Hoare Logic and Auxiliary Variables*

Thomas Kleymann†

LFCS, Division of Informatics
University of Edinburgh

October 28, 1998

Abstract: Auxiliary variables are essential for specifying programs in Hoare Logic. They are required to relate the value of variables in different states. However, the axioms and rules of Hoare Logic turn a blind eye to the rôle of auxiliary variables. We stipulate a new structural rule for adjusting auxiliary variables when strengthening preconditions and weakening postconditions. Courtesy of this new rule, Hoare Logic is adaptation complete, which benefits software re-use. This property is responsible for a number of improvements. Relative completeness follows uniformly from the Most General Formula property. Moreover, contrary to common belief, one can show that Hoare Logic subsumes VDM's operation decomposition rules in that every derivation in VDM can be naturally embedded in Hoare Logic. Furthermore, the new treatment leads to a significant simplification in the presentation for verification calculi dealing with more interesting features such as recursion or concurrency.

Keywords: Hoare Logic; Auxiliary variables; Adaptation Completeness; Most General Formula; VDM; Recursive procedures; Concurrency; Owicki-Gries

1 Introduction

Hoare Logic is a verification calculus which relates imperative programs with two assertions, both (first-order) logical formulae. These assertions are interpreted as *predicates on states* where free variables denote the value of program variables in a specific state. Variables for which no counterpart appears as a program variable in the program under consideration then take on the rôle of auxiliary variables. They are required to relate the value of program variables in *different states*.

In practice, auxiliary variables are essential ingredients for specifying properties about imperative programs [Vic91]. Nevertheless, the axioms and rules in Hoare Logic do not support the rôle of auxiliary variables. This is a known deficiency and has been overcome in other frameworks e.g., specification logic [Rey82] and the Vienna Development Method (VDM) [Jon90].

In our opinion, the rôle of auxiliary variables in Hoare Logic has been underestimated. We stipulate a new structural rule for adjusting auxiliary variables when strengthening preconditions

*This is an extended version of [Sch97].

†formerly Schreiber

and weakening postconditions. This alone leads to adaptation completeness. One may adapt arbitrary satisfiable specifications. As a consequence,

- we clarify how to uniformly establish completeness as a corollary of Gorelick's Most General Formula (MGF) theorem [Gor75] which focusses on deriving a specific correctness formula. One may adapt the MGF specification to an arbitrary specification in a single step.

- We can show that, contrary to common belief, Hoare Logic subsumes VDM's operation decomposition rules in that every derivation in VDM can be naturally embedded in Hoare Logic.

- The Hoare Logic presentation for recursive procedures can be simplified significantly. Specifically, we are able to show that Sokołowski's calculus [Sok77] is sound and complete if one replaces Hoare's rule of consequence with ours. Apt's remedy of adding three further structural rules had led to a complete but unsound system [Apt81, AdB90].

- Similar simplifications seem possible for verification calculi dealing with concurrency. Considering two standard motivating examples, we demonstrate that Owicki and Gries' elimination rule for auxiliary variables [OG76a] is redundant in our approach. With our method, auxiliary variables have to be dealt with at the level of assertions only.

The overview of this paper is as follows: Hoare Logic and the concepts of soundness, relative completeness and adaptation completeness are briefly introduced in Sect. 2.

In Sect. 3, we discuss the rôle of auxiliary variables in Hoare Logic. As our main contribution, we stipulate an improved rule of consequence which allows us to modify auxiliary variables while strengthening preconditions and weakening postconditions. The section concludes with a proof of adaptation completeness.

In sections 4–7, we illustrate the benefits of our new approach. In Sect. 4, we present a language independent completeness proof as a corollary of the MGF property. In Sect. 5, we establish that every operation decomposition rule can be simulated in Hoare Logic. In Sect. 6, we review the development of verification calculi for imperative programs with recursive (parameterless) procedures in the setting of total correctness. Our treatment of auxiliary variables leads to a significant simplification. We turn our attention to calculi for concurrent programs in Sect. 7. By way of examples, we show that, with our more principled treatment of auxiliary variables, Owicki and Gries' elimination rule for auxiliary variables does not seem to be required.

In Sect. 8, we comment on conducting machine-checked soundness and completeness proofs with the help of an interactive proof assistant.

2 Hoare Logic

For the purpose of this section, we consider a (very) simple imperative programming language merely considering of assignments, sequential composition, conditionals and loops.

Definition 2.1 (Syntax of Programs) *Imperative programs $S : \texttt{prog}$ are defined by the BNF grammar $S ::= x := e \mid S_1;\ S_2 \mid \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \mid \textbf{while}\ b\ \textbf{do}\ S$ where x is a program variable, e an expression and b a boolean expression.*

We assume that sequential composition is left-associative. Furthermore, we employ **begin** and **end** in our concrete syntax to clarify precedence.

We describe the meaning of imperative programs with the help of a structural operational semantics. In Sec. 2.2, assertions are introduced. We then present Hoare Logic. This includes a semantic account and a set of axioms and rules for deriving correctness formulae. Soundness and completeness relate these two views of Hoare Logic. Finally, in Sect. 2.6 we turn our attention to adaptation completeness.

2.1 Semantics of Imperative Programs

We employ structural operational semantics to axiomatise the meaning of an imperative programming language. In contrast to denotational semantics, operational semantics avoids more intricate concepts such as least fixpoints. Operational semantics relates a program with its initial and final state. A state $\sigma : \Sigma$ maps program variables to values. Evaluation of terms $\mathrm{eval}(\sigma)(e)$ is defined in the standard way.

Definition 2.2 (Structural Operational Semantics) *The operational semantics is defined as the least relation* $. \xrightarrow{\ .\ } . \subseteq \Sigma \times \texttt{prog} \times \Sigma$ *satisfying*

$$\sigma \xrightarrow{x := e} \sigma[x \mapsto \mathrm{eval}(\sigma)(e)]$$

$$\frac{\sigma \xrightarrow{S_1} \eta \quad \eta \xrightarrow{S_2} \tau}{\sigma \xrightarrow{S_1;\ S_2} \tau}$$

$$\frac{\sigma \xrightarrow{S_1} \tau}{\sigma \xrightarrow{\textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2} \tau} \qquad \textit{provided}\ \mathrm{eval}(\sigma)(b) = \textbf{true}\ .$$

$$\frac{\sigma \xrightarrow{S_2} \tau}{\sigma \xrightarrow{\textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2} \tau} \qquad \textit{provided}\ \mathrm{eval}(\sigma)(b) = \textbf{false}\ .$$

$$\sigma \xrightarrow{\textbf{while}\ b\ \textbf{do}\ S} \sigma \qquad \textit{provided}\ \mathrm{eval}(\sigma)(b) = \textbf{false}\ .$$

$$\frac{\sigma \xrightarrow{S} \eta \quad \eta \xrightarrow{\textbf{while}\ b\ \textbf{do}\ S} \tau}{\sigma \xrightarrow{\textbf{while}\ b\ \textbf{do}\ S} \tau} \qquad \textit{provided}\ \mathrm{eval}(\sigma)(b) = \textbf{true}\ .$$

2.2 Assertions

Traditionally, assertions are considered to be simply formulae of first-order logic $p : L$, which are interpreted in the usual way, except that the value of variables is determined by a state. Semantically, assertions denote sets on states $[\![p]\!] \subseteq \Sigma$. We write $[\![p]\!](\sigma)$ to denote that p is valid when all variables are interpreted according to the state σ. For convenience, we only consider standard interpretations.

Example 2.1 (Syntax and Semantics of Assertions) *The formula $y \geq 0$ denotes the set of states in which the value of the program variable y is positive.*

2.3 Semantics and Derivability of Hoare Logic

Hoare Logic is a verification calculus for deriving correctness formulae of the form $\{p\}\ S\ \{q\}$ for assertions p, q and programs S. We consider total correctness. Intuitively $\{p\}\ S\ \{q\}$ specifies that, provided S is executed in a state such that the precondition p holds, it terminates in a state where the postcondition q holds. We distinguish between the semantics of a correctness formulae $\models_{\text{Hoare}} \{p\}\ S\ \{q\}$ (which formalises the above intuition) and the notion of deriving a correctness formulae $\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}$ (which is employed in order to verify concrete programs). Again, we omit the issue of non-standard interpretations.

Definition 2.3 (Semantics of Hoare Logic for Total Correctness)

$$\begin{aligned}\models_{\text{Hoare}} \{p\}\ S\ \{q\} &\subseteq L \times \texttt{prog} \times L \\ &\stackrel{\text{def}}{=} \forall \sigma \cdot [\![p]\!](\sigma) \Rightarrow \exists \tau \cdot \sigma \xrightarrow{\quad S \quad} \tau \wedge [\![q]\!](\tau)\ .\end{aligned}$$

It is more challenging to capture total correctness for non-deterministic behaviour. It is common practice to then only consider partial correctness. Termination is assumed to have been established by other means.

Definition 2.4 (Semantics of Hoare Logic for Partial Correctness)

$$\begin{aligned}\models_{\text{Hoare}} \{p\}\ S\ \{q\} &\subseteq L \times \texttt{prog} \times L \\ &\stackrel{\text{def}}{=} \forall \sigma, \tau \cdot \left([\![p]\!](\sigma) \wedge \sigma \xrightarrow{\quad S \quad} \tau\right) \Rightarrow [\![q]\!](\tau)\ .\end{aligned}$$

Based on work by Floyd [Flo67], Hoare [Hoa69] proposed a syntax-directed proof system for deriving correctness formula. For every construct of the imperative programming language, Hoare Logic provides a rule which allows one to decompose a program. Programs mentioned in the premisses are strict subprograms of the programs mentioned in the conclusions. Unlike the operational semantics, this also holds for loops. One also needs a structural rule to strengthen the precondition and weaken the postcondition in a proof obligation. This is particularly useful when one wants to apply the rule for loops as the precondition must remain invariant with respect to the body of the loop.

Definition 2.5 (Derivability of Hoare Logic Correctness Formulae) *A verification calculus for Hoare Logic is defined as the least relation* $\vdash_{\text{Hoare}} \{.\}\ .\ \{.\} \subseteq L \times \texttt{prog} \times L$ *satisfying*

$$\vdash_{\text{Hoare}} \{p[x \mapsto e]\}\ x := e\ \{p\} \tag{HAssign}$$

$$\frac{\vdash_{\text{Hoare}} \{p\}\ S_1\ \{r\} \quad \vdash_{\text{Hoare}} \{r\}\ S_2\ \{q\}}{\vdash_{\text{Hoare}} \{p\}\ S_1;\ S_2\ \{q\}} \tag{HSeq}$$

$$\frac{\vdash_{\text{Hoare}} \{p \wedge b\}\ S_1\ \{q\} \qquad \vdash_{\text{Hoare}} \{p \wedge \neg b\}\ S_2\ \{q\}}{\vdash_{\text{Hoare}} \{p\}\ \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \{q\}} \tag{HCond}$$

$$\frac{\forall t : W \cdot \vdash_{\text{Hoare}} \{p \wedge b \wedge u = t\}\ S\ \{p \wedge u < t\}}{\vdash_{\text{Hoare}} \{p\}\ \mathbf{while}\ b\ \mathbf{do}\ S\ \{p \wedge \neg b\}} \quad \textit{where } (W, <) \textit{ is well-founded.} \tag{HLoop}$$

$$\frac{\vdash_{\text{Hoare}} \{p_1\}\ S\ \{q_1\}}{\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}} \quad \textit{provided } p \Rightarrow p_1 \textit{ and } q_1 \Rightarrow q. \tag{HCons}$$

2.4 Soundness

If a system is unsound, deriving a property for a particular program within the formal system does not guarantee that the program actually fulfils the property. Formally, one needs to show that whenever a correctness formulae $\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}$ is derivable, the proposition $\models_{\text{Hoare}} \{p\}\ S\ \{q\}$ holds.

Theorem 2.1 (Soundness) *The verification calculus introduced in Def. 2.5 is sound.*

Proof by induction on the derivation of $\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}$ [Kle98a]. □

2.5 Completeness

In an incomplete formal system, one may only verify a strict subset of all true formulae. A naive definition of completeness is bound to fail in the context of verification calculi. On the one hand, if the chosen underlying logical language is too weak, e.g., pure first-order logic together with the boolean constants **false** and **true**, some intermediate assertions cannot be expressed. Hence, derivations cannot be completed. On the other hand, if the logical language is too strong, e.g. Peano Arithmetic, it itself is already incomplete and the verification calculus inherits incompleteness.

To avoid this problem, Cook has proposed that one investigates *relative completeness* in an attempt to separate the reasoning about programs from the reasoning about the underlying logical language [Coo78]. Two concessions are made:

1. One only considers expressive (first-order) logics in the sense of Sect. 2.5.1 below.

2. Furthermore, the formal system is augmented by a theory[1] of first-order logic. Whenever a rule is protected by a first-order logic side-condition, the rule is already applicable when the side-condition is an element of the theory, and not only when the side-condition is itself *derivable*.

2.5.1 Expressiveness

One must be able to express the weakest precondition[2].

Definition 2.6 (Weakest Precondition) *Let S be an arbitrary program and $\Delta \subseteq \Sigma$ be a set of states.*

$$\mathrm{wp}(S,\Delta) \stackrel{\text{def}}{=} \left\{\sigma \mid \exists\tau \cdot \left(\sigma \xrightarrow{\quad S \quad} \tau\right) \wedge (\tau \in \Delta)\right\}$$

Expressiveness amounts to closure under weakest precondition.

Definition 2.7 (Expressiveness) *The logic L is expressive relative to the programming language* `prog` *if and only if for arbitrary programs S :* `prog` *and postconditions $q : L$ the logic contains a formula $p : L$ which characterises the weakest precondition i.e.,*

$$[\![p]\!] = \mathrm{wp}\left(S, [\![q]\!]\right)$$

2.5.2 A Model Theoretic View of Side Conditions

Furthermore, rules of the verification calculus may be applied in a derivation if the logical side-condition is valid rather than derivable. In particular, completeness no longer compares a proof-theoretic with a model-theoretic account. Thus, instead of the consequence rule (HCons), one needs to consider

$$\frac{\vdash_{\text{Hoare}} \{p_1\}\ S\ \{q_1\}}{\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}} \qquad \text{provided } \left(\forall\sigma \cdot [\![p \Rightarrow p_1]\!](\sigma)\right) \wedge \left(\forall\tau \cdot [\![q_1 \Rightarrow q]\!](\tau)\right).$$

With the modified proof system, completeness amounts to showing that, whenever the proposition $\models_{\text{Hoare}} \{p\}\ S\ \{q\}$ holds, the correctness formulae $\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}$ is derivable.

Theorem 2.2 (Completeness) *The verification calculus introduced in Def. 2.5 is (relative) complete.*

Proof by induction on the structure of S [Kle98a]. We discuss a refined proof technique in Sect. 4. □

[1] the set of all valid as opposed to derivable formulae

[2] For partial correctness, one needs to instead consider the weakest liberal precondition [Cou90].

2.6 Adaptation Completeness

For programming in the large, adaptation completeness is a desirable feature [Zwi89]. Whenever, irrespective of the details of the program S, two correctness formula $\{p_1\}\ S\ \{q_1\}$ and $\{p\}\ S\ \{q\}$ are equivalent, one would like to *derive* $\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}$ from $\vdash_{\text{Hoare}} \{p_1\}\ S\ \{q_1\}$ and vice versa. Taking software reuse seriously, imagine that $\vdash_{\text{Hoare}} \{p_1\}\ S\ \{q_1\}$ has been derived as part of the verification in a project. It is conceivable that in another project, one needs the equivalent correctness formula $\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}$. Unless the verification calculus is adaptation complete, one may be forced to rebuild $\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}$ from scratch. Of course in the new derivation one would benefit from analysing the first derivation, but such a methodology is nevertheless undesirable. Ideally, a verification calculus ought to support arbitrary adaptations of satisfiable specification.

Definition 2.8 (Satisfiable Specification) *A specification $\langle p, q\rangle : L \times L$ is satisfiable if there exists a program S such that $\models_{\text{Hoare}} \{p\}\ S\ \{q\}$.*

In the setting of *partial* correctness, every specification is trivially satisfied by an always diverging program.

Definition 2.9 (Adaptation Completeness) *Given assertions p_1, q_1, p, q such that $\langle p_1, q_1\rangle$ is satisfiable and*

$$\forall S : \texttt{prog} \cdot \models_{\text{Hoare}} \{p_1\}\ S\ \{q_1\} \Rightarrow\ \models_{\text{Hoare}} \{p\}\ S\ \{q\}\ \ ,$$

a formulation of Hoare Logic is adaptation complete if, and only if, for an arbitrary program S, one may close the derivation

$$\begin{array}{c} \vdash_{\text{Hoare}} \{p_1\}\ S\ \{q_1\} \\ \vdots \\ \vdash_{\text{Hoare}} \{p\}\ S\ \{q\} \end{array} \tag{1}$$

Pragmatically, one requires adaptation completeness under the additional constraint that some (auxiliary) variables can only occur in assertions but not programs. This is the subject of the following section.

3 Auxiliary Variables

Consider the specification

$$\{\mathbf{true}\}\ S\ \{y = x!\} \tag{2}$$

where x, y are program variables and S is a yet to be implemented imperative programs. This is not an adequate specification for the factorial function. The postcondition relates merely the value of program variables in the *final* state. One cannot state that the input x ought to remain invariant. Hoare Logic does not support input/output specifications. Hence, the program $S \stackrel{\text{def}}{=} x := 0;\ y := 1$ also satisfies specification (2). Auxiliary variables come (almost) to the rescue. These are simply program variables which only occur in assertions. In the precondition they freeze the value of other program variables so that in the postcondition one may refer to the initial value of program variables.

Let X and Y be further program variables. In Hoare Logic, we may paraphrase the above correctness formula (2) as

$$\{X = x\}\ S\ \{y = X!\} \quad .$$

It is a convention that auxiliary variables do not occur in programs. Unfortunately, this cannot be stated as part of the correctness formula. Thus, problems arise when the program to be verified is not fully given. This is for example the case when one wants to

- develop programs hand-in-hand with their proofs of correctness [Gri81, Mor90, BM93],
- consider free procedures which are specified with respect to pre- and postcondition, but for which one does not consider a concrete implementation [Tar85], or
- verify programs which invoke recursive procedures.

This is a well known deficiency and has been overcome in several other systems. In specification logic [Rey82], as part of the correctness formula, one may explicitly state that a program does not interfere with auxiliary variables. For example, an adequate specification for the factorial function could be stated by

$$\vdash_{\mathrm{SL}} \forall X \cdot \{X = x\}\ S\ \{y = X!\} \wedge S\#X \quad .$$

Another solution is the VDM formalism [Jon90, JS90]. In the postcondition one may employ hooked program variables $\overleftarrow{x}$ to refer to the value of program variables in the initial state e.g.,

$$\vdash_{\mathrm{VDM}} \{\mathbf{true}\}\ S\ \{y = \overleftarrow{x}\,!\} \quad .$$

3.1 A New Rule of Consequence

Our main contribution is to stipulate a new rule of consequence for Hoare Logic which allows one to modify auxiliary variables when strengthening preconditions and weakening postconditions. Assume that S is a program about which we only know that the program variable X does not occur in it. Then, the two specifications

$$\{X = x\}\ S\ \{X = x\} \tag{3}$$

and

$$\{X = x+1\}\ S\ \{X = x+1\} \tag{4}$$

where all variables denote integer values both assert that the program S leaves x invariant. Whenever one can derive (3), one can derive (4) and vice versa. Nevertheless, the consequence rule (HCons) does not allow one to derive (4) from (3) or vice versa, as one would have to show $x = x+1$.

The traditional presentation of Hoare Logic is not adaptation complete in the presence of auxiliary variables. This causes problems in verifying recursive procedures, see Sect. 6. In a nutshell, similar to an inductive step, during the verification of a recursive procedure, one needs to unroll the recursive call and can exploit the correctness formula before unrolling. Unfortunately, the hypothesis only matches modulo shifts of auxiliary variables reminiscent of the difference between (3) and (4).

Consider the following rule of consequence

$$\frac{\vdash_{\text{Hoare}} \{p_1\}\ S\ \{q_1\}}{\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}} \qquad \text{if } \forall Z \cdot \forall x \cdot p \Rightarrow \exists Z_1 \cdot \big(p_1\,[Z \mapsto Z_1] \wedge (\forall x \cdot q_1\,[Z \mapsto Z_1] \Rightarrow q)\big). \tag{HCons*}$$

where x is a list of all program variables and Z is a list of all auxiliary variables. We clarify the precise meaning of the side-condition later in this section. As an example, in inferring (4) from (3), the side condition amounts to

$$\forall X \cdot \forall x \cdot X = x+1 \Rightarrow \exists Z_1 \cdot \big(Z_1 = x \wedge (\forall x \cdot Z_1 = x \Rightarrow X = x+1)\big) \ . \tag{5}$$

Thus, employing the new rule of consequence, the two correctness formulae (3) and (4) are inter-derivable because the side condition (5) is satisfied for $Z_1 = X-1$.

The consequence rule (HCons*) is strictly stronger than (HCons). The side condition is more liberal in two respects:

- It suffices to show that the postcondition has been weakened under the additional assumption that the precondition of the conclusion holds. This idea has been borrowed from VDM's rule of consequence [Acz82b, Jon90].
- One may adjust the auxiliary variables in the premiss. Their value may depend on the value of auxiliary variables in the conclusion and the value of all program variables in the initial state.

In the sequel, we annotate derivations with HCons to denote that an inference is sanctioned by Hoare's original version (and thus, also, the stronger version HCons*). In contrast, HCons* signals that the inference is only possible with the stronger version.

In the above side-condition, one only substitutes *auxiliary* variables. This highlights that auxiliary variables and program variables deserve to be treated differently. At the syntactic level, one ought to (formally) distinguish between program variables and auxiliary variables. One could for example enforce that program variables have to start with a lower-case letter, whereas auxiliary variables must start with an upper-case letter. To be well-formed, programs may only refer to program variables [HM96].

Technically, it is more elegant to distinguish between program variables and auxiliary variables at the semantic level, too. Following a proposal by Apt and Meertens [AM80], we interpret assertions relative to an (arbitrary) domain of auxiliary variables and the state space. It is straightforward to revise the above definitions. In particular, semantics of Hoare Logic amounts to

Definition 3.1 (Revised Semantics of Hoare Logic for Total Correctness)

$$\begin{aligned}&\models_{\text{Hoare}} \{p\}\, S\, \{q\} \subseteq L \times \texttt{prog} \times L \\ &\qquad \stackrel{\text{def}}{=} \forall Z \cdot \forall \sigma \cdot [\![p]\!](Z,\sigma) \Rightarrow \exists \tau \cdot \sigma \xrightarrow{\quad S \quad} \tau \wedge [\![q]\!](Z,\tau)\ .\end{aligned}$$

For the purpose of investigating completeness, the new rule of consequence (HCons*) states

$$\frac{\vdash_{\text{Hoare}} \{p_1\}\, S\, \{q_1\}}{\vdash_{\text{Hoare}} \{p\}\, S\, \{q\}}$$

$$\text{if } \forall Z \cdot \forall \sigma \cdot [\![p]\!](Z,\sigma) \Rightarrow \exists Z_1 \cdot \Big([\![p_1]\!](Z_1,\sigma) \wedge \big(\forall \tau \cdot [\![q_1]\!](Z_1,\tau) \Rightarrow [\![q]\!](Z,\tau)\big)\Big). \quad (6)$$

However, this rule does not lead to adaptation completeness.

Example 3.1 ([Mor]) *It is not possible to derive*

$$\frac{\vdash_{\text{Hoare}} \{Z_1 = 0 \vee Z_1 = 1\}\, S\, \{x \neq Z_1\}}{\vdash_{\text{Hoare}} \{\textbf{true}\}\, S\, \{x \neq 0 \wedge x \neq 1\}}$$

because the side condition of (6) *is equivalent to* **false**.

In choosing the adapted auxiliary variable Z_1, one needs to additionally interrogate the value of program variables in the final state. This can be achieved by the following version.

$$\frac{\vdash_{\text{Hoare}} \{p_1\}\, S\, \{q_1\}}{\vdash_{\text{Hoare}} \{p\}\, S\, \{q\}}$$

$$\text{if } \forall Z \cdot \forall \sigma \cdot [\![p]\!](Z,\sigma) \Rightarrow \forall \tau \cdot \exists Z_1 \cdot \Big([\![p_1]\!](Z_1,\sigma) \wedge \big([\![q_1]\!](Z_1,\tau) \Rightarrow q(Z,\tau)\big)\Big). \quad (7)$$

Theorem 3.1 (Adaptation Completeness for Total Correctness) *The consequence rule* (7) *ensures adaptation completeness.*

Proof Given $\models_{\text{Hoare}} \{p_1\}\, S^*\, \{q_1\}$ for some program S^* and

$$\forall S : \texttt{prog} \cdot \models_{\text{Hoare}} \{p_1\}\, S\, \{q_1\} \Rightarrow \models_{\text{Hoare}} \{p\}\, S\, \{q\}\ , \tag{8}$$

we show that the rule of consequence (7) allows one to derive

$$\frac{\vdash_{\text{Hoare}} \{p_1\}\, S\, \{q_1\}}{\vdash_{\text{Hoare}} \{p\}\, S\, \{q\}} \tag{9}$$

for an arbitrary program S.

Assume that the side condition is not fulfilled i.e.,

$$\exists Z \cdot \exists \sigma \cdot [\![p]\!](Z,\sigma) \wedge \Big(\exists \tau \cdot \forall Z_1 \cdot \neg [\![p_1]\!](Z_1,\sigma) \vee \big([\![q_1]\!](Z_1,\tau) \wedge \neg [\![q]\!](Z,\tau)\big)\Big)\ . \tag{10}$$

Consider the particular interpretation for auxiliary variables Z, initial state σ and final state τ given by (10). Our strategy is to reach a contradiction by instantiating (8) with a particular program S' such that for every input σ' (including σ),

$$\forall Z_1 \cdot [\![p_1]\!](Z_1, \sigma') \Rightarrow \exists \tau \cdot \sigma' \xrightarrow{\quad S' \quad} \tau \wedge [\![q_1]\!](Z_1, \tau) \ , \tag{11}$$

yet there is no final state τ which satisfies

$$\sigma \xrightarrow{\quad S' \quad} \tau \wedge [\![q]\!](Z, \tau) \ . \tag{12}$$

We distinguish between two cases:

$\forall Z_1 \cdot \neg [\![p_1]\!](Z_1, \sigma)$: We construct the program S' as follows. On input σ, diverge (thus, (12) cannot hold). Otherwise, behave like S^*. For $\sigma' = \sigma$, (11) is fulfilled because p_1 is universally false. For other inputs, (11) amounts to a special case of the hypothesis (9).

$\exists Z_1 \cdot [\![p_1]\!](Z_1, \sigma)$: This time, for input σ, the new program S' should terminate with output τ. It otherwise simulates S^*.

To see that (11) holds for $\sigma' = \sigma$, we transform the second conjunctive clause of (10) into

$$\forall Z_1 \cdot [\![p_1]\!](Z_1, \sigma) \Rightarrow [\![q_1]\!](Z_1, \tau) \wedge \neg [\![q]\!](Z, \tau) \ . \tag{13}$$

For other inputs, (11) amounts again to a special case of the hypothesis (9).

As there is a specific witness for Z_1 such that $[\![p_1]\!](Z_1, \sigma)$, we may appeal to (13) to conclude $\neg [\![q]\!](Z, \tau)$. Hence, (12) cannot hold.

□

In the sequel, we are content with the weaker version of the consequence rule (6). The problematic issue raised in Example 3.1 does not surface in any of the soundness and completeness proofs, nor in any of the other examples considered in this paper and the author's thesis [Kle98a].

Theorem 3.2 (Adaptation Completeness for Partial Correctness) *For partial correctness, the rule of consequence*

$$\frac{\vdash_{\text{Hoare}} \{p_1\}\ S\ \{q_1\}}{\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}}$$

$$\textit{if}\ \forall Z \cdot \forall \sigma \cdot [\![p]\!](Z, \sigma) \Rightarrow \forall \tau \cdot (\forall Z_1 \cdot [\![p_1]\!](Z_1, \sigma) \Rightarrow [\![q_1]\!](Z, \tau)) \Rightarrow [\![q]\!](Z, \tau). \tag{14}$$

ensures adaptation completeness.

Proof Similar to the case for total correctness [Mor, Hof97]. □

4 Most General Formula

An immediate benefit of the new treatment is that completeness can be established directly from the MGF theorem by adapting assertions with the help of the consequence rule. This is easier than the direct proof because it avoids having to cater for arbitrary assertions. In other words, instead of directly deriving

$$\models_{\text{Hoare}} \{p\}\ S\ \{q\} \Rightarrow \vdash_{\text{Hoare}} \{p\}\ S\ \{q\}$$

(by induction on S), one considers the stronger property $\vdash_{\text{Hoare}} \text{MGF}_{\text{Hoare}}(S)$ for which induction (also on S) goes through more easily. In particular, the direct proof cannot be applied when one considers recursive procedures, because the induction hypotheses are not strong enough.

The proposition $\vdash_{\text{Hoare}} \text{MGF}_{\text{Hoare}}(S)$ asserts that, provided that one only considers input states in which the program S terminates, one may *derive* a correctness formula in which the postcondition relates all inputs with the appropriate outputs according to the underlying operational semantics of the programming language. At the semantic level, $\models_{\text{Hoare}} \text{MGF}_{\text{Hoare}}(S)$ holds trivially.

Definition 4.1 (MGF for Total Correctness)

$$\text{MGF}_{\text{Hoare}}(S) \stackrel{\text{def}}{=} \{p\}\ S\ \{q\} \qquad \textit{where } [\![p]\!](Z,\sigma) \stackrel{\text{def}}{=} \sigma \xrightarrow{\quad S \quad} Z \textit{ and } [\![q]\!](Z,\tau) \stackrel{\text{def}}{=} Z = \tau.$$

At the syntactic level, the postcondition of the MGF can be described by the formula

$$\bigwedge_{x \in \mathbf{VAR}} X = x \ .$$

It is considerably more difficult to devise a syntactic version of the precondition. If the assertion language is Peano Arithmetic, this construction involves encoding computations with the help of Gödel numbers [dB80]. However, in any case, the logic L must be able to express this assertion. This is guaranteed by the definition of expressiveness since $[\![p]\!] = \text{wp}\big(S, [\![q]\!]\big)$.

Theorem 4.1 (MGF) *For an arbitrary program S, one may derive* $\vdash_{\text{Hoare}} \text{MGF}_{\text{Hoare}}(S)$.

Proof by induction on the structure of S [Kle98a]. □

Corollary 4.1 (Completeness of Hoare Logic) *For any precondition p, program S and postcondition q, whenever* $\models_{\text{Hoare}} \{p\}\ S\ \{q\}$ *holds, then* $\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}$ *is derivable.*

It is instructive to study the completeness proof in order to appreciate the rôle of the main theorem 4.1. In particular, the same proof goes through regardless of the programming language features involved.

Proof Let S be a program and p, q be assertions. Given $\models_{\text{Hoare}} \{p\}\ S\ \{q\}$ i.e.,

$$\forall Z \cdot \forall \sigma \cdot [\![p]\!](Z,\sigma) \Rightarrow \exists \tau \cdot \sigma \xrightarrow{\quad S \quad} \tau \wedge [\![q]\!](Z,\tau) \tag{15}$$

we need to establish $\vdash_{\mathrm{Hoare}} \{p\}\ S\ \{q\}$. By the consequence rule (6), we may directly infer $\vdash_{\mathrm{Hoare}}$ $\{p\}\ S\ \{q\}$ from the MGF, triggering the side condition

$$\forall Z \cdot \forall \sigma \cdot [\![p]\!](Z,\sigma) \Rightarrow \exists Z_1 \cdot \big((\sigma \xrightarrow{\quad S \quad} Z_1) \wedge (\forall \tau \cdot \tau = Z_1 \Rightarrow [\![q]\!](Z,\tau))\big) \ . \tag{16}$$

Clearly, (15) implies (16). □

5 Hoare Logic Subsumes VDM

In the Vienna Development Method (VDM), assertions are objects of a logic of partial functions. In the postcondition one may employ *hooked* program variables $\overleftarrow{x}$ to refer to the value of the program variable in the initial state. Thus, the meaning of a postcondition is determined by both the initial and the final state:

Definition 5.1 (Semantics of VDM Correctness Formulae for Total Correctness)

$$\begin{aligned} &\models_{\mathrm{VDM}} \{p\}\ S\ \{q\} \subseteq L \times \mathtt{prog} \times (L \cup \overleftarrow{L}\,) \\ &\qquad \stackrel{\mathrm{def}}{=} \forall \sigma \cdot [\![p]\!](\sigma) \Rightarrow \exists \tau \cdot \sigma \xrightarrow{\quad S \quad} \tau \wedge [\![q]\!](\sigma,\tau) \end{aligned}$$

VDM encompasses two orthogonal methods:

1. refining abstract objects into concrete data types available in the programming language, and

2. refining correctness formulae, which allows one to decompose programs.

We will only consider VDM's operation decomposition rules [Acz82a, Acz82b, Jon90], see Fig. 1. Soundness and completeness has been machine-checked [Kle98a].

In the traditional understanding of Hoare Logic, assertions characterise *predicates* on states. VDM is more flexible in that postconditions may, in addition to the final state, also refer to the initial state. In that respect, one may argue that VDM subsumes Hoare Logic.

However, we question such a view, because, pragmatically, in Hoare Logic, one uses auxiliary variables to relate output to input. Thus, one may choose an *arbitrary* reference point. Specifically, any VDM correctness formula $\{p\}\ S\ \{q\}$ can be embedded in Hoare Logic $\{\ulcorner p \urcorner\}\ S\ \{\ulcorner q \urcorner\}$. As the domain for auxiliary variables, we choose the state space $T = \Sigma$. In the precondition $\ulcorner p \urcorner$, in addition to asserting p, one freezes all program variables to ensure that the initial state is the point of reference. In postconditions, hooked variables can simply be replaced by auxiliary variables:

Definition 5.2 (Embedding VDM assertions in Hoare Logic)

$$\begin{aligned} [\![\ulcorner p \urcorner]\!](Z,\sigma) &\stackrel{\mathrm{def}}{=} Z = \sigma \wedge [\![p]\!](\sigma) \\ [\![\ulcorner q \urcorner]\!](Z,\tau) &\stackrel{\mathrm{def}}{=} [\![q]\!](Z,\tau) \end{aligned}$$

$$\vdash_{\text{VDM}} \{\mathbf{true}\}\ x := e \left\{ x = \overleftharpoon{e} \wedge \bigwedge_{v \in \mathbf{VAR} \setminus \{x\}} v = \overleftharpoon{v} \right\} \qquad \text{(VDMAssign)}$$

$$\frac{\vdash_{\text{VDM}} \{p_1\}\ S_1\ \{p_2 \wedge r_1\} \qquad \vdash_{\text{VDM}} \{p_2\}\ S_2\ \{r_2\}}{\vdash_{\text{VDM}} \{p_1\}\ S_1;\ S_2\ \{r_1 \circ r_2\}} \qquad \text{(VDMSeq)}$$

$$\frac{\vdash_{\text{VDM}} \{p \wedge b\}\ S_1\ \{q\} \quad \vdash_{\text{VDM}} \{p \wedge \neg b\}\ S_2\ \{q\}}{\vdash_{\text{VDM}} \{p\}\ \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \{q\}} \qquad \text{(VDMCond)}$$

$$\frac{\vdash_{\text{VDM}} \{p \wedge b\}\ S\ \{p \wedge \text{sofar}\}}{\vdash_{\text{VDM}} \{p\}\ \mathbf{while}\ b\ \mathbf{do}\ S\ \{p \wedge \neg b \wedge (\text{sofar} \vee \bigwedge_{v \in \mathbf{VAR}} v = \overleftharpoon{v}\,)\}}$$

where sofar is transitive and well-founded. (VDMLoop)

$$\frac{\vdash_{\text{VDM}} \{p_1\}\ S\ \{q_1\}}{\vdash_{\text{VDM}} \{p\}\ S\ \{q\}} \qquad \text{provided } p \Rightarrow p_1 \text{ and } \overleftharpoon{p} \Rightarrow q_1 \Rightarrow q \qquad \text{(VDMCons)}$$

Figure 1: VDM's Operation Decomposition Rules

$$\dfrac{\dfrac{\dfrac{\vdash_{\text{VDM}} \{p_1\}\ r := r * x\ \{q_1\}}{\vdash_{\text{VDM}} \{p_2\}\ r := r * x\ \{q_3\}}\ \text{VDMCons} \qquad \vdash_{\text{VDM}} \{p_1\}\ y := y - 1\ \{q_4\}}{\vdash_{\text{VDM}} \{p_2\}\ r := r * x;\ y := y - 1\ \{q_5\}}\ \text{VDMSeq}}{\vdash_{\text{VDM}} \{p_2\}\ r := r * x;\ y := y - 1\ \{q_6\}}\ \text{VDMCons}$$

$$\begin{array}{ll}
p_1 \equiv \mathbf{true} & \ulcorner p_1 \urcorner \equiv R = r \wedge X = x \wedge Y = y \wedge \mathbf{true} \\
p_2 \equiv 0 < y & \ulcorner p_2 \urcorner \equiv R = r \wedge X = x \wedge Y = y \wedge 0 < y \\
q_1 \equiv r = \overleftharpoon{r} * \overleftharpoon{x} \wedge x = \overleftharpoon{x} \wedge y = \overleftharpoon{y} & \ulcorner q_1 \urcorner \equiv R = r * X \wedge X = x \wedge y = Y \\
q_2 \equiv 0 < y \wedge q_1 & \ulcorner q_2 \urcorner \equiv 0 < y \wedge \ulcorner q_1 \urcorner \\
q_3 \equiv p_1 \wedge q_2 & \ulcorner q_3 \urcorner \equiv \ulcorner p_1 \urcorner \wedge \ulcorner q_2 \urcorner \\
q_4 \equiv r = \overleftharpoon{r} \wedge x = \overleftharpoon{x} \wedge y = \overleftharpoon{y} - 1 & \ulcorner q_4 \urcorner \equiv R = r \wedge X = x \wedge Y = y - 1 \\
q_5 \equiv q_2 \circ q_4 & \ulcorner q_5 \urcorner \equiv \ulcorner q_2 \urcorner \circ \ulcorner q_4 \urcorner \\
q_6 \equiv 0 \le y \wedge \overleftharpoon{r} * \overleftharpoon{x}^{\overleftharpoon{y}} = r * x^y \wedge y < \overleftharpoon{y} & \ulcorner q_6 \urcorner \equiv 0 \le y \wedge R * X^Y = r * x^y \wedge y < Y
\end{array}$$

Figure 2: Derivation in VDM

For examples of syntactic embeddings, see Fig. 2.

By appealing to soundness and completeness, it is evident that for every derivation in VDM, one can reconstruct a proof of the corresponding correctness formula in Hoare Logic:

Theorem 5.1 (Embedding VDM Correctness Formulae in Hoare Logic) *Let p and q be assertions and S an arbitrary program. Whenever $\vdash_{\text{VDM}} \{p\}\ S\ \{q\}$ is derivable, we may also establish $\vdash_{\text{Hoare}} \{\ulcorner p \urcorner\}\ S\ \{\ulcorner q \urcorner\}$.*

Proof Assume $\vdash_{\text{VDM}} \{p\}\ S\ \{q\}$. By soundness of VDM, we have

$$\forall \sigma \cdot [\![p]\!](\sigma) \Rightarrow \exists \tau \cdot \sigma \xrightarrow{\quad S \quad} \tau \wedge [\![q]\!](\sigma, \tau)$$

which is equivalent to

$$\forall Z \cdot \forall \sigma \cdot \big(Z = \sigma \wedge [\![p]\!](\sigma)\big) \Rightarrow \exists \tau \cdot \sigma \xrightarrow{\quad S \quad} \tau \wedge [\![q]\!](Z, \tau) \ .$$

Hence, by completeness of Hoare Logic, we may derive $\vdash_{\text{Hoare}} \{\ulcorner p \urcorner\}\ S\ \{\ulcorner q \urcorner\}$. □

In the standard approach to Hoare Logic, this embedding is *not compositional* because Hoare Logic is not adaptation complete. In general, to simulate a VDM derivation for a correctness formula $\vdash_{\text{VDM}} \{p\}\ S\ \{q\}$ in Hoare Logic $\vdash_{\text{Hoare}} \{\ulcorner p \urcorner\}\ S\ \{\ulcorner q \urcorner\}$, one cannot simply embed all nodes in the tree and reconstruct the derivation with the rules of Hoare Logic. Instead, one needs to analyse the complete original derivation to design appropriate leaves in the Hoare Logic proof tree.

Example 5.1 *Employing VDM's rule of consequence, one may infer*

$$\frac{\vdash_{\text{VDM}} \{p_1\}\ r := r * x\ \{q_1\}}{\vdash_{\text{VDM}} \{p_2\}\ r := r * x\ \{q_2\}}$$

whereas in Hoare Logic, Hoare's rule of consequence does not *sanction the derivation*

$$\frac{\vdash_{\text{Hoare}} \{\ulcorner p_1 \urcorner\}\ r := r * x\ \{\ulcorner q_1 \urcorner\}}{\vdash_{\text{Hoare}} \{\ulcorner p_2 \urcorner\}\ r := r * x\ \{\ulcorner q_2 \urcorner\}} \ .$$

In order to derive

$$\vdash_{\text{Hoare}} \{\ulcorner p_2 \urcorner\}\ r := r * x\ \{\ulcorner q_2 \urcorner\} \ , \tag{17}$$

the leaf needs to already refer to y in the precondition. In a successful derivation, one can start with $\vdash_{\text{Hoare}} \{\ulcorner q_2 \urcorner [r \mapsto r * x]\}\ r := r * x\ \{\ulcorner q_2 \urcorner\}$. Employing the rule of consequence yields the desired (17). *In particular,*

$$\vdash_{\text{Hoare}} \{\ulcorner p_1 \urcorner\}\ r := r * x\ \{\ulcorner q_1 \urcorner\}$$

is not part of the derivation.

5.1 A Natural Embedding

However, due to adaptation completeness, a compositional translation does exist in the new treatment of auxiliary variables in Hoare Logic. More precisely, every correctness formula $\{p\}\ S\ \{q\}$ is reflected in the corresponding Hoare Logic translation as $\{\ulcorner p \urcorner\}\ S\ \{\ulcorner q \urcorner\}$. Furthermore, the new derivation is glued together with the help of the new consequence rule.

An axiom in VDM $\vdash_{\mathrm{VDM}} \{p\}\ S\ \{q\}$ is embedded as an instance of the corresponding axiom in Hoare Logic, followed by an application of the new consequence rule to produce

$$\vdash_{\mathrm{Hoare}} \{\ulcorner p \urcorner\}\ S\ \{\ulcorner q \urcorner\} \ .$$

If the VDM derivation employs one of the syntax-directed rules, we

1. embed all the premisses in Hoare Logic (induction step).

2. We modify (some of the) embedded premisses with the help of the new consequence rule.

3. We apply the corresponding syntax-directed rule in Hoare Logic

4. We (possibly) apply the new rule of consequence to the conclusion from step 3.

VDM's consequence rule can be simulated by our rule of consequence (but not Hoare's).

As an example, the derivation in Fig. 3 simulates the VDM derivation from Fig. 2. We refer to [Kle98a] for the technical details of the translation in the general case.

$$\dfrac{\dfrac{\dfrac{\vdash_{\mathrm{Hoare}} \{p_1^r\}\ r := r*x\ \{\ulcorner q_1 \urcorner\}}{\vdash_{\mathrm{Hoare}} \{\ulcorner p_1 \urcorner\}\ r := r*x\ \{\ulcorner q_1 \urcorner\}}\,\text{HCons}}{\vdash_{\mathrm{Hoare}} \{\ulcorner p_2 \urcorner\}\ r := r*x\ \{\ulcorner q_3 \urcorner\}}\,\text{HCons*} \qquad \dfrac{\dfrac{\vdash_{\mathrm{Hoare}} \{p_1^y\}\ y := y-1\ \{\ulcorner q_4 \urcorner\}}{\vdash_{\mathrm{Hoare}} \{\ulcorner p_1 \urcorner\}\ y := y-1\ \{\ulcorner q_4 \urcorner\}}\,\text{HCons}}{\vdash_{\mathrm{Hoare}} \{\ulcorner q_2 \urcorner\}\ y := y-1\ \{\ulcorner q_5 \urcorner\}}\,\text{HCons*}}{\dfrac{\vdash_{\mathrm{Hoare}} \{\ulcorner p_2 \urcorner\}\ r := r*x;\ y := y-1\ \{\ulcorner q_5 \urcorner\}}{\vdash_{\mathrm{Hoare}} \{\ulcorner p_2 \urcorner\}\ r := r*x;\ y := y-1\ \{\ulcorner q_6 \urcorner\}}\,\text{HCons}}\,\text{HSeq}$$

$$p_1^r \equiv r*x = R*X \wedge X = x \wedge Y = y$$
$$p_1^y \equiv R = r \wedge X = x \wedge y-1 = Y$$

Figure 3: Simulating a VDM Derivation in Hoare Logic

We should emphasise that our argument that Hoare Logic subsumes VDM reflects the *intended* use of both systems. Of course, one could mirror Hoare Logic derivations in VDM by also employing auxiliary variables. But notice that one would then have to also improve VDM's rule of consequence to simulate our version for Hoare Logic. VDM is not adaptation complete in the presence of auxiliary variables.

5.2 VDM versus Hoare Logic

There is a one-to-one correspondence between the rules of Hoare Logic and VDM's operation decomposition rules. According to Theorem 5.1, specifications in VDM correspond to a particular class of specification in Hoare Logic, in which the auxiliary variables are devoted to freezing the values of all program variables prior to execution. The additional flexibility available in Hoare Logic is to blame for the more elaborate consequence rule, whereas the hard-wired perspective between pre- and postconditions in VDM is responsible for a more complicated sequential rule. In the light of our new approach to Hoare Logic, it would be interesting to compare the two verification calculi Hoare Logic and VDM in case studies of practical interest. VDM's contribution in data reification is orthogonal and can also be applied on top of derivations in Hoare Logic.

6 Recursive Procedures

In this section, we show that our new approach to auxiliary variables leads to a significantly simpler sound and complete verification calculus for recursive procedures in the setting of total correctness. Specifically, it is an improvement over America and de Boer's [AdB90] system. Our new rule of consequence subsumes their four structural rules.

We restrict our attention to a single parameterless procedure. Let S_0 denote the body of the procedure. Invoking the procedure can be achieved unambiguously with a constructor **call**. Parameter passing is an orthogonal issue [Mor88, Kle98a]. An extension to multiple procedures is straightforward [Kle98a].

Example 6.1 (Factorial) *The body of a recursive procedure computing the factorial of x is given by*

$$\begin{aligned} S_0 \stackrel{\text{def}}{=} \textbf{if } x = 0 \textbf{ then } & y := 1 \\ \textbf{else } & \underline{\textbf{begin}}\ x := x - 1;\ \textbf{call};\ x := x + 1;\ y := y * x\ \underline{\textbf{end}} \end{aligned}$$

Notice that the argument x and the result y are not parameters, but global variables. Clearly, x plays the rôle of a call-by-value parameter. The procedure ensures that its value remains invariant. Similarly, the variable y is a designated call-by-name parameter.

We first consider procedures without recursion. We then move on to recursive procedures without worrying about termination. Finally, we present the development of Hoare Logic for recursive procedures in the setting of total correctness.

6.1 Partial Correctness

Without recursion, deriving a correctness formula is easy. Whenever one encounters a procedure invocation, one has to proceed by instead analysing the procedure body i.e.,

$$\frac{\vdash_{\text{Hoare}} \{p\}\ S_0\ \{q\}}{\vdash_{\text{Hoare}} \{p\}\ \textbf{call}\ \{q\}}\ .$$

This rule would however lead to infinite derivations when S_0 calls itself. Induction comes to the rescue. Let us first omit the issue of termination. We may simply assume $\vdash_{\text{Hoare}} \{p\}\ \textbf{call}\ \{q\}$ to conclude $\vdash_{\text{Hoare}} \{p\}\ S_0\ \{q\}$ [Hoa71] i.e.,

$$\frac{\{p\}\ \textbf{call}\ \{q\} \vdash_{\text{Hoare}} \{p\}\ S_0\ \{q\}}{\vdash_{\text{Hoare}} \{p\}\ \textbf{call}\ \{q\}} \quad .$$

This rule introduces a fundamental change in deriving correctness formulae. Derivations are now to be considered with respect to a context. Instead of a Hilbert-style calculus, Hoare Logic now amounts to a Gentzen-style sequent calculus[3]. All other rules need to be revised to preserve contexts, see Fig. 5.

Olderog [Old81] has observed that one does not need to cater for arbitrary contexts. It suffices to appeal to the rule for recursive procedures only *once* in a derivation. Thus, contexts can be restricted to at most one correctness formulae for each procedure body. We need only one structural rule to inspect contexts,

$$\Gamma \vdash_{\text{Hoare}} \{p\}\ S\ \{q\} \qquad \text{provided } \{p\}\ S\ \{q\} \in \Gamma.$$

In our scenario of a single procedure declaration, the context is either empty or contains a single correctness formula $\{p\}\ \textbf{call}\ \{q\}$.

6.2 Total Correctness

To guarantee termination, one needs to introduce a termination measure on states. For recursive procedures, one can guarantee progress if recursive procedure invocations are only permitted in strictly smaller states with respect to a termination measure. Sokołowski [Sok77] suggests

$$\frac{\forall n : \mathbb{N} \cdot \{p(n)\}\ \textbf{call}\ \{q\} \vdash_{\text{Hoare}} \{p(n+1)\}\ S_0\ \{q\}}{\vdash_{\text{Hoare}} \{\exists n : \mathbb{N} \cdot p(n)\}\ \textbf{call}\ \{q\}} \qquad \text{provided } \neg p(0). \tag{18}$$

He claims to establish soundness and completeness, but, as Apt [Apt81] points out, merely adding such a rule for procedure invocations in a setting with Hoare's consequence rule does not lead to a complete system. Unlike logic, where finding valid formulae which cannot be derived is often somewhat esoteric, a different story has to be told for the notion of (relative) completeness in verification calculi. Sokołowski's system is seriously incomplete in that it is difficult to come up with any non-contrived correctness formula of a recursive procedure which *can* be derived. One cannot establish the correctness of the factorial algorithm from Exa. 6.1, $\nvdash_{\text{Hoare}} \{X = x\}\ \textbf{call}\ \{y = X!\}$. One cannot even show that x remains invariant,

$$\nvdash_{\text{Hoare}} \{X = x\}\ \textbf{call}\ \{X = x\} \quad .$$

[3] In a Gentzen-style presentation, one may express the adaptation completeness condition (1) more concisely by $\{p_1\}\ S\ \{q_1\} \vdash_{\text{Hoare}} \{p\}\ S\ \{q\}$.

The rule of consequence is too weak to reconcile the required step in its derivation

$$\begin{array}{c} \vdash_{\text{Hoare}} \{n = x+1 \land X = x\}\ \mathbf{call}\ \{X = x\} \\ \vdots \\ \vdash_{\text{Hoare}} \{n = x+1 \land X = x+1\}\ \mathbf{call}\ \{X = x+1\} \end{array} \quad . \tag{19}$$

Apt added three more rules to be able to modify auxiliary variables in correctness formulae dealing with recursive procedures, see Fig. 4. These additional rules close the gap and one may derive (19). Unfortunately, despite Apt's "proof" to the contrary, one may also derive invalid correctness formulae [AdB90].

$$\frac{\Gamma \vdash_{\text{Hoare}} \{p\}\ \mathbf{call}\ \{q\}}{\Gamma \vdash_{\text{Hoare}} \left\{p\left[\overline{Z} \mapsto \overline{Y}\right]\right\}\ \mathbf{call}\ \left\{q\left[\overline{Z} \mapsto \overline{Y}\right]\right\}}$$
$$\text{where } \overline{Z} \cap \mathit{free}(S_0) = \emptyset \text{ and } \overline{Y} \cap \mathit{free}(S_0) = \emptyset. \quad \text{(Substitution)}$$

$$\frac{\{p\}\ \mathbf{call}\ \{q\}}{\{p \land r\}\ \mathbf{call}\ \{q \land r\}} \quad \text{where } \mathit{free}(r) \cap \mathit{free}(S_0) = \emptyset. \quad \text{(Invariance)}$$

$$\frac{\Gamma \vdash_{\text{Hoare}} \{p\}\ \mathbf{call}\ \{q\}}{\Gamma \vdash_{\text{Hoare}} \left\{\exists \overline{Z} \cdot p\right\}\ \mathbf{call}\ \{q\}} \quad \text{where } \overline{Z} \cap \mathit{free}(S_0, q) = \emptyset. \quad \text{(Elimination)}$$

Figure 4: Apt's Additional Structural Rules

America and de Boer's remedy consists of distinguishing different kinds of (auxiliary) variables. In particular, the universally quantified counter variable of the procedure invocation rule must not occur in the list of variables $\overline{Y}$ and $\overline{Z}$ in the Substitution and Elimination rules of Fig. 4. Moreover, America and de Boer have shown that, with these modifications, one obtains a sound and complete verification calculus.

In our opinion, this is not an ideal solution. With such a set of rules, using Hoare Logic to derive properties of even simple programs appears to be unnecessarily complicated.

6.3 No Need for Further Structural Rules

It is at best a questionable strategy to patch an incomplete system by adding lots of new rules. One needs to address the source of the problem and not merely its symptoms. The derivation (19) cannot be completed because auxiliary variables are not adequately supported in Hoare Logic. We are able to show that Sokołowski's calculus is sound and complete if one replaces Hoare's rule of consequence with ours. In particular, none of the other troublesome structural rules introduced by Apt are required. The soundness and completeness proofs have been machine-checked [Sch97].

Without procedures, completeness is not threatened because a mismatch of auxiliary variables such as (19) can always be reconciled by rebuilding the complete derivation. One may compensate by readjusting the auxiliary variables accordingly in the leaves of the proof tree. Whenever an axiom e.g., (HAssign), is encountered one instead chooses an equivalent version where auxiliary variables have been shifted. Such a patch cannot be applied in the presence of rule (18). It also introduces an axiom i.e., $\{p(n)\}$ **call** $\{q\}$ for some $n : \mathbb{N}$, but, unlike the axiom for an assignment, the pre- and postcondition is bound by the *specific* assertions of a correctness formula in the derivation.

In particular, completeness could be achieved in the standard presentation by integrating our new consequence rule in the procedure invocation rule. We will discuss a similar rule in Sect. 6.5. However, we think that pragmatically, it is better to provide an improved consequence rule.

6.4 Proving Termination by Well-Founded Induction

In practice, Sokołowski's rule (18) it is too cumbersome to apply, because of the requirement that the termination measure decreases by exactly one. We have therefore extended (18) to an arbitrary well-founded measure

$$\frac{\forall t : W \cdot \{\exists u : W \cdot p(u) \wedge u < t\}\ \mathbf{call}\ \{q\} \vdash_{\mathrm{Hoare}} \{p(t)\}\ S_0\ \{q\}}{\vdash_{\mathrm{Hoare}} \{\exists t : W \cdot p(t)\}\ \mathbf{call}\ \{q\}}$$

where $(W, <)$ is well-founded.

The verification calculus remains sound and complete.

Theorem 6.1 (Soundness and Completeness) *The set of verification rules for recursive procedures summarised in Fig. 5 is sound and complete.*

Proof Soundness is established by induction on the derivation of $\Gamma \vdash_{\mathrm{Hoare}} \{p\}\ S\ \{q\}$. Completeness (for empty contexts) follows as a corollary of the MGF. Machine-checked proofs are documented in the author's thesis [Kle98a]. □

6.5 Rules of Adaptation

Among the additional rules to add in order to retain completeness for imperative programs dealing with recursive procedures, a rule of adaptation has been considered by various authors [Hoa71, Mor, LGH+78, GL80]. In general, rules of adaptation are of the form

$$\frac{\Gamma \vdash_{\mathrm{Hoare}} \{p_1\}\ S\ \{q_1\}}{\Gamma \vdash_{\mathrm{Hoare}} \{p\}\ S\ \{q\}}$$

for arbitrary assertions p_1, q_1, q, and particular proposals for the adapted precondition p. Ideally, the rule should be left-maximal [Dah92] i.e., provided the specification $\langle p_1, q_1\rangle$ is satisfiable, the

$$\{p\}\ \mathbf{call}\ \{q\} \vdash_{\text{Hoare}} \{p\}\ \mathbf{call}\ \{q\}$$

$$\Gamma \vdash_{\text{Hoare}} \{p[x \mapsto e]\}\ x := e\ \{p\}$$

$$\frac{\Gamma \vdash_{\text{Hoare}} \{p\}\ S_1\ \{r\} \quad \Gamma \vdash_{\text{Hoare}} \{r\}\ S_2\ \{q\}}{\Gamma \vdash_{\text{Hoare}} \{p\}\ S_1;\ S_2\ \{q\}}$$

$$\frac{\Gamma \vdash_{\text{Hoare}} \{p \wedge b\}\ S_1\ \{q\} \qquad \Gamma \vdash_{\text{Hoare}} \{p \wedge \neg b\}\ S_2\ \{q\}}{\Gamma \vdash_{\text{Hoare}} \{p\}\ \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \{q\}}$$

$$\frac{\forall t : W \cdot \Gamma \vdash_{\text{Hoare}} \{p \wedge b \wedge u = t\}\ S\ \{p \wedge u < t\}}{\Gamma \vdash_{\text{Hoare}} \{p\}\ \mathbf{while}\ b\ \mathbf{do}\ S\ \{p \wedge \neg b\}} \qquad \text{where } (W, <) \text{ is well-founded.}$$

$$\frac{\forall t : W \cdot \{\exists u : W \cdot p(u) \wedge u < t\}\ \mathbf{call}\ \{q\} \vdash_{\text{Hoare}} \{p(t)\}\ S_0\ \{q\}}{\vdash_{\text{Hoare}} \{\exists t : W \cdot p(t)\}\ \mathbf{call}\ \{q\}}$$

where $(W, <)$ is well-founded.

$$\frac{\Gamma \vdash_{\text{Hoare}} \{p_1\}\ S\ \{q_1\}}{\Gamma \vdash_{\text{Hoare}} \{p\}\ S\ \{q\}}$$

$$\text{if } \forall Z \cdot \forall \sigma \cdot [\![p]\!](Z, \sigma) \Rightarrow \forall \tau \cdot \exists Z_1 \cdot \Big([\![p_1]\!](Z_1, \sigma) \wedge \big([\![q_1]\!](Z_1, \tau) \Rightarrow q(Z, \tau)\big)\Big).$$

Figure 5: New Set of Hoare Logic Rules for Recursive Procedures

precondition p should be the weakest possible so that

$$\forall S \cdot \models_{\text{Hoare}} \{p_1\}\ S\ \{q_1\} \Rightarrow \models_{\text{Hoare}} \{p\}\ S\ \{q\}$$

holds. Since the rules of consequence (14) and (7) (for partial and total correctness, respectively), are solely responsible for adaptation completeness, it is straightforward to synthesise an optimal rule of adaptation. For partial correctness, one obtains[4]

$$[\![p]\!](Z,\sigma) \stackrel{\text{def}}{=} \forall \tau \cdot \big(\forall Z_1 \cdot [\![p_1]\!](Z_1,\sigma) \Rightarrow [\![q_1]\!](Z_1,\tau)\big) \Rightarrow [\![q]\!](Z,\tau) \ .$$

For total correctness, one gets

$$[\![p]\!](Z,\sigma) \stackrel{\text{def}}{=} \forall \tau \cdot \exists Z_1 \cdot [\![p_1]\!](Z_1,\sigma) \wedge ([\![q_1]\!](Z_1,\tau) \Rightarrow [\![q]\!](Z,\tau)) \ .$$

Previously, rules of adaptation have not been considered in the setting of total correctness.

7 Concurrency

Verifying concurrent programs is challenging. Programs may execute in parallel while accessing the same memory[5]. For tackling concurrency, it is well known that auxiliary variables are essential to design a complete verification calculus in the style of Hoare Logic [OG76a, Cli81, Sou84, Sti88]. Correctness formulae need to be extended to explicitly record information about the status of all other programs running in parallel within the same state space. This is accomplished with the help of auxiliary variables e.g., one could record computation histories in auxiliary variables [Sou84].

Is this orthogonal to our approach to auxiliary variables? Employing two standard examples [AO91], we motivate that our rule of consequence might subsume Owicki's structural rule to deal with auxiliary variables for concurrency. We first consider disjoint parallel programs. The second example involves parallel programs with shared data. Finally, we sketch how our new treatment of auxiliary variables may also simplify the presentation of compositional verification calculi for concurrent programs.

7.1 Disjoint Parallel Programs

To decompose programs running in parallel, Hoare [Hoa75] introduces the rule

$$\frac{\{p_i\}\ S_i\ \{q_i\} \quad \text{for } i \in \{1,\ldots,n\}}{\{\bigwedge_{i=1}^n p_i\}\ S_1 \parallel \ldots \parallel S_n\ \{\bigwedge_{i=1}^n q_i\}} \qquad \text{where } \mathit{free}(p_i,q_i) \cap \text{change}(S_j) = \emptyset \text{ for } i \neq j. \qquad \text{(HPar)}$$

[4]This version is originally due to Morris [Mor]. A more elaborate version has been published by Olderog [Old83].

[5]We make the usual assumptions that boolean expressions and assignments are evaluated or executed as atomic actions.

In a scenario, where the only other structural rule is Hoare's original rule of consequence, the set of verification rules for concurrent programs is seriously incomplete. Consider the correctness formula

$$\{x = y\}\ x := x+1 \parallel y := y+1\ \{x = y\} \tag{20}$$

For (HPar) to be applicable, one needs to adapt specifications to fulfil the non-interference side-condition. Adaptation completeness is crucial. In the case of (20) one needs to introduce an auxiliary variable to untangle the dependencies. More precisely, in the derivation of Fig. 6, the auxiliary variable Z has been introduced to capture the value of the variables x and y in the initial state. This is a traditional input/output specification which is not adequately supported in the standard Hoare Logic presentation. With Hoare's version of the consequence rule, one cannot derive (20).

$$\dfrac{\dfrac{\vdash_{\text{Hoare}} \{x = Z\}\ x := x+1\ \{x = Z+1\} \qquad \vdash_{\text{Hoare}} \{y = Z\}\ y := y+1\ \{y = Z+1\}}{\vdash_{\text{Hoare}} \{x = Z \wedge y = Z\}\ x := x+1 \parallel y := y+1\ \{x = Z+1 \wedge y = Z+1\}}\ \text{HPar}}{\vdash_{\text{Hoare}} \{x = y\}\ x := x+1 \parallel y := y+1\ \{x = y\}}\ \text{HCons*}$$

Figure 6: Auxiliary Variables Avoid Interference

Previous approaches have been more cumbersome. Auxiliary variables are seen as a necessary evil that cannot be avoided for dealing with concurrency. Typically, the only rule for dealing with auxiliary variables is an elimination rule. This approach has been put forward by Owicki and Gries [OG76a]. In the process of verification, one needs to even include program segments dealing with auxiliary variables. For deriving the correctness formula (20), they suggest to first establish

$$\vdash_{\text{Hoare}} \{x = y\}\ Z := x;\ [x := x+1 \parallel y := y+1]\ \{x = y\}$$

and then eliminate assignments to auxiliary variables with the new structural rule

$$\frac{\vdash_{\text{Hoare}} \{p\}\ S\ \{q\}}{\vdash_{\text{Hoare}} \{p\}\ S^*\ \{q\}} \tag{21}$$

where for some set of auxiliary variables A of S with $\mathit{free}(q) \cap A = \emptyset$, the program S^* is obtained from S by deleting all assignments to the variables in A.

7.2 Parallel Programs with Shared Variables

In the Owicki-Gries approach [OG76a], auxiliary take on a new rôle in the setting or parallel programs with shared variables. In addition to relating the value of program variables at various states during the execution, they also record the computation history. Employing one of the standard examples in this area [AO91], we show that, courtesy of our stronger rule of consequence, it suffices to introduce auxiliary variables as reference points during the computation.

For parallel programs sharing variables, the side-condition of (HPar) is too strong. Instead of a syntactic check, the following rule due to Owicki and Gries allows an analysis of whether all parallel components preserve assertions .

$$\frac{\{p_i\}\ S_i\ \{q_i\} \quad \text{for } i \in \{1,\ldots,n\}}{\{\bigwedge_{i=1}^{n} p_i\}\ S_1 \parallel \ldots \parallel S_n\ \{\bigwedge_{i=1}^{n} q_i\}}$$

provided the set $\{\{p_i\}\ S_i\ \{q_i\} \mid i \in \{1,\ldots,n\}\}$ is interference free. (OGPar)

The (subtle) details of interference freedom are not important for the purpose of this paper. Suffice to say that in the particular case of $n = 2$ where S_1 and S_2 are both atomic statements e.g., assignments, the side condition triggers the two proof obligations $\vdash_{\text{Hoare}} \{p_1 \wedge q_2\}\ S_1\ \{q_2\}$ and $\vdash_{\text{Hoare}} \{p_2 \wedge q_1\}\ S_2\ \{q_1\}$.

Consider the two programs $S_1 \equiv x := 0$ and $S_2 \equiv x := x+2$ running in parallel. Depending on the order of execution, the value of x in the final state could be either 0 or 2. We show in Fig. 7 that $\{\mathbf{true}\}\ S_1 \parallel S_2\ \{x = 0 \vee x = 2\}$ is derivable with the help of (OGPar) and our rule of consequence. As an auxiliary variable X, we capture the value of the program variable x in the state before executing S_2.

$$\dfrac{\dfrac{\vdash_{\text{Hoare}} \{q_1[x \mapsto 0]\}\ S_1\ \{q_1\}}{\vdash_{\text{Hoare}} \{\mathbf{true}\}\ S_1\ \{q_1\}}\ \text{HCons} \qquad \dfrac{\vdash_{\text{Hoare}} \{q_2[x \mapsto x+2]\}\ S_2\ \{q_2\}}{\vdash_{\text{Hoare}} \{x = X\}\ S_2\ \{q_2\}}\ \text{HCons}}{\dfrac{\vdash_{\text{Hoare}} \{\mathbf{true} \wedge x = X\}\ S_1 \parallel S_2\ \{q_1 \wedge q_2\}}{\vdash_{\text{Hoare}} \{\mathbf{true}\}\ S_1 \parallel S_2\ \{x = 0 \vee x = 2\}}\ \text{HCons*}}\ \text{OGPar}$$

with $q_1 \equiv x = 0 \vee X = 0$ and $q_2 \equiv x = X + 2 \vee x = 0$.

Figure 7: Auxiliary Variables and Concurrent Programs with Shared Variables

It is essential to employ the new rule of consequence (HCons*). It triggers the side condition

$$\forall x \cdot \exists X \cdot x = X \wedge \forall x \cdot \big((x = 0 \vee X = 0) \wedge (x = X + 2 \vee x = 0)\big) \Rightarrow (x = 0 \vee x = 2)\ .$$

Previously, the above program could only be derived by modifying the program itself and taking advantage of the rule for eliminating auxiliary variables (21). Specifically, one needed to consider an equivalent program in which a flag *DONE* gets set immediately after S_2 has been executed [AO91]:

$$\vdash_{\text{Hoare}} \{\mathbf{true}\}\ \mathit{DONE} := \mathbf{false};\ \underline{\mathbf{begin}}\ S_1 \parallel \langle S_2;\ \mathit{DONE} := \mathbf{true}\rangle\ \underline{\mathbf{end}}\ \{x = 0 \vee x = 2\}$$

where $\langle S_2;\ \mathit{DONE} := \mathbf{true}\rangle$ denotes an atomic region. Then, all assignments to the auxiliary variable *DONE* are eliminated by appealing to the rule (21).

Encouraged by the new derivations of Fig. 6 and 7, it seems plausible that a more principled treatment of auxiliary variables would lead to simpler presentations of Hoare Logic for concurrent programs. However, at present, we do not have a definite result. In particular, we do not see how to arrive at

$$\vdash_{\text{Hoare}} \{x = 0\}\ x := x+1 \parallel x := x+1\ \{x = 2\}$$

without the elimination rule (21) [OG76b]. We suspect that modifications need to be made to the rule of parallel composition (OGPar) itself to reflect the rôle of auxiliary variables. Stølen [Stø91] has already demonstrated that one can design a complete verification calculus for concurrent programs and shared variables without having to modify the program text. We return to Stølen's contribution in the following section.

7.3 Compositionality

Owicki and Gries' rule for parallel composition (OGPar) is not compositional. To guarantee interference freedom, depending on the details of S_1 and S_2, one needs to generate further correctness formulae and check their derivability. This can be avoided by annotating correctness formulae by rely and guarantee conditions [Sti88]. Then, as a necessary criterion for parallel composition of two programs, the rely conditions of one must be guaranteed by the guarantee conditions of the other. Hoare's rule of consequence and Owicki and Gries' elimination rule for auxiliary variables are lifted to this generalisation. It seems plausible that a version of our new rule of consequence as the only structural rule would simplify presentations of compositional verification calculi for concurrency. Stirling writes

> "A major disadvantage of both the original Owicki-Gries system and its reformulation here is the need for the auxiliary variable rule."

Finally, we would like to remark on Stølen's approach [Stø91]. This differs from Stirling's calculus in two respects:

1. The elimination rule for auxiliary variables is more sophisticated. Like our new rule of consequence, one only needs to deal with auxiliary variables at the level of *assertions*.
2. Specifications are written in VDM format. Specifically, the precondition denotes a unary predicate and the postcondition a binary predicate on the state space.

Again, it seems plausible that our rule of consequence subsumes Stølen's (four) structural rules. Further improvements are likely. Stølen employs auxiliary variables and assertions in VDM format. We have shown in Sec. 5 that this is redundant. VDM correctness formulae are merely a special case of Hoare Logic correctness formulae with auxiliary variables.

8 On Developing Proofs on a Machine

We have taken advantage of the proof assistant LEGO [Pol94] to interactively develop machine-checked soundness and completeness proofs. With computer-aided proof systems, one attains a high-level of confidence concerning the correctness of machine-checked proofs e.g., in LEGO, less than 200 lines of SML code are responsible for checking that a given derivation is correct. Today's proof tools are sufficiently advanced to be of significant help in investigating meta-theory of verification calculi. In our opinion, soundness and completeness are difficult concepts which have to be tackled by appealing to intricate induction principles. A computer-aided proof system is very good at managing such tasks. For example, employing the system Isabelle, Nipkow has uncovered a bug in Winskel's completeness proof for Hoare Logic [Nip98].

Conducting formal proofs at a level that they can be checked by a machine requires that all aspects and all details must be thoroughly investigated. In our formalisation, we have taken the liberty of omitting the issue of representing *syntax* for expressions and assertions. Since the semantics of assertions has been characterised with the help of the proof tool's native higher-order logic, expressiveness was trivially guaranteed. This issue is explored in more detail in [Kle98b].

9 Conclusions

In Hoare Logic, assertions characterise predicates on states. Therefore, one cannot directly specify input/output behaviour. In actual examples, one employs auxiliary variables as additional points of reference. This crucial rôle of auxiliary variables has been underestimated in previous presentations of axioms and rules for Hoare Logic.

As our main contribution, we stipulate a new structural rule. It allows to adjust auxiliary variables while strengthening preconditions and weakening postconditions. The new rule subsumes all other previously suggested structural rules. Courtesy of this rule, one may adapt arbitrary satisfiable specifications. This leads to a number of improvements:

- We were able to show that, contrary to common belief, VDM's operation decomposition rules are more restrictive than Hoare Logic in that every derivation in VDM can be naturally embedded in Hoare Logic.
- One may derive completeness as a corollary of Gorelick's Most General Formula theorem. The proof works for arbitrary language features.
- We gain a significantly simpler presentation for Hoare Logic dealing with recursive procedures. No further structural rules are required.
- Similar simplifications seem possible for Hoare Logic dealing with concurrency.

Further work is required to substantiate the last claim. We have so far only been able to show that two of the standard examples in the area of concurrency can be derived without the need for any further structural rules. It seems likely that rules for parallel composition need to be redesigned in the light of the more principled approach to auxiliary variables.

Acknowledgements

I would like to thank Rod Burstall, Martin Hofmann and Paul Jackson for discussing the rôle of auxiliary in Hoare Logic with me on many occasions. This presentation has also benefited from constructive comments by Stephen Gilmore. Moreover, I would like to acknowledge the financial support of EPSRC (Applications of a Type-Theoretic Theorem Prover), the British Council (ARC Project Co-Development of Object-Oriented Programs in LEGO) and the European Commission (Marie Curie Fellowship).

References

[Acz82a] Peter Aczel. A note on program verification. Unpublished, see also [Jon86], January 1982.

[Acz82b] Peter Aczel. A system of proof rules for the correctness of iterative programs – some notational and organisational suggestions. Unpublished, August 1982.

[AdB90] Pierre America and Frank de Boer. Proving total correctness of recursive procedures. *Information and Computation*, 84(2):129–162, 1990.

[AM80] Krzysztof R. Apt and Lambert G. L. T. Meertens. Completeness with finite systems of intermediate assertions for recursive program schemes. *SIAM Journal on Computing*, 9(4):665–671, November 1980.

[AO91] Krzysztof R. Apt and Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. Texts and Monographs in Computer Science. Springer, New York, 1991.

[Apt81] Krzysztof R. Apt. Ten years of Hoare's logic: A survey – part I. *ACM Transactions on Programming Languages and Systems*, 3(4):431–483, October 1981.

[BM93] Rod Burstall and James McKinna. Deliverables: a categorical approach to program development in type theory. In Andrzej M. Borzyszkowski and Stefan Sokołowski, editors, *Proceedings of MFCS '93*, volume 711 of *Lecture Notes in Computer Science*, pages 32–67. Springer-Verlag, September 1993. An earlier version appeared as LFCS technical report ECS-LFCS-92-242 in 1992.

[Cli81] M. Clint. On the use of history variables. *Acta Informatica*, 16:15–30, 1981.

[Coo78] Stephen A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7(1):70–90, February 1978.

[Cou90] Patrick Cousot. Methods and logics for proving programs. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 15, pages 841–993. Elsevier, 1990.

[Dah92] Ole-Johan Dahl. *Verifiable Programming*. International Series in Computer Science. Prentice Hall, 1992.

[dB80] Jaco de Bakker. *Mathematical Theory of Program Correctness*. Prentice Hall, 1980.

[Flo67] Robert W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Proc. Symp. in Applied Mathematics*, volume 19, pages 19–32, 1967.

[GL80] David Gries and Gary Levin. Assignment and procedure call proof rules. *ACM Transactions on Programming Languages and Systems*, 2(4):564–579, 1980.

[Gor75] Gerald Arthur Gorelick. A complete axiomatic system for proving assertions about recursive and non-recursive programs. Technical Report 75, Department of Computer Science, University of Toronto, 1975.

[Gri81] David Gries. *The Science of Computer Programming*. Springer, 1981.

[HJ89] C. A. R. Hoare and Cliff B. Jones, editors. *Essays in Computing Science*. International Series in Computer Science. Prentice Hall, 1989.

[HM96] Peter V. Homeier and David F. Martin. Mechanical verification of mutually recursive procedures. In M. A. McRobbie and J. K. Slaney, editors, *Automated Deduction – CADE-13*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 201–215, New Brunswick, NJ, USA, July/August 1996. Springer-Verlag. 13th International Conference on Automated Deduction.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969. Also in [HJ89].

[Hoa71] C. A. R. Hoare. Procedures and parameters: An axiomatic approach. In E. Engeler, editor, *Symposium on Semantics of Algorithmic Languages*, volume 188 of *Lecture Notes in Mathematics*, pages 102–116. Springer-Verlag, 1971. Also in [HJ89].

[Hoa75] C. A. R. Hoare. Parallel programming: An axiomatic approach. *Computer Languages*, 1(2):151–160, 1975. Also in [HJ89].

[Hof97] Martin Hofmann. Semantik und Verifikation, 1997. Lecture Notes for a course held in Winter term 1997/98 at the University of Marburg. In German.

[Jon86] Cliff B. Jones. Systematic program development. In N. Gehani and A. MacGettrick, editors, *Software Specification Techniques*, pages 89–109. Addison-Wesley, 1986.

[Jon90] Cliff B. Jones. *Systematic Software Development Using VDM*. International Series in Computer Science. Prentice Hall, 2 edition, 1990.

[JS90] Cliff B. Jones and Roger C. F. Shaw. *Case Studies in Systematic Software Development*. Prentice Hall, 1990.

[Kle98a] Thomas Kleymann. Hoare Logic and VDM: Machine-checked soundness and completeness proofs. PhD thesis ECS-LFCS-98-392, Laboratory for Foundations of Computer Science, University of Edinburgh, September 1998.

[Kle98b] Thomas Kleymann. Metatheory of verification calculi in LEGO – to what extent does syntax matter? Technical Report ECS-LFCS-98-393, Laboratory for Foundations of Computer Science, University of Edinburgh, September 1998.

[LGH$^+$78] R. L. London, J. V. Guttag, J. J. Horning, B. W. Lampson, J. G. Mitchell, and G. J. Popek. Proof rules for the programming language Euclid. *Acta Informatica*, 10:1–26, 1978.

[Mor] James H. Morris. Comments on "procedures and parameters". Undated and unpublished.

[Mor88] Carroll Morgan. Procedures, parameters, and abstraction: separate concerns. *Science of Computer Programming*, 11:17–27, 1988.

[Mor90] Carrol C. Morgan. *Programming from Specifications*. Prentice Hall, 1990.

[Nip98] Tobias Nipkow. Winskel is (almost) right: Towards a mechanized semantics textbook. *Formal Aspects of Computing*, 1998. To appear.

[OG76a] Susan Owicki and David Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, 6(4):319–340, 1976.

[OG76b] Susan Owicki and David Gries. Verifying properties of parallel programs: An axiomatic approach. *Communications of the ACM*, 19(5):279–285, May 1976.

[Old81] Ernst-Rüdiger Olderog. Sound and complete Hoare-like calculi based on copy rules. *Acta Informatica*, 16:161–197, 1981.

[Old83] Ernst-Rüdiger Olderog. On the notion of expressiveness and the rule of adaptation. *Theoretical Computer Science*, 24:337–347, 1983.

[Pol94] Randy Pollack. *The Theory of LEGO, A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, Laboratory for Foundations of Computer Science, University of Edinburgh, 1994.

[Rey82] John C. Reynolds. Idealized Algol and its specification logic. In D. Néel, editor, *Tools & Notions for Program Construction*. Cambridge University Press, 1982.

[Sch97] Thomas Schreiber. Auxiliary variables and recursive procedures. In Michel Bidoit and Max Dauchet, editors, *Proceedings of TAPSOFT '97*, volume 1214 of *Lecture Notes in Computer Science*, pages 697–711, Lille, France, April 1997. Springer-Verlag.

[Sok77] Stefan Sokołowski. Total correctness for procedures. In J. Gruska, editor, *Sixth Mathematical Foundations of Computer Science (Tatranská Lomnica)*, volume 53 of *Lecture Notes in Computer Science*, pages 475–483. Springer-Verlag, 1977.

[Sou84] N. Soundararajan. A proof technique for parallel programs. *Theoretical Computer Science*, 31:13–29, 1984.

[Sti88] Colin Stirling. A generalization of Owicki-Gries's Hoare Logic for a concurrent while language. *Theoretical Computer Science*, 58:347–359, 1988.

[Stø91] Ketil Stølen. A method for the development of totally correct shared-state parallel programs. In Jos C. M. Baeten and Jan Friso Groote, editors, *Proceedings of CONCUR '91*, volume 527 of *Lecture Notes in Computer Science*, pages 510–525. Springer-Verlag, 1991.

[Tar85] Andrzej Tarlecki. A language of specified programs. *Science of Computer Programming*, 5:59–81, 1985.

[Vic91] Steven Vickers. Formal implementation. In John A. McDermid, editor, *Software engineer's reference book*, chapter 25. Butterworth-Heinemann, London, 1991.

[Zwi89] Job Zwiers. *Compositionality, Concurrency and Partial Correctness*, volume 321 of *Lecture Notes in Computer Science*. Springer, 1989.