Science of Computer Programming 20 (1993) 223–251
Elsevier

Statement inversion and strongest postcondition

R.J.R. Back and J. von Wright

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

Communicated by C.B. Jones
Received April 1990
Revised September 1992

Abstract

Back, R.J.R. and J. von Wright, Statement inversion and strongest postcondition, Science of Computer Science 20 (1993) 223–251.

A notion of inverse commands is defined for a language which permits both demonic and angelic nondeterminism, as well as miracles and nontermination. Every conjunctive and terminating command is invertible, the inverse being non-miraculous and disjunctive. A simulation relation between commands is described using inverse commands. A generalised form of inverse is defined for arbitrary conjunctive commands. The generalised inverses are shown to be closely related to strongest postconditions.

1. Introduction

The weakest precondition calculus of Dijkstra [7] originally confined itself to the language of guarded commands, containing only executable program constructs. Later extensions have added specification constructs, permitting unbounded nondeterminism, miracles, and angelic nondeterminism [1,4,11,13,14]. This way the weakest precondition calculus has been extended to non-executable program constructs. In addition to making the calculus mathematically simpler, this has made it possible to treat, e.g., arbitrary input–output specifications, data refinement and parallel programs within the same calculus. The weakest precondition calculus is the basis for the refinement

Correspondence to: J. von Wright, University of Cambridge Computer Laboratory, New Museums Site, Pembroke Street, Cambridge CB2 3QG, England, UK. E-mail: pjv1000@cl.cam.ac.uk. From August 1993: SHH, B.O. Box 287, SF-65101, Vasa, Finland.

calculus, invented by Back [1] and further developed by Back [2], Morgan [12], and Morris [13].

Identifying statements with their weakest precondition predicate transformers makes the language a subset of the complete lattice of monotonic predicate transformers, thus permitting lattice-based reasoning about programs [4,13]. We follow this by now reasonably well-established tradition, writing $S(Q)$ rather than $\mathrm{wp}(S, Q)$ for the weakest precondition of statement S with respect to predicate Q.

The weakest precondition calculus is a calculus of total correctness. Partial correctness can be studied through weakest liberal preconditions (wlp) or strongest postconditions (sp). Generally, wlp has been used, often in association with wp in order to give a more fine-grained semantics, while sp has not been used very much. Strongest postconditions are theoretically investigated by de Bakker [6]. The relation between strongest postcondition and weakest precondition is close to a relation of inversion; Back [2] gives postulates that characterise this relation.

In this paper we define a notion of *inverse commands* in the following way: S^{-1} is the inverse of S if

$$S^{-1}; S \leqslant \mathit{skip} \leqslant S; S^{-1},$$

where $\leqslant$ is the refinement relation. This is a generalisation of the concept of inverse used in function theory; if S is functional, then S^{-1} is the inverse function of S. We show how inverse commands can be computed directly in the command lattice defined by Back and von Wright in [4] and how ordinary program constructs are inverted. We also show how a simulation relation between commands can be characterised using inverse commands.

Inverses exist only for commands which are always terminating and conjunctive. To overcome the termination restriction, we define *generalised inverses* which exist for all conjunctive commands: the command S^{-} is defined to be a generalised inverse of S if S^{-} inverts S whenever S terminates. Generalised inverses are not unique, but every conjunctive command has a unique least generalised inverse. We also show how generalised inverses can be computed for arbitrary conjunctive commands.

An important aspect of this paper is that we work wholly within the total correctness framework of the refinement calculus. Thus we do not define the notions of weakest liberal precondition or strongest postcondition. Instead, we show how generalised inverses share many essential properties with strongest postconditions, permitting them to replace strongest postconditions in reasoning about programs. In particular, we characterise the refinement relation between conjunctive commands in a total correctness formula involving generalised inverses. This result is essentially a reformulation of a theorem in [2], where strongest postconditions are used.

Organisation of the paper

The rest of the paper is organised as follows. Section 2 gives a short description of the command language $\mathcal{C}$, defined in [4]. This command language contains all monotonic predicate transformers. We show how ordinary specification and program constructs can be defined in $\mathcal{C}$. This section contains mostly old material and it is rather dense; the reader is referred to [4] for more detail and for proofs. In Section 3, we define the concept of inverse command and show existence and uniqueness properties. We give rules for computing inverses and show how inverses can be used to describe data refinement between commands, a topic which is treated in more detail in a separate paper [3]. In Section 4 we define generalised inverses and show how a generalised inverse can be computed for an arbitrary conjunctive command. In Section 5 we show that a generalised inverse of a conjunctive command S is very similar to the strongest postcondition predicate transformer of S. We also characterise the refinement relation between conjunctive commands using a total correctness formula involving generalised inverses. Finally, Section 6 contains some concluding remarks.

Remark on proof style and notation

We use a calculational style of proof, with comments written in brackets [...]. In many proofs, we use distributivity properties of commands, which have been proved in [4]. In such cases, we simply justify the calculation by a reference to "distributivity". In formulas, we use the convention that substitution binds stronger than logical connectives. Also, the scope of quantifiers extends as far to the right as possible.

2. The lattice-based command language

We assume that the concepts of partial orders and lattices (complete, distributive, and boolean lattices) are familiar, as well as Dijkstra's weakest precondition calculus. The lattice of monotonic functions from one lattice to another is ordered by pointwise extension:

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

2.1. Predicate transformers

Let *Var* be a countable set of *program variables*. We assume that every variable x is associated with a *nonempty* set D_x of values (the *type* of x). Lists of variables are typically denoted u, while values are typically denoted c and lists of values d. A *state* is a function which maps every x in *Var* to some value in D_x. The set of all states (the *state space*) is denoted Σ.

Let $Bool = \{ff, tt\}$ be the complete lattice of truth values for a two-valued logic, ordered so that $ff \leqslant tt$. A *predicate* is a function from Σ to *Bool*. The set of all predicates is denoted *Pred*.

Substitutions and quantification

A *substitution* in Σ is defined in the following way: $\sigma[c/x]$ is the state which differs from σ only in that it assigns the value c to the variable x. Substitutions in Σ are extended to *Pred* in the following way:

$$P[d/u](\sigma) = P(\sigma[d/u]).$$

We also define *quantified predicates*:

$$\forall u.\ P \stackrel{\text{def}}{=} \bigwedge_d P[d/u],$$

$$\exists u.\ P \stackrel{\text{def}}{=} \bigvee_d P[d/u],$$

where d ranges over all lists of values (of appropriate type) of the same length as u. Given these definitions, we can treat predicates much in the same way as we treat ordinary first-order formulas.

Predicate transformers

A *predicate transformer* is a function on *Pred*. We write *Mtran* for the complete lattice of all *monotonic* predicate transformers. The top element of *Mtran* is *magic* which is the unit element of lattice meet. Similarly, the bottom element *abort* is the unit element of lattice join. The unit element of functional composition is the predicate transformer *skip*.

2.2. The command language

We now define the lattice-based command language of [4]. We call this language $\mathcal{C}$. It is powerful enough to express both program specifications and executable statements.

Syntax and semantics

The *commands* are defined by the following syntax:

$$\begin{array}{lll} S ::= & \{P\} & (\textit{assert command}) \\ \mid & [P] & (\textit{assume command}) \\ \mid & \langle u \leftarrow d \rangle & (\textit{store command}) \\ \mid & S_1; S_2 & (\textit{sequential composition}) \\ \mid & \bigwedge_{i \in I} S_i & (\textit{demonic choice}) \\ \mid & \bigvee_{i \in I} S_i & (\textit{angelic choice}) \end{array}$$

Here P is a predicate, S and S_i are commands for all i and I is an index set (I may be infinite), u is a list of distinct variables, and d a list of values of the same length as u.

A command S in $\mathcal{C}$ denotes a predicate transformer in *Mtran*. $S(Q)$ is the predicate that holds for exactly those initial states for which S is guaranteed to succeed in establishing Q. This is, in essence, the *weakest precondition* semantics of [7], extended to the larger set of program constructs considered here. The meaning of a command S is thus defined as follows:

$$\{P\}(Q) = P \wedge Q,$$

$$[P](Q) = P \Rightarrow Q,$$

$$\langle u \leftarrow d\rangle(Q) = Q[d/u],$$

$$(S_1; S_2)(Q) = S_1(S_2(Q)),$$

$$\left(\bigwedge_{i\in I} S_i\right)(Q) = \bigwedge_{i\in I} S_i(Q),$$

$$\left(\bigvee_{i\in I} S_i\right)(Q) = \bigvee_{i\in I} S_i(Q).$$

Operational meaning

The *assertion* $\{P\}$ leaves the state unchanged if the predicate P holds, otherwise it aborts. The *assumption* $[P]$ also leaves the state unchanged if P holds, but succeeds (miraculously) otherwise. Miraculous success means that the command succeeds in establishing *any* postcondition Q, even *false*. The *store command* $\langle u \leftarrow d\rangle$ assigns the variables u the values d, the other variables keeping their old values.

The execution of a compound command S can be described as a game between two parties, the *demon* and the *angel*. The demon chooses a command S_i to be executed in a demonic choice $\bigwedge_{i\in I} S_i$, while the angel chooses a command S_i to be executed in an angelic choice $\bigvee_{i\in I} S_i$. This interpretation of command execution is discussed in [5].

2.3. Sublanguages and completeness

The command language introduces a number of new features into the weakest precondition calculus which were not present in the original guarded command language of [7]. Of the original "healthiness" conditions proposed by Dijkstra, only the monotonicity condition is satisfied by all commands in $\mathcal{C}$.

We make the following definitions, for any $S \in \mathcal{C}$:

($\bot$) S is *non-miraculous* (*strict with respect to false*) if $S(\mathit{false}) = \mathit{false}$.
($\top$) S is *always terminating* (*strict with respect to true*) if $S(\mathit{true}) = \mathit{true}$.

($\wedge$) S is *conjunctive* if $S(\bigwedge_{i\in I} Q_i) = \bigwedge_{i\in I} S(Q_i)$ for all nonempty sets of predicates $\{Q_i\}_{i\in I}$.

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

These four properties are independent of each other. Thus there are sixteen different ways of combining them. We index $\mathcal{C}$ with some of the symbols for these properties to denote a sublanguage where all commands are required to have the properties in question. Thus, for example, $\mathcal{C}_\vee^\top$ is the set of all always terminating disjunctive commands.

Dual commands

Every command S has a *dual* S°, defined by

$$S^\circ(Q) \stackrel{\text{def}}{=} \neg S(\neg Q).$$

The duality operator is investigated in more detail in [4]. We recall that dualisation is antimonotonic:

$$S_1 \leqslant S_2 \quad \Leftrightarrow \quad S_2^\circ \leqslant S_1^\circ. \tag{1}$$

We also note the following fundamental dualities in the command language:

$$\{P\}^\circ = [P], \tag{2}$$

$$\langle u \leftarrow d\rangle^\circ = \langle u \leftarrow d\rangle, \tag{3}$$

$$(S_1; S_2)^\circ = S_1^\circ; S_2^\circ, \tag{4}$$

$$\Big(\bigwedge_{i\in I} S_i\Big)^\circ = \bigvee_{i\in I} S_i^\circ. \tag{5}$$

Completeness of the command language

By definition, each command corresponds to a monotonic predicate transformer. Conversely, in [4] we show that every monotonic predicate transformer can be constructed as a command.

In [5] we also show completeness results for a number of sublanguages of $\mathcal{C}$. We recall the results for the languages $\mathcal{C}_\wedge$ and $\mathcal{C}_\wedge^\top$, which will be used later on.

Lemma 2.1. *The commands in $\mathcal{C}_\wedge$ and $\mathcal{C}_\wedge^\top$ can be constructed as follows:*

(a) *Every conjunctive command can be constructed using the primitive commands $\{P\}$, $[P]$, and $\langle u \leftarrow d\rangle$ and the constructors ";" and "$\wedge$".*

(b) *Every conjunctive and always terminating command can be constructed using the primitive commands $[P]$ and $\langle u \leftarrow d\rangle$ and the constructors ";" and "$\wedge$".*

2.4. Specification and program constructs in the command language

The command language constructs are quite low level, and not as such very usable in program derivations. We now show how to define more useful derived constructs in the command language. These constructs are defined as abbreviations for certain compound commands in $\mathcal{C}$.

We first show how the unit elements of the three basic constructors can be expressed using assertions and assumptions:

$$abort = \{false\},$$

$$skip = \{true\} = [true],$$

$$magic = [false].$$

Update commands

The update commands permit an arbitrary postcondition to be established directly by assigning suitable values to the program variables. We define the *demonic update command* $\langle \wedge u.\ P \rangle$ and its dual, the *angelic update command* $\langle \vee u.\ P \rangle$, as follows:

$$\langle \wedge u.\ P \rangle \stackrel{\text{def}}{=} \left(\bigwedge_d \langle u \leftarrow d \rangle \right); [P],$$

$$\langle \vee u.\ P \rangle \stackrel{\text{def}}{=} \left(\bigvee_d \langle u \leftarrow d \rangle \right); \{P\}.$$

The predicate transformers for these commands can be computed from the definition. They are:

$$\langle \wedge u.\ P \rangle (Q) = \forall u.\ P \Rightarrow Q,$$

$$\langle \vee u.\ P \rangle (Q) = \exists u.\ P \wedge Q.$$

Both commands assign values to u nondeterministically, so that the postcondition P is established. If P cannot be established, then the demonic update succeeds miraculously while the angelic update aborts. Thus the demonic update is in $\mathcal{C}_\wedge^\top$ while the angelic update is in $\mathcal{C}_\vee^\perp$.

We note that the update commands can be described in the following simple way:

$$\langle \wedge u.\ P \rangle = \langle \wedge u.\ true \rangle; [P], \tag{6}$$

$$\langle \vee u.\ P \rangle = \langle \vee u.\ true \rangle; \{P\}. \tag{7}$$

Nondeterministic assignment commands

The update commands do not permit the new values of the variables to depend on the old values. The (nondeterministic) *assignments* defined below remedy this.

Assume that u' is a list of variables, not in *Var*. We then define the *demonic miraculous assignment* and its dual, the *angelic strict assignment*, as follows:

$$\langle \wedge u := u'.\ P \rangle \stackrel{\text{def}}{=} \bigwedge_d ([u = d]; \langle \wedge u.\ P[d, u/u, u'] \rangle),$$

$$\langle \vee u := u'.\ P \rangle \stackrel{\text{def}}{=} \bigvee_d (\{u = d\}; \langle \vee u.\ P[d, u/u, u'] \rangle).$$

Here the formula P may refer to the variables u and to u', the latter standing for the new values of u. In this way we indicate how the new values of u are to be related to the old values. The predicate transformers of these commands are as follows:

$$\langle \wedge u := u'.\ P \rangle(Q) = \forall u'.\ P \Rightarrow Q[u'/u],$$

$$\langle \vee u := u'.\ P \rangle(Q) = \exists u'.\ P \wedge Q[u'/u].$$

The ordinary multiple assignment command is defined using, e.g., the demonic assignment:

$$u := e \stackrel{\text{def}}{=} \langle \wedge u := u'.(u' = e) \rangle$$

determining the predicate transformer $(u := e)(Q) = Q[e/u]$.

Conditional composition, recursion, and iteration

The *conditional composition* is defined as follows:

$$\textbf{if } ([]_{i \in I}\ b_i \to S_i) \textbf{ fi} \stackrel{\text{def}}{=} \left\{ \bigvee_{i \in I} b_i \right\}; \bigwedge_{i \in I} ([b_i]; S_i).$$

Let X be a command variable and let $T(X)$ be a command constructed out of X together with the basic commands and constructors of $\mathcal{C}$. Then $\lambda X.\ T(X)$ is a monotonic function on a complete lattice. Thus the least fixed point of this function exists in $\mathcal{C}$. We let the *recursive composition* $\mu X.\ T(X)$ denote this least fixpoint.

The *iteration command* can be defined using recursion,

$$\textbf{do } b \to S \textbf{ od} = \mu X.\ ([b]; S; X \wedge [\neg b]).$$

3. Inverse commands

A *true inverse* of a command $S \in \mathcal{C}$ is a command S^{-1} that satisfies

$$S^{-1};S \;=\; skip \;=\; S;S^{-1},$$

i.e., a command that computes the input to S given the output. A true inverse of S exists if and only if S is bijective. The set of bijective commands form a subset of $\mathcal{C}^{\perp\top}_{\wedge\vee}$. This is a very restricted class of commands, making the usefulness of this notion of inverses rather limited.

We shall now define a more general notion of inverse command that permits an arbitrary command in $\mathcal{C}^{\top}_{\wedge}$ to be inverted. This notion of inverse turns out to be useful for describing coordinate transformations and data refinement. We say that S^{-1} is the *inverse* of S if

$$S^{-1};S \;\leqslant\; skip \;\leqslant\; S;S^{-1}.$$

Our definition means that S^{-1} is what is in category theory known as the *left adjoint* of S (however, we will not assume that the reader is familiar with category theory). We note in passing that we could construct a dual theory by using refinements in the opposite direction in the definition above.

3.1. Properties of inverse commands

The following theorem shows when inverse commands exist.

Theorem 3.1 (Existence and uniqueness). *Let S be a command in $\mathcal{C}$.*

(a) *S^{-1} is unique if it exists.*
(b) *S^{-1} exists if and only if $S \in \mathcal{C}^{\top}_{\wedge}$.*
(c) *$S^{-1} \in \mathcal{C}^{\perp}_{\vee}$ if this inverse exists.*

Proof. These results are well known in category theory. Thus we just give an outline of a non-categorical proof.

(a) Assume that $S';S \leqslant skip \leqslant S;S'$ and $S'';S \leqslant skip \leqslant S;S''$. Then

$$S' \;=\; S';skip \;\leqslant\; S';S;S'' \;\leqslant\; skip;S'' \;=\; S''$$

and $S'' \leqslant S'$ by symmetry.

(b) The if part is easily proved by showing that $\bigwedge\{Q \mid P \leqslant S(Q)\}$ is an inverse of S.

For the only-if part, we assume that S has an inverse S^{-1}. Then

$$
\begin{aligned}
& S\Big(\bigwedge_i Q_i\Big) \\
& \quad \geqslant \quad \text{[definition of inverse]} \\
& S\Big(\bigwedge_i S^{-1}(S(Q_i))\Big) \\
& \quad \geqslant \quad \text{[if } S \text{ is monotonic then } S(\textstyle\bigwedge Q_i) \leqslant \bigwedge S(Q_i)\text{]} \\
& S\Big(S^{-1}\Big(\bigwedge_i S(Q_i)\Big)\Big) \\
& \quad \geqslant \quad \text{[definition of inverse]} \\
& \bigwedge_i S(Q_i)
\end{aligned}
$$

Since $S(\bigwedge_i Q_i) \leqslant \bigwedge_i S(Q_i)$ holds by monotonicity, this proves part (b).

(c) This can be proved in the same way as part (b) above. □

We have the following alternative characterisation of inverse commands.

Theorem 3.2. *Let S be a command in $\mathcal{C}$. Then*

(a) *S^{-1} is the least solution (in $\mathcal{C}$) to the equation $skip \leqslant S; X$ in command variable X.*

(b) *S^{-1} is the greatest solution (in $\mathcal{C}$) to the equation $S; X \leqslant skip$ in command variable X.*

Proof. We prove only part (a) as the proof of part (b) is similar. By definition, S^{-1} is a solution to the equation $skip \leqslant S; X$. Now assume that S' is another solution to this equation. Then

$$S^{-1} \;=\; S^{-1}; skip \;\leqslant\; S^{-1}; S; S' \;\leqslant\; skip; S' \;=\; S',$$

so S^{-1} is the least solution. □

Relational interpretation of inverse commands

The full command language $\mathcal{C}$ is too rich to permit a simple relational interpretation. However, both the sublanguages $\mathcal{C}_\wedge^\top$ and $\mathcal{C}_\vee^\perp$ have a simple relational interpretation. We shall now show how the relational interpretations of inverse commands are related to each other.

A command S in $\mathcal{C}_\wedge^\top$ can be interpreted as a state transformer f_S (i.e., a function from Σ to the powerset $\mathcal{P}(\Sigma)$) as follows:

$$\sigma \in S(Q) \quad \Leftrightarrow \quad f_S(\sigma) \subseteq Q,$$

where $\sigma \in S(Q)$ means that $S(Q)$ holds in σ (this is correct since we can always treat predicates as sets of states). Similarly, a command S in $\mathcal{C}_{\vee}^{\perp}$ can be interpreted as a state transformer g_S defined as follows:

$$\sigma \in S(Q) \quad \Leftrightarrow \quad g_S(\sigma) \cap Q \neq \emptyset.$$

We shall now show that f_S and $g_{S^{-1}}$, viewed as relations on Σ, are inverse relations. This is seen as follows. In the proof of Theorem 3.1 it was noted that

$$S^{-1}(P) \;=\; \bigwedge\{Q \mid P \leqslant S(Q)\} \tag{8}$$

for $S \in \mathcal{C}_{\wedge}^{\top}$ and arbitrary predicate P. Then (treating predicates as sets of states)

$$\begin{aligned}
&\sigma \in g_{S^{-1}}(\sigma') \\
&\quad \Leftrightarrow \quad [\text{set theory}] \\
&g_{S^{-1}}(\sigma') \cap \{\sigma\} \neq \emptyset \\
&\quad \Leftrightarrow \quad [\text{definition of } g_{S^{-1}}] \\
&\sigma' \in S^{-1}(\{\sigma\}) \\
&\quad \Leftrightarrow \quad [(8)] \\
&\sigma' \in \bigwedge\{Q \mid \sigma \in S(Q)\} \\
&\quad \Leftrightarrow \quad [\text{definition of } f_S] \\
&\sigma' \in \bigwedge\{Q \mid f_S(\sigma) \subseteq Q\} \\
&\quad \Leftrightarrow \quad [\text{set theory}] \\
&\sigma' \in f_S(\sigma).
\end{aligned}$$

Inverses and duals

The inverse construct is antimonotonic with respect to the refinement relation, as the following lemma shows.

Lemma 3.3. *If* $S_1 \leqslant S_2$, *then* $S_2^{-1} \leqslant S_1^{-1}$.

Proof. Assume that $S_1 \leqslant S_2$. Then

$$\begin{aligned}
S_2^{-1} \;&=\; S_2^{-1}; skip \;\leqslant\; S_2^{-1}; S_1; S_1^{-1} \\
&\leqslant\; S_2^{-1}; S_2; S_1^{-1} \;\leqslant\; skip; S_1^{-1} \;=\; S_1^{-1}. \qquad \square
\end{aligned}$$

In many respects inverses resemble duals. In fact, both are lattice isomorphisms from $(\mathcal{C}_{\wedge}^{\top}, \leqslant)$ to $(\mathcal{C}_{\vee}^{\perp}, \geqslant)$. Inverses and duals also commute, in the following sense.

Lemma 3.4. *If* $S \in \mathcal{C}_{\wedge}^{\top}$, *then* $((S^{-1})^{\circ})^{-1} = S^{\circ}$.

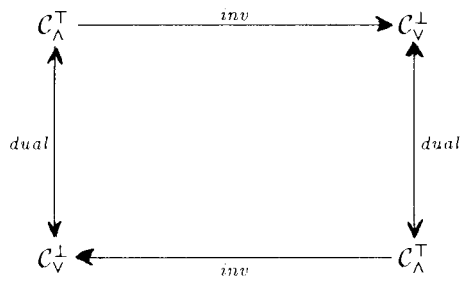

Fig. 1. Inverses and duals.

Proof. Since $skip^\circ = skip$, we have by (1) and (4):

$$S^{-1};S \leqslant skip \leqslant S;S^{-1} \quad \Rightarrow \quad S^\circ;(S^{-1})^\circ \leqslant skip \leqslant (S^{-1})^\circ;S^\circ$$

showing that S° is the inverse of $(S^{-1})^\circ$. □

Thus the diagram in Fig. 1 commutes (where *inv* denotes taking inverses and *dual* denotes taking duals).

3.2. Computing the inverse of a command

The following theorem shows that inverses of commands can be computed compositionally.

Theorem 3.5. *Let u be a list of variables, d a list of values, and S_i commands in $\mathcal{C}_\wedge^\top$. Then*

$$[P]^{-1} = \{P\},$$

$$\langle u \leftarrow d\rangle^{-1} = \{u = d\};\langle \vee u.\ true\rangle,$$

$$(S_1;S_2)^{-1} = S_2^{-1};S_1^{-1},$$

$$\left(\bigwedge S_i\right)^{-1} = \bigvee S_i^{-1}.$$

Proof. For the first case, we have

$$\{P\}([P](Q)) = P \wedge (P \Rightarrow Q) = P \wedge Q \leqslant Q$$

and

$$[P](\{P\}(Q)) = P \Rightarrow (P \wedge Q) = \neg P \vee Q \geqslant Q$$

showing that $\{P\};[P] \leqslant skip \leqslant [P];\{P\}$.

For the second case, we have that

$$
\begin{aligned}
&(\{u = d\}; \langle \vee u.\ true\rangle; \langle u \leftarrow d\rangle)(Q)\\
&\quad = \quad [\text{definitions}]\\
&(u = d) \wedge \exists u.\ Q[d/u]\\
&\quad = \quad [Q[d/u] \text{ does not depend on } u]\\
&(u = d) \wedge Q[d/u]\\
&\quad \leqslant \quad [\text{general property of existential quantification}\\
&\qquad\quad \text{over a nonempty domain}]\\
&\exists u.\ (u = d) \wedge Q[d/u]\\
&\quad = \quad [\text{one-point rule of predicate calculus}]\\
&Q
\end{aligned}
$$

and a similar calculation shows that $(\langle u \leftarrow d\rangle; \{u = d\}; \langle \vee u.\ true\rangle)(Q) \geqslant Q$ holds.

For sequential composition we have that

$$S_2^{-1}; S_1^{-1}; S_1; S_2 \leqslant S_2^{-1}; skip; S_2 = S_2^{-1}; S_2 \leqslant skip$$

by the definition of inverses and the properties of *skip*. In the same way, one proves $skip \leqslant S_1; S_2; S_2^{-1}; S_1^{-1}$.

Finally, for demonic choice we have that

$$
\begin{aligned}
&\Big(\bigvee_i S_i^{-1}\Big); \Big(\bigwedge_i S_i\Big)\\
&\quad = \quad [\text{distributivity properties of commands}]\\
&\bigvee_i \Big(S_i^{-1}; \Big(\bigwedge_j S_j\Big)\Big)\\
&\quad \leqslant \quad [\text{a meet is less than all its elements}]\\
&\bigvee_i (S_i^{-1}; S_i)\\
&\quad \leqslant \quad [\text{every disjunct is refined by } skip]\\
&skip
\end{aligned}
$$

and similarly that $skip \leqslant (\bigwedge_i S_i); (\bigvee_i S_i^{-1})$. □

Because of the completeness result in Lemma 2.1, we see that Theorem 3.5 shows how every inverse can be computed compositionally. For example, we have the following inverses:

$$skip^{-1} = skip,$$

$$magic^{-1} = abort.$$

3.3. Inverses of program constructs

Applying Theorem 3.5, we now compute inverses of some program constructs that are invertible.

Lemma 3.6.

$$\langle \wedge u.\ P\rangle^{-1} \ = \ \{P\};\langle \vee u.\ true\rangle,$$

$$\langle u := u'.\ P\rangle^{-1} \ = \ \bigvee_d (\{P[d,u/u,u']\};\langle u \leftarrow d\rangle).$$

Proof. First, straightforward calculations show that the inverse of $\langle \wedge u.\ true\rangle$ is $\langle \vee u.\ true\rangle$. Now we have

$$\begin{array}{l}
\langle \wedge u.\ P\rangle^{-1} \\
\quad = \quad [(6)] \\
(\langle \wedge u.\ true\rangle;[P])^{-1} \\
\quad = \quad [\text{above; Theorem 3.5}] \\
\{P\};\langle \vee u.\ true\rangle
\end{array}$$

completing the proof of the first part. For the demonic miraculous assignment, we have

$$\begin{array}{l}
\langle \wedge u := u'.\ P\rangle^{-1} \\
\quad = \quad [\text{definition of demonic miraculous assignment}] \\
\left(\bigwedge_d ([u = d];\langle \wedge u.\ P[d,u/u,u']\rangle)\right)^{-1} \\
\quad = \quad [\text{Theorem 3.5; first part of this lemma}] \\
\bigvee_d (\{P[d,u/u,u']\};\langle \vee u.\ true\rangle;\{u = d\}) \\
\quad = \quad [\text{straightforward calculation shows that} \\
\qquad\quad \langle \vee u.\ true\rangle;\{u = d\} = \langle u \leftarrow d\rangle] \\
\bigvee_d (\{P[d,u/u,u']\};\langle u \leftarrow d\rangle). \qquad \square
\end{array}$$

As an example, we can use Lemma 3.6 to compute the inverse of the ordinary assignment:

$$(u := e)^{-1} \ = \ \bigvee_d (\{u = e[d/u]\};\langle u \leftarrow d\rangle). \tag{9}$$

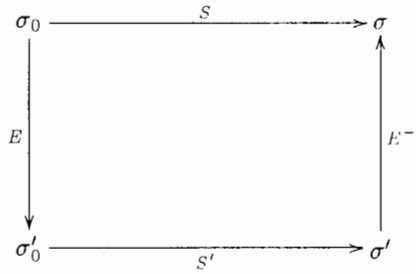

Fig. 2. Encoding and decoding.

3.4. Coordinate transformation and refinement

Consider two commands, S and S'. We want to model the intuitive idea that S' is constructed from S by changing the way in which the program state is represented, i.e., by a coordinate transformation on the state space.

Encoding and decoding

The basic idea is to introduce an *encoding* command E that computes the representation σ' of each state σ. We require that $E \in \mathcal{C}^{\top}_{\wedge}$, i.e., it is always terminating and demonic (but it may be miraculous). The inverse of E is the *decoding* command E^{-1}.

We say that a command S is *refined by* S' *through the encoding* E, denoted $S \leqslant_E S'$, if

$$S \leqslant E;S';E^{-1}. \tag{10}$$

If S is regarded as a specification and S' as an implementation, then $\leqslant_E$ can be regarded as a simulation relation, as illustrated by the diagram in Fig. 2.

We note that by the properties of inverse commands, the following characterisation is equivalent to (10)

$$E^{-1};S;E \leqslant S'. \tag{11}$$

In [3] we extend the command language with commands that introduce and delete variables from the state. We investigate the refinement relation $\leqslant_E$ in more detail, showing how it is useful for describing data refinement.

Example

We shall illustrate the idea of a coordinate transformation by a small example. Consider a program S working on the global variables (polar coordinates in the plane) ϕ and r, with the restrictions $0 \leqslant \phi < 2\pi$ and $r \geqslant 0$. The coordinate transformation that we are interested in is a reflection in the unit circle. It can be expressed as an encoding command

$$E: \quad [r > 0]r := 1/r.$$

Calculating the inverse yields

$$E^{-1}: \quad \{r > 0\}r := 1/r.$$

(Note the treatment of the origin where the transformation is undefined: E terminates miraculously while E^{-1} aborts.) We consider the following example commands:

$$S_1: \quad \phi, r := \phi + \tfrac{1}{2}\pi, 2r,$$

$$S_2: \quad \{r > 1\}; r := r - 1,$$

and try to determine commands S_i' such that $S_i \leqslant_E S_i'$ for $i = 1, 2$. By (11), we can choose $E^{-1}; S_i; E$ (or any command that refines this command). Straightforward calculations yield the following commands:

$$S_1': \quad \phi, r := \phi + \tfrac{1}{2}\pi, \tfrac{1}{2}r,$$

$$S_2': \quad \{r < 1\}; r := r/(1 - r).$$

We show the calculation for S_2:

$$\begin{aligned}
&(E^{-1}; S_2; E)(Q) \\
&\quad = r > 0 \wedge (r > 1 \wedge (r > 0 \Rightarrow Q[^{1/r}/r])[^{r-1}/r])[^{1/r}/r] \\
&\quad = r > 0 \wedge r < 1 \wedge (r < 1 \Rightarrow Q[^{r/(1-r)}/r]) \\
&\quad = r > 0 \wedge r < 1 \wedge Q[^{r/(1-r)}/r] \\
&\quad \leqslant r < 1 \wedge Q[^{r/(1-r)}/r],
\end{aligned}$$

where Q is an arbitrary predicate. The calculation shows that we could also have chosen S_2' to be the command $\{0 < r < 1\}; r := r/(1 - r)$.

4. Generalised inverses

For general conjunctive commands $S \in \mathcal{C}_\wedge$ there need not always exist any S' such that $skip \leqslant S; S'$ is satisfied. This is because *skip* always terminates, while $S; S'$ does not terminate if S does not terminate. However, we can get around the nontermination of S if we weaken the requirement on inverses as follows. We say that $S^- \in \mathcal{C}_\vee^\perp$ is a *generalised inverse* of S, if

$$S^-; S \leqslant skip \quad \text{and} \quad \{S(true)\} \leqslant S; S^-.$$

This does not define the generalised inverse uniquely. For example, choosing $S = abort$, we have the requirements

$$S^-; abort \leqslant skip \quad \text{and} \quad \{abort(true)\} \leqslant abort; S^-$$

and any non-miraculous command S^- satisfies these conditions.

4.1. Existence of generalised inverses

We first show that only conjunctive commands can have generalised inverses.

Lemma 4.1. *If the command S has a generalised inverse, then S is conjunctive.*

Proof. Assume that S^- is a generalised inverse of S and that $\{Q_i\}$ is a nonempty set of predicates. Then

$$
\begin{aligned}
& S\Big(\bigwedge_i Q_i\Big) \\
& \quad \geqslant \quad \text{[definition of generalised inverse]} \\
& S\Big(\bigwedge_i S^-(S(Q_i))\Big) \\
& \quad \geqslant \quad \text{[if } S \text{ is monotonic then } S(\textstyle\bigwedge_i(Q_i)) \leqslant \bigwedge_i S(Q_i)\text{]} \\
& S\Big(S^-\Big(\bigwedge_i S(Q_i)\Big)\Big) \\
& \quad \geqslant \quad \text{[definition of generalised inverse]} \\
& S(\mathit{true}) \wedge \Big(\bigwedge_i S(Q_i)\Big) \\
& \quad = \quad \text{[distributivity property for nonempty conjunctions]} \\
& \bigwedge_i (S(\mathit{true}) \wedge S(Q_i)) \\
& \quad = \quad \text{[monotonicity]} \\
& \bigwedge_i S(Q_i).
\end{aligned}
$$

Since $S(\bigwedge_i Q_i) \leqslant \bigwedge_i S(Q_i)$ follows from monotonicity, this proves the lemma. □

The following result shows how generalised inverses are closely connected to the inverses considered in the preceding section.

Theorem 4.2. *Assume that $S \in \mathcal{C}_\wedge$. Then S has a least generalised inverse which is*

$$([S(\mathit{true})];S)^{-1}.$$

Proof. Let $S \in \mathcal{C}_\wedge$ and let $S' = ([S(\mathit{true})];S)^{-1}$. Then

$$
\begin{aligned}
& \mathit{true} \\
& \quad \Leftrightarrow \quad \text{[definition of inverse]}
\end{aligned}
$$

$$
\begin{aligned}
& S'; ([S(true)]; S) \leqslant skip \\
& \quad \Leftrightarrow \quad \text{[definition of refinement]} \\
& S'(\neg S(true) \vee S(Q)) \leqslant Q \quad \text{for all } Q \\
& \quad \Leftrightarrow \quad [S' \text{ is disjunctive}] \\
& S'(\neg S(true)) \vee S'(S(Q)) \leqslant Q \quad \text{for all } Q.
\end{aligned}
$$

Thus the following holds for all predicates Q:

$$S'(\neg S(true)) \leqslant Q, \tag{12}$$

$$S'(S(Q)) \leqslant Q. \tag{13}$$

Also,

$$
\begin{aligned}
& \{S(true)\} \leqslant S; S' \\
& \quad \Leftrightarrow \quad \text{[definition of assert command]} \\
& S(true) \wedge Q \leqslant S(S'(Q)) \quad \text{for all } Q \\
& \quad \Leftrightarrow \quad \text{[general property of complete lattices]} \\
& Q \leqslant \neg S(true) \vee S(S'(Q)) \quad \text{for all } Q \\
& \quad \Leftrightarrow \quad \text{[definition of refinement]} \\
& skip \leqslant ([S(true)]; S); S' \\
& \quad \Leftrightarrow \quad \text{[definition of inverse]} \\
& true.
\end{aligned}
$$

Since (13) implies that $S'; S \leqslant skip$, we have shown that S' is in fact a generalised inverse of S.

To prove that S' is the least generalised inverse of S, we first note that choosing $Q = false$ in (12) we get

$$S'(\neg S(true)) = false. \tag{14}$$

Next,

$$
\begin{aligned}
& S'([S(true)](Q)) \\
& \quad = \quad \text{[definition of assumption command]} \\
& S'(\neg S(true) \vee Q) \\
& \quad = \quad [S' \text{ is disjunctive}] \\
& S'(\neg S(true)) \vee S'(Q) \\
& \quad = \quad [(14)] \\
& S'(Q).
\end{aligned}
$$

So we get

$$S'; [S(true)] = S'. \tag{15}$$

Now assume that S'' is another generalised inverse of S. Then we have

$S'; skip$
$\leqslant$ [$skip \leqslant [P]; \{P\}$ holds for all predicates P]
$S'; [S(true)]; \{S(true)\}$
$=$ [(15)]
$S'; \{S(true)\}$
$\leqslant$ [S'' is a generalised inverse]
$S'; S; S''$
$\leqslant$ [S' is a generalised inverse]
$skip; S'$.

So $S' \leqslant S''$ and S' is in fact the least generalised inverse of S. □

Since $\{S(true)\} = \{true\} = skip$ for $S \in \mathcal{C}_\wedge^\top$ we have that inverses and generalised inverses coincide in $\mathcal{C}_\wedge^\top$:

Corollary 4.3. *If $S \in \mathcal{C}_\wedge^\top$, then S^{-1} is the unique generalised inverse of S.*

Thus the generalised inverse is really a generalisation of the concept of inverse. A generalised inverse of S acts as an inverse whenever S is terminating.

4.2. Computing generalised inverses

We shall now show how generalised inverses can be computed, for all conjunctive commands.

Theorem 4.4. *Let S_i^- be any generalised inverse of S_i, for all i in some index set. Then*

(a) $\{P\}$ *is the least generalised inverse of* $\{P\}$,
(b) $\{P\}$ *is the least generalised inverse of* $[P]$,
(c) $\{u = d\}; \langle \forall u.\ true \rangle$ *is a generalised inverse of* $\langle u \leftarrow d \rangle$,
(d) $S_2^-; S_1^-$ *is a generalised inverse of* $S_1; S_2$,
(e) $\bigvee_i S_i^-$ *is a generalised inverse of* $\bigwedge_i S_i$.

Proof. Part (a) follows from Theorem 4.2 and the following calculation:

$$[\{P\}(true)]; \{P\} = [P]; \{P\} = [P].$$

Parts (b) and (c) follow immediately from Theorem 3.5 and Corollary 4.3.

We now prove part (d). First,

$$S_2^-;S_1^-;S_1;S_2 \leqslant skip$$

is shown as in the proof of Theorem 3.5. Next, we note that

$$S;\{P\} = \{S(P)\};S \quad \text{if } S \text{ is conjunctive} \tag{16}$$

since

$$(S;\{P\})(Q) = S(P \wedge Q) = S(P) \wedge S(Q) = (\{S(P)\};S)(Q).$$

Now,

$$\begin{aligned}
&S_1;S_2;S_2^-;S_1^- \\
&\quad \geqslant \quad [\text{definition of generalised inverse}] \\
&S_1;\{S_2(true)\};S_1^- \\
&\quad = \quad [(16)] \\
&\{S_1(S_2(true))\};S_1;S_1^- \\
&\quad \geqslant \quad [\text{definition of generalised inverse}] \\
&\{S_1(S_2(true))\};\{S_1(true)\} \\
&\quad = \quad [\text{definition of sequential composition and assert command}] \\
&\{S_1(S_2(true)) \wedge S_1(true)\} \\
&\quad = \quad [\text{monotonicity}] \\
&\{S_1(S_2(true))\} \\
&\quad = \quad [\text{definition of sequential composition}] \\
&\{(S_1;S_2)(true)\},
\end{aligned}$$

which completes the proof of the part (d).

Finally we prove part (e). First,

$$\left(\bigvee_i S_i^-\right);\left(\bigwedge_i S_i\right) \leqslant skip$$

is shown as in the proof of Theorem 3.5. Furthermore, we have

$$\begin{aligned}
&\left(\bigwedge_i S_i\right);\left(\bigvee_i S_i^-\right) \\
&\quad \geqslant \quad [\text{distributivity properties of commands}] \\
&\bigwedge_i \left(S_i;\left(\bigvee_j S_j^-\right)\right) \\
&\quad \geqslant \quad [\text{a join is greater than all of its elements}]
\end{aligned}$$

$$\bigwedge_i (S_i; S_i^-)$$

$\geqslant$ [definition of generalised inverse]

$$\bigwedge_i (\{S_i(true)\})$$

$=$ [definitions of assert and demonic choice]

$$\left\{\left(\bigwedge_i S_i\right)(true)\right\}$$

completing the proof. □

Since every command in $\mathcal{C}_\wedge$ can be constructed using assertions, assumptions, and store commands and the constructors ";" and "$\wedge$" (Lemma 2.1), we have shown how to calculate a generalised inverse to every conjunctive command in a compositional way.

Simple examples show that the rules of Theorem 4.4 cannot be used to compute *least* generalised inverses, e.g., $S_2^- \vee S_1^-$ need not be the least generalised inverse of $S_1 \wedge S_2$, even if S_1^- and S_2^- are least generalised inverses. To see this, set $S_1 = abort$ and $S_2 = skip$. Similarly, $S_2^-; S_1^-$ need not give the least generalised inverse of $S_1; S_2$ (choose S_1 to be $x := 0 \wedge x := 1$ and S_2 to be $([x = 0]; skip) \wedge ([x = 1]; abort)$). In both cases the lack of compositionality is caused by the fact that a demonic choice between nontermination and termination is in fact no choice at all, since nontermination is always chosen.

Generalised inverses of command constructors

If $T(X)$ is an expression built up of commands from $\mathcal{C}_\wedge$ and the symbol X, then $T = \lambda X.\ T(X)$ is a command constructor on $\mathcal{C}_\wedge$. We define a command constructor T^- on $\mathcal{C}_\vee^\perp$ to be a generalised inverse of T if, for all $S \in \mathcal{C}_\wedge$, $T^-(S^-)$ is a generalised inverse of $T(S)$ whenever S^- is a generalised inverse of S. We can compute a generalised inverse of a command constructor T by computing a generalised inverse for the expression $T(X)$ using Theorem 4.4, but leaving X unchanged. For example, if S^- is a generalised inverse of S and

$$T(X) \;=\; [b]S;\ X \wedge [\neg b],$$

then T^- is a generalised inverse of T, where

$$T^-(X) \;=\; X; S^-; \{b\} \vee \{\neg b\}.$$

4.3. Generalised inverses of program constructs

We now compute generalised inverses for program constructs which are conjunctive but not always terminating.

Lemma 4.5. *Assume that S_i^- is a generalised inverse of S_i. Then $\bigvee_i(S_i^-;\{b_i\})$ is a generalised inverse of* **if** $(\square_i\ b_i \to S_i)$ **fi**. *Furthermore, $\bigvee_i(S_i^{-1};\{b_i\})$ is the least generalised inverse of* **if** $(\square_i\ b_i \to S_i)$ **fi** *if all S_i are in $\mathcal{C}_\wedge^\top$.*

Proof. Let IF denote the command **if** $(\square_i\ b_i \to S_i)$ **fi**. By the definition of conditional composition,

$$IF = \left\{\bigvee_i b_i\right\};\bigwedge_i([b_i];S_i).$$

Theorem 4.4 yields the following generalised inverse:

$$IF^- = \left(\bigvee_i(S_i^-;\{b_i\})\right);\left\{\bigvee_i b_i\right\} = \bigvee_i(S_i^-;\{b_i\}).$$

Finally assume that all S_i are in $\mathcal{C}_\wedge^\top$. Then $IF(true) = \bigvee_i b_i$ and

$$[IF(true)];IF = \left[\bigvee_i b_i\right];\left\{\bigvee_i b_i\right\};\bigwedge_i([b_i];S_i) = \bigwedge_i([b_i];S_i)$$

having the inverse $\bigvee_i(S_i^{-1};\{b_i\})$. Thus by Theorem 4.2, $\bigvee_i(S_i^{-1};\{b_i\})$ is the least generalised inverse of IF. □

Generalised inverse of recursion

We now consider recursion. Let T be a command constructor on $\mathcal{C}_\wedge$ and define T_α for all ordinals α as follows:

$$T_0 = abort,$$

$$T_{\alpha+1} = T(T_\alpha),$$

$$T_\alpha = \bigvee_{\beta<\alpha} T_\beta \quad \text{for limit ordinals } \alpha.$$

It is well known that there exists an ordinal α_T such that

$$\mu X.\ T(X) = T_{\alpha_T}.$$

The following lemma now shows how to construct a generalised inverse to $\mu X.\ T(X)$.

Lemma 4.6. *Assume that T is a command constructor on $\mathcal{C}_\wedge$ and let T^- be a generalised inverse of T. Then $\mu X.\ T^-(X)$ is a generalised inverse of $\mu X.\ T(X)$.*

Proof. We define T_α^- analogously with T_α for all ordinals α:

$$T_0^- = abort,$$

$$T_{\alpha+1}^- = T^-(T_\alpha^-),$$

$$T_\alpha^- = \bigvee_{\beta<\alpha} T_\beta^- \quad \text{for limit ordinals } \alpha.$$

Since command constructors are by definition monotonic, we have by induction that $T_\beta \leqslant T_\alpha$ and $T_\beta^- \leqslant T_\alpha^-$ if $\beta \leqslant \alpha$. By ordinal induction it is proved that T_α^- is a generalised inverse of T_α, for all ordinals α. We show the induction argument for limit ordinals α. First,

$$\begin{aligned}
&\left(\bigvee_{\beta<\alpha} T_\beta\right);\left(\bigvee_{\beta<\alpha} T_\beta^-\right)\\
&\quad \geqslant \quad \text{[distributivity properties of commands]}\\
&\bigvee_{\beta<\alpha}\left(T_\beta;\bigvee_{\gamma<\alpha} T_\gamma^-\right)\\
&\quad \geqslant \quad \text{[a join is greater than all its elements]}\\
&\bigvee_{\beta<\alpha}(T_\beta;T_\beta^-)\\
&\quad \geqslant \quad \text{[definition of generalised inverse]}\\
&\bigvee_{\beta<\alpha}\{T_\beta(true)\}\\
&\quad = \quad \text{[definition of assert command;}\\
&\qquad\quad \text{meet distributes over arbitrary joins]}\\
&\left\{\bigvee_{\beta<\alpha} T_\beta(true)\right\}\\
&\quad = \quad \text{[definition of angelic choice]}\\
&\left\{\left(\bigvee_{\beta<\alpha} T_\beta\right)(true)\right\}.
\end{aligned}$$

Second,

$$\begin{aligned}
&\left(\bigvee_{\beta<\alpha} T_\beta^-\right);\left(\bigvee_{\beta<\alpha} T_\beta\right)\\
&\quad = \quad \text{[distributivity and disjunctivity]}\\
&\bigvee_{\beta<\alpha}\bigvee_{\gamma<\alpha}(T_\beta^-;T_\gamma).
\end{aligned}$$

When $\beta \leqslant \gamma$, we have

$$T_\beta^-; T_\gamma \;\leqslant\; T_\gamma^-; T_\gamma \;\leqslant\; skip,$$

and when $\gamma < \beta$, we have

$$T_\beta^-; T_\gamma \;\leqslant\; T_\beta^-; T_\beta \;\leqslant\; skip.$$

Thus

$$\Big(\bigvee_{\beta<\alpha}\bigvee_{\gamma<\alpha} T_\beta^-; T_\gamma\Big) \;\leqslant\; \Big(\bigvee_{\beta<\alpha}\bigvee_{\gamma<\alpha} skip\Big) \;=\; skip,$$

which finishes the induction argument.

Since T_α^- is a generalised inverse of T_α for all ordinals α, this must also be true for the special ordinal α_T, meaning that $\mu X.\ T^-(X)$ is a generalised inverse of $\mu X.\ T(X)$. □

5. Strongest postcondition and generalised inverses

In this section we show how the concept of generalised inverse is closely related to the concept of strongest postcondition.

5.1. Strongest postconditions

The strongest postcondition of a statement S with respect to a precondition P is intuitively characterised in the following way [7,10]: $\mathrm{sp}(S, Q)$ is the strongest predicate such that execution of S with Q holding in the initial state implies that $\mathrm{sp}(S, Q)$ holds on termination. Strongest postconditions have been characterised inductively for simple nondeterministic languages by de Bakker [6] and Back [2]. In our notation, these characterisations amount to the following:

$$\mathrm{sp}(u := e, Q) \;=\; \exists v.\ Q[v/u] \wedge (u = e[v/u]), \tag{17}$$

$$\mathrm{sp}(\{P\}, Q) \;=\; P \wedge Q, \tag{18}$$

$$\mathrm{sp}([P], Q) \;=\; P \wedge Q, \tag{19}$$

$$\mathrm{sp}(S_1; S_2, Q) \;=\; \mathrm{sp}(S_2, \mathrm{sp}(S_1, Q)), \tag{20}$$

$$\mathrm{sp}(S_1 \wedge S_2, Q) \;=\; \mathrm{sp}(S_1, Q) \vee \mathrm{sp}(S_2, Q). \tag{21}$$

Furthermore, the strongest postcondition for recursion in the case of bounded nondeterminism is defined in [2] as follows:

$$\mathrm{sp}(\mu X.T(X), Q) \;=\; \bigvee_{n=0}^{\infty} \mathrm{sp}(T_n, Q), \tag{22}$$

where T_n is defined inductively for all natural numbers n:

$$T_0 = abort,$$

$$T_{n+1} = T(T_n).$$

Another characterisation of strongest postconditions is given by Back in [2]. He gives four postulates that characterise strongest postconditions:

$$\mathrm{sp}(S, false) = false, \tag{23}$$

$$\mathrm{sp}(S, P \vee Q) = \mathrm{sp}(S, P) \vee \mathrm{sp}(S, Q), \tag{24}$$

$$\mathrm{sp}(S, \mathrm{wp}(S, Q)) \leqslant Q, \tag{25}$$

$$P \leqslant \mathrm{wp}(S, true) \Rightarrow P \leqslant \mathrm{wp}(S, \mathrm{sp}(S, P)). \tag{26}$$

Strongest postconditions are used in [2] to give a first-order characterisation of the refinement relation: the refinement relation $S \leqslant S'$ holds if and only if the following total correctness formula is valid:

$$\mathrm{wp}(S, true) \wedge (u = u_0)\ [S']\ \mathrm{sp}(S, (u = u_0)), \tag{27}$$

where u_0 is a list of fresh variables corresponding to the list of program variables u.

5.2. Strongest postconditions and generalised inverses

We now show that for an arbitrary conjunctive command S, $S^-(Q)$ has the same properties as $\mathrm{sp}(S, Q)$. The correspondence between strongest postcondition and generalised inverse is not complete since the generalised inverse is not uniquely defined. However, it turns out that the characterisation theorem for the refinement relation can be formulated using generalised inverses instead of strongest postconditions.

We first note that all the following are immediate consequences of Theorems 4.4 and 3.5, Lemmas 4.5 and 4.6, and (9), slightly abusing the generalised inverse notation.

$$(u := e)^-(Q) = \bigvee_d (Q[d/u] \wedge u = e[d/u]), \tag{28}$$

$$\{P\}^-(Q) = P \wedge Q, \tag{29}$$

$$[P]^-(Q) = P \wedge Q, \tag{30}$$

$$(S_1; S_2)^- = S_2^-; S_1^-, \tag{31}$$

$$(S_1 \wedge S_2)^- = S_1^- \vee S_2^-, \tag{32}$$

$$(\mu X.T(X))^- = \bigvee_{n=0}^{\infty} T_n^-, \tag{33}$$

where T_n^- is a generalised inverse of T_n, as defined in Section 4.2. Comparing (28)–(33) with equations (17)–(22), we see that generalised inverses have all the essential properties of strongest postcondition.

In order to check the correspondence between generalised inverses and strongest postcondition as regards postulates (23)–(26), we want to show the following:

$$S^-(\mathit{false}) = \mathit{false}, \tag{34}$$

$$S^-(P \vee Q) = S^-(P) \vee S^-(Q), \tag{35}$$

$$S^-(S(Q)) \leqslant Q, \tag{36}$$

$$P \leqslant S(\mathit{true}) \Rightarrow P \leqslant S(S^-(P)). \tag{37}$$

The first two conditions state that generalised inverses are non-miraculous and disjunctive, which is true by definition. Noting that condition (37) is equivalent to

$$P \wedge S(\mathit{true}) \leqslant S(S^-(P)),$$

we see that the last two conditions are just the definition of generalised inverse. Thus our generalised inverses match the postulates for strongest postconditions given in [2].

We finish by proving the characterisation theorem for refinements within the framework of generalised inverses.

Theorem 5.1. *Let S and S' be conjunctive commands and let u be a list containing all program variables in S and S'. Then $S \leqslant S'$ holds if and only if*

$$S(\mathit{true}) \wedge (u = u_0)\ [S']\ S^-(u = u_0),$$

where u_0 is a list of fresh variables of the same length as u and S^- is any generalised inverse of S.

Proof. We have to show that $S \leqslant S'$ if and only if $S(\mathit{true}) \wedge (u = u_0) \leqslant S'(S^-(u = u_0))$. Let S^- be a generalised inverse of S and assume that $S \leqslant S'$. Then the definition of generalised inverses and monotonicity give that

$$S(\mathit{true}) \wedge (u = u_0) \leqslant S(S^-(u = u_0)) \leqslant S'(S^-(u = u_0)),$$

which proves the if part.

To prove the only-if part, assume that $S(true) \wedge (u = u_0) \leqslant S'(S^-(u = u_0))$ and let Q be an arbitrary predicate. Now, if we can assume that

$$(u = u_0) \leqslant S(Q) \tag{38}$$

holds, then

$$S^-(u = u_0) \leqslant S^-(S(Q)) \leqslant Q$$

and by the assumption we have

$$S(true) \wedge (u = u_0) \leqslant S'(S^-(u = u_0)) \leqslant S'(Q)$$

from which we can conclude

$$(u = u_0) \leqslant S'(Q)$$

since assumption (38) implies that $(u = u_0) \leqslant S(true)$.

We have now shown that

$$(u = u_0) \leqslant S(Q) \Rightarrow (u = u_0) \leqslant S'(Q)$$

for arbitrary u_0. Taking the join over all lists of values d such that $S(Q)$ holds in the state σ_d defined by $\sigma_d(u) = d$, we have

$$S(Q) = \bigvee_{d:S(Q)(\sigma_d)} (u = d) \leqslant S'(Q)$$

and thus $S \leqslant S'$, completing the proof of the only-if part. □

The importance of Theorem 5.1 lies in the fact that it gives us a first-order condition for checking refinement between commands. Compared to the formulation in [2], our version of the theorem does not need additional postulates of strongest postcondition. Instead, we use generalised inverses, which can be computed directly in the command language.

6. Conclusion

The idea of program inversion goes back to Dijkstra [8] and Gries [10]. A program S^{-1} is the (true) inverse of the program S if it computes the input of S, given the output. This means that S is not invertible if its input is not defined uniquely by its output. Our work shows how the command lattice framework, introduced in [4], permits a rich theory of command inversion. By permitting angelic nondeterminism and miraculously terminating commands we can consider a program to be invertible even though its input is not uniquely determined by its output. In particular, we define a notion of inverse which permits every conjunctive and always terminating command to be inverted.

The inverse of a command S can intuitively be interpreted as the relational inverse, with angelic nondeterminism instead of demonic and with partiality interpreted as nontermination instead of miracles. Inverses are compositional in the sense that the inverse of an arbitrary command can be calculated by inverting its subcomponents separately. The properties of inverses make it possible to define a simulation relation between commands using inverses. This is generalised to cover data refinement in [3] where the command language is extended with commands that add and delete variables from the state space.

Recently, Dijkstra and Scholten [9] have defined a notion of *converse predicate transformers*, used to relate weakest liberal precondition and strongest postcondition (these are adjoints of each other). Essentially, t' is the converse of t, if t' is (in our terminology) the dual of the inverse of t. We have not used this approach, since we want to stay within the framework of weakest precondition (total correctness) semantics.

We generalised the notion of inverses by defining S^- to be the generalised inverse of a conjunctive possibly nonterminating command S if S^- inverts S whenever S terminates. Generalised inverses can be computed compositionally even though they are not unique. We also showed that generalised inverses have properties that make them behave as strongest postcondition predicate transformers. This lets us formulate the characterisation theorem for refinement without postulating separate properties for strongest postconditions.

The connection between inverse commands and program inversion in the traditional sense is investigated further in [15], where we show how generalised inverses can be used in a calculational theory of program inversion.

Acknowledgements

The work reported here was supported by the Finsoft III program sponsored by the Technology Development Centre of Finland. We thank the referees for their helpful comments.

References

[1] R.J.R. Back, *Correctness Preserving Program Refinements: Proof Theory and Applications*, Mathematical Center Tracts **131** (Mathematical Centre, Amsterdam, 1980).

[2] R.J.R. Back, A calculus of refinements for program derivations, *Acta Inform.* **25** (1988) 593–624.

[3] R.J.R. Back and J. von Wright, Command lattices, variable environments and data refinement, Reports on Computer Science and Mathematics 102, Åbo Akademi, Turku, Finland (1990).

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

[5] R.J.R. Back and J. von Wright, Combining angels, demons and miracles in program specifications, *Theoret. Comput. Sci.* **100** (1992) 365–383.

[6] J.W. de Bakker, *Mathematical Theory of Program Correctness* (Prentice-Hall, Englewood Cliffs, NJ, 1980).
[7] E.W. Dijkstra, *A Discipline of Programming* (Prentice–Hall International, Englewood Cliffs, NJ, 1976).
[8] E.W. Dijkstra, *Selected Writings on Computing: A Personal Perspective* (Springer, Berlin, 1981).
[9] E.W. Dijkstra and C.S. Scholten, *Predicate Calculus and Program Semantics* (Springer, Berlin, 1990).
[10] D. Gries, *The Science of Programming* (Springer, New York, 1981).
[11] C.C. Morgan, Data refinement by miracles, *Inform. Process. Lett.* **26** (1988) 243–246.
[12] C.C. Morgan, *Programming from Specifications* (Prentice-Hall, Englewood Cliffs, NJ, 1990).
[13] J.M. Morris, A theoretical basis for stepwise refinement and the programming calculus, *Sci. Comput. Programming* **9** (1987) 287–306.
[14] G. Nelson, A generalization of Dijkstra's calculus, *ACM Trans. Programming Languages Systems* **11** (1989) 517–561.
[15] J. von Wright, Program inversion in the refinement calculus, *Inform. Process. Lett.* **37** (2) (1991) 95–100.