Command Algebras, Recursion and Program Transformation

Hesselink, Wim H.

Published in:
Formal Aspects of Computing

Publication date:
1990

Formal Aspects of Computing (1990) 2: 60–104

Formal Aspects of Computing

Command Algebras, Recursion and Program Transformation

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

Keywords: Command algebra; Computational induction; Fixpoint; Lattice; Nondeterminacy/nondeterminism; Program transformation; Recursive procedure (or recursion); Semantic equivalence; Semantics; Stack; Weakest precondition

Abstract. Dijkstra's language of guarded commands is extended with recursion and transformed into algebra. The semantics is expressed in terms of weakest preconditions and weakest liberal preconditions. Extreme fixed points are used to deal with recursion. Unbounded nondeterminacy is allowed. The algebraic setting enables us to develop efficient transformation rules for recursive procedures. The main result is an algebraic version of the rule of computational induction. In this version, certain parts of the programs are restricted to finite nondeterminacy. It is shown that without this restriction the rule would not be valid. Some applications of the rule are presented. In particular, we prove the correctness of an iterative stack implementation of a class of simple recursive procedures.

0. Introduction

0.0. In this paper we investigate the laws that govern manipulation of commands in an imperative language with recursion and (possibly unbounded) nondeterminacy. We concentrate on the control structure and do not consider data structures or data refinement.

The paper is organised as follows. This introduction contains brief sketches of the main ideas and of some of the results. Command algebras are introduced in Section 1. Section 2 contains auxiliary material on complete lattices. Recursive

Correspondence and offprint requests to: Wim H. Hesselink, Rijksuniversiteit Groningen, Department of Computing Science, P.O. Box 800, 9700 AV Groningen, The Netherlands.

procedures are treated in Section 3. Section 4 is devoted to upper continuity, a concept related to finite nondeterminacy. The technical heart of the paper is Section 5, where transformation rules for recursion are obtained. The final rules are stated in Section 6. This section also contains some general applications. In Section 7 we treat the example that motivated our investigation.

In every section, the formulae are numbered consecutively. For reference to formulae of other sections we use the convention that $i(j)$ denotes formula (j) of section i.

0.1. Semantics, Equivalence and Program Transformation

The elements of our imperative programming language are called *commands*. The semantics of commands is expressed by *weakest preconditions* (*wp*) and *weakest liberal preconditions* (*wlp*). For a command s and a condition x on the state space, $wp.s.x$ is the weakest precondition such that execution of s terminates in a state where x holds; $wlp.s.x$ is the weakest precondition such that execution of s does not terminate or terminates in a state where x holds. We do not exclude miracles: it is possible to specify commands that can terminate "in a state where *false* holds". We come back to the question of miracles in Sections 0.4 and 1.3.

Commands can be connected by means of the operators for sequential composition ";" and nondeterministic choice "$[\!]$". These operators are characterised by the rules

$$w.(s;\, t).x = w.s.(w.t.x) \tag{0}$$
$$w.(s \,[\!]\, t).x = w.s.x \wedge w.t.x$$

for both $w = wp$ and $w = wlp$. Notice that we use the infix operator "." for function application. This operator binds from left to right, to allow currying. It has a higher binding power than all other operators. We also provide a mechanism for procedure abstraction and (possibly mutual) recursion.

We use X to denote the set of the conditions on the state space. For a command s the expressions $wp.s$ and $wlp.s$ are functions $X \to X$. We define *semantic equivalence* "$\cong$" to be the relation on commands given by

$$s \cong t \;\equiv\; (wp.s = wp.t) \wedge (wlp.s = wlp.t) \tag{1}$$

where "=" stands for equality of functions $X \to X$. Knowledge of "$\cong$" is the essence of program transformation. For, if $s \cong t$ then command s may be replaced by t without changing the semantics, but the executing mechanism may have different efficiencies for s and t. In formula (1), neither of the two conjuncts of the right-hand side can be omitted. In fact, if *skip* is the command that leaves the state unchanged and *abort* is the command that never terminates, then the command

$$skip \,[\!]\, abort$$

has the same *wp* as *abort* and the same *wlp* as *skip*, but it is not semantically equivalent to either command.

It often happens that two commands s and t are only known to be equivalent under a certain precondition c. This can be expressed by

$$?c;\, s \cong ?c;\, t$$

where command $?c$ is defined by

$$wp.(?c).x = wlp.(?c).x = \neg c \vee x \tag{3}$$

see Section 1.3 below. It may also happen that command s can be replaced by t because every possible outcome of t is a possible outcome of s. This situation can be expressed by

$$s \cong s \,[\!]\, t \tag{4}$$

Formulae (2) and (4) are further evidence that knowledge of "$\cong$" is the essence of program transformation.

0.2. Command Algebras and Procedures

At the basis of program transformation are algebraic rules like

$(s; t); u \;\cong\; s; (t; u)$, and

if c **then** s **else** t **fi**; $u \;\cong$ **if** c **then** s; u **else** t; u **fi**

For calculational purposes it is convenient to postulate some equalities that imply these equivalences. Thus, we arrive at the notion of command algebra. A command algebra is a set (of commands) with two binary operators ";" and "$[\!]$", that satisfy a list of axioms; see Section 1.1 below. The next step is to introduce homomorphisms of command algebras, cf. formula 1(10). For example, it turns out that formula (0) reflects the fact that *wp* and *wlp* are homomorphisms to the algebra $C.X$ of the conjunctive predicate transformers, cf. Section 1.2.

In order to introduce recursion we fix a basis to build upon. This basis consists of a command algebra B together with homomorphisms *wp* and *wlp* from B. Algebra B may be thought of as generated from simple commands by means of the operators ";" and "$[\!]$". Therefore, the commands in B may be regarded as the straight-line commands, cf. [DiS90] Chapter 7.

We let H denote the set of the procedure names. The command algebra $B[H]$ is the set of all command algebra expressions in elements of B and H. Such expressions are regarded as equal if and only if that is implied by the equalities of B and H together with the axioms.

Every procedure name $h \in H$ is supposed to be equipped with a body $d.h \in B[H]$. So, we have a function $d : H \to B[H]$, which is called the *declaration function*. Since the bodies $d.h$ may contain procedure names, recursion (both simple and mutual) is possible. The semantics of the commands in $B[H]$ is determined by extending the homomorphisms *wp* and *wlp* as given on B to the bigger algebra $B[H]$. The precise definition is given in Section 3.3. For the moment, it suffices to mention consequence 3(10), which says that every procedure is equivalent to its body:

$$h \cong d.h \tag{5}$$

0.3. Quantifications and Predicates

We write $(\forall x \in X : P : Q)$ to denote the predicate that Q holds for all $x \in X$ such that P. Similarly, $(\exists x \in X : P : Q)$ denotes that Q holds for some $x \in X$ such that P. The indication "$\in X$" is often omitted. If the range condition P is omitted, a default condition is meant, which is either *true* or specified in the context. We use the operators "$\equiv$" and "$\Rightarrow$" and "$\Leftarrow$" for logical equivalence and implication.

For x a predicate on a state space, $[x]$ denotes the proposition that x holds everywhere on the state space. This is used in particular in the notation $[x \Rightarrow y]$, which means that predicate x is stronger than y.

In the theory, it is convenient to apply an abstraction at this point. The set of programming predicates is treated as a distributive lattice with the order relation given by

$$x \leq y \equiv [x \Rightarrow y] \tag{6}$$

Therefore, a universal quantification $(\forall y \in Y :: y)$ of a set Y of programming predicates is treated as the greatest lower bound $(\bigwedge y \in Y :: y)$. Similarly, the existential quantification $(\exists y \in Y :: y)$ is treated as the least upper bound $(\bigvee y \in Y :: y)$.

0.4. Healthiness Conditions

In [Dij76], Dijkstra postulated some conditions on the predicate transformer *wp.s* of a given command *s*. The term "healthiness condition" seems to be due to Hoare [Hoa78]. We consider the following versions ([Dij90] Chapter 7):

Law of the excluded miracle: $wp.s.false = false$ (7)

Termination rule: for any predicate x, (8)

$$wp.s.x = wp.s.true \wedge wlp.s.x$$

Universal conjunctivity: for any set Y of predicates, (9)

$$wlp.s.(\bigwedge y \in Y :: y) = (\bigwedge y \in Y :: wlp.s.y)$$

Or-continuity: for any weakening sequence $(i : i \geq 0 : z.i)$ of predicates $z.i$, (10)

$$wp.s.(\bigvee i : i \geq 0 : z.i) = (\bigvee i : i \geq 0 : wp.s.(z.i))$$

In recent papers [Nel87] and [Mor87], it has been argued that the law of the excluded miracle (7) is an obstacle on the road to effective calculational rules for program development. We claim that it also hinders program transformation. Therefore, we abolish law (7). Actually, in our command algebras it is often convenient to have an element *magic* with

$$wp.magic.false = true.$$

We do not claim that magic can be implemented.

As indicated in [Dij76], condition (10) cannot be combined with unbounded nondeterminacy. Reasons for allowing unbounded nondeterminacy are given in [Bac87]. Therefore, we do not postulate (10). Our transformation rules for recursion will need a condition that some commands s are upper continuous (cf. Section 4.1). For general lattices, this concept is slightly stronger than or-continuity (10). In 4(15), we prove that or-continuity, upper continuity and finite nondeterminacy are equivalent in the model of relational calculus.

For us, the remaining healthiness conditions (8) and (9) have the status of axioms for commands s in command algebra B. In Sections 3.4 and 3.5, we prove that the validity of (8) and (9) extends to procedures $h \in H$. Actually, finite conjunctivity of *wp* and *wlp* is forced into the theory by the distributivity axiom

1(5). We need condition (9), since it is used in the proof of the transformation rule for recursion. It may be mentioned, however, that law (9) has been abolished in [MoG88].

0.5. The Transformation Rule for Recursion

The main result of this paper is an extension of the rule of computational induction to the case of unbounded nondeterminacy. In a first approximation, computational induction is the rule that a command expression s can be replaced by t if there is a congruence relation on command expressions that contains the pair $\langle s, t\rangle$ and that is stable under the operation of replacing every procedure name h by its body $d.h$. The induction starts with the case that all procedure names are replaced by *abort*. We allow unbounded nondeterminacy, but our commands must satisfy a certain technical condition expressed by $s, t \in Lia$.

Formally, the result is as follows. Let $d^*: B[H] \to B[H]$ be the function such that $d^*.s$ is obtained from s by substituting the body $d.h$ for every procedure name h in expression s. Similarly, let $da^*: B[H] \to B[H]$ be the function such that $da^*.s$ is obtained from s by substituting *abort* for every h in s. Then we have

Theorem. Let "$\smile$" be a binary relation on $E[H]$ such that

$$(\forall s, t \in B[H]: s \smile t: d^*.s \mathrel{\dot{\smile}} d^*.t \wedge da^*.s \cong da^*.t \wedge s, t \in Lia)$$

Then it follows that $(\forall s, t \in B[H]: s \smile t: s \cong t)$.

Here, relation "$\dot{\smile}$" is the congruence generated by relation "$\smile$" and *Lia* is a certain subset of $B[H]$. For more details we refer to the introduction of Section 5.

In Section 6.1, the final version of the transformation rule is obtained, in a form that allows accumulation of knowledge of congruences. The remainder of Section 6 contains some applications, commutation theorems that are used in Section 7. In Section 7, we use the transformation rule to prove the correctness of a stack implementation of a simple recursive procedure. As a special case, we get a relation between the euclidean algorithm and Dijkstra's function *fusc*.

0.6. Related Work and New Features

Our transformation rule for recursion is an unboundedly nondeterministic version of computational induction [Bak80, BaW81, Man74]. The semantic framework with *wp* and *wlp* is due to Dijkstra([Dij76]). Our command algebras are basic process algebras in the sense of [BBK87]. The observation that the law of the excluded miracle is an obstacle for efficient calculations and therefore should be abolished is due to Nelson [Nel87].

The theory of this paper is a complete reworking of the theory of [Hes89a]. The main change in the theory is that important algebraic properties as studied in [Hes88b] are here encapsulated in the concept of command algebra. We needed the shift from syntax to algebra for the development of program transformation.

New features in this paper are the treatment of *wp* and *wlp* as homomorphisms of algebras and the use of congruences for the study of program transformation. New also is the result that computational induction extends partially but not completely to programs with unbounded nondeterminacy. Dijkstra found his

function *fusc* while playing with the euclidean algorithm, but as far as we know the relationship between *fusc* and the euclidean algorithm has never been mentioned in the literature.

0.7. Linear Annotated Proofs

Whenever convenient, we use Feijen's proof format. For example, a proof of an implication $A \Rightarrow C$ may be given in the form

$$\begin{array}{ll} & C \\ \Leftarrow & \{\text{indication why } C \text{ follows from } B\} \\ & B \\ \equiv & \{\text{indication why } A \text{ and } B \text{ are equivalent}\} \\ & A \end{array}$$

If A is identical to *true*, this form may be used as a proof of C; for examples see Section 1.2. The proof format is also used for other relations than the implication; see the proofs of 2(5), 4(9) and 5(36). Occasionally, the justification of a step is postponed as a remaining proof obligation. In that case, we indicate the occurrence of a forward reference by means of a marginal (*). See the proofs of 5(29) and 5(36).

We do not doubt that the proofs can be polished further, but we leave the pleasure of doing so to the reader. In our view, proofs should not be regarded as a burden but as a challenge or pleasure.

1. Algebras and Lattices

1.0. This section contains the fundamental constructions and the algebraic and order-theoretic preliminaries. In Section 1.1, we introduce process algebras and command algebras, the order of determinacy on such algebras, and their homomorphisms and subalgebras. In Section 1.2, we show that every inf-lattice X has an associated process algebra $F.X$ and an associated command algebra $C.X$, which is a subalgebra of $F.X$, and we determine the order of determinacy of $F.X$ and $C.X$. In 1.3, we descend to the level of predicates (boolean functions on a state space) and languages that manipulate states and predicates.

1.1. Process Algebras and Command Algebras

The main algebraic concept in this paper is the concept of command algebra. It is a specialisation of the concept of (basic) process algebra, as introduced by Baeten, Bergstra and Klop [BBK87]. Therefore, it is convenient to define process algebras first, and then add the extra distributivity axiom that converts a process algebra into a command algebra.

A *process algebra* is a triple $(A, \mathbb{I}, ;)$, where A is a set and "$\mathbb{I}$" and ";" are binary operators on A. The elements of A are called *processes* or *commands*. The operators "$\mathbb{I}$" and ";" are associated with nondeterminate choice and sequential composition, respectively. The following axioms are postulated:

$$a \mathbb{I} a = a \tag{0}$$

$$a \| b = b \| a \tag{1}$$

$$(a \| b) \| c = a \| (b \| c) \tag{2}$$

$$(a; b); c = a; (b; c) \tag{3}$$

$$(a \| b); c = a; c \| b; c \tag{4}$$

In (4) and henceforth, we give the operator ";" a higher priority than "‖". In fact, ";" is regarded as a multiplication and "‖" is regarded as an addition operator. If no ambiguity can arise, we speak of the process algebra A instead of $(A, \|, ;)$.

A process algebra A is called a *command algebra* if and only if it also satisfies the other distributive law

$$a; (b \| c) = a; b \| a; c \tag{5}$$

On a process algebra A we define the binary relation $\leq$ by

$$a \leq b \;\equiv\; a = a \| b \tag{6}$$

As is well known (and easily verified), axioms (0), (1), (2) imply that $\leq$ is an order on A. Notice that we use the term *order* with the meaning of "partial order". Relation $\leq$ is called the *order of determinacy*. In fact, $a \leq b$ means $a = a \| b$, so that b is a possible choice for a; in other words, a is less determinate than b.

One can easily verify that $a \| b$ is the greatest lower bound of a and b in the ordered set $(A, \leq)$, and that

$$a \leq b \wedge c \leq d \;\Rightarrow\; a \| c \leq b \| d \tag{7}$$

We also have the rule

$$a \leq b \;\Rightarrow\; a; c \leq b; c \tag{8}$$

This is proved in

$$\begin{array}{lll} & a; c \leq b; c & \\ \equiv & \{(6)\} & a; c = a; c \| b; c \\ \equiv & \{(4)\} & a; c = (a \| b); c \\ \Leftarrow & \{(6)\} & a \leq b \end{array}$$

In the same way, if A is a command algebra, axiom (5) is used to prove

$$a \leq b \;\Rightarrow\; c; a \leq c; b \tag{9}$$

A function $w : A \to B$ between process algebras A and B is called a *homomorphism* if and only if it satisfies

$$\begin{aligned} w.(p \| q) &= w.p \| w.q \\ w.(p; q) &= w.p; w.q \end{aligned} \tag{10}$$

Every homomorphism of process algebras is monotone, i.e.

$$a \leq b \;\Rightarrow\; w.a \leq w.b \tag{11}$$

This is proved in

$$\begin{array}{lll} & w.a \leq w.b & \\ \equiv & \{(6)\} & w.a = w.a \| w.b \\ \Leftarrow & \{(10)\} & a = a \| b \\ \equiv & \{(6)\} & a \leq b \end{array}$$

A subset U of a process algebra A is called a *subalgebra* if and only if

$$(\forall p, q \in U :: p \| q \in U \wedge p; q \in U) \tag{12}$$

A subalgebra U of A is a process algebra in its own right and the identity function $U \to A$ is a homomorphism.

1.2. The Algebras of an Inf-Lattice

An *inf-lattice* is defined to be an ordered set $(X, \leq)$ with a biggest element $tt \in X$ and, for every two elements x and y, a greatest lower bound $x \wedge y$. If no ambiguity can arise we write X instead of $(X, \leq)$. A *lattice* is defined to be an inf-lattice X with a smallest element ff and, for every two elements x and y, a least upper bound $x \vee y$.

Let X be an inf-lattice. We write $F.X$ to denote the set of functions $f: X \to X$. It is equipped with the structure of a process algebra by defining

$$\begin{aligned} (f \| g).x &= f.x \wedge g.x \\ (f; g).x &= f.(g.x) \end{aligned} \tag{13}$$

The verification of the axioms (0) up to (3) is immediate. Axiom (4) is verified in

$$\begin{array}{ll} & (f \| g); h = f; h \| g; h \\ \equiv & \{\text{equality of functions}\} \\ & (\forall x :: ((f \| g); h).x = (f; h \| g; h).x) \\ \equiv & \{(13)\} \\ & (\forall x :: (f \| g).(h.x) = (f; h).x \wedge (g; h).x) \\ \equiv & \{(13)\} \\ & \textit{true}. \end{array}$$

A function $f: X \to X$ is called *conjunctive* if and only if

$$f.(x \wedge y) = f.x \wedge f.y \tag{14}$$

We write $C.X$ to denote the set of conjunctive functions $f: X \to X$. One verifies that

$$(\forall f, g \in C.X :: f \| g \in C.X \wedge f; g \in C.)$$

so that $C.X$ is a subalgebra of $F.X$. Actually, we have

$$C.X \text{ is a command algebra.} \tag{15}$$

This is proved by observing that (5) holds, since for any f, g, $h \in C.X$ we have

$$\begin{array}{ll} & f; (g \| h) = f; g \| f; h \\ \equiv & \{\text{equality of functions, (13)}\} \\ & (\forall x :: f.(g.x \wedge h.x) = f.(g.x) \wedge f.(h.x)) \\ \equiv & \{f \text{ is conjunctive, (14)}\} \\ & \textit{true} \end{array}$$

By a variation of this calculation one can show that, if X has an element different from ff, process algebra $F.X$ is not a command algebra.

The order of determinacy of the algebras $F.X$ and $C.X$ satisfies

$$f \leq g \;\equiv\; (\forall x :: f.x \leq g.x) \tag{16}$$

This proved in

$$
\begin{array}{ll}
& f \leq g \\
\equiv & \{(6) \text{ and equality of functions}\} \\
& (\forall x :: f.x = (f \| g).x) \\
\equiv & \{(13)\} \\
& (\forall x :: f.x = f.x \wedge g.x) \\
\equiv & \{\wedge \text{ gives greatest lower bound}\} \\
& (\forall x :: f.x \leq g.x)
\end{array}
$$

1.3. Predicate Calculus and Programming

For us, the main example of an inf-lattice is the set of the programming predicates. This is the set X of the boolean functions on a state space St. The set X is ordered by formula 0(6), or more explicitly

$$x \leq y \equiv (\forall r \in St :: x.r \Rightarrow y.r) \tag{17}$$

With this order, X is a lattice (actually, a complete lattice, cf. 2.1 below). In fact, an arbitrary subset Y of X has a greatest lower bound $\bigwedge(Y)$ and a least upper bound $\bigvee(Y)$, given by

$$
\begin{aligned}
&\textstyle\bigwedge(Y).r = (\forall y \in Y :: y.r) \\
&\textstyle\bigvee(Y).r = (\exists y \in Y :: y.r)
\end{aligned}
\tag{18}
$$

In particular, we have

$$
\begin{aligned}
&(x \wedge y).r = x.r \wedge y.r \\
&(x \vee y).r = x.r \vee y.r
\end{aligned}
$$

where $\wedge$ and $\vee$ on the right-hand side are boolean conjunction and disjunction. The smallest and the biggest element of X are the functions ff and tt, respectively, which are given by

$$ff.r = false, \quad tt.r = true$$

The algebra of straight-line commands is constructed as follows. Let S be a set of simple commands. The semantics of the simple commands is supposed to be given by functions wp, $wlp : S \to C.X$ that satisfy healthiness conditions 0(8) and 0(9). For $s \in S$ and $x \in X$, function $wp.s.x$ is the weakest precondition such that execution of command s terminates in a state that satisfies condition x. Function $wlp.s.x$ is the weakest precondition such that execution of s does not terminate or terminates in a state that satisfies x. Note that we use the standard postulate that $wp.s$ and $wlp.s$ are conjunctive predicate transformers for every command s.

Let $S\#$ be the language that is generated by S and the rules summarised in

$$(\forall p, q \in S\# :: (p, q) \in S\# \wedge (p \| q) \in S\#)$$

We extend functions wp and wlp to functions $S\# \to C.X$ by the condition that (10) holds with w replaced by wp and wlp. Let semantic equivalence ($\cong$) in $S\#$ be defined as in 0(1), so that

$$p \cong q \equiv (wp.p = wp.q) \wedge (wlp.p = wlp.q) \tag{19}$$

It is clear that $\cong$ is an equivalence relation on $S\#$. Let B be the quotient set $(S\#)/\cong$, i.e. the set of equivalence classes for $\cong$. In $S\#$, it holds that

$$(a \cong a1) \wedge (b \cong b1) \Rightarrow (a; b \cong a1; b1) \wedge (a \| b \cong a1 \| b1)$$

Therefore, the operators "‖" and ";" induce corresponding operators on the set B. Since $C.X$ is a command algebra, it is easy to verify that the relation $\cong$ on $S\#$ satisfies the formulae obtained from axioms (0) through (5) after replacing "=" by "$\cong$":

$$a \| a \cong a, \quad a \| b \cong b \| a, \quad \text{etc.}$$

It follows that the quotient set B with the induced operators "‖" and ";" is a command algebra, and that *wp* and *wlp* are well-defined homomorphisms $B \to C.X$. The algebra B is called the *algebra of straight-line commands.*

It is useful to note that conditional statements can be implemented on this level. In fact, let simple commands of the form $?b$ be defined by

$$\begin{aligned} wp.(?b).x &= \neg b \vee x \\ wlp.(?b).x &= \neg b \vee x \end{aligned} \tag{20}$$

Now the construction

if b **then** $s0$ **else** $s1$ **fi**

is equivalent to $(?b; s0) \| (?\neg b; s1)$. Compare [Bak80] *p.* 271. Commands of the form $?b$ are called *guards.* Although, in general, they do not satisfy the law of the excluded miracle, cf. 0(7), they are very useful, see e.g. formula 0(2) and Section 7.

Let us also introduce a command $!b$ that skips if b holds and does not terminate if $\neg b$ holds. It is given by

$$\begin{aligned} wp.(!b).x &= b \wedge x \\ wlp.(!b).x &= \neg b \vee x \end{aligned} \tag{21}$$

Now one can verify that Dijkstra's conditional construct

if $b0 \to s0 \| b1 \to s1$ **fi**

is equivalent to

$$!(b0 \vee b1); (?b0; s0 \| ?b1; s1)$$

For proofs and for more results on $?b$ and $!b$, we refer to [Hes88b].

2. Completeness

2.0. This section contains technical preparations and reference material. Therefore, some readers may prefer to skip the section and come back when references to the section appear. In Section 2.1, we give some results on complete lattices of functions. Section 2.2 contains an extended version of the theorem of Knaster and Tarski.

2.1. Completeness of Lattices and Algebras

An inf-lattice X is called *complete* if and only if every subset Y or X has a greatest lower bound $\bigwedge(Y)$. As is well known, a complete inf-lattice is a complete lattice. For, if Y is a subset of X, the greatest lower bound of the set of the upper bounds of Y is the least upper bound of Y. This proves that every subset of X has a least upper bound.

Let X be a complete lattice. For a set V, let the set of functions $V \to X$ be equipped with the order $\leq$ defined by

$$f \leq g \equiv (\forall v \in V :: f.v \leq g.v) \tag{0}$$

Then we have

(1) **Theorem** (extrema of functions). The set of functions $V \to X$ is complete. If F is some subset of $V \to X$, the greatest lower bound $\bigwedge(F)$ is the function $f0$ given by

$$f0.v = (\bigwedge f \in F :: f.v) \tag{2}$$

The least upper bound $\bigvee(F)$ is the function $f1$ given by

$$f1.v = (\bigvee f \in F :: f.v) \tag{3}$$

Proof. Since X is complete, function $f0$ is well defined. For any function $g : V \to X$, it holds

$$\begin{array}{lll} & g \leq f0 & \\ \equiv & \{(0)\} & (\forall v \in V :: g.v \leq f0.v) \\ \equiv & \{(2)\} & (\forall v \in V, f \in F :: g.v \leq f.v) \\ \equiv & \{(0)\} & (\forall f \in F :: g \leq f) \end{array}$$

This proves that $f0 = \bigwedge(F)$. In the same way, one proves that $f1 = \bigvee(F)$. □

Remark. The formulae 1(18) are the cases of this result where $V = St$ and where X is the set of the two booleans values with *false* < *true*.

A subset L of an ordered set X is defined to be *linear* if and only if

$$(\forall x, y \in L :: x \leq y \vee y \leq x) \tag{4}$$

The importance of linearity (for us) stems from the following result:

(5) **Theorem** (diagonalisation). Let X be a complete lattice. Let L be a linear subset of some ordered set. Let $p : L \to L \to X$ be a monotone function in both arguments, i.e.

$$(\forall x, y, z \in L : x \leq y : p.x.z \leq p.y.z \wedge p.z.x \leq p.z.y)$$

Then we have

(a) $(\bigvee x, y \in L :: p.x.y) = (\bigvee x \in L :: p.x.x)$

(b) $(\bigwedge x, y \in L :: p.x.y) = (\bigwedge x \in L :: p.x.x)$

Proof. (a) is proved in the following calculation

$$\begin{array}{ll} & (\bigvee x, y \in L :: p.x.y) \\ = & \{\text{range union with (4)}\} \\ & (\bigvee x, y : x \geq y : p.x.y) \vee (\bigvee x.y : y \geq x : p.x.y) \\ = & \{\text{quantifications}\} \\ & (\bigvee x :: (\bigvee y : x \geq y : p.x.y)) \vee (\bigvee y :: (\bigvee x : y \geq x : p.x.y)) \\ = & \{p \text{ is monotone in both arguments}\} \\ & (\bigvee x :: p.x.x) \vee (\bigvee y :: p.y.y) \\ = & \{\text{calculus}\} \\ & (\bigvee x :: p.x.x) \end{array}$$

(b) follows by symmetry. □

Let X be a complete lattice. Let Q stand for one of the two quantifiers $\bigvee$ or $\bigwedge$, which are used to denote least upper bounds and greatest lower bounds, respectively. A subset V of X is defined to be *Q-closed* if and only if

$$(\forall F: F \subset V: Q(F) \in V) \tag{6}$$

It is defined to be *Q-decked* if and only if

$$(\forall L: L \subset V \wedge L \text{ is linear}: Q(L) \in V) \tag{7}$$

Clearly, every Q-closed set is Q-decked. The empty set $\varnothing$ is linear, so that a Q-decked set V contains $Q(\varnothing)$. If $Q = \bigvee$, then $Q(\varnothing)$ is the smallest element of X. If $Q = \bigwedge$, then $Q(\varnothing)$ is the greatest element of X.

We now consider the ordered set $F.X$ and its subalgebra $C.X$, cf. Section 1.2. By 1(16), the order of determinacy of $F.X$ is equal to the order given by (0). Therefore, $F.X$ is complete by Theorem (1).

(8) **Theorem.** $C.X$ is $\bigwedge$-closed in $F.X$. Therefore, command algebra $C.X$ is a complete lattice.

Proof. Let F be a subset of $C.X$. Put $f0 = \bigwedge(F)$, cf. (2). We show that $f0 \in C.X$, i.e. that $f0$ is conjunctive, cf. 1(14). This is proved in

$$\begin{array}{ll} & f0.(x \wedge y) = f0.x \wedge f0.y \\ \equiv & \{(2), \text{ let } f \text{ range in the set } F\} \\ & (\bigwedge f :: f.(x \wedge y)) = (\bigwedge f :: f.x) \wedge (\bigwedge f :: f.x) \\ \equiv & \{\text{all } f \text{ are conjunctive: 1(14)}\} \\ & (\bigwedge f :: f.x \wedge f.y) = (\bigwedge f :: f.x) \wedge (\bigwedge f :: f.x) \\ \equiv & \{\text{calculus with greatest lower bounds}\} \\ & \mathit{true} \end{array}$$

This proves that $C.X$ is *$\bigwedge$-closed in $F.X$*. Therefore, it is a complete inf-lattice and hence a complete lattice. □

(9) **Theorem.** Assume that the complete lattice X is such that every subset y satisfies the distributive law

$$x \wedge (\bigvee y \in Y :: y) = (\bigvee y \in Y :: x \wedge y).$$

Then $C.X$ is $\bigvee$-decked in $F.X$. Therefore, least upper bounds of linear subsets of $C.X$ can be calculated in $F.X$.

Proof. Let F be a linear subset of $C.X$. Let the function $f1 \in C.X$, i.e. that $f1$ is conjunctive, cf. 1(14). This is proved by the following calculation:

$$\begin{array}{ll} & f1.x \wedge f1.y \\ = & \{(3), \text{ let } f \text{ and } g \text{ range over } F\} \\ & (\bigvee f :: f.x) \wedge (\bigvee g :: g.y) \\ = & \{\text{the distributivity, twice}\} \\ & (\bigvee f, g :: f.x \wedge g.y) \\ = & \{\text{diagonalization (5)(a) with } x := f,\ y := g \text{ and } p.f.g := f.x \wedge g.y\} \\ & (\bigvee f :: f.x \wedge y) \\ = & \{\text{all } f \text{ are conjunctive 1(14), and (3)}\} \\ & f1.(x \wedge y) \quad \square \end{array}$$

2.2. A Version of the Theorem of Knaster and Tarski

In this subsection, we prove a convenient version of the theorem of Knaster and Tarski. This version can be proved by transfinite induction, but it was a rewarding exercise to try and avoid that powerful theory.

(10) **Theorem.** Let $f: X \to X$ be a monotone function. Then f has a smallest fixed point xa and a biggest fixed point xb in X. Let V be a subset of X that is *f-invariant*, i.e. $(\forall x \in V :: f.x \in V)$.

(a) If V is $\bigvee$-decked, then $xa \in V$

(b) If V is $\bigwedge$-decked, then $xb \in V$

Remark. This version is stronger than the version in [Hes89a] Section 1.3, for there we proved $xa \in V$ ($xb \in V$) under the stronger condition that V is $\bigvee$-closed ($\bigwedge$-closed).

Proof. By symmetry, it suffices to prove that f has a smallest fixed point xa, which satisfies (a). This is done in four parts. In part *A*, we construct a subset Y of X (and of V) that is $\bigvee$-decked and f-invariant. In part *B*, we prove a formula that implies that all elements of Y are below all fixed points of f. In part *C*, we prove that set Y is linear. In part *D*, we prove that the least upper bound of Y is an element xa that satisfies all claims of the theorem.

Part A. Let Y be defined as the intersection of all subsets U of X that are $\bigvee$-decked and f-invariant. Thus, for any $x \in X$, we have

$$x \in Y \equiv (\forall U : U \text{ is } \bigvee\text{-decked} \wedge U \text{ is } f\text{-invariant}: x \in U) \tag{11}$$

As the given set V is $\bigvee$-decked and f-invariant, we have

$$Y \subset V \tag{12}$$

We prove that Y is $\bigvee$-decked and f-invariant. In order to prove that Y is $\bigvee$-decked, cf. (7), let a linear subset L of Y be given. We observe

$$\begin{array}{ll} & \bigvee(L) \in Y \\ \equiv & \{(11)\} \\ & (\forall U : U \text{ is } \bigvee\text{-decked} \wedge U \text{ is } f\text{-invariant}: \bigvee(L) \in U) \\ \Leftarrow & \{Y \subset U \text{ by (11); and } L \subset Y\} \\ & (\forall U : U \text{ is } \bigvee\text{-decked} \wedge L \subset U : \bigvee(L) \in U) \\ \equiv & \{(7) \text{ and } L \text{ is linear}\} \\ & \mathit{true} \end{array}$$

The f-invariance of Y is proved in

$$\begin{array}{ll} & f.y \in Y \\ \equiv & \{(11)\} \\ & (\forall U : U \text{ is } \bigvee\text{-decked} \wedge U \text{ is } f\text{-invariant}: f.y \in U) \\ \Leftarrow & \{U \text{ is } f\text{-invariant}\} \\ & (\forall U : U \text{ is } \bigvee\text{-decked} \wedge U \text{ is } f\text{-invariant}: y \in U) \\ \equiv & \{(11)\} \\ & y \in Y \end{array}$$

This proves

$$Y \text{ is } \bigvee\text{-decked} \wedge Y \text{ is } f\text{-invariant} \tag{13}$$

Part B. In the next stages, formula (11) is used as an induction principle: we prove properties of the elements of Y by showing that the set where the property

holds is $\bigvee$-decked and f-invariant, and therefore contains Y. As an easy case, we first establish

$$(\forall y \in Y :: (\forall x \in X : f.x \leq x : y \leq x)) \tag{14}$$

Let the set $Z0$ be defined by

$$y \in Z0 \equiv (\forall x \in X : f.x \leq x : y \leq x) \tag{15}$$

We prove that $Z0$ is $\bigvee$-decked and f-invariant. For any linear subset L of $Z0$, we have

$$\begin{array}{ll} & \bigvee(L) \in Z0 \\ \equiv & \{(15)\} \\ & (\forall x \in X : f.x \leq x : \bigvee(L) \leq x) \\ \equiv & \{\text{definition } \bigvee(L)\} \\ & (\forall x \in X : f.x \leq x : (\forall y \in L :: y \leq x)) \\ \equiv & \{\text{interchange of quantifications}\} \\ & (\forall y \in L :: (\forall x \in X : f.x \leq x : y \leq x)) \\ \equiv & \{L \subset Z0 \text{ and } (15)\} \\ & \mathit{true} \end{array}$$

Therefore $Z0$ is $\bigvee$-decked, cf. (7). The f-invariance of $Z0$ is proved in

$$\begin{array}{ll} & f.z \in Z0 \\ \equiv & \{(15)\} \\ & (\forall x : f.x \leq x : f.z \leq x) \\ \Leftarrow & \{\text{transitivity}\} \\ & (\forall x : f.x \leq x : f.z \leq f.x) \\ \Leftarrow & \{\text{monotonicity of } f\} \\ & (\forall x : f.x \leq x : z \leq x) \\ \equiv & \{(15)\} \\ & z \in Z0 \end{array}$$

By (11) with $U := Z0$, this proves that Y is contained in $Z0$. By (15), this establishes formula (14).

Part C. We turn to the linearity of Y. In view of (4), we consider the set $Z1$ given by

$$z \in Z1 \equiv (\forall y \in Y :: y \leq z \vee z \leq y) \tag{16}$$

For any linear subset L of $Z1$, we observe that

$$\begin{array}{ll} & \bigvee(L) \in Z1 \\ \equiv & \{(16)\} \\ & (\forall y \in Y :: y \leq \bigvee(L) \vee \bigvee(L) \leq y) \\ \Leftarrow & \{\text{definition of } \bigvee(L)\} \\ & (\forall y \in Y :: (\exists v \in L :: y \leq v) \vee (\forall w \in L :: w \leq y)) \\ \equiv & \{\text{distribution of } \vee \text{ over } \forall\} \\ & (\forall y \in Y :: (\forall w \in L :: (\exists v \in L :: y \leq v) \vee w \leq y)) \\ \Leftarrow & \{\text{one point rule}\} \\ & (\forall y \in Y :: (\forall w \in L :: y \leq w \vee w \leq y)) \\ \equiv & \{L \subset Z1 \text{ and } (16)\} \\ & \mathit{true} \end{array}$$

By (7), this proves

$$Z1 \text{ is } \bigvee\text{-decked} \tag{17}$$

In order to prove that $Z1$ is f-invariant, we consider a given $z \in Z1$. We want to

prove that $f.z \in Z1$. By (16), we have

$$f.z \in Z1 \;\equiv\; (\forall y \in Y :: y \leq f.z \vee f.z \leq y) \tag{18}$$

Therefore, we introduce $Z2$ given by

$$y \in Z2 \;\equiv\; y \in Y \;\wedge\; (y \leq f.z \vee f.z \leq y) \tag{19}$$

We verify that $Z2$ is $\bigvee$-decked and f-invariant. For any linear subset L of $Z2$, we have

$$\begin{array}{ll} & \bigvee(L) \in Z2 \\ \equiv & \{(13) \text{ and } (19)\} \\ & \bigvee(L) \leq f.z \;\vee\; f.z \leq \bigvee(L) \\ \Leftarrow & \{\text{definition of } \bigvee(L)\} \\ & (\forall v \in L :: v \leq f.z) \vee (\exists w \in L :: f.z \leq w) \\ \equiv & \{\text{distribution of } \vee \text{ over } \forall\} \\ & (\forall v \in L :: v \leq f.z \vee (\exists w \in L :: f.z \leq w)) \\ \Leftarrow & \{\text{one point rule}\} \\ & (\forall v \in L :: v \leq f.z \vee f.z \leq v) \\ \equiv & \{L \subset Z2 \text{ and } (19)\} \\ & \mathit{true} \end{array}$$

The f-invariance of $Z2$ is proved in

$$\begin{array}{ll} & f.y \in Z2 \\ \equiv & \{(19)\} \\ & f.y \in Y \;\wedge\; (f.y \leq f.z \vee f.z \leq f.y) \\ \Leftarrow & \{(13) \text{ and } f \text{ monotone}\} \\ & y \in Y \;\wedge\; (y \leq z \vee z \leq y) \\ \Leftarrow & \{\text{first conjunct of } (19), \text{ then } (16) \text{ and } z \in Z1\} \\ & y \in Z2 \end{array}$$

By (11) with $U := Z2$, this proves that Y is contained in $Z2$. By (18) and (19), this proves that $f.z \in Z1$. Here, z was an arbitrary element of $Z1$. Therefore, $Z1$ is f-invariant. By (17), and (11) with $U := Z1$, it follows that Y is contained in $Z1$. By (16) and (4), this proves that the set Y is linear.

Part D. We define $xa = \bigvee(Y)$. Since Y is $\bigvee$-decked, cf. (13), and a linear subset of itself, it contains its own least upper bound xa, by (7). By (12) and (14), it follows that

$$xa \in V \;\wedge\; (\forall x \in X : f.x \leq x : xa \leq x) \tag{20}$$

The element xa is a fixed point of function f because of

$$\begin{array}{ll} & xa = f.xa \\ \equiv & \{\text{antisymmetry}\} \\ & xa \leq f.xa \;\wedge\; f.xa \leq xa \\ \Leftarrow & \{(20) \text{ with } x := f.xa\} \\ & f.(f.xa) \leq f.xa \;\wedge\; f.xa \leq xa \\ \equiv & \{f \text{ is monotone}\} \\ & f.xa \leq xa \\ \Leftarrow & \{xa = \bigvee(Y)\} \\ & f.xa \in Y \\ \equiv & \{xa \in Y \text{ and } (13)\} \\ & \mathit{true} \end{array}$$

For any fixed point x of f, we have $f.x \leq x$, and hence $xa \leq x$ by (20). Therefore, xa is the smallest fixed point of f. □

Remark. Definitions (16) and (19) are due to Erik Saaman, who simplified an earlier proof.

3. The Semantics of Recursion

3.0. Let X be a complete lattice that satisfies the distributive law (cf. Theorem 2(9))

$$x \wedge (\bigvee y \in Y :: y) = (\bigvee y \in Y :: x \wedge y) \tag{0}$$

The biggest element of X is denoted by tt, the smallest element by ff.

Let B be a command algebra. The semantics of commands in B is supposed to be given by homomorphisms wp and $wlp : B \to C.X$ that satisfy for any $s \in B$, any $x \in X$, and any subset Y of X the healthiness conditions (analogous to 0(8) and 0(9))

$$wp.s.x = wp.s.tt \wedge wlp.s.x \tag{1}$$

$$wlp.s.(\bigwedge y \in Y :: y) = (\bigwedge y \in Y :: wlp.s.y) \tag{2}$$

Now procedures are introduced. One declares a set H of procedure names h, with associated procedure bodies $d.h$. Because of recursion, the bodies $d.h$ may contain procedure names. Therefore, the bodies $d.h$ are command algebra expressions in elements of B and H. The semantics of recursion is supposed to be such that procedure h is semantically equivalent to its body $d.h$.

In program transformation, we need algebraic manipulation of procedure bodies before the identification of semantically equivalent commands. In fact, an arbitrary declaration $d.h = E$ would lead to $h \cong E$, but we are not allowed to replace the declaration $d.h = E$ by the declaration $d.h = h$, since the latter declaration is expected to give a non-terminating procedure. On the other hand, we want to be able to argue that e.g. declaration

$$d.h = (a;\ h \,[\!]\, b);\ c$$

is the same as declaration

$$d.h = a;\ h;\ c \,[\!]\, b;\ c$$

Therefore, we introduce a command algebra $B[H]$ that contains the elements of H and that has B as a subalgebra. A declaration is a function $d : H \to B[H]$. The semantics of such a declaration is defined by extending the homomorphisms wp and wlp to homomorphisms $B[H] \to C.X$. We use the theorem of Knaster and Tarski to construct these extensions. The formal development is as follows.

3.1. Polynomial Command Algebras and Recursive Procedures

Let H a set of symbols disjoint from B. The polynomial command algebra $B[H]$ is defined as the set of all command algebra expressions in elements of B and H, modulo the equalities induced by the axioms 1(0)-1(5).

(3) **Theorem.** $B[H]$ contains the set H. It contains B as a subalgebra. For every homomorphism of command algebras $w : B \to A$ and every function $d : H \to A$,

there is precisely one homomorphism of command algebras $d^*: B[H] \to A$ that extends the functions d and w. The only subalgebra of $B[H]$ that contains B and H is $B[H]$ itself.

Sketch of proof. On the set of command expressions there is precisely one function to A that satisfies formula 1(10) and that extends the functions w and d. This is proved by structural induction over the expressions. Since the axioms 1(0)-1(5) hold in A, the extended function respects the equalities induced by the six axioms. Therefore, the function is well defined on $B[H]$. Clearly, it is a homomorphism. Uniqueness is obvious. It is clear that $B[H]$ is generated by B and H. □

A *declaration* is defined to be a function $d: H \to B[H]$. The intention is that the elements of H are procedure names of a particular program and that $d.h$ is the body of procedure h, for every $h \in H$. In this way, mutually recursive procedures are possible. By Theorem (3), a declaration d has a unique extension to a homomorphism $d^*: B[H] \to B[H]$ that extends function d and the identity function of B.

Henceforth, we let $d: H \to B[H]$ be a declaration. Let a homomorphism of command algebras $wg: B \to C.X$ be given. Function wg is introduced in order to avoid a case distinction between wp and wlp.

An *interpretation* of declaration d over wg is defined to be a homomorphism $w: B[H] \to C.X$ that satisfies

$$(\forall a \in B :: w.a = wg.a) \\ \wedge (\forall h \in H :: w.h = w.(d.h)) \tag{4}$$

The problem of the interpretation of recursion is to guarantee the existence of an interpretation and to choose a convenient candidate. Actually, the theoretician has less freedom than suggested here. For the intention of our calculus is that command algebra B models some imperative programming language and that the interpretation of recursion coincides with a convenient operational semantics.

3.2. The Extension Theorem

Let WG be the set of functions $w: H \to C.X$. By Theorem (3), every such function w has a unique extension $w^*: B[H] \to C.X$ that satisfies $w^*.a = wg.a$ for all $a \in B$. If $w \in WG$, the composition $w^* \circ d$ is also an element of WG. Therefore, we can define $D: WG \to WG$ by

$$D.w = w^* \circ d \tag{5}$$

For $w \in WG$, the homomorphism w^* is an interpretation if and only if $D.w = w$, i.e. if and only if w is a fixed point of D. This is proved in

$$\begin{array}{ll} & D.w = w \\ \equiv & \{(5)\} \\ & w^* \circ d = w : H \to C.X \\ \equiv & \{(4);\ w^* \text{ extends } w \text{ and } wg\} \\ & w^* \text{ is an interpretation} \end{array}$$

Therefore, the standard way to define interpretation w would be to apply the theorem of Knaster and Tarski 2(10) to the monotone function D. So, we have to define an order $\leq$ on WG, and to verify that $(WG, \leq)$ is complete and that function $D: WG \to WG$ is monotone.

We use the order of determinacy of $C.X$ to make WG into a complete lattice with the order given by

$$v \leq w \;\equiv\; (\forall h \in H :: v.h \leq w.h) \tag{6}$$

This order is a special case of 2(0) with $v := H$ and $X := C.X$. Therefore, by Theorem 2(1), we have that $(WG, \leq)$ is complete.

In order to prove monotonicity of D, we claim

$$v \leq w \;\Rightarrow\; (\forall r \in B[H] :: v^*.r \leq w^*.r) \tag{7}$$

This formula is proved by structural induction on r. If $r \in B$ then $v^*.r = wg.r = w^*.r$. The case $r \in H$ follows from (6). The induction step is proved by observing that for $p, q \in B[H]$ and elements $v, w \in WG$ we have

$$\begin{array}{ll} & v^*.p \leq w^*.p \;\wedge\; v^*.q \leq w^*.q \\ \Rightarrow & \{1(7), 1(8), 1(9)\} \\ & v^*.p \,\|\, v^*.q \leq w^*.p \,\|\, w^*.q \;\wedge\; v^*.p;\, v^*.q \leq w^*.p;\, w^*.q \\ \equiv & \{v^*, w^* \text{ are homomorphisms}\} \\ & v^*.(p \,\|\, q) \leq w^*.(p \,\|\, q) \;\wedge\; v^*.(p;\, q) \leq w^*.(p;\, q) \end{array}$$

This concludes the proof of (7).

Now we can prove that the function $D: WG \to WG$ is monotone, i.e.

$$v \leq w \Rightarrow D.v \leq D.w \tag{8}$$

This is proved in

$$\begin{array}{lll} & D.v \leq D.w & \\ \equiv & \{(6)\} & (\forall h \in H :: D.v.h \leq D.w.h) \\ \equiv & \{(5)\} & (\forall h \in H :: v^*.(d.h) \leq w^*.(d.h)) \\ \Leftarrow & \{(7)\} & v \leq w \end{array}$$

Since WG is complete, it follows from the theorem of Knaster and Tarski 2(10), that the monotone function $D: WG \to WG$ has a smallest fixed point (say wa) and a greatest fixed point (say wb). This proves

(9) **Theorem.** Let $wg: B \to C.X$ be a homomorphism of command algebras. Every declaration $d: H \to B[H]$ has a smallest interpretation $wa^*: B[H] \to C.X$ and a greatest interpretation $wb^*: B[H] \to C.X$ over wg.

3.3. The Application to Programming

We extend the given homomorphisms wp and wlp to $B[H]$ by defining $wp: B[H] \to C.X$ to be the smallest interpretation of declaration d over $wp: B \to C.X$, and $wlp: B[H] \to C.X$ to be the greatest interpretation of declaration d over $wlp: B \to C.X$, cf. theorem (9). For an operational justification of this definition we refer to [Hes88a], Theorems 4.3 and 4.4. For $h \in H$, we observe

$$\begin{array}{ll} & h \cong d.h \\ \equiv & \{0(1)\} \\ & wp.h = wp.(d.h) \;\wedge\; wlp.h = wlp.(d.h) \\ \equiv & \{wp \text{ and } wlp \text{ are interpretations (4)}\} \\ & true \end{array} \tag{10}$$

For proving properties of wp and wlp, we need some definitions. Since we now have two homomorphisms $wp, wlp: B \to C.X$, every element $w \in WG$ induces two

homomorphisms w^r, $w^l : B[H] \to C.X$, characterised by

$$(\forall a \in B :: w^r.a = wp.a \wedge w^l.a = wlp.a) \ \wedge \ (\forall h \in H :: w^r.h = w^l h = w.h) \qquad (11)$$

Accordingly, function D given by (5) specialises to two functions Dp, $Dlp : WG \to WG$ given by

$$Dp.w = w^r \circ d, \quad Dlp.w = w^l \circ d \qquad (12)$$

The extended homomorphisms wp and wlp are given by $wp = wa^r$ and $wlp = wb^l$, where wa is the smallest fixpoint of Dp and wb is the biggest fixpoint of Dlp.

3.4. The Universal Conjunctivity of *wlp* for *B*[*H*]

We extend healthiness condition (2) to the commands in $B[H]$. The results of this subsection will be used again in Section 5.

Let Wun be the subset of WG given by

$$w \in Wun \ \equiv \qquad (13)$$

$$(\forall h, Y : h \in H \wedge Y \subset X : w.h.(\bigwedge y \in Y :: y) = (\bigwedge y \in Y :: w.h.y))$$

The term Wun can be associated with "universal conjectivity". We claim that formula (13) can be extended to

$$(\forall w \in Wun, p \in B[H], Y : Y \subset X : \qquad (14)$$

$$w^l.p.(\bigvee y \in Y :: y) = (\bigwedge y \in Y :: w^l.p.y))$$

In order to prove (14), we consider $w \in Wun$ and define the subset K of $B[H]$ by

$$p \in K \ \equiv$$
$$(\forall Y : Y \subset X : w^l.p.(\bigwedge y \in Y :: y) = (\bigwedge y \in Y :: w^l.p.y))$$

As the triple (B, wp, wlp) satisfies formula (2), it follows from (11) that $B \subset K$. From (13) we get $H \subset K$. By a straightforward calculation, which is based on Definition 1(13), one can prove that

$$(\forall p, q \in K :: p \,[\!]\, q \in K \wedge p ; q \in K)$$

Therefore, structural induction shows that $K = B[H]$. This proves (14).

Now in order to extend (2) to the algebra $B[H]$, it suffices by (14) to show that $wb \in Wun$, where wb is the biggest fixed point of Dlp in WG. By the extended theorem of Knaster and Tarski, 2(10)(b), it suffices to prove that Wun is Dlp-invariant and $\bigwedge$-decked in WG. Let us first consider Dlp-invariance. For any $w \in Wun$, any $h \in H$, and any $Y \subset X$, with y ranging over Y, we observe

$$\begin{array}{ll} & Dlp.w.h.(\bigwedge y :: y) = (\bigwedge y :: Dlp.w.h.y) \\ \equiv & \{\text{definition of } Dlp \text{ in (12)}\} \\ & w^l.(d.h).(\bigwedge y :: y) = (\bigwedge y :: w^l.(d.h).y) \\ \equiv & \{(14) \text{ and } w \in Wun\} \\ & true \end{array}$$

By (13), this proves that $Dlp.w \in Wun$. Therefore, Wun is Dlp-invariant.

We investigate $\bigwedge(U)$ for an arbitrary subset U of WG. For any $h \in H$ and $x \in X$ we have

$$\begin{array}{ll} & \bigwedge(U).h. \\ = & \{\text{theorem 2(1)}\} \\ & (\bigwedge w \in U :: w.h).x \\ = & \{C.X \text{ is } \bigwedge\text{-closed: 2(8)}\} \\ & (\bigwedge w \in U :: w.h.x) \end{array}$$

This proves that

$$\bigwedge(U).h.x = (\bigwedge w \in U :: w.h.x) \tag{15}$$

Now we can prove that Wun is $\bigwedge$-decked, cf. 2(7). Let L be a linear subset of Wun. For any $h \in H$ and any $Y \subset X$, with y ranging over Y and w over L, we have

$$\begin{array}{ll} & \bigwedge(L).h.(\bigwedge y :: y) = (\bigwedge y :: \bigwedge(L).h.y) \\ \equiv & \{(15)\} \\ & (\bigwedge w :: w.h.(\bigwedge y :: y)) = (\bigwedge y :: w.h.y)) \\ \equiv & \{(13) \text{ and interchange of quantifications}\} \\ & true \end{array}$$

By (13), this proves that $\bigwedge(L) \in Wun$. Therefore, Wun is $\bigwedge$-decked in WG. So, by 2(10)(b), we have $wb \in Wun$. Since $wlp = wb^l$, this proves that for any $p \in B[H]$ and any $Y \subset X$, we have the healthiness rule

$$wlp.p.(\bigwedge y \in Y :: y) = (\bigwedge y \in Y :: wlp.p.y) \tag{16}$$

3.5. The Termination Rule

In this subsection we prove that the triple $(B[H], wp, wlp)$ satisfies termination rule (1). This result will not be needed elsewhere.

Recall that, by Section 3.3, the interpretation $wp: B[H] \to C.X$ is wa^τ where wa is the smallest fixed point in WG of the monotone function $Dp: WG \to WG$. In view of (1) we define the subset Wt of WG by

$$w \in Wt \equiv (\forall h \in H, x \in X :: w.h.x = w.h.tt \wedge wlp.h.x) \tag{17}$$

Just as in the previous proof, we claim that (17) can be extended to

$$(\forall w \in Wt, p \in B[H], x \in X :: w^\tau.p.x = w^\tau.p.tt \wedge wlp.p.x) \tag{18}$$

To prove (18), let $w \in Wt$ be given. Define the subset K of $B[H]$ by

$$p \in K \equiv (\forall x \in X :: w^\tau.p.x = w^\tau.p.tt \wedge wlp.p.x) \tag{19}$$

Since B satisfies the termination rule (1), it follows from (11) that B is contained in K. As $w \in Wt$, it follows from (17) that H is contained in K. For elements p, $q \in K$ and $x \in X$, we observe

$$\begin{array}{ll} & w^\tau.(p \,[\!]\, q).x \;=\; w^\tau.(p \,[\!]\, q).tt \wedge wlp.(p \,[\!]\, q).x \\ \equiv & \{w^\tau \text{ and } wlp \text{ are homomorphisms}\} \\ & w^\tau.p.x \wedge w^\tau.q.x \;=\; w^\tau.p.tt \wedge w^\tau.q.tt \wedge wlp.p.x \wedge wlp.q.x \\ \equiv & \{p \text{ and } q \text{ are in } K, (19)\} \\ & true \end{array}$$

and

$$
\begin{array}{ll}
& w^\tau.(p;q).x \;=\; w^\tau.(p;q).tt \wedge wlp.(p;q).x \\
\equiv & \{w^\tau \text{ and } wlp \text{ are homomorphisms}\} \\
& w^\tau.p.(w^\tau.q.x) \;=\; w^\tau.p.(w^\tau.q.tt) \wedge wlp.p.(wlp.q.x) \\
\equiv & \{p \in K, (19), \text{twice}\} \\
& w^\tau.p.tt \wedge wlp.p.(w^\tau.q.x) \\
& \quad = w^\tau.p.tt \wedge wlp.p.(w^\tau.q.tt) \wedge wlp.p.(wlp.q.x) \\
\Leftarrow & \{\text{calculus and } wlp.p \text{ is conjunctive}\} \\
& w^\tau.q.x \;=\; w^\tau.q.tt \wedge wlp.q.x \\
\equiv & \{q \in K\} \\
& true
\end{array}
$$

This shows that $p \,[\!]\, q \in K$ and $p; q \in K$. Therefore, K is a subalgebra of $B[H]$ that contains B and H, so that $K = B[H]$ by Theorem (3). By (19), this proves (18).

We proceed to prove that $wa \in Wt$ where wa is the smallest fixpoint of Dp in WG. By the extended theorem of Knaster and Tarski 2(10)(a), it is sufficient to prove that Wt is Dp-invariant and $\bigvee$-decked in WG. The Dp-invariance of Wt is proved by observing that any $w \in WG$ satisfies

$$
\begin{array}{ll}
& Dp.w \in Wt \\
\equiv & \{(17)\} \\
& (\forall h \in H, x \in X :: Dp.w.h.x = Dp.w.h.tt \wedge wlp.h.x) \\
\equiv & \{\text{definition of } Dp\text{: (12), and } wlp \text{ is an interpretation: (4)}\} \\
& (\forall h \in H, x \in X :: w^\tau.(d.h).x = w^\tau.(d.h).tt \wedge wlp.(d.h).x) \\
\Leftarrow & \{(18)\} \\
& w \in Wt
\end{array}
$$

It remains to prove that Wt is $\bigvee$-decked in WG. Let L be a linear subset of Wt. We have to prove that $\bigvee(L) \in Wt$. For $h \in X$ we observe

$$
\begin{array}{ll}
& \bigvee(L).h.x \\
= & \{(\text{Theorem } 2(1)\} \\
& (\bigvee w \in L :: w.h).x \\
= & \{L \text{ linear, } C.X \text{ is } \bigvee\text{-decked: } 2(9)\} \\
& (\bigvee w \in L :: w.h.x) \\
= & \{L \text{ is contained in } Wt, (17)\} \\
& (\bigvee w \in L :: w.h.tt \wedge wlp.h.x) \\
= & \{\text{distributivity } (0)\} \\
& (\bigvee w \in L :: w.h.tt) \wedge wlp.h.x \\
= & \{\text{by the same steps, backward}\} \\
& \bigvee(L).h.tt \wedge wlp.h.x
\end{array}
$$

By (17), this proves that $\bigvee(L)$ is an element of Wt. Therefore, Wt is $\bigvee$-decked in WG, cf. 2(7). By 2(10)(a), this proves that $wa \in Wt$. Since $wp = wa^\tau$, formula (18) implies that

$$(\forall p \in B[H], x \in X :: wp.p.x = wp.p.tt \wedge wlp.p.x) \tag{20}$$

i.e. $B[H]$ satisfies termination rule (1).

4. Upper Continuity and Finite Nondeterminacy

4.0. In this section we introduce upper continuity. We do so because it plays an important role in the induction theorem of program transformation for the

semantic function *wp*, cf. rule 5(8). The relevant theory is developed in Sections 4.1 and 4.2.

In Section 4.3, we compare upper continuity with the concept of or-continuity of 0(10). Section 4.4 is a presentation of the model of relational calculus. In Section 4.5, we show that in that model upper continuity is equivalent to finite nondeterminacy, and also to or-continuity.

4.1. Upper Continuity

A function $f: X \to X$ is called *upper continuous* if and only if it commutes with least upper bounds of nonempty linear subsets of X. The reason for this definition is purely technical. In fact, in Theorem 5(15) we prove a specific property of *wp*. The proof is based on the definition of *wp* as a smallest fixed point of the function $D: WG \to WG$ and it uses Theorem 2(10)(a), in which least upper bounds of linear sets play a crucial role. Therefore, we need a condition that certain functions commute with least upper bounds of nonempty linear sets.

The set of the upper continuous elements of $C.X$ is denoted by *Cup*. So, $f \in Cup$ if and only if $f \in C.X$ and for every nonempty linear subset L of X

$$f.\bigvee(L) = (\bigvee x \in L :: f.x) \tag{0}$$

One easily verifies that *Cup* is closed under composition. On the other hand, for any $f, g \in Cup$, and any nonempty linear subset L of X, we have

$$\begin{aligned}
&(f \,[\!]\, g).\bigvee(L)\\
=\ &\{1(13)\}\\
&f.\bigvee(L) \wedge g.\bigvee(L)\\
=\ &\{(0), \text{ let } x \text{ and } y \text{ range over } L\}\\
&(\bigvee x :: f.x) \wedge (\bigvee y :: g.y)\\
=\ &\{\text{distributive law 3(0) twice}\}\\
&(\bigvee x, y :: f.x \wedge g.y)\\
=\ &\{L \text{ linear, diagonalisation: 2(5)}\}\\
&(\bigvee x :: f.x \wedge g.x)\\
=\ &\{1(13)\}\\
&(\bigvee x :: (f \,[\!]\, g).x)
\end{aligned}$$

This proves that

Cup is a subalgebra of $C.X$ (1)

In Section 4.2, we shall need the result that

Cup is $\bigvee$-decked in $C.X$ (2)

i.e. closed under least upper bounds of linear subsets, cf. Section 2.1. The proof of (2) is an easy calculation and uses Theorem 2(9).

4.2. Upper Continuous Commands

We now shift attention to command algebra B. We define the subset *Bup* of B by

$$a \in Bup \ \equiv\ a \in B \ \wedge\ wp.a \in Cup \tag{3}$$

As *Cup* is a subalgebra of $C.X$, cf. (1), it is easy to see that

Bup is a subalgebra of B. (4)

We choose a subset *Hup* of H such that

$$(\forall h \in Hup :: d.h \in Bup[Hup]) \tag{5}$$

Here, $Bup[Hup]$ is the subalgebra of $B[H]$ generated by Bup and Hup. We could choose Hup to be empty. A more useful choice is the biggest subset Hup of H that satisfies (5). In fact, that extreme solution exists by the theorem of Knaster and Tarski. For our purposes, however, any solution of (5) will do.

We define the subset Wup of WG by

$$w \in Wup \;\equiv\; (\forall h \in Hup :: w.h \in Cup) \tag{6}$$

As is usual (by now), we first have to prove an extension of (6), viz.

$$(\forall w \in Wup, p \in Bup[Hup] :: w^\tau.p \in Cup) \tag{7}$$

This is proved by structural induction on p. For $w \in Wup$ and $p \in Bup$, we have

$$w^\tau.p = wp.p \in Cup$$

by (3) and 3(11). For $p \in Hup$, definition (6) gives $w^\tau.p \in Cup$. Since Cup is a subalgebra of $C.X$ by (1), the elements $p \in B[H]$ with $w^\tau.p \in Cup$ form a subalgebra of $B[H]$. This subalgebra contains Bup and Hup, and therefore also $Bup[Hup]$. This proves (7).

We use the theorem of Knaster and Tarski, cf. 2(10)(a), to prove that the smallest fixpoint wa of Dp in WG satisfies

$$wa \in Wup \tag{8}$$

In fact, Wup is Dp-invariant in Wg, since for any $w \in WG$ we have

$$\begin{array}{ll} & Dp.w \in Wup \\ \equiv & \{(6) \textit{ and } 3(12)\} \\ & (\forall h \in Hup :: w^\tau.(d.h) \in Cup) \\ \Leftarrow & \{(5)\} \\ & (\forall p \in Bup[Hup] :: w^\tau.p \in Cup) \\ \Leftarrow & \{(7)\} \\ & w \in Wup \end{array}$$

Wup is $\bigvee$-decked in WG, since for any linear subset U of Wup and any $h \in Hup$ we have

$$\begin{array}{ll} & \bigvee(U).h \\ = & \{2(1)\} \\ & (\bigvee w \in U :: w.h) \\ \in & \{(6), \text{ all } w.h \in Cup,\ U \text{ linear}, (2)\} \\ & Cup \end{array}$$

Thus we have proved that

$$Wup \text{ is } Dp\text{-invariant and } \bigvee\text{-decked in } WG \tag{9}$$

By 2(10)(a), this proves (8). Since $wp = wa^\tau$, it follows from (7) and (8) that

$$(\forall p \in Bup[Hup] :: wp.p \in Cup) \tag{10}$$

4.3. Upper Continuity and Or-Continuity

Upper continuity as considered in 4.1 can be compared with or-continuity used by Dijkstra and Scholten in [DiS90] chapter 6. A function $f: X \to X$ is said to

be *or-continuous* if and only if in lattice X every ascending (i.e. weakening) sequence $(x.i : i \geq 0)$ satisfies

$$f.(\bigvee i :: x.i) = (\bigvee i :: f.(x.i)) \tag{11}$$

Since every ascending sequence in X forms a nonempty linear subset of X, every upper continuous function f is or-continuous. Without additional assumptions, the converse implication cannot be proved, as is shown at the end of Section 4.5 below. For practical purposes, however, or-continuity and upper continuity are equivalent. This claim is also justified in Section 4.5.

4.4. Relational Calculus

The main example of the whole theory is the model of relational calculus. We present this model as an illustration for its conceptual simplicity. In programming practice, however, the paradigm of relational calculus seems to be less useful, since it tends to encourage operational reasoning.

Let St be a set, to be called the state space. The elements of St are called states. Let ∞ be a symbol not in St, and let $Stt = St \cup \{\infty\}$. An operation is represented by a subset of the cartesian product $St \times Stt$. Conceptually, the operation represented by subset b is such that

$(r, t) \in b \equiv$ the operation may transform state r into t
$(r, \infty) \in b \equiv$ the operation starting r need not terminate

Let B be the set of subsets b of $St \times Stt$. B is converted into a command algebra by defining for any $b, c \in B$ and any $(r, t) \in St \times Stt$

$$\begin{aligned} &b \| c = b \cup c \\ &(r, t) \in b; c \equiv (t = \infty \wedge (r, t) \in b) \vee (\exists s \in St :: (r, s) \in b \wedge (s, t) \in c) \end{aligned} \tag{12}$$

The verification that B is a command algebra is left to the reader.

We take the lattice X to be the set of the boolean functions on St, cf. Section 1.3, ordered by formula 1(17). So, X is a complete lattice, cf. 1(18), and Section 2.1. We define functions $wp, wlp : B \to F.X$ by

$$\begin{aligned} &wp.b.x.r = (\forall t : (r, t) \in b : t \neq \infty \wedge x.t) \\ &wlp.b.x.r = (\forall t : (r, t) \in b \wedge t \neq \infty : x.t) \end{aligned} \tag{13}$$

Thus, $wp.b.x$ is the weakest precondition such that the operation of b is guaranteed to terminate in a state that satisfies x, and $wlp.b.x$ is the weakest condition such that execution of b does not terminate or terminates in a state that satisfies x. It is well-known and easy to prove that functions $wp.b$ and $wlp.b$ are conjunctive. Therefore, wp and wlp are functions

$$wp, wlp : B \to C.X$$

Actually, they are homomorphisms of command algebras. The verifications are standard, though nontrivial; see [Hes89b]. Similarly, we leave it to the reader to verify that wp and wlp satisfy the healthiness conditions 3(1) and 3(2).

4.5. Finite Nondeterminacy

In the model of relational calculus, both or-continuity and upper continuity are equal to finite nondeterminacy. In fact, an element $b \in B$ is said to be of *finite*

nondeterminacy if and only if

$$(\forall r \in St :: (r, \infty) \in b \vee (\mathbf{FIN}\ t :: (r, t) \in b)) \tag{14}$$

where $(\mathbf{FIN}\ t :: x.t)$ says that the number of elements t where $x.t$ holds, is finite. In this situation we have

(15) **Theorem.** The following conditions are equivalent:

(a) b is of finite nondeterminacy.

(b) $wp.b$ is upper continuous.

(c) $wp.b$ is or-continuous.

Proof. (a)⇒(b). Let L be a nonempty linear subset of X. For any state r with $(r, \infty) \in b$, we have

$$\begin{array}{ll} & wp.b.\bigvee(L).r \\ = & \{(13) \text{ and } (r, \infty) \in b\} \\ & \mathit{false} \\ = & \{\text{calculus, } (r, \infty) \in b \text{ and } (13)\} \\ & (\exists x \in L :: wp.b.x.r) \\ = & \{1(18)\} \\ & (\bigvee x \in L :: wp.b.x).r \end{array}$$

For any state r with $(\mathbf{FIN}\ t :: (r, t) \in b)$, we observe

$$\begin{array}{ll} & wp.b.\bigvee(L).r \\ = & \{(13)\} \\ & (\forall t : (r, t) \in b : t \neq \infty \wedge \bigvee(L).t) \\ = & \{1(18)\} \\ & (\forall t : (r, t) \in b : t \neq \infty \wedge (\exists x \in L :: x.t)) \\ = & \{\text{distributive law}\} \\ & (\forall t : (r, t) \in b : (\exists x \in L :: t \neq \infty \wedge x.t)) \\ = & \{(\mathbf{FIN}\ t :: (r, t) \in b) \text{ and } L \text{ is linear and nonempty}\} \\ & (\exists x \in L :: (\forall t : (r, t) \in b : t \neq \infty \wedge x.t)) \\ = & \{(13)\} \\ & (\exists x \in L :: wp.b.x.r) \\ = & \{1(18)\} \\ & (\bigvee x \in L :: wp.b.x).r \end{array}$$

This proves that $wp.b.\bigvee(L) = (\bigvee x \in L :: wp.b.x)$, so that $wp.b$ is upper continuous by (0). The implication (b)⇒(c) has been mentioned in Section 4.3.

The implication (c)⇒(a) is proved by contraposition. Let b be not of finite nondeterminacy. So, there is a state r with $(r, \infty) \notin b$, and an infinite sequence of states $(u.i : i \geq 0)$ in St such that $u.i \neq u.j$ whenever $i \neq j$, and that $(r, u.i) \in b$ for all i. Let the predicates $x.k$ be defined by

$$x.k.t = (\forall i : i > k : t \neq u.i) \tag{16}$$

One verifies that the sequence of predicates $x.k$ is weakening. The least upper bound of the sequence is the predicate tt, since for any state t it holds that

$$\begin{array}{ll} & (\bigvee k :: x.k).t \\ = & \{1(18), (16)\} \\ & (\exists k :: (\forall i : i > k : t \neq u.i)) \\ = & \{\text{all } u.i \text{ are different}\} \\ & \mathit{true} \end{array}$$

Since $(r, \infty) \notin b$, this implies $wp.b.(\bigvee k :: x.k).r = true$. On the other hand, we have

$$
\begin{array}{ll}
 & (\bigvee k :: wp.b.(x.k)).r \\
= & \{1(18)\} \\
 & (\exists k :: wp.b.(x.k).r) \\
\Rightarrow & \{(13)\} \\
 & (\exists k :: (\forall t : (r, t) \in b : x.k.t)) \\
\Rightarrow & \{\text{all } (r, u.j) \in b\} \\
 & (\exists k :: (\forall j :: x.k.(u.j))) \\
= & \{(16)\} \\
 & (\exists k :: (\forall j :: (\forall i : i > k : u.j \neq u.i))) \\
\Rightarrow & \{\text{calculus}\} \\
 & (\exists k :: (\forall j :: (\forall i : i > k : j \neq i))) \\
= & \{\text{calculus}\} \\
 & false
\end{array}
$$

This shows that $wp.b$ is not or-continuous, cf. (11). □

Example. In this example, we show that or-continuous functions $X \to X$ need not be upper continuous. Let the set St be uncountable. As above, X is the set of boolean functions on St. For $x \in X$, let $|x|$ be the subset of St where x holds. Let the function $f : X \to X$ be given by

$$f.x.s = (|x| \text{ is uncountable})$$

for all $x \in X$ and all $s \in St$. One can verify that function f is or-continuous. On the other hand, if set St can be equipped with a well-ordering, then f is not upper continuous, since the first uncountable ordinal is the union of the countable ordinals, cf. [TaZ71]. The details of that statement are left to the interested reader. Greg Nelson informed me that this is problem B of Chapter 2 of [Kel55].

5. Transformation Rules for Recursion

5.0. Program transformation is the act of replacing one command by another command that is semantically equivalent to the first command. So, for the purpose of program transformation, we have to investigate the relation $\cong$ of semantic equivalence on $B[H]$, cf. 0(1). We need not know relation $\cong$ completely. It suffices to provide methods for proving semantic equivalence between specific commands.

In the remainder of this section, we give an overview of the arguments that could have lead us in the investigation. The formal development in the Sections 5.1 up to 5.6 is independent and much slower.

> *Rule* 1. *If you have a conjecture, set out to prove it and to refute it. Inspect the proof carefully to prepare a list of non-trivial lemmas* (*proof-analysis*); *find counterexamples both to the conjecture* (*global counterexamples*) *and to suspect lemmas* (*local counterexamples*) (Lambda, in [Lak76], p. 50).

The starting point is Hoare's recursion rule, cf. [Hoa71] and [Apt81] p. 444. This rule states that in order to prove conditional correctness of a procedure h with respect to a specification S, it suffices to prove that its body $d.h$ satisfies S under

assumption that h satisfies S. More precisely, if we write $x\{s\}y$ to denote $[x \Rightarrow wlp.s.y]$, cf. 0(6), the proof rule is summarised in

$$\frac{x\{h\}y \vdash x\{d.h\}y}{x\{h\}y} \tag{0}$$

The horizontal line separates the premiss of the rule from the conclusion. Notice that if the symbol "$\vdash$" were replaced by a classical implication, the rule would always imply $x\{h\}y$ by a reductio ad absurdum. This would be absurd. In fact, the symbol "$\vdash$" stands for constructive derivability. For other versions of (0), we refer to [Man74] p. 394 and [Hes89a] Section 4.3.

Rule (0) may suggest the following conjectural rule for procedures h and k:

$$\frac{h \cong k \vdash d.h \cong d.k}{h \cong k} \ ? \tag{1}$$

Notice that rule (1) includes the termination behaviour, whereas rule (0) is postulated for conditional correctness only.

For practical purposes, rule (1), if valid, is too limited. It only allows comparison of one pair of procedure names without context. We often need comparison of many pairs of composite commands. Recall that function d has been extended to a homomorphism $d^*: B[H] \to B[H]$. So, for any binary relation "$\smile$" on $B[H]$, we would like to have

$$\frac{(\forall s, t: s \smile t: s \cong t) \vdash (\forall s, t: s \smile t: d^*.s \cong d^*.t)}{(\forall s, t: s \smile t: s \cong t)} \ ? \tag{2}$$

In order to eliminate the symbol "$\vdash$" in (1) and (2), we introduce congruences. A *congruence* on a command algebra A is defined to be an equivalence relation "$\sim$" on A such that for all $p, q, s, t \in A$ it holds

$$p \sim q \wedge s \sim t \Rightarrow (p; s \sim q; t) \wedge (p \,[\!]\, s \sim q \,[\!]\, t) \tag{3}$$

For example, it is easy to prove that "$\cong$" is a congruence on B.

In the rest of this section, we let "$\smile$" denote an arbitrary binary relation on $B[H]$ as in (2). We define "$\overset{*}{\smile}$" to be the smallest congruence on $B[H]$ that contains "$\smile$" (this congruence is easily shown to exist and to be unique). Notice that, as "$\cong$" is a congruence on $B[H]$ and "$\overset{*}{\smile}$" is the smallest congruence that contains "$\smile$", we have

$$(\forall s, t: s \smile t: s \cong t) \ \equiv \ (\forall s, t: s \overset{*}{\smile} t: s \cong t). \tag{4}$$

Inspired by (2), we might conjecture

$$\frac{(\forall s, t: s \smile t: d^*.s \overset{*}{\smile} d^*.t)}{(\forall s, t: s \overset{*}{\smile} t: s \cong t)} ? \tag{5}$$

This conjecture is obviously false. For, if relation "$\smile$" is identically true, then "$\overset{*}{\smile}$" is identically true, but "$\cong$" need not be identically true.

Conjecture (5) is an induction principle without a base case. We add a base case by substituting *abort* for all procedure names, where $abort \in B$ is a command that never terminates. So, let $da^*: B[H] \to B$ be the homomorphism of command algebras such that $da^*.s$ is obtained from s by substituting *abort* for every procedure name h in expression s. Now conjecture (5) is replaced by

$$\frac{(\forall s, t: s \smile t: \ d^*.s \overset{*}{\smile} d^*.t \ \wedge \ da^*.s \cong da^*.t)}{(\forall s, t: s \overset{*}{\smile} t: s \cong t)} \ ? \tag{6}$$

The rule is still false (see Section 5.6 below). Yet it is "almost" true. In the deterministic case, and in the case of finite nondeterminacy, rule (6) is valid and known as the rule of computational induction or the induction rule of De Bakker and Scott (see [Bak80] 7.16 and [Man74] p. 397). For conditional correctness, we prove in Section 5.4 the rule

$$\frac{(\forall s, t : s \smile t :\ d^*.s \mathrel{\dot{\smile}} d^*.t \ \wedge\ wlp.(da^*.s) = wlp.(da^*.t))}{(\forall s, t : s \mathrel{\dot{\smile}} t : wlp.s = wlp.t)} \tag{7}$$

The rule for total correctness is also proved in Section 5.4. It is the rule obtained from (6) by adding a third conjunct in the premiss:

$$\frac{(\forall s, t : s \smile t :\ d^*.s \mathrel{\dot{\smile}} d^*.t \ \wedge\ da^*.s \cong da^*.t \ \wedge\ s, t \in Lia)}{(\forall s, t : s \mathrel{\dot{\smile}} t : s \cong t)} \tag{8}$$

Here *Lia* is a certain subset of $B[H]$, the definition of which is given in Section 5.4. In order to show that *Lia* has sufficiently many elements, we proceed as follows. Let *Bup* be the set of the commands $b \in B$ that are of finite nondeterminacy, cf. 4(3). We chose a subset *Hup* of H such that $d.h. \in Bup[Hup]$ for all $h \in Hup$, cf. 4(5). In Section 5.5, it is proved that

$$\begin{aligned} &B \cup H \cup Bup[Hup] \subset Lia \\ &\wedge (\forall p, q \in Lia :: p \,\|\, q \in Lia) \\ &\wedge (\forall p, q \in Lia : p \in Bup[Hup] \vee q \in B : p;\ q \in Lia) \end{aligned} \tag{9}$$

In particular, if all elements of B are of finite nondeterminacy, then $Bup = B$ and we can take $Hup = H$, so that $Lia = B[H]$.

One might be tempted to simplify rules (7) and (8) by imposing the condition that relation "$\smile$" is a congruence, so that $(\mathrel{\dot{\smile}}) = (\smile)$. This simplification is undesirable, however, since it would lead to much heavier proof obligations in the applications of the rules.

In Section 6.1, we present the final version of the result, in a form convenient for the applications and slightly stronger than rule (8).

During the search from conjecture (1) to rule (8), I was not hindered by awareness of computational induction. The two extra conjuncts that appear in the premiss between (5) and (8) are both proof-generated, in the sense that I did not expect them before the proof without them failed, cf. [Lak76].

5.1. Congruences

Recall from Section 5.0 that a binary relation $\sim$ on a command algebra A is called a *congruence* if and only if it is an equivalence relation and for all $p, q, r, s \in A$

$$p \sim q \wedge r \sim s \Rightarrow (p;\ r \sim q;\ s) \wedge (p \,\|\, r \sim q \,\|\, s) \tag{10}$$

We often need binary relations denoted by variable symbols or expressions. Therefore, it is convenient to identify a binary relation $\smile$ on A with a subset $(\smile)$ of the cartesian product $A \times A$ via the rule

$$p \smile q \ \equiv\ \langle p, q \rangle \in (\smile)$$

Conversely, a subset of $A \times A$ may be treated as a binary relation. If $w : A \to A'$ is a homomorphism of command algebras, the *equaliser* $Eq.w$ of w is defined to

be the binary relation on A given by

$$\langle p, q\rangle \in Eq.w \;\equiv\; w.p = w.q \tag{11}$$

An easy verification shows that $Eq.w$ is a congruence on A.

It is easy to see that the intersection of a set of congruences is a congruence. Therefore, semantic equivalence (cf. 0(1)) is a congruence, since it is equal to the intersection $Eq.wp \cap Eq.wlp$.

If E is a binary relation on a command algebra A, there is one smallest congruence $cg.E$ on E that contains E. The easiest proof is to define $cg.E$ as the intersection of all congruences that contain E, and to verify that $cg.E$ is a congruence that contains E. As it is contained in all congruences that contain E, it is the smallest one. The congruence $cg.E$ is called the congruence *generated* by E. If E is given as the infix operator "$\smile$", we use the infix operator "$\overset{*}{\smile}$" to denote $cg.E$.

5.2. Relations Strong for *wlp* and *wp*

We define a binary relation E on $B[H]$ to be *d-invariant* if and only if

$$(\forall \langle p, q\rangle \in E :: \langle d^*.p, d^*.q\rangle \in cg.E). \tag{12}$$

For a binary relation E on $B[H]$ we want results like $E \subset Eq.wp$ and $E \subset Eq.wlp$. Now, $wlp = wb^l$ and $wp = wa^\tau$ where wb is the biggest fixed point of Dlp in Wun and wa is the smallest fixed point of Dp in Wup, cf. Sections 3.4 and 4.2. Therefore, we introduce the subsets $Wp.E$ and $Wlp.E$ of WG by

$$w \in Wp.E \;\equiv\; w \in Wup \;\wedge\; E \subset Eq.(w^\tau) \tag{13}$$

$$w \in Wlp.E \;\equiv\; w \in Wun \;\wedge\; E \subset Eq.(w^l) \tag{14}$$

Relation E is defined to be *wp-strong* if and only if E is d-invariant and $Wp.E$ is $\bigvee$-decked in WG. Similarly, E is defined to be *wlp-strong* if and only if E is d-invariant and $Wlp.E$ is $\bigwedge$-decked in WG. Relation E is called *strong* if and only if it is both *wlp*-strong and *wp*-strong.

(15) **Theorem.** (a) If E is *wp*-strong then $E \subset Eq.wp$.

(b) If E is *wlp*-strong then $E \subset Eq.wlp$.

(c) If E is strong then $p \cong q$ for all pairs $\langle p, q\rangle \in E$.

Proof. (a) Since $wp = wa^\tau$, it suffices to prove that $wa \in Wp.E$, cf. (13). By the extended theorem of Knaster and Tarski, cf. 2(10)(a), it suffices to prove that $Wp.E$ is Dp-invariant and $\bigvee$-decked in WG. Therefore, it remains to prove that $Wp.E$ is Dp-invariant. This is proved by observing that for any $w \in Wup$

$$\begin{array}{ll} & Dp.w \in Wp.E \\ \equiv & \{(13);\ Wup \text{ is } Dp\text{-invariant, cf. 4(9); (11)}\} \\ & (\forall \langle p, q\rangle \in E :: (Dp.w)^\tau.p = (Dp.w)^\tau.q) \\ \equiv & \{\text{formula (16) below}\} \qquad (*) \\ & (\forall \langle p, q\rangle \in E :: w^\tau.(d^*.p) = w^\tau.(d^*.q)) \\ \Leftarrow & \{(12);\ E \text{ is } d\text{-invariant; (11)}\} \\ & cg.E \subset Eq.w^\tau \\ \Leftarrow & \{Eq.w^\tau \text{ is a congruence; definition } cg.E\} \\ & E \subset Eq.w^\tau \end{array}$$

$$\Leftarrow \quad \{(13)\}$$
$$w \in Wp.E$$

It remains to verify the step with (*):

$$(Dp.w)^\tau = w^\tau \circ d^* \in B[H] \to C.X \tag{16}$$
$$\equiv \quad \{\text{theorem 3(3) and definition } d^* \text{ in 3.1}\}$$
$$(\forall a \in B :: (Dp.w)^\tau.a = w^\tau.a) \ \wedge \ (\forall h \in H :: (Dp.w)^\tau.h = w^\tau.(d.h))$$
$$\equiv \quad \{\text{definitions of } w^\tau \text{ and } Dp \text{ in 3.3}\}$$
$$true.$$

The proof of (b) is completely analogous. Part (c) follows from (a) and (b) together with the definition of "$\cong$" in 0(1). □

In Section 6, it will turn out to be very useful that there is a biggest strong relation:

(17) **Theorem.** Let Es be defined as the union of all strong relations on $B[H]$.
(a) Relation Es is strong and, therefore, the biggest strong relation on $B[H]$.
(b) Relation Es is a congruence.

Proof. (a) Relation Es is d-invariant, since, for any pair $\langle p, q\rangle \in Es$, there is a strong relation E with $\langle p, q\rangle \in E$ and hence $\langle d^*.p, d^*.q\rangle \in cg.E$, and hence $\langle d^*.p, d^*.q\rangle \in cg.Es$. If L is a linear subset of $Wp.Es$, then $L \subset Wp.E$ for every strong relation E, and hence $\bigvee(L) \in Wp.E$ for every strong relation E, so that $\bigvee(L) \in Wp.Es$. This proves that $Wp.Es$ is $\bigvee$-decked in WG. The proof that $Wlp.Es$ is $\bigwedge$-decked is analogous. This proves that Es is strong.

(b) Since d^* is a homomorphism, one can easily verify that $cg.Es$ is d-invariant. On the other hand, it follows from definitions (13) and (14) that $Wp.(cg.Es) = Wp.Es$ and $Wlp.(cg.Es) = Wlp.Es$. Therefore, $cg.Es$ is a strong congruence. By part (a), it follows that $cg.Es \subset Es$. Therefore, $cg.Es = Es$. □

5.3. The Abortive Congruence

The complete set WG has the smallest element wff and the biggest element wtt given by

$$wff.h.x = ff, \quad wtt.h.x = tt \tag{18}$$

Let $wpa, wlpa : B[H] \to C.X$ be the homomorphisms given by

$$wpa = wff^\tau, \quad wlpa = wtt^l \tag{19}$$

Then we have

$$(\forall a \in B :: wpa.a = wp.a \wedge wlpa.a = wlp.a) \tag{20}$$
$$\wedge (\forall h \in H, x \in X :: wpa.h.x = ff \wedge wlpa.h.x = tt)$$

The intersection of the equalisers of wpa and $wlpa$ is called the *abortive* congruence. The term abortive is justified as follows. Let us assume that command algebra B contains an element *abort* with for all $x \in X$

$$wp.abort.x = ff, \quad wlp.abort.x = tt$$

The *abortive* declaration $da : H \to B[H]$ is defined by

$$(\forall h \in H :: da.h = abort)$$

As usual, this declaration is extended to a homomorphism $da^*: B[H] \to B[H]$ that is the identity on B. Notice that by structural induction

$$(\forall p \in B[H] :: da^*.p \in B)$$

It is not difficult to verify that $wp \circ da^* = wpa$ and $wlp \circ da^* = wlpa$. It follows that, for any $p, q \in B[H]$, we have

$$\langle p, q \rangle \in Eq.wpa \cap Eq.Wlpa \;\equiv\; da^*.p \cong da^*.q \tag{21}$$

5.4. Towards More Concrete Conditions

In this section, we analyse the strength conditions of theorem (15) in order to get more concrete conditions. In the case of *wlp*-strength, we obtain a necessary and sufficient condition in terms of function *wlpa*. In the case of *wp*-strength, we get a sufficient condition that requires further investigation. We start with *wlp*. Here, the crucial point is the following seemingly innocent result.

(22) **Lemma.** For $p \in B[H]$ and a nonempty linear subset L of *Wun*, we have

$$\bigwedge (L)^l.p = (\bigwedge w \in L :: w^l.p) \tag{23}$$

Proof. We use structural induction on p. We fix L and let dummies w (and v) range over the set L. For $p \in B$ we have

$$\begin{array}{ll} & (\bigwedge w :: w^l.p) \\ = & \{p \in B\} \quad (\bigwedge w :: wlp.p) \\ = & \{L \text{ is nonempty}\} \quad wlp.p \\ = & \{p \in B\} \quad \bigwedge (L)^l.p \end{array}$$

This proves (23) for all $p \in B$. It follows from Theorem 2(1) that formula (23) holds for all $p \in H$. The settles the base cases of the induction.

The induction step is taken as follows. Let $p, q \in B[H]$ satisfy formula (23). The command $p \,[\!]\, q$ satisfies (23), since

$$\begin{array}{ll} & \bigwedge (L)^l.(p \,[\!]\, q) = (\bigwedge w :: w^l.(p \,[\!]\, q)) \\ \equiv & \{\bigwedge (L)^l \text{ and all } w^l \text{ are homomorphisms}\} \\ & \bigwedge (L)^l.p \wedge \bigwedge (L)^l.q = (\bigwedge w :: w^l.p \wedge w^l.q) \\ \equiv & \{\text{calculus}\} \\ & \bigwedge (L)^l.p \wedge \bigwedge (L)^l.q = (\bigwedge w :: w^l.p) \wedge (\bigwedge w :: w^l.q) \\ \equiv & \{\text{induction hypothesis (23) for both } p \text{ and } q\} \\ & true. \end{array}$$

The command $p; q$ satisfies (23), since for any $x \in X$ it holds

$$\begin{array}{ll} & \bigwedge (L)^l.(p; q).x \\ = & \{\bigwedge (L)^l \text{ is a homomorphism and 1(13)}\} \\ & \bigwedge (L)^l.p.(\bigwedge (L)^l.q.x) \\ = & \{\text{induction hypothesis (23) for } p \text{ and } q\} \\ & (\bigwedge v :: v^l.p).((\bigwedge w :: w^l.q).x) \\ = & \{C.X \text{ is } \bigwedge\text{-closed: 2(8), twice}\} \\ & (\bigwedge v :: v^l.p.(\bigwedge w :: w^l.q.x)) \\ = & \{\text{all } v \in Wun \text{ and formula 3(14)}\} \\ & (\bigwedge v, w :: v^l.p.(w^l.q.x)) \end{array}$$

$= \quad$ {diagonalisation: 2(5) and linearity of L}
$(\bigwedge w :: w^l.p.(w^l.q.x))$
$= \quad$ {all w^l are homomorphisms and 1(13)}
$(\bigwedge w :: w^l.(p; q).x)$
$= \quad$ {$C.X$ is $\bigwedge$-closed: 2(8)}
$(\bigwedge w :: w^l.(p; q)).x$

This concludes the induction step. □

Now we can prove the following result, which together with Theorem (15) proves rule (7).

(24) **Theorem.** Let E be a binary relation on $B[H]$.

(a) *Wlp.E* is $\bigwedge$-decked in *WG* if and only if $E \subset Eq.wlpa$.

(b) E is *wlp*-strong if and only if E be d-invariant and $E \subset Eq.wlpa$.

Proof. (a) We first observe that for any *nonempty* linear subset L of *Wun*

$\bigvee (L) \in Wlp.E$
$\equiv \quad$ {(14); *Wun* is $\bigwedge$-closed in *WG*}
$(\forall \langle p, q\rangle \in E :: \bigwedge (L)^l.p = \bigwedge (L)^l.q)$
$\equiv \quad$ {lemma (22), L nonempty linear}
$(\forall \langle p, q\rangle \in E :: (\bigwedge w \in L :: w^l.p) = (\bigwedge w \in L :: w^l.q))$
$\Leftarrow \quad$ {calculus}
$(\forall w \in L, \langle p, q\rangle \in E :: w^l.p = w^l.q)$
$\Leftarrow \quad$ (14)}
$L \subset Wlp.E$

Wlp.E is $\bigwedge$-decked if and only if $\bigwedge (L) \in Wlp.E$ for every linear subset L. By the above calculation, it remains to consider the empty linear subset $\varnothing$. Since $\bigwedge (\varnothing)$ is the biggest element *wtt* of *WG*, we get the condition $wtt \in Wlp.E$. Using definitions (18) and 3(13), one can prove that $wtt \in Wun$, so that, by definitions (14) and (19), we have that *Wlp.E* is $\bigwedge$-decked if and only if $E \subset Eq.wlpa$.

Part (b) follows from (a) and the definition of *wlp-strong*. □

In the case of *wp*, the analogue of lemma (22) fails. Therefore, we impose the analogue of (22) as a condition on the commands in relation E.

We define a command $p \in B[H]$ to be *linearly approximated* if and only if for every nonempty linear subset L of *Wup*

$$\bigvee (L)^\tau.p = (\bigvee w \in L :: w^\tau.p) \tag{25}$$

We write *Lia* to denote the set of the linearly approximated elements of $B[H]$. In a way that is completely analogous to the proof of Theorem (24), one can prove

(26) **Theorem.** Let E be a binary relation on $B[H]$ such that $p, q \in Lia$ for all pairs $\langle p, q\rangle \in E$

(a) *Wp.E* is $\bigvee$-decked in *WG* if and only if $E \subset Eq.wpa$.

(b) E is *wp*-strong if and only if E is d-invariant and $E \subset Eq.wpa$.

5.5. Linear Approximation

For Theorem (26) to be useful, we have to ensure that the subset *Lia* of $B[H]$ is sufficiently big. By arguments completely analogous to those used to start the structural induction in the proof of Lemma (22), one can easily prove

$$B \cup H \subset Lia \tag{27}$$

We proceed as if we are using structural induction to prove that $Lia = B[H]$. So, we consider $p, q \in Lia$. For any nonempty linear subset L of Wup and any $x \in X$, we observe

$$
\begin{array}{ll}
 & \bigvee (L)^\tau.(p \,\|\, q).x \\
= & \{\bigvee (L)^\tau \text{ is a homomorphism and } 1(13)\} \\
 & \bigvee (L)^\tau.p.x \wedge \bigvee (L)^\tau.q.x \\
= & \{p, q \in Lia, (25), \text{ let } v \text{ and } w \text{ range over } L\} \\
 & (\bigvee v :: v^\tau.p.x) \wedge (\bigvee w :: w^\tau.q.x) \\
= & \{X \text{ satisfies the distributive law } 3(0)\} \\
 & (\bigvee v, w :: v^\tau.p.x \wedge w^\tau.q.x) \\
= & \{\text{diagonalisation: } 2(5), L \text{ linear}\} \\
 & (\bigvee w :: w^\tau.p.x \wedge w^\tau.q.x) \\
= & \{w^\tau \text{ homomorphism, } 1(13)\} \\
 & (\bigvee w :: w^\tau.(p \,\|\, q).x)
\end{array}
$$

By (25), this proves

$$(\forall p, q \in Lia :: p \,\|\, q \in Lia) \tag{28}$$

The composition of two elements of Lia need not be element of Lia. In the comment of the next calculation we show which condition we need, to prove that the composition of two elements is an element of Lia. Let $p, q \in B[H]$, and let L be a nonempty linear subset of Wup, and let $x \in X$. We observe

$$
\begin{array}{ll}
 & \bigvee (L)^\tau.(p; q).x \\
= & \{\bigvee (L)^\tau \text{ is a homomorphism and } 1(13)\} \\
 & \bigvee (L)^\tau.p.(\bigvee (L)^\tau.q.x) \\
= & \{p, q \in Lia, (25), \text{ let } v \text{ and } w \text{ range over } L\} \\
 & (\bigvee v :: v^\tau.p.(\bigvee w :: w^\tau.q.x)) \\
= & \{\text{we need: } v^\tau.p \in Cup \text{ and } 4(0), \text{ or } w^\tau.q \text{ constant}\} \qquad (*) \\
 & (\bigvee v, w :: v^\tau.p.(w^\tau.q.x)) \\
= & \{L \text{ linear, diagonalization: } 2(5)\} \\
 & (\bigvee w :: w^\tau.p.(w^\tau.q.x)) \\
= & \{w^\tau \text{ homomorphism, } 1(13)\} \\
 & (\bigvee w :: w^\tau.(p; q).x)
\end{array}
$$

So the gap in the argument is the step in the middle, indicated by (*). Since $v \in L \subset Wup$, it follows from 4(7), that the first alternative is implied by $p \in Bup[Hup]$. The second alternative is implied by $q \in B$. This proves

$$(\forall p, q \in Lia : p \in Bup[Hup] \vee q \in B : p; q \in Lia) \tag{29}$$

By structural induction with (27), (28), (29), we get

$$Bup[Hup] \subset Lia \tag{30}$$

It follows from the formulae (27) up to (30) that

$$(\forall p, q, r : p \in Bup[Hup] \wedge q \in B \cup H \wedge r \in B : p; q; r \in Lia) \tag{31}$$

5.6. An Example with Unbounded Nondeterminacy

In this section, we give a global counterexample, cf. [Lak76], to conjecture (6) of Section 5.0. The example shows that the condition $p, q \in Lia$ in theorem (26) cannot be left out. It also exhibits elements $p \in B$ and $h \in Hup$ such that $p; h \notin Lia$. Thus the delicate arguments in Section 5.5 are shown to be essential.

We assume that command algebra B contains all commands that we need. Moreover, we assume that B is separated in the sense that semantically equivalent commands in B are equal. We work with a state space consisting of one integer variable i. A typical element of B that is not in *Bup*, is the unboundedly nondeterminate command

$p =$ "choose $i \geq 0$ arbitrarily"

It is characterised by

$$wp.p.x = (\forall i : i \geq 0 : x) \qquad (32)$$
$$wlp.p.x = (\forall i : i \geq 0 : x)$$

Since $wp.p.tt = tt$, command p is guaranteed to terminate. Let procedure h be declared by

$$d.h = \textbf{if } i \leq 0 \rightarrow skip \;[\!]\; i > 0 \rightarrow i := i-1; h; i := i+1 \textbf{ fi}.$$

In the present calculus, we use the translation

$$d.h \;=\; ?(i \leq 0) [\!] ?(i > 0); i := i-1; h; i := i+1 \qquad (33)$$

Recall that operator ";" has higher priority than "[]". As is shown in [Hes88b] Section 9, procedure h is semantically equivalent to *skip*. The proof only uses formula 3(10).

Since p is guaranteed to terminate and h is equivalent to *skip*, we have

$$wp.(p; h) \;\neq\; wp.(p; h [\!] abort) \qquad (34)$$

We consider the binary relation $\smile$ that only contains the pair

$$p; h \;\smile\; p; h [\!] abort \qquad (35)$$

By (34) and Theorem (15)(a), relation $\smile$ is not *wp*-strong.

We verify that relation $\smile$ is d-invariant:

$$d^*.(p; h) \;\overset{*}{\smile}\; d^*.(p; h [\!] abort) \qquad (36)$$

This is proved in

$$\begin{array}{ll}
 & d^*.(p; h) \\
= & \{(33), \text{distributive law in } B[H]\} \\
 & p; ?(i \leq 0) [\!] p; ?(i > 0); i := i-1; h; i := i+1 \\
= & \{\text{formula (37) below}\} \qquad (*) \\
 & p; ?(i \leq 0) [\!] p; h; i := i+1 \\
\overset{*}{\smile} & \{\overset{*}{\smile} \text{ is a congruence that contains the pair of (35)}\} \\
 & p; ?(i \leq 0) [\!] (p; h [\!] abort); i := i+1 \\
= & \{\text{distribution, and } (abort; i := i+1) = abort \text{ in } B\} \\
 & p; ?(i \leq 0) [\!] p; h; i := i+1 [\!] abort \\
= & \{\text{equality derived above: } d^*.(p; h) = p; ?(i \leq 0) [\!] p; h; i := i+1\} \\
 & d^*.(p; h) [\!] abort \\
= & \{d^*.abort = abort\} \\
 & d^*.(p; h [\!] abort)
\end{array}$$

It remains to verify step (*). For any predicate x, we observe that

$$
\begin{array}{ll}
 & wp.(p;\ ?(i>0);\ i:=i-1).x \\
= & \quad \{\text{assignment, and guard, } 1(20)\} \\
 & wp.p.(\neg(i>0) \vee x^i_{i-1}) \\
= & \quad \{(32)\} \\
 & (\forall i: i \geq 0: \neg(i>0) \vee x^i_{i-1}) \\
= & \quad \{\text{trading; renaming } j:=i-1\} \\
 & (\forall j: j \geq 0: x^i_j) \\
= & \quad \{\text{renaming } i:=j;\ (32)\} \\
 & wp.p.x
\end{array}
$$

The same calculation goes through if *wp* is replaced by *wlp*. This shows that

$$p;\ ?(i>0);\ i:=i-1 \ \cong\ p$$

Since semantically equivalent commands in B are equal, this implies that

$$p;\ ?(i>0);\ i:=i-1 \ =\ p \tag{37}$$

This concludes the proof of (36). □

Relation $(\smile)$ is contained in *Eq.wpa*. For, if h replaced by *abort*, both commands in (35) are semantically equivalent to *abort*. A formal verification is left to the reader. Since $\smile$ is d-invariant, contained in *Eq.wpa*, and not *wp*-strong, it follows that the premiss of Theorem (26) is not satisfied and that this premiss cannot be omitted. This implies

$$p;\ h \notin Lia \ \vee\ p;\ h \,[\!]\, abort \notin Lia.$$

Since $abort \in B$, we have $abort \in Lia$. By formula (28), this proves that $p;\ h \notin Lia$.

6. Program Transformation Methods

6.0. In this section we reformulate the results of Section 5 in order to get a useful tool for program transformation. Results (1), (2) and (3) form our version of the rule of computational induction. Theorems (8) and (13) are applications of the theory and are used in the proofs of 7(16) and 7(24).

6.1. The Accumulative Transformation Rule

Let *Es* be the biggest strong congruence, cf. theorem 5(17). We define relation $\approx$ on $B[H]$ to be congruence *Es* used as an infix operator, so that

$$p \approx q \ \equiv\ \langle p, q\rangle \in Es \tag{0}$$

By Theorem 5(15), we have

$$(\forall p, q \in B[H]: p \approx q: p \cong q) \tag{1}$$

The definition of $\approx$ allows us to derive the following rule, which enables accumulation of knowledge of congruences. We use homomorphism da^* introduced in Section 5.3, and set *Lia* as analysed in Section 5.5.

(2) **Theorem.** Let $(\sim)$ be the congruence generated (cf. 5.1) by the union of the strong congruence $(\approx)$ and an arbitrary binary relation $(\smile)$. Assume that

(a) $(\forall p, q \in B[H]: p \smile q: da^*.p \cong da^*.q \wedge p, q \in Lia)$

(b) $(\forall p, q \in B[H]: p \smile q: d^*.p \sim d^*.q)$

Then it holds that

(c) $(\forall p, q \in B[H] : p \smile q : p \approx q)$

Proof. Let E be the union of $(\smile)$ and Es. By condition (b), relation E is d-invariant. Theorem 5(17) implies that $Wp.Es$ is $\bigvee$-decked and that $Wlp.Es$ is $\bigwedge$-decked in WG. By condition (a), formula 5(21) and Theorems 5(24) and 5(26), we have that $Wp.(\smile)$ is $\bigvee$-decked and $Wlp.(\smile)$ is $\bigwedge$-decked. It follows that $Wp.E$ is $\bigvee$-decked and that $Wlp.E$ is $\bigwedge$-decked. Therefore, E is strong and hence contained in Es. □

Some important properties of "$\approx$" can be proved without linear approximation. We assume that command algebra B contains elements *abort* (cf. Section 5.3), and *skip* and *magic* with

$$wp.skip.x = wlp.skip.x = x,$$
$$wp.magic.x = wlp.magic.x = tt.$$

(3) **Theorem.** (a) If $p, q \in B$ satisfy $p \cong q$, then $p \approx q$.

(b) For all $p \in B[H]$ we have

$$magic; p \approx magic, \quad abort; p \approx abort$$
$$skip; p \approx p, \quad p; skip \approx p, \quad p \,[\!]\, magic \approx p$$

Proof. Let $E0$ be the set of pairs $\langle p, q\rangle$ with $p, q \in B$ and $p \cong q$. Let $E1$ be the set of pairs

$$\langle magic; p, magic\rangle, \quad \langle abort; p, abort\rangle$$
$$\langle skip; p, p\rangle, \quad \langle p; skip, p\rangle, \quad \langle p \,[\!]\, magic, p\rangle$$

with $p \in B[H]$. It suffices to prove that the union $E = E0 \cup E1$ is strong, cf. Section 5.2. Since d^* is a homomorphism and restricts to the identity on B, we have

$$(\forall \langle p, q\rangle \in E :: \langle d^*.p, d^*.q\rangle \in E)$$

so that E is d-invariant, cf. Section 5.2. Using the definitions of $Wp.E$ and $Wlp.E$ in Section 5.2, and the definitions of $\cong$, *magic*, *abort*, *skip*, it is easy to verify that

$$Wp.E = Wup, \quad Wlp.E = Wun$$

Since Wup is $\bigvee$-decked in WG and Wun is $\bigwedge$-decked in WG, this proves that relation E is strong and, hence, contained in $(\approx)$. □

6.2. Change of Declaration

(4) **Corollary.** Let $h, k \in H$ and $p \in B[H]$ be such that $d.h \approx p$ and that $d.k$ is obtained from p by substituting k for h, i.e. $d.k = (h := k).p$. Then $h \approx k$.

Proof. We use Theorem (2) with relation $(\smile)$ consisting of the pair $h \smile k$. Since $h, k \in H$, condition (2)(a) follows from 5(20), 5(21), 5(27). Condition (2)(b) is proved in

$$\begin{array}{ll} & d.h \\ \sim & \{(\sim) \text{ contains } (\approx)\} \\ & p \\ \sim & \{(\sim) \text{ is a congruence; } h \sim k \text{ and } d.k = (h := k).p\} \\ & d.k \quad \square \end{array}$$

Remark. This result shows that a declaration d with $d.h \approx p$ may be replaced by one with $d.h = p$ without changing the semantics.

6.3. Generated Subalgebras

In the applications below we need the following definition. For any subset U of $B[H]$, we define the subset generated by U (denoted by $gen.U$) to be the smallest subalgebra (cf. 1(12)) of $B[H]$ that contains U. Since any intersection of subalgebras is a subalgebra, $gen.U$ can be characterised by

$$p \in gen.U \;\equiv\; (\forall V: U \subset V \;\wedge\; V \text{ subalgebra}: p \in V) \tag{5}$$

6.4. Invariants

In programming, we say that a predicate x is an *invariant* of command p if and only if $[x \Rightarrow wlp.p.x]$. In [Hes88b], Section 7, we proved that for predicates x and y

$$?x;\, p \cong ?x;\, p;\, ?y \;\equiv\; [x \Rightarrow wlp.p.y].$$

Therefore, x is an invariant of p if and only if $?x;\, p \cong ?x;\, p;\, ?x$.

In our general context, let a command s be called an *invariant* of p if and only if $s;\, p \cong s;\, p;\, s$. Let Bs be the subset of B given by

$$p \in Bs \;\equiv\; s;\, p \cong s;\, p;\, s \tag{6}$$

Let Hs be a subset of H such that

$$(\forall h \in Hs :: d.h \in gen.(Bs \cup Hs)) \tag{7}$$

In the next result, we need the subalgebra Bup of B, as defined in Section 4.2.

(8) **Theorem.** If $s \in Bup$, then $s;\, h \approx s;\, h;\, s$ for all $h \in Hs$.

Proof. We apply theorem (2) to the congruence $\sim$ generated by the strong congruence $\approx$ and the pairs

$$s;\, h \smile s;\, h;\, s \quad \text{with } h \in Hs \tag{9}$$

A straightforward calculation shows that the pairs (9) belong to the abortive congruence, i.e. $da^*.(s;\, h) \cong da^*.(s;\, h;\, s)$ for all $h \in Hs$. Since $s \in Bup$, it follows from 5(31) that the pairs (9) are contained in Lia. This proves condition (2)(a). As for condition (2)(b), it follows from (9) and the definitions of Bs and $\sim$; that

$$(\forall p \in Bs \cup Hs :: s;\, p \sim s;\, p;\, s)$$

Using structural induction, one can obtain

$$(\forall p \in gen.(Bs \cup Hs) :: s;\, p \sim s;\, p;\, s)$$

Now it follows with (7) that $s;\, d.h \sim s;\, d.h;\, s$ for all $h \in Hs$. Since $s \in B$, this implies

$$(\forall h \in Hs :: d^*.(s;\, h) \sim d^*.(s;\, h;\, s))$$

By (2), this proves that the pairs (9) satisfy relation $\approx$. □

6.5. Commuting Commands

Let s be a fixed command in B. A command p is said to *commute* with s if and only if $p;\, s \cong s;\, p$. We define Bt to be the set of the commands $p \in B$ that commute with s. So we have

$$p \in Bt \;\equiv\; p \in B \;\wedge\; p;\, s \cong s;\, p \tag{10}$$

Let a subset Ht of H be given such that

$$(\forall h \in Ht :: d.h \in gen.(Bt \cup Ht)) \tag{11}$$

Recall from [Hes88b] formula (4), that command s is called *total* if and only if it satisfies Dijkstra's law of the excluded miracle, i.e.

$$wp.s.ff = ff \tag{12}$$

(13) **Theorem.** If command s is total and $s \in Bup$, then $h; s \approx s; h$ for all $h \in Ht$.

Proof. We use Theorem (2). Let $\sim$ denote the infix operator that denotes the congruence on $B[H]$ that is generated by strong congruence and the pairs

$$h; s \smile s; h \quad \text{with } h \in Ht \tag{14}$$

Totality of command s is used to prove that, for all $h \in Ht$,

$$da^*.(h; s) \cong da^*.(s; h)$$

Since $s \in Bup$, it follows from 5(31) that the pairs (18) are contained in *Lia*. Therefore, condition (2)(a) holds. The verification of (2)(b) in this case is completely analogous to the case of Theorem (8). The details of the proof are left to the reader. □

Remarks. The condition in theorem (13), that command s be total, is necessary. For, let $s = magic$ (a non-total command). Let procedure $h0$ be declared by $d.h0 = h0$. Then procedure $h0$ is semantically equivalent to *abort*, so that

$$h0; magic \not\approx magic; h0$$

but s is element of *Bup* and the singleton set Ht that consists of $h0$, satisfies condition (11).

In our proof of (13), condition $s \in Bup$ is essential. (*Added in proof.* Recently we found an example to show that this condition is essential for the validity of (13).)

7. Programming Examples

7.0. We come back to the level of a state space, as considered in Sections 1.3 and 4.4. We assume that command algebra B contains all commands that we need. For a predicate x we have a command $?x \in B$, cf. 1(20). One easily verifies that

$$?x; ?y \cong ?(x \wedge y), \quad ?x \,[\!]\, ?y \cong ?(x \vee y)$$

With Theorem 6(3) it follows that

$$?x; ?y \approx ?(x \wedge y), \quad ?x \,[\!]\, ?y \approx ?(x \vee y)$$

In the same way, we get from 6(3) that, for commands $p, q \in B[H]$,

$$p \,[\!]\, ?ff; q \;\approx\; p \tag{1}$$

It follows from 6(3) and [Hes88b] Section 7, that for predicates x and y and any command $p \in B$, we have

$$[x \Rightarrow wlp.p.y] \;\equiv\; ?x; p \approx ?x; p; ?y \tag{2}$$

In the programming examples below, this equality will be used to insert or delete guards like $?y$ in sequences of commands.

7.1. Localised Relations

If f is a function on the state space and m is a value, we write $(f = m)$ to denote the predicate on the state space given by

$$(f = m).r \equiv (f.r = m) \quad \text{for } r \in St \tag{3}$$

In this notation, the operator "=" yields a predicate instead of a boolean value. Let us call such an operator a *localised* relation.

If f is an integer valued function and m is an integer, we have predicates $(f > m)$, $(f \geq m)$, etc., that are defined in the same way. It is clear that these predicates satisfy

$$(f \geq m) = (f = m) \vee (f > m) \tag{4}$$

where the symbol "=" in the middle stands for equality of predicates.

Remark. The introduction of localised relations is the price we pay for choosing to avoid mentioning states whenever possible. In the programming examples below, the use of localised relations is very natural. In the theory of the previous chapters, however, the global interpretation of equality and inequalities enabled us to apply the abstraction of treating predicates as elements of an abstract lattice, cf. Section 0.3. Therefore, we did not adopt the choice of Dijkstra and Scholten, cf. [DiS90] Chapter 1, of always using the localised interpretation.

7.2. For-Loops and Unbounded Choice

A for-loop is a repetitive command in which the number of repetitions is a function of the pre-state. If v is the state function and q is the body of the loop, the loop can be defined as the guarded but unbounded choice

$$k = (\mathbb{I}\, r : r \geq 0 : ?(v = r); q^r).$$

In the formal treatment, an unbounded choice $(\mathbb{I}\, r :: p.r)$ is defined as the greatest lower bound of the elements $p.r$. Thus, we postulate that command algebra B has greatest lower bounds of arbitrary nonempty families, that the choice of such families satisfies both unbounded distributive laws

$$(\mathbb{I}\, r :: p.r); s = (\mathbb{I}\, r :: p.r; s) \tag{5}$$

$$s; (\mathbb{I}\, r :: p.r) = (\mathbb{I}\, r :: s; p.r) \tag{6}$$

and that both $w = wp$ and $w = wlp$ satisfy, for any $x \in X$,

$$w.(\mathbb{I}\, r :: p.r)x = (\bigwedge r :: w.(p.r).x) \tag{7}$$

Remark. If we admit the empty choice in formula (7), the empty choice is equivalent to *magic*, so that formula (6) is not valid for the empty choice and $s = abort$. Therefore, the empty choice cannot be admitted at least in formula (6).

7.3. A Pushdown Stack for Recursion

Let procedure $h0 \in H$ be declared by

$$d.h0 = s\mathbb{I}(\mathbb{I}\, j :: p.j; h0; q.j) \tag{8}$$

where s, $p.j$ and $q.j$ are commands in B. The dummy j is supposed to range over a nonempty finite set J, so that any family of commands $(j \in J :: u.j)$ has a well-defined choice $(\mathbb{I}j :: u.j)$.

In the standard implementation of a recursive procedure one builds a stack of return addresses, so that the recursion can be replaced by a repetition (i.e. tail recursion). In the special case of declaration (8), the repetition has three parts. It starts with a repetition in which some commands $p.j$ are executed and the stack is built. Then follows command s. Finally, the stack is broken down again and corresponding commands $q.j$ are executed.

We want to give a formal proof of the correctness of this standard implementation of $h0$ by means of tail recursion and a pushdown stack. This is done in two stages. In the first stage we specify the stack and its operations, and we add the appropriate stack operations to declaration (8). We introduce convenient names, so that the declaration gets the simple form (17). We specify the two repetitions in the form of a tail-recursive procedure and a for-loop, and get a relationship between $h0$ and the composition of the two repetitions, cf. (22). The proof is based on Theorem 6(2).

We assume that the state space contains a pushdown stack for elements of J. Let $push.j \in B$ be the command that pushes value j onto the stack. Let $pop \in B$ be the command that removes the top element of the stack. Let top be the J-valued function on the state space that returns the value of the top of the stack. We postulate, accordingly,

$$push.j;\ pop \ \approx\ skip \tag{9}$$

$$[true \ \Rightarrow\ wlp.(push.j).(top = j)] \tag{10}$$

It follows from (2) that (10) is equivalent to

$$push.j \ \approx\ push.j;\ ?(top = j) \tag{11}$$

We introduce commands pp, $qq \in B$ by

$$pp \ =\ (\mathbb{I}j :: p.j;\ push.j) \tag{12}$$

$$qq \ =\ (\mathbb{I}j :: ?(top = j);\ pop;\ q.j) \tag{13}$$

We observe that

$$\begin{array}{ll} & push.j;\ qq \\ \approx & \{(11), (13)\} \\ & push.j;\ ?(top = j);\ (\mathbb{I}i :: ?(top = i);\ pop;\ q.i) \\ \approx & \{(0)\} \\ & push.j;\ (\mathbb{I}i :: ?(top = j \wedge top = i);\ pop;\ q.i) \\ \approx & \{(1)\} \\ & push.j;\ ?(top = j);\ pop;\ q.j \\ \approx & \{(9), (11)\} \\ & q.j \end{array}$$

This proves

$$q.j \ \approx\ push.j;\ qq \tag{14}$$

In order to use the stack for an iterative implementation of declaration (8), we need the assumption that commands s, $p.j$ are independent of the stack. Therefore, we postulate

$$(\forall i,j::\ s;\mathit{push.j} \approx \mathit{push.j};s \wedge p.i;\mathit{push.j} \approx \mathit{push.j};p.i \wedge q.i;\mathit{push.j} \approx \mathit{push.j};q.i) \tag{15}$$

It follows from (9) that commands *push.j* are total. We may assume that they are deterministic, so that they are element of *Bup*, cf. 4(15). Now, we may apply theorem 6(13) with $s := \mathit{push.j}$. By (8) and (15), we may assume that $h0 \in Ht$. So we obtain

$$h0;\mathit{push.j} \approx \mathit{push.j};h0 \tag{16}$$

We observe

$$\begin{array}{ll} & d.h0 \\ \approx & \{(8),(14)\} \\ & s \,[\!]\, ([\!]\, j::p.j;h0;\mathit{push.j};qq) \\ \approx & \{(16)\} \\ & s \,[\!]\, ([\!]\, j::p.j;\mathit{push.j};h0;qq) \\ = & \{(12)\} \\ & s \,[\!]\, pp;h0;qq \end{array}$$

This proves

$$d.h0 \approx s \,[\!]\, pp;h0;qq \tag{17}$$

We assume that the depth of the stack is given by an integer-valued state function *dp*, that satisfies for all n and j

$$\begin{array}{l} [dp=n \Rightarrow \mathit{wlp.}(\mathit{push.j}).(dp=n+1)] \\ [dp=n \Rightarrow \mathit{wlp.pop.}(dp=n-1)] \\ [dp=n \Rightarrow \mathit{wlp.s.}(dp=n)] \\ [dp=n \Rightarrow \mathit{wlp.}(p.j).(dp=n)] \\ [dp=n \Rightarrow \mathit{wlp.}(q.j).(dp=n)] \end{array} \tag{18}$$

It follows from (12), (13) and (18) that

$$\begin{array}{l} [dp=n \Rightarrow \mathit{wlp.pp.}(dp=n+1)] \\ [dp=n \Rightarrow \mathit{wlp.qq.}(dp=n-1)] \end{array} \tag{19}$$

In this simple case, the stack implementation of recursion consists of two phases: a first phase when the stack is built, and a second one, during which the stack is inspected and broken down. In view of formula (17), we declare a tail-recursive procedure $h1$ by

$$d.h1 = s \,[\!]\, pp;h1, \tag{20}$$

and we define for-loops $k.n \in B$, for integer n, by

$$k.n = ([\!]\, r: r \geq n: ?(dp=r);qq^{r-n}) \tag{21}$$

compare Section 7.2.

Now, procedure $h0$, if called with a stack of depth n, is equivalent to the composition $(h1;k.n)$. This is formalised in

(22) **Theorem.** $?(dp=n);h0 \approx ?(dp=n);h1;k.n.$

Proof. It follows from (18) and (19) that

$$[dp \geq n \Rightarrow wlp.s.(dp \geq n)]$$
$$[dp \geq n \Rightarrow wlp.pp.(dp \geq n)]$$

and hence by (2)

$$?(dp \geq n); s' \approx ?(dp \geq n); s; ?(dp \geq n) \tag{23}$$
$$?(dp \geq n); pp \approx ?(dp \geq n); pp; ?(dp \geq n)$$

By (20) and Theorem 6(8) with $s := ?(dp \geq n)$, it follows from (23) that

$$?(dp \geq n); h1 \approx ?(dp \geq n); h1; ?(dp \geq n) \tag{24}$$

Using the unbounded distributivity laws (5) and (6), and calculations with guards and the rules (0) and (1), one can verify

$$?(dp = n); k.n \approx ?(dp = n) \tag{25}$$
$$?(dp > n); k.n \approx ?(dp > n); k.(n+1); qq$$

We are ready for the application of Theorem 6(2). Let $\sim$ be the congruence on $B[H]$ generated by $\approx$ and the pairs

$$?(dp = n); h0 \smile ?(dp = n); h1; k.n \quad \text{for all integers } n \tag{26}$$

It is easy to verify that these pairs belong to the abortive congruence. As we have seen earlier, guards belong to *Bup*. By 5(31), it follows that the pairs (26) belong to *Lia*. It remains to verify condition 6(2)(b). We observe that

$$\begin{aligned}
& d^*.(?(dp = n); h1; k.n) \\
= \quad & \{(20)\} \\
& ?(dp = n); (s \,\|\, pp; h1); k.n \\
\approx \quad & \{(2), (18), (19) \text{ and } (24)\} \\
& ?(dp = n); (s; ?(dp = n) \,\|\, pp; ?(dp = n+1); h1; ?(dp > n)); k.n \\
\approx \quad & \{\text{distribution of } k.n; (25)\} \\
& ?(dp = n); (s; ?(dp = n) \,\|\, pp; ?(dp = n+1); h1; ?(dp > n); k.(n+1); qq) \\
\approx \quad & \{(24)\} \\
& ?(dp = n); (s; ?(dp = n) \,\|\, pp; ?(dp = n+1); h1; k.(n+1); qq) \\
\sim \quad & \{\text{induction hypothesis } (26)\} \\
& ?(dp = n); (s; ?(dp = n) \,\|\, pp; ?(dp = n+1); h0; qq) \\
\approx \quad & \{(2), (18), (19)\} \\
& ?(dp = n); (s \,\|\, pp; h0; qq) \\
\approx \quad & \{(17)\} \\
& d^*.(?(dp = n); h0)
\end{aligned}$$

Therefore, the pairs (26) satisfy condition 6(2)(b). By Theorem 6(2), this proves the theorem. □

Later on, it is convenient to replace the for-loop $k.1$ by a procedure. So we declare the tail-recursive procedure kk by

$$d.kk = ?(dp = 1) \,\|\, ?(dp > 1); qq; kk \tag{27}$$

Using induction on r and the second formula of (19), one can prove that, for $r \geq 1$,

$$?(dp = r); kk \cong ?(dp = r); qq^{r-1}$$

and, hence, that $kk \cong k.1$ in $B[H]$. By Theorem (22) and 6(1), it follows that

$$?(dp = n);\ h0 \cong ?(dp = n);\ h1;\ kk \tag{28}$$

Remark. In our view, one of the nice aspects of the above development is that the stack is characterised by the formulae (9), (10), (15) and (18). So we did not need any form of data abstraction.

7.4. A Strong Version of the Euclidean Algorithm

In this section, Theorem (22) is applied to a recursive version of the euclidean algorithm. This version is the extension of the standard algorithm, in which the greatest common divisor of a given pair of positive integers is expressed as an integral linear combination of that pair. So, we use a state space with integer variables i, j, a, b, and the algorithm consists of a procedure $h0$ specified by

$$[i = i0 \wedge j = j0 \Rightarrow wp.h0.(gcd.i0.j0 = a * i0 - b * j0)] \tag{29}$$

where $i0$ and $j0$ are arbitrary positive specification consistants. A standard program derivation, which is left to the reader, yields

$$\begin{aligned} d.h0 = {} & ?(i = j);\ a, b := 1, 0 \\ & [\!] \ ?(i < j);\ j := j - i;\ h0;\ a := a + b \\ & [\!] \ ?(j < i);\ i := i - j;\ h0;\ b := a + b \end{aligned} \tag{30}$$

In this declaration, we recognise declaration (8) with

$$\begin{aligned} s &= (?(i = j);\ a, b := 1, 0) \\ p.0 &= (?(i < j);\ j := j - i) \\ q.0 &= (a := a + b) \\ p.1 &= (?(j < i);\ i := i - j) \\ q.1 &= (b := a + b) \end{aligned} \tag{31}$$

where we choose J to be the set of the numbers 0 and 1. We let the stack be represented by a positive integer m, which is treated as a sequence of binary digits headed by 1. The corresponding operations and functions are given by

$$push.0 = (m := 2 * m) \tag{32}$$

$$push.1 = (m := 2 * m + 1)$$

$$pop = (!(m \neq 1);\ m := m \ \mathbf{div}\ 2) \quad \{\text{see } 1(21) \text{ for } !(m \neq 1)\}$$

$$top = m \ \mathbf{mod}\ 2$$

$$dp = \text{"number of binary digits of } m\text{"}$$

Notice that $m = 1$ represents the empty stack, of depth 1, and that popping the empty stack yields *abort*. So we may assume that $m > 0$ holds throughout the state space. Now it is easy to verify the formulae (9), (10), (15), (18). Formulae (12) and (13) specialise to

$$\begin{aligned} pp &= ?(i<j);\ j := j-i;\ m := 2*m \\ &\quad [\!] \ ?(j<i);\ i := i-j;\ m := 2*m+1 \\ qq &= ?(m \bmod 2 = 0);\ !(m \neq 1);\ m := m \operatorname{\mathbf{div}} 2;\ a := a+b \\ &\quad [\!] \ ?(m \bmod 2 = 1);\ !(m \neq 1);\ m := m \operatorname{\mathbf{div}} 2;\ b := a+b \end{aligned} \tag{33}$$

As condition $dp = 1$ is equivalent to $m = 1$, formula (28) yields

$$(m := 1;\ h0)^{\cdot} \cong (m := 1;\ h1;\ kk) \tag{34}$$

where procedures $h1$ and kk are declared, cf. (20) and (27), by

$$\begin{aligned} d.h1 &= ?(i=j);\ a, b := 1, 0 \\ &\quad [\!] \ ?(i<j);\ j := j-i;\ m := 2*m;\ h1 \\ &\quad [\!] \ ?(j<i);\ i := i-j;\ m := 2*m+1;\ h1 \\ d.kk &= ?(m=1) \\ &\quad [\!] \ ?(m \bmod 2 = 0);\ m := m \operatorname{\mathbf{div}} 2;\ a := a+b;\ kk \\ &\quad [\!] \ ?(m \neq 1 \wedge \bmod 2 = 1);\ m := m \operatorname{\mathbf{div}} 2;\ b := a+b;\ kk \end{aligned} \tag{35}$$

The bodies $d.h1$ and $d.kk$ have been rewritten according to some equivalences in $B[H]$ that are based on Theorem 6(3). In particular, the assertions $!(m \neq 1)$ in $d.kk$ were superfluous and have been omitted.

The procedures $h1$ and kk use tail recursion of a kind that can be translated directly into iteration. Thus, formula (34) implies that the composition $[\![m := 1;\ h0]\!]$ is equivalent to the program in guarded commands

$$\begin{aligned} &[\![m := 1 \\ &;\mathbf{do}\ i<j \rightarrow j := j-i;\ m := 2*m \\ &\ [\!]\ j<i \rightarrow i := i-j;\ m := 2*m+1 \\ &\ \mathbf{od} \\ &;a, b := 1, 0 \\ &;\mathbf{do}\ m \bmod 2 = 0 \rightarrow m := m \operatorname{\mathbf{div}} 2;\ a := a+b \\ &\ [\!]\ m \neq 1 \wedge m \bmod 2 = 1 \rightarrow m := m \operatorname{\mathbf{div}} 2;\ b := a+b \\ &\ \mathbf{od} \\ &]\!] \end{aligned} \tag{36}$$

Remark. If we compare program (36) with the program in EWD 570, cf. [Dij82], we see a striking similarity. Dijkstra's program is

$$\begin{aligned} &[\![m, a, b := N, 1, 0 \\ &;\mathbf{do}\ m \neq 0 \wedge m \bmod 2 = 0 \rightarrow m := m \operatorname{\mathbf{div}} 2;\ a := a+b \\ &\ [\!]\ m \bmod 2 = 1 \rightarrow m := m \operatorname{\mathbf{div}} 2;\ b := a+b \\ &\ \mathbf{od}\quad \{b = fusc.N\} \\ &]\!] \end{aligned}$$

where *fusc* is the function given by

$$\begin{aligned} &fusc.1 = 1 \\ &fusc.(2*n) = fusc.n \\ &fusc.(2*n+1) = fusc.n + fusc.(n+1) \end{aligned}$$

It follows that the final values of a and b of (36) satisfy $a+b = fusc.N$ where N is the value of m after the first repetition of (36).

This is not a coincidence. In fact, Dijkstra found *fusc* while working on the inversion of the euclidean algorithm, cf. [Gri81] chapter 21. For more about *fusc*, see EWD 578 in [Dij82]. I found the transformation from (30) into (36) in October 1987. The problem of its justification was the inspiration for this paper. Jan van de Snepscheut recognised *fusc*.

References

[Apt81] Apt, K. R.: Ten Years of Hoare's Logic. A Survey - Part 1. ACM *Transactions on Programming Languages and Systems*, **3**, 431-483 (1981).

[Bac87] Back, R. J. R.: A calculus of refinements for program derivations. *Dept. Comp. Sci. Abo Finland, Ser. A*, **54**, (1987).

[BBK87] Baeten, J. C. M., Bergstra, J. A. and Klop, J. W.: On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science*, **51**, 129-176 (1987).

[Bak80] Bakker, J. W. de: *Mathematical Theory of Program Correctness*, Prentice-Hall, 1980.

[BaW81] Bauer, F. L. and Wossner H.: *Algorithmische Sprache und Programmentwicklung*, Springer-Verlag, 1981.

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

[Dij82] Dijkstra, E. W.: *Selected Writings on Computing: a Personal Perspective*, Springer-Verlag, 1982.

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

[Gri81] Gries, D.: *The Science of Programming.*, Springer-Verlag, 1981.

[Hes88a] Hesselink, W. H.: Interpretations of Recursion Under Unbounded Nondeterminacy. *Theoretical Computer Sceience*, **59**, 211-234 (1988).

[Hes89a] Hesselink, W. H.: Predicate Transformer Semantics of General Recursion (WHH 8). *Acta Informatica*, **26**, 309-332 (1989).

[Hes88b] Hesselink, W. H.: An Algebraic Calculus of Commands (WHH 13). Tech. Rep. CS 8808, Groningen University, 1988.

[Hes89b] Hesselink, W. H.: Modalities of Nondeterminacy. Draft, September 1989.

[Hoa71] Hoare, C. A. R.: Procedures and Parameters: an Axiomatic Approach. In: *Symposium on Semantics of Algorithmic Languages*, E. Engeler (ed.), Lecture Notes in Math. 188, pp. 102-116, Springer-Verlag, 1971.

[Hoa78] Hoare, C. A. R.: Some Properties of Predicate Transformers. *J. ACM*, **25**, 461-480 (1978).

[Kel55] Kelley, J. L.: *General Topology.* Van Nostrand, 1955.

[Lak76] Lakatos, I.: *Proofs and Refutations: the Logic of Mathematical Discovery* J. Worrall and E. Zahar, (eds), Cambridge University Press, 1976.

[Man74] Manna, Z.: *Mathematical Theory of Computation.* McGraw-Hill, 1974.

[Mor87] Morgan, C. C.: Data Refinement by Miracles. *Information Processing Letters*, **26**, 243-246 (1987/88).

[MoG88] Morgan, C. C. and Gardiner, P. H. B.: Data refinement by calculation. Draft, July 1988.

[Nel87] Nelson, G.: A generalization of Dijkstra's Calculus. ACM Transactions on Programming Languages and Systems **11**, 517-561 (1989).

[TaZ71] Takeuti, G. and Zaring, W. M.: *Introduction to Axiomatic Set Theory.* Graduate texts in mathematics 1, Springer-Verlag, 1971.

Received December 1988
Accepted in November 1989 by D. Gries