Calculating Sharp Adaptation Rules

David A. Naumann

Stevens Institute of Technology, Hoboken, NJ 07030 USA

Abstract

Adaptation rules adapt the pre-post specification of a procedure to contexts where it is called. Such rules are important for practical reasons and necessary for completeness for languages with recursive procedures. A sharp rule is one that gives the weakest precondition with respect to a given postcondition. A number of rules have been proposed, most unsound or incomplete or non-sharp. Using refinement algebra, we clarify and extend the applicability of previously proposed sharp rules for total correctness, and show how further rules may be found.

Key words: program verification, proof rules, procedures, refinement, relation algebra, predicate transformers

0 Introduction

For reasoning about correctness of while-programs, the rules proposed by Hoare [10] have stood the test of time. But for total correctness of procedure calls, a number of different rules have appeared (e.g., [11,9,5,12]). There appears to be no consensus on the “right” rule, and some proposals even turn out to be unsound (e.g., [2], see [1]). The results reported in this note were found in an attempt to derive an adaptation rule—rather than pulling it from a magician’s hat—using tools from refinement calculus. This sheds new light on the subject, explaining and extending the applicability of recent proposals, and it brings to light a new form of specification statement.

Adaptation rules. For the moment, let us take for granted a semantics for commands and predicates. Say a triple \{ pre \} \{ S \} \{ post \} is valid if every computation of command \( S \) from a state satisfying \( pre \) terminates in a

---

1 This work was supported in part by the US NSF under grant INT-9813854.
2 E-mail: naumann@cs.stevens-tech.edu
state satisfying \textit{post}. A collection of rules is \textit{sound} if every provable triple is valid, and \textit{relatively complete} if every valid triple can be proved “using only the rules”. The last phrase is in quotes because the best we can hope for is completeness relative to reasoning about predicates; we assume that all valid implications can be proved. For while-programs, Hoare proposed one rule for each program construct, along with a rule of consequence that lets us deduce \{pre\} \; S \; \{post\} from \{pre\} \; S \; \{post\} provided the implications \textit{pre} \Rightarrow \textit{pre} and \textit{post} \Rightarrow \textit{post}' are valid. These rules are relatively complete for programs without procedures [2].

For a procedure, modular reasoning requires verification with respect to a single specification that can be adapted to various calling contexts. Rules for such adaptation are subject to a number of complications to do with parameter passing and aliasing; for recursive procedures some form of induction is also needed. There is also the frame problem; specifying what is not changed by the procedure, which can include variables not visible in the scope of the procedure’s declaration. We set these separate concerns aside, in order to deal with an issue independent from procedures.

\textbf{Adaptation completeness.} Let us consider an example specification (from [5]). For integer auxiliary \(a\), integer variable \(x\), and real variable \(r\), the program (which cannot mention \(a\)) is to set \(x\) to an integer near \(r\):

\[
\text{pre0: } \quad a \leq r \leq a + 1 \quad \quad \text{post0: } \quad x = a \lor x = a + 1
\]

Suppose a program \(S\) meets the specification. From \{\textit{pre0}\} \; S \; \{\textit{post0}\} it is sound to infer the instantiation \{\(b - 1 \leq r \leq b\)\} \; S \; \{x = b - 1 \lor x = b\} because the meaning of an auxiliary variable in a triple is that the triple holds for all its values. The example is a little unusual: in an initial state where \(r\) has an integral value there are two values of \(a\) satisfying \textit{pre0}. The specification seems to require that if the initial value of \(r\) is an integer the final value of \(x\) is that number. In particular, it would seem that any program \(S\) satisfying this specification also satisfies \{\(r = 0\)\} \; S \; \{x = 0\}. Neither consequence nor instantiation license that conclusion; however, using two instantiations \{-1 \leq r \leq 0\} \; S \; \{x = -1 \lor x = 0\} and \{0 \leq r \leq 1\} \; S \; \{x = 0 \lor x = 1\}, a rule of conjunction [2] does yield \{\(r = 0\)\} \; S \; \{x = 0\}. But such proofs are unattractive for program construction.

Hoare’s rule of adaptation [11] is in harmony with the constructive approach based on weakest preconditions pioneered by Dijkstra. The rule gives a pre-condition valid with respect to a desired postcondition and the given specification. To reconcile the apparent need for extra rules with the completeness result mentioned above, let us define an ordering \(\preceq\) on specifications:
$$(\text{pre}', \text{post}') \preceq (\text{pre}, \text{post}) \text{ iff}$$

\begin{align}
(0) \text{ for all } S, \text{ if } \{ \text{pre}\} S \{ \text{post} \} \text{ is valid then } \{ \text{pre}' \} S \{ \text{post}' \} \text{ is valid.}
\end{align}

Now say a collection of rules is adaptation complete provided: If $(\text{pre}', \text{post}') \preceq (\text{pre}, \text{post})$ then any proof of a triple $\{ \text{pre}\} S \{ \text{post} \}$ can be extended to a proof of $\{ \text{pre}' \} S \{ \text{post}' \}$. Validity of $\text{pre}' \Rightarrow \text{pre}$ and $\text{post} \Rightarrow \text{post}'$ is sufficient but not necessary for $(\text{pre}', \text{post}') \preceq (\text{pre}, \text{post})$, which is why the rule of consequence can suffice for relative completeness yet not adaptation completeness. With the constructive approach in mind, let us say a condition is sharp [5] if, for any $\text{pre}, \text{post}, \text{post}'$, it gives the weakest $\text{pre}'$ such that $(\text{pre}', \text{post}') \preceq (\text{pre}, \text{post})$. For the example above, we expect that $r = 0$ is the weakest $\text{pre}'$ with $(\text{pre}', x = 0) \preceq (\text{pre}0, \text{post}0)$.

Our next step is to review notions of refinement calculus that we will use in Section 2 to derive sharp rules.

1 Refinement algebra: predicate transformers and relations

Soundness and sharpness proofs in the literature run to several pages of predicate calculations, and they are often given for a specific language only. We avoid commitment to a particular language by taking a semantic approach. We identify programs with their weakest precondition predicate transformers (transformers henceforth), i.e., monotonic functions on predicates.

**Relations.** We streamline our predicate calculations by using point-free relational calculus. To that end, we separate the auxiliary variables from program variables. Pre- and postconditions are taken to be binary relations, written $p : A \rightarrow X$, from auxiliaries to program variables, and thus specifications take the form $X \overset{\text{pre}}{\leftarrow} A \overset{\text{post}}{\rightarrow} Y$. Here $X$ is the initial state space, $Y$ the final state space, and $A$ the auxiliary state space. Programs suited to this specification are transformers $S : \mathcal{P}Y \rightarrow \mathcal{P}X$ acting on predicates over the program state. Although a similar development can be carried out for partial correctness, we consider here total correctness: For predicate $\varphi$ on the final state space $Y$, $S\varphi$ is interpreted as the weakest predicate on initial states such that all computations from $S\varphi$ terminate in $\varphi$. (Function application is denoted by left-associative invisible dot.) For ordinary commands, $Y$ is $X$, but allowing the state spaces to differ facilitates application of our results to procedures with value and result parameters; in proof rules the distinction between parameter types manifests itself in quantifications and in restrictions on specifications (e.g., preconditions being independent from result parameters [4]). For $\text{pre}0$ and $\text{post}0$, the auxiliary state is a single integer, and the program state, initial
and final, is a pair \((r, x)\) with \(r\) real and \(x\) an integer.

For relations \(p : X \rightarrow Y\) and \(q : Y \rightarrow Z\) we write \((p \; ; q)\) for ordinary relational composition, and we also write \((S \; ; S')\) for the functional composition of transformers \(S : \mathcal{P}X \rightarrow \mathcal{P}Y\) and \(S' : \mathcal{P}Y \rightarrow \mathcal{P}Z\). Note that \((S \; ; S')\) corresponds to executing \(S'\) first. For \(r : X \rightarrow Z\) the quotient \(r / q : X \rightarrow Y\) is characterized by \(p \; ; q \subseteq r \implies p \subseteq r / q\). At the level of states we have \(x(r / q)y \equiv (\forall z : y \; q \; z \Rightarrow x \; r \; z)\).

**Transformers.** The pointwise order \(\subseteq\) on transformers is defined by \(S \subseteq S'\) iff \((\forall \varphi : S \varphi \subseteq S' \varphi)\). It can be used to link specifications with programs because for \(\text{pre} : A \rightarrow X\) and \(\text{post} : A \rightarrow Y\) there is a transformer \(\text{pre} \rightarrow \text{post}\), of type \(\mathcal{P}Y \rightarrow \mathcal{P}X\), satisfying the fundamental theorem of refinement calculus [3,15]: for all \(S\),

\[
\text{pre} \rightarrow \text{post} \subseteq S \quad \text{iff} \quad \{\text{pre}\} \; S \{\text{post}\}
\]

Before defining \(\text{pre} \rightarrow \text{post}\) we review some connections between relations and transformers [8,7,16]. A transformer is universally disjunctive if it distributes over arbitrary unions; such transformers are called maps for short. A comap is a universally conjunctive transformer. For relation \(p : A \rightarrow X\), the direct image function \(\langle p \rangle : \mathcal{P}A \rightarrow \mathcal{P}X\) and inverse image function \(\lbrack p \rbrack : \mathcal{P}X \rightarrow \mathcal{P}A\) are defined by

\[
x \in \langle p \rangle \; \psi \equiv (\exists a : a \; p \; x \land a \in \psi) \quad \text{for all } x \in X, \psi \in \mathcal{P}A
\]

\[
a \in \lbrack p \rbrack \; \varphi \equiv (\forall x : a \; p \; x \Rightarrow x \in \varphi) \quad \text{for all } a \in A, \varphi \in \mathcal{P}X
\]

Now \(\langle p \rangle\) is a map and any map is \(\langle p \rangle\) for some \(p\). Moreover, \(\lbrack p \rbrack\) is a comap and all comaps take this form. The two image functions form a Galois connection:

\[
\lbrack p \rbrack \; ; S \subseteq S' \equiv S \subseteq \langle p \rangle \; ; S' \quad \text{and} \quad S \; ; \langle p \rangle \subseteq S' \equiv S \subseteq S' \; ; \lbrack p \rbrack
\]

for all transformers \(S, S'\). Direct images also satisfy

\[
p \subseteq q \equiv \langle p \rangle \subseteq \langle q \rangle \quad \text{and} \quad \langle p ; q \rangle = \langle p \rangle ; \langle q \rangle
\]

At the level of states we have

\[
x \in \lbrack p \rbrack \; \varphi \equiv \langle p \rangle \{x\} \subseteq \varphi \quad \text{for all } x, p, \varphi
\]

To define \(\text{pre} \rightarrow \text{post}\), recall that auxiliaries are universally quantified in triples.
Thus validity of \( \{ \text{pre} \} S \{ \text{post} \} \) is equivalent to the left side of

\[
(5) \quad (\forall a :: \langle \text{pre} \rangle \{a\} \subseteq S(\langle \text{post} \rangle \{a\})) \equiv \langle \text{pre} \rangle \subseteq \langle \text{post} \rangle ; S
\]

in which \( S \) is applied to predicates on the program state. We leave it to the reader to show equivalence \( (5) \). Applying connection \( (2) \) to the right side of \( (5) \) yields \( [\text{post}] ; \langle \text{pre} \rangle \subseteq S \). Defining \( \text{pre} \sim \text{post} = [\text{post}] ; \langle \text{pre} \rangle \), we get the fundamental theorem \( (1) \). In the sequel we drop the \( \sim \) notation.

In any poset, \( \alpha \leq \beta \) equilaves \( (\forall \gamma :: \beta \leq \gamma \Rightarrow \alpha \leq \gamma) \). This is called indirect inequality. Let us suppose \( S \) in \( (0) \) ranges over all transformers. Then, using indirect equality and the fundamental theorem \( (1) \), definition \( (0) \) leads to

\[
(6) \quad (\text{pre}', \text{post}') \leq (\text{pre}, \text{post}) \equiv [\text{post}'] ; \langle \text{pre}' \rangle \subseteq [\text{post}] ; \langle \text{pre} \rangle
\]

In Section 2 we use this characterization of \( \leq \) to derive a sharp rule. But first we prepare a few more ingredients.

**Floor and ceiling.** For any transformer \( S : PY \rightarrow PX \) the relation \( \text{rd} S : Y \rightharpoonup X \) is defined by \( y(\text{rd} S)x \equiv x \in S\{y\} \). If \( S \) is a map it is determined by its action on singletons, and so \( S = \langle \text{rd} S \rangle \). For any \( S \), the disjunctive floor is defined by \( [S] = \langle \text{rd} S \rangle \); it is a map and satisfies the floor property:

\[
S' \subseteq [S] \equiv S' \subseteq S \quad \text{for all maps } S'. \quad \text{We have } [S] ; [S'] \subseteq [S ; S'] \quad \text{but the fact we use later is}
\]

\[
(7) \quad [S ; \langle p \rangle] = [S] ; \langle p \rangle
\]

There is a relation \( \text{rc} S : X \rightharpoonup Y \) such that if \( S \) is a comap then \( S = [\text{rc} S] \). The definition is \( x(\text{rc} S)y \equiv (\forall \varphi :: x \in S \varphi \Rightarrow y \in \varphi) \). There is a conjunctive ceiling \( [S] \) of any \( S \), defined by \( [S] = [\text{rc} S] \). It is a comap and satisfies the ceiling property: \( S \subseteq S' \equiv [S] \subseteq S' \quad \text{for all comaps } S' \). Properties of floor and ceiling can be proved straightforwardly (or see \([14,7,16]\)).
2 Sharp Adaptation Rules Derived

We aim to find the weakest $pre' : B \rightarrow X$ for given $post' : B \rightarrow Y$, relative to a given $(pre, post)$ with types as before. We calculate:

\[
\begin{align*}
(pre', post') \preceq (pre, post) \\
\quad \equiv [post'] \sqsubseteq [pre'] \sqsubseteq [post]; \langle pre \rangle \\
\quad \equiv \langle pre' \rangle \sqsubseteq \langle post' \rangle ; [post] ; \langle pre \rangle \\
\quad \equiv \langle pre' \rangle \sqsubseteq [\langle post' \rangle ; [post] ; \langle pre \rangle] \\
\quad \equiv \langle pre' \rangle \sqsubseteq [\langle post' \rangle ; [post]] ; \langle pre \rangle
\end{align*}
\]

description (6)  
connection (2)  
floor, $\langle pre' \rangle$ is a map  
floor property (7)

Now we seek a simple expression $E$ with $E = [\langle post' \rangle ; [post]]$. Aiming to exploit the floor property, we explore an indirect equality in the poset of maps, namely $\langle p \rangle \sqsubseteq E \equiv \langle p \rangle \sqsubseteq [\langle r \rangle ; [q]]$ for arbitrary $p, r, q$ (recall that all maps take the form $\langle p \rangle$). Observe that

\[
\begin{align*}
\langle p \rangle \sqsubseteq [\langle r \rangle ; [q]] & \quad \text{floor, $\langle p \rangle$ is a map} \\
\langle p \rangle \sqsubseteq \langle r \rangle ; [q] & \quad \text{connection (2)} \\
\langle p \rangle ; \langle q \rangle \sqsubseteq \langle r \rangle & \quad \langle \rightarrow \rangle \, \text{properties (3)} \\
p ; q \sqsubseteq r & \quad \text{quotient} \\
p \sqsubseteq r / q & \quad \langle \rightarrow \rangle \, \text{property (3)} \\
\langle p \rangle \sqsubseteq \langle r / q \rangle
\end{align*}
\]

so we have proved the lemma (also found in [7])

(8) \quad \langle r / q \rangle = [\langle r \rangle ; [q]] \quad \text{for all relations $r, q$}

We continue the main calculation by observing

\[
\begin{align*}
\langle pre' \rangle \sqsubseteq [\langle post' \rangle ; [post]] ; \langle pre \rangle & \quad \text{lemma (8)} \\
\langle pre' \rangle \sqsubseteq \langle post' / post \rangle ; \langle pre \rangle & \quad \langle \rightarrow \rangle \, \text{properties (3)} \\
pre' \sqsubseteq (post' / post) ; pre
\end{align*}
\]

Thus the weakest $pre'$ is $(post' / post) ; pre$ and we have proved

**Theorem 1** For all pre, post, pre', and post'

\[
(pre', post') \preceq (pre, post) \equiv pre' \subseteq (post' / post) ; pre
\]
By the definitions of (\ref{def:pre}) and $/$, we have for the weakest $\text{pre}'$ that $b \text{pre}' \ x$
equivales

$$
\text{(T1)} \quad (\exists a :: a \ \text{pre} \ x \land (\forall y :: a \ \text{post} \Rightarrow b \ \text{post}' \ y))
$$

Let us apply the result to the example $\text{pre}0, \text{post}0$. Taking $\text{post}'$ to be $x = 0$, the theorem gives for $\text{pre}'$ an expression that simplifies to $\text{false}$, contrary to the expectation that it should be the weaker predicate $r = 0$.

**Towards a sharper rule.** What is given by Theorem 1 is weakest in terms of the order $\leq$ characterized in (6) under the assumption that $S$ in (0) ranges over all transformers. But, in contrast with exotica like $\langle \text{pre} \rangle$ and $[\text{post}]$, transformers corresponding to ordinary imperative programs satisfy Dijkstra’s *healthiness conditions*: positive conjunctivity (distribution over nonempty intersections), strictness ($S0 = 0$), and continuity. Some of the literature on adaptation rules restricts attention to deterministic programs, which correspond to transformers that are universally disjunctive as well as satisfying the previous conditions. We could also restrict the state spaces to be countable, or go even further and take computability into account. To guide our choice, we look to the example in Section 0.

We claimed that for $\text{pre}0, \text{post}0$ and postcondition $x = 0$, the precondition $r = 0$ could be justified using the standard rule of conjunction [2]. That rule is only sound for (finitely conjunctive transformers; let us focus on the class $PC$ of positively conjunctive transformers. Define a relation $\leq$ on specifications by

$$
(9) \quad \text{for all } S \in PC, \text{ if } \{ \text{pre} \} S \{ \text{post} \} \text{ then } \{ \text{pre}' \} S \{ \text{post}' \}
$$

(eliding the phrase “is valid”). The fundamental theorem (1) holds for $S$ in $PC$, but for $\leq$ the analog of (6) fails so we need another tool.

**Positively conjunctive ceiling.** Write $\top$ for true, i.e. the set of all states (of a type left implicit), so the range $\text{rng} \ p$ of a relation $p$ is $\langle p \rangle \top$. Let $S$ be a transformer $S : \mathcal{P}Y \to \mathcal{P}X$. Define the *termination transformer* $\star S : \mathcal{P}Y \to \mathcal{P}X$ to be the constant function with $\star S \varphi = ST$ for all $\varphi \in \mathcal{P}Y$. It is convenient to choose other domains for $\star S$, which can be done because it is constant-valued. Thus $\star S$ is a right-zero for (\ref{def:pre}) in that for any $S'$ we have $S' ; \star S = \star S$. It is straightforward to show

$$
(10) \quad \star ([\text{post}]; \langle \text{pre} \rangle) \varphi = \text{rng} \ \text{pre} \quad \text{for all } pre, post, \varphi
$$

Pointwise intersection of transformers gives greatest lower bounds in the poset of transformers and also in $PC$. We write $\cap$ for the g.l.b., and define the *positively conjunctive ceiling* $\Gamma S \varphi$ by $\Gamma S \varphi = [S] \cap \star S$. Thus $\Gamma S \varphi = [S] \varphi \cap ST$
for all \( \varphi \). Moreover, \( \succeq \) is positively conjunctive and has the ceiling property: 
\( S \subseteq S' \equiv \succeq \succeq S' \) for all positively conjunctive \( S' \). The proof is similar to the proofs of the ceiling property of \([-\) (see [13]).

**Sharp adaptation for positive conjunctivity.** We calculate:

\[
\begin{align*}
    (\text{pre}', \text{post}') & \preceq (\text{pre}, \text{post}) \\
    \equiv & \quad (\forall S \in PC : [\text{post}] ; \langle \text{pre} \rangle \subseteq S \Rightarrow [\text{post}'] ; \langle \text{pre} \rangle \subseteq S) \quad \text{(9) and (1)} \quad \text{ceiling}, S \in PC \\
    \equiv & \quad (\forall S \in PC : \succeq [\text{post}] ; \langle \text{pre} \rangle \subseteq S \Rightarrow \succeq [\text{post}'] ; \langle \text{pre} \rangle \subseteq S) \quad \text{indirect ineq.}
\end{align*}
\]

\[
\begin{align*}
    [\text{post}'] ; \langle \text{pre} \rangle \subseteq \succeq [\text{post}] ; \langle \text{pre} \rangle \quad & \text{ceiling definition \( \succeq \)} \\
    \equiv & \quad [\text{post}'] ; \langle \text{pre} \rangle \subseteq \succeq [\text{post}] ; \langle \text{pre} \rangle \\
    \equiv & \quad [\text{post}'] ; \langle \text{pre} \rangle \subseteq \succeq [\text{post}] \cap *([\text{post}] ; \langle \text{pre} \rangle) \quad \text{g.l.b.}
\end{align*}
\]

We deal with the conjuncts separately, starting with the second.

\[
\begin{align*}
    [\text{post}] ; \langle \text{pre} \rangle & \subseteq *([\text{post}] ; \langle \text{pre} \rangle) \quad \text{connection (2)} \\
    \equiv & \quad \langle \text{pre} \rangle \subseteq \langle \text{post}' \rangle \cap *([\text{post}] ; \langle \text{pre} \rangle) \quad \text{\( \ast \) S right zero} \\
    \equiv & \quad \langle \text{pre} \rangle \subseteq *([\text{post}] ; \langle \text{pre} \rangle) \quad \text{definition \( \subseteq \)} \\
    \equiv & \quad (\forall \varphi :: \langle \text{pre} \rangle \varphi \subseteq *([\text{post}] ; \langle \text{pre} \rangle) \varphi) \quad \text{property (10) of \( \ast \)} \\
    \equiv & \quad (\forall \varphi :: \langle \text{pre} \rangle \varphi \subseteq mg \text{ pre}) \quad \text{\( \langle \text{pre} \rangle \) monotonic} \\
    \equiv & \quad \langle \text{pre} \rangle \top \subseteq mg \text{ pre} \quad \text{definition mg}
\end{align*}
\]

For the first conjunct, we have

\[
\begin{align*}
    [\text{post}'] ; \langle \text{pre} \rangle & \subseteq [\text{post}] ; \langle \text{pre} \rangle \\
    \equiv & \quad [\text{post}'] ; \langle \text{pre} \rangle \subseteq [\rc ([\text{post}] ; \langle \text{pre} \rangle)] \\
    \equiv & \quad \langle \text{pre} \rangle \subseteq \langle \text{post}' \rangle ; [\rc ([\text{post}] ; \langle \text{pre} \rangle)] \quad \text{defintion \([-\) floor} \\
    \equiv & \quad \langle \text{pre} \rangle \subseteq \langle \text{post}' \rangle ; [\rc ([\text{post}] ; \langle \text{pre} \rangle)] \quad (8)q, r := post' , \rc ([\text{post}] ; \langle \text{pre} \rangle) \\
    \equiv & \quad \langle \text{pre} \rangle \subseteq \langle \text{post}' \rangle ; [\rc ([\text{post}] ; \langle \text{pre} \rangle)] \quad \langle - \rangle \text{ property}
\end{align*}
\]
To continue calculating at the level of relations we would need more of the algebra of $\text{rc}$, $[-]$, and $(-)$ than we have space to introduce, so let us unfold the definitions at the state level. The weakest $\text{pre}^\prime$ in the last line of the calculation above is equivalent, for all $b, x$, to $b(\text{post}'/\text{rc} ([\text{post}]; \langle \text{pre} \rangle))x$. Using the state-level expression for $/$ this equates $\forall y :: x(\text{rc} ([\text{post}]; \langle \text{pre} \rangle)y \Rightarrow b \text{ post}' y)$. We expand the antecedent:

\[
\begin{align*}
\forall y :: x(\text{rc} ([\text{post}]; \langle \text{pre} \rangle))y \\
\equiv (\forall \psi :: x \in ([\text{post}]; \langle \text{pre} \rangle) \psi \Rightarrow y \in \psi) \quad \text{definition \text{rc}} \\
\equiv (\forall \psi :: (\exists a :: a \in [\text{post}] \psi \land a \text{ pre } x) \Rightarrow y \in \psi) \quad \text{definitions \langle ; \rangle, \langle - \rangle} \\
\equiv (\forall \psi :: (\exists a :: \langle \text{post} \rangle \{a\} \subseteq \psi \land a \text{ pre } x) \Rightarrow y \in \psi) \quad \text{(4)} \\
\equiv (\forall a, \psi :: \langle \text{post} \rangle \{a\} \subseteq \psi \land a \text{ pre } x \Rightarrow y \in \psi) \quad \text{logic} \\
\equiv (\forall a :: a \text{ pre } x \Rightarrow y \in \langle \text{post} \rangle \{a\}) \quad \text{set theory, def. \langle - \rangle} \\
\equiv (\forall a :: a \text{ pre } x \Rightarrow a \text{ post } y) \quad \text{definition \langle - \rangle}
\end{align*}
\]

Finally, at the level of states, the other conjunct $\text{rng \ pre}' \subseteq \text{rng \ pre}$ equates $(\forall b, x :: b \text{ pre}' x \Rightarrow (\exists a :: a \text{ pre } x))$. Combining these results we have

**Theorem 2** For all pre, post, and post', the weakest $\text{pre}'$ with $(\text{pre}', \text{post}') \leq (\text{pre}, \text{post})$ is the following predicate on $b, x$:

\[(T2) \quad (\exists a :: a \text{ pre } x) \land (\forall y :: (\forall a :: a \text{ pre } x \Rightarrow a \text{ post } y) \Rightarrow b \text{ post}' y)\]

3 Discussion

Here is the adaptation of Bijlsma et al. [5] for procedure call, using their notation: $(\exists m :: U_{a,b}^{\text{call}}) \land (\forall y, z :: (\forall m :: U_{a,b}^{\text{eq}} \Rightarrow V_{a}^{\text{eq}}) \Rightarrow E_{b,c}^{\text{eq}})$. About six pages of predicate calculation are used in [5] to show that this is sound and sharp. It can be obtained directly from (T2) by expressing procedure call in terms of assignments to local variables, from which the substitutions arise. Rules that can be obtained from (T1) in the same way appear in several places, e.g. [9]. Bijlsma et al [5] show their adaptation to be sharp for strict, positively conjunctive transformers, under the assumption that specification $(\text{pre}, \text{post})$ is satisfiable. For a satisfiable specification, $(\text{post} ; \langle \text{pre} \rangle)$ is strict; our proof of Theorem 2 also yields their result because $\tau \equiv$ preserves strictness.

Kleymann [12] considers adaptation independently from procedures, but the proofs still involve complicated predicate calculation. Sharpness for deterministic programs is proved using an operational semantics. Kleymann’s proposal is obtained from (T1) by re-nesting to allow the auxiliary to depend on the
final state:³

(K) \[(\forall y :: (\exists a :: a \text{ pre } x \land (a \text{ post } y \Rightarrow b \text{ post}' y)))\]

It may not be immediately obvious, but (K) equivales (T2).

Bijlsma [4] changes the inner quantifier in (T2) to obtain

(B) \[(\exists a :: a \text{ pre } x) \land (\forall y :: (\exists a :: a \text{ pre } x \land a \text{ post } y) \Rightarrow b \text{ post}' y)\]

This strictly implies (T2), because \((\exists a :: a \text{ pre } x \land a \text{ post } y)\) is weaker than \((\forall a :: a \text{ pre } x \Rightarrow a \text{ post } y)\), given \((\exists a :: a \text{ pre } x)\). It is also straightforward to show that (B) implies (T1), and this implication is also strict (for example, take \(a \text{ pre } x \equiv a = 0 \lor a = 1, a \text{ post } y \equiv a = 0 \land y = 1, \text{ and } b \text{ post}' y \equiv y \neq 1\)).

Bijlsma [4] argues that (B) is more useful than (T2) in program construction, and points out that (B) equivales (T2) (hence (T1), (T2), (K), and (B) are all equivalent) if for any \(x\) there is at most one \(a\) with \(a \text{ pre } x\). More succinctly: if the converse \(\text{ pre}^o\) of \(\text{ pre}\) is a partial function. The most common use for auxiliary variables is to “snapshot” initial variables —indeed, special notations like zero-subscripted variables [15] are often recommended for this purpose—in which case \(\text{ pre}^o\) is a partial function. This may explain why (T1) has seen wide use, and why the recommendation is a good one.⁴ It also lessens the temptation of exploring sharpness for other classes of programs.

Morgan’s [15] specification statements and auxiliary variables (called \textbf{con}stants) have semantics \((\textbf{con} a \bullet [\text{pre}, \text{post}])\text{post}' = (\exists a :: \text{pre} \land (\forall y :: \text{post} \Rightarrow \text{post}'))\) explained by an informal operational reading (and justified by the fundamental theorem; note that \((\textbf{con} a \bullet [\text{pre}, \text{post}] ) = [\text{post}] ; (\langle \text{pre} \rangle)\). Except for our pedantic distinction of auxiliaries from program variables in \(\text{post}'\), the right side is exactly (T1). Theorem 2 suggests another form of specification statement, suitable for use in development of feasible programs, and satisfying the analog of (6) for \(\leq\). There is little enticement to study the refinement laws for this new statement, because it is the same as Morgan’s for most specifications.

³ Apparently (K) was devised independently from [5], as [5,4] are not cited in [12].
⁴ Bijlsma [4] shows that every specification is “equivalent” to one in which \(\text{ pre}^o\) is a partial function. What is not emphasized in [4] is the meaning of equivalence: inspections of the proofs shows that transformers are assumed to be strict and positively conjunctive. In fact every positively conjunctive transformer can be expressed as \([p] ; (r)\) for some \(p,r\) with \(r^o\) a partial function, and every strict one can be expressed this way with \(p\) total on its domain.
Acknowledgement

Thanks to Edsger W. Dijkstra for encouragement and inspiration in the pursuit of transparent and disentangled arguments which free the imagination to relish the dance of formulas. Thanks also for specific help: In response to my failure to find find a nice calculational proof of (K)≐(T2), Edsger pointed out the “range diffusion” laws [6] needed in such a proof, which the reader may enjoy pursuing. By way of thanks, I offer another exercise: show that (T1)≡(T2) in case pre is a partial function. For this, one may care to calculate with relations, writing (T2) as (true ; pre) ∩ (post/(post/ pre))

References


http://www.cs.utexas.edu/users/EWD/ewd12xx/EWD1201.PDF.


