatest fixpoint of $\lambda x.ax + b$ and $^\omega$ is isotone with respect to $\leq$. Any weak omega algebra is bounded since $1^\omega = \top$. Moreover, $a^\omega \top = a^\omega = a^*a^\omega$ and $a^*0 \leq a^*a^\omega 0 = a^\omega 0$ and $c \leq a \cdot c \Rightarrow c \leq a^\omega$, whence a^ω is the greatest fixpoint of $\lambda x.ax$.

The Kleene star and omega operations represent finite and infinite iteration, respectively. They are used to conveniently express the semantics of loops.

2.3 Tests and Domain

A *test semiring* [29] is an idempotent weak semiring $(S, +, 0, \cdot, 1)$ with a distinguished set of elements $\mathrm{test}(S) \subseteq S$ called *tests* and a *negation* operation $\neg$ on tests such that $(\mathrm{test}(S), +, 0, \cdot, 1, \neg)$ is a Boolean algebra. We also write $\overline{p}$ for $\neg p$.

Tests allow us to represent the conditional statement by if p then a else $b =_{\mathrm{def}} pa+\overline{p}b$. Note that negation applies only to elements of test(S), not to all elements of S. We use the letters p, q, r to denote tests in this paper.

A *domain semiring* [9] is a structure $(S, \ulcorner)$ such that S is a test semiring and the domain operation $\ulcorner : S \to \mathrm{test}(S)$ satisfies the axioms

$$a \leq \ulcorner a \cdot a \qquad \ulcorner(p \cdot a) \leq p \qquad \ulcorner(a \cdot \ulcorner b) \leq \ulcorner(a \cdot b)$$

for $a, b \in S$ and $p \in \mathrm{test}(S)$. The alternative axiomatisation of [11, 10] can also be used. Useful properties for $a, b \in S$ and $p \in \mathrm{test}(S)$ are

$$\begin{array}{llll} \ulcorner a \leq p \Leftrightarrow a \leq pa & a \leq b \Rightarrow \ulcorner a \leq \ulcorner b & a = \ulcorner a a & \ulcorner p = p \\ a \leq 0 \Leftrightarrow \ulcorner a \leq 0 & \ulcorner(a+b) = \ulcorner a + \ulcorner b & \ulcorner(pa) = p \ulcorner a & \ulcorner(a \cdot \ulcorner b) = \ulcorner(a \cdot b) \end{array}$$

If S is bounded, a characterisation of domain is $\ulcorner a \leq p \Leftrightarrow a \leq p\top$ [1]. We also use its instance $a \leq \ulcorner a\top$.

An element $a \in S$ is *total* if $\ulcorner a = 1$. In a bounded setting without domain, we can use $a\top = \top$ instead. In a bounded domain semiring, $a\top = \top$ implies $\ulcorner a = 1$, but the converse does not hold in general.

The domain of the program a represents the initial states from which a transition under a is possible. In this paper, we use it for the axiomatic treatment of general correctness in Sections 3.3 and 4; see [31, 5, 20] for applications of the domain operation in partial and total correctness.

3 Axioms for Partial, Total and General Correctness

In this section, we characterise partial, total and general correctness, and show how to obtain each by extending the axioms introduced in Section 2. We particularly focus on general correctness and the axioms necessary to represent the Egli-Milner order. The general setting is a bounded domain semiring; recursion is treated in Section 4.

To motivate the axioms, let loop denote the endless loop or the program which never terminates [34]. In a conventional setting, we expect that the endless loop satisfies $\mathsf{loop} \cdot a = \mathsf{loop}$ for all programs a, and $a \cdot \mathsf{loop} = \mathsf{loop}$ if a is total. Based on these requirements, loop is formally characterised below. For a theory without the law $\mathsf{loop} \cdot a = \mathsf{loop}$, describing lazy execution of programs, we refer to [18].

3.1 Partial Correctness

Because partial correctness ignores termination issues, there is no need to represent that the execution of a program might not terminate. Nevertheless a semantics must be assigned to loop, and this is done by interpreting the absence of terminating executions as non-termination. As a consequence, if a program has both a terminating and a non-terminating execution, the terminating one is chosen, simply because it is present. Thus partial correctness can be informally characterised by ‘termination absorbs non-termination’.

In the algebraic setting this translates to the law $\mathsf{loop} + a = a$ for each program a, hence $+$ models angelic non-determinism according to [36]. By definition of the natural order, we thus obtain $\mathsf{loop} \leq a$ for each a, and therefore $\mathsf{loop} = 0$. It follows that loop is the least solution of the equation $a = a$, or the least fixpoint of the corresponding identity function. This motivates why recursion is modelled by least fixpoints with respect to the natural order.

Another consequence of $\mathsf{loop} = 0$ is the law $a \cdot 0 = 0$ for total a, and hence even for all a. This law cannot be proved from the axioms introduced in Section 2, but has to be added as the characteristic axiom of partial correctness. Hence the underlying structures are semirings, Kleene algebras and omega algebras, without the qualifier 'weak'. Observe that the new axiom can equivalently be expressed as $\top \cdot 0 = 0$.

Theories of partial correctness include Hoare logic [21], weakest liberal preconditions [12], Kleene algebra with tests [28] and with domain [9].

3.2 Total Correctness

On the other hand, total correctness takes the issue of termination very seriously; in concrete models it is typically represented by adding a special value, predicate or variable. To guarantee that the execution of a program terminates irrespective of non-deterministic choices, only the complete absence of non-terminating executions is interpreted as termination. As a consequence, if a program has both a terminating and a non-terminating execution, the non-terminating one is chosen. Thus total correctness can be informally characterised by 'non-termination absorbs termination'.

In the algebraic setting this translates to the law $\mathsf{loop} + a = \mathsf{loop}$ for each program a, hence $+$ models demonic non-determinism according to [36]. By definition of the natural order, we thus obtain $a \leq \mathsf{loop}$ for each a, and therefore $\mathsf{loop} = \top$. Thus recursion in total correctness is modelled by greatest fixpoints with respect to the natural order.

Another consequence of $\mathsf{loop} = \top$ is the law $\top \cdot a = \top$ for all a, which also cannot be proved from the axioms in Section 2. It can equivalently be expressed as $\top \cdot 0 = \top$, and has to be added as the characteristic axiom of total correctness.

Theories of total correctness include weakest preconditions [12], the Unifying Theories of Programming [22], demonic refinement algebra [37] and demonic algebra [5].

3.3 General Correctness

In general correctness, terminating and non-terminating executions are treated independently. It therefore offers a finer distinction than either partial or total correctness [25, 15]. For example, the program $1 + \mathsf{loop}$, which equals 1 in partial correctness and loop in total correctness, is neither 1 nor loop in general correctness; it has both a terminating and a non-terminating execution. The choice operation $+$ models erratic non-determinism according to [36].

However, a price must be paid for this additional precision: the natural order has to be replaced by the more complex Egli-Milner order to model recursion. In fact, the semantics of recursion is given by least fixpoints with respect to this order. The natural order is still used for refinement, as it is in partial and total correctness.

Theories of general correctness include those of [2, 4, 25, 3], the commands of [34, 32] and the prescriptions of [14].

As observed above, $\mathsf{loop} + a \notin \{a, \mathsf{loop}\}$ in general, and therefore $\mathsf{loop} \notin \{0, \top\}$. To obtain a representation for loop, we first recall that $\top \cdot 0 = 0 = \mathsf{loop}$ in partial correctness and $\top \cdot 0 = \top = \mathsf{loop}$ in total correctness. It is therefore manifest to try $\mathsf{loop} = \top \cdot 0$, and this attempt is confirmed by verifying it in concrete models of general correctness such as those of [34, 14, 32].

The element $\top \cdot 0$ occurs in several contexts, such as infinite computations and temporal logic [13, 33, 29]. For general correctness, it is important as the intended least element of the Egli-Milner order, hence we call this element $\mathsf{L} =_{\mathrm{def}} \top \cdot 0$. By itself, this definition does not impose any restrictions; this has to be done by adding further axioms to obtain general correctness.

Such axioms, about L and another element H corresponding to the command havoc of [34], are studied in our previous work [17]. There we propose the following four axioms:

$$\begin{array}{llll}
(\mathsf{L1}) & \ulcorner a\mathsf{L} \leq a\mathsf{L} & (\mathsf{H1}) & a \leq b + \mathsf{L} \wedge a \leq b + \mathsf{H} \Rightarrow a \leq b \\
(\mathsf{L2}) & 1 \leq \ulcorner\mathsf{L} & (\mathsf{H2}) & a \leq a0 + \mathsf{H}
\end{array}$$

Based on these axioms, a theory of general correctness is derived including correctness statements, a correctness calculus, specification constructs, a loop refinement rule, and a representation of least fixpoints with respect to the Egli-Milner order in terms of the natural order. By domain axioms, $(\mathsf{L1})$ is equivalent to $\ulcorner a\mathsf{L} = a\mathsf{L}$ and $(\mathsf{L2})$ is equivalent to $1 = \ulcorner\mathsf{L}$.

Although the above axioms are independent, our previous work does not answer which of them, or yet others, are essential to build a meaningful theory of general correctness. In the remainder of this section, we show that under reasonable assumptions we cannot do away with $(\mathsf{L1})$ and $(\mathsf{L2})$. The topic of Section 4 is to show that these two axioms indeed suffice to derive a useful theory.

We start by defining the relation $\sqsubseteq$ that is planned to become the Egli-Milner order, whence we call it the *Egli-Milner relation*:

$$a \sqsubseteq b \Leftrightarrow_{\mathrm{def}} a \leq b + a0 \wedge b \leq a + \ulcorner(a0)\top \ .$$

This definition is justified by verifying that it instantiates to the Egli-Milner order given in [34, 32, 16]; the calculation is similar to the one in [17]. The second condition $b \leq a + \ulcorner(a0)\top$ is equivalent to $\neg\ulcorner(a0)b \leq a$.

Remark. Intuitively, forming the product $a0$ cuts away all terminating executions of a, hence the term $\ulcorner(a0)$ represents those states from which non-terminating executions of a exist. Its complement $\neg\ulcorner(a0)$ represents the initial states from which termination is guaranteed. By restricting both inequalities of the Egli-Milner relation to these states, we obtain $\neg\ulcorner(a0)a = \neg\ulcorner(a0)b$, which means that a and b must have the same executions in states from which a certainly terminates. The intuition for the remaining states is that b may or may not terminate, but it must have at least the terminating executions of a.

The main result of this section precisely characterises the Egli-Milner order. To state it, we consider the following consequence of (L1):

$$(\mathsf{L3}) \quad \ulcorner(a0)\mathsf{L} \le a$$

It is equivalent to $\ulcorner(a0)\mathsf{L} = a0$ and [17, Lemma 2] shows $(\mathsf{L1}) \wedge (\mathsf{L2}) \Leftrightarrow (\mathsf{L3}) \wedge (\mathsf{L2})$.

Theorem 1. *The relation $\sqsubseteq$ is a partial order if and only if* (L3) *holds. It has the least element* L *if and only if* (L2) *holds. It has no greatest element. The operations $+$ and $\cdot$ are isotone with respect to $\sqsubseteq$.*

Proof. Reflexivity follows immediately from the definition of $\sqsubseteq$.

We show that $\sqsubseteq$ is antisymmetric if and only if (L3) holds. Assume (L3) and $a \sqsubseteq b$ and $b \sqsubseteq a$. Then $b0 \le (a + \ulcorner(a0)\top)0 = a0 + \ulcorner(a0)\mathsf{L} = a0 + a0 = a0$, and symmetrically $a0 \le b0$. Thus $a \le b + a0 \le b + b0 = b$ and symmetrically $b \le a$, hence $a = b$.

For the converse implication assume that $\sqsubseteq$ is antisymmetric. Then (L3) holds if we can show $\ulcorner(a0)\mathsf{L} \sqsubseteq a0$ and $a0 \sqsubseteq \ulcorner(a0)\mathsf{L}$. The latter follows immediately from the definition of $\sqsubseteq$ and the former follows by $\ulcorner(a0)\mathsf{L} = \ulcorner(a0)\mathsf{L}0 \le a0 + \ulcorner(a0)\mathsf{L}0$ and

$$a0 \le \ulcorner(a0)a0 \le \ulcorner(a0)\top 0 = \ulcorner(a0)\mathsf{L} \le \ulcorner(a0)\mathsf{L} + \ulcorner(\ulcorner(a0)\mathsf{L}0)\top$$

using a domain axiom in the first step.

We next show that (L3) implies that $\sqsubseteq$ is transitive, which completes the proof of the first claim. Assume $a \sqsubseteq b$ and $b \sqsubseteq c$. Then $b0 \le a0$ as shown above, hence $a \le b + a0 \le c + b0 + a0 = c + a0$ and $c \le b + \ulcorner(b0)\top \le a + \ulcorner(a0)\top$. But this implies $a \sqsubseteq c$.

For the second claim, consider $\mathsf{L} \sqsubseteq b$. Since $\mathsf{L} \le b + \mathsf{L}0 = b + \mathsf{L}$ always holds, $\mathsf{L} \sqsubseteq b$ is equivalent to $b \le \mathsf{L} + \ulcorner(\mathsf{L}0)\top = \ulcorner\mathsf{L}\mathsf{L} + \ulcorner\mathsf{L}\top = \ulcorner\mathsf{L}\top$. This holds for all b if and only if $\top \le \ulcorner\mathsf{L}\top$. By characterisation of domain, this is equivalent to $1 \le \ulcorner\mathsf{L}$.

For the third claim, assume $0 \sqsubseteq b$ and $1 \sqsubseteq b$. The former implies that $b \le 0 + \ulcorner(0 \cdot 0)\top = \ulcorner 0\top = 0\top = 0$. Thus the latter implies $1 \le b + 1 \cdot 0 = 0 + 0 = 0$, a contradiction.

To see that $+$ and $\cdot$ are isotone, assume $a \sqsubseteq b$. Then $a + c \sqsubseteq b + c$, since $a + c \le b + a0 + c = b + c + (a + c)0$ and $b + c \le a + \ulcorner(a0)\top + c \le a + c + \ulcorner((a+c)0)\top$. Moreover $ca \sqsubseteq cb$, since $ca \le c(b + a0) = cb + ca0$ and

$$cb \le c(a + \ulcorner(a0)\top) = ca + \ulcorner(c\ulcorner(a0))c\ulcorner(a0)\top \le ca + \ulcorner(ca0)\top\ .$$

Finally $ac \sqsubseteq bc$, since $ac \le (b + a0)c = bc + a0 \le bc + ac0$ holds as well as $bc \le (a + \ulcorner(a0)\top)c \le ac + \ulcorner(ac0)\top$. $\square$

This means that axioms (L1) and (L2) are equivalent to the requirement that the Egli-Milner relation is a partial order with least element L. Let us discuss why this is a reasonable assumption. First, it holds in concrete models such as that of [34]. Second, if we do not assume a partial order, it is difficult to define least fixpoints, which are necessary for loops and recursion. Third, a characteristic of general correctness is that the endless loop L is the least fixpoint of the identity function with respect to the Egli-Milner order, hence its least element.

Remark. This leaves open the question, whether the Egli-Milner relation can be defined in another way, which requires less or different axioms. Besides the alternative definition of [17], which is based on (H1) and (H2), we have investigated the following candidates:

- $a \sqsubseteq_1 b \Leftrightarrow_{\mathrm{def}} b \leq a + \ulcorner(a0)\top \wedge a \leq b + \mathsf{L}$
- $a \sqsubseteq_2 b \Leftrightarrow_{\mathrm{def}} b \leq a + \ulcorner(a0)\top \wedge a \leq b + \ulcorner(a0)\mathsf{L}$
- $a \sqsubseteq_3 b \Leftrightarrow_{\mathrm{def}} b \leq a + \ulcorner(a0)\top \wedge a \leq b + \neg\ulcorner(b0)\mathsf{L}$
- $a \sqsubseteq_4 b \Leftrightarrow_{\mathrm{def}} b \leq a + \ulcorner(a0)\top \wedge a \leq b + \neg\ulcorner(b0)\ulcorner(a0)\mathsf{L}$

The relations $\sqsubseteq_{1,2}$ are preorders; they are orders if and only if (L3) holds; they have least element L if and only if (L2) holds. The relations $\sqsubseteq_{3,4}$ are orders; they have least element L if and only if both (L2) and (L3) hold. Assuming (L3), the orders $\sqsubseteq$ and $\sqsubseteq_2$ and $\sqsubseteq_4$ coincide, as do $\sqsubseteq_1$ and $\sqsubseteq_3$.

In all of the above cases, (L2) and (L3) are necessary to obtain a partial order with least element L. We have not found a way to replace the condition $b \leq a + \ulcorner(a0)\top$ to obtain a suitable, yet different relation.

An improvement over previous treatments such as [34, 32] is that we abstract from the concrete definition of commands as pairs of termination and transition information.

An improvement over our previous work [17] is that we no longer require the axioms (H1) and (H2), and hence it is not necessary to introduce the new element H. This is beneficial, since with (H1) we omit an axiom which is a conditional equation, that is, an implication. In the basic case of a bounded domain semiring we thus have an equational axiomatisation, which can be handled significantly better by automated theorem provers [11, 7]. In the presence of loops or recursion, however, the necessary fixpoints are introduced by conditional equations such as those of Kleene algebra.

4 General Correctness and Loops

In this section, we develop the theory of general correctness based on the axioms (L1) and (L2), and hence the Egli-Milner order $\sqsubseteq$ with least element L. The combined treatment with partial and total correctness is resumed in Section 5.

We treat full recursion, loops, correctness statements and correctness calculus, in this sequence. Our previous work [17] proves some of the following results based on the axioms (L1), (L2), (H1) and (H2); the contribution here is to show that (L1) and (L2) suffice. The combined iteration operator is new.

Throughout this section the underlying structure is a bounded domain semiring S satisfying (L1) and (L2). Additional axioms for fixpoints and loops are introduced as required. In Section 4.4 we give an outlook on pre-post specifications which require further axioms.

This section serves a twofold purpose. First, it shows that a useful theory of general correctness can be derived from very basic assumptions. Second, it is a precursor for Section 5 by deriving a semantics of while-loops.

4.1 Recursion

We first derive least fixpoints with respect to the Egli-Milner order $\sqsubseteq$ from fixpoints with respect to the natural order $\leq$. This is interesting, because $\leq$ is much simpler than $\sqsubseteq$ and hence it becomes much easier to compute the semantics of recursive programs. The results in this section require (L3) only.

Let $f : S \to S$ be the characteristic function of the recursion. We assume that f is isotone with respect to $\leq$ and $\sqsubseteq$. Moreover we assume that the least fixpoint μf and the greatest fixpoint νf of f with respect to $\leq$ exist. These assumptions are reasonable, since they are satisfied for typical programming constructs; they certainly apply for the derivation of the while-loop in Section 4.2. The least fixpoint of f with respect to the Egli-Milner order $\sqsubseteq$ is denoted by ξf.

The proof of our first result is very similar to that in [17], except that it accounts for the modified Egli-Milner order. We reproduce it to be self-contained.

Theorem 2. *Let $a \in S$, then $a = \xi f \Leftrightarrow \mu f \leq a \leq \nu f \wedge a \sqsubseteq \mu f \wedge a \sqsubseteq \nu f$.*

Proof. The forward implication is clear since ξf is the least fixpoint with respect to $\sqsubseteq$. For the backward implication, let $\mu f \leq a \leq \nu f$ and $a \sqsubseteq \mu f$ and $a \sqsubseteq \nu f$. By isotony of f we obtain $\mu f = f(\mu f) \leq f(a) \leq f(\nu f) = \nu f$ and $f(a) \sqsubseteq f(\mu f) = \mu f$ and $f(a) \sqsubseteq f(\nu f) = \nu f$. From these facts and the assumptions we obtain:

- $a \sqsubseteq f(a)$ since $a \leq \mu f + a0 \leq f(a) + a0$ and $f(a) \leq \nu f \leq a + \ulcorner(a0)\top$.
- $f(a) \sqsubseteq a$ since $f(a) \leq \mu f + f(a)0 \leq a + f(a)0$ and $a \leq \nu f \leq f(a) + \ulcorner(f(a)0)\top$.

Hence $a = f(a)$ by (L3) and Theorem 1. Let $b \in S$ such that $b = f(b)$, hence $\mu f \leq b \leq \nu f$. Then $a \sqsubseteq b$ since $a \leq \mu f + a0 \leq b + a0$ and $b \leq \nu f \leq a + \ulcorner(a0)\top$. ☐

Its important consequence is an explicit formula for ξf. This requires a new proof due to the modified Egli-Milner order and the limited set of axioms.

Corollary 3. *ξf exists $\Leftrightarrow \nu f \leq \mu f + \ulcorner(\nu f0)\top \Leftrightarrow \xi f = \nu f0 + \mu f$.*

Proof. Assuming $\nu f \leq \mu f + \ulcorner(\nu f0)\top$, we show $\xi f = \nu f0 + \mu f$ by Theorem 2. Since $\mu f \leq \nu f0 + \mu f \leq \nu f$ it suffices to show $\nu f0 + \mu f \sqsubseteq \mu f$ and $\nu f0 + \mu f \sqsubseteq \nu f$. But these follow since $\nu f0 + \mu f = \mu f + (\nu f0 + \mu f)0 \leq \nu f + (\nu f0 + \mu f)0$ and $\mu f \leq \nu f \leq \mu f + \ulcorner(\nu f0)\top = \mu f + \nu f0 + \ulcorner(\nu f0)\top = \nu f0 + \mu f + \ulcorner((\nu f0 + \mu f)0)\top$ using characterisation of domain in the third step.

If $\xi f = \nu f0 + \mu f$, then clearly ξf exists.

Finally assume that ξf exists. By Theorem 2 we obtain $\xi f \leq \mu f + \xi f0$ and $\nu f \leq \xi f + \ulcorner(\xi f0)\top$, hence $\nu f \leq \mu f + \xi f0 + \ulcorner(\xi f0)\top = \mu f + \ulcorner(\xi f0)\top \leq \mu f + \ulcorner(\nu f0)\top$ using characterisation of domain in the second step and $\xi f \leq \nu f$ in the third. ☐

4.2 While-loops

We instantiate the results of Section 4.1 to obtain the semantics of the while-loop. To this end, let $f(x) = ax + b$. Then f is isotone with respect to $\leq$ and, by Theorem 1, also with respect to $\sqsubseteq$. To describe the extremal fixpoints with respect to $\leq$, we now assume that S is also a weak omega algebra. Then $\mu f = a^*b$ and $\nu f = a^\omega + a^*b$.

For the following results, we also need ($\mathsf{L2}$). An equivalent characterisation of ($\mathsf{L2}$) in a weak omega algebra is $a^\omega \leq \ulcorner(a^\omega 0)\top$ for all a. Indeed, instantiating $a = 1$ yields $\top \leq \ulcorner\mathsf{L}\top$ and hence ($\mathsf{L2}$), and the converse implication follows by characterisation of domain from $\ulcorner a^\omega = \ulcorner(a^\omega \ulcorner\mathsf{L}) = \ulcorner(a^\omega \top\mathsf{L}) = \ulcorner(a^\omega \top 0) = \ulcorner(a^\omega 0)$.

By ($\mathsf{L2}$) we thus obtain $\nu f = \mu f + a^\omega \leq \mu f + \ulcorner(a^\omega 0)\top \leq \mu f + \ulcorner(\nu f 0)\top$, hence ξf exists by Corollary 3, and $\xi f = \nu f 0 + \mu f = a^\omega 0 + a^* b$. This prompts us to introduce the notation $x^\circ =_{\text{def}} x^\omega 0 + x^*$ combining infinite and finite iteration appropriately. In different axiomatic settings, we find the combinations $x^\omega + x^*$ called 'strong iteration' [37] and $x^\omega + x^* y$ [6].

As usual in general correctness [2, 34, 32, 16], the semantics of the while-loop is given by $\mathsf{while}\ p\ \mathsf{do}\ a = \xi(\lambda x.pax + \overline{p})$, hence we have established the following result.

Corollary 4. *Let $p \in \text{test}(S)$ and $a \in S$, then $\mathsf{while}\ p\ \mathsf{do}\ a = (pa)^\circ \overline{p}$.*

Remark. The test $\nabla x =_{\text{def}} \ulcorner x^\omega$ represents the initial states of x from which x can be iterated infinitely. In [17] we show that in presence of ($\mathsf{L1}$) and ($\mathsf{L2}$), this complies with the axiomatisation of ∇ given in [8]. In particular, we obtain $x^\omega 0 = \nabla x \mathsf{L}$, and hence $\mathsf{while}\ p\ \mathsf{do}\ a = \nabla(pa)\mathsf{L} + (pa)^*\overline{p}$.

Theorem 1 shows that choice and composition are isotone with respect to $\sqsubseteq$. Hence also the conditional statement $\mathsf{if}\ p\ \mathsf{then}\ a\ \mathsf{else}\ b = pa + \overline{p}b$ is isotone in a and b. We complete this by showing that $\mathsf{while}\ p\ \mathsf{do}\ a$ is isotone in a, which follows from Theorem 6 below. It needs a few general results about our combined iteration operator. They supplement the sliding, unfold and decomposition laws for the star operation and will be useful in Section 5, too.

Lemma 5. *Let a and b be elements of a weak omega algebra. Then*

1. $a(ba)^\omega = (ab)^\omega$.
2. $a(ba)^\circ = (ab)^\circ a$.
3. $a^\circ = 1 + aa^\circ = 1 + a^\circ a$.
4. $(a+b)^\omega = (a^*b)^\omega + (a^*b)^* a^\omega$.
5. $(a+b)^\circ = (a^*b)^\circ a^\circ = (a^\circ b)^\circ a^\circ = a^\circ (ba^\circ)^\circ$.

Proof.

1. $a(ba)^\omega = aba(ba)^\omega$ by omega unfold, hence $a(ba)^\omega \leq (ab)^\omega$ by omega co-induction. By swapping a and b, this implies $(ab)^\omega = ab(ab)^\omega \leq a(ba)^\omega$.
2. $a(ba)^\circ = a((ba)^\omega 0 + (ba)^*) = a(ba)^\omega 0 + a(ba)^* = (ab)^\omega 0 + (ab)^* a = (ab)^\circ a$.
3. $1 + aa^\circ = 1 + a(a^\omega 0 + a^*) = 1 + aa^\omega 0 + aa^* = a^\omega 0 + a^* = a^\circ$. The second equality follows since $aa^\circ = a^\circ a$ by sliding with $b = 1$.

4. The part ($\leq$) follows by omega co-induction from $(a+b)^\omega \leq a^\omega + a^*b(a+b)^\omega$. But this follows from $(a+b)^\omega = (a+b)(a+b)^\omega = a(a+b)^\omega + b(a+b)^\omega$ again by omega co-induction. For the part ($\geq$) we observe

$$(a^*b)^*a^\omega \leq (a^*b)^*a^*(a+b)^\omega = (a+b)^*(a+b)^\omega = (a+b)^\omega \ .$$

The remaining $(a^*b)^\omega \leq (a+b)^\omega$ follows by omega co-induction from

$$\begin{aligned}(a^*b)^\omega &= a^*b(a^*b)^\omega = (1+aa^*)b(a^*b)^\omega = b(a^*b)^\omega + aa^*b(a^*b)^\omega \\ &= b(a^*b)^\omega + a(a^*b)^\omega = (a+b)(a^*b)^\omega \ .\end{aligned}$$

5. The first equality follows since

$$\begin{aligned}(a+b)^\circ &= (a+b)^\omega 0 + (a+b)^* = ((a^*b)^\omega + (a^*b)^*a^\omega)0 + (a^*b)^*a^* \\ &= (a^*b)^\omega 0 + (a^*b)^*(a^\omega 0 + a^*) = (a^*b)^\circ a^\circ \ .\end{aligned}$$

By isotony the second equality reduces to $(a^\circ b)^\circ a^\circ \leq (a^*b)^\circ a^\circ$. But

$$\begin{aligned}(a^\circ b)^\omega 0 &= (a^*b + a^\omega 0)^\omega 0 = (((a^*b)^*a^\omega 0)^\omega + ((a^*b)^*a^\omega 0)^*(a^*b)^\omega)0 \\ &= (a^*b)^*a^\omega 0 + (1+(a^*b)^*a^\omega 0)(a^*b)^\omega 0 = (a^*b)^*a^\omega 0 + (a^*b)^\omega 0 \\ &= (a^*b)^\circ a^\omega 0 \leq (a^*b)^\circ a^\circ\end{aligned}$$

and $(a^\circ b)^* a^\circ \leq (a^*b)^\circ a^\circ$ follows by star induction from

$$a^\circ + a^\circ b(a^*b)^\circ a^\circ = a^\circ + a^\omega 0 + a^*b(a^*b)^\circ a^\circ = (1 + a^*b(a^*b)^\circ)a^\circ = (a^*b)^\circ a^\circ \ .$$

The third equality now follows by sliding. ☐

Theorem 6. *Let $a, b \in S$ such that $a \sqsubseteq b$. Then $a^* \sqsubseteq b^*$ and $a^\circ \sqsubseteq b^\circ$.*

Proof. Assume $a \sqsubseteq b$, hence $a \leq b + a0$ and $b \leq a + \ulcorner(a0)\top$. Then $a^* \sqsubseteq b^*$ amounts to $a^* \leq b^* + a^*0$ and $b^* \leq a^* + \ulcorner(a^*0)\top$. The former follows by star induction from $1 + a(b^* + a^*0) \leq 1 + (b + a0)b^* + aa^*0 = 1 + bb^* + a0 + aa^*0 \leq b^* + a^*0$. The latter follows using star decomposition in

$$\begin{aligned}b^* &\leq (a + \ulcorner(a0)\top)^* = a^*(\ulcorner(a0)\top a^*)^* = a^*(\ulcorner(a0)\top)^* = a^*(1 + \ulcorner(a0)\top(\ulcorner(a0)\top)^*) \\ &= a^* + a^*\ulcorner(a0)\top \leq a^* + \ulcorner(a^*0)\top \ .\end{aligned}$$

The last step holds by characterisation of domain since $\ulcorner(a^*\ulcorner(a0)\top) = \ulcorner(a^*a0) \leq \ulcorner(a^*0)$.

The second claim $a^\circ \sqsubseteq b^\circ$ amounts to $a^\circ \leq b^\circ + a^\circ 0$ and $b^\circ \leq a^\circ + \ulcorner(a^\circ 0)\top$. The former follows since $a^\omega 0 \leq a^\circ 0$ and $a^* \leq b^* + a^*0 \leq b^\circ + a^\circ 0$ as above. The latter follows by replacing * with $^\circ$ in the calculation above, using Lemma 5. ☐

4.3 Correctness Statements and Calculus

Let $a \in S$ represent a program and three tests $p, q, r \in \text{test}(S)$ represent conditions on the state of a. As shown in [17], the following correctness claim is appropriate in a general correctness setting:

$$ra0 \leq 0 \land pa\overline{q} \leq \mathsf{L} \ .$$

In its first part, r describes the initial states from where termination of a has to be guaranteed. In the second part, the precondition p describes the initial states from where the postcondition q is established provided a terminates; a hypothetical transition from p to $\overline{q}$ would have to result in non-termination. Thus claims about terminating and non-terminating executions are distinguished.

The first part $ra0 \leq 0$ is the algebraic equivalent of the Hoare triple $r\,\{a\}\,1$ defined by [28, 32]. It can be derived using existing Hoare calculi, with the additional triple $\neg{}^{\ulcorner}(pa)^{\omega}\,\{\mathsf{while}\ p\ \mathsf{do}\ a\}\,1$. This triple is valid by Corollary 4, since $\neg{}^{\ulcorner}(pa)^{\omega}(pa)^{\circ}\overline{p}0 = \neg{}^{\ulcorner}(pa)^{\omega}(pa)^{\circ}0 = \neg{}^{\ulcorner}(pa)^{\omega}(pa)^{\omega}0 = 0$, using $(pa)^{*}0 \leq (pa)^{\omega}0$.

The second part $pa\overline{q} \leq \mathsf{L}$ has to be treated differently. It is implied by the 'weak correctness' claim $pa = paq$ of [37], but in contrast to our previous treatment, the two are no longer equivalent due to the restricted axioms. Hence we define the 'weaker correctness' claim $p\,[\![a]\!]\,q \Leftrightarrow_{\mathrm{def}} pa\overline{q} \leq \mathsf{L}$. A calculus is given as follows; its correctness is proved in [17].

Theorem 7. *Let $a, b \in S$ and $p, q, r \in \mathrm{test}(S)$. Then*

$$\begin{array}{llll}
p\,[\![0]\!]\,q & p\,[\![\mathsf{L}]\!]\,q & q\,[\![1]\!]\,q & pr\,[\![1]\!]\,q \Rightarrow p\,[\![r]\!]\,q \\
\multicolumn{3}{l}{p\,[\![a]\!]\,q \wedge p\,[\![b]\!]\,q \Rightarrow p\,[\![a+b]\!]\,q} & p\,[\![a]\!]\,q \wedge q\,[\![b]\!]\,r \Rightarrow p\,[\![ab]\!]\,r \\
\multicolumn{3}{l}{rp\,[\![a]\!]\,q \wedge \overline{r}p\,[\![b]\!]\,q \Rightarrow p\,[\![\mathsf{if}\ r\ \mathsf{then}\ a\ \mathsf{else}\ b]\!]\,q} & pq\,[\![a]\!]\,q \Rightarrow q\,[\![\mathsf{while}\ p\ \mathsf{do}\ a]\!]\,\overline{p}q
\end{array}$$

4.4 Pre-post Specifications

We finally discuss the extensions necessary to obtain specifications given by pre- and postconditions. An algebraic treatment in total correctness is given by [37]. In general correctness, we have to account for the termination precondition r, and hence obtain the specification $(r \,|\, p \rightsquigarrow q)$ with three components. As in Section 4.3, the test r describes the initial states from which execution must terminate. Moreover, if the precondition p holds in the initial state, the postcondition q must be established in the final states of terminating executions.

With tests p, q and r, the specification $(r \,|\, p \rightsquigarrow q)$ is axiomatised as the greatest element of S satisfying the general correctness claim:

$$x \leq (r \,|\, p \rightsquigarrow q) \Leftrightarrow rx0 = 0 \wedge px\overline{q} \leq \mathsf{L}\ .$$

We can then recover the element $\mathsf{H} =_{\mathrm{def}} \sup\{\, x \mid x0 = 0 \,\}$ as the particular specification $\mathsf{H} = (1 \,|\, 1 \rightsquigarrow 1)$. While this allows us to prove several properties about H, such as $1 \leq \mathsf{H} = \mathsf{H}^2 = \mathsf{H}^*$ and $\mathsf{HL} = \mathsf{L} \neq \mathsf{H}$, we cannot derive (H1) and (H2). Hence the present axioms are still less restrictive than those of our previous work [17]. Further axioms about $(r \,|\, p \rightsquigarrow q)$, which also give an explicit characterisation of such specifications, are the subject of future work.

To show the usefulness of pre-post specifications, let us refine the specification $(r \,|\, q \rightsquigarrow q\overline{p})$ by the loop $\mathsf{while}\ p\ \mathsf{do}\ a$ as proposed in [16]. Algebraically, this means $\mathsf{while}\ p\ \mathsf{do}\ a \leq (r \,|\, q \rightsquigarrow q\overline{p})$, which is equivalent to $r\,\{\mathsf{while}\ p\ \mathsf{do}\ a\}\,1$ and $q\,[\![\mathsf{while}\ p\ \mathsf{do}\ a]\!]\,q\overline{p}$ by the axiom above. The former follows from $r \leq \neg{}^{\ulcorner}(pa)^{\omega}$ as shown in Section 4.3; this means that pa cannot be iterated infinitely from states satisfying r. The latter follows from $pq\,[\![a]\!]\,q$ by Theorem 7; this is implied by $qpa = qpaq$, meaning that pa preserves the invariant q.

5 While-Programs

We now return to the combined treatment of partial, total and general correctness. The setting in this section is a weak omega algebra with tests S.

We first give a unified semantics of while-programs. Using this semantics, we state and prove the normal form theorem of [27, 35] in our more general setting. This newly establishes that the result holds in general correctness. We finally extend it to while-programs with non-deterministic choice.

5.1 Unified Semantics

We first observe that the semantics of the while-loop derived for general correctness in Corollary 4 is adequate also for partial and total correctness:

– Recall from Section 3.1 that $a \cdot 0 = 0$ for all a in partial correctness. Hence we obtain $(pa)^\circ\overline{p} = (pa)^\omega 0 + (pa)^*\overline{p} = (pa)^*\overline{p} = \mu(\lambda x.pax + \overline{p})$. But this is precisely the semantics of the while-loop in this setting, since recursion is modelled by least fixpoints with respect to $\leq$.
– Recall from Section 3.2 that $\top \cdot 0 = \top$ in total correctness. It follows that $a^\omega 0 = a^\omega \top 0 = a^\omega \top = a^\omega$. Hence we obtain the proper semantics also in this setting by $(pa)^\circ\overline{p} = (pa)^\omega 0 + (pa)^*\overline{p} = (pa)^\omega + (pa)^*\overline{p} = \nu(\lambda x.pax + \overline{p})$, since recursion is modelled by greatest fixpoints with respect to $\leq$.

We can thus define a unified semantics of program constructs, including the while-loop, which is appropriate for partial, total and general correctness:

$$\begin{aligned} a\,;\,b &=_{\text{def}} ab \\ \mathsf{if}\ p\ \mathsf{then}\ a\ \mathsf{else}\ b &=_{\text{def}} pa + \overline{p}b \\ \mathsf{if}\ p\ \mathsf{then}\ a &=_{\text{def}} pa + \overline{p} \\ \mathsf{while}\ p\ \mathsf{do}\ a &=_{\text{def}} (pa)^\circ\overline{p} \end{aligned}$$

This gives the correct semantics in each particular correctness approach. Results proved using this semantics, applying only the axioms of Section 2, hold in partial, total and general correctness. We call programs composed from atomic programs by the above constructs *while-programs.*

Remark. Since the semantics of the while-loop has been derived as a special case of recursion, the question is raised whether the general correctness semantics of full recursion $\xi f = \nu f 0 + \mu f$ obtained in Corollary 3 also suits partial and total correctness.

For partial correctness, this is true, since $\nu f 0 + \mu f = 0 + \mu f = \mu f$ and μf is the appropriate fixpoint operator.

For total correctness the appropriate fixpoint operator is νf, but $\nu f 0 + \mu f \neq \nu f$ in general, as the following counterexample shows. It uses the meet operation inf, which is available in relational models such as [22]. Let $f(x) = \inf\{1, x\}$, hence $\nu f = f(\nu f) \leq 1$. Since $f(0) = 0$ and $f(1) = 1$, we have $\mu f = 0$ and $\nu f = 1$, but $\nu f 0 + \mu f = 1 \cdot 0 + 0 = 0 \neq 1 = \nu f$.

5.2 Normal Form Theorem

A while-program is in *normal form* if it has the form a ; $\mathsf{while}\ p\ \mathsf{do}\ b$ where a and b are while-free. Using this definition, [27] goes on to prove that every while-program can be transformed into normal form; the statement is made more precise below. The given proof is valid in partial correctness only, since it uses both the axiom $\top \cdot 0 = 0$ and the least fixpoint $(pa)^*\overline{p}$ as the semantics of the loop $\mathsf{while}\ p\ \mathsf{do}\ a$.

Making this observation, [35] reproduces the result in total correctness, using the axiom $\top \cdot 0 = \top$ and the greatest fixpoint $(pa)^\omega + (pa)^*\overline{p}$ for the above loop. Actually, the setting is demonic refinement algebra [37], which is interdefinable with weak omega algebra extended by $\top \cdot 0 = \top$ as shown in [24].

Our goal in the following is to prove the normal form result using the unified semantics of Section 5.1 and only axioms of weak omega algebra with tests. This subsumes the theorems of [27, 35] and newly establishes the result for general correctness. The structure of the proof is the same as for partial and total correctness: while-programs are successively transformed by moving inner while-loops to the outside.

We discuss the three kinds of program transformations necessary to perform these steps, and prove their correctness. Before that, let us introduce the necessary tools and formally state the theorem.

For $a \in S$ and $p \in \mathrm{test}(S)$ we say that a *preserves* p if $pa \leq ap$ and $\overline{p}a \leq a\overline{p}$. Observe that $pa \leq ap$ is equivalent to $pa = pap$. Moreover, tests preserve any test. The following two lemmas record general facts related to preservation and the import of tests into iterations. Some of these are known from [26, 29]; we are particularly interested in the properties of our combined iteration $^\circ$.

Lemma 8. *Let a and b be elements of a weak omega algebra such that $ba \leq ab$. Then*

1. $ba^* \leq a^*b$ *and* $b^*a \leq ab^*$.
2. $ba^\omega \leq a^\omega$ *and* $(ab)^\omega \leq a^\omega$.
3. $ba^\circ \leq a^\circ b$.

Proof. Assume $ba \leq ab$.

1. $b + a^*ba \leq b + a^*ab = (1 + a^*a)b \leq a^*b$, hence $ba^* \leq a^*b$ by star induction. $a + bab^* \leq a + abb^* = a(1 + bb^*) \leq ab^*$, hence $b^*a \leq ab^*$ by star induction.
2. $ba^\omega = baa^\omega \leq aba^\omega$, hence $ba^\omega \leq a^\omega$ by omega co-induction. For the second claim we have $(ab)^\omega = a(ba)^\omega \leq a(ab)^\omega$ by Lemma 5, thus $(ab)^\omega \leq a^\omega$ by omega co-induction.
3. $ba^\circ = b(a^\omega 0 + a^*) = ba^\omega 0 + ba^* \leq a^\omega 0 + a^*b = (a^\omega 0 + a^*)b = a^\circ b$. □

Lemma 9. *Let $a \in S$ and $p \in \mathrm{test}(S)$ such that $pa \leq ap$. Then*

1. $pa^* = p(pa)^*$.
2. $pa^\omega = p(pa)^\omega = (pa)^\omega$.
3. $pa^\circ = p(pa)^\circ$.

Proof. Assume $pa \leq ap$.

1. Since $ppa \leq pap$ by the assumption, we have $p(pa)^* \leq (pa)^*p$ by Lemma 8, hence $p + p(pa)^*a \leq p + p(pa)^*pa = p(1 + (pa)^*pa) \leq p(pa)^*$, and therefore $pa^* \leq p(pa)^*$ by star induction. The converse inequality follows immediately.
2. $pa^\omega = paa^\omega = papa^\omega$, hence $pa^\omega \leq (pa)^\omega$ by omega co-induction, and moreover $(pa)^\omega = pa(pa)^\omega \leq paa^\omega = pa^\omega$. Thus $pa^\omega = ppa^\omega = p(pa)^\omega$.
3. $pa^\circ = p(a^\omega 0 + a^*) = pa^\omega 0 + pa^* = p(pa)^\omega 0 + p(pa)^* = p((pa)^\omega 0 + (pa)^*) = p(pa)^\circ$. ☐

For $s \in S$ and $p, q \in \text{test}(S)$ we say that s *assigns* p *to* q if $s = s(pq + \overline{p}\,\overline{q})$. Intuitively, s models a program that assigns the value of p to a new Boolean variable whose value is tested by q. The consequence of using a *new* variable is that programs can be augmented by the assigning subprogram s without essential changes and that q is preserved by components of the original program.

Theorem 10. *Every while-program, suitably augmented with assigning subprograms, is equivalent to a while-program in normal form under certain preservation assumptions.*

The following construction makes explicit where to add assigning subprograms and which preservation assumptions are required.

Remark. Both previous versions of the normal form theorem [27, 35] talk about augmenting with 'subprograms of the form' s ; $(pq + \overline{p}\,\overline{q})$, and [27] adds that s is an 'uninterpreted atomic program symbol'. Let us therefore clarify our understanding of this. Syntactically, with each augmentation a new atomic subprogram is inserted, which we denote by s. It is semantically interpreted as an element $s \in S$ such that $s = s(pq + \overline{p}\,\overline{q})$. The program transformation is an equation in which s is *universally* quantified, over all elements satisfying this equation. For some values of s, such as $s = 0$ or $s = \top \cdot 0$, the claim is trivial; but there are also sensible choices as described above.

The individual program transformations, that move while-loops out of the three kinds of programming constructs, are stated and proved similarly to [27, 35]. However, the different semantics of the while-loop must be taken into account; this is done by Lemmas 5, 8 and 9 about the $^\circ$ operator. Moreover, we take care to use only axioms of weak omega algebra with tests.

The first transformation moves two while-programs in normal form out of a conditional, and hence into normal form. Note that any while-free program a can be brought into normal form a ; while 0 do 1. This can be applied first if one of the branches is while-free as, for example, in the one-armed conditional. A similar remark applies to programs in sequential composition below.

Lemma 11. *Let s assign p to q and let a_1, a_2, b_1, b_2 preserve q. Then*

$$\begin{aligned} & s \text{ ; if } p \text{ then } (a_1 \text{ ; while } r_1 \text{ do } b_1) \text{ else } (a_2 \text{ ; while } r_2 \text{ do } b_2) \\ = {} & s \text{ ; (if } q \text{ then } a_1 \text{ else } a_2) \text{ ; while } qr_1 + \overline{q}r_2 \text{ do (if } q \text{ then } b_1 \text{ else } b_2) \ . \end{aligned}$$

Proof. Since $s = s(pq + \overline{p}\,\overline{q})$, it suffices to show

$$\begin{aligned} &(pq + \overline{p}\,\overline{q})(pa_1(r_1b_1)^\circ\overline{r_1} + \overline{p}a_2(r_2b_2)^\circ\overline{r_2}) \\ &= (pq + \overline{p}\,\overline{q})(qa_1 + \overline{q}a_2)((qr_1 + \overline{q}r_2)(qb_1 + \overline{q}b_2))^\circ\overline{qr_1 + \overline{q}r_2} \ . \end{aligned}$$

The right hand side of this equation is simplified by

- $(pq + \overline{p}\,\overline{q})(qa_1 + \overline{q}a_2) = pqqa_1 + pq\overline{q}a_2 + \overline{p}\,\overline{q}qa_1 + \overline{p}\,\overline{q}\,\overline{q}a_2 = pqa_1 + \overline{p}\,\overline{q}a_2$,
- $(qr_1 + \overline{q}r_2)(qb_1 + \overline{q}b_2) = qr_1qb_1 + qr_1\overline{q}b_2 + \overline{q}r_2qb_1 + \overline{q}r_2\overline{q}b_2 = qr_1b_1 + \overline{q}r_2b_2$,
- $\overline{qr_1 + \overline{q}r_2} = (\overline{q} + \overline{r_1})(q + \overline{r_2}) = \overline{q}\,\overline{r_2} + q\overline{r_1} + \overline{r_1}\,\overline{r_2} = q\overline{r_1} + \overline{q}\,\overline{r_2}$.

Similarly simplifying the left hand side, it suffices to show

$$pqa_1(r_1b_1)^\circ\overline{r_1} + \overline{p}\,\overline{q}a_2(r_2b_2)^\circ\overline{r_2} = (pqa_1 + \overline{p}\,\overline{q}a_2)(qr_1b_1 + \overline{q}r_2b_2)^\circ(q\overline{r_1} + \overline{q}\,\overline{r_2}) \ .$$

But this follows since

$$\begin{aligned} &pqa_1(qr_1b_1 + \overline{q}r_2b_2)^\circ(q\overline{r_1} + \overline{q}\,\overline{r_2}) = pqa_1q(qr_1b_1 + \overline{q}r_2b_2)^\circ(q\overline{r_1} + \overline{q}\,\overline{r_2}) \\ &= pqa_1q(qqr_1b_1 + q\overline{q}r_2b_2)^\circ(q\overline{r_1} + \overline{q}\,\overline{r_2}) = pqa_1q(qr_1b_1)^\circ(q\overline{r_1} + \overline{q}\,\overline{r_2}) \\ &= pqa_1q(r_1b_1)^\circ(q\overline{r_1} + \overline{q}\,\overline{r_2}) = pqa_1q(r_1b_1)^\circ q(q\overline{r_1} + \overline{q}\,\overline{r_2}) \\ &= pqa_1q(r_1b_1)^\circ q\overline{r_1} = pqa_1q(r_1b_1)^\circ\overline{r_1} = pqa_1(r_1b_1)^\circ\overline{r_1} \end{aligned}$$

and similarly $\overline{p}\,\overline{q}a_2(qr_1b_1 + \overline{q}r_2b_2)^\circ(q\overline{r_1} + \overline{q}\,\overline{r_2}) = \overline{p}\,\overline{q}a_2(r_2b_2)^\circ\overline{r_2}$ using Lemmas 8 and 9 because a_1, a_2, b_1, b_2 preserve q. □

The second transformation moves a while-program in normal form out of a while-loop, and hence into normal form by subsequent application of Lemma 11.

Lemma 12.

$$\textsf{while}\ p\ \textsf{do}\ (a\ ;\ \textsf{while}\ q\ \textsf{do}\ b) = \textsf{if}\ p\ \textsf{then}\ (a\ ;\ \textsf{while}\ p + q\ \textsf{do}\ (\textsf{if}\ q\ \textsf{then}\ b\ \textsf{else}\ a)) \ .$$

Proof. The claim follows since

$$\begin{aligned} &pa((p+q)(qb + \overline{q}a))^\circ\overline{p+q} + \overline{p} = pa(qb + \overline{q}pa)^\circ\overline{p}\,\overline{q} + \overline{p} \\ &= pa(qb)^\circ(\overline{q}pa(qb)^\circ)^\circ\overline{p}\,\overline{q} + \overline{p} = (pa(qb)^\circ(\overline{q}pa(qb)^\circ)^\circ\overline{q} + 1)\overline{p} \\ &= (pa(qb)^\circ\overline{q}(pa(qb)^\circ\overline{q})^\circ + 1)\overline{p} = (pa(qb)^\circ\overline{q})^\circ\overline{p} \ , \end{aligned}$$

using Lemma 5 several times. □

The third transformation moves two while-programs in normal form out of a sequential composition, and hence into normal form. Consider the composition a_1 ; (while p_1 do b_1) ; a_2 ; (while p_2 do b_2) of two programs in normal form. We first replace p_1 with a new test q that is preserved by the second program a_2 ; while p_2 do b_2. By Lemma 8 it suffices to assume that a_2 and b_2 preserve q.

Lemma 13. *Let s assign p to q. Then*

$$s\ ;\ (\textsf{while}\ p\ \textsf{do}\ (a\ ;\ s))\ ;\ b = s\ ;\ (\textsf{while}\ q\ \textsf{do}\ (a\ ;\ s))\ ;\ b \ .$$

Proof. Since $s = s(pq + \overline{p}\,\overline{q})$, it suffices to show, using $c = as$,

$$(pq + \overline{p}\,\overline{q})(pc(pq + \overline{p}\,\overline{q}))^\circ \overline{p} = (pq + \overline{p}\,\overline{q})(qc(pq + \overline{p}\,\overline{q}))^\circ \overline{q} \ .$$

But this follows from

$$((pq + \overline{p}\,\overline{q})pc)^\circ (pq + \overline{p}\,\overline{q})\overline{p} = (pqc)^\circ \overline{p}\,\overline{q} = ((pq + \overline{p}\,\overline{q})qc)^\circ (pq + \overline{p}\,\overline{q})\overline{q}$$

by sliding according to Lemma 5. □

The two occurrences of the assigning subprogram s can be absorbed into a_1 and b_1. We thus return to the composition a_1 ; (while p_1 do b_1) ; a_2 ; (while p_2 do b_2) and assume that p_1 is preserved by a_2 ; while p_2 do b_2 without losing generality. By the following lemma, we absorb this program into the first loop, introducing a copy for the case where the first loop is not executed.

Lemma 14. *Let b preserve p. Then*

$$(\mathsf{while}\ p\ \mathsf{do}\ a)\ ;\ b = \mathsf{if}\ \overline{p}\ \mathsf{then}\ b\ \mathsf{else}\ (\mathsf{while}\ p\ \mathsf{do}\ (a\ ;\ \mathsf{if}\ \overline{p}\ \mathsf{then}\ b))\ .$$

Proof. If we can show $p(pa(\overline{p}b + p))^\circ \overline{p} = pa(pa)^\circ \overline{p}b$, the claim holds since

$$\overline{p}b + p(pa(\overline{p}b + p))^\circ \overline{p} = \overline{p}b + pa(pa)^\circ \overline{p}b = (1 + pa(pa)^\circ)\overline{p}b = (pa)^\circ \overline{p}b$$

by Lemma 5. But the missing step follows because

$$\begin{aligned} &p(pa(\overline{p}b + p))^\circ \overline{p} = p(1 + pa(\overline{p}b + p)(pa(\overline{p}b + p))^\circ)\overline{p} \\ &= pa(\overline{p}b + p)(pa(\overline{p}b + p))^\circ \overline{p} = pa((\overline{p}b + p)pa)^\circ (\overline{p}b + p)\overline{p} = pa(\overline{p}bpa + pa)^\circ \overline{p}b\overline{p} \\ &= pa(\overline{p}b0 + pa)^\circ \overline{p}b = pa(pa)^\circ (\overline{p}b0)^\circ \overline{p}b = pa(pa)^\circ \overline{p}b(0\overline{p}b)^\circ = pa(pa)^\circ \overline{p}b\ , \end{aligned}$$

using Lemma 5 several times. □

By applying this transformation, we obtain the program

$$\begin{aligned} &a_1\ ;\ \mathsf{if}\ \overline{p_1}\ \mathsf{then}\ (a_2\ ;\ \mathsf{while}\ p_2\ \mathsf{do}\ b_2) \\ &\qquad \mathsf{else}\ (\mathsf{while}\ p_1\ \mathsf{do}\ (b_1\ ;\ \mathsf{if}\ \overline{p_1}\ \mathsf{then}\ (a_2\ ;\ \mathsf{while}\ p_2\ \mathsf{do}\ b_2)))\ . \end{aligned}$$

Afterwards, we proceed as in [27, 35]. First, the else branch is normalised by moving the inner while-loop outside of the inner conditional and the outer while-loop using Lemmas 11 and 12. Second, the outer conditional is normalised again by Lemma 11. We thus obtain a program in normal form.

By successively applying the transformations of Lemmas 11–14 we can move while-loops from the inside to the outside of the program, until the whole program is in normal form. This proves Theorem 10.

5.3 Non-deterministic Programs

The choice operator $+$ occurs in while-programs only in a restricted form, namely within the conditional if p then a else $b = pa + \overline{p}b$. However, the normal form theorem does not assume that while-programs are deterministic; in fact the atomic subprograms might as well be non-deterministic. This observation helps us to turn an explicit non-deterministic choice into the restricted form of the conditional.

For $t \in S$ and $r \in \text{test}(S)$ we say that t *tosses* r if $t = trt = t\overline{r}t$. Intuitively, t models a program that non-deterministically assigns true or false to a new Boolean variable whose value is tested by r. It is impossible to distinguish which value was assigned between two immediately following tosses.

Programs composed from the constructs allowed for while-programs and the non-deterministic choice

$$a \text{ or } b =_{\text{def}} a + b$$

are called *non-deterministic while-programs*. We can extend Theorem 10 to such programs as follows.

Theorem 15. *Every* non-deterministic *while-program, suitably augmented with assigning* and tossing *subprograms, is equivalent to a while-program in normal form under certain preservation assumptions.*

In view of the transformations of Section 5.2, it remains to give one that deals with the non-deterministic choice between two while-programs in normal form. The following lemma eliminates the choice operation.

Lemma 16. *Let t toss r. Then*

$$t \,;\, (a \text{ or } b) = t \,;\, \text{if } r \text{ then } (t \,;\, a) \text{ else } (t \,;\, b) \;.$$

Proof. Since $t = trt = t\overline{r}t$, the claim follows by $t(a+b) = ta + tb = trta + t\overline{r}tb = t(rta + \overline{r}tb)$. ☐

Applying it to the program $t \,;\, ((a_1 \,;\, \text{while } p_1 \text{ do } b_1) \text{ or } (a_2 \,;\, \text{while } p_2 \text{ do } b_2))$, where the choice is between programs in normal form, we obtain

$$t \,;\, \text{if } r \text{ then } (t \,;\, a_1 \,;\, \text{while } p_1 \text{ do } b_1) \text{ else } (t \,;\, a_2 \,;\, \text{while } p_2 \text{ do } b_2) \;.$$

But this is brought into normal form using Lemma 11. Together with Lemmas 11–14 this proves Theorem 15.

Remark. We briefly discuss tossing elements. To this end, let t toss r. Then $tt = t(r + \overline{r})t = trt + t\overline{r}t = t + t = t$. To simplify the transformation above, the value of r is erased by another toss immediately after the conditional choice is made. Hence there is no way to determine the outcome of the first toss within the programs a and b of Lemma 16.

6 Conclusion

This work makes a point for using less axioms. It shows that many results of general correctness can be derived in a very basic setting, despite the complexity caused by the Egli-Milner order and the finer termination information. It also shows that by omitting characteristic axioms, a unified treatment of partial, total and general correctness is possible, with a common semantics of programs and common proofs of program transformations.

Future investigations concern axioms for pre-post specifications, refinement, and further operators for general correctness [16], as well as applications in the area of hybrid systems [23]. We also consider program transformations between symmetric linear and tail recursion, known from a total correctness setting [20].

Acknowledgement. I thank all anonymous referees for their valuable remarks and helpful suggestions.

References

1. C. J. Aarts. Galois connections presented calculationally. Master's thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1992.
2. J. W. de Bakker. Semantics and termination of nondeterministic recursive programs. In S. Michaelson and R. Milner, editors, *Automata, Languages and Programming: Third International Colloquium*, pages 435–477. Edinburgh University Press, 1976.
3. R. Berghammer and H. Zierer. Relational algebraic semantics of deterministic and nondeterministic programs. *Theoretical Computer Science*, 43:123–147, 1986.
4. 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 *LNCS*, pages 553–592. Springer-Verlag, 1979.
5. 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 *LNCS*, pages 120–134. Springer-Verlag, 2006.
6. E. Cohen. Separation and reduction. In R. Backhouse and J. N. Oliveira, editors, *Mathematics of Program Construction*, volume 1837 of *LNCS*, pages 45–59. Springer-Verlag, 2000.
7. H.-H. Dang and P. Höfner. First-order theorem prover evaluation w.r.t. relation- and Kleene algebra. In R. Berghammer, B. Möller, and G. Struth, editors, *Relations and Kleene Algebra in Computer Science: PhD Programme at RelMiCS10/AKA5*, Report 2008-04, pages 48–52. Institut für Informatik, Universität Augsburg, April 2008.
8. J. Desharnais, B. Möller, and G. Struth. Algebraic notions of termination. Report 2006-23, Institut für Informatik, Universität Augsburg, October 2006.
9. J. Desharnais, B. Möller, and G. Struth. Kleene algebra with domain. *ACM Transactions on Computational Logic*, 7(4):798–833, October 2006.
10. J. Desharnais and G. Struth. Domain axioms for a family of near-semirings. In J. Meseguer and G. Roşu, editors, *Algebraic Methodology and Software Technology*, volume 5140 of *LNCS*, pages 330–345. Springer-Verlag, 2008.
11. J. Desharnais and G. Struth. Modal semirings revisited. In P. Audebaud and C. Paulin-Mohring, editors, *Mathematics of Program Construction*, volume 5133 of *LNCS*, pages 360–387. Springer-Verlag, 2008.
12. E. W. Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

13. R. M. Dijkstra. Computation calculus bridging a formalization gap. *Science of Computer Programming*, 37(1–3):3–36, May 2000.
14. 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.
15. S. Dunne and A. Galloway. Lifting general correctness into partial correctness is *ok*. In J. Davies and J. Gibbons, editors, *Integrated Formal Methods*, volume 4591 of *LNCS*, pages 215–232. Springer-Verlag, 2007.
16. S. Dunne, I. Hayes, and A. Galloway. Reasoning about loops in total and general correctness. In A. Butterfield, editor, *Second International Symposium on Unifying Theories of Programming*, volume 5713 of *LNCS*. Springer-Verlag, to appear.
17. 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 *LNCS*, pages 150–165. Springer-Verlag, 2009.
18. W. Guttmann. Lazy UTP. In A. Butterfield, editor, *Second International Symposium on Unifying Theories of Programming*, volume 5713 of *LNCS*. Springer-Verlag, to appear.
19. W. Guttmann and B. Möller. Modal design algebra. In S. Dunne and W. Stoddart, editors, *Unifying Theories of Programming*, volume 4010 of *LNCS*, pages 236–256. Springer-Verlag, 2006.
20. W. Guttmann and B. Möller. Normal design algebra. *Journal of Logic and Algebraic Programming*, 79(2):144–173, February 2010.
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. P. Höfner and B. Möller. An algebra of hybrid systems. *Journal of Logic and Algebraic Programming*, 78(2):74–97, January 2009.
24. P. Höfner, B. Möller, and K. Solin. Omega algebra, demonic refinement algebra and commands. In R. Schmidt, editor, *Relations and Kleene Algebra in Computer Science*, volume 4136 of *LNCS*, pages 222–234. Springer-Verlag, 2006.
25. D. Jacobs and D. Gries. General correctness: A unification of partial and total correctness. *Acta Informatica*, 22(1):67–83, April 1985.
26. D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, May 1994.
27. D. Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.
28. D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Transactions on Computational Logic*, 1(1):60–76, July 2000.
29. B. Möller. Lazy Kleene algebra. In D. Kozen, editor, *Mathematics of Program Construction*, volume 3125 of *LNCS*, pages 252–273. Springer-Verlag, 2004.
30. B. Möller. The linear algebra of UTP. In T. Uustalu, editor, *Mathematics of Program Construction*, volume 4014 of *LNCS*, pages 338–358. Springer-Verlag, 2006.
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 *LNCS*, pages 200–211. Springer-Verlag, 2006.

33. B. C. Moszkowski. A complete axiomatization of Interval Temporal Logic with infinite time. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, pages 241–252. IEEE, 2000.
34. G. Nelson. A generalization of Dijkstra's calculus. *ACM Transactions on Programming Languages and Systems*, 11(4):517–561, October 1989.
35. K. Solin. A while program normal form theorem in total correctness. In R. Berghammer, A. M. Jaoua, and B. Möller, editors, *Relations and Kleene Algebra in Computer Science*, volume 5827 of *LNCS*, pages 322–336. Springer-Verlag, 2009.
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.