Theoretical Computer Science 100 (1992) 365-383
Elsevier

Combining angels, demons and miracles in program specifications

R.J.R. Back

Åbo Akademi University, Department of Computer Science, Lemminkäisenkatu 14, SF-20520 Turku, Finland

J. von Wright

Swedish School of Economics and Business Education, Biblioteksgatan 16, SF-65100 Vasa, Finland

Communicated by J.W. de Bakker
Received December 1989
Revised November 1990

Abstract

Back, R.J.R., and J. von Wright, Combining angels, demons and miracles in program specifications, Theoretical Computer Science 100 (1992) 365-383.

The complete lattice of monotonic predicate transformers is interpreted as a command language with a weakest precondition semantics. This command lattice contains Dijkstra's guarded commands as well as miracles. It also permits unbounded nondeterminism and angelic nondeterminism. The language is divided into sublanguages using criteria of demonic and angelic nondeterminism, termination and absence of miracles. We investigate dualities between the sublanguages and how they can be generated from simple primitive commands. The notions of total correctness and refinement are generalized to the command lattice.

1. Introduction

The weakest precondition calculus of Dijkstra [10] identifies the meaning of a program statement with its weakest precondition predicate transformer. Dijkstra's "healthiness conditions" state that these predicate transformers for executable program statements are strict (they satisfy the "Law of Excluded Miracle"), monotonic, conjunctive and continuous.

Extensions to the language of guarded commands that drop some of the healthiness conditions have subsequently been used to allow treatment of specifications, parallel programs and data refinement. Back [1, 2] introduces weakest preconditions for specifications. He permits unbounded nondeterminism, thus dropping the continuity

requirement. Further, a refinement relation on statements is introduced, such that S is refined by S' iff

$$\forall Q.(\mathrm{wp}_S(Q) \Rightarrow \mathrm{wp}_{S'}(Q))$$

where wp_S is the weakest precondition predicate transformer for S. The refinement relation is a preorder, and $S \leqslant S'$ means that S' preserves the total correctness of S. The *refinement calculus*, based on this notion of refinement, gives a framework for the stepwise refinement of programs [1, 2, 4]. An initial specification S_0 is refined in small steps,

$$S_0 \leqslant S_1 \leqslant \cdots \leqslant S_n$$

where each step preserves the total correctness of the previous one. The transitivity of the refinement relation guarantees that the final program S_n satisfies the initial specification S_0.

The weakest precondition calculus was extended by de Bakker [9] to cover partial state transformers, i.e., *miraculous statements*. Miraculous statements are used in program refinements by Morgan [17] and Back [6]. The *angelic statement* of Back [5], used in nonfunctional data refinement, is not conjunctive but disjunctive. Thus, in going from a pure programming language to specification languages, most of the original healthiness conditions have been questioned, in order to gain expressive power and to develop calculi for program development. In this sense a specification language is truly more general than a programming language, for which all the original healthiness conditions are well motivated.

The conjunctivity condition reflects the view that the nondeterminism associated with the execution of a statement is *demonic*, i.e., in order for a computation to be successful, all possible execution paths must lead to a successful result. Dropping the conjunctivity condition means accepting other kinds of nondeterminism. If the conjunctivity condition is replaced with a disjunctivity condition, the nondeterminism is *angelic*, i.e., in order for a computation to be successful it is enough that there exists a possible successful execution path. Angelic nondeterminism in a weakest precondition semantics based on state transformers has been considered by Jacobs and Gries [16], and in another setting by Broy [8].

Hehner [13] and Morris [19] identify statements with weakest precondition predicate transformers, an approach which we follow in this paper. Thus we write $S(Q)$ rather than $\mathrm{wp}_S(Q)$. As a consequence, our theory does not permit reasoning about partial correctness. Morris [19] notes that the monotonic predicate transformers form a lattice, with the partial order corresponding to the refinement ordering of statements. Lattice-like operators on statements have been considered by Gardiner and Morgan in [11] and by Hoare and others in [12]. An algebraic approach to weakest preconditions is also investigated by Hesselink in [14, 15].

In an earlier paper [7] we define a specification language $\mathscr{C}$ within the complete lattice of monotonic predicate transformers, using only very simple primitive commands and functional (sequential) composition in addition to meets and joins. It

is shown that all monotonic predicate transformers can be generated in this language. A duality operator *dual* is introduced in $\mathscr{C}$ and the duality properties of the command lattice are investigated.

This paper extends the work in [7], focusing on the duality between angelic and demonic nondeterminism together with the duality between nontermination and miracles. We consider a number of sublanguages of the command lattice $\mathscr{C}$, showing how they are interrelated and how they can be constructed. We extend the completeness results from [7] to cover these sublanguages. We also generalize the notion of refinement to $\mathscr{C}$ and give an intuitive characterization of what the refinement relation means in $\mathscr{C}$.

The rest of the paper is organized as follows: Section 2 contains background material on lattices and predicate transformers. Section 3 decribes how the complete lattice of monotonic predicate transformers is interpreted as a command language $\mathscr{C}$. Section 4 describes some interesting sublanguages of $\mathscr{C}$ and their interrelations. In particular, we consider languages permitting nontermination and permitting demonic and/or angelic nondeterminism. We show that many sublanguages of $\mathscr{C}$ are complete lattices. The language of nonmiraculous conjunctive commands (i.e., the language where Dijkstra's guarded commands are embedded) is shown to have a more irregular structure. In Section 5, we show how the languages described in the preceding sections can be generated from very simple primitive commands (again with the language of nonmiraculous conjunctive commands as a notable exception). In Section 6 we give a characterization of the refinement relation in $\mathscr{C}$. Finally, we end with some concluding remarks in Section 7.

2. Prerequisites

2.1. Lattice theory

We assume that the concepts of partial orders and lattices (complete, distributive, boolean and atomic lattices) are familiar, as well as the weakest precondition technique of [10].

Function lattices. If L is a lattice and K any set, then the set of total functions from K to L is a lattice, with the *pointwise extension* of the partial order $\leqslant$ on L,

$$f \leqslant g \stackrel{\text{def}}{=} \forall x.(f(x) \leqslant g(x)).$$

The monotonic (order-preserving) functions from one lattice L to another lattice L' form a lattice, denoted $[L \to L']$. It is complete whenever L' is complete.

Proving orderings in a complete boolean lattice. The following important fact about complete boolean lattices will be used in many proofs: if x and y are arbitrary elements in L and $\neg y$ denotes the inverse of y, then

$$x \leqslant y \vee z \Leftrightarrow x \wedge \neg y \leqslant z. \tag{1}$$

Semilattices. Semilattices are partially ordered sets where finite meets or joins exist. A $\bigvee$-*semilattice* is a partially ordered set H such that $x \vee y$ (the least upper bound of $\{x, y\}$) exists for all x, y in H. We say that H is *complete* if $\bigvee H'$ exists for all nonempty subsets H' of H. The concept of $\bigwedge$-*semilattice* is defined dually.

2.2. Basic programming concepts

Let *Bool* be the complete lattice of truth values for a two-valued logic, $Bool = \{ff, tt\}$, ordered so that $ff < tt$ (the implication ordering).

Variables, values and states. We assume that *Var* is a nonempty countable set of *program variables* with typical list of distinct elements v (lists can be finite or infinite). Sometimes we will need to consider all the elements of *Var* as a list, which we will then denote V.

We further assume a nonempty set D of values, with typical list of elements d. The length of the list d is assumed to be clear from the context. A *state* is an assignment of a value to each variable, i.e., a (total) function from *Var* to D. We assume that there are no undefined values. The set of all states is called the *state space* and is denoted Σ. A typical element of Σ is denoted σ.

A (*semantic*) *substitution* in Σ is defined in the following way. The state σ with c (semantically) substituted for u, denoted $\sigma[c/u]$, is the state which differs from σ only in that it assigns the value c to the variable u. Substitutions of lists of values for lists of distinct variables are defined similarly.

Predicates. A *predicate* is a (total) function from Σ to *Bool*. We denote the set of all predicates *Pred*; typical predicates are denoted P and Q. Instead of $P(\sigma) = tt$ we often write just $P(\sigma)$, saying that *P holds in the state* σ.

Pred is a complete boolean lattice with the pointwise extended partial order from *Bool*:

$$P \leqslant Q \stackrel{\text{def}}{=} \forall \sigma.(P(\sigma) \leqslant Q(\sigma)).$$

Note that we can interpret $P \leqslant Q$ a "P implies Q (everywhere)" and that we allow infinite conjunction (meets) and disjunctions (joins) of predicates.

The bottom element of *Pred* is called *false* while the top element is called *true*. The inverse of a predicate P is $\neg P$ and $P \Rightarrow Q$ is an abbreviation for $\neg P \vee Q$. The lattice *Pred* is atomic. Its atoms are the one-state predicates b_σ, defined by

$$b_\sigma(\sigma') = tt \text{ iff } \sigma' = \sigma. \tag{2}$$

Semantic substitutions are extended to predicates, with $P[d/v]$ assigning the same truth value to σ as P assigns to $\sigma[d/v]$, i.e., $P[d/v](\sigma) = P(\sigma[d/v])$.

2.3. Predicate transformers

Predicate transformers are (total) functions from predicates to predicates. Predicate transformers are typically denoted S. The set of all predicate transformers is a

complete boolean lattice with the pointwise extension of the partial order on *Pred*:

$$S \leqslant S' \stackrel{\text{def}}{=} \forall Q.(S(Q) \leqslant S'(Q)).$$

Actually, we are only interested in the complete lattice of *monotonic predicate transformers*, $[Pred \to Pred]$. Its bottom element is the predicate transformer which maps every predicate to the predicate *false*, called *abort*. The top element, which maps every predicate to the predicate *true*, is called *magic*. Note that *abort* is the unit element of the join operator while *magic* is the unit element of the meet operator.

Functional composition of predicate transformers is called *sequential composition* and denoted by the symbol ";". The unit element of sequential composition is the identity predicate transformer *skip*.

3. The lattice of commands

We now define the command language $\mathscr{C}$. Our definition is semantic: we define every monotonic predicate transformer to be a command (thus $\mathscr{C} = [Pred \to Pred]$).

$\mathscr{C}$ is a *command lattice*, with the basic operators $\wedge$ (meet), $\vee$ (join) and ; (sequential composition). The bottom element of $\mathscr{C}$ is *abort*, and the top element is *magic*. Commands are given a weakest precondition interpretation. The meaning of a command S is described with respect to a final condition Q that it is intended to establish. Assume that the command S is executed in an initial state σ_0. Then $S(Q)$ is a predicate which holds in σ_0 if and only if the execution succeeds in establishing the postcondition Q (i.e., it reaches a final state where Q holds).

A meet of commands $\bigwedge S_i$ is interpreted as a *demonic choice* between the commands, i.e. it establishes a postcondition Q iff all commands S_i establish Q. Similarly, a join of commands $\bigvee S_i$ can be interpreted as an *angelic choice* between the commands, i.e., it establishes a postcondition Q iff one of the commands S_i establishes Q.

A sequential composition $S; S'$ is interpreted as executing the commands S and S' in sequence. However, if S fails (does not terminate) or it succeeds miraculously (see below), then S' is not executed, as it is never reached.

The *abort* command always fails; it never succeeds in establishing any postcondition. On the other hand, *magic* always succeeds. It even establishes the postcondition *false*, thus it is *miraculous*.

3.1. Characterizing properties

The command language introduces a number of new features into the weakest precondition approach which are not present in the original guarded command language of [10]. We will now characterize these features in somewhat more detail and identify a number of sublanguages of $\mathscr{C}$ based on these features.

Dijkstra originally proposed five healthiness conditions that every statement language would have to satisfy. Of these, only the monotonicity condition is satisfied

by all commands in $\mathscr{C}$; it is the defining characteristics for the predicate transformers considered to be commands. The fact that we permit arbitrary meets means that the assumption of bounded nondeterminism is not satisfied, the miraculous commands violate the "Law of Excluded Miracles" and the angelic choice violates the conjunctivity condition. We define a command S to be

($\bot$) *nonmiraculous* if $S(\mathit{false}) = \mathit{false}$,

($\top$) *always terminating* if $S(\mathit{true}) = \mathit{true}$,

($\wedge$) *conjunctive* if $S(\bigwedge_{i\in I} Q_i) = \bigwedge_{i\in I} S(Q_i)$
for any nonempty family $\{Q_i\}_{i\in I}$ of predicates,

($\vee$) *disjunctive* if $S(\bigvee_{i\in I} Q_i) = \bigvee_{i\in I} S(Q_i)$
for any nonempty family $\{Q_i\}_{i\in I}$ of predicates.

We call $S(\mathit{false})$ the *domain of miracles* and $S(\mathit{true})$ the *domain of termination* of S.

A command which is both nonmiraculous and always terminating is said to be *total* and a command which is both conjunctive and disjunctive is said to be *deterministic*. The definition of determinism is motivated by the following argument, showing a correspondence between deterministic commands and deterministic state transformers. A *deterministic state transformer* is a function from the state space Σ to the augmented state space $\Sigma_+ = \Sigma \cup \{\bot, \top\}$ where $\bot$ stands for nontermination and $\top$ for miraculous termination. For every deterministic command S we define the state transformer f_S by

$$f_S(\sigma) = \begin{cases} \top & \text{if } \sigma \in S(\mathit{false}), \\ \bot & \text{if } \sigma \notin S(\mathit{true}), \\ \bigcap \{Q \mid \sigma \in S(Q)\} & \text{otherwise,} \end{cases}$$

where $\sigma \in Q$ means that Q holds in the state σ. It is then straightforward to show that f_S is deterministic (i.e., that $f(\sigma)$ is always singleton). One can show that every deterministic state transformer corresponds to a unique deterministic command and that the following holds:

$$\sigma \in S(Q) \;\Leftrightarrow\; f_S(\sigma) \in Q,$$

i.e., $S(Q)$ holds in a state σ if and only if execution of S in the initial state σ can terminate only in one final state and Q holds in that final state. Thus execution of S can be intuitively interpreted as being deterministic.

It should be noted that by introducing the "miracle state" $\top$ we do not require that deterministic commands be nonmiraculous. In this respect our definition is different from the definition in e.g., [14].

Refinement of commands. Since the commands are given a weakest precondition interpretation, the partial order on $\mathscr{C}$ is the *refinement relation* introduced by Back [1, 2] and later used in [19, 11]. We define *total correctness* of a command S with respect to precondition P and postcondition Q, written $P[S]Q$, as follows:

$$P[S]Q \stackrel{\text{def}}{=} P \leqslant S(Q).$$

Then $S \leqslant S'$ holds if and only if S' preserves the total correctness of S. Thus S' is a refinement of S, in the sense that S' satisfies any specification that S satisfies. In the restricted context of conjunctive nonmiraculous commands and bounded nondeterminism the refinement order is essentially the same as the Smyth order [21], as pointed out by Back [3] and Plotkin [20].

3.2. *Extreme cases of command lattices*

We have assumed that both the set of variables *Var* and the set of variable values D are nonempty. Otherwise the state space Σ would be empty and *Pred* would contain only one predicate (thus we would have *false* = *true*). This would also mean that $\mathscr{C}$ would contain only one element. By assuming that D is nonempty we avoid this degenerate case.

If D contains exactly one element, then Σ also contains one element and *Pred* consists of the two distinct elements *false* and *true*. In this case $\mathscr{C}$ contains exactly the three distinct elements *abort*, *skip* and *magic*, with

$$abort < skip < magic.$$

If D contains two or more elements, then $\mathscr{C}$ is no longer totally ordered. In the rest of this paper, we assume that D contains at least two elements.

4. Sublanguages of $\mathscr{C}$

We return to the four characterizing properties defined in Section 3. These properties are independent of each other, thus there are sixteen different ways of combining them. We will use the symbols $\bot$, $\top$, $\wedge$, $\vee$ for the properties as indicated in Section 3.1 and we will index the name $\mathscr{C}$ with these symbols to denote a sublanguage where all commands are required to have the property in question. Thus, for example, $\mathscr{C}_{\vee}^{\bot}$ is the set of all disjunctive, nonmiraculous commands. Dijkstra's language of guarded commands is a subset of $\mathscr{C}_{\wedge}^{\bot}$, where conjunctivity and nonmiraculousness are required to hold.

Geometrically, we can view the sixteen languages as corners of a unit 4-dimensional hypercube. Each property corresponds to a dimension and the value 0 corresponds to not requiring the property while the value 1 corresponds to requiring the property. At the corner $(0, 0, 0, 0)$ we have the language $\mathscr{C}$, and at the corner $(1, 1, 1, 1)$ we have the language $\mathscr{C}_{\wedge\vee}^{\bot\top}$, containing all deterministic total commands.

4.1. *Symmetries between sublanguages*

There is a symmetry between the nonmiraculousness and the property of always terminating and another symmetry between conjunctivity and disjunctivity. We can describe this symmetry in tems of execution mechanisms. Thus, e.g., a command in $\mathscr{C}^{\bot}$ can be transformed into one in $\mathscr{C}^{\top}$ by replacing nontermination with miraculous

success. Similarly, a command in $\mathscr{C}_\wedge$ can be transformed into one in $\mathscr{C}_\vee$ by replacing demonic resolution of nondeterminism with angelic resolution.

Dual commands and symmetries. In [7], we defined the *dual* of an arbitrary command S by

$$dual(S)(Q) = \neg S(\neg Q)$$

It was shown that *dual* is a lattice-isomorphisms from $(\mathscr{C}, \leqslant)$ to $(\mathscr{C}, \geqslant)$ which is compositional in the sense that $dual(S; S') = dual(S); dual(S')$. Thus dualization interchanges miracles and nontermination on one hand and demonic and angelic nondeterminism on the other hand. We now investigate how the *dual* function can be divided into two parts, separating the two symmetries from each other.

The miracles-nontermination symmetry. We define the function *mt* (toggling miraculousness and termination properties) in the following way:

$$mt(S)(Q) = \neg S(false) \wedge S(Q) \vee \neg S(true),$$

(no parentheses are needed on the right-hand side because the expressions are associative in this case).

The following lemma shows that *mt* interchanges miracles and nontermination without affecting the mode of nondeterminism.

Lemma 1. *Let S be an arbitrary command in $\mathscr{C}$. Then*

(a) $mt(mt(S)) = S$.
(b) $mt(S)$ *is conjunctive iff* S *is conjunctive.*
(c) $mt(S)$ *is disjunctive iff* S *is disjunctive.*
(d) $mt(S)$ *is nonmiraculous iff* S *is always terminating.*
(e) $mt(S)$ *is always terminating iff* S *is nonmiraculous.*

Proof. We first prove (a). Let Q be an arbitrary predicate.

$mt(mt(S))(Q)$
$=$ [definition of *mt*]
$\neg mt(S)(false) \wedge mt(S)(Q) \vee \neg mt(S)(true)$
$=$ [definition of *mt*; properties of inverse in a boolean lattice]
$S(true) \wedge (\neg S(false) \wedge S(Q) \vee \neg S(true)) \vee S(false)$
$=$ [distributivity; property of inverse in a boolean lattice]
$(S(true) \wedge \neg S(false) \wedge S(Q)) \vee S(false)$
$=$ [by monotonicity, $S(Q) \leqslant S(true)$]
$(\neg S(false) \wedge S(Q)) \vee S(false)$
$=$ [distributivity; property of inverse in a boolean lattice]
$S(Q) \vee S(false)$
$=$ [monotonicity]
$S(Q)$.

Next we prove (b). Let $\{Q_i\}$ be a family of predicates.

S is conjunctive
$\Rightarrow$ [definition of conjunctivity]
$S(\bigwedge Q_i) = \bigwedge(S(Q_i))$
$\Rightarrow$ [applying same operations to both sides; distributivity in boolean lattice]
$\neg S(\mathit{false}) \wedge S(\bigwedge Q_i) \vee \neg S(\mathit{true}) = \bigwedge(\neg S(\mathit{false}) \wedge S(Q_i) \vee \neg S(\mathit{true}))$
$\Leftrightarrow$ [definition of *mt*, conjunctivity]
$mt(S)$ is conjunctive.

The implication in the opposite direction now follows by (a). The proof of (c) is exactly like the proof of (b). Now we prove (d) and (e) at the same time.

First assume that $S(\mathit{true}) = \mathit{true}$. Then rewriting shows that $mt(S)(\mathit{false}) = \mathit{false}$. In the same way it is shown that $S(\mathit{false}) = \mathit{false} \Rightarrow mt(S)(\mathit{true}) = \mathit{true}$. Both implications in the opposite direction now follow from (a). □

The junctivity symmetry. We define the function *cd* (toggling conjunctivity and disjunctivity) in the following way:

$$cd(S)(Q) = S(\mathit{true}) \wedge \neg S(\neg Q) \vee S(\mathit{false}),$$

(no parentheses are needed on the right-hand side because the expressions are associative in this case).

The following lemma shows that *cd* exchanges demonic and angelic nondeterminism in a command without affecting the termination or miracle properties.

Lemma 2. *Let S be an arbitrary command in* $\mathscr{C}$. *Then*
(a) $cd(cd(S)) = S$.
(b) $cd(S)$ *is disjunctive iff S is conjunctive.*
(c) $cd(S)$ *is conjunctive iff S is disjunctive.*
(d) $cd(S)$ *is nonmiraculous iff S is nonmiraculous.*
(e) $cd(S)$ *is always terminating iff S is always terminating.*

Proof. The proof is similar to the proof of Lemma 1. □

As an example let us assume that $S \in \mathscr{C}_\wedge^\perp$, i.e., S is a nonmiraculous conjunctive command (it could be a guarded command in the sense of Dijkstra [10]). Then $mt(S)$ is always terminating and conjunctive, i.e., $mt(S) \in \mathscr{C}_\wedge^\top$, and $cd(S)$ is nonmiraculous and disjunctive, i.e., $cd(S) \in \mathscr{C}_\vee^\perp$. Furthermore, $S \leqslant mt(S)$ and $S \leqslant cd(S)$, as the following lemma shows.

Lemma 3. *Let S be any command in* $\mathscr{C}$.
(a) $S \leqslant mt(S)$ *if and only if* $S \in \mathscr{C}^\perp$.
(b) $mt(S) \leqslant S$ *if and only if* $S \in \mathscr{C}^\top$.
(c) *If* $S \in \mathscr{C}_\wedge$ *then* $S \leqslant cd(S)$.
(d) *If* $S \in \mathscr{C}_\vee$ *then* $cd(S) \leqslant S$.

Proof. We first prove (a). For an arbitrary $Q \in Pred$,

$$
\begin{aligned}
& S \leqslant mt(S) \\
\Leftrightarrow\ & [\text{definition of } mt] \\
& \forall Q.(S(Q) \leqslant \neg S(false) \wedge S(Q) \vee \neg S(true)) \\
\Leftrightarrow\ & [(1)] \\
& \forall Q.(S(Q) \wedge S(true) \leqslant \neg S(false) \wedge S(Q)) \\
\Leftrightarrow\ & [\text{monotonicity}] \\
& \forall Q.(S(Q) \leqslant \neg S(false) \wedge S(Q)) \\
\Leftrightarrow\ & [\text{properties of complete lattices}] \\
& \forall Q.(S(Q) \leqslant \neg S(false)) \\
\Leftrightarrow\ & [(1) \text{ with } z = false] \\
& \forall Q.(S(Q) \wedge S(false) = false) \\
\Leftrightarrow\ & [\text{properties of complete lattices}] \\
& S(false) = false.
\end{aligned}
$$

The proof of (b) is similar. To prove (c), assume that S is conjunctive and let Q be an arbitrary predicate. Then

$$
\begin{aligned}
& S(Q) \wedge \neg cd(S)(Q) \\
=\ & [\text{definition of } cd] \\
& S(Q) \wedge (\neg S(true) \vee S\neg Q) \wedge \neg S(false)) \\
=\ & [\text{distributivity}] \\
& (S(Q) \wedge \neg S(true)) \vee (S(Q) \wedge S(\neg Q) \wedge \neg S(false)) \\
=\ & [(1);\ S(Q) \leqslant S(true)] \\
& S(Q) \wedge S(\neg Q) \wedge \neg S(false) \\
=\ & [S \text{ conjunctive; } Pred \text{ is a boolean lattice}] \\
& false.
\end{aligned}
$$

Thus, by (1) with $z = false$ it follows that $S(Q) \leqslant cd(S)(Q)$. Finally, (d) follows from (c) and Lemma 2. □

From Lemma 3 we see that $mt(S) = S$ if and only if S is total. We also see that $cd(S) = S$ if S is deterministic. However, the following example shows that S need not be deterministic even if $cd(S) = S$. Let S_1, S_2 and S_3 be arbitrary commands. Then $conj(S) = S$ holds for the command

$$S = (S_1 \wedge S_2) \vee (S_1 \wedge S_3) \vee (S_2 \wedge S_3).$$

However, S is not deterministic if S_1, S_2 and S_3 are the commands $x := 0$, $x := 1$ and $x := 2$.

Symmetries and duals. The *mt* and *cd* functions together make up the *dual* function, in the following way.

Lemma 4. $dual = mt \circ cd = cd \circ mt$.

Proof. We show that $mt(cd(S))(Q) = \neg S(\neg Q)$.

$mt(cd(S))(Q)$
$=$ [definitions]
$\neg S(false) \wedge (S(true) \wedge \neg S(\neg Q) \vee S(false)) \vee \neg S(true)$
$=$ [distributivity]
$(\neg S(false) \wedge S(true) \wedge \neg S(\neg Q)) \vee (\neg S(false) \wedge S(false)) \vee \neg S(true)$
$=$ [$\neg S(\neg Q) \leqslant \neg S(false)$; properties of inverses]
$(S(true) \wedge \neg S(\neg Q)) \vee \neg S(true)$
$=$ [distributivity; properties of inverses]
$\neg S(\neg Q) \vee \neg S(true)$
$=$ [monotonicity]
$\neg S(\neg Q)$.

In the same way it is shown that $cd(mt(S))(Q) = \neg S(\neg Q)$. □

From the above lemmas the following result follows.

Theorem 1. *The diagrams in Fig.* 1 *and Fig.* 2 *commute. In particular, mt, cd and dual are bijections on* $\mathscr{C}$.

The functions *mt* and *cd* are not compositional (i.e., it is generally not possible to compute e.g. $mt(S \wedge S')$, given only $mt(S)$ and $mt(S')$). This limits their usefulness when reasoning about the symmetries between sublanguages of $\mathscr{C}$.

4.2. Self-dual commands and determinism

We say that a command is *self-dual* if $S = dual(S)$. The concept of self-dualism is close to the concepts of determinism and totality.

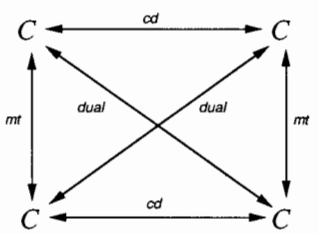

Fig. 1. Commuting diagram for $\mathscr{C}$.

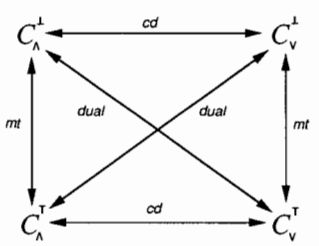

Fig. 2. Commuting diagram for sublanguages.

Lemma 5. *Let S be an arbitrary command in $\mathscr{C}$. Then*

$$S = dual(S) \Leftrightarrow S = mt(S) \wedge S = cd(S).$$

Proof. First we note that

$$S \leqslant cd(S) \Leftrightarrow \forall Q.(S(Q) \wedge S(\neg Q) = S(false)) \tag{3}$$

(this is easily proved in the same way as Lemma 3(a)). We now have the following:

$S \leqslant dual(S)$
$\Leftrightarrow$ [definitions]
$\forall Q.(S(Q) \leqslant \neg S(\neg Q))$
$\Leftrightarrow$ [(1)]
$\forall Q.(S(Q) \wedge S(\neg Q) = false)$
$\Leftrightarrow$ [choosing $Q = false$]
$S(false) = false \wedge \forall Q.(S(Q) \wedge S(\neg Q) = S(false))$
$\Leftrightarrow$ [Lemma 3(a), (3)]
$S \leqslant mt(S) \wedge S \leqslant cd(S).$

The rest of the proof now follows by symmetry. □

From Lemmas 3 and 5 we can see that if S is deterministic and total then S is self-dual. We also see that self-dual commands are always total. However, the example following Lemma 3 shows that self-dual commands need not be deterministic.

4.3. Interesting sublanguages and their properties

We are mainly interested in command languages that permit nontermination and nondeterminism. Thus the sublanguages of main interest are the following:

(a) $\mathscr{C}$, containing all commands,
(b) $\mathscr{C}^{\perp}$, the nonmiraculous commands,
(c) $\mathscr{C}_{\wedge}$, the conjunctive commands,
(d) $\mathscr{C}_{\vee}$, the disjunctive commands,
(e) $\mathscr{C}^{\perp}_{\wedge}$, the nonmiraculous conjunctive commands, and
(f) $\mathscr{C}^{\perp}_{\vee}$, the nonmiraculous disjunctive commands.

The following theorem gives a general description of these sublanguages of $\mathscr{C}$.

Theorem 2. *The six interesting sublanguages of $\mathscr{C}$ can be characterized as follows:*

(a) *$\mathscr{C}$ is a complete lattice with bottom element abort and top element magic.*

(b) *$\mathscr{C}^{\perp}$ is a complete lattice where all nonempty meets and joins give the same result as in $\mathscr{C}$. It has bottom element abort and top element serve, which maps false to false and all other predicates to true.*

(c) *$\mathscr{C}_{\wedge}$ is a complete lattice where all meets give the same result as in $\mathscr{C}$, but joins generally do not. It has bottom element abort and top element magic.*

(d) *$\mathscr{C}_{\vee}$ is a complete lattice where all joins give the same result as in $\mathscr{C}$, but meets generally do not. It has bottom element abort and top element magic.*

(e) $\mathscr{C}_{\wedge}^{\perp}$ *is a complete* $\bigwedge$*-semilattice where all meets give the same result as in* $\mathscr{C}$*. It has the least element abort and its maximal elements are all the deterministic total commands.*

(f) $\mathscr{C}_{\vee}^{\perp}$ *is a complete lattice where all joins give the same result as in* $\mathscr{C}$*, but meets generally do not. It has bottom element abort and top element serve.*

Proof. Parts (a) and (b) are proved in [7]. Since meets in $\mathscr{C}$ preserve conjunctivity it is obvious that all nonempty meets in $\mathscr{C}_{\wedge}$ exist and that they give the same results as in $\mathscr{C}$. Both *abort* and *magic* are conjunctive, thus they must be the bottom and top elements of $\mathscr{C}_{\wedge}$, respectively. Finally, since any family $\{S_i\}$ of conjunctive commands has at least one conjunctive upper bound (*magic*), we can form the meet (in $\mathscr{C}$) of all conjunctive upper bounds which is a least conjunctive upper bound, i.e., a least upper bound in $\mathscr{C}_{\wedge}$. Thus (c) is proved. The proof of (d) is dual to the proof of (c).

Since meets in $\mathscr{C}$ preserve both strictness with respect to *false* and conjunctivity, nonempty meets in $\mathscr{C}_{\wedge}^{\perp}$ give the same results as in $\mathscr{C}$. Since *abort* is nonmiraculous and conjunctive, it must be the least element of $\mathscr{C}_{\wedge}^{\perp}$. The formal proof of the fact that the maximal elements of $\mathscr{C}_{\wedge}^{\perp}$ are exactly the commands in $\mathscr{C}_{\wedge\vee}^{\perp\top}$ involves reasoning about the atoms of *Pred*. Here we give the following informal argument: From the refinement calculus [2, 4, 18] it is known that $S < S'$ holds in $\mathscr{C}_{\wedge}^{\perp}$ if S' is more terminating or more deterministic than S. However, since the commands in $\mathscr{C}_{\wedge\vee}^{\perp\top}$ are always terminating and deterministic we cannot find any command in $\mathscr{C}_{\wedge}^{\perp}$ that is more terminating or more deterministic. Since we assume that the value set D contains at least two elements, there is more than one maximal element. Thus (e) is proved.

Finally, the proof of (f) is similar to the proof of (d). □

Note that duality gives similar results for $\mathscr{C}^{\top}$ as in (b), for $\mathscr{C}_{\vee}^{\top}$ as in (e) and for $\mathscr{C}_{\wedge}^{\top}$ as in (f).

Theorem 2 shows that $\mathscr{C}_{\wedge}^{\perp}$ differs from the other languages; it is irregular, in the sense that it is not a lattice. It is interesting to note that Dijkstra's guarded commands and other similar languages not permitting miracles are a subset of $\mathscr{C}_{\wedge}^{\perp}$, the only one of the sublanguages considered here that is irregular. Adding miracles extends the languages into $\mathscr{C}_{\wedge}$, which is a complete lattice. This can be considered an argument for dropping Dijkstra's "Law of Excluded Miracle": it gives the language a more regular mathematical structure. In the next section we will show that the language $\mathscr{C}_{\wedge}^{\perp}$ differs from the other languages in another way: it is not as easily constructible from simple primitives.

5. Constructing command languages

In this section we show how $\mathscr{C}$ and some of its sublanguages can be constructed using simple primitive commands in addition to the constructors meet, join and

sequential composition. We shall consider those sublanguages that were described in the preceding section, i.e., those that permit nontermination and nondeterminism. The primitive commands that we introduce intuitively represent the two most primitive possible operations that imperative programs are built from: assigning a value to a variable, and testing whether a variable has a certain value.

5.1. Construction of $\mathscr{C}$

We define three primitive commands, the *substitution command* $\langle d/v\rangle$, the *strict test command* $\{v=d\}$ and the *miraculous test command* $[v=d]$ (where v is any list of distinct variables and d a list of values of the same length as v), as in [7].

The primitive commands have the following semantics:

$$\langle d/v\rangle(Q) \stackrel{\text{def}}{=} Q[d/v],$$

$$\{v=d\}(Q) \stackrel{\text{def}}{=} (v=d)\wedge Q,$$

$$[v=d](Q) \stackrel{\text{def}}{=} (v=d)\Rightarrow Q,$$

(thus the miraculous test command is the dual of the strict test command while the substitution command is its own dual).

The substitution command $\langle d/v\rangle$ assigns values d to the variables v, leaving the rest of the state unchanged. The strict test $\{v=d\}$ acts as *skip* if $v=d$ holds, otherwise it aborts. The miraculous test $[v=d]$, which is the dual of $\{v=d\}$, also acts as *skip* if $v=d$ but it succeeds miraculously otherwise. We permit empty lists in the definitions; $\langle\varepsilon/\varepsilon\rangle=\{\varepsilon=\varepsilon\}=[\varepsilon=\varepsilon]=$ *skip* where ε is the empty list.

Applying the command constructors of $\mathscr{C}$ (sequential composition, meet and join) to the primitive commands defined above, we can generate the whole of $\mathscr{C}$. This is the completeness theorem for $\mathscr{C}$, proved in [7].

Theorem 3. *Every command in $\mathscr{C}$ can be generated using the primitive commands $\langle d/v\rangle$, $\{v=d\}$ and $[v=d]$ and the constructors meet, join and sequential composition.*

5.2. Construction of $\mathscr{C}^{\perp}$, $\mathscr{C}_{\wedge}$, $\mathscr{C}_{\vee}$ and $\mathscr{C}_{\wedge}^{\perp}$

The primitive commands and constructors of $\mathscr{C}$ are connected with the four properties that characterize the sublanguages of $\mathscr{C}$. The command $\{v=d\}$ is possibly nonterminating, violating property $(\top)$ while the command $[v=d]$ is possibly miraculous, violating property $(\perp)$. Joins can introduce angelic nondeterminism, violating property $(\wedge)$ and meets can introduce demonic nondeterminism, violating property $(\vee)$. The substitution command $\langle d/v\rangle$ and the sequential composition are both neutral in the sense that they preserve all four characterizing properties.

We now investigate to what extent the interesting sublanguages of $\mathscr{C}$ can be constructed using those primitive commands and constructors that fit into the sublanguage in question. It turns out that most languages are constructible in this way; the only exception is the language $\mathscr{C}_{\wedge}^{\perp}$, the one that is not a lattice.

In [7] we show that $\mathscr{C}^{\perp}$ can be generated in the same way as $\mathscr{C}$ by dropping the primitive command $[v = d]$ which introduces miracles.

Theorem 4. *Every command in $\mathscr{C}^{\perp}$ can be generated using the primitive commands $\langle d/v \rangle$ and $[v = d]$ and the constructors meet, join and sequential composition.*

We now show that similar results hold for $\mathscr{C}_{\wedge}$ and $\mathscr{C}_{\vee}$.

Theorem 5. *Every command in $\mathscr{C}_{\vee}$ can be generated using the primitive commands $\langle d/v \rangle$, $\{v = d\}$ and $[v = d]$ and the constructors join and sequential composition.*

Proof. One can show that every command S in $\mathscr{C}_{\vee}$ can be written as

$$S = \bigvee_{\sigma \in \Sigma} \left(\left(\bigvee_{d : S(b_\sigma)} \{V = d\} \right); \langle \sigma(V)/V \rangle \right) \vee \left(\left(\bigvee_{d : S(false)} \{V = d\} \right); \bigvee_{d} [V = d] \right) \tag{4}$$

where the notation $\bigvee_{d:P} S_d$ is used for the qualified join of commands $\bigvee (d : P[d/v] = true : S_d)$. □

Since the *dual* function is compositional, we have a dual construction of all commands in $\mathscr{C}_{\wedge}$.

Corollary 1. *Every command in $\mathscr{C}_{\wedge}$ can be generated using the primitive commands $\langle d/v \rangle$, $\{v = d\}$ and $[v = d]$ and the constructors meet and sequential composition.*

Furthermore, dropping the second disjunct in the construction (4) forces strictness with respect to *false*, giving the following corollary.

Corollary 2. *Every command in $\mathscr{C}_{\vee}^{\perp}$ can be generated using the primitive commands $\langle d/v \rangle$ and $\{v = d\}$ and the constructors join and sequential composition.*

Thus we have shown that $\mathscr{C}^{\perp}$, $\mathscr{C}_{\wedge}$, $\mathscr{C}_{\vee}$ and $\mathscr{C}_{\vee}^{\perp}$ can all be constructed by dropping from the construction of $\mathscr{C}$ the constructs that introduce the unwanted properties. The result for $\mathscr{C}_{\vee}$ also show that every disjunctive command can be constructed using only deterministic components and angelic choice. Thus we can interpret every disjunctive command intuitively as an *angelic command.* Dually, every conjunctive command can be interpreted as a *demonic command.*

Summary of constructions. We summarize the construction of this section in Table 1, showing what is sufficient to construct the sublanguages of $\mathscr{C}$ considered so far.

Table 1

language	$\langle d/v\rangle$	$\{v=d\}$	$[v=d]$	$\wedge$	$\vee$	;
$\mathscr{C}$	×	×	×	×	×	×
$\mathscr{C}^-$	×	×		×	×	×
$\mathscr{C}_\wedge$	×	×	×	×		×
$\mathscr{C}_\vee$	×	×	×		×	×
$\mathscr{C}_\vee^\perp$	×	×			×	×

From the properties of the *dual* function it follows that dual languages have dual constructions. Thus, e.g. $\mathscr{C}_\wedge^\top$ can be constructed using the primitive commands $\langle d/v\rangle$ and $[v=d]$ in addition to meet and sequential composition, since $\mathscr{C}_\wedge^\top$ is the dual of $\mathscr{C}_\vee^\perp$. However, as noted above, we will not further investigate languages that exclude nontermination.

5.3. Constructing $\mathscr{C}_\wedge^\perp$

The language $\mathscr{C}_\wedge^\perp$ is different from those considered above, as it is not a lattice but only a semilattice. One may still ask if every command in $\mathscr{C}_\wedge^\perp$ can be constructed using the corresponding primitives and constructors from $\mathscr{C}$, i.e., $\{v=d\}$, $\langle d/v\rangle$, ";" and "$\wedge$". The answer is negative, as the following theorem shows.

Theorem 6. *If the value set D contains two or more elements then there are commands in $\mathscr{C}_\wedge^\perp$ that cannot be constructed using only the primitive commands $\{v=d\}$ and $\langle d/v\rangle$ and the constructors ";" and "$\wedge$".*

Proof. Let σ_1 and σ_2 be two states that differ on every variable in *Var*. Then b_{σ_1} and b_{σ_2} (as defined by (2)) are two atoms of *Pred*. We now define a command S to have property (J) if

$$S(b_{\sigma_1}) = \mathit{false} \quad \text{or} \quad S(b_{\sigma_2}) = \mathit{false} \quad \text{or} \quad S = \mathit{skip}. \tag{J}$$

One can then show by structural induction that all commands have property (J). Now let S be the command defined by

$$S(Q) = \begin{cases} \mathit{true} & \text{if } Q \geqslant b_{\sigma_1} \text{ and } Q \geqslant b_{\sigma_2}, \\ b_{\sigma_2} & \text{if } Q \geqslant b_{\sigma_1} \text{ and } Q \not\geqslant b_{\sigma_2}, \\ b_{\sigma_1} & \text{if } Q \not\geqslant b_{\sigma_1} \text{ and } Q \geqslant b_{\sigma_2}, \\ \mathit{false} & \text{if } Q \not\geqslant b_{\sigma_1} \text{ and } Q \not\geqslant b_{\sigma_2}. \end{cases}$$

S is conjunctive and strict with respect to *false* but it does not have property (J), hence it cannot be constructed. □

Thus we have shown that the irregular (see Theorem 2) language $\mathscr{C}_\wedge^\perp$ is not constructible in the same way as the other sublanguages of $\mathscr{C}$ considered previously. The problem is the lack of means to construct conditional commands (e.g., if-else commands). To construct these we need to combine strict test commands with joins or to combine miraculous tests with meets.

6. Characterizing refinement in the command languages

In the traditional context of nonmiraculous conjunctive commands, a refinement $S \leqslant S'$ holds when S' decreases the domain of nontermination of S or S' decreases the nondeterminism of S. We shall now generalize this characterization of refinement to arbitrary commands in $\mathscr{C}$.

We first note the following rules of refinement in $\mathscr{C}$.

Lemma 6. *Let v be an arbitrary list of variables, d a list of values of the same length as v, and let $\{S_i\}_{i \in I}$ be a family of commands in $\mathscr{C}$. Then*

(a) $\{v = d\} \leqslant \langle d/v \rangle \leqslant [v = d]$,

(b) $abort \leqslant \{v = d\} \leqslant skip \leqslant [v = d] \leqslant magic$,

(c) $\bigwedge_{i \in I} S_i \leqslant \bigwedge_{i \in I'} S_i$ *when* $I' \subseteq I$,

(d) $\bigvee_{i \in I'} S_i \leqslant \bigvee_{i \in I} S_i$ *when* $I' \subseteq I$.

Proof. To prove (a) we have to show that

$$(v = d) \wedge Q \leqslant Q[d/v] \leqslant (v = d) \Rightarrow Q$$

holds for all predicates Q. This follows from the one-point rule of predicate calculus. To prove (b) we have to show that

$$false \leqslant (v = d) \wedge Q \leqslant Q \leqslant (v = d) \Rightarrow Q \leqslant true$$

which follows from the basic properties of the lattice *Pred*. Finally, (c) and (d) are obvious properties in any complete lattice. □

Cases (a) and (b) of Lemma 6 are examples of refinement by decreasing nontermination and increasing miracles in commands. Cases (c) and (d) are examples of refinement by decreasing the demonic nondeterminism and increasing the angelic nondeterminism of commands.

The command constructors (meet, join and sequential composition) are monotonic with respect to subcommand replacement, as shown in [7]. Thus, applying the rules given in Lemma 6 to any subcomponent of a command S yields a refinement of S. Also, any refinement $S \leqslant S'$ can be described as a case of the rule of Lemma 6(c), in the trivial sense of dropping the first conjunct of $S \wedge S'$. Thus we can give the following general characterization of refinement in $\mathscr{C}$ and all its sublanguages: A command S is refined by another command S' if

(a) S' decreases the domain of nontermination of S, or

(b) S' increases the domain of miracles of S, or

(c) S' decreases the demonic nondeterminism of S, or

(d) S' increases the angelic nondeterminism of S.

Returning to the results in Lemma 3, we see that they verify this characterization of refinement.

7. Conclusion

We have considered the complete lattice of monotonic predicate transformers $\mathscr{C}$ interpreting it as a command language. Using the properties of nontermination, miraculousness, conjunctivity and disjunctivity, we have defined a number of sublanguages of $\mathscr{C}$. The *dual* function expresses a symmetry between sublanguages, interchanging nontermination and miraculous success and also interchanging demonic and angelic nondeterminism. We divided the *dual* function into two components, the functions *mt* (interchanging nontermination and miraculous success) and the function *cd* (interchanging demonic and angelic nondeterminism). We considered a number of interesting sublanguages (i.e., sublanguages permitting nontermination and nondeterminism) of $\mathscr{C}$. It was shown that they are complete lattices and that they can be constructed using only very simple primitive commands as building blocks and the lattice operators and functional composition as constructors. The language of nonmiraculous conjunctive commands ($\mathscr{C}^{\perp}_{\wedge}$) was shown to be an exception; it is not a lattice and it is not constructible in the same simple way as the other languages without introducing some way of expressing conditional composition. This is interesting, considering that Dijkstra's language of guarded commands is a subset of $\mathscr{C}^{\perp}_{\wedge}$. Finally, we extended the notion of refinement between commands to the command language $\mathscr{C}$, and a general characterization of refinement was given.

Since we identify commands with their weakest precondition predicate transformers, our framework does not permit reasoning about programs that are only partially correct. This is a deliberate choice; we are working within the framework of the refinement calculus which is a calculus of total correctness. It is possible to build a similar theory for partial correctness, identifying commands with their weakest liberal precondition predicate transformers. It is also possible to combine the two theories, identifying each command S with the pair $(\mathrm{wlp}_S, \mathrm{wp}_S)$. This is left as a subject for further study.

Acknowledgment

We thank the anonymous referee for many helpful comments and suggestions. The work reported here was supported by the Finsoft III program sponsored by the Technology Development Centre of Finland.

References

[1] R.J.R. Back, On the correctness of refinement in program development, PhD. thesis Report A-1978-4, Department of Computer Science, University of Helsinki, 1978.

[2] R.J.R. Back, Correctness preserving program refinements: proof theory and applications, *Math. Center Tracts* **131** (1980).

[3] R.J.R. Back, On correct refinement of programs, *J. Comput. System Sci.* **23** (1) (1981) 49–68.
[4] R.J.R. Back, A calculus of refinements for program derivations, *Acta Inform.* **25** (1988) 593–624.
[5] R.J.R. Back, Changing data representation in the refinement calculus, in: *Proc. 21st Hawaii Internat. Conf. on System Sciences*, January 1989.
[6] R.J.R. Back, Refining atomicity in parallel algorithms, in: *Proc. PARLE Conf. on Parallel Architectures and Languages Europe*, Eindhoven, Netherlands, June 1989.
[7] R.J.R. Back and J. von Wright, Duality in specification languages: a lattice-theoretical approach, *Acta Inform.* **27** (1990) 583–625.
[8] M. Broy, A theory for nondeterminism, parallellism, communications and concurrency. *Theoret. Comput. Sci.* **46** (1986) 1–61.
[9] J.W. de Bakker, *Mathematical Theory of Program Correctness* (Prentice-Hall, Englewood Cliffs, 1980).
[10] E.W. Dijkstra, *A Discipline of Programming* (Prentice-Hall International, London, 1976).
[11] P.H. Gardiner and C.C. Morgan, Data refinement of predicate transformers, manuscript, 1988, (to appear in Theoretical Computer Science).
[12] I.J. Hayes, C.A.R. Hoare, A.W. Roscoe, Jifeng He, C.C. Morgan, J.M. Spivey, J.W. Sanders, I.H. Sorensen and A. Sufrin, Laws of programming, *Comm. ACM* **30** (8) (1987) 672–686.
[13] E. Hehner, *The Logic of Programming* (Prentice-Hall, Englewood Cliffs, 1984).
[14] W.H. Hesselink, An algebraic calculus of commands, Report CS 8808, Department of Mathematics and Computer Science, University of Groningen, 1988.
[15] W.H. Hesselink, Command algebras, recursion and program transformation, *Formal Aspects of Computing* **2**(1) (1990) 60–104.
[16] D. Jacobs and D. Gries, General correctness: a unification of partial and total correctness, *Acta Inform.* **22** (1985) 67–83.
[17] C.C. Morgan, The specification statement, *ACM Trans. Programming Languages Systems* **10** (3) (1988) 403–419.
[18] C.C. Morgan, Types and invariants in the refinement calculus, in: *Mathematics of Program Construction*, Lecture Notes in Computer Science, Vol. 375 (Springer, Berlin, 1989).
[19] J.M. Morris, A theoretical basis for stepwise refinement and the programming calculus, *Sci. Comput. Programming* **9** (1987) 287–306.
[20] G.D. Plotkin, Dijkstra's weakest preconditions and Smyth's powerdomains, in: *Abstract Software Specifications*, Lecture Notes in Computer Science, Vol. 86 (Springer, Berlin, 1979).
[21] M.B. Smyth, Powerdomains, *J. Comput. System Sci.* **16** (1978) 23–36.