Formal Aspects of Computing (1999) 11: 616–636

Formal Aspects of Computing

Predicate Transformers for Recursive Procedures with Local Variables

Wim H. Hesselink

Department of Mathematics and Computing Science, Rijksuniversiteit Groningen, The Netherlands

Keywords: Predicate transformers; Frames; Recursive procedures; Proof rule

Abstract. The weakest precondition semantics of recursive procedures with local variables are developed for an imperative language with demonic and angelic operators for unbounded nondeterminate choice. This does not require stacking of local variables. The formalism serves as a foundation for a proof rule for total correctness of (mutually) recursive procedures with local variables. This rule is illustrated by a simple example. Its soundness is proved for arbitrary well-founded variant functions.

1. Introduction

In imperative programming, procedures are complex commands designed to satisfy a given specification. We regard Hoare triples as the most adequate way to specify procedures and other commands. One can use Hoare logic to define derivability of Hoare triples, but weakest preconditions form a more convenient semantic formalism that is sufficiently close to Hoare triples. We therefore use Hoare triples in our correctness rule for recursive procedures and then prove this rule by means of weakest preconditions. The proof of this correctness rule is a kind of case study that reveals many interesting points. We do not aim at a general description of a rich procedural language.

Formalisms for weakest preconditions for imperative programs usually treat predicates as boolean functions on a single state space, cf. [BvW90, DiS90, Hes92, Nel89]. It follows that procedures cannot have local variables. For nonrecursive procedures, local variables can be made global by careful renaming. Actually, this

Correspondence and offprint requests to: Wim H. Hesselink, Department of Mathematics and Computing Science, Rijksuniversiteit Groningen, P.O.Box 800, 9700 AV Groningen, The Netherlands. Email: wim@cs.rug.nl, Web site: http://www.cs.rug.nl/~wim.

is also possible for recursive procedures by means of methods analogous to those developed below. A major drawback, however, is the lack of abstractness: it is preferable that local variables be hidden from global arguments, cf. [OHT95]. The present paper, therefore, is devoted to a clean treatment of weakest preconditions of recursive procedures with local variables.

The urge to do so was triggered by a recent investigation in the semantics of object-oriented languages [AbL98] and the predicate transformer approach to Z of [Mah99]. Even more urgent to us was the realization that, in [Hes93], we had applied and advocated a proof rule for total correctness of recursive procedures that, in the presence of local variables, lacked a firm semantic foundation. Such a foundation is supplied here.

The language treated consists of commands constructed from basic commands (e.g., assignments, guards, assertions) by means of operators for sequential composition, Dijkstra's demonic choice, the angelic choice of [BvW90, Hes90, MoG90], together with procedures that may have local variables and may be mutually recursive.

We construct predicate transformation semantics for this language. The same was done for the language without local variables in [Hes94]. There we also constructed a corresponding operational semantics, based on a kind of game theory, similar to the alternating Turing machines of [CKS81]. This approach is possible here as well, but it requires a stack for the local variables, cf. [AbL98], and the proof of adequacy will be tedious.

We introduce pairs of frames to determine the accessible and the modifiable variables. We use extension and abstraction functions to transform predicate transformers defined over one pair of frames to another pair of frames. We finally show that the operators for extension, abstraction and composition of predicate transformers have the natural operational interpretations when restricted to relationally defined predicate transformers.

1.1. Overview

In Section 2, we briefly discuss related work. In Section 3, we introduce frames and the corresponding state spaces and spaces of predicates. We then introduce framed Hoare triples to specify commands, present a correctness rule for parameterless recursive procedures with local variables, and give an example to show the application of this rule. In Section 4, we generalize this rule to a theorem about the correctness of specifications of mutually recursive procedures, where termination is guaranteed by means of an arbitrary well-founded ordering. The theorem is proved by assuming that Hoare triples are based on weakest preconditions and that they satisfy certain rules.

In order to underpin these rules, we build in Section 5 a theory of extension and abstraction between predicate transformer monoids for frame pairs. Pairs of frames are considered here in order to distinguish variables that can be modified from variables that can only be inspected. We abstain from introducing categories and functors, but this section has a strong categorical flavour and can be generalized widely.

In Section 6 we develop the formal programming language and its weakest precondition semantics, which consist of a family of predicate transformers indexed by frame pairs. In Section 7, we come back to Hoare triples and prove the postulates needed for the recursion theorem.

Section 8 deals with parts of the adequacy problem. We show that, for relationally defined predicate transformers, extension, abstraction and composition correspond to the natural relational definitions. Section 9 contains concluding remarks.

1.2. Notation

We write $X \to Y$ for the set of functions from X to Y. Function application is denoted by an infix dot, which binds stronger than other binary operators and binds to the left; e.g., we have $\xi.h.p.x = ((\xi.h).p).x$.

2. Review of Existing Work

The semantics of procedural languages with local variables have been investigated and explained in categorical terms in [Ole85, OHT95] and the references given there. In these papers the languages get meaning by means of functors from the category of the state spaces or frames to certain semantic sets or domains. They concentrate on semantical models for arbitrary procedures, not on proving that a given procedure satisfies a given specification. The languages allowed are much richer than ours.

The recent book of Back and Von Wright [BvW98] treats recursive procedures with reference parameters, value parameters, and local variables. The procedures are transformed in such a way that the local variables are replaced by value parameters. This approach is less elegant than the categorical approach via variable state spaces or frames. The latter approach does occur in the recent papers [BaB98, Mah99]. Our technical contributions are frame pairs and the three operators ξ, ρ, $\bullet$. We refer to [Hes93] for references to papers on proof rules for recursive procedures.

3. Frames and Hoare Triples

A *frame* F is a set of typed program variables, e.g. variable v with type $T.v$, where a type is a nonempty set. The *state space* corresponding to frame F is therefore the cartesian product $[F] = (\Pi\ v \in F :: T.v)$. It follows that $[F]$ is always nonempty. For the empty frame $\emptyset$, the state space $[\emptyset]$ is a one-point set.

We want to perform program verification by means of preconditions and postconditions. So, we have to consider predicates on state spaces.

For a frame F, the boolean functions on the state space $[F]$ are called predicates over F. We write $\mathbb{P}.F = ([F] \to \mathbb{B})$ for the set of predicates over F. The boolean operators $\wedge$, $\vee$, $\neg$ are lifted to $\mathbb{P}.F$ in the usual pointwise fashion, i.e., $(p \wedge q).x = p.x \wedge q.x$ for all $x \in [F]$, etc. We use the equality sign on $\mathbb{P}.F$ for equality of functions. The set $\mathbb{P}.F$ is ordered by universal implication, which is denoted by $F \models p \leqslant q$ for $p, q \in \mathbb{P}.F$; so we have

$$(0) \qquad (F \models p \leqslant q) \equiv (\forall\, x \in [F] :: p.x \Rightarrow q.x)\ .$$

It is well known that $\mathbb{P}.F$ with this ordering is a complete boolean lattice.

We now consider frames F and G with $G \subseteq F$. An element $x \in [F]$ has a natural *projection* $(x|G) \in [G]$. For $x \in [F]$ and $y \in [G]$, the *update* of x according to y along G is defined as the element $(x; G : y)$ of $[F]$ given by $(x; G : y)|G = y$ and $(x; G : y)|(F \setminus G) = x|(F \setminus G)$.

We use " _ " at the position of a variable as a shorthand for functional abstraction. E.g., if $x \in [F]$ then $(x; G : \_)$ is a function from $[G]$ to $[F]$, the *update* function of x. Also, any predicate p over G induces a predicate $p \circ (\_|G)$ over F, to be called the *extension* of p. If p is given as a syntactic expression in the variables of G, the same syntactic expression can be used for the extension.

Example. Consider frames $F = \{\mathtt{j} : \mathbb{Z}, \mathtt{k} : \mathbb{N}\}$ and $G = \{\mathtt{k} : \mathbb{N}\}$. Then $[F] = \mathbb{Z} \times \mathbb{N}$ and $[G] = \mathbb{N}$. A state $(s, t) \in [F]$ has the projection t in $[G]$. Predicate $p = (\mathtt{k} > 3)$ over G extends to a predicate with the same denotation over F. □

For each command c, we introduce frames $D_0.c$ and $D_1.c$ with D standing for declaration. The *modifiable* frame $D_0.c$ consists of the variables that c is allowed to modify. The *accessible* frame $D_1.c$ consists of the variables that c has access to (may read). We assume $D_0.c \subseteq D_1.c$.

The call of a procedure h is a command and has therefore two frames $D_0.h \subseteq D_1.h$. Procedure h is declared with a body **body**.h, which is a command with two frames $B_i.h = D_i.(\mathbf{body}.h)$ for $i = 0$, 1. The body must have the same external access rights as the call. So, we postulate that $D_i.h \subseteq B_i.h$ for $i = 0$, 1. We also postulate that all external variables modifiable in the body are externally modifiable, i.e., $D_1.h \cap B_0.h = D_0.h$. The elements of $B_1.h \setminus D_1.h$ are local variables of h.

On the semantic level, the presence of local variables forces us to distinguish the state spaces where procedure h and its body are acting. The body of h uses the state space $[B_1.h]$, but the semantics of h is an abstraction that restricts the attention to $[D_1.h]$. On the other hand, the meaning of a call of h requires an extension of the semantics of h to the frame of the variables relevant at the position of the call. So we need to define the concepts of abstraction and extension of semantics.

A program is a rooted tree of procedure declarations with the root *main*. We write H to denote the set of procedure names. If procedure k is a child of procedure h, we must have $D_1.k \subseteq B_1.h$ and $D_0.k \subseteq B_0.h$. Child k can be called in the body of h, and also in the bodies of all descendants h' of h with $D_1.k \subseteq D_1.h'$ and $D_0.k \subseteq D_0.h'$. In fact, external variables of k cannot be local variables of h'.

We now introduce framed Hoare triples to specify commands. If c is a command that only refers to program variables in a frame F (i.e. $D_1.c \subseteq F$), and P and Q are predicates over F, or of a subframe of F, the *framed Hoare triple*

$$F \;\models\; \{P\}\; c\; \{Q\}$$

is interpreted to mean that, in state space $[F]$, every execution of command c that starts in a state where P or its extension holds, terminates in a state where Q or its extension holds. So, we use framed Hoare triples for total correctness, and the predicates in a framed Hoare triple are extended to the frame.

Following [Dij76], we specify a declaration of a parameterless procedure h by a heading of the form

(1) $$\begin{array}{l}\mathbf{proc}\ h \\ \{\ \mathbf{glovar}\ Fm,\ \mathbf{glocon}\ Fc\ ;\ \mathbf{all}\ i \in I\ ::\ \mathbf{pre}\ P.i\ ,\ \mathbf{post}\ Q.i\ \}\ .\end{array}$$

List Fm holds the modifiable external variables, so that $D_0.h = Fm$. List Fc holds the constant external variables, i.e., the external variables that can be accessed but must not be modified. We thus have $D_1.h = Fm \cup Fc$. The specification means that $P.i$ and $Q.i$ are predicates over $D_1.h$ such that procedure h satisfies the framed Hoare triples $D_1.h \models \{P.i\}\ h\ \{Q.i\}$, for all $i \in I$.

An implementation of h consists of a frame Fl of local variables, disjoint from $Fm \cup Fc$ and a command **body** that only uses variables in $B_1.h = Fm \cup Fc \cup Fl$, and only modifies variables in $B_0.h = Fm \cup Fl$. Total correctness of a possibly recursive implementation can be verified by means of the rule:

(2) **Correctness Rule.** *Let, for all $i \in I$, integer valued functions $vf.i$ in the external variables be given. Consider, for given integer m, the induction hypothesis $IH(m)$ asserting, for all values $i \in I$ and all predicates $R \in \mathbb{P}.(Fc \cup Fl)$, the validity of the framed Hoare triples*

$$B_1.h \models \{\ P.i \quad \wedge \quad vf.i < m \quad \wedge \quad m \geqslant 0 \quad \wedge \quad R\ \} \\ h \quad \{\ Q.i \quad \wedge \quad R\ \}\ . \tag{3}$$

Assume that, for every integer m, hypothesis $IH(m)$ implies, for every $i \in I$,

$$B_1.h \models \{P.i \quad \wedge \quad vf.i = m\} \quad \mathbf{body} \quad \{Q.i\}\ . \tag{4}$$

Then the Hoare triples of specification (1) *are correct.*

This rule is a "framed" version of rule (10) of [Hes93]. That rule also allows value parameters and reference parameters. For simplicity, we have omitted these here. In applications of the rule, the induction hypothesis is used to handle the recursive calls of h in its body. The induction hypothesis is only useful if the recursive calls occur in a context with a smaller value of $vf.i$. For negative m, the induction hypothesis is vacuously true and therefore useless. So, then, recursive calls should not occur.

Note that the conjunctions in the precondition and the postcondition of (3) require that predicates $P.i$ and $Q.i$ be extended from frame $D_1.h$ to $B_1.h$ and that R be extended from $Fc \cup Fl$ to $B_1.h$. The first conjunction in the precondition of (3) can be taken both before and after extension. This ambiguity is harmless since, as is easily seen, extension commutes with all logical operators.

We prove rule (2) in Section 4 below, but let us first give an application to show how the rule works out in the presence of recursion and local variables.

Example. We give a simple example with integers where a local variable must be retained during a recursive call. Let q be a constant with $q > 1$ and let H be the function of two integer arguments given by

$$\begin{aligned} H(y,z) &= z\ ,\ \text{ if } y \leqslant 0\ , \\ H(y,z) &= q * H(y \ \mathbf{div}\ q, z) + y \ \mathbf{mod}\ q\ ,\ \text{ if } y > 0\ . \end{aligned}$$

We claim that function H is computed by

```
proc rev
{ glovar y, z ; all i ∈ ℤ :: pre i = H(y, z) , post i = z }
    var x : integer ;
    if y > 0 then
        x := y mod q ;
        y := y div q ;
        rev ;
        z := q * z + x ;
    fi
endproc .
```

Note how the specification constant i is used to relate the final value of z with the initial values of y and z. For simplicity, we have no constant external variables; we therefore omit **glocon** and have $D_0.rev = D_1.rev$. Operationally, the procedure needs a stack for the local variables x. Yet the formal semantics can do without the stack. Correctness of the declaration is proved by means of proof rule (2) as follows.

We choose function $vf = \mathrm{y}$, independent of specification constant i. The accessible frame of the body is $B_1.rev = \{\mathrm{y}, \mathrm{z}, \mathrm{x}\}$. The induction hypothesis analogous to (3) is that for all values i, and all predicates $R \in \mathbb{P}.\{\mathrm{x}\}$, we have

$$\begin{array}{l} B_1.rev \ \models \ \{\ i = H(\mathrm{y},\mathrm{z}) \quad \wedge \quad \mathrm{y} < m \quad \wedge \quad m \geqslant 0 \quad \wedge \quad R\ \} \\ \qquad rev \quad \{\ i = \mathrm{z} \quad \wedge \quad R\ \}\ . \end{array}$$

Now it suffices to verify the correctness of the analogue of (4), i.e., of the following annotated procedure body.

$$\begin{array}{l}
B_1.rev \ \models \ \{\ i = H(\mathrm{y},\mathrm{z}) \quad \wedge \quad \mathrm{y} = m\ \} \\
\quad \textbf{if} \ \ \mathrm{y} > 0 \ \ \textbf{then} \\
\qquad\qquad \{\ i = q * H(\mathrm{y}\ \textbf{div}\ q, \mathrm{z}) + \mathrm{y}\ \textbf{mod}\ q \quad \wedge \quad 0 < \mathrm{y} = m\ \}\ \ ; \\
\qquad \mathrm{x} := \ \mathrm{y}\ \textbf{mod}\ q\ ; \\
\qquad \mathrm{y} := \ \mathrm{y}\ \textbf{div}\ q\ ; \\
\qquad\qquad \{\ i = q * H(\mathrm{y},\mathrm{z}) + \mathrm{x} \quad \wedge \quad 0 \leqslant \mathrm{y} < m\ \}\ \ ; \\
\qquad\qquad\quad (*\ \ \text{introduce a new specification constant } j\ \ *) \\
\qquad\qquad \{\ j = H(\mathrm{y},\mathrm{z}) \quad \wedge \quad 0 \leqslant \mathrm{y} < m \quad \wedge \quad i = q * j + \mathrm{x}\ \} \\
\qquad rev\ \ ; \quad (*\ \ \text{induction hypothesis with } i := j \text{ and} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad R :\ i = q * j + \mathrm{x}\ \ *) \\
\qquad\qquad \{\ j = \mathrm{z} \quad \wedge \quad i = q * j + \mathrm{x}\ \} \\
\qquad \mathrm{z} := \ q * \mathrm{z} + \mathrm{x}\ ; \\
\qquad\qquad \{\ i = \mathrm{z}\ \} \\
\quad \textbf{else} \quad \{\ i = H(\mathrm{y},\mathrm{z}) \quad \wedge \quad \mathrm{y} \leqslant 0\ \} \\
\qquad\qquad \{\ i = \mathrm{z}\ \} \\
\quad \textbf{fi} \quad (*\ \ \text{collect branches}\ \ *) \\
\quad \{\ i = \mathrm{z}\ \}\ \ .
\end{array}$$

Note that predicate R indeed only uses program variable x. The same proof can be used if q is treated as an external variable in Fc. Then, indeed, q is also allowed to occur in $R \in \mathbb{P}.[Fc \cup Fl]$. □

There is a second point to Hoare triples: one does not only want to prove them, one also wants to apply them. We therefore need the following rule.

(5) **Extension rule.** Assume that command c over frame F satisfies Hoare triple $F \models \{P\}\ c\ \{Q\}$.
(a) Then $G \models \{P\}\ c\ \{Q\}$ holds for every frame G that contains F.
(b) If R is a predicate over $G \setminus D_0.c$, then it holds that $G \models \{P \wedge R\}\ c\ \{Q \wedge R\}$.

Rule (5) will be proved in Section 7.

4. Well-founded Triples and the Recursion Theorem

In this section, we generalize correctness rule (2) and prove the generalization from more basic principles, namely from extension rule (5), the existence of weakest preconditions, and the "equivalence" of a procedure with its body. The generalization is to mutual recursion and arbitrary well-founded triples. In fact, mutual recursion is useful to accomodate value parameters, and termination arguments of recursive procedures sometimes need arbitrary well-founded triples, compare [DiG86].

A *well-founded triple* consists of a set Z, a binary relation $<$ on Z and a subset N of Z such that every nonempty subset S of N has a minimal element with respect to $<$; here $s \in S$ is called a minimal element of S iff $t \notin S$ for every element $t \in Z$ with $t < s$.

Remarks. The standard example is that Z is the set of the integers, N the set of the natural numbers, and $<$ is the ordinary "less than" relation. In general, however, relation $<$ need not be transitive. We include Z in the triple for greater flexibility. E.g., in (2), function $vf.i$ may have negative values. □

The principle of well-founded induction over the triple $(Z, <, N)$ states that, for any boolean function $f \in Z \to \mathbb{B}$,

$$(6) \qquad \begin{array}{l} (\forall\, m \in Z \,:\, m \notin N \ \vee\ (\forall\, n \in Z \,:\, n < m : f.n) : f.m) \\ \Rightarrow \quad (\forall\, m \in Z \,::\, f.m)\ . \end{array}$$

We now lift this principle to families of predicates over various frames. This lifted version is the key to the recursion theorem below. Consider a family of triples $(F.i, p.i, q.i)$ consisting of frames $F.i$ and predicates $p.i,\ q.i \in \mathbb{P}.(F.i)$ for every $i \in I$. We let i range over I.

(7) **Local well-founded induction.** Let $vf.i \in [F.i] \to Z$ be a family of functions. Assume that, for every $m \in Z$, it holds that

$$(8) \qquad \begin{array}{l} (\forall\, i :: F.i \models (p.i \ \wedge\ vf.i < m \ \wedge\ m \in N) \leqslant q.i) \\ \Rightarrow \quad (\forall\, i :: F.i \models (p.i \ \wedge\ vf.i = m) \leqslant q.i)\ . \end{array}$$

Then we have $F.i \models p.i \leqslant q.i$ for all $i \in I$.

Proof. We define $f.m = (\forall\, i :: F.i \models (p.i \wedge vf.i = m) \leqslant q.i)$ for $m \in Z$. Then our proof obligation satisfies

$$\begin{array}{ll} & (\forall\, i :: F.i \models p.i \leqslant q.i) \\ \equiv & \{ \text{ definition } \leqslant\text{; let } x \text{ range over } [F.i] \ \} \\ & (\forall\, i, x :: p.i.x \quad \Rightarrow \quad q.i.x) \\ \equiv & \{ \text{ one-point rule, let } m \text{ range over } Z \ \} \\ & (\forall\, i, x, m :: p.i.x \ \wedge\ vf.i.x = m \quad \Rightarrow \quad q.i.x) \\ \equiv & \{ \text{ definition of } f \text{ and } \leqslant \} \\ & (\forall\, m :: f.m)\ . \end{array}$$

By rule (6), it now suffices to verify, for arbitrary $m \in Z$, that

$$\begin{array}{ll} & f.m \\ \Leftarrow & \{ \text{ definition of } f \text{ and (8) } \} \\ & (\forall\, i :: F.i \models (p.i \ \wedge\ vf.i < m \ \wedge\ m \in N) \leqslant q.i) \\ \equiv & \{ \text{ definition of } \leqslant \} \\ & (\forall\, i, x :: (p.i.x \ \wedge\ vf.i.x < m \ \wedge\ m \in N) \quad \Rightarrow \quad q.i.x) \\ \Leftarrow & \{ \text{ calculus, let } n \text{ range over } Z \ \} \\ & m \notin N \quad \vee \quad (\forall\, n, i, x :: p.i.x \ \wedge\ vf.i.x = n \ \wedge\ n < m \quad \Rightarrow \quad q.i.x) \\ \Leftarrow & \{ \text{ definition of } f \text{ and } \leqslant \} \\ & m \notin N \quad \vee \quad (\forall\, n \in Z \,:\, n < m : f.n) \end{array}$$

□

Remark. The antecedent of (8) is called the induction hypothesis. The negation of the conjunct $m \in N$ in the lefthand side of this induction hypothesis serves as the base case of the induction. □

In order to prove a generalization of rule (2), we postulate a relation between the meaning of a procedure and its body.

(9) **Body rule.** For predicates P, Q over $D_1.h$, we have

$$\begin{aligned}&(D_1.h \models \{P\} \quad h \quad \{Q\}\ ) \\ \equiv \quad &(B_1.h \models \{P\} \quad \mathbf{body}.h \quad \{Q\}\ )\ .\end{aligned}$$

We also postulate that the framed Hoare triples are defined by means of framed weakest precondition functions wp_F via

(10) $$(F \models \{P\}\, c\, \{Q\}\ ) \quad \equiv \quad (F \models P \leqslant wp_F.c.Q)\ ,$$

where, by convention, P and Q are extended to frame F if necessary.

These two postulates in combination with extension rule (5) are sufficient to prove the following generalization of rule (2).

Let $(Z, <, N)$ be a well-founded triple. Consider a family of mutually recursive procedures $(i \in I :: h.i)$ with families $(i :: P.i)$ and $(i :: Q.i)$ of predicates over frames $F.i = D_1.(h.i)$ and a family of state functions $vf.i$ from $[F.i]$ to Z. Write $G.i = B_1.(h.i)$.

(11) **Recursion theorem.** Assume that, for all $m \in Z$ and $i \in I$, we have the implications

(12) $$\begin{aligned}&(\forall\, k \in I, R \in \mathbb{P}.(G.i \setminus D_0.(h.k)) :\ F.k \subseteq G.i\ : \\ &\quad G.i \models \{P.k \ \wedge\ vf.k < m \ \wedge\ m \in N \ \wedge\ R\} \\ &\quad h.k \quad \{Q.k \ \wedge\ R\}\ ) \\ &\Rightarrow \quad (G.i \models \{P.i \ \wedge\ vf.i = m\}\, \mathbf{body}.(h.i)\, \{Q.i\}\ )\ .\end{aligned}$$

Then it holds that $F.i \models \{P.i\}\, h.i\, \{Q.i\}$ for all $i \in I$.

Proof. In view of rule (7), we observe that

$$\begin{aligned}&(\forall\, k :: F.k \models (P.k \ \wedge\ vf.k < m \ \wedge\ m \in N) \leqslant wp_{F.k}.(h.k).(Q.k)) \\ \equiv \quad &\{\ (10)\ \} \\ &(\forall\, k :: F.k \models \{P.k \ \wedge\ vf.k < m \ \wedge\ m \in N\}\, h.k\, \{Q.k\}\ ) \\ \Rightarrow \quad &\{\ (5);\ \text{allow predicates } R \text{ over } G.i \setminus D_0.(h.k)\ \} \\ &(\forall\, i, k, R :\ F.k \subseteq G.i\ : \\ &\quad G.i \models \{P.k \ \wedge\ vf.k < m \ \wedge\ m \in N \ \wedge\ R\}\, h.k\, \{Q.k \ \wedge\ R\}\ ) \\ \Rightarrow \quad &\{\ (12)\ \} \\ &(\forall\, i :: G.i \models \{P.i \ \wedge\ vf.i = m\}\, \mathbf{body}.(h.i)\, \{Q.i\}\ ) \\ \equiv \quad &\{\ (9)\ \} \\ &(\forall\, i :: F.i \models \{P.i \ \wedge\ vf.i = m\}\, h.i\, \{Q.i\}\ ) \\ \equiv \quad &\{\ (10)\ \} \\ &(\forall\, i :: F.i \models (P.i \ \wedge\ vf.i = m) \leqslant wp_{F.i}.(h.i).(Q.i))\ .\end{aligned}$$

By rule (7), this implication implies that $F.i \models P.i \leqslant wp_{F.i}.(h.i).(Q.i)$ for all i. By (10), this concludes the proof. □

Rule (2) is the special case of this recursion theorem with the standard triple $(\mathbb{Z}, <, \mathbb{N})$ where all procedures $h.i$ are equal.

Our aim is to justify all our postulates. For this purpose it now remains to construct functions wp_F such that the framed Hoare triples defined by (10) satisfy rules (5) and (9).

5. Extension, Abstraction and Composition

In order to construct the framed weakest preconditions that can serve in (10), we have to define extension and abstraction of predicate transformers. In view of rule (5.b), we have to distinguish from the outset between the modifiable frame and the accessible frame. We therefore combine these frames in a so-called frame pair.

Definition. A *frame pair* E is a pair of frames $E = (E_0, E_1)$ with $E_0 \subseteq E_1$.

In 5.1, we associate to each frame pair E a boolean lattice $pt.E$ of predicate transformers over E and to each inclusion of one frame pair into another, say E into F, an extension function $\xi \in pt.E \to pt.F$. In 5.2, we construct a left inverse of function ξ, which is called abstraction. In 5.3, we introduce composition of predicate transformers, so that the sets $pt.E$ turn into monoids. Finally, in 5.4, we restrict the attention to monotone predicate transformers. The requirement $E_0 \subseteq E_1$ in the definition of frame pairs is needed for the definitions of extension and composition.

5.1. Frame Pairs and Extensions

A command that uses a frame E_1 but only modifies variables in a frame $E_0 \subseteq E_1$ can be specified by how it transforms postconditions over E_0 to preconditions over E_1. It can therefore be specified by a function from $\mathbb{P}.E_0$ to $\mathbb{P}.E_1$. This leads to the following formalization.

We define the set of predicate transformers over frame pair E as $pt.E = (\mathbb{P}.E_0 \to \mathbb{P}.E_1)$. This set is ordered by the order induced from $\mathbb{P}.E_1$. Since the latter is a complete boolean lattice, $pt.E$ is a complete boolean lattice as well.

The frame pairs are ordered by pairwise inclusion:

$$E \sqsubseteq F \quad \equiv \quad E_0 \subseteq F_0 \quad \wedge \quad E_1 \subseteq F_1 \ .$$

For frame pairs E and F with $E \sqsubseteq F$, we define the *extension* function $\xi^E_F \in pt.E \to pt.F$ by

$$\text{(13)} \qquad \xi^E_F.h.p.x \quad = \quad h.(p \circ (x|F_0; E_0 : \_)).(x|E_1) \ ,$$

where $h \in pt.E$, $p \in \mathbb{P}.F_0$, $x \in [F_1]$; it follows that $(x|F_0; E_0 : \_) \in [E_0] \to [F_0]$ and hence $h.(p \circ (x|F_0; E_0 : \_)) \in \mathbb{P}.E_1$. So, the definition is well typed. Our definition of ξ is equivalent to the more complicated definition of frame extension in [Mah99] 2.2.

This definition of ξ can be justified as follows. State x is regarded as giving the initial values of all variables in frame F_1. Predicate p is regarded as a postcondition only concerned with values for F_0. The values of x for $F_0 \setminus E_0$ do not change; therefore p is composed with the update function $(x|F_0; E_0 : \_)$. Function h yields a precondition in terms of frame E_1; therefore the final argument is the restriction $x|E_1$. A more formal but also operational justification is given in Section 8.

If the two pairs are equal, function ξ is the identity; more precisely, ξ_E^E is the identity of $pt.E$. This follows from (13) and $E_0 = F_0$ and $E_1 = F_1$. Prescription ξ is functorial in the sense that

$$(14) \qquad E \sqsubseteq F \sqsubseteq G \quad \Rightarrow \quad \xi_G^F \circ \xi_F^E = \xi_G^E \ .$$

This is proved by observing that, for $h \in pt.E$, $p \in \mathbb{P}.G_0$, $x \in [G_1]$,

$$\begin{array}{cl} & (\xi_G^F \circ \xi_F^E).h.p.x \\ = & \{ \text{ composition and (13) for } \xi_G^F \} \\ & \xi_F^E.h.(p \circ (x|G_0; F_0 : \_)).(x|F_1) \\ = & \{ \text{ (13) for } \xi_F^E \text{ and restrictions } \} \\ & h.(p \circ (x|G_0; F_0 : \_) \circ (x|F_0; E_0 : \_)).(x|E_1) \\ = & \{ \text{ calculus } \} \\ & h.(p \circ (x|G_0; E_0 : \_)).(x|E_1) \\ = & \{ \text{ (13) for } \xi_G^E \} \\ & \xi_G^E.h.p.x \ . \end{array}$$

In a context with only two frame pairs, E and F with $E \sqsubseteq F$, we usually omit the decorations and write ξ instead of ξ_F^E.

Using the pointwise definition of conjunction and disjunction on predicate transformers, one can easily verify that $\xi \in pt.E \to pt.F$ is universally bijunctive, i.e., both universally conjunctive and universally disjunctive, cf. [DiS90]. It also commutes with negation. The next result is the key to the proof of rule (5.b).

(15) **Lemma.** Let E, F be frame pairs with $E \sqsubseteq F$. Let G be a frame with $G \subseteq F_0$ and $G \cap E_0 = \emptyset$. Let $h \in pt.E$, $q \in \mathbb{P}.F_0$, $r \in \mathbb{P}.G$. Then

$$F_1 \ \models \ \xi.h.q \ \wedge \ (r \circ (\_|G)) \ \leqslant \ \xi.h.(q \ \wedge \ (r \circ (\_|G))) \ .$$

Note that the underline to the left refers to an argument in $[F_1]$, whereas the underline to the right is of type $[F_0]$.

Proof. It suffices to observe that, for every $x \in [F_1]$,

$$\begin{array}{cl} & \xi.h.(q \ \wedge \ (r \circ (\_|G))).x \\ \equiv & \{ \text{ (13) } \} \\ & h.((q \ \wedge \ (r \circ (\_|G))) \circ (x|F_0; E_0 : \_)).(x|E_1) \\ \equiv & \{ \text{ calculus using } G \cap E_0 = \emptyset \} \\ & h.((q \circ (x|F_0; E_0 : \_)) \ \wedge \ r.(x|G)).(x|E_1) \\ \Leftarrow & \{ \text{ use second conjunct } \} \\ & h.(q \circ (x|F_0; E_0 : \_)).(x|E_1) \quad \wedge \quad r.(x|G) \\ \equiv & \{ \text{ (13) } \} \\ & \xi.h.q.x \quad \wedge \quad r.(x|G) \ . \end{array}$$

□

5.2. Abstraction and Order

For frame pairs E and F, with $E \sqsubseteq F$, we also define the *abstraction* function $\rho_E^F \in pt.F \to pt.E$ given by

$$(16) \qquad \rho_E^F.h.p.x \quad = \quad (\forall\, y \in [F_1] :: h.(p \circ (\_|E_0)).(y; E_1 : x)) \ ,$$

where $h \in pt.F$, $p \in \mathbb{P}.E_0$, $x \in [E_1]$. Here, since function h acts on postconditions over F_0, the argument p is extended to F_0 in the usual way. The result is regarded as a precondition over F_1, but since x only specifies values over E_1, prestate x is extended to F_1 by demonic nondeterminacy. A more formal justification of definition (16) is given in Section 8.

Function ρ_E^F is the left inverse of ξ_F^E. In fact, for $h \in pt.E$, $p \in \mathbb{P}.E_0$, $x \in [E_1]$,

$$\begin{array}{ll} & (\rho_E^F \circ \xi_F^E).h.p.x \\ = & \{ (16) \} \\ & (\forall\, y \in [F_1] :: \xi_F^E.h.(p \circ (\_|E_0)).(y; E_1 : x)) \\ = & \{ (13) \} \\ & (\forall\, y \in [F_1] :: h.(p \circ (\_|E_0) \circ ((y; E_1 : x)|F_0; E_0 : \_)).((y; E_1 : x)|E_1)) \\ = & \{ \text{calculus} \} \\ & (\forall\, y \in [F_1] :: h.p.x) \\ = & \{ [F_1] \text{ is nonempty} \} \\ & h.p.x \ . \end{array}$$

This proves that $\rho_E^F \circ \xi_F^E$ is the identity of $pt.E$. Therefore ξ_F^E is injective and ρ_E^F is surjective.

More generally, for frame pairs F, G, H with $F \sqsubseteq H$ and $G \sqsubseteq H$, one can define the intersection $E = (E_0, E_1)$ with $E_i = F_i \cap G_i$ for $i = 0, 1$. In this situation, we have

$$(17) \qquad \rho_G^H \circ \xi_H^F \ = \ \xi_G^E \circ \rho_E^F \ \text{ in } \ pt.F \to pt.G.$$

This is proved by observing, for $h \in pt.F$, $p \in \mathbb{P}.G_0$, $x \in [G_1]$, that

$$\begin{array}{ll} & (\rho_G^H \circ \xi_H^F).h.p.x \\ = & \{ (16) \} \\ & (\forall\, y \in [H_1] :: \xi_H^F.h.(p \circ (\_|G_0)).(y; G_1 : x)) \\ = & \{ (13) \} \\ & (\forall\, y \in [H_1] :: h.(p \circ (\_|G_0) \circ ((y; G_1 : x)|H_0; F_0 : \_)).((y; G_1 : x)|F_1)) \\ = & \{ \text{calculus and definition of } E_1 \} \\ & (\forall\, y \in [H_1] :: h.(p \circ (((y; G_1 : x)|H_0; F_0 : \_)|G_0)).(y|F_1; E_1 : (x|E_1))) \\ = & \{ \text{calculus and definition of } E_0 \} \\ & (\forall\, y \in [H_1] :: h.(p \circ (x|G_0; E_0 : (\_|E_0))).(y|F_1; E_1 : (x|E_1))) \\ = & \{ \text{calculus and replace } y \text{ by } y|F_1 \} \\ & (\forall\, y \in [F_1] :: h.(p \circ (x|G_0; E_0 : \_) \circ (\_|E_0)).(y; E_1 : (x|E_1))) \\ = & \{ (16) \} \\ & \rho_E^F.h.(p \circ (x|G_0; E_0 : \_)).(x|E_1) \\ = & \{ (13) \} \\ & (\xi_G^E \circ \rho_E^F).h.p.x \ . \end{array}$$

Just as with ξ, we often write ρ instead of ρ_E^F in a context with only two frame pairs, E and F with $E \sqsubseteq F$.

Abstraction preserves the ordering of predicates over the smaller frame in the sense that, for frame pairs $E \sqsubseteq F$, $h \in pt.F$, $p \in \mathbb{P}.E_1$, $q \in \mathbb{P}.E_0$,

$$(18) \qquad (E_1 \ \models \ p \leqslant \rho.h.q) \ \ \equiv \ \ (F_1 \ \models \ p \circ (\_|E_1) \leqslant h.(q \circ (\_|E_0))) \ .$$

This is proved by

$$
\begin{array}{ll}
 & E_1 \models p \leqslant \rho.h.q \\
\equiv & \{ (0) \text{ and } (16) \} \\
 & (\forall\, x \in [E_1] :: p.x \Rightarrow (\forall\, y \in [F_1] :: h.(q \circ (\_|E_0)).(y; E_1 : x))) \\
\equiv & \{ \text{calculus} \} \\
 & (\forall\, x \in [E_1], y \in [F_1] :: p.x \Rightarrow h.(q \circ (\_|E_0)).(y; E_1 : x)) \\
\equiv & \{ \text{take } z = (y; E_1 : x), \text{ then } x = (z|E_1) \} \\
 & (\forall\, z \in [F_1] :: p.(z|E_1) \Rightarrow h.(q \circ (\_|E_0)).z) \\
\equiv & \{ (0) \} \\
 & F_1 \models p \circ (\_|E_1) \leqslant h.(q \circ (\_|E_0)) \;.
\end{array}
$$

By application of (18) to $h := \xi.k$ for $k \in pt.E$, using that ρ is the left inverse of ξ, we also obtain

(19) $$(E_1 \models p \leqslant k.q) \quad \equiv \quad (F_1 \models p \circ (\_|E_1) \leqslant \xi.k.(q \circ (\_|E_0))) \;.$$

Remark. It can be shown that, in general, ρ is not the upper adjoint of ξ. In particular, there exist frames $E \sqsubseteq F$ with $h \in pt.F$ and $q \in \mathbb{P}.F_0$ such that $\neg\,(F_1 \models \xi.(\rho.h).q \leqslant h.q)$. Take $E_0 = E_1 = \emptyset$ and $F_0 = F_1 = \{\mathrm{v}\}$ for a variable v of type integer. Let q be the predicate $\mathrm{v} = 0$ and take $h = wp.(\mathrm{v} := \mathrm{v} + 1)$. It turns out that $\xi.(\rho.h).q = q$, whereas $\neg\,(F_1 \models q \leqslant h.q)$. □

5.3. Monoid Structures

Predicate transformers on a single state space can be composed. Thus, if $E_0 = E_1$ then $pt.E$ equals $(\mathbb{P}.E_1 \to \mathbb{P}.E_1)$ and, hence, is a monoid under function composition. For every frame pair E, we now provide $pt.E$ with a composition operator "$\bullet$" given by

(20) $$(h_0 \bullet h_1).p.x \quad = \quad h_0.((h_1.p) \circ (x; E_0 : \_)).x \;,$$

for all h_0, $h_1 \in pt.E$, $p \in \mathbb{P}.E_0$ and $x \in [E_1]$. Operator "$\bullet$" is a generalization of "$\circ$" in the sense that $h_0 \bullet h_1 = h_0 \circ h_1$ if $E_0 = E_1$. In Section 8 we sketch a relational justification of definition (20).

Functions ξ distribute over "$\bullet$". In fact, for frame pairs $E \sqsubseteq F$ and h_0, $h_1 \in pt.E$, we have

(21) $$\xi.(h_0 \bullet h_1) = \xi.h_0 \bullet \xi.h_1 \;.$$

This is proved as follows. Let $p \in \mathbb{P}.E_0$ and $x \in [F_1]$. On the one hand, we have

$$
\begin{array}{ll}
 & \xi.(h_0 \bullet h_1).p.x \\
= & \{ (13) \} \\
 & (h_0 \bullet h_1).(p \circ (x|F_0; E_0 : -)).(x|E_1) \\
= & \{ (20) \text{ in } pt.E \} \\
 & h_0.q.(x|E_1) \;, \quad \text{where} \\
 & q \quad = \quad (h_1.(p \circ (x|F_0; E_0 : -))) \circ (x|E_1; E_0 : -) \;.
\end{array}
$$

On the other hand, we observe that

$$
\begin{array}{ll}
 & (\xi.h_0 \bullet \xi.h_1).p.x \\
= & \{ (20) \text{ in } pt.F \} \\
 & \xi.h_0.((\xi.h_1.p) \circ (x; F_0 : \_)).x \\
= & \{ (13) \text{ for } \xi.h_0 \}
\end{array}
$$

$$h_0.r.(x|E_1)\ , \quad \text{where}$$
$$r \quad = \quad (\xi.h_1.p) \circ (x; F_0 : -) \circ (x|F_0; E_0 : -)\ .$$

It remains to prove that $q = r$ in $\mathbb{P}.E_0$. This is done by observing that, for every $y \in [E_0]$,

$$\begin{aligned}
& r.y \\
= \quad & \{ \text{ definition } r \text{ and calculus } \} \\
& \xi.h_1.p.(x; E_0 : y) \\
= \quad & \{ \ (13) \ \} \\
& h_1.(p \circ ((x; E_0 : y)|F_0; E_0 : _)).((x; E_0 : y)|E_1) \\
= \quad & \{ \text{ calculus } \} \\
& h_1.(p \circ (x|F_0; E_0 : _)).(x|E_1; E_0 : y) \\
= \quad & \{ \text{ definition of } q \ \} \\
& q.y\ .
\end{aligned}$$

This concludes the proof of (21).

For a frame pair B with $B_0 = B_1$, the identity function is the neutral element of the monoid $pt.B$. Let us denote it by id_B. For $B \sqsubseteq E$, $p \in \mathbb{P}.E_0$ and $x \in [E_1]$, we have

(22)
$$\begin{aligned}
& \xi^B_E.id_B.p.x \\
= \quad & id_B.(p \circ (x|E_0; B_0 : _)).(x|B_1) \\
= \quad & p.(x|E_0; B_0 : (x|B_1)) \\
= \quad & p.(x|E_0)\ .
\end{aligned}$$

This proves $\xi^B_E.id_B$ is the function $jd_E \in pt.E$ given by $jd.p = p \circ (\_|E_0)$. In particular, if $E_0 = E_1$, then $\xi^B_E.id_B = id_E$.

(23) **Theorem.** For every frame pair E, the set $pt.E$ is a monoid with operator "$\bullet$" and neutral element jd_E. If $E \sqsubseteq F$, then ξ^E_F is a morphism of monoids from $pt.E$ to $pt.F$.

Proof. Let B and G be the frame pairs defined by $B_0 = B_1 = E_0$ and $G_0 = G_1 = E_1$. Then $B \sqsubseteq E \sqsubseteq G$. Calculation (22) implies that $jd_E = \xi^B_E.id_B$. Using (14) and (22), we therefore have $\xi^E_G.jd_E = id_G$. We now prove that jd_E is the neutral element of $pt.E$ by observing that, for $h \in pt.E$,

$$\begin{aligned}
& jd_E \bullet h = h \quad \wedge \quad h \bullet jd_E = h \\
\equiv \quad & \{ \ \xi^E_G \text{ is injective } \} \\
& \xi^E_G.(jd_E \bullet h) = \xi^E_G.h \quad \wedge \quad \xi^E_G.(h \bullet jd_E) = \xi^E_G.h \\
\equiv \quad & \{ \ (21) \text{ and } \xi^E_G.jd_E = id_G \ \} \\
& id_G \bullet \xi^E_G.h = \xi^E_G.h \quad \wedge \quad \xi^E_G.h \bullet id_G = \xi^E_G.h \\
\equiv \quad & \{ \ id_G \text{ is neutral for } \circ \text{ and } \circ = \bullet \ \} \\
& true\ .
\end{aligned}$$

Since $pt.G$ is a monoid with operation $\bullet = \circ$, a similar argument can be used to prove that $\bullet$ is associative on $pt.E$. In (21), we have that ξ distributes over $\bullet$. Preservation of the neutral elements follows from (22). □

5.4. Monotone Predicate Transformers

A predicate transformer $h \in pt.E$ is called monotone iff, for every pair $p, q \in \mathbb{P}.E_0$, we have

$$E_0 \models p \leqslant q \quad \Rightarrow \quad E_1 \models h.p \leqslant h.q\ .$$

The set of the monotone predicate transformers in $pt.E$ is denoted by $mt.E$. The greatest lower bound (infimum) and least upper bound (supremum) of a family of monotone predicate transformers are both monotone. Therefore $mt.E$ is a complete lattice in its own right and the injection from $mt.E$ into $pt.E$ is universally bijunctive. Since the negation of a monotone predicate transformer is usually not monotone, $mt.E$ is not a boolean lattice.

The functions $\xi_F^E \in pt.E \to pt.F$ and $\rho_E^F \in pt.F \to pt.E$ preserve monotony of predicate transformers and therefore induce functions $\xi_F^E \in mt.E \to mt.F$ and $\rho_E^F \in mt.F \to mt.E$. If h_0 and h_1 are monotone then $h_0 \bullet h_1$ is also monotone. The neutral element jd_E is also monotone. Therefore $mt.E$ is a submonoid of $pt.E$. It is easy to see that, for functions h, k, h', $k' \in mt.E$, we have

(24) $$h \leqslant h' \quad \wedge \quad k \leqslant k' \quad \Rightarrow \quad h \bullet k \;\leqslant\; h' \bullet k' \ .$$

This property does not hold for $pt.E$; monotony of h or h' is needed for the proof.

6. Weakest Preconditions Defined

In 6.1, we introduce the programming language to define recursive procedures, together with its formal semantics. In 6.2, guards and parametrized commands are introduced to model conditional statements, assignments, and procedures with value parameters. Subsection 6.3 contains two lemmas on weakest preconditions needed to justify the postulates on Hoare triples.

6.1. Syntax and Semantics

We use the following abstract syntax for the definition of recursive procedures. It is a variation of the syntax in [Hes94].

Let A be a set of symbols, to be called *elementary commands*. We assume that a frame pair $D.a = (D_0.a, D_1.a)$ is specified for each command $a \in A$. As before, $D_0.a$ is the modifiable frame of a and $D_1.a$ is its accessible frame.

In order to allow infinite choice operators while remaining in classical set theory, we fix a set of sets Unv, called the universe. Now, for every frame pair E, we define the set of commands $Cmd.E$ inductively by the clauses

- if $a \in A$ has $D.a \sqsubseteq E$ then $a \in Cmd.E$,
- if c, $d \in Cmd.E$ then $c\,;d \in Cmd.E$,
- if $I \in Unv$ and $(i \in I :: c.i)$ is a family in $Cmd.E$ then $(\,[\!]\; i \in I :: c.i) \in Cmd.E$ and $(\diamond\, i \in I :: c.i) \in Cmd.E$.

Here, the symbols ;, $[\!]$, and $\diamond$ are operators for sequential composition, demonic choice and angelic choice, respectively. Note that $Cmd.E \subseteq Cmd.F$ for frame pairs E and F with $E \sqsubseteq F$.

We assume that the set A is the disjoint union of two sets S and H, which may be infinite. The elements of S are called simple commands. The elements of H are called procedure names. As before, every procedure $h \in H$ is equipped with a frame pair $B.h = (B_0.h, B_1.h)$ such that $D.h \sqsubseteq B.h$ and $B_0.h \cap D_1.h = D_0.h$, and with a body

$$\mathbf{body}.h \in Cmd.(B.h)\ .$$

Remark. Of course, we retain the condition that, if procedure h is a sibling or a descendant of a sibling of procedure k and k is called in the body of h, then $D.k \sqsubseteq D.h$. In fact, the call of k must not access local variables of the body of h. □

We now turn to the semantic side. We assume that every simple command $s \in S$ has given semantics $ws.s \in mt.(D.s)$. In other words, we assume that, for simple command s and postcondition $q \in \mathbb{P}.(D_0.s)$, the weakest precondition is given as $ws.s.q \in \mathbb{P}.(D_1.s)$.

The aim is to define for every procedure $h \in H$ the weakest precondition function $wp.h \in mt.(D.h)$. Since the procedures are possibly nested and mutually recursive, we have to define $wp.h$ for all procedures h at once. So we have to define wp as an element of the cartesian product

$$mtH \quad = \quad (\Pi\ h \in H :: mt.(D.h)) \ .$$

As a product of complete lattices, the set mtH is itself a complete lattice.

In order to define $wp \in mtH$, we start to extend an arbitrary $w \in mtH$ to a function on all commands. First, for an elementary command $a \in A$, we define $w^+.a$ by $w^+.a = w.a$ if $a \in H$ and $w^+.a = ws.a$ if $a \in S$. We now define $w_E \in Cmd.E \to mt.E$ inductively by the clauses

- if $a \in A$ has $D.a \sqsubseteq E$ then $w_E.a = \xi_E^{D.a}.(w^+.a)$.
- if $c, d \in Cmd.E$ then $w_E.(c\,; d) = (w_E.c) \bullet (w_E.d)$.
- if $I \in Unv$ and $(i \in I :: c.i)$ is a family in $Cmd.E$ then $w_E.(\,[\!]\ i :: c.i) = (\inf\ i :: w_E.(c.i))$ and $w_E.(\diamondsuit\, i :: c.i) = (\sup\ i :: w_E.(c.i))$.

We use the notation inf for the infimum, the greatest lower bound in the lattice, and sup for the supremum, the least upper bound. Since the ordering is induced by universal implication, it follows that, e.g., $w_E.(\,[\!]\ i :: c.i).q = (\forall\ i :: w_E.(c.i).q)$. If $[\!]$ and $\diamondsuit$ act on a family of two elements c and d, we use the infix notation $c\,[\!]\,d$ and $c \diamondsuit d$, which have a lower binding power than sequential composition ";".

The function $w \mapsto w_E$ from mtH to $Cmd.E \to mt.E$ is monotone, since ξ is monotone, composition of monotone functions is monotone (24), and forming infima and suprema is monotone.

(25) **Lemma.** Let E and F be frame pairs with $E \sqsubseteq F$ and let $w \in mtH$. For every command $c \in Cmd.E$, we have $c \in Cmd.F$ and $w_F.c = \xi_F^E.(w_E.c)$.

Proof. This is proved by straightforward induction over the structure of Cmd. We use (14) for elementary commands, (21) for sequential composition, and the universal bijunctivity of ξ for the two choice operators. □

We now use **body**.$h \in Cmd.(B.h)$ to define function rec by

$$rec.w.h \quad = \quad \rho_{D.h}^{B.h}.(w_{B.h}.(\mathbf{body}.h)) \ \in \ mt.(D.h) \ .$$

This defines rec as an endofunction of the complete lattice mtH. The abstraction function ρ is monotone. Since $w \mapsto w_E$ is also monotone, we have that endofunction rec is monotone. Therefore, by the Theorem of Knaster-Tarski, function rec has a least fixpoint.

Definition. Function $wp \in mtH$ is defined as the least fixpoint of rec. This function induces a function $wp_E \in Cmd.E \to mt.E$ for every frame pair E. Finally, the semantics of a command $c \in Cmd.E$ is defined as $wp_E.c \in mt.E$.

It follows that an elementary command a with $D.a \sqsubseteq E$ has semantics $wp_E.a = \xi_E^{D.a}.(wp^+.a)$. In particular, a simple command s has $wp_E.a = \xi_E^{D.a}.(ws.a)$ and a procedure name h has $wp_E.h = \xi_E^{D.a}.(wp.h)$.

Remark. The main difference with the construction in [Hes94] is the appearance of ρ in the definition of function *rec*: the meaning of a procedure is the abstraction of the meaning of its body. Function ξ and operator "$\bullet$" are also new, but they are rather innocent, since ξ is injective and is able to transform "$\bullet$" into "$\circ$". $\square$

6.2. Conditional Statements and Parameters

Conditional statements are modelled by means of so-called guards. Guards are unimplementable simple statements, regarded as the quarks of programming [War93]. They are attributed to [Kar59]. We use the notation of [Hes92].

If F is a frame, we associate to a predicate $b \in \mathbb{P}.F$ the guard $?b \in S$ given by $D_0.(?b) = \emptyset$, $D_1.(?b) = F$, and $ws.(?b).q = (b \Rightarrow q)$ for $q \in \mathbb{P}.\emptyset = \mathbb{B}$. One can verify that $wp_E.(?b).q = (b \Rightarrow q)$ for every frame pair E with $D.(?b) \sqsubseteq E$ and $q \in \mathbb{P}.E_0$, where the predicates b and q are extended to E_1. Guards are introduced primarily to model conditional statements. In fact, if $c, d \in Cmd.E$ and $b \in \mathbb{P}.E_1$, we define

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

It is easy to verify that this command has the weakest preconditions expected.

Guards are also needed for the application of parametrized commands. A parametrized command is a family of commands $(u \in U :: c.u)$, say in $Cmd.E$. In order to apply such a command, we need an argument that may depend on the state in which the command is called. So, in general, a call of c uses a function $f \in [E_1] \to U$ to supply the argument. We define the call with argument supplied by f by means of

$$c \otimes f \quad = \quad (\,[\!]\, u \in U :: ?(f = u) \,;\, c.u)\ .$$

In fact, we have

$$wp_E.(c \otimes f).q \quad = \quad (\forall\, u \in U :: (f = u) \Rightarrow wp_E.(c.u).q)\ .$$

Using the one-point rule, it follows that, for $x \in [E_1]$, we have the expected equality

$$wp_E.(c \otimes f).q.x \quad = \quad wp_E.(c.(f.x)).q.x\ .$$

The most important parametrized command is the (multiple) assignment. The modifiable frame of a multiple assignment consists of the variables to be modified. An actual multiple assignment is parametrized by the tuple of expressions. Therefore, as a parametrized command, a multiple assignment is a family indexed by the tuples of possible values. This is formalized as follows.

For every frame F, the multiple assignment $ass.F$ is defined as the family $(u \in [F] :: ass.F.u)$ with $D.(ass.F.u) = (F, F)$ and $ws.(ass.F.u).q.x = q.u$ for all u, $x \in [F]$. For a frame pair E with $(F, F) \sqsubseteq E$ and a function $f \in [E_1] \to [F]$, the assignment $F := f$ is defined as $ass.F \otimes f$ in $Cmd.E$. One can verify that, for $q \in \mathbb{P}.E_0$ and $x \in [E_1]$, it satisfies

$$\begin{array}{ll} & wp_E.(F := f).q.x \\ = & \{\ \text{definition}\ \} \\ & wp_E.(ass.F \otimes f).q.x \\ = & \{\ \text{see above}\ \} \\ & wp_E.(ass.F.(f.x)).q.x \\ = & \{\ wp \text{ of simple command}\ \} \\ & \xi_E^{(F,F)}.(ws.(ass.F.(f.x))).q.x \end{array}$$

$$
\begin{aligned}
&= \{ (13) \} \\
&\quad ws.(ass.F.(f.x)).(q \circ (x|E_0; F : \_)).(x|F) \\
&= \{ \text{definition of } ws \text{ of } ass \} \\
&\quad (q \circ (x|E_0; F : \_)).(f.x) \\
&= \{ \text{calculus} \} \\
&\quad q.(x|E_0; F : f.x) \ ,
\end{aligned}
$$

as should be expected.

A procedure with value parameters is treated as a parametrized procedure. If the procedure is recursive, it is modelled as a family of mutually recursive procedures. A systematic treatment of reference parameters or of procedure parameters falls outside the scope of this paper.

6.3. Conjunction and Body Rule

The next result is the reason for introducing frame pairs.

(26) **Lemma.** Let E, F be frame pairs with $E \sqsubseteq F$. Let G be a frame with $G \subseteq F_0$ and $G \cap E_0 = \emptyset$. For $c \in Cmd.E$, $q \in \mathbb{P}.F_0$ and $r \in \mathbb{P}.G$, we have

$$F_1 \ \models\ wp_F.c.q \ \wedge\ (r \circ (\_|G)) \ \leqslant\ wp_F.c.(q \ \wedge\ (r \circ (\_|G))) \ .$$

Proof. This follows immediately from rule (25) with $w := wp$ and rule (15) with $h := wp_E$. □

In order to prove body rule (9), we formulate and prove an analogue for wp.

(27) **Lemma.** Let h be a procedure. Let F and G be frame pairs with $D.h \sqsubseteq F \sqsubseteq G$ and $B.h \sqsubseteq G$ and $F_1 \cap B_1.h = D_1.h$. Then we have

$$wp_F.h = \rho_F^G.(wp_G.(\mathbf{body}.h)) \ .$$

Proof. We first note that the assumptions yield the following sequence of inclusions

$$D_0.h \subseteq F_0 \cap B_0.h \subseteq (F_1 \cap B_1.h) \cap B_0.h = D_1.h \cap B_0.h = D_0.h \ .$$

This implies that $D_0.h = F_0 \cap B_0.h$. We conclude by observing

$$
\begin{aligned}
&\quad wp_F.h \\
&= \{ \text{definitions of } wp \text{ and } rec \} \\
&\quad \xi_F^{D.h}.(\rho_{D.h}^{B.h}.(wp_{B.h}.(\mathbf{body}.h))) \\
&= \{ (17),\ D_0.h = F_0 \cap B_0.h \text{ and } D_1.h = F_1 \cap B_1.h \} \\
&\quad \rho_F^G.(\xi_G^{B.h}.(wp_{B.h}.(\mathbf{body}.h))) \\
&= \{ (25) \text{ with } w := wp \} \\
&\quad \rho_F^G.(wp_G.(\mathbf{body}.h)) \ . \ \square
\end{aligned}
$$

7. Back to Hoare Triples

In this section, we fulfil the remaining obligations: we define framed Hoare triples in such a way that rules (5), (9) and (10) hold.

In framed Hoare triples, we want to interpret the precondition and the postcondition with respect to the same frame. For each frame F, we therefore define the frame pair $d.F = (F, F)$. Now framed Hoare triples are defined by postulating (10) where, by abuse of notation, we read wp_F as standing for $wp_{d.F}$.

In order to prove body rule (9), we consider a procedure $h \in H$ with its frame pairs $D.h$ and $B.h$. We write $F = D_1.h$ and $G = B_1.h$. For predicates $P, Q \in \mathbb{P}.F$, we have

$$\begin{array}{ll} & F \models \{P\} \quad h \quad \{Q\} \\ \equiv & \{ (10) \} \\ & F \models P \leqslant wp_{d.F}.h.Q \\ \equiv & \{ (27) \text{ with } F := d.F \text{ and } G := d.G \} \\ & F \models P \leqslant \rho^{d.G}_{d.F}.(wp_{d.G}.(\mathbf{body}.h)).Q \\ \equiv & \{ (18) \text{ with } E := d.F \text{ and } F := d.G\} \\ & G \models P \circ (\_|F) \leqslant wp_{d.G}.(\mathbf{body}.h).(Q \circ (\_|F)) \\ \equiv & \{ (10) \text{ where predicates are extended to } G \} \\ & G \models \{P\}\, \mathbf{body}.h\, \{Q\} \quad . \end{array}$$

This proves body rule (9).

We can now prove rule (5.a) and strengthen it to an equivalence. For frames F and G with $F \subseteq G$, a command $c \in Cmd.(d.F)$, and predicates P and Q, we have

$$(28) \qquad (F \models \{P\}\, c\, \{Q\}\,) \quad \equiv \quad (G \models \{P\}\, c\, \{Q\}\,) \; .$$

This is proved by

$$\begin{array}{ll} & F \models \{P\}\, c\, \{Q\} \\ \equiv & \{ (10) \} \\ & F \models P \leqslant wp_{d.F}.c.Q \\ \equiv & \{ (19) \} \\ & G \models P \circ (\_|F) \leqslant \xi^{d.F}_{d.G}.(wp_{d.F}.c).(Q \circ (\_|F)) \\ \equiv & \{ (25) \} \\ & G \models P \circ (\_|F) \leqslant wp_{d.G}.c.(Q \circ (\_|F)) \\ \equiv & \{ (10) \} \\ & G \models \{P\}\, c\, \{Q\} \quad . \end{array}$$

Finally, rule (5.b) follows from

Disjointness. Let $c \in Cmd.E$ for a frame pair E. Let P, Q, R be predicates over E_1. Assume that $E_1 \models \{P\}\; c\; \{Q\}$, and that $R = (r \circ (\_|G))$ for some $r \in \mathbb{P}.G$ where $G = E_1 \setminus E_0$. Then we have

$$E_1 \models \{P \wedge R\}\, c\, \{Q \wedge R\} \quad .$$

This is proved by taking $F = d.E_1$ and observing

$$\begin{array}{ll} & E_1 \models \{P \wedge R\}\, c\, \{Q \wedge R\} \\ \equiv & \{ (10) \text{ and definition of } F \} \\ & E_1 \models P \wedge R \leqslant wp_F.c.(Q \wedge R) \\ \Leftarrow & \{ (26) \text{ and transitivity of } \leqslant \} \\ & E_1 \models P \wedge R \leqslant wp_F.c.Q \;\wedge\; R \\ \Leftarrow & \{ \text{calculus and } (10) \} \\ & E_1 \models \{P\}\, c\, \{Q\} \quad . \end{array}$$

8. Relational Interpretation

We now give relational justifications for the definitions of extension (13), abstraction (16), and composition (20) for predicate transformers. For this purpose we use relational semantics of programs as formalized in [Hes92], chapter 6. In this context, function *wp* is accompanied by the weakest liberal precondition function *wlp* even though *wlp* does not combine usefully with angelic choice, cf. [Hes94].

More directly than by *wp*, a command can be specified by the relation between its initial state and its final state. Nontermination is formalized by the symbol ∞, which is not in $[F]$ for any frame F. For a frame pair E, a *relation* over E is defined to be a function $R \in ([E_0] \cup \{\infty\} \to \mathbb{P}.E_1)$. For a relation R over E we interpret $R.x'.x$ to mean that an execution starting in x may terminate in state x' if $x' \neq \infty$, and that it may execute forever (nontermination) if $x' = \infty$. Note we use primed variables for the resulting states. More precisely, $x \in [E_1]$ specifies the values of all accessible variables in the inital state whereas $x' \in [E_0]$ only specifies the modifiable variables in the final state; the other accessible variables are unchanged.

We write $Rel.E = ([E_0] \cup \{\infty\} \to \mathbb{P}.E_1)$ for the set of relations over E. For $R \in Rel.E$, we define the weakest precondition functions $wlp.R$ and $wp.R$ in $mt.E$ by

$$
\begin{array}{lrcl}
(29) & wlp.R.p.x & = & (\forall\, x' \in [E_0] : R.x'.x : p.x') \\
 & wp.R.p.x & = & \neg R.\infty.x \quad \wedge \quad wlp.R.p.x\ .
\end{array}
$$

We now introduce the relational versions of extension and abstraction of commands. So, let F be a frame pair with $E \sqsubseteq F$. We introduce an extension function $\varphi \in Rel.E \to Rel.F$ that extends the relation on E by the identity on the complement of E_0, since the command corresponding to the relation should only modify the E_0-component. In this way we arrive at definition

$$
\begin{array}{lrcl}
(30) & \varphi.R.y'.y & = & R.(y'|E_0).(y|E_1) \quad \wedge \quad (y'|F_0 \setminus E_0) = (y|F_0 \setminus E_0)\ , \\
 & \varphi.R.\infty.y & = & R.\infty.(y|E_1)\ .
\end{array}
$$

Note that this is indeed the way the relational semantics of a command is extended when the state space is enlarged by adding fresh variables.

Extension function ξ of (13) corresponds to the relational extension function φ because of the equalities $wlp \circ \varphi = \xi \circ wlp$ and $wp \circ \varphi = \xi \circ wp$. The first equality is proved by observing that, for all $R \in Rel.E$, $q \in \mathbb{P}.F_0$, $y \in [F_1]$,

$$
\begin{array}{cl}
 & wlp.(\varphi.R).q.y \\
= & \{\ (29) \text{ over } F\ \} \\
 & (\forall\, y' \in [F_0] : \varphi.R.y'.y : q.y') \\
= & \{\ (30)\ \} \\
 & (\forall\, y' \in [F_0] : R.(y'|E_0).(y|E_1) \ \wedge\ (y'|F_0 \setminus E_0) = (y|F_0 \setminus E_0) : q.y') \\
= & \{\ \text{take } x' = y'|E_0 \text{ then } y' = (y|F_0; E_0 : x')\ \} \\
 & (\forall\, x' \in [E_0] : R.x'.(y|E_1) : q.(y|F_0; E_0 : x')) \\
= & \{\ (29)\ \} \\
 & wlp.R.(q \circ (y|F_0; E_0 : \_)).(y|E_1) \\
= & \{\ (13)\ \} \\
 & \xi.(wlp.R).q.y\ .
\end{array}
$$

The proof of the second equality uses the same ingredients and may be left to the reader.

The abstraction function $\psi \in Rel.F \to Rel.E$ is only concerned with the effect on the E-component and starts with an arbitrary value for the complement. For

$S \in Rel.F$, relation $\psi.S \in Rel.E$ is defined by

$$\begin{aligned} \psi.S.x'.x &= (\exists\, y \in [F_1], y' \in [F_0] :: S.(y'; E_0 : x').(y; E_1 : x)) \ , \\ \psi.S.\infty.x &= (\exists\, y \in [F_1] :: S.\infty.(y; E_1 : x)) \ . \end{aligned}$$

Note that the relational semantics of a procedure is indeed the abstraction to the external frame of the meaning of its body (possible initialization of local variables must be done in the body). Abstraction ρ of (16) corresponds to the relational abstraction ψ because of the equalities $wlp \circ \psi = \rho \circ wlp$ and $wp \circ \psi = \rho \circ wp$, the proofs of which are left to the reader.

The set $Rel.E$ is supplied with an operator "$\star$" for sequential composition, defined by

$$\begin{aligned} (R \star S).y.x &= (\exists\, z \in [E_0] : R.z.x : S.y.(x; E_0 : z)) \ , \\ (R \star S).\infty.x &= R.\infty.x \ \vee \ (\exists\, z \in [E_0] : R.z.x : S.\infty.(x; E_0 : z)) \ , \end{aligned}$$

for R, $S \in Rel.E$, $y \in [E_0]$, $x \in [E_1]$. Here state x serves as an initial state for an application of R followed by S; state $z \in [E_0]$ holds intermediate values of the modifiable variables and $(x; E_0 : z)$ holds intermediate values of the accessible variables. It is a straightforward though tedious exercise to prove that

$$\begin{aligned} wlp.(R \star S) &= wlp.R \bullet wlp.S \ , \\ wp.(R \star S) &= wp.R \bullet wp.S \ . \end{aligned}$$

This justifies definition (20) of "$\bullet$", or rather its application in the definition of w_E in Section 6.1.

We do not want to restrict the predicate transformers allowed to those that correspond to relations, but it is only for these predicate transformers that we have an operational intuition. We therefore regard the above equalities as sufficient justifications for the definitions of extension, abstraction, and composition.

9. Concluding Remarks

We have shown how local variables can be incorporated in the weakest precondition semantics of recursive procedures. No stacking is required. More specifically, we have shown the soundness of Recursion Theorem (11) and the consequent Correctness Rule (2) with respect to a weakest precondition semantics of recursive procedures with local variables. For this purpose, we have generalized predicate transformers on a single state space to predicate transformers for frame pairs. New aspects in our approach are the introduction of frame pairs and the concentration on the triple of special functions: the extension function ξ, the abstraction function ρ, and the composition operator $\bullet$.

It is possible to replace the ordering $\sqsubseteq$ between frame pairs E and F by injective renaming functions $E_1 \to F_1$ that map E_0 into F_0 and preserve types. This would give a handle on procedures with reference parameters and may also be useful in the treatment of modules.

The introduction of frames with the corresponding operations of extension, composition and abstraction makes predicate transformers a more flexible tool to specify modifications in various contexts. This may be a starting point for formal verification of object-oriented programs. Indeed, objects are frames created dynamically, and extension function ξ models inheritance. A systematic application of the above results to object orientation is a matter of future research.

Acknowledgements

Criticisms and suggestions by two anonymous referees have definitely improved the paper.

References

[AbL98] Abadi, M. and Leino, K.R.M.: *A logic of object-oriented programs*. SRC Research Report 161, 1998.

[BaB98] Back, R.J.R. and Butler, M.: Fusion and simultaneous execution in the refinement calculus. *Acta Informatica* 35 (1998) 921–949.

[BvW90] Back, R.J.R. and von Wright, J.: Duality in specification languages: a lattice–theoretical approach. *Acta Informatica* 27 (1990) 583–625.

[BvW98] Back, R.J.R. and von Wright, J.: *Refinement Calculus*. Graduate Texts in Computer Science, Springer V. 1998.

[CKS81] Chandra, A., Kozen, D. and Stockmeyer, L.: Alternation. *Journal ACM* 28 (1981) 114–133.

[Dij76] Dijkstra, E.W.: *A Discipline of Programming*. Prentice–Hall 1976.

[DiG86] Dijkstra, E.W. and van Gasteren, A.J.M.: A simple fixpoint argument without the restriction to continuity. *Acta Informatica* 23 (1986) 1–7.

[DiS90] Dijkstra, E.W. and Scholten, C.S.: *Predicate Calculus and Program Semantics*. Springer V. 1990.

[Hes90] Hesselink, W.H.: Modalities of nondeterminacy. In: W.H.J. Feijen et al. (eds.): *Beauty is our business, a birthday salute to Edsger W. Dijkstra*. Springer V. 1990, pp. 182–192.

[Hes92] Hesselink, W.H.: *Programs, Recursion and Unbounded Choice, predicate transformation semantics and transformation rules*. Cambridge University Press, 1992.

[Hes93] Hesselink, W.H.: Proof rules for recursive procedures. *Formal Aspects of Computing* 5 (1993) 554–570.

[Hes94] Hesselink, W.H.: Nondeterminacy and recursion via stacks and games. *Theoretical Computer Science* 124 (1994) 273–295.

[Kar59] Karp, R.M.: *Some applications of logical syntax to digital computer programming*. Thesis, Harvard University, 1959.

[Mah99] Mahony, B.P.: The least conjunctive refinement and promotion in the refinement calculus. *Formal Aspects of Computing* 11 (1999) 75–105.

[MoG90] Morgan, C. and Gardiner, P.H.B.: Data refinement by calculation. *Acta Informatica* 27 (1990) 481–503.

[Nel89] Nelson, G.: A generalization of Dijkstra's calculus. *ACM Transactions on Programming Languages and Systems*, 11 (1989) 517–561.

[Ole85] Oles, F.J.: Type algebras, functor categories and block structure. In M. Nivat and J.C. Reynolds (eds.): *Algebraic Methods in Semantics*, pp. 543–573. Cambridge University Press, Cambridge, 1985.

[OHT95] O'Hearn, P.W. and Tennent, R.D.: Parametricity and local variables. *J. ACM* 42 (1995) 658–709.

[War93] Ward, N.: *A recursion removal theorem*. Preprint Durham, 1993.

Received March 1999
Accepted in revised form January 2000 by E C R Hehner