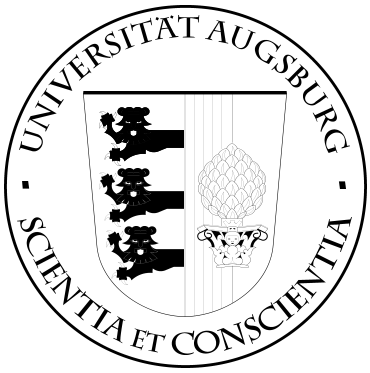
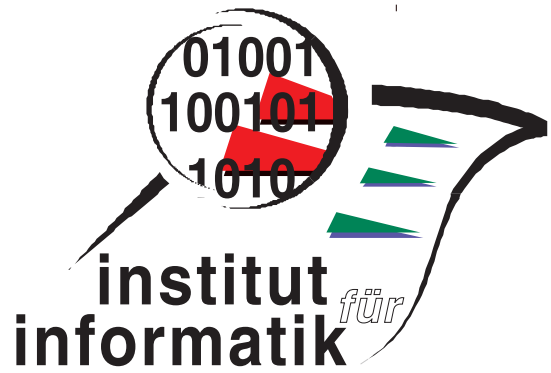
Universität Augsburg



On Two Dually Nondeterministic Refinement Algebras

Kim Solin

Report 2006-05 February 2006



Institut für Informatik
D-86135 Augsburg

On Two Dually Nondeterministic Refinement Algebras

Kim Solin*

Turku Centre for Computer Science
Lemminkäinengatan 14 A, FIN-20520 Åbo, Finland
kim.solin@utu.fi

Abstract. A dually nondeterministic refinement algebra with a negation operator is proposed. The algebra facilitates reasoning about total-correctness preserving program transformations and nondeterministic programs. The negation operator is used to express enabledness and termination operators through a useful explicit definition. As a small application, a property of action systems is proved employing the algebra. A dually nondeterministic refinement algebra without the negation operator is also discussed.

1 Introduction

Refinement algebras are abstract algebras for reasoning about program refinement [18, 21, 22, 20]. Axiomatic reasoning can, in a certain sense, provide a simpler reasoning tool than the classical set and order-theoretic frameworks [1, 5, 16]. Different classes of predicate transformers over a fixed state space form the motivating models, but should not be seen as exclusive.

The first papers on refinement algebras, in our sense of the term, were von Wright's initial paper [21], followed by [22], which builds on the aforementioned. In these papers von Wright outlines an axiomatisation with the set of isotone predicate transformers as an intended model. He also proposes the introduction of angelic choice as a separate operator in the algebra.

This paper proposes a refinement algebra that extends the original framework with three operators: the angelic choice (as suggested by von Wright), angelic iterative choice and a negation operator. Looking at the predicate-transformer models, the negation operator demands that the set of all predicate transformers be a model, whereas the iteration operator demands that the predicate transformers be isotone (as a consequence of needing to establish the existence of fixpoints via the Knaster-Tarski theorem). To solve this conflict, we let the carrier set be the set of all predicate transformers over a fixed state space and impose isotony conditions on elements of axioms involving iteration. Taking one step further, we also add ways of imposing conjunctivity and disjunctivity conditions on elements. Thus one could say that the algebra we propose is an algebra intended for reasoning about isotone predicate transformers, but having the whole class of predicate transformers as a model.

* Currently visiting Institut für Informatik, Universität Augsburg.

In the earlier frameworks, assertions were always defined in terms of guards. In the framework we propose here, guards can also be defined in terms of assertions. The guards and the assertions thus have equal status. Together with von Wright we have investigated an enabledness and a termination operator in refinement algebra [20]. The enabledness operator applied to a program denotes those states from which the program is enabled, that is those states from which the program will not terminate miraculously. The termination operator, on the other hand, yields those states from which the program is guaranteed to terminate in some state, that is, the program will not abort. In this paper, these operators are defined in terms of the other operators as opposed to our earlier work where they were introduced with an implicit axiomatisation. Thus, the framework of this paper subsumes the one of [20].

Action systems comprise a formalism for reasoning about parallel programs [2, 4]. The intuition is that an action system is an iteration of a fixed number of demonic choices that terminates when none of the conjunctive *actions* are enabled any longer. In the refinement algebra, an action system can be expressed using the enabledness operator. An action system can be decomposed so that the order of execution is clearly expressed; this has been shown by Back and von Wright using predicate transformer reasoning [6]. In the axiomatisation of [20] we were able to prove one direction of action system decomposition, but the other direction seems to be harder. Using the framework we present here, both directions can be derived quite easily.

When the negation operator is left out we obtain a dually nondeterministic refinement algebra for which the isotone predicate transformers constitute a model. This means that no special conditions need to be imposed on the elements to guarantee the existence of fixpoints. Also in this framework guards and assertions can be defined in terms of each other. On the other hand, explicit definitions of the enabledness and termination operators, upon which the proof of action-system decomposition relies, seem not to be possible.

The following work can be traced in the history 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 [12, 14]. Von Wright's non-conservative extension of Kleene algebra with tests was the first abstract algebra that was genuinely an algebra for total correctness (it drops right-annihilation) [21]. It rests upon previous work on algebraic program reasoning by Back and von Wright [6]. Desharnais, Möller, and Struth extended Kleene algebra with a domain operator [8], upon which Möller relaxed Kleene algebra by giving up right-annihilation (as in [21]) *and* right-distributivity of composition. These two papers laid a firm ground to the the developments in [20], where the enabledness and termination operators were introduced. Angelic nondeterminism takes off in the theory of nondeterministic automata and Floyd's nondeterministic programs [10]. In the context of program refinement, Broy and Nelson [15, 7], Back and von Wright [3], and Gardiner and Morgan [11] are early names. The present paper extends an earlier workshop version [19].

The paper is set up as follows. First the abstract algebra is proposed and a program intuition is given. Then a predicate-transformer model for the algebra is provided, which serves as a program-semantical justification. After the model, basic properties of the algebra are discussed. The third section extends the algebra by guards and assertions. After this the termination and enabledness operators are introduced. Action systems are considered under the abstract-algebraic view in Section 4. The final section before the concluding one, remarks on a dually nondetereministic refinement algebra without the negation operator.

The purpose of this paper is not to provide more grandiose applications nor to give a complete algebraic treatment; the purpose is to lay down the first strokes of the brush, the purpose is to get started.

2 A dually nondeterministic refinement algebra with negation

In this section we propose a dually nondeterministic refinement algebra with negation, give a predicate transformer model, and have a glance at some basic properties that should be taken into account.

2.1 Axiomatisation

A *dually nondeterministic refinement algebra with negation* (dndRAn) is a structure over the signature $(\sqcap, \sqcup, \neg, ;, ^\omega, ^\dagger, \bot, \top, 1)$ such that $(\sqcap, \sqcup, \neg, \bot, \top)$ is a Boolean algebra, $(;, 1)$ is a monoid, and the following equations hold (; left implicit, $x \sqsubseteq y \overset{\text{def}}{\Leftrightarrow} x \sqcap y = x$):

$$\begin{aligned} \neg xy &= \neg(xy) \\ \top x &= \top \qquad & \bot x &= \bot \\ (x \sqcap y)z &= xz \sqcap yz \qquad & (x \sqcup y)z &= xz \sqcup yz \end{aligned}$$

Moreover, if an element x satisfies $y \sqsubseteq z \Rightarrow xy \sqsubseteq xz$ we say that x is *isotone* and if x and y are isotone, then

$$\begin{aligned} x^\omega &= xx^\omega \sqcap 1 \qquad & xz \sqcap y \sqsubseteq z \;&\Rightarrow\; x^\omega y \sqsubseteq z \\ x^\dagger &= xx^\dagger \sqcup 1 \qquad & z \sqsubseteq xz \sqcup y \;&\Rightarrow\; z \sqsubseteq x^\dagger y \end{aligned}$$

hold. If x satisfies $x(y \sqcap z) = xy \sqcap xz$ and $x(y \sqcup z) = xy \sqcup xz$ we say that x is *conjunctive* and *disjunctive*, respectively. Of course, conjunctivity or disjunctivity implies isotony.

The operator $\neg$ binds stronger than the equally strong $^\omega$ and †, which in turn bind stronger than ;, which, finally, binds stronger than the equally strong $\sqcap$ and $\sqcup$.

Let us remark that the signature could be reduced to $(\sqcap, \neg, ;, ^\omega, 1)$, since the other operators can be defined in terms of these. Some of the axioms could also

be left out, since they can be derived as theorems. For clarity, we choose to have the more spelled-out axiomatisation.

As a rough intuition, the elements of the carrier set can be seen as program statements. The operators should be understood so that $\sqcap$ is demonic choice (a choice we cannot affect), $\sqcup$ is angelic choice (a choice we can affect), ; is sequential composition, $\neg x$ terminates in any state where x would not terminate and the other way around, $^\omega$, the strong (demonic) iteration, is an iteration that either terminates *or* goes on indefinitely, in which case it aborts; and †, the strong angelic iteration, is an iteration that terminates *or* goes on indefinitely, in which case a miracle occurs. If y establishes anything that x does and possibly more, then x is refined by y: $x \sqsubseteq y$. The constant $\bot$ is abort, an always aborting program statement; $\top$ is magic, a program statement that establishes any postcondition; and 1 is skip. A conjunctive element can be seen as facilitating demonic nondeterminism, but not angelic, whereas a disjunctive element can have angelic nondeterminism, but not demonic. An isotone element permits both kinds of nondeterminism.

2.2 A model

A predicate transformer S is a function $S : \wp(\Sigma) \to \wp(\Sigma)$, where Σ is any set. Programs can be modelled by predicate transformers according to a weakest precondition semantics [9, 5]: $S.q$ denotes those sets of states from which the execution of S is bound to terminate in q.

If $p, q \in \wp(\Sigma)$ and S satisfies $p \subseteq q \Rightarrow S.p \subseteq S.q$ then S is *isotone*. If S for any set I satisfies $S.(\bigcap_{i\in I} p_i) = \bigcap_{i\in I} S.p_i$ and $S.(\bigcup_{i\in I} p_i) = \bigcup_{i\in I} S.p_i$ it is *conjunctive* and *disjunctive*, respectively. There are three named predicate transformers $\mathsf{abort} = (\lambda q \bullet \emptyset)$, $\mathsf{magic} = (\lambda q \bullet \Sigma)$, and $\mathsf{skip} = (\lambda q \bullet q)$. A predicate transformer S *is refined by* T, written $S \sqsubseteq T$, if $(\forall q \in \wp(\Sigma) \bullet S.q \subseteq T.q)$. This paper deals with six operations on predicate transformers defined by

$$
\begin{aligned}
(S \sqcap T).q &= S.q \cap T.q \\
(S \sqcup T).q &= S.q \cup T.q \\
\neg S.q &= (S.q)^{\mathbf{C}} \\
(S;T).q &= S.(T.q) \\
S^\omega &= \mu.(\lambda X \bullet S; X \sqcap \mathsf{skip}) \\
S^\dagger &= \nu.(\lambda X \bullet S; X \sqcup \mathsf{skip})
\end{aligned}
$$

where $^{\mathbf{C}}$ is set complement, μ denotes the least fixpoint with respect to $\sqsubseteq$, and ν denotes the greatest.

That our isotony condition of the axiomatisation actually singles out the isotone predicate transformers is settled by the next lemma. Similarly it can be proved that our conjunctivity and disjunctivity conditions single out the conjunctive and disjunctive predicate transformers, respectively.

Lemma 1. *Let $S : \wp(\Sigma) \to \wp(\Sigma)$. Then S is isotone if and only if for all predicate transformers $T, U : \wp(\Sigma) \to \wp(\Sigma)$, if $T \sqsubseteq U$ then $S;T \sqsubseteq S;U$.*

Proof. If S is isotone, then clearly $T \sqsubseteq U \Rightarrow S;T \sqsubseteq S;U$. Assume now that for all predicate transformers U and T it holds that $T \sqsubseteq U \Rightarrow S;T \sqsubseteq S;U$. We show that this implies that S is isotone. Indeed, suppose that $p, q \in \wp(\Sigma)$ and $p \subseteq q$. Then construct two predicate transformers I and J such that for any $r \in \wp(\Sigma)$ it holds that $I.r = p$ and $J.r = q$. Since $p \subseteq q$, we then have that $I \sqsubseteq J$. By the assumption, this means that $S;I \sqsubseteq S;J$, that is $(\forall r \in \wp(\Sigma) \bullet S.I.r \subseteq S.J.r)$, or in other words $(\forall r \in \wp(\Sigma) \bullet S.p \subseteq S.q)$. Removing the idle quantifier, this says exactly that $S.p \subseteq S.q$. □

With the aid of the above lemma and the Knaster-Tarski theorem, it is easily verified that the set of all predicate transformers forms a model for the dndRAn with the interpretation of the operators given as above.

Proposition 1. *Let Ptran_Σ be the set of all predicate transformers over a set Σ. Then*

$$(\mathsf{Ptran}_\Sigma, \neg, \sqcap, \sqcup, ;, ^\omega, ^\dagger, \mathsf{magic}, \mathsf{abort}, \mathsf{skip})$$

is a dndRAn*, when the interpretation of the operators is given according to the above.*

2.3 What is going on?

The basic properties of the algebra differ from the algebras in [21, 20] in the fact that not all operators are isotone any longer and that some propositions are weakened.

The ; is not right isotone for all elements, but for isotone *elements* it is. In fact, the isotony condition on elements says exactly this. All other operators are isotone, except the negation operator which is antitone

$$x \sqsubseteq y \Rightarrow \neg y \sqsubseteq \neg x$$

This is to be kept in mind when doing derivations. The leapfrog property of strong iteration (strong angelic iteration is dual) is weakened, but the decomposition property is not. That is, if x and y are isotone, then

$$x(yx)^\omega \sqsubseteq (xy)^\omega x \tag{1}$$

is in general only a refinement, whereas

$$(x \sqcap y)^\omega = x^\omega (yx^\omega)^\omega \tag{2}$$

is always an equality. If x and y are conjunctive, then (1) can be strengthened to an equality [21].

3 Guards and assertions

This section extends the algebra with guards and assertions, shows that they can be defined in terms of each other, and provides an interpretation in the predicate-transformer model.

3.1 Definitions and properties

First, some notation. If an element of a dndRAn is both conjunctive and disjunctive, then we say that it is *functional*. A functional element thus permits no kind of nondeterminism.

Guards should be thought of as programs that check if some predicate holds, skip if that is the case, and otherwise a miracle occurs.

Definition 1. *An element g of a* dndRAn *is a* guard *if*

(1g) g is functional
(2g) g has a complement $\bar{g}$ satisfying
$$g\bar{g} = \top \quad \textit{and} \quad g \sqcap \bar{g} = 1$$
(3g) for any g' also satisfying (1g) and (2g) it holds that
$$gg' = g \sqcup g'$$

Assertions are similar to guards, 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.

Definition 2. *An element p is an* assertion *if*

(1a) p is functional
(2a) p has a complement $\bar{p}$ satisfying
$$p\bar{p} = \bot \quad \textit{and} \quad p \sqcup \bar{p} = 1$$
(3a) for any p' also satisfying (1a) and (2a) it holds that
$$pp' = p \sqcap p'$$

It is easily established that the guards and the assertions form Boolean algebras, since guards and assertions are closed under the operators $\sqcap, \sqcup$, and ;.

Proposition 2. *Let G be the set of guards and let A be the set of assertions of a* dndRAn*. Then*

$$(G, \sqcap, ;, \bar{}, 1, \top) \qquad \textit{and} \qquad (A, ;, \sqcup, \bar{}, \bot, 1)$$

are Boolean algebras.

From this we get the following useful fact, by verifying that $g\bot \sqcup 1$ is the unique complement of $\bar{g}$ in the sense of (2g).

Corollary 1. *For any guard g of a* dndRAn*, we have that $g = g\bot \sqcup 1$.*

We will also need the following lemma.

Lemma 2. *For any x in the carrier set of an* dndRAn *it holds that $x\bot$ and $x\top$ are functional.*

Proof. The first case is proved by

$$x\bot(y \sqcap z) = x\bot = x\bot \sqcap x\bot = x\bot y \sqcap x\bot z$$

and the other three cases are similar. □

With the aid of the previous lemma, the guard and assertion conditions yielding the following proposition are easily verified.

Proposition 3. *Let g be a guard and let p be an assertion in a* dndRAn*. Then*

$\bar{g}\bot \sqcap 1$	*is an assertion with the complement*	$g\bot \sqcap 1$, *and*
$\bar{p}\top \sqcup 1$	*is a guard with the complement*	$p\top \sqcup 1$

We can now prove the following the following theorem.

Theorem 1. *Any guard/assertion can be defined in terms of an assertion/guard.*

Proof. Let G be the set of guards and let A be the set of assertions in a dndRAn. We establish a bijection between the set of guards and the set of assertions. First define $^\circ : G \to A$ by

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

and $^\diamond : A \to G$ by

$$p^\diamond = \bar{p}\top \sqcup 1$$

Clearly, the mappings are well-defined by Proposition 3. Now, we show that they are surjective and each other's inverses, thus bijections. Take any $g \in G$. Then $(g^\circ)^\diamond = g\bot \sqcup 1 = g$, by Proposition 3 and Corollary 1. Thus $^\diamond$ is surjective and is the inverse function of $^\circ$. The case for $^\circ$ is analogous. □

This means that the set of guards and the set of assertions can be defined in terms of each other.

3.2 A predicate-transformer model

Consider the function $[\cdot] : \wp(\Sigma) \to (\wp(\Sigma) \to \wp(\Sigma))$ such that $[p].q = p^{\mathbf{C}} \cup q$, when $p, q \in \wp(\Sigma)$. For every element $p \in \wp(\Sigma)$ there is thus a predicate transformer $S_p : \wp(\Sigma) \to \wp(\Sigma),\ \ q \mapsto p^{\mathbf{C}} \cup q$. These predicate transformers are called *guards*. There is also a dual, an *assertion* and it is defined by $\{p\}.q = p \cap q$. Complement $^-$ is defined on guards and assertions by $\overline{[p]} = [p^{\mathbf{C}}]$ and $\overline{\{p\}} = \{p^{\mathbf{C}}\}$.

It follows directly from the definitions that the complement of any guard is also a guard, and moreover, that the guards are closed under the operators $\sqcap, \sqcup$, and ; defined in Section 2.2. If $[p]$ is any guard, it holds that

$$[p].(q_1 \cap q_2) = p^{\mathbf{C}} \cup (q_1 \cap q_2) = (p^{\mathbf{C}} \cup q_1) \cap (p^{\mathbf{C}} \cup q_2) = [p].q_1 \cap [p].q_2$$

for any $q_1, q_2 \in \wp(\Sigma)$. Similarly one can show that $[p].(q_1 \cup q_2) = [p].q_1 \cup [p].q_2$, so any guard is functional. Finally, it is easily verified that the axioms (g2) and (g3) also hold when the guards are interpreted in the predicate-transformer sense above. This means that guards in the predicate-transformer sense constitute a model for the guards in the abstract-algebraic sense. A similar argumentation shows that assertions in the predicate-transformer sense are a model for assertions in the abstract-algebraic sense.

4 Enabledness and Termination

We here introduce explicit definitions of the enabledness and the termination operators and show that, in this framework, the explicit definitions are equivalent to the implicit ones of [20].

4.1 Definitions

The enabledness operator ϵ is an operator that 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. It binds stronger than all the other operators and is a mapping from the set of isotone elements to the set of guards defined by

$$\epsilon x = x\bot \sqcup 1 \tag{3}$$

To see that the operator is well-defined, note that $x\bot \sqcup 1$ can be shown to be a guard with

$$\neg x\bot \sqcup 1 \tag{4}$$

as the complement.

In [20] the enabledness operator was defined implicitly similarly to the domain operator of Kleene algebra with domain (KAD) [8]. The next theorem shows that, in this framework, the implicit definition found in [20] is equivalent to the explicit definition above (in fact, as shown below only the two first axioms of the implicit axiomatisation are needed). Note that a similar move could not be done in KAD, since the explicit definition (3) relies on the lack of the right annihilation axiom for $\top$.

Theorem 2. *For any guard g and any isotone x in the carrier set of an* dndRAn*, ϵx satisfies*

$$\epsilon x x = x \tag{5}$$

$$g \sqsubseteq \epsilon(gx) \tag{6}$$

if and only if

$$\epsilon x = x\bot \sqcup 1 \tag{7}$$

Proof. The first two axioms of ϵ can be replaced by the equivalence

$$gx \sqsubseteq x \Leftrightarrow g \sqsubseteq \epsilon x \tag{8}$$

This can proved by reusing the proofs from [8]. Uniqueness of ϵx then follows from the principle of indirect equality and (8). Then it suffices to show that the right hand side of the explicit definition satisfies (5–6). This is verified by

$$\begin{array}{ll} & (x\bot \sqcup 1)x \sqsubseteq x \\ \Leftrightarrow & \{\text{axiom}\} \\ & x\bot x \sqcup x \sqsubseteq x \\ \Leftrightarrow & \{\text{axioms}\} \\ & x\bot \sqcup x \sqsubseteq x1 \sqcup x \\ \Leftarrow & \{\text{isotony}\} \\ & \bot \sqsubseteq 1 \\ \Leftrightarrow & \{\bot \text{ bottom element}\} \\ & \mathsf{True} \end{array}$$

and

$$\begin{array}{ll} & g \sqsubseteq gx\bot \sqcup 1 \\ \Leftrightarrow & \{\text{Corollary 1}\} \\ & g\bot \sqcup 1 \sqsubseteq gx\bot \sqcup 1 \\ \Leftrightarrow & \{\text{axiom}\} \\ & g\bot\bot \sqcup 1 \sqsubseteq gx\bot \sqcup 1 \\ \Leftarrow & \{\text{isotony}\} \\ & \bot \sqsubseteq x \\ \Leftrightarrow & \{\bot \text{ bottom element and left annihilator}\} \\ & \mathsf{True} \end{array}$$

which proves the proposition. □

Moreover, in contrast to the domain operator of [8], the compositionality property

$$\epsilon(xy) = \epsilon(x\epsilon y) \tag{9}$$

can be shown to always hold for the enabledness operator in a dndRAn (in [20] this was taken as an axiom of ϵ):

$$\begin{array}{ll} & \epsilon(xy) = \epsilon(x\epsilon y) \\ \Leftrightarrow & \{\text{definitions}\} \\ & xy\bot \sqcup 1 = x(y\bot \sqcup 1)\bot \sqcup 1 \\ \Leftrightarrow & \{\text{axiom}\} \\ & xy\bot \sqcup 1 = x(y\bot\bot \sqcup \bot) \sqcup 1 \\ \Leftrightarrow & \{\bot \text{ bottom element}\} \\ & xy\bot \sqcup 1 = xy\bot \sqcup 1 \\ \Leftrightarrow & \{\text{reflexivity}\} \\ & \mathsf{True} \end{array}$$

Using the explicit definition, the properties

$$\epsilon(x \sqcap y) = \epsilon x \sqcap \epsilon y \tag{10}$$
$$\epsilon(x \sqcup y) = \epsilon x \sqcup \epsilon y \tag{11}$$

can be proved by the calculations

$$\epsilon(x \sqcap y) = (x \sqcap y)\bot \sqcup 1 = (x\bot \sqcap y\bot) \sqcup 1 = (x\bot \sqcup 1) \sqcap (y\bot \sqcup 1) = \epsilon x \sqcap \epsilon y$$

and

$$\epsilon(x \sqcup y) = (x \sqcup y)\bot \sqcup 1 = x\bot \sqcup y\bot \sqcup 1 \sqcup 1 = x\bot \sqcup 1 \sqcup y\bot \sqcup 1 = \epsilon x \sqcup \epsilon y$$

respectively. From this, isotony of enabledness

$$x \sqsubseteq y \Rightarrow \epsilon x \sqsubseteq \epsilon y \tag{12}$$

easily follows.

The termination operator τ is a mapping from isotone elements to the set of assertions defined by

$$\tau x = x\top \sqcap 1$$

It binds equally strong as ϵ. The intuition is that the operator τ applied to a program denotes those states from which the program is guaranteed to terminate, that is, states from which it will not abort. Analogously to the enabledness operator, it can be shown that $x\top \sqcap 1$ is an assertion with complement $\neg x\top \sqcap 1$, so τ is well-defined. Moreover, using similar reasoning as above, it can also be shown that τ can equivalently be defined as

$$\tau x x = x \tag{13}$$
$$\tau(px) \sqsubseteq p \tag{14}$$

That τ satisfies the properties

$$\tau(x \sqcap y) = \tau \sqcap \tau y \tag{15}$$
$$\tau(x \sqcup y) = \tau \sqcup \tau y \tag{16}$$
$$x \sqsubseteq y \Rightarrow \tau x \sqsubseteq \tau y \tag{17}$$

can be proved as for enabledness.

4.2 A predicate-transformer model and a digression

In [5] the *miracle guard* is defined by $\neg(\bigcap_{q \in \wp(\Sigma)} S.q)$ and the *abortion guard* by $\bigcup_{q \in \wp(\Sigma)} S.q$. Intuitively, the miracle guard is a predicate that holds in a state $\sigma \in \Sigma$ if and only if the program S is guaranteed not to perform a miracle, that is S does not establish every postcondition starting in σ. The abortion guard holds in a state $\sigma \in \Sigma$ if and only if the program S will always terminate starting in σ, it will establish some postcondition when starting in σ. When S is isotone

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 $S.\Sigma$.

A predicate-transformer interpretation of ϵx is $[\neg S.\emptyset]$, when x is interpreted as the predicate transformer S. The termination operator τx is interpreted as $\{S.\Sigma\}$. The enabledness operator and the termination operator thus correspond to the miracle guard and the abortion guard of [5], respectively, but lifted to predicate-transformer level. That the interpretation is sound is seen by the fact that $[\neg S.\emptyset] = S; \mathsf{abort} \sqcup 1$ and $\{S.\Sigma\} = S; \mathsf{magic} \sqcap 1$.

The following is a slight digression. What we did above was to turn the miracle and abortion guards into a guard and an assertion (in the predicate-transformer and the abstract-algebraic sense), respectively, since predicate transformers make up our concrete carrier set in this model. There is, however, another way of lifting the miracle and the abortion guard to the predicate-transformer which is closer to their original definition. This is done by setting the miracle guard to be $\neg S\bot$ and the termination guard to be $S\top$. This interpretation does not, however, satisfy the the respective axioms of enabledness and termination, so the connection to KAD is lost. Nonetheless, in certain applications the possibility to work with miracle and abortion guard without turning them into a guard and an assertion could prove useful.

4.3 Expressing relations between programs

The enabledness and the termination operator can be used to express relations between programs. We list here some examples of the use of the first-mentioned operator. First note that $\overline{\epsilon x}$ is a guard that skips in those states where x is *disabled*.

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: $x = \overline{\epsilon y}x$. A program x *enables* y if y is enabled after having executed x: $x = x\epsilon y$. Similarly as above x *disables* y if $x = x\overline{\epsilon y}$. The exclusion condition will be used in the application of the next section.

5 A small application: action-system decomposition

Action systems comprise a formalism for reasoning about parallel programs [2, 4]. The intuition is that an *action system* $\mathsf{do}\, x_1 [\!] \ldots [\!] x_n\, \mathsf{od}$ is an iteration of a demonic choice $x_0 \sqcap \cdots \sqcap x_n$ between a fixed number of demonically nondeterministic *actions*, $x_0, \ldots, x_n$, that terminates when none of them are any longer enabled. In dndRAn, an action system can be expressed as $(x_0 \sqcap \cdots \sqcap x_n)^\omega \overline{\epsilon(x_0)} \ldots \overline{\epsilon(x_n)}$ where $x_0, \ldots, x_n$ are conjunctive. The actions are thus iterated, expressed with the strong iteration operator, until none of them is any longer enabled, expressed with the enabledness operator.

An action system can be decomposed so that the order of execution is clearly expressed: if x excludes y, i.e. $x = \overline{\epsilon y}x$, then

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

Note that $(xy^\omega\overline{\epsilon y})^\omega\overline{\epsilon(xy^\omega\overline{\epsilon y})}$ is of the form $z^\omega\overline{\epsilon z}$.

Action-system decomposition has been shown by Back and von Wright using predicate transformer reasoning [6]. We now prove this axiomatically. We begin by an outer derivation, collecting assumptions as needed:

$$
\begin{array}{ll}
 & (x \sqcap y)^\omega\overline{\epsilon x}\ \overline{\epsilon y} \\
= & \{(2)\} \\
 & y^\omega(xy^\omega)^\omega\overline{\epsilon x}\ \overline{\epsilon y} \\
= & \{\text{assumption}\} \\
 & y^\omega(\overline{\epsilon y}xy^\omega)^\omega\overline{\epsilon x}\ \overline{\epsilon y} \\
= & \{\text{guards Boolean algebra}\} \\
 & y^\omega(\overline{\epsilon y}xy^\omega)^\omega\overline{\epsilon y}\ \overline{\epsilon x} \\
= & \{\text{leapfrog, conjunctivity}\} \\
 & y^\omega\overline{\epsilon y}(xy^\omega\overline{\epsilon y})^\omega\ \overline{\epsilon x} \\
= & \{\textbf{collect:}\ \text{if}\ \epsilon x = \epsilon(xy^\omega\overline{\epsilon y})\} \\
 & y^\omega\overline{\epsilon y}(xy^\omega\overline{\epsilon y})^\omega\ \overline{\epsilon(xy^\omega\overline{\epsilon y})}
\end{array}
$$

The collected assumption is then, in turn, proved by two refinements. First we refine the left term into the right by (setting $z = y^\omega\overline{\epsilon y}$)

$$
\begin{array}{ll}
 & \epsilon x \sqsubseteq \epsilon(xz) \\
\Leftrightarrow & \{\text{definitions}\} \\
 & x\bot \sqcup 1 \sqsubseteq xz\bot \sqcup 1 \\
\Leftarrow & \{\text{isotony}\} \\
 & \bot \sqsubseteq z\bot \\
\Leftrightarrow & \{\bot\ \text{bottom element}\} \\
 & \mathsf{True}
\end{array}
$$

and then the right into the left by

$$
\begin{array}{ll}
 & \epsilon(xy^\omega\overline{\epsilon y}) \sqsubseteq \epsilon x \\
\Leftrightarrow & \{\text{definition}\} \\
 & xy^\omega\overline{\epsilon y}\bot \sqcup 1 \sqsubseteq x\bot \sqcup 1 \\
\Leftarrow & \{\text{isotony}\} \\
 & y^\omega\overline{\epsilon y}\bot \sqsubseteq \bot \\
\Leftarrow & \{\text{induction}\} \\
 & y\bot \sqcap \overline{\epsilon y}\bot \sqsubseteq \bot \\
\Leftrightarrow & \{\text{definition and (4)}\} \\
 & y\bot \sqcap (\neg y\bot \sqcup 1)\bot \sqsubseteq \bot \\
\Leftrightarrow & \{\text{axioms}\} \\
 & y\bot \sqcap (\neg y\bot \sqcup \bot) \sqsubseteq \bot \\
\Leftrightarrow & \{\bot\ \text{bottom element}\} \\
 & y\bot \sqcap \neg y\bot \sqsubseteq \bot \\
\Leftrightarrow & \{\text{axiom}\} \\
 & (y \sqcap \neg y)\bot \sqsubseteq \bot \\
\Leftrightarrow & \{\text{axioms}\} \\
 & \mathsf{True}
\end{array}
$$

Using the implicit definition without the negation operator, the first refinement of the assumption can also easily be proved [20], but the second refinement seems to require some additional axioms for the enabledness operator (see below).

6 Leaving out the negation

This section contains some remarks on a dually nondeterministic refinement algebra wihtout negation. The algebra was suggested by von Wright in [21, 22], but without the strong angelic iteration. The negation operator with its axiom is dropped and we strenghten the axioms so that all elements are isotone by adding the axioms $x(y \sqcap z) \sqsubseteq xy \sqcap xz$ and $xy \sqcup xz \sqsubseteq x(y \sqcup z)$. The structure over $(\sqcap, ;, \top, 1)$ is thus a left semiring [17]. Dropping the negation means that we no longer have a Boolean algebra, however we have a complete bounded distributive lattice over $(\sqcap, \sqcup, \bot, \top)$. The spelled out axiomatisation over the signature $(\sqcap, \sqcup, ;, ^\omega, ^\dagger, \bot, \top, 1)$ is thus given by the following:

$$
\begin{array}{ll}
x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z & x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z \\
x \sqcap y = y \sqcap x & x \sqcup y = y \sqcup x \\
x \sqcap \top = x & x \sqcup \bot = x \\
x \sqcap x = x & x \sqcup x = x \\
x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z) & x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z) \\
x(yz) = (xy)z & \\
1x = x & \\
x1 = x & \\
\top x = \top & \\
\bot x = \bot & \\
x(y \sqcap z) \sqsubseteq xy \sqcap xz & x(y \sqcup z) \sqsupseteq xy \sqcup xz \\
(x \sqcap y)z = xz \sqcap yz & (x \sqcup y)z = xz \sqcup yz \\
x^\omega = xx^\omega \sqcap 1 & x^\dagger = xx^\dagger \sqcup 1 \\
xz \sqcap y \sqsubseteq z \;\Rightarrow\; x^\omega y \sqsubseteq z & z \sqsubseteq xz \sqcup y \;\Rightarrow\; z \sqsubseteq x^\dagger y
\end{array}
$$

Since the isotone predicate transformers are closed under union, they constitute a predicate-transfomer model for the algebra under the interpretation given in Section 2.2.

By examining the proofs of Section 3, it is clear that the results regarding guards and assertions can be re-proved without the negation operator. On the other hand, it seems to us that the enabledness operator cannot be cast in the explicit form, since we cannot express the complement $\neg x\bot \sqcup 1$ of $x\bot \sqcup 1$ and this is needed for showing that $x\bot \sqcup 1$ actually is a guard. Analogously, the termination operator cannot be given an explicit definition either. Thus, the operators have to be axiomatised along the lines of [20]. The termination operator is axiomatized by

$$x = \tau x x \tag{18}$$

$$\tau(g^\circ x) \sqsubseteq g^\circ \tag{19}$$

$$\tau(x \tau y) = \tau(xy) \tag{20}$$

$$\tau(x \sqcap y) = \tau x \sqcap \tau y \tag{21}$$

and the enabledness operator by

$$\epsilon x x = x \tag{22}$$
$$g \sqsubseteq \epsilon(gx) \tag{23}$$
$$\epsilon(xy) = \epsilon(x\epsilon y) \tag{24}$$
$$\epsilon(x \sqcup y) = \epsilon x \sqcup \epsilon y \tag{25}$$

The last axiom for ϵ was not given in [20], since in that framework angelic choice is not even present. We conjecture that (21) and (25) are independent from the other axioms of their respective operators.

In [20] it is noted that to prove action-system decomposition an additional axiom for the enabledeness operator seems to be required: $\epsilon x\bot = x\bot$. The addition of the angelic choice operator does not seem to facilitate a proof. Due to this, the proof of action-system decomposition does not follow as neatly as when the negation operator is at hand. Dually, it is possible that $\tau x\top = x\top$ needs to be postulated for some specific purpose.

The dually nondeterminisitic refinement algebra without negation thus gives a cleaner treatment of iteration, but as a drawback the enabledness and the termination operators start crackling.

7 Concluding remarks

We have proposed a dually nondeterministic refinement algebra with a negation operator for reasoning about program refinement and applied it to proving a rather humble property of action systems. The negation operator facilitates useful explicit definitions of the enabledness and the termination operators and it is a powerful technical tool. It is, however, antitone, which perhaps makes the reasoning a bit more subtle. When dropping the negation operator, but keeping the angelic choice, guards and assertions can still be defined in terms of each other, whereas the enabledness and termination operators no longer can be given explicit definitions.

Finding more application areas of this refinement algebra is one of our intents. Applications that genuinely include angelic nondeterminism (here it only comes into play indirectly via the definition of enabledness) is a field where the algebra could be put to use. The strong angelic iteration and the termination operator beg for application. A systematic investigation striving towards a collection of calculational rules is yet to be done.

Acknowledgements. Thanks are due to Orieta Celiku, Peter Höfner, Linas Laibinis, Bernhard Möller, Ville Piirainen, Joakim von Wright and the anonymous referees who read an earlier version of this paper for stimulating discussions, helpful suggestions, and careful scrutiny.

References

1. R.J. Back. *Correctness Preserving Program Refinements: Proof Theory and Applications*, volume 131 of *Mathematical Centre Tracts*. Mathematical Centre, Amsterdam, 1980.
2. R.J. Back and R. Kurki-Suonio Decentralization of Process Nets with Centralized Control. In *2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, ACM, Montreal, Quebec, Canada, 1983.
3. R.J. Back and J. von Wright. Duality in specification languages: A lattice theoretical approach. Acta Informatica, 27(7), 1990.
4. R.J. Back and K. Sere. Stepwise refinement of action systems. Structured Programming, 12, 1991.
5. R.J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998.
6. R.J. Back and J. von Wright. Reasoning algebraically about loops. Acta Informatica, 36, 1999.
7. M. Broy and G. Nelson Adding Fair Choice to Dijkstra's Calculus. ACM Transactions on Programming Languages and Systems, Vol 16, NO 3, 1994.
8. J. Desharnais, B. Möller and G. Struth. Kleene algebra with domain. Technical Report 2003-7, Universität Augsburg, Institut für Informatik, 2003.
9. E.W. Dijkstra. *A Discipline of Programming*. Prentice-Hall International, 1976.
10. R.W. Floyd. Nondeterministic algorithms. Journal of the ACM, 14(4), 1967.
11. P.H. Gardiner and C.C. Morgan. Data refinement of predicate transformers. Theoretical Computer Science, 87(1), 1991.
12. D. Kozen A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. Inf. Comput. 110(2),1994.
13. D. Kozen *Automata and Computability*. Springer-Verlag, 1997.
14. D. Kozen. Kleene algebra with tests. ACM Transactions on Programming Languages and Systems, 19(3), 1997.
15. G. Nelson A Generalization of Dijkstra's Calculus. ACM Transactions on Programming Languages and Systems, 11(4), 1989.
16. C.C. Morgan. *Programming from Specifications* (2nd edition). Prentice-Hall, 1994.
17. B. Möller Lazy Kleene algebra. In D. Kozen (ed.): Mathematics of Program Construction, LNCS 3125, Springer, 2004.
18. Sampaio, A.C.A. *An Algebraic Approach To Compiler Design.* World Scientific, 1997.
19. K. Solin. An Outline of a Dually Nondeterministic Refinement Algebra with Negation. In Peter Mosses, John Power, and Monika Seisenberger (eds.): CALCO Young Researchers Workshop 2005, Selected Papers. Univ. of Wales, Swansea, Technical Report CSR 18-2005, 2005.
20. K. Solin and J. von Wright. Refinement Algebra Extended with Operators for Enabledness and Termination. TUCS Technical Report Nr 658, Turku Centre for Computer Science, Turku, Finland, 2005.
21. J. von Wright. From Kleene algebra to refinement algebra. In *Mathematics of Program Construction,* volume 2386 of *Lecture Notes in Computer Science*, Germany, Springer-Verlag, 2002.
22. J. von Wright. Towards a refinement algebra. Science of Computer Programming, 51, 2004.