Theoretical Computer Science 124 (1994) 273–295
Elsevier

Nondeterminacy and recursion via stacks and games

Wim H. Hesselink

Rijksuniversiteit Groningen, Department of Computing Science, P.O. Box 800, 9700 AV Groningen, The Netherlands

Communicated by J.W. de Bakker
Received August 1991
Revised May 1992

Abstract

Hesselink, W.H., Nondeterminacy and recursion via stacks and games, Theoretical Computer Science 124 (1994) 273–295.

The weakest-precondition interpretation of recursive procedures is developed for a language with a combination of unbounded demonic choice and unbounded angelic choice. This compositional formal semantics is proved to be equal to a game-theoretic operational semantics. Two intermediate stages are exploited. One step consists of unfolding the declaration of the recursive procedures. Fixpoint induction is used to prove the validity of this step. The compositional semantics of the unfolded declaration is proved to be equal to a formal semantics of a stack implementation of the recursive procedures. After an introduction to boolean two-person games, this stack semantics is shown to correspond to a game-theoretic operational semantics.

0. Introduction

In sequential programming, Dijkstra's weakest preconditions (cf. [5, 6]) form the semantic formalism most adequate for programming methodology. In this formalism, the meaning of a command expression c is expressed by $wp.c$, the function that assigns to a postcondition p the weakest precondition such that c is guaranteed to terminate in a state where p holds (to avoid circularity or contradictions, this should be read as an informal introduction and not as a formal definition). We let the application operator . bind to the left and therefore write $wp.c.p$ instead $(wp.c).p$ for the weakest precondition of postcondition p.

Correspondence to: W.H. Hesselink, Rijksuniversiteit Groningen, Department of Computing Science, P.O. Box 800, 9700 AV Groningen, The Netherlands. Email: wim@cs.rug.nl.

0304-3975/94/$07.00 © 1994—Elsevier Science B.V. All rights reserved
SSDI 0304-3975(92)00016-A

Sequential composition of command expressions, demonic nondeterminate choice of command expressions and (mutually) recursive procedures are treated in the following way. The weakest precondition of the sequential composition $(c; d)$ of command expressions c and d for postcondition p is given by

$$wp.(c;d).p = wp.c.(wp.d.p).$$

Using composition of predicate transformers, this rule is equivalently expressed by

$$wp.(c;d) = wp.c \circ wp.d. \tag{0}$$

The weakest precondition of the *demonic* choice $(\Box i :: c.i)$ of a family of command expressions $(i \in I :: c.i)$ is given by

$$wp.(\Box i :: c.i).p = (\forall i :: wp.(c.i).p). \tag{1}$$

For a recursive procedure h, the predicate transformer $wp.h$ is defined as the strongest solution of the equation

$$wp.h = wp.(\mathbf{body}.h). \tag{2}$$

In case of mutual recursion, function wp is the strongest solution of the system of equations (2) where h ranges over all procedure names.

In compilers, recursive procedures are implemented by means of a stack of commands, say q. In that case, the meaning of stack q is given by the weakest precondition $wp.q$ which is supposed to be the strongest solution of a system of equations like

$$\begin{aligned} &wp.\varepsilon = identity, \\ &wp.(c;q) = wp.c \circ wp.q, \\ &wp.(h;q) = wp.(\Box r \in \mathbf{alt}.h :: r; q), \end{aligned} \tag{3}$$

where ε is the empty stack, c ranges over simple commands, q ranges over strings of commands, h ranges over procedure names and $\mathbf{body}.h = (\Box r \in \mathbf{alt}.h :: r)$. Of course, (3) is a direct consequence of (0) and (2), but that does not mean that the strongest solution of (2) also yields the strongest solution of (3). In fact, it is not obvious that the strongest solution of (3) satisfies (0).

As early as 1975, De Bakker proved the compatibility of (0), (2) and (3) as defining equations in a special case by means of Scott induction, see [3]. The induction rule of De Bakker and Scott, however, is not valid for unbounded demonic nondeterminacy and has not been developed for angelic nondeterminacy. Therefore, we are glad to be able to prove this compatibility by other means and under weaker assumptions.

In the versions of [6, 9], predicate-transformation semantics relies on certain postulates (the healthiness laws) that are equivalent to the condition that every command has well-defined relational semantics (total in the case of [6]). Recently, these healthiness laws have come under attack. In fact, in the refinement calculus of Back, Morgan, Morris and others, several specification constructs have been

proposed that lead to command expressions violating the healthiness laws. The simplest case is the operator $\diamond$ for angelic choice (cf. [1, 13–15]).

We use the notation $(\diamond i :: c.i)$ for the *angelic* choice between command expressions $c.i$. It is specified by its weakest precondition for postcondition p

$$wp.(\diamond i :: c.i).p = (\exists i :: wp.(c.i).p). \tag{4}$$

In the case of a choice between two command expressions, the symbols $\diamond$ and $[]$ can be used as infix operators, so that

$$wp.(c \diamond d).p = wp.c.p \vee wp.d.p,$$

$$wp.(c [] d).p = wp.c.p \wedge wp.d.p.$$

Remark. In [13], the angelic choice $(\diamond\ i \in I :: c.i)$ is denoted **con** $i : I \bullet c.i$; the dummy i is called a logical constant. The construct is introduced in [13, Ch. 6, Section 21.3.5]. A nontrivial example is given in [13, Section 15.4]. We have the impression that in the cases where the construct is used, it is used mainly for convenience, not because it is indispensable. The development of constructs useful to reduce the complexity of the design process, however, is an important issue.

It must be admitted that the angelic choice in isolation has rather strange properties. If, for example, j is an integer program variable, then command $c = (\mathrm{j} := 0 \diamond \mathrm{j} := 1)$ satisfies

$$\begin{aligned} & wp.c.(\mathrm{j} = 0) \\ & \quad = wp.(\mathrm{j} := 0).(\mathrm{j} = 0) \vee wp.(\mathrm{j} := 1).(\mathrm{j} = 0) \\ & \quad = true \vee false \\ & \quad = true \end{aligned}$$

and, by similar calculations,

$$wp.c.(\mathrm{j} = 1) = true$$

and

$$wp.c.(\mathrm{j} = 0 \wedge \mathrm{j} = 1) = false.$$

So the command is able to establish $\mathrm{j} = 0$ as well as $\mathrm{j} = 1$. It seems to guess the postcondition we have in mind. For this reason one might prefer to speak of specification c instead of command c. The angelic properties of c prohibit an operational model in the usual style. In this special case, $wp.c$ can be expressed by means of the diamond operator of dynamic logic, cf. [7], but that formalism cannot express mixtures of demonic choice and angelic choice. In [2], a general command expression c is regarded as a game between a demon and an angel: predicate $wp.c.p$ means that the angel has a winning strategy to establish postcondition p. We shall formalize this point of view in Sections 3 and 4, see also [8].

Remark. The term angelic choice is often used to refer to termination only, e.g., cf. [12]. Angelic choice in that restricted sense is still representable in relational calculus.

For a command expression c constructed by means of angelic choice, the predicate transformer $wp.c$ is monotone, but it may fail to be conjunctive (as is shown in the above example). This paper is therefore addressed to the foundations of a *wp*-calculus in which $wp.c$ can be any monotone predicate transformer. We determine an abstract system of equations of the form (3), such that its strongest solution satisfies (0), restricts to the strongest solution of (2) and allows unbounded demonic and angelic choices in procedure bodies.

The plan of the paper is as follows. In Section 1 we introduce the syntax, the semantics and the compositional interpretation. Section 2 contains the development of the stack interpretation. In Section 3 we provide a brief theory of boolean two-person games, similar to the alternating Turing machines of [4]. This theory is used in Section 4 to present the game-theoretic operational semantics that corresponds to the stack interpretation of Section 2. In Section 5, we prove that the interpretations of Sections 1 and 2 agree.

In Section 2 we need a rather severe restriction on the syntax of procedure declarations. Every declaration can be rewritten such as to satisfy this restriction. This is a kind of preprocessing called unfolding. The fact that unfolding does not change the predicate transformation semantics is proved in Section 6. In Section 7 we briefly sketch the treatment of conditional correctness. Section 8 contains the conclusions.

1. Unbounded choices and recursion with homomorphic interpretation

In this section we introduce an abstract syntax for command expressions, a semantic formalism and a compositional interpretation function from syntax to semantics.

We first discuss the syntax to be used. Let A be a set of symbols, to be called *commands*. We assume that A contains all simple commands and all procedure names that may be needed. Starting from A, we define the class *Cmd* of command expressions inductively by the clauses

- $A \subseteq Cmd$,
- if $c, d \in Cmd$ then $c; d \in Cmd$,
- if $(i \in I :: c.i)$ is a family in *Cmd* then $(\Box\, i :: c.i) \in Cmd$ and $(\diamond\, i :: c.i) \in Cmd$.

Since the index sets I are arbitrary, *Cmd* is not a set but a proper class in the sense of set theory, cf. [10]. This fact need not be a matter of great concern.

We assume that the set of commands A is the disjoint union of two sets S and H, which may be infinite. The elements of S are called simple commands. Their semantics are supposed to be given. The elements of H are called procedure names. Every procedure $h \in H$ is supposed to be equipped with a body $\mathbf{body}.h \in Cmd$.

The semantics are expressed in terms of predicates on a given state space X. Let $\mathbb{P}$ be the set of predicates on X. The set $\mathbb{P}$ is ordered by strength, i.e.,

$$p \leqslant q \equiv (\forall x \in X :: p.x \Rightarrow q.x).$$

Let MT be the set of monotone predicate transformers, i.e., montone functions $\mathbb{P} \to \mathbb{P}$. The set MT is equipped with the induced order given by

$$f \leqslant g \equiv (\forall p \in \mathbb{P} :: f.p \leqslant g.p).$$

It is well known that $\mathbb{P}$ and MT with these orders are complete lattices. The infimum or greatest lower bound ($\sqcap$) of a family of predicates $p.i$ with $i \in I$ is denoted $(\inf i :: p.i)$; it is the universal quantification $(\forall i :: p.i)$. Similarly, the supremum or least upper bound $(\sup i :: p.i)$ is the existential quantification $(\exists i :: p.i)$. The infimum of a family of predicate transformers $f.i$ with $i \in I$ is the predicate transformer given (argumentwise) by

$$(\inf i :: f.i).p = (\inf i :: f.i.p)$$

for every predicate p. Similarly for the supremum.

Recall that the well-known theorem of Knaster–Tarski (cf. [17]) asserts that, for any complete lattice W, a monotone function $D \in W \to W$ has a least fixpoint and a greatest fixpoint.

Every simple command $s \in S$ has an associate predicate transformer $ws.s \in MT$. So the semantics of the simple commands are given by a fixed function $ws \in S \to MT$. We assume that S contains sufficiently many assignments and also the so-called guards. For a predicate b, the guard $?b$ is the simple command with $ws.(?b).p = (\neg b \vee p)$. It is well known that conditional choice can be expressed by

$$\textbf{if } b \textbf{ then } c \textbf{ else } d \textbf{ fi} = (?b; c \,[]\, ?\neg b; d).$$

In view of (0), (1), (2) and (4), the weakest precondition function $wp \in Cmd \to MT$ is supposed to satisfy

$$\begin{aligned}
&wp.(c; d) = wp.c \circ wp.d,\\
&wp.([]i :: c.i) = (\inf i :: wp.(c.i)),\\
&wp.(\diamond\, i :: c.i) = (\sup i :: wp.(c.i)),\\
&wp.s = ws.s \quad \text{for } s \in S,\\
&wp.h = wp.(\textbf{body}.h) \quad \text{for } h \in H.
\end{aligned} \tag{5}$$

The first three clauses of (5) form a compositionality property. More generally, we define a function $v \in Cmd \to MT$ to be a *homomorphism* if and only if

$$\begin{aligned}
&v.(c; d) = v.c \circ v.d,\\
&v.([]i :: c.i) = (\inf i :: v.(c.i)),\\
&v.(\diamond\, i :: c.i) = (\sup i :: v.(c.i)).
\end{aligned}$$

Every function $u \in A \to MT$ permits precisely one homomorphism $u^{\odot} \in Cmd \to MT$ with restriction $(u^{\odot} \mid A) = u$, which is called the *homomorphic extension* of u. In fact, $u^{\odot}$ can be easily constructed inductively.

In the presence of recursive procedures, function wp need not be completely determined by system (5). We therefore define wp to be the homomorphic extension $wp = wa^{\odot}$ where wa is the *least* solution $\omega \in A \to MT$ of the system of equations

$$\begin{aligned} &\omega.s = ws.s \quad \text{for } s \in S, \\ &\omega.h = \omega^{\odot}.(\mathbf{body}.h) \quad \text{for } h \in H. \end{aligned} \tag{6}$$

In fact, the set $A \to MT$ is made into a complete lattice by argumentwise extension of the order of MT. Then it is easy to see that the right-hand side of (6) is a monotone function of w, so that the theorem of Knaster–Tarski yields a unique least solution for (6). Function wp is called the *homomorphic interpretation.*

Remark. Let us say that procedure h calls h' if h' occurs in the command expression $\mathbf{body}.h$. If relation "calls" is well founded, function wp is the only solution of system (5) (and hence of (6)).

2. The stack interpretation

In this section we introduce the stack interpretation, which looks like the stack implementation in actual compilers. In Section 4, we show that the stack interpretation corresponds to a game-theoretic operational semantics. Since the stack is treated as the current task, the stack interpretation may be regarded as a kind of continuation semantics.

For simplicity, our stack implementation of recursive procedures requires that the program be preprocessed in such a way that function **body** satisfies a much stricter syntax in which every choice is governed by a separate procedure name. The additional procedure names play rôles similar to labels for jumps and stacked return addresses in compilers.

Let A^* be the set of strings of commands. We write ε to denote the empty string and use the semicolon; to denote catenation of strings. We identify an element of A with the corresponding singleton string, so that A is regarded as a subset of A^*.

The restricted syntax is described as follows. We assume that the set H of procedure names is the disjoint union of two sets Hd and Ha and that every procedure $h \in H$ is equipped with a set of strings (alternatives), denoted by $\mathbf{alt}.h \subseteq A^*$ such that the body $\mathbf{body}.h$ of h is given by

$$\mathbf{body}.h = \begin{cases} (\square r \in \mathbf{alt}.h :: r) & \text{if } h \in Hd, \\ (\diamond\, r \in \mathbf{alt}.h :: r) & \text{if } h \in Ha. \end{cases} \tag{7}$$

The preprocessing to transform an arbitrary declaration into one that can be expressed by means of (7) is called *unfolding*. More generally, unfolding is the process of replacing command subexpressions of procedure bodies by procedure names with the command subexpression as body. We are only interested in replacing nested choices. A declaration that satisfies (7) is thus regarded as completely unfolded. It may seem obvious that unfolding preserves the semantics. The proof of that fact, however, is nontrivial. It is given in Section 6. Meanwhile, we assume that (7) is satisfied.

Example. Let x be an integer program variable. Let the recursive procedure h be declared by

$$\textbf{body}.h \;=\; skip \diamond \mathrm{x}:=2*\mathrm{x};(skip \,[]\, \mathrm{x}:=\mathrm{x}+1);h.$$

If x is initially positive this procedure extends the binary representation of x with a "convenient" number of bits, but the values of these bits are not under control. For example, $wp.h.(\mathrm{x}\in\{5,8,9\})$ is implied by $\mathrm{x}=2$. In this case, the unfolding uses one additional procedure name, say k, with $h\in Ha$ and $k\in Hd$ and

$$\textbf{alt}.h=\{skip,\ \mathrm{x}:=2*\mathrm{x};k;h\},$$

$$\textbf{alt}.k=\{skip,\ \mathrm{x}:=\mathrm{x}+1\}.$$

Of course, the example is only an illustration of the technique and not intended to be useful. □

The stack implementation of recursion is based on the treatment of a string of commands as a stack. This means that the interpretation is a function in $A^*\to MT$. The recurrence equation that defines the stack implementation is obtained as follows. The empty stack is interpreted as the command "do nothing" and therefore should induce the identity transformation. The meaning of a nonempty stack $(a; q)$ should be a predicate transformer expressed in terms of the head a of the stack and the meaning of the tail q of the stack.

We therefore define the *stack interpretation wk* to be the *least* solution of the system of equations in $v\in A^*\to MT$:

$$\begin{aligned}
&v.\varepsilon = identity,\\
&v.(a;q)=ws.a\circ v.q \quad \text{if } a\in S,\\
&v.(a;q)=(\inf r\in \textbf{alt}.a::v.(r;q)) \quad \text{if } a\in Hd,\\
&v.(a;q)=(\sup r\in \textbf{alt}.a::v.(r;q)) \quad \text{if } a\in Ha
\end{aligned} \tag{8}$$

for all $q\in A^*$.

In (8), the argument of v is the stack. The idea is that, repeatedly, the next command is popped from the stack. If it is a simple command it is executed. If it is a procedure name, one of its alternatives is pushed onto the stack. In actual compilers, the original program is kept, so that only return addresses and parameters need to be stacked.

The set $A^* \to MT$ with the induced order is a complete lattice. The right-hand side of (8) is a monotone function of v. Therefore, the theorem of Knaster–Tarski implies that system (8) has a least solution. This least solution is the stack interpretation wk. Note that it is not yet clear that $wk.(q;r) = wk.q \circ wk.r$ for strings q and r. This will be proved in Section 5. The relationship between the interpretations wk and wp is also treated in Section 5.

Definition (8) is a kind of tail recursion. We can therefore fix the postcondition, say p, and derive a recursive definition for the function $f \in A^* \to \mathbb{P}$ given by $f.q = wk.q.p$. In fact, it follows from the above that f is the least solution of the system of equations

$$\begin{aligned} &f.\varepsilon = p, \\ &f.(a;q) = ws.a.(f.q) \quad \text{if } a \in S, \\ &f.(a;q) = (\inf r \in \mathbf{alt}.a :: f.(r;q)) \quad \text{if } a \in Hd, \\ &f.(a;q) = (\sup r \in \mathbf{alt}.a :: f.(r;q)) \quad \text{if } a \in Ha \end{aligned} \tag{9}$$

for all $q \in A^*$.

3. Boolean two-person games

In this section we prepare the game-theoretic semantics of the stack interpretation wk by developing a small theory of boolean games. The alternating Turing machines of [4] form a special case of this theory. Conceptually, the theory is closely related to the AND/OR trees that occur in logic programming and artificial intelligence, cf. [16].

Let a *boolean two-person game* be defined as a quadruple $(V, \to, k, v)$ where V is a set, $\to$ is a binary relation on V and k is a predicate on V and $v \in V$. The pair $(V, \to)$ may be regarded as a possibly infinite directed graph. The function $k \in V \to \mathbb{B}$ might be regarded as a colouring of the nodes, but we do not require that connected vertices have different colours.

The set V is interpreted as the set of configurations of the game; relation $\to$ is the set of moves. Function k indicates the player that takes the move. The element v is the initial configuration of the game. There are two players. The situation is symmetrical in the two players, but, for simplicity, we take sides and decide that the players are called *angel* and *demon*. For each vertex x:

$k.x$ means that the demon takes the move,

$\neg k.x$ means that the angel takes the move.

A player wins directly if its opponent takes the move and has no move. For the moment, any infinite execution path is regarded as a draw.

In principle, we are only interested in the question whether one of the players has a winning strategy when the game starts in the initial configuration v. It turns out that the initial configuration only plays a rôle in the final interpretation of the game. For the investigation of the game itself, v need not be specified.

Example. Let V be the set of the integers. Let $\rightarrow$ and k be given by

$$x \rightarrow y \equiv x \notin \{0,9\} \wedge y \in \{x-1, x-5\},$$

$$k.x \equiv x \bmod 2 = 0.$$

The demon is lost in position 0. It follows that the angel has a winning strategy in the positions x with

$$x \geqslant 0 \wedge x \bmod 6 \in \{0,1,5\}.$$

Similarly, the demon wins in position 9 and has a winning strategy in the positions x with

$$x \geqslant 9 \wedge x \bmod 6 \in \{2,3,4\}. \quad \square$$

We now assume that both players use an optimal strategy. Let W be the set of configurations where the angel does not lose. Then, once inside W, the demon cannot escape from W and the angel can remain inside W. This implies that

$$(\forall x \in W :: (k.x \Rightarrow (\forall y: x \rightarrow y: y \in W)) \wedge (\neg k.x \Rightarrow (\exists y: x \rightarrow y: y \in W))). \tag{10}$$

Conversely, any set W that satisfies (10) can serve the angel as a refuge where it cannot lose. We therefore introduce the function D from subsets of V to subsets of V given by

$$x \in D.U \equiv (k.x \Rightarrow (\forall y: x \rightarrow y: y \in U)) \wedge (\neg k.x \Rightarrow (\exists y: x \rightarrow y: y \in U)). \tag{11}$$

Function D is monotone with respect to inclusion. It now follows that the greatest fixpoint of D is the greatest set where the angel does not lose.

By symmetry, it follows that the set W' where the demon does not lose is the greatest fixpoint of function D' given by

$$x \in D'.U \equiv (k.x \Rightarrow (\exists y: x \rightarrow y: y \in U)) \wedge (\neg k.x \Rightarrow (\forall y: x \rightarrow y: y \in U)). \tag{12}$$

Consequently, the set of configurations where the angel wins is the complement $V \setminus W'$. In order to characterize $V \setminus W'$ as a fixpoint, we observe that the complement of any fixpoint of D' is a fixpoint of D, and vice versa. This follows from the identity

$$V \setminus D'.U = D.(V \setminus U),$$

which is proved in

$$\begin{aligned} x \notin D'.U \equiv\ & \{(12) \text{ and calculus}\} \\ & (k.x \Rightarrow \neg(\exists y: x \rightarrow y: y \in U)) \\ & \wedge (\neg k.x \Rightarrow \neg(\forall y: x \rightarrow y: y \in U)) \\ \equiv\ & \{\text{calculus and (11) with } U := V \setminus U\} \\ & x \in D.(V \setminus U). \end{aligned}$$

This implies that $V \setminus W'$ is the least fixpoint of D. This proves the following theorem.

Theorem 3.1. *The least fixpoint of D is the set of configurations where the angel wins. The greatest fixpoint of D is the set of configurations where the angel does not lose.*

4. Game-theoretic semantics

We now develop game-theoretic semantics for the formalism of Section 2. The first task is to give an interpretation to the simple commands.

We assume that every simple command $c \in S$ has an input–output relation $[\![c]\!]$ on the state space X and that the set of simple commands is a disjoint union $S = Sa \cup Sd$. Here Sa (Sd) stands for the set of simple commands with angelic (demonic) nondeterminacy. The simple commands are interpreted in the following way. For a predicate p on X, we define

$$\begin{aligned} &ws.c.p.x \equiv (\forall y: (x,y) \in [\![c]\!]: p.y) \quad \text{if } c \in Sd, \\ &ws.c.p.x \equiv (\exists y: (x,y) \in [\![c]\!]: p.y) \quad \text{if } c \in Sa. \end{aligned} \tag{13}$$

Remark. This is a restriction on the simple commands. In fact, for $c \in Sd$ ($c \in Sa$) the predicate transformer $ws.c$ commutes with arbitrary conjunctions (disjunctions). Conversely, every predicate transformer that commutes with arbitrary conjunctions or disjunctions can be admitted as a simple command. For deterministic commands such as assignments the choice between Sd and Sa is irrelevant.

Example. For a predicate b, the guard $?b \in Sd$ and the assertion $!b \in Sa$ are defined by the same input–output relation $[\![!b]\!] = [\![?b]\!]$ given by

$$(x,y) \in [\![?b]\!] \;\equiv\; x = y \wedge b.x.$$

It follows from (13) that

$$ws.(?b).p.x \;\equiv\; \neg b.x \vee p.x,$$

$$ws.(!b).p.x \;\equiv\; b.x \wedge p.x.$$

In fact, a universal quantification over the empty domain yields true and an existential one yields false. □

For a given postcondition p, the *game-theoretic* semantics of a string $t \in A^*$ with respect to the initial state z and postcondition p is defined as the boolean two-person game $(V, \rightarrow, k, v)$, where V is the cartesian product $X \times A^*$ and the initial configuration v is $(z,t) \in V$. Function k is given by stating that, for all $x \in X$, $q \in A^*$ and $a \in A$,

$$k.(x,\varepsilon) \equiv p.x,$$

$$k.(x,a;q) \equiv (a \in Sd \cup Hd).$$

Relation $\rightarrow$ is defined as the least relation on V that satisfies

- if $c \in S \wedge (x, y) \in [\![c]\!]$ then $(x, c; q) \rightarrow (y, q)$,
- if $h \in H \wedge r \in \mathbf{alt}.h$ then $(x, h; q) \rightarrow (x, r; q)$.

Note that the configurations (x, ε) have no transitions.

The interpretation is that state z satisfies the weakest precondition of command string t for postcondition p if and only if, in the game starting at $v = (z, t)$, the angel has a winning strategy, i.e., a strategy such that the game is guaranteed to reach a configuration (x, ε) where $p.x$ holds.

In this case, function D of (11) satisfies

$$(x, \varepsilon) \in D.U \equiv p.x \quad \text{for all } x \in X,$$

$$(x, c; q) \in D.U \equiv (\forall y: (x, y) \in [\![c]\!]: (y, q) \in U) \quad \text{if } c \in Sd,$$

$$(x, c; q) \in D.U \equiv (\exists y: (x, y) \in [\![c]\!]: (y, q) \in U) \quad \text{if } c \in Sa,$$

$$(x, h; q) \in D.U \equiv (\forall r \in \mathbf{alt}.h :: (x, r; q) \in U) \quad \text{if } h \in Hd,$$

$$(x, h; q) \in D.U \equiv (\exists r \in \mathbf{alt}.h :: (x, r; q) \in U) \quad \text{if } h \in Ha.$$

Let $W0$ be the greatest region where the angel wins. By Theorem (3.1), $W0$ is the least fixpoint of function D. Let function $f \in A^* \rightarrow \mathbb{P}$ be given by

$$f.q.x \equiv (x, q) \in W0.$$

Then f is the strongest solution of the system of equations:

$$f.\varepsilon.x \equiv p.x \quad \text{for all } x \in X,$$

$$f.(c; q).x \equiv (\forall y: (x, y) \in [\![c]\!]: f.q.y) \quad \text{if } c \in Sd,$$

$$f.(c; q).x \equiv (\exists y: (x, y) \in [\![c]\!]: f.q, y) \quad \text{if } c \in Sa,$$

$$f.(h; q).x \equiv (\forall r \in \mathbf{alt}.h :: f.(r; q).x) \quad \text{if } h \in Hd,$$

$$h.(h; q).x \equiv (\exists r \in \mathbf{alt}.h :: f.(r; q).x) \quad \text{if } h \in Ha.$$

By definition (13), the second clause and the third one can be unified to

$$f.(c; q).x \equiv ws.c.(f.q).x \quad \text{if } c \in S.$$

Since infima and suprema in $\mathbb{P}$ correspond to pointwise universal and existential quantification, this implies that f is the strongest (i.e. least) solution of system (9) obtained in Section 2. This proves that the game-theoretic semantics coincides with the stack interpretation *wk* of Section 2.

5. The stack theorem

In this section we show the agreement between the stack interpretation *wk* and the homomorphic interpretation *wp*. More precisely, we show that *wp* and *wk* agree on the set A^* of strings of commands.

The functions *wp* and *wk* are defined by means of the least solutions of the systems (6) and (8). System (8) implies that *wk* behaves properly with respect to the choice operators, but does not (yet) guarantee $wk.(q;r) = wk.q \circ wk.r$. In order to discuss this property, we define a function $v \in A^* \to MT$ to be *multiplicative* if and only if

$$v.\varepsilon = identity,$$

$$v.(q;r) = v.q \circ v.r.$$

Every function $u \in A \to MT$ permits precisely one multiplicative function $u^* \in A^* \to MT$ with restriction $(u^* | A) = u$, which is called the *multiplicative extension* of u. In fact, it is easy to see that u^* can be constructed inductively.

We now observe that, under assumption (7), the function $wa = (wp | A)$ defined as the least solution of the system of equations (6) is also the least solution $\omega \in A \to MT$ of the system

$$\begin{aligned} &\omega.a = ws.a \quad \text{if } a \in S, \\ &\omega.a = (\inf r \in \mathbf{alt}.a :: \omega^*.r) \quad \text{if } a \in Hd, \\ &\omega.a = (\sup r \in \mathbf{alt}.a :: \omega^*.r) \quad \text{if } a \in Ha. \end{aligned} \tag{14}$$

Now the aim is to prove that $wk = wa^*$. In order not to be hindered by the case distinctions in (8) and (14), we apply abstraction and introduce a function

$$Q \in (A^* \to MT) \to (A \to MT)$$

to express the right-hand sides of (8) and (14). This function Q is defined by

$$Q.v.a = \begin{cases} ws.a \circ v.\varepsilon & \text{if } a \in S, \\ (\inf r \in \mathbf{alt}.a :: v.r) & \text{if } a \in Hd, \\ (\sup r \in \mathbf{alt}.a :: v.r) & \text{if } a \in Ha, \end{cases} \tag{15}$$

for all $v \in A^* \to MT$. Here the reader may recall that $Q.v.a = (Q.v).a$.

We first show that Q can be used to express (14). For every $\omega \in A \to MT$, function $\omega^* \in A^* \to MT$ is multiplicative and therefore satisfies $ws.a \circ \omega^*.\varepsilon = ws.a$ for all $a \in S$. This implies that system (14) is equivalent to

$$\omega.a = Q.\omega^*.a \quad \text{for all } a \in A. \tag{16}$$

It is easy to see that function Q is monotone. Function *wa* can therefore be defined as the least solution $\omega \in A \to MT$ of system (16).

In order to use Q to express system (8), we note that the term $v.q$ in the second clause of (8) can also be written as $v.(\varepsilon; q)$. So, in each case, the argument of v at the right-hand side of (8) is postfixed with string q. We therefore introduce the function

$$R \in A^* \to ((A^* \to MT) \to (A^* \to MT)),$$

given by

$$R.q.v.s = v.(s;q) \quad \text{for all } q, s \in A^*,\ v \in A^* \to MT. \tag{17}$$

We claim that system (8) is equivalent to the system of equations in $v \in A^* \to MT$

$$\begin{aligned} &v.\varepsilon = \mathit{identity}, \\ &v.(a;q) = Q.(R.q.v).a \quad \text{for all } a \in A,\ q \in A^*. \end{aligned} \tag{18}$$

In fact, if $a \in S$ then $Q.(R.q.v).a = ws.a \circ R.q.v.\varepsilon = ws.a \circ v.q$. For $a \in Hd$ we have

$$Q.(R.q.v).a = (\inf r \in \mathbf{alt}.a :: R.q.v.r) = (\inf r \in \mathbf{alt}.a :: v.(r;q)).$$

The calculation for procedure names $a \in Ha$ is completely analogous.

With respect to the comparison of the interpretations *wk* and *wa**, we now have that *wa* is the least solution of equation (16) and *wk* is the least solution of equation (18). It turns out that we need only two abstract properties of function Q for the proof of $wk = wa^*$. The first one is that function Q is monotone. The second property is contained in the next lemma.

Lemma 5.1. *Let function* $L \in MT \to ((A^* \to MT) \to (A^* \to MT))$ *be given by*

$$L.f.v.s = v.s \circ f \quad \textit{for all } f \in MT,\ s \in A^*,\ v \in A^* \to MT.$$

Then $Q.(L.f.v).a = Q.v.a \circ f$ *for all* $f \in MT, v \in A^* \to MT$ *and* $a \in A$.

Proof. There are three cases to consider. For $a \in S$, it suffices to observe that

$$\begin{aligned} &Q.(L.f.v).a = Q.v.a \circ f \\ &\quad \equiv \{(15)\} \\ &\qquad ws.a \circ L.f.v.\varepsilon = ws.a \circ v.\varepsilon \circ f \\ &\quad \equiv \{\text{definition of } L\} \\ &\qquad \text{true}. \end{aligned}$$

For $a \in Hd$, we observe

$$\begin{aligned} &Q.(L.f.v).a = Q.v.a \circ f \\ &\quad \equiv \{(15);\ \text{let } r \text{ range over } \mathbf{alt}.a\} \\ &\qquad (\inf r :: L.f.v.r) = (\inf r :: v.r) \circ f \\ &\quad \equiv \{\text{definition of } L\} \\ &\qquad (\inf r :: v.r \circ f) = (\inf r :: v.r) \circ f \\ &\quad \equiv \{\text{a rule for predicate transformers}\} \\ &\qquad \text{true}. \end{aligned}$$

The proof for $a \in Ha$ is similar. □

The main stepping stones for the proof of $wk = wa^*$ are the following two lemmas. We first need a result concerning fixpoints in an arbitrary complete lattice W. It is well known that the least fixpoint $x0$ of a monotone function $D \in W \to W$ can be constructed by

$$x0 = (\inf \omega \in W : D.\omega \leqslant \omega : \omega). \tag{19}$$

A subset V of W is said to be *sup-closed* if it contains the suprema of all its subsets. Note that the least element of W is the supremum of the empty set and, hence, an element of every sup-closed set. A subset V of W is said to be *D-invariant* if $D.v \in V$ for all $v \in V$. In our favorite version of the theorem of Knaster–Tarski (cf. [9, Section 4.2]), it is stated that the least fixpoint $x0$ of D is an element of every subset V of W that is sup-closed and D-invariant. This is a kind of induction principle. It is used in the first lemma.

Lemma 5.2. *Let W be a complete lattice and $D \in W \to W$ a monotone function with least fixpoint $x0$. Let Z be an ordered set with a monotone function $E \in Z \to Z$. Let $F \in W \to Z$ be such that $F \circ D = E \circ F$ and that F commutes with suprema. Then $F.x0$ is the least fixpoint of E.*

$$\begin{array}{lccc} x0 \in & W & \xrightarrow{D} & W \\ & {\scriptstyle F}\downarrow & & \downarrow{\scriptstyle F} \\ & Z & \xrightarrow{E} & Z \end{array}$$

Proof. $F.x0$ is a fixpoint of E since $E.(F.x0) = F.(D.x0) = F.x0$.

In order to show that $F.x0$ is the least fixpoint, we argue as follows. Let $z \in Z$ be any fixpoint of E. We have to prove that $F.x0 \leqslant z$. For this purpose, it suffices to prove that $x0$ is an element of the subset V of W given by

$$u \in V \;\equiv\; F.u \leqslant z.$$

Since $x0$ is the least fixpoint of D, the result quoted above implies that it suffices to prove that the set V is sup-closed and D-invariant.

The set V is sup-closed in W since, for any subset U of W,

$$\begin{aligned} & F.(\sup U) \leqslant z \\ & \quad \equiv \{F \text{ commutes with suprema}\} \\ & \qquad (\sup u \in U :: F.u) \leqslant z \\ & \quad \equiv \{\text{definition supremum}\} \\ & \qquad (\forall u \in U :: F.u \leqslant z). \end{aligned}$$

The set V is D-invariant since, for any $\omega \in W$,

$$
\begin{aligned}
& F.(D.\omega) \leqslant z \\
& \equiv \{F \circ D = E \circ F;\ z \text{ fixpoint of } E\} \\
& \quad E.(F.\omega) \leqslant E.z \\
& \Leftarrow \{E \text{ is monotone}\} \\
& \quad F.\omega \leqslant z. \qquad \square
\end{aligned}
$$

We use Lemma 5.2 to prove the next stepping stone: the function wk is "submultiplicative".

Lemma 5.3. *$wk.s \circ wk.r \leqslant wk.(s;r)$ for all $s, r \in A^*$.*

Proof. We fix string r and define function $g \in MT$ by $g = wk.r$ and function $y \in A^* \to MT$ by $y.s = wk.(s;r)$. Now the assertion is equivalent to $wk.s \circ g \leqslant y.s$ for all $s \in A^*$, or equivalently to

$$L.g.wk \leqslant y. \tag{20}$$

Formula (20) is proved by showing that y is a fixpoint of a function E and that $L.g.wk$ is the least fixpoint of E.

We clearly have $y.\varepsilon = wk.r = g$. For every string $q \in A^*$ and command $a \in A$, we observe that

$$
\begin{aligned}
& y.(a;q) \\
& = \{\text{definition of } y\} \\
& \quad wk.(a;q;r) \\
& = \{wk \text{ solves (18)}\} \\
& \quad Q.(R.(q;r).wk).a \\
& = \{\text{(17) with definition of } y \text{ yields} \\
& \quad R.(q;r).wk.s = wk.(s;q;r) = y.(s;q) = R.q.y.s \\
& \quad \text{for every } s \in A^* \text{ and hence } R.(q;r).wk = R.q.y\} \\
& \quad Q.(R.q.y).a.
\end{aligned}
$$

This proves that y is a fixpoint of the function

$$E \in (A^* \to MT) \to (A^* \to MT),$$

given by

$$E.v.\varepsilon = g, \tag{21}$$

$$E.v.(a;q) = Q.(R.q.v).a \quad \text{for } v \in A^* \to MT,\ a \in A,\ q \in A^*.$$

So, in order to prove (20), it suffices to show that $L.g.wk$ is the least fixpoint of E.

Function wk is the least solution of (18) and hence the least fixpoint of the function

$$D \in (A^* \to MT) \to (A^* \to MT),$$

given by

$$D.v.\varepsilon = identity \tag{22}$$

$$D.v.(a;q) = Q.(R.q.v).a \quad \text{for } v \in A^* \to MT,\ a \in A,\ q \in A^*.$$

We now use Lemma 5.2 with $x0 = wk$ and $F = L.g$ to prove that $L.g.wk$ is the least fixpoint of E. Therefore, it remains to prove that $L.g$ commutes with suprema and satisfies

$$L.g \circ D = E \circ L.g. \tag{23}$$

Function $L.g$ commutes with suprema in $A^* \to MT$, since for any subset U of $A^* \to MT$ and any string $s \in A^*$ and any predicate p,

$$L.g.(\sup U).s.p = (\sup u \in U :: L.g.u).s.p$$

$$\equiv \{\text{Lemma 5.1}\}$$

$$(\sup U).s.(g.p) = (\sup u \in U :: L.g.u).s.p$$

$$\equiv \{\text{suprema in } A^* \to MT \text{ and in } MT \text{ are pointwise}\}$$

$$(\sup u \in U :: u.s.(g.p)) = (\sup u \in U :: L.g.u.s.p)$$

$$\equiv \{\text{Lemma 5.1}\}$$

$$\text{true}.$$

It remains to prove formula (23). We first observe that, for every $v \in A^* \to MT$ and $s \in A^*$,

$$(L.g \circ D).v.s = (E \circ L.g).v.s$$

$$\equiv \{\text{composition twice}\}$$

$$L.g.(D.v).s = E.(L.g.v).s$$

$$\equiv \{\text{Lemma 5.1}\}$$

$$D.v.s \circ g = E.(L.g.v).s. \tag{24}$$

Formula (24) is proved by a case distinction according to the definitions (21) and (22). For $s=\varepsilon$ both sides of (24) reduce to g. For $s:=a;q$, we observe that

$$
\begin{aligned}
& D.v.(a;q)\circ g = E.(L.g.v).(a;q) \\
& \quad \equiv \{(21) \text{ and } (22)\} \\
& \qquad Q.(R.q.v).a \circ g = Q.(R.q.(L.g.v)).a \\
& \quad \equiv \{\text{Lemma 5.1 with } v:=R.q.v \text{ and } f:=g\} \\
& \qquad Q.(L.g.(R.q.v)).a = Q.(R.q.(L.g.v)).a \\
& \quad \Leftarrow \{\text{equals for equals}\} \\
& \qquad (\forall t \in A^* :: L.g.(R.q.v).t = R.q.(L.g.v).t) \\
& \quad \equiv \{\text{Lemma 5.1 and (17)}\} \\
& \qquad (\forall t \in A^* :: R.q.v.t \circ g = L.g.v.(t;q)) \\
& \quad \equiv \{(17) \text{ and Lemma 5.1}\} \\
& \qquad \text{true.}
\end{aligned}
$$

This concludes the proof of (24) and hence of (23). □

Remark. In this proof, one can go further and show that y is the least fixpoint of function E. This implies that function wk is multiplicative. Then the proof of our next result becomes simpler. The extension required, however, is longer than the gain in the proof of the next result. Since the latter also implies that wk is multiplicative, we prefer the present order of presentation.

Theorem 5.4. $wk = wa^*$.

Proof. The equality is proved by means of three inequalities

$$(wk|A)^* \leqslant wk \leqslant wa^* \leqslant (wk|A)^*, \tag{25}$$

where $(wk|A)^*$ is the multiplicative extension of the restriction of wk to the set A.

Since wk is a solution of (18) we have $wk.\varepsilon = identity$. Using Lemma 5.3 and induction on the length of string t, we then get $(wk|A)^*.t \leqslant wk.t$ for every string t. This is the first inequality of (25).

For the second inequality, we show that function wa^* is a solution of equation (18). Since wa^* is multiplicative, we have $wa^*.\varepsilon = identity$. In order to verify the other

conjunct of (18), we first observe that

$$
\begin{aligned}
&(\forall s \in A^* :: R.q.wa^*.s = L.(wa^*.q).wa^*.s) \\
&\quad \equiv \{(17) \text{ and Lemma } 5.1\} \\
&\qquad (\forall s \in A^* :: wa^*.(s;q) = wa^*.s \circ wa^*.q) \\
&\quad \equiv \{wa^* \text{ is multiplicative}\} \\
&\qquad \text{true.}
\end{aligned} \tag{26}
$$

Now function wa^* satisfies the second conjunct of (18) because of

$$
\begin{aligned}
&wa^*.(a;q) = Q.(R.q.wa^*).a \\
&\quad \equiv \{wa^* \text{ is multiplicative and (26)}\} \\
&\qquad wa^*.a \circ wa^*.q = Q.(L.(wa^*.q).wa^*).a \\
&\quad \equiv \{\text{Lemma } 5.1\} \\
&\qquad wa^*.a \circ wa^*.q = Q.wa^*.a \circ wa^*.q \\
&\quad \equiv \{a \in A \text{ and } wa \text{ solves (16)}\} \\
&\qquad \text{true.}
\end{aligned}
$$

This proves that wa^* solves (18). Since wk is the least solution of (18), this implies $wk \leqslant wa^*$, the second inequality of (25).

Since wk satisfies (18) with $q := \varepsilon$, we have $wk.a = Q.wk.a$ for all $a \in A$. From the first inequality of (25) and the fact that function Q is monotone, it follows that

$$Q.(wk|A)^* \leqslant Q.wk = wk|A.$$

From (19) it now follows that the least solution wa of (16) satisfies

$$
\begin{aligned}
&wa \\
&\quad = \{(19), (16)\} \\
&\qquad (\inf \omega : Q.\omega^* \leqslant \omega : \omega) \\
&\quad \leqslant \{Q.(wk|A)^* \leqslant (wk|A)\} \\
&\qquad (wk|A).
\end{aligned}
$$

Since multiplicative extension is monotone, this implies $wa^* \leqslant (wk|A)^*$, the third inequality of (25). □

Theorem 5.4 implies that wk and wp are equal for all command expressions on which both are defined. In particular, these functions give the same semantics to every procedure name. It also follows that function wk is multiplicative.

6. Unfolding

Up to this point, we assumed that the declaration **body** was fixed and satisfied condition (7). From the point of view of programming methodology this is an awkward assumption. So we need a formalism to compare different declarations, possibly with different sets of procedure names. More precisely, the unfolding may use more procedure names than the original declaration.

Let $K \subseteq H$. Put $B = S \cup K$ so that B is a subset of A. Let $Cmdk$ be the subclass of Cmd generated by B(in the same way as Cmd is generated by A). A declaration $\varphi \in H \to Cmd$ will be called an *unfolding* of a declaration $\psi \in K \to Cmdk$ if, for every $h \in K$, the body $\psi.h$ can be obtained from the body $\varphi.h$ by, repeatedly, replacing some procedure name h' in $\varphi.h$ by its body $\varphi.h'$. Before giving the formal definition, we first give an example.

Example. Take $K = \{h0\}$ and $H = \{h0, h1\}$. Let c, d, e and f be simple commands. Let the declarations φ and ψ be given by

$$\psi.h0 = c \diamond d; (e \,[]\, f; h0),$$

$$\varphi.h0 = c \diamond d;\ h1,$$

$$\varphi.h1 = e \,[]\, f; h0.$$

Then φ is an unfolding of ψ, since $\psi.h0$ can be obtained from $\varphi.h0$ by replacing $h1$ by its body $\varphi.h1$ (just once, here). □

The formal definition of unfolding is as follows. We use the auxiliary concept of admissible preorders.

Recall that a *preorder* is a binary relation that is reflexive and transitive. Let a preorder $\lhd$ on Cmd be called *admissible* if and only if sequential composition, angelic choice and demonic choice are all monotonic with respect to $\lhd$, in the sense that

- $s \lhd s' \wedge t \lhd t' \Rightarrow s; t \lhd s'; t'$ for all $s, s', t, t' \in Cmd$,
- if $(i :: s.i)$ and $(i :: t.i)$ are families of commands with index i ranging over the same set and $s.i \lhd t.i$ for all i, then $([]\, i :: s.i) \lhd ([]\, i :: t.i)$ and $(\diamond\, i :: s.i) \lhd (\diamond\, i :: t.i)$.

Let $\varphi \in H \to Cmd$ be a declaration. The *rewrite preorder* $\lhd_\varphi$ on Cmd is such that $s \lhd_\varphi t$ expresses that command t can be obtained from s by replacing some procedure names h in s by their bodies $\varphi.h$. Relation $\lhd_\varphi$ is formally defined as the least admissible preorder that satisfies $h \lhd_\varphi \varphi.h$ for all $h \in H$. Declaration φ is called an *unfolding* of a declaration $\psi \in K \to Cmdk$ if and only if $\varphi.h \lhd_\varphi \psi.h$ for all $h \in K$.

The reader may verify that this adequately formalizes the concept of unfolding described in Section 2. We now have to show that an unfolding φ of a declaration ψ induces the same semantics. For this purpose the dependence of wp on the declaration is made explicit by subscription with the relevant declaration. So we write $wp_\varphi \in Cmd \to MT$ and $wp_\psi \in Cmdk \to MT$ to denote wp induced by φ and ψ, respectively.

Theorem 6.1. *Let φ be an unfolding of ψ. Then wp_ψ is equal to the restriction of wp_φ to Cmdk.*

Proof. We first recall the definitions: $wp_\varphi = wa^\odot \in Cmd \to MT$ where $wa \in A \to MT$ is the least solution of (6), i.e., the least fixpoint of function $D \in MT^A \to MT^A$ given by

$$D.\omega.s = ws.s \quad \text{for } s \in S,$$

$$D.\omega.h = \omega^\odot.(\varphi.h) \quad \text{for } h \in H.$$

Similarly, $wp_\psi = wb^\odot \in Cmdk \to MT$ where $wb \in B \to MT$ is the least fixpoint of function $E \in MT^B \to MT^B$ given by

$$E.\omega.s = ws.s \quad \text{for } s \in S,$$

$$E.\omega.h = \omega^\odot.(\psi.h) \quad \text{for } h \in K.$$

Note that the operator $^\odot$ is overloaded: it is used to form the homomorphic extension from A to *Cmd*, and also from B to *Cmdk*. This overloading is harmless in the sense that, for $\omega \in MT^A$, the restriction of $\omega^\odot$ *to Cmdk* equals the extension $(\omega|B)^\odot$. This equality implies that it is sufficient to prove that $wb = (wa|B)$.

We use the unfolding property to prove that, for every $\omega \in MT^A$,

$$\begin{aligned} &\omega \leqslant D.\omega \;\Rightarrow\; ((D.\omega)|B) \leqslant E.(\omega|B), \\ &\omega \geqslant D.\omega \;\Rightarrow\; ((D.\omega)|B) \geqslant E.(\omega|B). \end{aligned} \tag{27}$$

In fact, assume $\omega \leqslant D.\omega$. This implies $\omega^\odot.h \leqslant \omega^\odot.(\varphi.h)$ for all $h \in H$, and hence $h \lhd \varphi.h$ for all $h \in H$, where relation $\lhd$ on *Cmd* is defined by

$$s \lhd t \;\equiv\; \omega^\odot.s \leqslant \omega^\odot.t.$$

It is easy to verify that relation $\lhd$ is an admissible preorder. Since $\lhd_\varphi$ is defined as the least admissible preorder that satisfies $h \lhd_\varphi \varphi.h$ for all $h \in H$, this implies $\lhd_\varphi \subseteq \lhd$. Since φ is an unfolding of ψ, the definition of unfolding yields that $\varphi.h \lhd_\varphi \psi.h$ for all $h \in K$ and hence $\varphi.h \lhd \psi.h$ for all $h \in K$. In other words, we have

$$\omega^\odot.(\varphi.h) \leqslant \omega^\odot.(\psi.h) \quad \text{for all } h \in K.$$

By the definitions of D and E, this implies $((D.\omega)|B) \leqslant E.(\omega|B)$. This proves the first formula of (27). The proof of the second one is completely analogous.

We now show how (27) is used to prove $wb = (wa|B)$. Since $wa = D.wa$, it follows from (27) that $E.(wa|B) = ((D.wa)|B) = wa|B$, so that $wa|B$ is a fixpoint of E. Since wb is the least fixpoint of E, this implies $wb \leqslant (wa|B)$. In order to prove the other inequality, we let X be the subset of MT^A that consists of the functions ω with

$$\omega \leqslant D.\omega \wedge (\omega|B) \leqslant wb.$$

In order to prove that $wa \in X$, we first verify that X is D-invariant. In fact, for every $\omega \in X$, we have $D.\omega \in X$ because of

$$D.\omega \leqslant D.(D.\omega) \wedge ((D.\omega)|B) \leqslant wb$$

$$\Leftarrow \quad \{wb \text{ fixpoint of } E \text{ and transitivity}\}$$

$$D.\omega \leqslant D.(D.\omega) \wedge ((D.\omega)|B) \leqslant E.(\omega|B) \wedge E.(\omega|B) \leqslant E.wb$$

$$\Leftarrow \quad \{\text{monotony and (27)}\}$$

$$\omega \leqslant D.\omega \wedge (\omega|B) \leqslant wb.$$

It is easy to verify that the set X is sup-closed, so by the version of the theorem of Knaster–Tarski given after formula (19), this implies that the least fixpoint of D is an element of X. This proves that $wa \in X$, so $(wa|B) \leqslant wb$ and hence $(wa|B) = wb$. □

7. Weakest liberal preconditions

We presented above the case of total correctness, where the weakest precondition of a command expression implies its termination (the angel has a winning strategy). It is often useful also to consider conditional correctness, represented by the weakest liberal precondition function *wlp*. In terms of the game theory, $wlp.c.p$ is the predicate that the angel need not lose. The simplest way to incorporate *wlp* into the theory is to define *wlp* as the multiplicative extension of the greatest solution of equation (6), (14) or (16). Indeed, least fixpoints can be replaced by greatest fixpoints throughout the theory.

This elegant solution has the drawback that it implies that every simple command c satisfies $wlp.c = wp.c$, so that all simple commands always terminate. The alternative is to introduce two functions for the interpretation of the simple commands, say $ws_0, ws_1 \in S \to MT$ and two corresponding functions Q_0 and Q_1 as defined in formula (15). Then *wp* is defined by means of least fixpoints with respect to Q_0 and *wlp* is defined by means of greatest fixpoints with respect to Q_1. The remainder of the theory of Sections 1, 2, 5 and 6 can be preserved.

It should be mentioned, however, that conditional correctness is not a very useful concept in combination with the angelic choice. In particular, separate assertions of conditional correctness and termination do not combine to total correctness, as is shown in the next example.

Example. Let j be an integer program variable and let repetition L be given by

$$L \;=\; \textbf{while } \mathrm{j} \neq 0 \textbf{ do } \varepsilon \diamond \mathrm{j} := \mathrm{j} - 1 \textbf{ od}.$$

In the unfolded syntax, this command is represented by taking procedure names $L \in Hd$ and $K \in Ha$ with declaration

$$\mathbf{alt}.L = \{?(\mathrm{j} \neq 0); K; L,\ ?(\mathrm{j} = 0)\},$$

$$\mathbf{alt}.K = \{\varepsilon,\ \mathrm{j} := \mathrm{j} - 1\}.$$

Operationally, it is easy to see that

$$wp.L.true = (\mathrm{j} \geqslant 0), \tag{28}$$

$$wlp.L.(\mathrm{j} \neq 0) = (\mathrm{j} \neq 0),$$

$$wp.L.(\mathrm{j} \neq 0) = false. \tag{29}$$

It follows that $wp.L.(\mathrm{j} \neq 0)$ differs from $wp.L.true \wedge wlp.L.(\mathrm{j} \neq 0)$. The point is that the angelic choices needed for reaching termination in (28) differ from those needed for conditional correctness in (29). □

8. Conclusions

We have related four semantic models for a programming language with recursion and with both (possibly unbounded) demonic and angelic choice. The four models are:

- a compositional fixpoint semantics for an arbitrary declaration,
- a compositional fixpoint semantics for a completely unfolded declaration,
- a continuation semantics, called the stack interpretation,
- a game-theoretic semantics.

The four models have been proved to be equivalent.

Acknowledgment

I gratefully acknowledge the contributions of J.E. Jonker to the proof of Theorem 5.4 and of A. Thijs to the concepts and results of Section 6. The criticisms of two referees have led to great improvements of the presentation.

References

[1] R.J.R. Back and J. von Wright, Refinement calculus, Part I: sequential nondeterministic programs, in: J.W. de Bakker, W.-P. de Roever and G. Rozenberg, eds., *Stepwise Refinement of Distributed Systems*, Lecture Notes in Computer Science, Vol. 430 (Springer, Berlin, 1990) 42–66.

[2] R.J.R. Back and J. von Wright, Duality in specification languages: a lattice theoretical approach, *Acta Inform.* **27** (1990) 583–625.

[3] J.W. de Bakker, Een bewijsmethode voor recursieve procedures, *Math. Centre Syllabus* **21** (VII) (1975) 111–126 (in Dutch).

[4] A. Chandra, D. Kozen and L. Stockmeyer, Alternation, *J. ACM* **28** (1981) 114–133.

[5] E.W. Dijkstra, *A Discipline of Programming* (Prentice-Hall, Englewood Cliffs, NJ, 1976).
[6] E.W. Dijkstra and C.S. Scholten, *Predicate Calculus and Program Semantics* (Springer, Berlin, 1990).
[7] D. Harel, Dynamic logic, in: D. Gabbay, F. Guenthner, eds., *Handbook of Philosophical Logic, Vol.* 2 (Reidel, Dordrecht, 1984) 497–604.
[8] W.H. Hesselink, Modalities of nondeterminacy, in: W.H.J. Feijen et al. eds., *Beauty is Our Business, a Birthday Salute to Edsger W. Dijkstra* (Springer, Berlin, 1990) 182–192.
[9] W.H. Hesselink, *Programs, Recursion and Unbounded Choice* (Cambridge Univ. Press, Cambridge 1992).
[10] W.H. Hesselink, Processes and formalisms for unbounded choice, *Theoret. Comput. Sci.* **99** (1992) 105–119.
[11] W.H. Hesselink and R. Reinds, Temporal preconditions of recursive procedures, in: J.W. de Bakker, W.-P. de Roever and G. Rozenberg, eds., *Semantics: Foundations and Applications; Proc. REX Workshop June 1992*, Lecture Notes in Computer Science, Vol. 666 (Springer, Berlin, 1993) 236–260.
[12] C.A.R. Hoare, *Communicating Sequential Processes* (Prentice-Hall, Englewood Cliffs, NJ, 1985).
[13] C. Morgan, *Programming from Specifications* (Prentice-Hall, Englewood Cliffs, NJ, 1990).
[14] C. Morgan and P.H.B. Gardiner, Data refinement by calculation, *Acta Inform.* **27** (1990) 481–503.
[15] J.M. Morris, A theoretical basis for stepwise refinement and the programming calculus, *Sci. Comput. Programming* **9** (1987) 287–306.
[16] N.J. Nilsson, *Principles of Artificial Intelligence* (Springer, Berlin, 1982).
[17] A. Tarski, A lattice theoretical fixpoint theorem and its applications, *Pacific J. Math.* **5** (1955) 285–309.