Science of Computer Programming 74 (2009) 654–668

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

# Enabledness and termination in refinement algebra

Kim Solin *, Joakim von Wright

*Dept. of IT, Åbo Akademi, Joukahainengatan 3-5, FIN-20520 Åbo, Finland*

ARTICLE INFO

*Article history:*
Received 13 February 2007
Received in revised form 30 October 2007
Accepted 28 November 2007
Available online 10 February 2009

*Keywords:*
Refinement algebra
Enabledness
Termination
Action systems

ABSTRACT

Refinement algebras are abstract algebras for reasoning about programs in a total correctness framework. We extend a reduct of von Wright's demonic refinement algebra with two operators for modelling enabledness and termination of programs. We show how the operators can be used for expressing relations between programs and apply the algebra to reasoning about action systems.



## 1. Introduction

Abstract algebra is both mathematically solid and allows for transparent and simple reasoning. This means that abstract-algebraic formulations of classical frameworks often make reasoning easier and more perspicuous, see for example [19,9, 25]. Moreover, when mechanisation and automation are concerned, the level of abstraction provided by abstract algebra seems to avail [1,15]. The focus of the present paper, refinement algebras, are abstract algebras for reasoning about program refinement [28,24,20]. Applications of the refinement algebras include reasoning about data refinement and program transformation, and there is also support for automation [16].

In this paper we introduce a reduct of von Wright's demonic refinement algebra [27]. It differs from previous abstract-algebraic approaches to total correctness reasoning [27,28,9] in that it has only one iteration operator: strong iteration. In program intuition, strong iteration of a statement *either* terminates *or* goes on infinitely. Along the lines of von Wright in [27, 28], we extend the algebra with guards and assertions. Guards should be thought of as programs that check if some predicate holds, skip if that is the case, and otherwise bring about a miracle. Assertions are similar, but instead of performing a miracle when the predicate does not hold, they abort. That is to say, an assertion that is executed in a state where the predicate does not hold establishes no postcondition. In Floyd's terminology, our guards are called assumptions [13].

As the main contribution of this paper, we extend the refinement algebra with two new operators. The first one maps elements in the carrier set to the subset of guards. The intuition is that the operator applied to a program returns a guard that skips in those states in which the program is enabled (that is, the program will not block). This operator is axiomatised in almost the same way as the domain operator in [10]. The second operator maps a program to an assertion that characterises the set of states in which termination is guaranteed (that is, the program will not run forever). Different relations between programs, such as exclusion and a program inversion condition, can be expressed using the new operators. Moreover, with the aid of the enabledness operator, we can encode action systems [3–5] into the refinement algebra and use it for proving correctness of action-system transformations.

Five papers stand out in the lineage of this paper. Kozen's axiomatisation of Kleene algebra and his introduction of tests into the algebra has been a very significant inspiration for us [18,19]. The first abstract algebra that was genuinely an algebra

* Corresponding address: Department of Philosophy, Uppsala University, Box 627, SE-751 26 Uppsala, Sweden.
*E-mail address:* kim.solin@filosofi.uu.se (K. Solin).

0167-6423/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2007.11.004

for total correctness was von Wright's demonic refinement algebra [27], which rests upon previous work on algebraic program reasoning by Back and von Wright [7]. Desharnais, Möller and Struth extended Kleene algebra with a domain operator [10] and successfully applied it to reasoning about different structures. In a slightly different guise, the domain operator appears again in the present paper. The paper at hand is a revision and an extension of an earlier conference paper [26] that was invited to this selection, and continues previous work published in this journal [28].

The paper is organised as follows. We begin by presenting a refinement algebra and extend it with guards and assertions. Then we introduce the new operators, investigate their basic algebraic properties and apply them. We end with some concluding remarks and an outlook on future work. Intertwined with the abstract-algebraic development, we provide a predicate-transformer model for the concepts introduced.

The purpose of this paper is to introduce the enabledness and the termination operators, settle their basic properties and lay a first ground for more elaborate investigations and applications. Our intended readers are those interested in abstract-algebraic reasoning in computer science and those familiar with the research on program refinement. Persons working with action systems might also find the paper worth reading.

## 2. Refinement algebra

By a *demonic refinement algebra* (dRA) we shall in this paper understand a structure over the signature

$$(\sqcap, ;, {}^{\omega}, \top, 1)$$

that satisfies the identities (; is left implicit)

$$x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z, \tag{1}$$
$$x \sqcap y = y \sqcap x, \tag{2}$$
$$x \sqcap \top = x, \tag{3}$$
$$x \sqcap x = x, \tag{4}$$
$$x(yz) = (xy)z, \tag{5}$$
$$1x = x, \tag{6}$$
$$x1 = x, \tag{7}$$
$$\top x = \top, \tag{8}$$
$$x(y \sqcap z) = xy \sqcap xz, \tag{9}$$
$$(x \sqcap y)z = xz \sqcap yz \text{ and} \tag{10}$$
$$x^{\omega} = xx^{\omega} \sqcap 1, \tag{11}$$

and the equational implication

$$xz \sqcap y \sqcap z = xz \sqcap y \;\Rightarrow\; x^{\omega}y \sqcap z = x^{\omega}y.$$

When $\sqsubseteq$ is defined as

$$x \sqsubseteq y \Leftrightarrow x \sqcap y = x \tag{12}$$

the equational implication can be written as

$$xz \sqcap y \sqsubseteq z \;\Rightarrow\; x^{\omega}y \sqsubseteq z \tag{13}$$

and (3) as $x \sqsubseteq \top$. Note that $\sqsubseteq$ is a partial order with $\top$ as its greatest element.

Our axiomatisation is similar to Kozen's Kleene algebra [18]. The difference is that there is no right annihilation axiom, so $x\top = \top$ does not hold in general, that $^*$ is replaced by $^{\omega}$ — and there is only one induction axiom. The operator $^{\omega}$ is different from the iteration operator in Cohen's $\omega$-algebra [9]. Cohen's infinite iteration operator should be interpreted as an infinite repetition of a program statement, whereas our $^{\omega}$ should be seen as a repetition of a program statement that *either* terminates *or* goes on infinitely. Höfner, Möller and Solin have shown that under certain conditions imposed on the omega algebra, Cohen's infinite iteration operator applied to an element $x$ is equal to $x^{\omega}\top$ [17]. The star operator of Kleene algebra cannot be defined in terms of the other operators of dRA.

The reason for not having a right annihilation axiom is that we want to reason about nontermination, we want a total correctness framework. Right annihilation would prevent that (this is elaborated further below, and a semantical clarification is given). In $\omega$-algebra right annihilation holds, which renders it an algebra for partial correctness. The intention to reason about total correctness also motivates the restriction of the signature to one iteration operator. The demonic refinement algebra by von Wright in [27,28] has two related iteration operators, one equal to our $^{\omega}$ and the other equal to $^*$ in Kleene algebra. The intuition for $x^*$ is a terminating repetition of a program statement $x$ (assuming that $x$ is terminating). Since total correctness is what we are actually interested in and the strong iteration operator captures an iteration that will either terminate or go on infinitely we can here exclude $^*$ to get a more comprehensible framework.

Elements of the carrier set can be seen as program statements. The operator ; is sequential composition and $\sqcap$ is demonic choice. The demonic choice operator $\sqcap$ applied to two programs, $x \sqcap y$, is to be seen as a choice between $x$ and $y$ made by a demon. That the choice is made by a demon means that we have no influence over it and that it can be done in the, for us,

most undesirable way: striving to abortion. We will extensively use this way of looking at demonic choice in the sequel. The iteration $x^\omega$ is, as mentioned earlier, thought of as a terminating *or* infinitely repeating execution of a program statement $x$. The order $\sqsubseteq$ is refinement; $x \sqsubseteq y$ means that $y$ establishes anything that $x$ does and possibly more. Finally, $\top$ is interpreted as magic, a nonimplementable program statement that establishes *any* postcondition, and 1 is skip. Below, a semantical justification for this intuition is given in terms of predicate transformers. Indeed, we show how the operators and constants can be given interpretations in terms of predicate transformers so that the set of conjunctive predicate transformers forms a dRA.

We define a syntactic constant $\bot$ with the intuition that it stands for an always nonterminating program, an abort statement [27]:

$$\bot = 1^\omega.$$

It is easily seen that $\bot$ is a least element

$$\bot \sqsubseteq x \tag{14}$$

and that it is a left annihilator

$$\bot x = \bot. \tag{15}$$

The axiomatic reason for excluding $x\top = \top$ is now apparent: if $x\top = \top$ would hold, we would have

$$\bot = \bot\top = \top$$

and, then, since $\bot$ is a least element we would only have a one-point model.

Many properties of the Kleene-algebra $^*$ have analogies for $^\omega$. For example, leapfrog,

$$x(yx)^\omega = (xy)^\omega x, \tag{16}$$

decomposition,

$$(x \sqcap y)^\omega = x^\omega (yx^\omega)^\omega, \tag{17}$$

and outer inheritance of semi-commutativity

$$yx \sqsubseteq xz \Rightarrow y^\omega x \sqsubseteq xz^\omega \tag{18}$$

hold. However, there are differences such as the fact that the inner inheritance of semi-commutativity

$$xz \sqsubseteq yx \Rightarrow xz^\omega \sqsubseteq y^\omega x \tag{19}$$

does *not* hold in general (take $y = z = 1$ and $x = \top$) [27].

### 2.1. Predicate transformers as a model

A predicate transformer [12] is a function $S : \wp(\Sigma) \to \wp(\Sigma)$, where $\Sigma$ is any set. Let $p, q \in \wp(\Sigma)$. If a predicate transformer $S$ satisfies

$$p \subseteq q \Rightarrow S.p \subseteq S.q$$

– where the dot denotes function application – it is *isotone* (or *monotone*) and if for a nonempty $I$ it satisfies

$$S.\left(\bigcap_{i\in I} q_i\right) = \bigcap_{i\in I} S.q_i$$

it is *conjunctive*; it is *universally conjunctive* if $S$ is conjunctive and $S.\Sigma = \Sigma$. Universal conjunctivity implies conjunctivity, and conjunctivity implies isotony.

Programs can be modelled by predicate transformers according to a weakest precondition semantics [12]: $S.q$ denotes the set of states from which the execution of $S$ is bound to terminate in $q$. Universally conjunctive predicate transformers cannot properly model nontermination. To see this, suppose that $S$ is an always nonterminating program, that is,

$$(\forall q \in \wp(\Sigma) \bullet S.q = \emptyset). \tag{20}$$

Now, if $S$ is universally conjunctive, then $S.\Sigma = \Sigma$ so clearly (20) does not hold.

There are three distinguished predicate transformers

$$\begin{aligned} \mathsf{abort} &= (\lambda q \bullet \emptyset),\\ \mathsf{magic} &= (\lambda q \bullet \Sigma) \text{ and}\\ \mathsf{skip} &= (\lambda q \bullet q), \end{aligned}$$

and a predicate transformer $S$ *is refined by* a predicate transformer $T$, written $S \sqsubseteq T$, if

$$(\forall q \in \wp(\Sigma) \bullet S.q \subseteq T.q).$$

This paper deals with three operations on predicate transformers [2,6,22] defined by

$$(S; T).q = S.(T.q), \tag{21}$$

$$(S \sqcap T).q = S.q \cap T.q \text{ and} \tag{22}$$

$$S^{\omega} = \mu.(\lambda X \bullet S; X \sqcap \mathsf{skip}), \tag{23}$$

where $\mu$ denotes the least fixpoint with respect to $\sqsubseteq$. Taking the least fixpoint, and not the greatest, corresponds to identifying infinite unfolding with abort [6].

Let $\mathsf{Ctran}_{\Sigma}$ be the set of conjunctive predicate transformers over a set $\Sigma$. Then it can be verified that

$$(\mathsf{Ctran}_{\Sigma}, \sqcap, ;, ^{\omega}, \mathsf{magic}, \mathsf{skip})$$

is a dRA [27]. It is also clear that abort models $\bot$, since it can be shown that $\mathsf{skip}^{\omega} = \mathsf{abort}$ [6].

We can now give a semantical justification for not having a right annihilation axiom, $x\top = \top$. If we would have right annihilation, then for any predicate transformer $S$ and any $q \in \wp(\Sigma)$

$$S.\Sigma = S.(\mathsf{magic}.q) = (S; \mathsf{magic}).q = \mathsf{magic}.q = \Sigma,$$

so our predicate-transformer model would be universally conjunctive. As noted above, universally conjunctive predicate transformers cannot model nontermination, that is, they do not facilitate our goal: total correctness reasoning.

## 3. Guards and assertions

An element $g$ of the carrier set that has a complement $\bar{g}$ satisfying

$$g\bar{g} = \bar{g}g = \top \quad \text{and} \quad g \sqcap \bar{g} = 1 \tag{24}$$

is called a *guard*. It can be established that the *set of guards* forms a Boolean algebra over $(\sqcap, ;, \bar{}, 1, \top)$, where $\sqcap$ is meet, ; is join, $\bar{}$ is complement, 1 is the least element, and $\top$ is the greatest element. Intuitively, guards are statements that check if a predicate holds and, if so, skip, otherwise do magic. The first guard axiom says that either a predicate or its negation holds, so a sequential composition of a guard and its complement is always a miracle. The second guard axiom says that a demon will always be able to skip when choosing between a guard or the guard's complement.

Every guard is defined to have a corresponding *assertion*

$$g^{\circ} = \bar{g}\bot \sqcap 1. \tag{25}$$

This means that $^{\circ}$ is a mapping from guards to a subset of the carrier set, the *set of assertions*. Assertions are similar to guards, but abort if the predicate does not hold. If the predicate does not hold, then a demon would choose the left hand side of the demonic choice and the negated guard would skip and the whole program abort (which is what a demon wants). If, on the other hand, the predicate holds, then a demon would choose the right-hand side, since otherwise the negated guard would do magic and the demon could then no longer establish abortion. It is easy to show that assertions have the properties

$$(g_1g_2)^{\circ} = g_1^{\circ}g_2^{\circ} = g_1^{\circ} \sqcap g_2^{\circ}, \qquad g^{\circ}\bar{g}^{\circ} = \bot, \qquad g^{\circ} = g^{\circ}\top \sqcap 1 \quad \text{and} \quad g^{\circ}g^{\circ} = g^{\circ}, \tag{26}$$

and that

$$g^{\circ} \sqsubseteq 1 \sqsubseteq g \tag{27}$$

holds for any assertion and any guard [27]. Note that to be able to define assertions explicitly by a set of axioms, it seems we would need an angelic-choice operator [24]. That we do not have angelic choice thus means that guards and assertions are not fully dual.

Assertions corresponding to guards could have been defined implicitly by means of Galois connections: one part of a Galois connection is *uniquely* defined by the other and each of the Galois connections

$$g^{\circ}x \sqsubseteq y \;\Leftrightarrow\; x \sqsubseteq gy \quad \text{and} \quad xg \sqsubseteq y \;\Leftrightarrow\; x \sqsubseteq yg^{\circ} \tag{28}$$

is satisfied by $g^{\circ} = \bar{g}\bot \sqcap 1$ [27].

### 3.1. The predicate-transformer model

Consider the function $[\cdot] : \wp(\Sigma) \to (\wp(\Sigma) \to \wp(\Sigma))$ such that when $p, q \in \wp(\Sigma)$

$$[p].q = \neg p \cup q,$$

where $\neg$ is set complement. These predicate transformers are called *guards*. There is also a dual, an *assertion* and it is defined by

$$\{p\}.q = p \cap q.$$

Guards as well as assertions are conjunctive. Complement $\bar{}$ is defined on guards and assertions by $\overline{[p]} = [\neg p]$ and $\overline{\{p\}} = \{\neg p\}$. It is easy to see that the guards in the predicate-transformer sense satisfy the abstract-algebraic guard axioms. For example, let $[p]$ be any guard and $q \in \wp(\Sigma)$. Then

$$([p] \sqcap \overline{[p]}).q = ([p] \sqcap [\neg p]).q = (\neg p \cup q) \cap (\neg(\neg p) \cup q) = q = \mathsf{skip}.q.$$

The other axioms are verified similarly. This means – by the fact that the guard axioms together with the other axioms of dRA imply the axioms of Boolean algebra, and the easily-proved facts that guards in the predicate-transformer sense are conjunctive and closed under $\sqcap$ and ; – that

$$(\mathsf{Grd}_\Sigma, \sqcap, ;, \bar{}, \mathsf{skip}, \mathsf{magic})$$

is a Boolean algebra, where $\sqcap$ is meet, ; is join, and $\bar{}$ is complement, and $\mathsf{Grd}_\Sigma$ is the set of (predicate-transformer) guards over a state space $\Sigma$. The derivation

$$([\neg p]; \mathsf{abort} \sqcap \mathsf{skip}).q = (\neg(\neg p) \cup \emptyset) \cap q = p \cap q = \{p\}.q,$$

where $p, q \in \wp(\Sigma)$, shows that assertions in the predicate-transformer sense are a model for assertions in the abstract-algebraic sense.

### 3.2. Basic properties

The following propositions summarise some important properties of guards and assertions. First we repeat a proposition reported by von Wright [27]. Intuitively, the first part says that total correctness of $x$ with respect to the precondition denoted by the guard $g_1$ and the postcondition denoted by the guard $g_2$ implies weak correctness (that is, partial correctness in a total correctness framework) with the same parameters. The second statement says that the two characterisations of weak correctness are equivalent. In this paper, these properties will serve primarily as technical tools when proving transformation rules and not for analysing precondition–postcondition properties.

**Proposition 3.1.** *Let $x$ be an element in the carrier set of a* dRA *and let $g_1$ and $g_2$ be any guards in the same set. Then*

$$\top = g_1 x \bar{g_2} \Rightarrow g_1 x g_2 = g_1 x \quad \textit{and} \quad g_1 x g_2 = g_1 x \Leftrightarrow x g_2 \sqsubseteq g_1 x \tag{29}$$

*hold.*

The following two propositions, like the one above, will be useful for the applications in Section 5. The first one can be read as a generalisation of the fact that if $\bar{g}$ does magic, then $g$ skips (note that the refinements on both sides of the implication can be equivalently replaced by equalities, by $\top$ being a greatest element and property (27), respectively). The second one says the same for assertions.

**Proposition 3.2.** *Let $x$ be an element in the carrier set of a* dRA *and let $g$ be any guard in the same set. Then*

$$\top \sqsubseteq \bar{g} x \quad \Leftrightarrow g x \sqsubseteq x \quad \textit{and} \tag{30}$$

$$\bar{g}^\circ x \sqsubseteq \bot \;\Leftrightarrow x \sqsubseteq g^\circ x \tag{31}$$

*hold.*

**Proof.** First,

$$\begin{array}{ll} & g x \sqsubseteq x \\ \Rightarrow & \{\text{isotony}\} \\ & \bar{g} g x \sqsubseteq \bar{g} x \\ \Leftrightarrow & \{\text{basic guard property (24)}\} \\ & \top x \sqsubseteq \bar{g} x \\ \Leftrightarrow & \{\text{axiom (8)}\} \\ & \top \sqsubseteq \bar{g} x \end{array}$$

and second

$$\begin{array}{ll} & \top \sqsubseteq \bar{g} x \\ \Rightarrow & \{\text{isotony}\} \\ & \top \sqcap g x \sqsubseteq \bar{g} x \sqcap g x \\ \Leftrightarrow & \{\text{axioms (2), (3), (10)}\} \\ & g x \sqsubseteq (\bar{g} \sqcap g) x \\ \Leftrightarrow & \{\text{basic guard property (24)}\} \\ & g x \sqsubseteq 1 x \\ \Leftrightarrow & \{\text{axiom (6)}\} \\ & g x \sqsubseteq x. \end{array}$$

This establishes (30). Then, for one direction of (31) calculate

$$\begin{array}{ll} & x \sqsubseteq g^\circ x \\ \Rightarrow & \{\text{isotony}\} \\ & \bar{g}^\circ x \sqsubseteq \bar{g}^\circ g^\circ x \\ \Leftrightarrow & \{\text{basic assertion property (26)}\} \\ & \bar{g}^\circ x \sqsubseteq \bot x \\ \Leftrightarrow & \{\bot \text{ left annihilator (15)}\} \\ & \bar{g}^\circ x \sqsubseteq \bot. \end{array}$$

For the other direction, first note that the left-hand side is equivalent to

$$g\bot \sqcap x \sqsubseteq \bot$$

by (25), (10) and (15). Now assume that this holds. Then

$$\begin{array}{ll} & x \\ \sqsubseteq & \{\text{property (27)}\} \\ & \bar{g}x \\ = & \{\text{axiom (3)}\} \\ & (\top \sqcap \bar{g})x \\ = & \{\text{axiom (8)}\} \\ & (\top\bot \sqcap \bar{g})x \\ = & \{\text{basic guard property (24)}\} \\ & (\bar{g}g\bot \sqcap \bar{g})x \\ = & \{\text{axiom (9)}\} \\ & \bar{g}(g\bot \sqcap 1)x \\ = & \{\text{axiom (10)}\} \\ & \bar{g}(g\bot x \sqcap x) \\ = & \{\bot \text{ left annihilator (15)}\} \\ & \bar{g}(g\bot \sqcap x) \\ \sqsubseteq & \{\text{assumption and isotony}\} \\ & \bar{g}\bot. \end{array}$$

This proves the claim, since $x \sqsubseteq \bar{g}\bot \Leftrightarrow x \sqsubseteq \bar{g}\bot \sqcap x \Leftrightarrow x \sqsubseteq g^\circ x$. □

The first part of the next proposition is similar to the outer version of the second part of (29) and is in fact by (30) an alternative total correctness characterisation (it matches the total correctness condition of (29) when all guards are negated). The second part states a "dual" for assertions.

**Proposition 3.3.** *Let $x$ be an element in the carrier set of a* dRA *and let $g_1$ and $g_2$ be any guards in the same set. Then*

$$g_2xg_1 = xg_1 \quad \Leftrightarrow g_2x \sqsubseteq xg_1 \quad \textit{and} \tag{32}$$

$$xg_1^\circ = g_2^\circ xg_1^\circ \quad \Leftrightarrow xg_1^\circ \sqsubseteq g_2^\circ x \tag{33}$$

*hold.*

**Proof.** Assume $g_2xg_1 = xg_1$. Since $1 \sqsubseteq g$ for any guard $g$, $g_2x \sqsubseteq g_2xg_1 = xg_1$. Conversely, assume that $g_2x \sqsubseteq xg_1$. Then $g_2xg_1 \sqsubseteq xg_1g_1 = xg_1$. The other direction follow directly from the fact that $1 \sqsubseteq g$ for any guard $g$. The case for assertions is proved in a similar fashion. □

## 4. Enabledness and termination

In this section we introduce two new operators, the enabledness operator and the termination operator, and investigate their basic properties. These operators are fundamental in the predicate-transformer approach to program semantics [6], and since predicate transformers are our inspiring model, the operators are likely to be of fundamental importance also in the abstract-algebraic framework. The investigation of their theoretical importance in the abstract-algebraic approach is, however, left for future work; as announced, we in this paper concentrate on their basic properties and light applications.

### 4.1. Enabledness

Let the *enabledness* operator $\epsilon$ be a unary operator on a dRA which maps an element of the carrier set to a guard and that satisfies the axioms

$$\epsilon xx = x, \tag{34}$$

$$g \sqsubseteq \epsilon(gx), \tag{35}$$

$$\epsilon(xy) = \epsilon(x\epsilon y) \text{ and} \tag{36}$$

$$\epsilon x\bot = x\bot. \tag{37}$$

We will call a demonic refinement algebra with an enabledness operator a dRAe.

We intend enabledness to bind stronger than the other operators, so for example $\epsilon xx$ is $(\epsilon x)x$. The intuition behind $\epsilon$ is that it maps any program to a guard that skips in those states in which the program is enabled, that is, in those states from which the program will not terminate miraculously. Axiom (34), for example, says that a program $x$ equals a program that first checks if $x$ is enabled and then executes $x$. The enabledness operator allows us to express implicit guards, which is important when modelling, for example, action systems (cf. Section 5).

The first three axioms of enabledness are essentially the domain axioms of Desharnais, Möller and Struth's Kleene algebra with domain (KAD) [10]. This means that many properties proved for the domain operator in Kleene algebra with domain will also hold for $\epsilon$ in our algebra, but not necessarily all due to the lack of right annihilation. Möller has shown what can be recovered of KAD when the right annihilation axiom and right distributivity (9) are dropped, but right isotony of ; is retained [21]. However, the strong iteration operator is different from the Kleene star of KAD and we retain right distributivity, so our framework is not fully symmetric with Möller's. The fourth axiom is independent of the first three as shown by Desharnais, Möller and Struth in a related structure [11]. It plays an important rôle in the proof of action-system decomposition (Section 5).

As shown by Möller [21], the first two axioms of $\epsilon$ can be replaced by the equivalence

$$gx \sqsubseteq x \Leftrightarrow g \sqsubseteq \epsilon x \tag{38}$$

and, moreover, the properties

$$\epsilon(x \sqcap y) = \epsilon x \sqcap \epsilon y \quad \text{and} \tag{39}$$

$$x \sqsubseteq y \Rightarrow \epsilon x \sqsubseteq \epsilon y \tag{40}$$

hold. These are proved by reusing and slightly modifying proofs from [10].

### 4.2. Termination

Let the *termination* operator $\tau$ be a unary operator on a dRA which maps an element in the carrier set to an assertion and that satisfies the following axioms

$$x = \tau xx, \tag{41}$$

$$\tau(g^\circ x) \sqsubseteq g^\circ, \tag{42}$$

$$\tau(x\tau y) = \tau(xy) \text{ and} \tag{43}$$

$$\tau x\top = x\top. \tag{44}$$

We will call a demonic refinement algebra with a termination operator a dRAt.

By convention, the termination operator has the same precedence as the enabledness operator. Similar models that were used for enabledness might be used to show that the fourth axiom cannot be derived from the first three.

The termination operator applied to a program denotes those states from which the program is guaranteed to terminate, that is, states from which it will not abort. It is important to keep in mind that $\tau x$ is an assertion (whereas $\epsilon x$ is a guard). Axiom (42), for example, says that an assertion that checks if the program $g^\circ x$ will terminate is refined by the assertion $g^\circ$. Since the program $x$ might not terminate, it can be argued that an assertion $g^\circ$ can do everything that can be done by an assertion that checks if the program $g^\circ x$ terminates. In Section 5 we show how the termination operator allows us to model program inversion and how to express a relation between total and weak correctness.

Axiom (44) can be shown to be equivalent – without the use of axioms (41)–(43) – to the axiom

$$\tau x = x\top \sqcap 1 \tag{45}$$

by the derivation

$$\begin{array}{ll} & \tau x\top \\ = & \{\text{assume: (45)}\} \\ & (x\top \sqcap 1)\top \\ = & \{\text{axioms (10), (6), (3), (8)}\} \\ & x\top \end{array}$$

and the derivation

$$\begin{array}{ll} & x\top \sqcap 1 \\ = & \{\text{assume: (44)}\} \\ & \tau x\top \sqcap 1 \\ = & \{\text{property (26)}\} \\ & \tau x. \end{array}$$

This does not mean that $\tau$ can be defined in terms of the other operators, since in that case one would also have to show that the right-hand side actually is an assertion. This seems impossible without the introduction of a negation operator [24],

and so axiom (45) is truly an *axiom* already assuming that $\tau$ is a mapping from the carrier set to the set of assertions. It can be shown – using "dual" reasoning to that of Solin [24] – that axioms (41)–(43) follow from axiom (45), so for $\tau$ we actually *only need one axiom*: (45) or, equivalently, (44). A "dual" result for enabledness to the equivalence of axioms (44) and (45) – involving an angelic-choice operator – was foreshadowed by Solin [24] and later proved by Möller (reported in [11]). In the present framework a similar move for enabledness is not possible, since we lack angelic choice.

It is easy to establish that

$$\tau(x \sqcap y) = \tau x \sqcap \tau y \text{ and} \tag{46}$$

$$x \sqsubseteq y \Rightarrow \tau x \sqsubseteq \tau y \tag{47}$$

follow from (45), cf. [24], and the next proposition can be proved similarly to the analogue for enabledness (38).

**Proposition 4.1.** *Let $x$ be any element in the carrier set of a* dRA *and let $g$ be any guard in the same set. Then*

$$x \sqsubseteq g^\circ x \Leftrightarrow \tau x \sqsubseteq g^\circ \tag{48}$$

*is equivalent to the axioms* (41) *and* (42).

### 4.3. The predicate-transformer model

In [6] the *miracle guard* is defined by the predicate

$$\neg\left(\bigcap_{q \in \wp(\Sigma)} S.q\right)$$

and the *abortion guard* by the predicate

$$\bigcup_{q \in \wp(\Sigma)} S.q.$$

Note that the concept of guard is now used – in line with the literature – in three different ways: guard in the abstract-algebraic sense, guard in the predicate-transformer sense and guard as in the predicates of the miracle and abortion guards.

Intuitively, the miracle guard is a predicate that holds in a state $\sigma \in \Sigma$ when the program $S$ is guaranteed to not perform a miracle, that is $S$ does not establish every postcondition starting in $\sigma$. The abortion guard holds in a state $\sigma \in \Sigma$ if the program $S$ will always terminate starting in $\sigma$, that is, it will establish some postcondition when starting in $\sigma$. When $S$ is isotone (which it is when conjunctive) the least $S.q$ is $S.\emptyset$ and the greatest $S.\Sigma$, so the miracle guard can be written $\neg(S.\emptyset)$ and the abortion guard $S.\Sigma$; this is the way that Nelson defines *grd* (guard) and *hlt* (halt), respectively, in [23].

We want the miracle guard and the abortion guard to model the enabledness operator and the termination operator, respectively. To do this, we lift them to the predicate-transformer level and, if $x$ is interpreted as the conjunctive predicate transformer $S$, set $\epsilon x$ to be $[\neg S.\emptyset]$ and $\tau x$ to be $\{S.\Sigma\}$. It can easily be established that this interpretation is sound for the axioms of $\epsilon$, (34)–(37) and of $\tau$, (41)–(44). For example, the axioms (34) and (42) are verified by

$$\begin{array}{ll} & [\neg(S.\emptyset)]; S \sqsubseteq S \\ \Leftrightarrow & \{\text{definitions}\} \\ & (\forall q \in \wp(\Sigma) \bullet [\neg(S.\emptyset)].(S.q) \subseteq S.q) \\ \Leftrightarrow & \{\text{definitions}\} \\ & (\forall q \in \wp(\Sigma) \bullet S.\emptyset \cup S.q \subseteq S.q) \\ \Leftrightarrow & \{\text{isotony of } S\} \\ & (\forall q \in \wp(\Sigma) \bullet \mathsf{true}) \\ \Leftrightarrow & \{\text{logic}\} \\ & \mathsf{true} \end{array}$$

and

$$\begin{array}{ll} & (\forall p \in \wp(\Sigma) \bullet \{(\{p\}; S).\Sigma\} \sqsubseteq \{p\}) \\ \Leftrightarrow & \{\text{definitions}\} \\ & (\forall p, q \in \wp(\Sigma) \bullet \{\{p\}.(S.\Sigma)\}.q \subseteq \{p\}.q) \\ \Leftrightarrow & \{\text{definitions}\} \\ & (\forall p, q \in \wp(\Sigma) \bullet p \cap S.\Sigma \cap q \subseteq p \cap q) \\ \Leftrightarrow & \{\text{set theory}\} \\ & (\forall p, q \in \wp(\Sigma) \bullet \mathsf{true}) \\ \Leftrightarrow & \{\text{logic}\} \\ & \mathsf{true}, \end{array}$$

respectively. The validity of the other axioms can be verified similarly.

### 4.4. Some basic properties

In this section we investigate some of the basic properties of $\epsilon$, $\tau$ and $^\omega$. The investigation reveals that some propositions that can be shown in KAD regarding the domain operator and the Kleene star, for example the induction rule [10], do not necessarily hold for $\epsilon$ and $^\omega$ in dRA. Note that, in this framework, the operators $\epsilon$ and $\tau$ are not fully dual, since the guards and assertions are not fully dual. As mentioned, this is related to the absence of angelic choice. If angelic choice is available, guards and assertions are fully dual and can be defined in terms of each other and, by that very fact, $\epsilon$ and $\tau$ are dual (cf. [24]).

Let us give an intuition for two of the statements of the next proposition. The first statement says that a guard is enabled (will not terminate miraculously) in exactly those states in which the guard-predicate holds. The fourth statement says that skip always terminates. The other statements can be understood similarly. The proposition also shows that the two operators have some symmetry with respect to the constants.

**Proposition 4.2.** *Let* $1$, $\top$ *and* $\bot$ *be the constants in a* dRA. *Then*

$$\epsilon g = g, \tag{49}$$
$$\tau g^\circ = g^\circ, \tag{50}$$
$$\epsilon 1 = 1, \tag{51}$$
$$\tau 1 = 1, \tag{52}$$
$$\epsilon \top = \top, \tag{53}$$
$$\tau \bot = \bot, \tag{54}$$
$$\epsilon \bot = 1 \quad \textit{and} \tag{55}$$
$$\tau \top = 1 \tag{56}$$

*hold.*

**Proof.** The first two statements, (49)–(50), follow from (7) and (27), and axioms (34)–(35) and (41)–(42), respectively. Parts (51)–(54) are direct consequences of the first two. The seventh part, (55), is proved by (27) and

$$\begin{array}{ll} & \epsilon\bot \\ \sqsubseteq & \{\text{properties (14), (40)}\} \\ & \epsilon 1 \\ = & \{\text{property (51)}\} \\ & 1. \end{array}$$

The last part, (56), is proved similarly to the seventh. □

The next proposition nests enabledness and termination and again reveals symmetry. For example, the first one says that termination-check is always enabled and that enabledness-check will always terminate. In fact, the first statement could be generalised: any assertion is always enabled and any guard will always terminate.

**Proposition 4.3.** *Let* $x$ *and* $y$ *be any elements in a* dRA. *Then*

$$\epsilon(\tau x) = 1 = \tau(\epsilon x), \tag{57}$$
$$\epsilon(x\tau y) = \epsilon x \textit{ and} \tag{58}$$
$$\tau(x\epsilon y) = \tau x \tag{59}$$

*hold.*

**Proof.** For the first part, note that $1 \sqsubseteq \epsilon(\tau x)$ follows from (27), whereas

$$\epsilon(\tau x) \sqsubseteq \epsilon 1 = 1$$

follows from (27), isotony (40) and (51). In turn, $\tau(\epsilon x) \sqsubseteq 1$ follows from (27), whereas

$$1 = \tau 11 = \tau 1 \sqsubseteq \tau(\epsilon x)$$

follows from the axioms of $\tau$ and (27). The second part follows from

$$\begin{array}{ll} & \epsilon(x\tau y) \\ = & \{\text{axiom (36)}\} \\ & \epsilon(x\epsilon(\tau y)) \\ = & \{\text{axiom (57)}\} \\ & \epsilon(x1) \\ = & \{\text{axiom (7)}\} \\ & \epsilon x. \end{array}$$

The third part is shown similarly to the second. □

We also have asymmetries, as the following proposition shows.

**Proposition 4.4.** *Let $x$ be an element in a* dRA. *Then*

$$(\epsilon x)^\omega \sqsubseteq 1, \tag{60}$$
$$\epsilon(x^\omega) = 1, \tag{61}$$
$$(\tau x)^\omega = \bot \text{ and} \tag{62}$$
$$\tau(x^\omega) \sqsubseteq 1 \tag{63}$$

*hold.*

**Proof.** The first part follows from (11). The second part holds since $\epsilon(x^\omega) \sqsubseteq \epsilon 1 = 1$ by (11), (39), isotony and (51), and the converse follows from (27). For the third part, first note that one way follows from $\bot$ being a least element. The other direction follows from (27) and isotony by $(\tau x)^\omega \sqsubseteq 1^\omega = \bot$. The last part follows from (27). To see that the converse does not hold, take $x = 1$. □

Similarly to KAD we have unfolding rules.

**Proposition 4.5.** *Let $x$ and $y$ be elements in a* dRA. *Then*

$$\epsilon(x^\omega y) = \epsilon(x\epsilon(x^\omega y)) \sqcap \epsilon y \quad \text{and} \tag{64}$$
$$\tau(x^\omega y) = \tau(x\tau(x^\omega y)) \sqcap \tau y \tag{65}$$

*hold.*

**Proof.** The calculation

$$\begin{aligned}
&\epsilon(x^\omega y)\\
=\ &\{\text{axiom (11)}\}\\
&\epsilon((xx^\omega \sqcap 1)y)\\
=\ &\{\text{axiom (10)}\}\\
&\epsilon(xx^\omega y \sqcap y)\\
=\ &\{\text{property (39)}\}\\
&\epsilon(xx^\omega y) \sqcap \epsilon y\\
=\ &\{\text{axiom (36)}\}\\
&\epsilon(x\epsilon(x^\omega y)) \sqcap \epsilon y
\end{aligned}$$

establishes the first part. The second part is proved in a similar fashion. □

We do not, however, have an induction rule analogous to the one of KAD.

**Proposition 4.6.** *Let $x$ be any element and $g$ be any guard in a* dRA. *Then the implication*

$$g \sqsubseteq \epsilon(xg) \Rightarrow g \sqsubseteq \epsilon(x^\omega g)$$

*does* not *hold in general. That is, there is an instantiation of $x$ and $g$ such that $g \sqsubseteq \epsilon(xg)$ holds, but $g \sqsubseteq \epsilon(x^\omega g)$ does not.*

**Proof.** Take $x = 1$ and $g = \top$. Then the antecedent becomes $\top \sqsubseteq \epsilon\top = \top$, which clearly holds. The consequent becomes $\top \sqsubseteq \epsilon(1^\omega\top) = \epsilon(\bot\top) = \epsilon\bot = 1$, which clearly does not hold. □

The reason that this does not hold is related to the fact that we cannot prove (19), that is,

$$xz \sqsubseteq yx \Rightarrow xz^\omega \sqsubseteq y^\omega x$$

does not hold in dRA. This is easily seen when trying to prove the implication along the lines of [10]:

$$\begin{aligned}
&g \sqsubseteq \epsilon(xg)\\
\Leftrightarrow\ &\{\text{property (38)}\}\\
&gxg \sqsubseteq xg\\
\Leftrightarrow\ &\{\text{property (32)}\}\\
&gx \sqsubseteq xg\\
\not\Leftarrow\ &\{(19)\}\\
&gx^\omega \sqsubseteq x^\omega g\\
\Leftrightarrow\ &\{\text{property (32)}\}\\
&gx^\omega g \sqsubseteq x^\omega g\\
\Leftrightarrow\ &\{\text{property (38)}\}\\
&g \sqsubseteq \epsilon(x^\omega g).
\end{aligned}$$

But as can be seen from the above (by reading backwards), we do nevertheless have the following result.

**Proposition 4.7.** *Let $x$ be any element and $g$ be any guard in a* dRA. *Then*

$$gx^\omega \sqsubseteq x^\omega g \Leftrightarrow g \sqsubseteq \epsilon(x^\omega g) \tag{66}$$

*holds.*

On the other hand, we have an induction rule for $\tau$, which again reveals some asymmetry between $\epsilon$ and $\tau$.

**Proposition 4.8.** *Let $x$ be any element and $g$ be any guard in a* dRA. *Then*

$$\tau(xg^\circ) \sqsubseteq g^\circ \;\Rightarrow\; \tau(x^\omega g^\circ) \sqsubseteq g^\circ \tag{67}$$

*holds.*

**Proof.** The derivation

$$\begin{array}{ll} & \tau(xg^\circ) \sqsubseteq g^\circ \\ \Leftrightarrow & \{\text{property (48)}\} \\ & xg^\circ \sqsubseteq g^\circ xg^\circ \\ \Leftrightarrow & \{\text{property (33)}\} \\ & xg^\circ \sqsubseteq g^\circ x \\ \Rightarrow & \{\text{property (18)}\} \\ & x^\omega g^\circ \sqsubseteq g^\circ x^\omega \\ \Leftrightarrow & \{\text{property (33), (48)}\} \\ & \tau(x^\omega g^\circ) \sqsubseteq g^\circ \end{array}$$

proves the claim. □

### 4.5. More on termination

Cohen and others have used $x^\omega\top = \top$ to express the termination of $x$ (remember that our $x^\omega\top$ is equivalent to Cohen's omega operator applied to $x$). Intuitively, this says that $x^\omega$ is always guaranteed to end up in *some* state, so it will not abort. This notion is equivalent to our notion of termination of $x^\omega$ expressed by $\tau(x^\omega) = 1$ and follows from this more general equivalence:

$$\tau x = 1 \Leftrightarrow x\top = \top. \tag{68}$$

The equivalence is proved by the derivation

$$\begin{array}{ll} & \tau x \\ = & \{\text{axiom (45)}\} \\ & x\top \sqcap 1 \\ = & \{\text{assume: } x\top = \top\} \\ & \top \sqcap 1 \\ = & \{\text{axiom (3)}\} \\ & 1 \end{array}$$

and the derivation

$$\begin{array}{ll} & \top \\ = & \{\text{axiom (6)}\} \\ & 1\top \\ \sqsubseteq & \{\text{assume: } \tau x = 1,\ \text{that is } x\top \sqcap 1 = 1,\ \text{so } 1 \sqsubseteq x\top\} \\ & x\top\top \\ = & \{\text{axiom (8)}\} \\ & x\top \end{array}$$

in combination with the fact that $\top$ is the greatest element. If we would have the weak iteration operator [27,28] available, then we could show that

$$\tau(x^\omega) = 1 \Leftrightarrow x^\omega = x^* \quad \text{and} \quad \tau x = 1$$

holds (by isolation, $^*$-induction, the above property (68) and a couple of other basic properties). That is, $x^\omega$ terminates if the iteration is finite *and* $x$ terminates. Although this characterisation might be good for intuition or could serve as a technical aid, the equivalence shows that the characterisation can, in essence, be expressed solely via $^\omega$ and so forms yet another argument for leaving out weak iteration from the framework.

It is also easy to see that

$$\tau x = \bot \Leftrightarrow x = \bot.$$

This follows from the fact that, assuming $\tau x = \bot$, we have

$$\begin{array}{ll} & x = \bot \\ \Leftrightarrow & \{\text{property (15)}\} \\ & x = \bot x \\ \Leftrightarrow & \{\text{assumption}\} \\ & x = \tau x x \\ \Leftrightarrow & \{\text{axiom (41)}\} \\ & \mathsf{true} \end{array}$$

and that, assuming $x = \bot$,

$$\begin{array}{ll} & \tau x = \bot \\ \Leftrightarrow & \{\text{axiom (45)}\} \\ & x\top \sqcap 1 = \bot \\ \Leftrightarrow & \{\text{assumption}\} \\ & \bot\top \sqcap 1 = \bot \\ \Leftrightarrow & \{\text{properties (15), (14)}\} \\ & \mathsf{true} \end{array}$$

holds.

## 5. The algebra in action

We show how different relations between programs can be expressed in the algebra. We also demonstrate the algebra's applicability by using it for proving two transformation rules for action systems.

### 5.1. Expressing relations between programs

The enabledness and termination operators can be used to express properties between programs; we here list some examples. First note that $\overline{\epsilon x}$ is a guard that skips in those states where $x$ is *disabled*.

**Excludes, enables, disables.** A program $x$ *excludes* a program $y$ if whenever $x$ is enabled $y$ is not. This can be formalised by saying that $x$ is equal to first executing a guard that checks that $y$ is disabled and then executing $x$, algebraically: $x = \overline{\epsilon y}x$. A program $x$ *enables* $y$ if $y$ is enabled after having executed $x$, algebraically: $x = x\epsilon y$. Similarly as above $x$ *disables* $y$ if $x = x\overline{\epsilon y}$.

Using the algebra, we can prove that exclusion is commutative, i.e., $x$ excludes $y$ if and only if $y$ excludes $x$:

$$\begin{array}{ll} & x = \overline{\epsilon y}x \\ \Leftrightarrow & \{\text{property (38)}\} \\ & \overline{\epsilon y} \sqsubseteq \epsilon x \\ \Leftrightarrow & \{\text{guards form a Boolean algebra}\} \\ & \overline{\epsilon x} \sqsubseteq \epsilon y \\ \Leftrightarrow & \{\text{property (38)}\} \\ & y = \overline{\epsilon x}y. \end{array}$$

We can also express that termination of $x$ requires termination or enabledness of $y$, $x = \tau yx$ and $x = \epsilon yx$, respectively.

**Program inversion.** A program $x'$ inverts a program $x$ when execution of the sequence $xx'$ results in the final state being the same as the initial state [14,8]. Using the termination operator, program inversion can be defined as

$$x' \text{ inverts } x \Leftrightarrow \tau x \sqsubseteq xx'.$$

Intuitively, this says that the assertion that skips in those states from which $x$ terminates and aborts in all other states can be replaced by the program $xx'$: if $x$ terminates and $x'$ inverts $x$ then $xx'$ skips, otherwise (that is, $x$ does not terminate) $xx'$ aborts. In [27] von Wright used the the right-hand side of (45) for proving a number of program inversion rules, but without explicitly relating it to termination.

**Correctness.** Total correctness and weak correctness ("partial correctness in a total correctness framework") were characterised abstract-algebraically by von Wright [27,28]. Total correctness is characterised by

$$g_1 x \bar{g_2} = \top$$

for any program $x$ and any precondition–postcondition expressed by the guards $g_1$ and $g_2$, respectively. Using the enabledness operator, total correctness can be expressed by

$$\bar{g_1} \sqsubseteq \epsilon(x\bar{g_2}),$$

since by (38) and (30) this is equivalent to the total correctness condition.

Total correctness says that a program is correct with respect to a precondition–postcondition pair *and* that the program terminates (from the states denoted by the precondition). Weak correctness says that a program is correct with respect to a precondition–postcondition pair *if* the program terminates. This suggests that total correctness can be expressed as a weak-correctness assertion and the extra assumption that the program terminates. Indeed, this can be proved in our algebra using the termination operator and von Wright's characterisation of weak correctness. Weak correctness of a program $x$ with respect to a precondition expressed by the guard $g_1$ and a postcondition expressed by a guard $g_2$ is characterised by $xg_2 \sqsubseteq g_1 x$, and for the termination of $x$ under the precondition expressed by a guard $g$ we use the termination operator and state $\tau(gx) = 1$. We can then show that total correctness is equivalent to weak correctness plus the assumption that the program under consideration is terminating (this is the *pairing condition* of Nelson [23]):

$$g_1 x \bar{g_2} = \top \Leftrightarrow xg_2 \sqsubseteq g_1 x \quad \text{and} \quad \tau(g_1 x) = 1.$$

Assuming the weak-correctness and the termination assertions, the total correctness assertion can be showed to hold by the derivation

$$
\begin{array}{ll}
 & \top = g_1 x \bar{g_2} \\
\Leftrightarrow & \{\text{axiom (3)}\} \\
 & \top \sqsubseteq g_1 x \bar{g_2} \\
\Leftrightarrow & \{\text{assumption, property (29)}\} \\
 & \top \sqsubseteq g_1 x g_2 \bar{g_2} \\
\Leftrightarrow & \{\text{basic guard property (24)}\} \\
 & \top \sqsubseteq g_1 x \top \\
\Leftrightarrow & \{\text{axiom (8)}\} \\
 & \top \sqsubseteq g_1 x \top \top \\
\Leftarrow & \{\text{isotony}\} \\
 & 1 \sqsubseteq g_1 x \top \\
\Leftrightarrow & \{(45) \text{ and assumption}\} \\
 & \mathsf{true}.
\end{array}
$$

Assuming total correctness, weak correctness follows from

$$
\begin{array}{ll}
 & g_1 x \\
= & \{\text{basic guard property (24) and axiom (10)}\} \\
 & g_1 x g_2 \sqcap g_1 x \bar{g_2} \\
= & \{\text{assumption}\} \\
 & g_1 x g_2 \sqcap \top \\
= & \{\text{axiom (3)}\} \\
 & g_1 x g_2
\end{array}
$$

and (29). The derivation

$$
\begin{array}{ll}
 & 1 \\
\sqsubseteq & \{\text{axiom (3)}\} \\
 & \top \\
= & \{\text{assumption}\} \\
 & g_1 x \bar{g_2} \\
= & \{\text{weak correctness follows from total correctness and (29)}\} \\
 & g_1 x g_2 \bar{g_2} \\
= & \{\text{basic guard property (24)}\} \\
 & g_1 x \top
\end{array}
$$

shows, together with (45), that the termination condition follows from the total correctness assertion.

### 5.2. Action systems

Action systems comprise a formalism for reasoning about distributed systems [3–5]. An *action system*, denoted

$$y; \mathsf{do}\, x_0 [] \ldots [] x_n \,\mathsf{od}; z,$$

is an iteration of a demonic choice $x_0 \sqcap \cdots \sqcap x_n$ between a fixed number of *actions*, $x_0, \ldots, x_n$, that terminates when none of the actions is any longer enabled. The statement $y$ is the initialising action and the statement $z$ is the finalising action.

In the refinement algebra, an action system takes the form

$$y(x_0 \sqcap \cdots \sqcap x_n)^\omega \overline{\epsilon x_0} \ldots \overline{\epsilon x_n} z.$$

The actions are thus iterated – which is expressed with the strong iteration operator, so infinite iteration means abortion – until none of them is any longer enabled, which is expressed with the enabledness operator.

We begin by showing that action systems have a leapfrog property:

$$x; \mathsf{do}\, y; x \,\mathsf{od} \sqsubseteq \mathsf{do}\, x; y \,\mathsf{od}; x.$$

We will prove this property in the algebra and at the same time expose a methodology for performing derivations. Action-system leapfrog takes the form

$$x(yx)^\omega \overline{\epsilon(yx)} \sqsubseteq (xy)^\omega \overline{\epsilon(xy)} x \tag{69}$$

in the algebra. We can now embark on proving (69) collecting assumptions, which are then, in turn, proved:

$$
\begin{array}{ll}
 & x(yx)^\omega \overline{\epsilon(yx)} \\
= & \{\text{leapfrog (16)}\} \\
 & (xy)^\omega x \overline{\epsilon(yx)} \\
\sqsubseteq & \{\text{assume: } x\overline{\epsilon(yx)} \sqsubseteq \overline{\epsilon(xy)} x\} \\
 & (xy)^\omega \overline{\epsilon(xy)} x.
\end{array}
$$

The assumption collected in the second step is shown to hold by the following derivation.

$$
\begin{array}{ll}
 & x\epsilon(\overline{yx}) \sqsubseteq \overline{\epsilon(xy)}x \\
\Leftarrow & \{\text{property (29)}\} \\
 & \top \sqsubseteq \overline{\epsilon(xy)}x\epsilon(yx) \\
\Leftrightarrow & \{\text{property (30)}\} \\
 & \epsilon(xy)x\epsilon(yx) \sqsubseteq x\epsilon(yx) \\
\Leftrightarrow & \{\text{property (38)}\} \\
 & \epsilon(xy) \sqsubseteq \epsilon(x\epsilon(yx)) \\
\Leftrightarrow & \{\text{axiom (36)}\} \\
 & \epsilon(xy) \sqsubseteq \epsilon(xyx) \\
\Leftrightarrow & \{\text{axiom (36)}\} \\
 & \epsilon(xy) \sqsubseteq \epsilon(xy\epsilon x) \\
\Leftarrow & \{\text{property (27) and isotony}\} \\
 & \text{true}.
\end{array}
$$

The same result has been shown in the predicate transformer model by Back and von Wright [7], but our proof is much cleaner and more simple.

An action system can be decomposed as

$$\mathsf{do}\ x\ [\!]\ y\ \mathsf{od} = \mathsf{do}\ y\ \mathsf{od};\ \mathsf{do}\ x;\ \mathsf{do}\ y\ \mathsf{od}\ \mathsf{od}$$

provided that $x$ excludes $y$. This property does not depend on the initialising and the finalising actions. In the refinement algebra, action-system decomposition can be encoded as

$$(x \sqcap y)^\omega \overline{\epsilon x}\ \overline{\epsilon y} = y^\omega \overline{\epsilon y}(xy^\omega \overline{\epsilon y})^\omega \overline{\epsilon(xy^\omega \overline{\epsilon y})} \tag{70}$$

and the assumption as $x = \overline{\epsilon y}x$. This result was also proved by Back and von Wright [7], but again the reasoning in the abstract algebra presented here is slicker and leaner. We calculate

$$
\begin{array}{ll}
 & (x \sqcap y)^\omega \overline{\epsilon x}\,\overline{\epsilon y} \\
= & \{\text{decomposition (17)}\} \\
 & y^\omega (xy^\omega)^\omega \overline{\epsilon x}\,\overline{\epsilon y} \\
= & \{\text{assumption}\} \\
 & y^\omega (\overline{\epsilon y}xy^\omega)^\omega \overline{\epsilon x}\,\overline{\epsilon y} \\
= & \{\text{guards form a Boolean algebra}\} \\
 & y^\omega (\overline{\epsilon y}xy^\omega)^\omega \overline{\epsilon y}\,\overline{\epsilon x} \\
= & \{\text{leapfrog (16)}\} \\
 & y^\omega \overline{\epsilon y}(xy^\omega \overline{\epsilon y})^\omega\ \overline{\epsilon x} \\
= & \{\text{assume: } \epsilon x = \epsilon(xy^\omega \overline{\epsilon y})\} \\
 & y^\omega \overline{\epsilon y}(xy^\omega \overline{\epsilon y})^\omega\ \overline{\epsilon(xy^\omega \overline{\epsilon y})}.
\end{array}
$$

The assumption follows from the derivation

$$
\begin{array}{ll}
 & \epsilon(xy^\omega \overline{\epsilon y}) = \epsilon x \\
\Leftrightarrow & \{\text{axiom (7)}\} \\
 & \epsilon(xy^\omega \overline{\epsilon y}1) = \epsilon(x1) \\
\Leftrightarrow & \{\text{property (55)}\} \\
 & \epsilon(xy^\omega \overline{\epsilon y}\epsilon\bot) = \epsilon(x\epsilon\bot) \\
\Leftrightarrow & \{\text{axiom (36)}\} \\
 & \epsilon(xy^\omega \overline{\epsilon y}\bot) = \epsilon(x\bot) \\
\Leftarrow & \{\text{property (40) and isotony of ;}\} \\
 & y^\omega \overline{\epsilon y}\bot = \bot \\
\Leftrightarrow & \{\bot \text{ least element (14)}\} \\
 & y^\omega \overline{\epsilon y}\bot \sqsubseteq \bot \\
\Leftarrow & \{\text{axiom (13)}\} \\
 & y\bot \sqcap \overline{\epsilon y}\bot \sqsubseteq \bot \\
\Leftrightarrow & \{\text{axiom (37)}\} \\
 & \epsilon y\bot \sqcap \overline{\epsilon y}\bot \sqsubseteq \bot \\
\Leftrightarrow & \{\text{axiom (10)}\} \\
 & (\epsilon y \sqcap \overline{\epsilon y})\bot \sqsubseteq \bot \\
\Leftrightarrow & \{\text{basic guard property (24)}\} \\
 & 1\bot \sqsubseteq \bot \\
\Leftrightarrow & \{\text{axiom (6)}\} \\
 & \text{true},
\end{array}
$$

which concludes the proof. □

## 6. Concluding remarks

We have introduced a demonic refinement algebra restricted to strong iteration and extended it with the enabledness operator and the termination operator. We have investigated basic algebraic properties of the new operators and also applied them. The applications showed how reasoning in the abstract algebra is more perspicuous and elegant than earlier model-theoretic reasoning.

The reduced refinement algebra and its extension deserve further investigation. Since total correctness is what we are interested in, the restriction of the signature to merely the strong iteration operator is motivated. However, some propositions concerning $^\omega$ that were proved in [27] rely on the weak iteration operator (Kleene star) in their proofs. To what extent these types of propositions can be proved in the reduced algebra should be investigated. More generally, the completeness of the axiomatisation with respect to the predicate transformer model and decidability results are to be settled. Case studies where the new operators are applied to larger problems should also be interesting.

## Acknowledgements

Thanks are due to R.J.R. Back, Orieta Celiku, Jules Desharnais, E.C.R. Hehner, Peter Höfner, L.A. Meinicke, Bernhard Möller and Viorel Preoteasa for elucidating discussions and careful scrutiny. The authors are also grateful to several anonymous referees for helpful comments. The first author's work was partly done while visiting Institut für Informatik, Universität Augsburg.

## References

[1] K. Aboul-Hosn, D.C. Kozen, KAT-ML: An interactive theorem prover for Kleene algebra with tests, J. Appl. Non-Classical Logics 16 (1–2) (2006) 9–34.
[2] R.J.R. Back, Correctness Preserving Program Refinements: Proof Theory and Applications, in: Mathematical Centre Tracts, vol. 131, Mathematical Centre, Amsterdam, 1980.
[3] R.J.R. Back, R. Kurki-Suonio, Decentralisation of process nets with centralised control, in: Proc. of 2nd Ann. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing, PODC '83, Montreal, Aug. 1983, ACM Press, New York, 1983, pp. 131–142.
[4] R.J.R. Back, A method for refining atomicity in parallel algorithms, in: E. Odijk, M. Rem, J.-C. Syre (Eds.), Proc. of Conf. on Parallel Architectures and Languages Europe, PARLE '89, Eindhoven, June 1989, in: Vol. II: Parallel Languages, Lect. Notes in Comput. Sci., vol. 366, Springer, Berlin, 1989, pp. 199–216.
[5] R. J. R. Back, K. Sere, Stepwise refinement of action systems, Struct. Program. 12 (1) (1991) 17–30.
[6] R.J.R. Back, J. von Wright, Refinement calculus: A systematic introduction, in: Graduate Texts in Computer Science, Springer, New York, 1998.
[7] R.J.R. Back, J. von Wright, Reasoning algebraically about loops, Acta Inform. 36 (4) (1999) 295–334.
[8] W. Chen, J.T. Udding, Program inversion: more than fun!, Sci. Comput. Programming 15 (1) (1990) 1–13.
[9] E. Cohen, Separation and reduction, in: R. Backhouse, J.N. Oliveira (Eds.), Proc. of 5th Int. Conf. on Mathematics of Program Construction, MPC 2000, Ponte de Lima, July 2000, in: Lect. Notes in Comput. Sci., vol. 1837, Springer, Berlin, 2000, pp. 45–59.
[10] J. Desharnais, B. Möller, G. Struth, Kleene algebra with domain, ACM Trans. Comput. Logic 7 (4) (2006) 798–833.
[11] J. Desharnais, B. Möller, G. Struth, Algebraic notions of termination, Technischer Bericht 2006-23, Institut für Informatik, Universität Augsburg, 2006.
[12] E.W. Dijkstra, A Discipline of Programming, Prentice Hall, Englewood Cliffs, 1976.
[13] R.W. Floyd, Assigning meanings to programs, in: J.T. Schwartz (Ed.), Mathematical Aspects of Computer Science, in: Proc. of Symp. on Applied Math, vol. 19, Amer. Math. Soc., Providence, RI, 1967, pp. 19–32.
[14] D. Gries, The science of programming, in: Texts and Monographs in Computer Science, Springer, New York, 1981.
[15] P. Höfner, G. Struth, Automated reasoning in Kleene algebra, in: F. Pfenning (Ed.), Proc. of 21st Int. Conf. on Automated Deduction, CADE-21, Bremen, July 2007, in: Lect. Notes in Artif. Intell., vol. 4603, Springer, Berlin, 2007, pp. 279–294.
[16] P. Höfner, G. Struth, Can refinement be automated?, in: E.A. Boiten, J. Derrick, G. Smith (Eds.), Proc. of 12th BCS-FACS Refinement Wksh, REFINE 2007, Oxford, July 2007, in: Electron. Notes in Theor. Comput. Sci., vol. 201, 2008, pp. 197–222.
[17] P. Höfner, B. Möller, K. Solin, Omega algebra, demonic refinement algebra and commands, in: R. Schmidt (Ed.), Proc. of 9th Int. Conf. on Relational Methods in Computer Science and 4th Int. Wksh. on Applications and Kleene Algebra, RelMiCS/AKA 2006, Manchester, Aug./Sept. 2006, in: Lect. Notes in Comput. Sci., vol. 3125, Springer, Berlin, 2006, pp. 222–234.
[18] D. Kozen, A completeness theorem for Kleene algebras and the algebra of regular events, Inform. and Comput. 110 (2) (1994) 366–390.
[19] D. Kozen, Kleene algebra with tests, ACM Trans. Program. Lang. Syst. 19 (3) (1997) 427–443.
[20] L.A. Meinicke, K. Solin, Refinement algebra for probabilistic programs, in: E.A. Boiten, J. Derrick, G. Smith (Eds.), Proc. of 12th BCS-FACS Refinement Wksh, REFINE 2007, Oxford, July 2007, in: Electron. Notes in Theor. Comput. Sci., vol. 201, 2008, pp. 177–195.
[21] B. Möller, Kleene getting lazy, Sci. Comput. Programming 65 (2) (2007) 195–214.
[22] C.C. Morgan, Programming from Specifications, 2nd ed., in: Prentice Hall Int. Series in Comput. Sci., Prentice Hall, New York, 1994.
[23] G. Nelson, A generalization of Dijkstra's calculus, ACM Trans. Program. Lang. Syst. 11 (4) (1989) 517–561.
[24] K. Solin, On two dually nondeterministic refinement algebras, in: R. Schmidt (Ed.), Proc. of 9th Int. Conf. on Relational Methods in Computer Science and 4th Int. Wksh. on Applications and Kleene Algebra, RelMiCS/AKA 2006, Manchester, Aug./Sept. 2006, in: Lect. Notes in Comput. Sci., vol. 4136, Springer, Berlin, 2006, pp. 373–387.
[25] K. Solin, A sketch of a dynamic epistemic semiring, in: D. Leivant, R.J.G.B. de Queiroz (Eds.), Proc. of 14th Int. Wksh. on Logic, Language, Information and Computation, WoLLIC 2007, Rio de Janeiro, July 2007, in: Lect. Notes in Comput. Sci., vol. 4576, Springer, Berlin, 2007, pp. 337–350.
[26] K. Solin, J. von Wright, Refinement algebra with operators for enabledness and termination, in: T. Uustalu (Ed.), Proc. of 8th Int. Conf. on Mathematics of Program Construction, MPC 2006, Kuressaare, July 2006, in: Lect. Notes in Comput. Sci., vol. 4014, Springer, Berlin, 2006, pp. 397–415.
[27] J. von Wright, From Kleene algebra to refinement algebra, in: E.A. Boiten, B. Möller (Eds.), Proc. of 6th Int. Conf. on Mathematics of Program Construction, MPC 2002, Dagstuhl, July 2002, in: Lect. Notes in Comput. Sci., vol. 2386, Springer, Berlin, 2002, pp. 233–262.
[28] J. von Wright, Towards a refinement algebra, Sci. Comput. Programming 51 (1–2) (2004) 23–45.