Acta Informatica 16, 161-197 (1981)

Sound and Complete Hoare-like Calculi Based on Copy Rules

Ernst-Rüdiger Olderog

Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, Olshausenstr. 40-60, D-2300 Kiel 1, Germany (Fed. Rep.)

Contents

Summary. This paper presents a uniform approach to known and new results on relative completeness of Hoare-like calculi for languages of ALGOL-like programs with procedures as procedure parameters. First the notion of a copy rule is introduced. It provides a uniform framework for dealing with different variants of semantics reaching from dynamic to static scope. Then for each copy rule $\mathscr{C}$ a Hoare-like calculus $\mathscr{H}(\mathscr{C})$ is presented, the soundness of which is shown by using an approximating semantics. The key to the completeness results lies in a general completeness theorem on the calculi $\mathscr{H}(\mathscr{C})$ which has these results as corollaries. Finally, a new type of theorem on Hoare-like calculi is proved by which the notion of formal provability in $\mathscr{H}(\mathscr{C})$ is completely characterized. This characterization theorem is the main result of the paper. It covers both soundness and completeness of the calculi $\mathscr{H}(\mathscr{C})$ and additionally gives an idea of what the limits of presently known Hoare-like proof techniques for programming languages with procedures are.

0001-5903/81/0016/0161/$07.40

1. Introduction

Hoare-like proof systems [18] have been proposed for various language constructs [2, 6]. Since they are intended to *capture* the (partial) correctness theory of these constructs, it is necessary to supply rigorous proofs of soundness and completeness (in some appropriate sense) for these systems. Among the language constructs which are difficult or even impossible to deal with in such a way is the procedure concept.

The purpose of this paper is to present a uniform approach to known and new results on (soundness and) relative completeness of Hoare-like systems. This is done in the framework of ALGOL-like programs where the attention is restricted to problems which are due to procedures. For a better understanding of these problems let us briefly discuss several approaches which can be found in the literature.

The first major investigation of the completeness problem for Hoare-like systems was presented by Cook [10]. He introduced the notion of relative completeness and proved his system to be relatively complete for programs with non-recursive procedures. Gorelick [13] extended Cook's result to programs with recursive procedures. Languages of programs allowing procedures as procedure parameters were studied by Clarke [9]. He proved that it is impossible to obtain a sound and relatively complete Hoare-like system for such languages unless some restrictions of the procedure concept are accepted. Clarke proposed several such restrictions and stated corresponding completeness results which are farer reaching than those of Cook and Gorelick. Further completeness results can be found in Apt [1], de Bakker [6] and Harel et al. [17].

However, these results are not completely satisfying. Both Cook [10] and Gorelick [13] consider dynamic scope instead of static scope semantics as observed in Donahue [12]. Also Clarke [9] proves only his result on dynamic scope. (As we shall see in our paper, his methods must be extended in order to prove his claimed results on static scope.) Static scope semantics is used in Apt [1] and de Bakker [6], but neither Apt's nor de Bakker's proof techniques - though very sophisticated - are sufficient to prove Clarke's results since Apt considers parameterless procedures only whereas de Bakker disallows procedure nestings. Also in Harel et al. [17] procedure nestings are disallowed.

Additionally, in all papers above restrictions are imposed on the actual parameters of procedure calls: Sharing (or aliasing) cannot occur or is explicitly disallowed. This is also true for the programming language EUCLID, a major design goal of which was the verifiability of programs [16, 34].

Sound and relatively complete rules which deal with sharing are proposed in Cartwright and Oppen [8]. On the other hand, this paper disallows procedures as parameters and restricts global variables in such a way that static scope reduces to dynamic scope.

According to this discussion the aims in our paper are the following. Firstly, we would like to provide rigorous proofs for Clarke's completeness claims on languages with static scope and procedures as parameters. Our proof methods should also cover previous results dealing with dynamic scope only. Secondly, our completeness results should hold for languages without restrictions concerning sharing, procedure nestings and global variables. Thirdly, since the first two aims clearly lead to a discussion of various cases of languages we would like to prove all these completeness results in a uniform way, namely by deducing them from one general completeness theorem. Finally, we would like to have some clear idea of what the limit of our method proposed here is and thus - by the first aim - what the limits of presently known Hoare-like proof techniques are as far as procedures are concerned.

To explain how we try to reach these aims let us now outline the structure of this paper. Section 2 introduces a number of basic notions. Special emphasis is put on the concept of program function since it allows us to verify the easy proof rules in advance - before introducing the actual syntax of the programs. This is done in Sect. 3 where we define the class L_{Algol} of ALGOL-like programs.

In order to deal with different variants of semantics for L_{Algol} in a uniform way we introduce the notion of a copy rule in Sect. 4. A semantics defined in a copy rule style is used in several papers (e.g. Clarke [9], Lauer [32]). However, our treatment of copy rules is more related to the formal approach presented in Langmaack [26]. Three particular copy rules are studied in our paper: the naive copy rule (yielding a semantics closely related to dynamic scope), the "most recent" copy rule and the ALGOL 60 copy rule (yielding static scope semantics).

In Sect. 5 we introduce the concept of substitutional equivalent programs, i.e. programs which differ only syntactically by a certain substitution. In the Substitution Lemma we show that such programs differ also semantically only by this substitution. Because of this lemma the notion of substitutional equivalence plays a central role later on in the proof of the Completeness Theorem. In Sect. 6 we discuss the problems which arise when the sharing or aliasing restriction is assumed in programming languages and we show how to overcome this restriction by using the methods developed in Sect. 5.

In Sect. 7 a Hoare-like calculus $\mathscr{H}(\mathscr{C})$, depending on the chosen copy rule $\mathscr{C}$, is presented. The term "calculus" instead of "system" indicates that $\mathscr{H}(\mathscr{C})$ is based on the same simple notion of formal proof as Gentzen's sequent calculi for first order logic. This leads - together with our particular interpretation of proof lines which is defined by means of an approximating semantics - to a fairly simple proof of the Soundness Theorem in Sect. 8.

Section 9 investigates the completeness problem of $\mathscr{H}(\mathscr{C})$. It is clear from Clarke's results that $\mathscr{H}(\mathscr{C})$ is necessarily incomplete for the full language L_{Algol} when the ALGOL 60 copy rule (static scope) is applied [9]. Thus $\mathscr{H}(\mathscr{C})$ can only be relatively complete when we restrict ourselves either to proper sublanguages of L_{Algol} or to simpler copy rules (inducing simplified semantics). To handle such restrictions in a uniform way we introduce the notion of $\mathscr{C}$-bounded programs. It turns out that these are exactly the programs which have a so-called finite $\mathscr{C}$-index w.r.t. the substitutional equivalence. This fact allows us to prove in the Completeness Theorem that $\mathscr{H}(\mathscr{C})$ is relatively complete provided all programs in the considered language are $\mathscr{C}$-bounded.

Applications of this theorem are presented in Sect. 10 where we arrive at new completeness results on ALGOL-like programs, viz. that the calculi $\mathscr{H}(\mathscr{C})$ are relatively complete

- for the full language L_{Algol} provided one of the following copy rules is applied:

1. naive copy rule
2. "most recent" copy rule

- for the ALGOL 60 copy rule provided the considered language satisfies one of the following restrictions:

3. no global formal procedure identifiers
4. the formal "most recent" property holds.

In all cases unrestricted use of sharing, procedure nestings and global variables is allowed. We relate our results to results and claims published in the literature.

In Sect. 11 we investigate the problem whether the converse of the Completeness Theorem holds, i.e. whether formal provability in $\mathscr{H}(\mathscr{C})$ implies $\mathscr{C}$-boundedness. We show in the Characterization Theorem that this is true. This theorem is the main result of our paper since it precisely characterizes power and limits of our calculi $\mathscr{H}(\mathscr{C})$ and covers both the Soundness and the Completeness Theorem.

Finally, in Sect. 12, we discuss two issues related to the Characterization Theorem. One issue addresses the notion of Hoare logic. The second issue is motivated by one of Clarke's [9] completeness claim and concerns programs without self-application and without global variables. We show that presently known Hoare-like proof methods – including our calculi $\mathscr{H}(\mathscr{C})$ – are not powerful enough to deal with these programs. This observation (stated also in [31]) indicates further directions for research.

2. Preliminary Concepts

Syntactic objects considered in this paper are strings over an infinite set T of *tokens* or *basic symbols* $(t \in T)$[1]. As usual T^* denotes the set of all strings and ε the empty string over T. In particular we have the disjoint, infinite subsets $VI \subseteq T$ and $PI \subseteq T$ of *variable identifiers* or simply *variables* $(x, y, \ldots \in VI)$ and *procedure identifiers* $(p, q, \ldots \in PI)$ respectively. Additionally, there are constants, operators, relators and special symbols such as **begin** or **end** in T. $ID = VI \cup PI$ is the set of *identifiers* $(\alpha, \beta, \ldots \in ID)$. We use the notation $\bar{\alpha}$ for lists $\alpha_1, \ldots, \alpha_n$ of identifiers. The length n of $\bar{\alpha}$ is denoted by $|\bar{\alpha}|$ and the set $\{\alpha_1, \ldots, \alpha_n\}$ of list components by $\{\bar{\alpha}\}$.

As logical basis for our correctness investigations serves a first order language FOL with "$=$" as equality symbol. FOL determines the sets EX of *expressions* or *terms* $(e \in EX)$, LF of *logical formulas* or simply *formulas* $(P, Q, \ldots \in LF)$ and BE of *Boolean expressions* $(b \in BE)$, i.e. of quantifier-free formulas. These sets are defined in the usual way using variables, constants, operators and relators. We assume that there is a special constant term ω. The set of free variables in a formula P is denoted by free(P).

Throughout this paper we assume that a certain *interpretation* $\mathscr{I}$ of FOL with domain $\mathscr{D} \neq \emptyset$ $(d \in \mathscr{D})$ is given. $\mathscr{I}$ is said to be *finite* if $|\mathscr{D}| < \infty$ holds (where $|\mathscr{D}|$ denotes the cardinality of $\mathscr{D}$). The meaning of expressions and formulas in $\mathscr{I}$ depends on the values of the (free) variables. These values are determined by means of *states*, i.e. mappings[2] σ: $VI \to \mathscr{D}$. (We shall soon see that this simple

[1] Together with the definition of a certain class of objects we usually list typical elements of this class

[2] Unless explicitly stated we always assume mappings (functions) to be totally defined

notion of state is sufficient to describe the meaning of blockstructured programs.) Now the domain value $\mathscr{I}(e)(\mathfrak{s})$ of an expression e and the truth value $\mathscr{I}(P)(\mathfrak{s})$ of a formula P in $\mathscr{I}$ w.r.t. a state $\mathfrak{s}$ can be defined in a standard way. We write $\models_{\mathscr{I},\mathfrak{s}} P$ if $\mathscr{I}(P)(\mathfrak{s})$ is true and $\models_{\mathscr{I}} P$ if $\models_{\mathscr{I},\mathfrak{s}} P$ is true for every state $\mathfrak{s}$. The set of all states is denoted by $\mathscr{St}(\mathfrak{s}\in\mathscr{St})$. $\mathfrak{s}\{d/x\}$ is that variant $\mathfrak{s}'$ of state $\mathfrak{s}$ with $\mathfrak{s}'(x)=d$ and $\mathfrak{s}'(y)=\mathfrak{s}(y)$ for $y\neq x$ (cf. [6]). For subsets $X\subseteq VI$ the restriction of $\mathfrak{s}$ to X is denoted by $\mathfrak{s}\restriction X$. The *theory* $Th(\mathscr{I})$ of $\mathscr{I}$ is given by $Th(\mathscr{I})=\{P\mid\models_{\mathscr{I}} P\}$. By $\mathscr{St}^{\mathscr{I}}(P)$ we denote the set of all states *expressed* by P, i.e. $\mathscr{St}^{\mathscr{I}}(P)=\{\mathfrak{s}\mid\models_{\mathscr{I},\mathfrak{s}} P\}$.

Next we develop the concept of substitution which plays a central role in our paper. Let $[\tau_1,\ldots,\tau_n/\alpha_1,\ldots,\alpha_n]$ with $\alpha_i\in ID(\subseteq T)$ and $\tau_i\in T\cup EX$ denote the following relation ρ between T and $T\cup EX$:

$$\rho=\{(\alpha_1,\tau_1),\ldots,(\alpha_n,\tau_n)\}\cup\{(t,t)\mid t\in T \text{ and } t\neq\alpha_1,\ldots,\alpha_n\}.$$

Then a *general substitution* σ is a mapping of the form

$$\sigma=[\tau_1,\ldots,\tau_n/\alpha_1,\ldots,\alpha_n]$$

with $\sigma(VI)\subseteq EX$ and $\sigma(PI)\subseteq PI$. σ is called a *substitution* if additionally $\sigma(VI)\subseteq VI$ $(\subseteq EX)$ holds. σ is said to be *injective on* $X\subseteq VI$ if $\sigma\restriction X$ is an injective mapping. We distinguish between several *applications* of a general substitution σ:

(1) A "left-hand" application of σ indicates that σ is used as a word homomorphism: $\sigma(t_1\ldots t_n)=\sigma(t_1)\ldots\sigma(t_n)$ where $t_i\in T$.

(2) The notation $P\sigma$ is well-known. (Recall that bound variables in P have to be renamed in order to avoid clashes with inserted variables.)

(3) If σ is a substitution, $\mathfrak{s}\sigma$ denotes the state with $\mathfrak{s}\sigma(x)=\mathfrak{s}(\sigma(x))$ for all $x\in VI$.

Further applications will be defined in connection with program functions and blocks.

Subsequently we study those transition functions between states which can be specified by ALGOL-like programs, called here program functions. We do this to single out several basic properties of these functions – needed later on in proofs of soundness and completeness – which do not depend on the actual syntax of the programs.

Let X be a finite subset of VI and f be a partially defined function from $\mathscr{St}$ to $\mathscr{St}$. Then f is called a *program function on* X if the following properties hold:

(1) If $f(\mathfrak{s})=\mathfrak{s}'$ then $\mathfrak{s}\restriction VI\setminus X=\mathfrak{s}'\restriction VI\setminus X$.

(2) If $f(\mathfrak{s})=\mathfrak{s}'$ and $\mathfrak{s}_1\restriction X=\mathfrak{s}\restriction X$ for some state $\mathfrak{s}_1$ then there exists a state $\mathfrak{s}_1'$ with $f(\mathfrak{s}_1)=\mathfrak{s}_1'$ and $\mathfrak{s}_1'\restriction X=\mathfrak{s}'\restriction X$.

Intuitively speaking f can manipulate and inspect only the finitely many variables in X. Thus X can be considered as the set of input-output or "active" variables of f. According to Schwarz [42] property (1) is called *stability* and (2) *aloofness* of f w.r.t. the "inactive" variables in $VI\setminus X$. f is called a *program function* if there exists an X such that f is a program function on X. The *image* $f(\mathscr{S})$ of $\mathscr{S}\subseteq\mathscr{St}$ under f is given by

$$f(\mathscr{S})=\{\mathfrak{s}'\mid \text{there exists a state } \mathfrak{s} \text{ with } f(\mathfrak{s})=\mathfrak{s}'\}.$$

We now consider several examples of program functions which will be used in Sect. 4. Let f and g be program functions on X and Y respectively and σ be a substitution which is injective on X. Then the program functions

(*) $$\Omega, \text{ass}_{\mathscr{I}}(x, e), \text{if}_{\mathscr{I}}(b, f, g), \text{block}_{\mathscr{I}}(x, f), f\sigma$$

are defined as follows:

- Ω denotes the empty function.
- $\text{ass}_{\mathscr{I}}(x, e)(\mathfrak{s}) = \mathfrak{s}\{\mathscr{I}(e)(\mathfrak{s})/x\}$. Note that $\text{ass}_{\mathscr{I}}(x, e)$ is a totally defined program function on $\text{free}(e) \cup \{x\}$.
- If $\mathscr{I}(b)(\mathfrak{s})$ is true then $\text{if}_{\mathscr{I}}(b, f, g)(\mathfrak{s}) = f(\mathfrak{s})$ holds. Otherwise $\text{if}_{\mathscr{I}}(b, f, g)(\mathfrak{s}) = g(\mathfrak{s})$. Thus $\text{if}_{\mathscr{I}}(b, f, g)$ is a program function on $\text{free}(b) \cup X \cup Y$.
- $\text{block}_{\mathscr{I}}(x, f)(\mathfrak{s}) = \mathfrak{s}'$ iff there exists a state $\mathfrak{s}''$ with $f(\mathfrak{s}\{\mathscr{I}(\omega)/x\}) = \mathfrak{s}''$ and $\mathfrak{s}' = \mathfrak{s}''\{\mathfrak{s}(x)/x\}$.

This definition for blocks is motivated by Sieber [43]. Due to the implicit stack mechanism expressed in the equation $\mathfrak{s}' = \mathfrak{s}''\{\mathfrak{s}(x)/x\}$ neither *addresses* as in [1] or [6] nor *explicit stack mechanisms* as in [9] or [37] are needed here. For simplicity we assume that the local variable x is initialized with the value of ω when entering the block. Note that $\text{block}_{\mathscr{I}}(x, f)$ is a program function on $X \setminus \{x\}$.

- $(f\sigma)(\mathfrak{s}) = \mathfrak{s}'$ iff there exists a state $\mathfrak{s}''$ with $f(\mathfrak{s}\sigma) = \mathfrak{s}''$ such that for all $y \in VI$

$$\mathfrak{s}'(y) = \begin{cases} \mathfrak{s}''(\sigma^{-1}(y)) & \text{if } y \in \sigma(X) \\ \mathfrak{s}(y) & \text{otherwise} \end{cases}$$

holds where $\sigma^{-1}(y)$ denotes the uniquely determined $x \in X$ with $\sigma(x) = y$.

$f\sigma$ is said to result from f by an application of σ and is a program function on $\sigma(X)$: the "active" variables $x \in X$ of f have been renamed to $\sigma(x)$. This construction will be used to compare the meaning of programs which differ only (syntactivally) by a substitution. The intuitive meaning of this definition is displayed in Fig. 1.

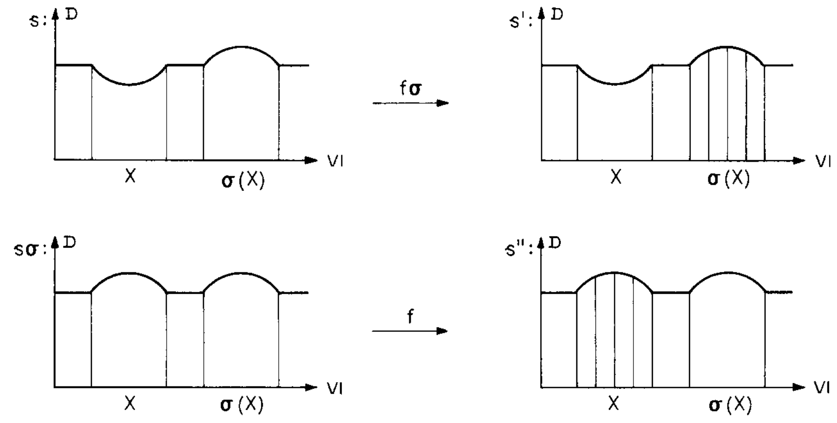


Fig. 1

At the end of this section we state those properties of program functions which are important in the proofs of soundness and completeness. For a better readability we take $f(P) \subseteq Q$ as an abbreviation for $f(\mathscr{St}^{\mathscr{I}}(P)) \subseteq \mathscr{St}^{\mathscr{I}}(Q)$

Lemma 1. *Let f, g be program functions on X and σ be a substitution. Then*
a) $f\sigma(P\sigma) \subseteq Q\sigma$ *if $f(P) \subseteq Q$ and σ is injective on $X \cup free(P \vee Q)$.*
b) $f\sigma(P\sigma) \supseteq Q\sigma$ *if $f(P) \supseteq Q$ and σ is injective on $X \cup free(P \vee Q)$.*
c) $f(P\sigma) \subseteq Q\sigma$ *if $f(P) \subseteq Q$ and $\sigma = [x_1, \ldots, x_n / y_1, \ldots, y_n]$ such that for all $k = 1, \ldots, n$ i) $y_k \notin X$ and ii) $x_k \notin X$ if $y_k \in free(Q)$.*
d) $f(P \wedge R) \subseteq Q \wedge R$ *if $f(P) \subseteq Q$ and $free(R) \cap X = \emptyset$.*
e) $\Omega(P) \subseteq Q$ *for all $P, Q \in LF$.*
f) $\mathrm{ass}_{\mathscr{I}}(x, e)(P) \subseteq Q$ *iff $\models_{\mathscr{I}} P \rightarrow Q[e/x]$.*
g) $(f \circ g)(P) \subseteq Q$ *iff there exists a set $Y \subseteq \mathscr{St}$ such that $f(P) \subseteq Y$ and $g(Y) \subseteq Q$.*
h) $\mathrm{if}_{\mathscr{I}}(b, f, g)(P) \subseteq Q$ *iff $f(P \wedge b) \subseteq Q$ and $g(P \wedge \neg b) \subseteq Q$.*
i) $\mathrm{block}_{\mathscr{I}}(x, f)(P) \subseteq Q$ *iff $f(P[y/x] \wedge x = \omega) \subseteq Q[y/x]$ where $y \notin X \cup free(P \vee Q)$.*

The proof is left to the reader. Observe that f) might fail if $\mathrm{ass}_{\mathscr{I}}(x, e)$ is not totally defined and that there might be no formula R with $Y = \mathscr{St}^{\mathscr{I}}(R)$ in g) (cf. [10, 44] and Sect. 9).

3. ALGOL-like Programs

In this section we introduce the class of programs we are interested in, and list a few notions concerning programs (cf. [26]). First we define the sets STM of *statements* ($S \in STM$), BLK of *blocks* ($B \in BLK$) and ENV of *environments* ($E \in ENV$), i.e. sequences of *procedure declarations* (*procedures* for short):

$S ::= \textbf{error} \mid x := e \mid S_1; S_2 \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \mid B \mid p(\bar{x}:\bar{q})$

$B ::= \textbf{begin } E S \textbf{ end} \mid \textbf{begin var } x;\ E S \textbf{ end}$

$E ::= \varepsilon \mid \textbf{proc } p(\bar{y}:\bar{r});\ B; \mid E_1 E_2$

In the definition of E it is required that the components of the formal parameter lists $\bar{y}$ and $\bar{r}$ are distinct. Furthermore, if $E = \textbf{proc } p_1(\ldots); \ldots; \ldots; \textbf{proc } p_n(\ldots); \ldots;$, the procedures identifiers $p_1, \ldots, p_n$ must be distinct. These are the usual well-formedness conditions for procedure declarations. No restrictions are imposed, however, on the actual parameter lists $\bar{x}$ and $\bar{q}$ of procedure calls $p(\bar{x}:\bar{q})$. For simplicity blocks with multiple variable declarations are not allowed. They can be considered as abbreviations of nested blocks with single variable declarations.

We distinguish here between strings and occurrences of strings. Formally let S be a statement of the form $S = t_1 \ldots t_n$ with $t_i \in T$. Then the pair $(i, t_i \ldots t_j)$ with $1 \leq i \leq j \leq n$ denotes the *occurrence* of the substring $t_i \ldots t_j$ at position i in S. Such occurrences can be compared by the relations "contained in" and "smaller than" which are defined in an obvious way. An occurrence (i, α) of an identifier α in S is either *free* or *bound* to a uniquely determined *defining* occurrence $(j, \alpha) = \mathrm{def}(i, \alpha)$ of α, i.e. an occurrence (j, α) such that (1) α occurs in a formal parameter list or (2) the symbol t_{j-1} preceding α is **var** or **proc**. The identifier occurrence (i, α) is called *formal* if (1) holds for $\mathrm{def}(i, \alpha)$, otherwise it is *non-formal*.

Now the class L_{Algol} of *ALGOL-like programs* or simply *programs* ($\pi \in L_{\text{Algol}}$) is defined as follows:

$\pi \in L_{\text{Algol}}$ iff π is a block without free occurrences of procedure identifiers.

Since correctness proofs in Hoare-like calculi proceed by structural induction, we need an appropriate way of talking about program segments instead of whole programs. To this end we introduce the class UNT of *units* which consists of all pairs $E|S$ such that **begin** ES **end** is a program, i.e. $E|S$ is a *closed* pair in terms of de Bakker [6]. By free $(E|S)$ we denote the set of all variables occurring freely in **begin** ES **end**. The set of identifiers occurring in $E|S$ is given by idf$(E|S)$. Notations as free(S), idf(E) or idf(S) are defined analogously.

Though an environment is defined to be a certain *sequence* of procedures, it is convenient to identify E with the *set* of these procedures. (Think of E as a standard representation of this set.) Thus we may use set-theoretic operations in connection with environments. By add(E, E_1) we denote the environment $E_0 \cup E_1$ where E_0 is obtained from E by deleting all procedure declarations **proc** $p(\ldots); \ldots;$ from E for which there exists a (new) procedure declaration with identifier p in E_1 [9].

A substring of a statement S is said to occur in the *main part* of S if it occurs outside of all procedures declared within S. Given a procedure $\mathscr{P}=$ **proc** $p(\bar{y}:\bar{r}); B;$ then $(\bar{y}:\bar{r}); B;$ is called the *extended* procedure body of $\mathscr{P}$. Blocks and extended procedure bodies form the class of *regions*. With each occurrence (i, α) in a program π we associate an occurrence of a region denoted by region$_\pi(i, \alpha)$: This is the smallest occurrence of a region which contains def(i, α) if def(i, α) is defined. Otherwise region$_\pi(i, \alpha) = (1, \alpha)$.

Consider an occurrence $(i, t_i \ldots t_j)$ of a substring in a program π. Call an occurrence $(l, \mathscr{P})$ of a *procedure* $\mathscr{P}=$ **proc** $p(\ldots); \ldots;$ in π *visible from* $(i, t_i \ldots t_j)$ if region$_\pi(l+1, p)$ properly contains $(i, t_i \ldots t_j)$ and there is no other procedure occurrence with identifier p in π the region of which is in between region$_\pi(l+1, p)$ and $(i, t_i \ldots t_j)$. Additionally an occurrence (k, α) of an *identifier* α is called *visible from* $(i, t_i \ldots t_j)$ if (k, α) is contained either in $(i, t_i \ldots t_j)$ itself or in a procedure occurrence $(l, \mathscr{P})$ visible from $(i, t_i \ldots t_j)$. The concept of visibility gives the connection between units and programs: Given a unit $E|S$ then E is the set of procedures and idf$(E|S)$ the set of identifiers visible from S.

Let a block B and a substitution σ be given. Then $B\sigma$ denotes the result of relacing every free occurrence of α in B by $\sigma(\alpha)$ (for all $\alpha \in$ idf(B)). In contrast to the definition of $P\sigma$ bound identifiers in B are not renamed. (This definition makes sense and is advantageous in connection with the application of copy rules.)

A program π is called *distinguished* if different defining occurrences of identifiers are denoted differently and no variable occurs both free and bound in π. Statements S and S' are said to have the *same structure* if they coincide after deleting all identifiers. They are called *congruent* ($S \approx S'$ for short) if they differ only by a bound renaming of identifiers. If this renaming is *injective* (i.e. if for all bound occurrences (i, α), (j, β) in S and (i, α'), (j, β') in S' $\alpha = \beta$ holds iff $\alpha' = \beta'$ holds) we write $S \underset{\text{inj}}{\approx} S'$. Note that every program $\pi \in L_{\text{Algol}}$ has (in general infinitely many) congruent distinguished programs $\pi' \approx \pi$. For simplicity we take π_d to denote a particular distinguished program $\pi_d \approx \pi$ depending on π only.

4. Semantics Defined by Copy Rules

An *ALGOL-like programming language* is defined to be a pair $\Pi=(L, \Sigma)$ where the *syntax* L is a decidable subset of L_{Algol} and where the *semantics* Σ is a mapping which - given an interpretation $\mathscr{I}$ - assigns a program function $\Sigma^{\mathscr{I}}(\pi)$ to every program $\pi \in L$. We require that L is closed w.r.t. congruence and that $\Sigma^{\mathscr{I}}$ is invariant for congruent programs. To define the semantics we proceed in two steps - reflecting the fact that all basic tools for this definition are already developed and only procedures need an extra treatment.

Step 1. Let STM_0 be the class of all statements *without procedures.* The semantics $\Sigma_0^{\mathscr{I}}$ of STM_0 is a mapping $\Sigma_0^{\mathscr{I}}: STM_0 \rightarrow \{\text{program functions}\}$ defined in an obvious way by using the special program functions introduced in Sect. 2, line (*). For example

$$\Sigma_0^{\mathscr{I}}(\textbf{begin var } x;\, S \textbf{ end}) = \text{block}_{\mathscr{I}}(x, \Sigma_0^{\mathscr{I}}(S)).$$

Note that $\Sigma_0^{\mathscr{I}}(S)$ is a program function on free(S).

Lemma 2. *For all $S, S_1, S_2 \in STM_0$ the following holds:*

a) *If $S_1 \approx S_2$ then $\Sigma_0^{\mathscr{I}}(S_1) = \Sigma_0^{\mathscr{I}}(S_2)$.*

b) *If σ is a substitution which is injective on* idf(S) *then $\Sigma_0^{\mathscr{I}}(\sigma(S)) = (\Sigma_0^{\mathscr{I}}(S))\sigma$, i.e. syntactic and semantic application of σ coincide.*

Proof. By induction on the structure of S_1 resp. S. $\square$

Step 2. We use so-called copy rules to *extend* the semantics $\Sigma_0^{\mathscr{I}}$ to the whole class of programs. These rules explain the semantics of a procedure call by the semantics of an associated modified procedure body.

Definition 1. A *copy rule* $\mathscr{C}$ is a certain relation between pairs (B, I), consisting of a block B and a finite set I of identifiers, and blocks B': $(B, I)\mathscr{C}B'$. In this paper the following three copy rules are studied in detail:

The *ALGOL 60 copy rule* $\mathscr{C}_{60}$:

$(B, I)\mathscr{C}_{60}B'$ iff $B \underset{\text{inj}}{\approx} B'$ and no identifier bound in B' occurs in I.

The "*most recent*" *copy rule* $\mathscr{C}_{mr}$:

$(B, I)\mathscr{C}_{mr}B'$ iff $B \underset{\text{inj}}{\approx} B'$, no variable bound in B' occurs in I and procedure identifiers have not been renamed.

The *naive copy rule* $\mathscr{C}_n$:

$(B, I)\mathscr{C}_n B'$ iff $B = B'$.

The general definition of $\mathscr{C}$ runs as follows:

$(B, I)\mathscr{C}B'$ iff $B \underset{\text{inj}}{\approx} B'$, no identifier bound at a position in $\text{pos}_{\mathscr{C}}(B')$ occurs in I and identifiers bound at positions different from those in $\text{pos}_{\mathscr{C}}(B')$ have not been renamed.

(Here $\text{pos}_{\mathscr{C}}$ is a mapping which assigns to every block B a set of positions in B such that $\text{pos}_{\mathscr{C}}(B_1) = \text{pos}_{\mathscr{C}}(B_2)$ holds whenever B_1 and B_2 have the same structure. For example $\text{pos}_{\mathscr{C}_n}(B) = \emptyset$.)

Intuitively a copy rule describes a certain (non-deterministic) renaming mechanism for identifiers which will be used to handle scope problems. If we wish to restrict this mechanism to a deterministic one, we consider so-called specimens: $\underline{\mathscr{C}}$ is called a (deterministic) *specimen* of $\mathscr{C}$ if $\underline{\mathscr{C}}$ is a mapping with $\underline{\mathscr{C}} \subseteq \mathscr{C}$ which is defined for all (B, I). Hence B' with $(B, I)\underline{\mathscr{C}}B'$ is uniquely determined and we write $B' = \underline{\mathscr{C}}(B, I)$.[3]

Next we define how to apply a copy rule to a program.

Definition 2. Let $\mathscr{C}$ be a copy rule and $\underline{\mathscr{C}}$ be a specimen of $\mathscr{C}$. The *application of* $\underline{\mathscr{C}}$ is a relation $\vdash_{\underline{\mathscr{C}}}$ between programs: $\pi \vdash_{\underline{\mathscr{C}}} \pi'$ iff the following holds:

(1) A correct procedure call $p(\bar{y}:\bar{r})$ occurs in the main part of π, say at position i. *Correct* means that the procedure declaration belonging to the occurrence (i, p) is of the form **proc** $p(\bar{x}:\bar{q})$; B; where $|\bar{x}| = |\bar{y}|$ and $|\bar{q}| = |\bar{r}|$.

(2) π' results from π by replacing $p(\bar{y}:\bar{r})$ at position i by the modified procedure body $B' = \underline{\mathscr{C}}(B\sigma, I)$ where $\sigma = [\bar{y}, \bar{r}/\bar{x}, \bar{q}]$ and $I = \mathrm{proj}_2(\mathrm{visible}(i, p(\bar{y}:\bar{r})))$.[4]

As abbreviation we define $\mathrm{stm}(\pi, \pi') = (i, p(\bar{y}:\bar{r}))$ and $\mathrm{blk}(\pi, \pi') = B'$. The *application of* $\mathscr{C}$ is a relation $\vdash_{\mathscr{C}}$ between programs which is defined as the union of all relations $\vdash_{\underline{\mathscr{C}}}$ such that $\underline{\mathscr{C}}$ is a specimen of $\mathscr{C}$, i.e. for B' simply $(B\sigma, I)\mathscr{C}B'$ is required.

This definition generalizes the relation $\vdash$ introduced in Langmaack [26] to arbitrary copy rules $\mathscr{C}$. We give an example for the application of the copy rule $\mathscr{C}_{60}$: $\pi \vdash_{\mathscr{C}_{60}} \pi'$.

```
π  = begin
        var y;
        proc p(z:q); begin var u; u:=z; q(z:) end;
        proc r(z:); begin x:=z end;
                          p(x:r)
                         /      \
      end               /        \
                       /          \
π' = begin...begin var y; y:=x; r(x:) end...end
```
[5]

In Definition 2 π' is obtained from π by replacing *just one* correct procedure call in the main part of π by the associated modified procedure body. We define now a mapping $\mathit{Copy}_{\underline{\mathscr{C}}}: L_{\mathrm{Algol}} \to L_{\mathrm{Algol}}$ as follows: $\pi' = \mathit{Copy}_{\underline{\mathscr{C}}}(\pi)$ if π' results from π by replacing simultaneously *all* correct procedure calls in the main part of π by their associated modified procedure bodies - modified according to $\underline{\mathscr{C}}$. (By convention $\pi' = \pi$ if there is no correct procedure call in the main part of π.) Notice that $\pi \vdash^{*}_{\underline{\mathscr{C}}} \pi'$ holds.[6] Let $\mathit{Copy}_{\underline{\mathscr{C}}}$ be the j-th iteration of $\mathit{Copy}_{\underline{\mathscr{C}}}(j \geq 0)$. Intuitively $\mathit{Copy}_{\underline{\mathscr{C}}, j}$ describes the process of simultaneously copying up to depth $j \geq 0$.

Next we discuss the following question: What is the relation between $\mathit{Copy}_{\underline{\mathscr{C}}_1, j}(\pi)$ and $\mathit{Copy}_{\underline{\mathscr{C}}_2, j}(\pi)$ where $\underline{\mathscr{C}}_1$ and $\underline{\mathscr{C}}_2$ are two specimens of the same copy rule $\mathscr{C}$? To this end we introduce

[3] When renaming an identifier α in B a copy rule $\mathscr{C}$ just requires that the new identifier α' in B' is not in I whereas a specimen $\underline{\mathscr{C}}$ of $\mathscr{C}$ could require more strictly that α' is the *first* identifier not in I

[4] proj_2 is the projection onto the second component of a pair. Here proj_2 is applied to a set of identifier ocurrences and yields a set of identifiers

[5] Notice that variable y is *not* visible from $p(x:r)$ in π. Hence we were allowed to rename the bound variable u into y in π'

[6] As usual R^* denotes the reflexive, transitive closure of a binary relation R

Definition 3. π_1 is *strongly congruent* to π_2 ($\pi_1 \underset{\text{str}}{\approx} \pi_2$ for short) if $\pi_1 \approx \pi_2$ and the following holds: Given any identifier occurrences (i, α_1), (j, β_1), (k, γ_1) in π_1 with (i, α_1), $(j, \beta_1) \in \text{visible } \pi_1(k, \gamma_1)$ and – at the same positions – identifier occurrences (i, α_2), (j, β_2), (k, γ_2) in π_2 with (i, α_2), $(j, \beta_2) \in \text{visible } \pi_2(k, \gamma_2)$ then $\alpha_1 = \beta_1$ holds iff $\alpha_2 = \beta_2$ holds.

Note that congruent distinguished programs are strongly congruent, but not every program has a strongly congruent distinguished one. By the following lemma the relation $\underset{\text{str}}{\approx}$ is invariant w.r.t. applications of the copy rule. Such an invariance does not hold directly for $\approx$.

Lemma 3. *Let $\mathscr{C}$ be a copy rule and $\pi_1 \underset{\text{str}}{\approx} \pi_2$, $\pi_1 \vdash_{\overline{\mathscr{C}}} \pi_1'$ and $\pi_2 \vdash_{\overline{\mathscr{C}}} \pi_2'$ (copied at the same position) hold. Then also $\pi_1' \underset{\text{str}}{\approx} \pi_2'$ holds.*

Proof. A detailed proof is given in [37]. □

Corollary 1. *Let $\underline{\mathscr{C}}_1$ and $\underline{\mathscr{C}}_2$ be specimens of the same copy rule $\mathscr{C}$. Then for all $j \geq 0$ and $\pi \in L_{\text{Algol}}$*

$$\mathscr{C}\!\mathit{opy}_{\underline{\mathscr{C}}_1, j}(\pi) \underset{\text{str}}{\approx} \mathscr{C}\!\mathit{opy}_{\underline{\mathscr{C}}_2, j}(\pi).$$

We are now prepared to define the semantics of units and programs.

Definition 4. Let $\mathscr{C}$ be a copy rule and $j \geq 0$. Then the *approximating semantics* $\Sigma^{\mathscr{I}}_{\mathscr{C}, j}$ is defined by

$$\Sigma^{\mathscr{I}}_{\mathscr{C}, j}(E|S) = \Sigma^{\mathscr{I}}_0(\mathscr{D}\!\mathit{el}(\mathscr{C}\!\mathit{opy}_{\underline{\mathscr{C}}, j}(\textbf{begin } ES \textbf{ end})))$$

where $\underline{\mathscr{C}}$ is an *arbitrary* specimen of $\mathscr{C}$ and $\mathscr{D}\!\mathit{el}$ is a mapping $\mathscr{D}\!\mathit{el}\colon STM \to STM_0$ such that $\mathscr{D}\!\mathit{el}(S')$ results from S' by deleting every procedure in S' and replacing every remaining procedure call by **error.**

The *full semantics* is given by

$$\Sigma^{\mathscr{I}}_{\mathscr{C}}(E|S) = \bigcup_{j \geq 0} \Sigma^{\mathscr{I}}_{\mathscr{C}, j}(E|S).$$

For programs π we define

$$\Sigma^{\mathscr{I}}_{\mathscr{C}, j}(\pi) = \Sigma^{\mathscr{I}}_{\mathscr{C}, j}(\emptyset|\pi_d) \quad \text{and} \quad \Sigma^{\mathscr{I}}_{\mathscr{C}}(\pi) = \Sigma^{\mathscr{I}}_{\mathscr{C}}(\emptyset|\pi_d)$$

where $\pi_d \approx \pi$ is distinguished.

By Corollary 1 and Lemma 2, a) the definition of $\Sigma^{\mathscr{I}}_{\mathscr{C}, j}(E|S)$ makes sense. The possibility to choose a suitable specimen $\underline{\mathscr{C}}$ of $\mathscr{C}$ is convenient in proofs, for example of Lemma 4 in the next section. Note that $\Sigma^{\mathscr{I}}_{\mathscr{C}, j}(E|S)$ and $\Sigma^{\mathscr{I}}_{\mathscr{C}}(E|S)$ are program functions on free$(E|S)$. Further on, the definition of $\Sigma^{\mathscr{I}}_{\mathscr{C}}(\pi)$ shows that all renamings of identifiers necessary to handle scope problems are done here firstly by a preprocessing step $\pi \to \emptyset|\pi_d$ and secondly whenever the copy rule is applied.

In this paper we study programming languages

$$\Pi = (L, \Sigma_{\mathscr{C}}).$$

$\Sigma_{\mathscr{C}_{60}}$ is the usual *static scope* semantics for ALGOL-like programs as defined in the ALGOL 60 report [4]. Notice that we use here the parameter mechanism "call by name". Of course, with help of copy rules there is no difficulty to model also other parameter mechanism such as "call by value" (in connection with arbitrary expressions e instead of simple variables x as actual procedure parameters) or "call by reference" (in the presence of array variables).

$\Sigma_{\mathscr{C}_n}$ and $\Sigma_{\mathscr{C}_{mr}}$ are simplified semantics. $\Sigma_{\mathscr{C}_n}$ is closely related to what is usually called *dynamic scope* semantics Σ_{dyn}. The only difference is that the preprocessing step $\pi \to \emptyset | \pi_d$ is omitted, i.e. Σ_{dyn} is defined simply by $\Sigma^{\mathscr{I}}_{\mathrm{dyn}}(\pi) = \Sigma^{\mathscr{I}}_{\mathscr{C}_n}(\emptyset | \pi)$. (Hence Σ_{dyn} is not a proper semantics in our sense because it is not invariant w.r.t. congruence of programs.) $\Sigma_{\mathscr{C}_{mr}}$ will be important when dealing with programs π satisfying the so-called "most recent" property because for these programs $\Sigma^{\mathscr{I}}_{\mathscr{C}_{mr}}(\pi) = \Sigma^{\mathscr{I}}_{\mathscr{C}_{60}}(\pi)$ holds (see Sect. 10).

Finally, let us study the approximating semantics $\Sigma^{\mathscr{I}}_{\mathscr{C},j}$.

Corollary 2. *Let $\mathscr{C}$ be a copy rule and $j \geqq 0$. Then $\Sigma^{\mathscr{I}}_{\mathscr{C},j}$ satisfies the following equations:*

$$\Sigma^{\mathscr{I}}_{\mathscr{C},j}(E|\mathbf{error}) = \Omega,$$

$$\Sigma^{\mathscr{I}}_{\mathscr{C},j}(E|x := e) = \mathrm{ass}_{\mathscr{I}}(x, e),$$

$$\Sigma^{\mathscr{I}}_{\mathscr{C},j}(E|S_1; S_2) = \Sigma^{\mathscr{I}}_{\mathscr{C},j}(E|S_1) \circ \Sigma^{\mathscr{I}}_{\mathscr{C},j}(E|S_2),$$

$$\Sigma^{\mathscr{I}}_{\mathscr{C},j}(E|\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}) = if_{\mathscr{I}}(b, \Sigma^{\mathscr{I}}_{\mathscr{C},j}(E|S_1), \Sigma^{\mathscr{I}}_{\mathscr{C},j}(E|S_2)),$$

$$\Sigma^{\mathscr{I}}_{\mathscr{C},j}(E|\mathbf{begin\ var}\ x;\ E_0 S_1\ \mathbf{end}) = \mathrm{block}_{\mathscr{I}}(x, \Sigma^{\mathscr{I}}_{\mathscr{C},j}(E_1|S_1))$$

where $E_1 = \mathrm{add}(E, E_0)$.

$$\Sigma^{\mathscr{I}}_{\mathscr{C},j}(E|p(\bar{y}:\bar{r})) = \begin{cases} \Sigma^{\mathscr{I}}_{\mathscr{C},j-1}(E|B_{\mathscr{C}}) & \text{if } j \geqq 1, \mathbf{proc}\ p(\bar{x}:\bar{q}); B; \in E \\ & \quad \text{and} \quad |\bar{x}| = |\bar{y}|, |\bar{q}| = |\bar{r}| \\ \Omega & \text{otherwise} \end{cases}$$

where $(B\sigma, I)\mathscr{C}B_{\mathscr{C}}$ with $\sigma = [\bar{y}, \bar{r}/\bar{x}, \bar{q}]$ and $I = \mathrm{idf}(E|p(\bar{y}:\bar{r}))$.

Proof. Study the mappings $\mathscr{C}\!\mathit{opy}_{\mathscr{C},j}$ and apply Definition 4. □

Corollary 2 gives the connection to Lemma 1. The equations above show that the copy rule approach to semantics leads to a partly operational style of semantics. Other papers using operational semantics *define* their semantics by equations of this kind (e.g. Lauer [32], Clarke [9]). In our context this would be problematic because $B_{\mathscr{C}}$ is not uniquely determined.

5. Substitutional Equivalence

In this section we study the computational behaviour of units which differ only by a certain renaming of identifiers. These investigations are essential for our completeness results and needed for a satisfactory treatment of sharing situations (see Sect. 6).

Given a unit $E|S$ we take $\min(E|S)$ to denote that unit $E_0|S$ where E_0 is the least element w.r.t. $\subseteq$ within the set $\{E' | \text{where } E' \subseteq E \text{ and } \mathbf{begin}\ E'S\ \mathbf{end} \in L_{\mathrm{Algol}}\}$. Intuitively speaking, E_0 contains only those procedures which are needed to understand S. With this notation $\sigma(\min(E|S))=\sigma(E_0|S)=\sigma(E_0)|\sigma(S)$ holds for substitutions σ (cf. Sect. 2).

Definition 5. Two units $E|S$ and $E'|S'$ are called *substitutional equivalent* if there is a substitution σ with $\sigma(\min(E|S))=\min(E'|S')$ which is injective on $\mathrm{idf}(\min(E|S))$. (As a shorthand we write $E|S_\sigma \cong E'|S'$ or simply $E|S \cong E'|S'$ if we do not bother about the particular substitution σ.)

Notice that both variables and procedure identifiers may be substituted by σ. As an example we consider the following units $E|S$ and $E'|S'$ which are substitutional equivalent w.r.t. $\sigma=[u, v, w, q/x, y, z, p]$:

$$
{}_{\sigma}{\cong}\left(\begin{array}{l} \nearrow E|S = \begin{array}{l}\textbf{proc } p(x:);\ \textbf{begin } x:=y+1 \textbf{ end};\\ \textbf{proc } r;\ \textbf{begin } u:=\mathrm{v} \textbf{ end};\end{array} \left|\ p(z:)\right. \\ \searrow E'|S' = \textbf{proc } q(u:);\ \textbf{begin } u:=v+1 \textbf{ end};|\ q(w:)\end{array}\right.
$$

Note that the procedure **proc** $r; \ldots;$ does not occur in $\min(E|S)$.

The next lemma states units which differ only syntactically by a substitution σ (in the sense of substitutional equivalence) differ also semantically only by this σ.

Lemma 4. (Substitution Lemma). *Let $E|S_\sigma \cong E'|S'$. Then for every copy rule $\mathscr{C}$ and $j \geqq 0$*

$$(*) \qquad (\Sigma^{\mathscr{J}}_{\mathscr{C},j}(E'|S')=(\Sigma^{\mathscr{J}}_{\mathscr{C},j}(E|S))\,\sigma.$$

For the ALGOL 60 copy rule $\mathscr{C}_{60}$ the assertion of this lemma looks rather natural, but here we deal with arbitrary copy rules $\mathscr{C}$. To prove the semantical equation $(*)$ we study the computational behaviour of $E|S$ and $E'|S'$ by means of *formal computation paths*

$$\not{p}\colon\ E|S=E_1|S_1 \xrightarrow[\mathscr{C}]{} \ldots \xrightarrow[\mathscr{C}]{} E_n|S_n,$$

i.e. sequences of units generated by a relation $\xrightarrow[\mathscr{C}]{}$ between units which is closely related to the relation $\vdash_{\mathscr{C}}$ between programs. *Formal* means that we abstract from the actual operations on data. First we define $\xrightarrow[\underline{\mathscr{C}}]{}$ for a specimen $\underline{\mathscr{C}}$ of $\mathscr{C}$ as follows:

(1) $S=S_1;S_2$ or $S=$ **if** b **then** S_1 **else** S_2 **fi**:

$$E|S \xrightarrow[\underline{\mathscr{C}}]{} E|S_1 \quad \text{and} \quad E|S \xrightarrow[\underline{\mathscr{C}}]{} E|S_2$$

(2) $S=$ **begin var** $x;\ E_0 S_1$ **end**:

$$E|S \xrightarrow[\underline{\mathscr{C}}]{} E_1|S_1 \qquad \text{where } E_1=\mathrm{add}(E, E_0)$$

(3) $S=p(\bar{y}:\bar{r})$ with **proc** $p(\bar{x}:\bar{q});\ B; \in E$ and $|\bar{x}|=|\bar{y}|$, $|\bar{q}|=|\bar{r}|$:

$$E|S \xrightarrow[\underline{\mathscr{C}}]{} E|B_{\mathscr{C}} \quad \text{where } B_{\underline{\mathscr{C}}}=\underline{\mathscr{C}}(B\sigma, I) \quad \text{with} \quad \sigma=[\bar{y},\bar{r}/\bar{x},\bar{q}] \quad \text{and} \quad I=\mathrm{idf}(E|S).$$

Now $\xrightarrow[\mathscr{C}]{}$ is the union of all relations $\xrightarrow[\underline{\mathscr{C}}]{}$ where $\underline{\mathscr{C}}$ is a specimen of $\mathscr{C}$, i.e. in case (3) of the definition above $E|S \xrightarrow[\mathscr{C}]{} E|B_{\mathscr{C}}$ with $(B\sigma, I)\mathscr{C}B_{\mathscr{C}}$ holds. A path

$$\not{p}\colon\ E_1|S_1 \xrightarrow[\mathscr{C}]{} \ldots \xrightarrow[\mathscr{C}]{} E_n|S_n$$

is said to correspond to a path $\mathscr{p}'$: $E_1'|S_1' \underset{\mathscr{C}}{\longrightarrow} \dots \underset{\mathscr{C}}{\longrightarrow} E_n'|S_n'$ if for every $k \in \{1, \dots, n\}$ S_k and S_k' have the same structure and in case (1) of the definition above S_{k+1} and S_{k+1}' are chosen to be both the first or both the second substatement of S_k and S_k' respectively.

We are now prepared for the

Proof of Lemma 4. Let $\sigma = [\bar{\beta}/\bar{\alpha}]$ and $\mathscr{C}$ be a copy rule. Then there is a specimen $\underline{\mathscr{C}}$ of $\mathscr{C}$ with the following properties: For every block B and every finite set I of identifiers

(i) $\underline{\mathscr{C}}(B, I) = \underline{\mathscr{C}}(B, I \cup I')$ where $I' = \mathrm{idf}(E) \cup \mathrm{idf}(E') \cup \{\bar{\alpha}\} \cup \{\bar{\beta}\}$.

(ii) $\underline{\mathscr{C}}(\sigma(B), I) = \sigma(\underline{\mathscr{C}}(B, I))$ if σ is injective on $\mathrm{idf}(B)$.

We show that

(1) σ is injective on the set X of all identifiers β which occur in some unit $E^*|S^*$ with $E|S \underset{\underline{\mathscr{C}}}{\overset{*}{\longrightarrow}} E^*|S^*$.

(2) For every path $\mathscr{p}_i$: $E|S = E_1|S_1 \underset{\underline{\mathscr{C}}}{\longrightarrow} \dots \underset{\underline{\mathscr{C}}}{\longrightarrow} E_i|S_i$ there is a corresponding path $\mathscr{p}_i'$: $E'|S' = E_1'|S_1' \underset{\underline{\mathscr{C}}}{\longrightarrow} \dots \underset{\underline{\mathscr{C}}}{\longrightarrow} E_i'|S_i'$ with

(a_i) $E_i|S_{i\sigma} \cong E_i'|S_i'$ and

(b_i) $(\mathrm{idf}(E_i) \backslash \mathrm{idf}(E_i')) \cup (\mathrm{idf}(E_i') \backslash \mathrm{idf}(E_i)) \subseteq I'$.

(2') Vice versa: For every path $\mathscr{p}_i'$: ... there exists a corresponding path $\mathscr{p}_i$: ... with (a_i) and (b_i).

Assertion (1) is a consequence of (i). To prove (2) we proceed by induction on i. Since the assertion is trivial for $i = 1$, let us consider the induction step "$i \to i+1$", i.e. a path $\mathscr{p}_{i+1}$: $\mathscr{p}_i \underset{\underline{\mathscr{C}}}{\longrightarrow} E_{i+1}|S_{i+1}$: We have to specify an appropriate unit $E_{i+1}'|S_{i+1}'$ which extends $\mathscr{p}_i'$ to $\mathscr{p}_{i+1}'$. The only interesting case is that S_i is a procedure call $p(\bar{y}:\bar{r})$:

Let $\mathbf{proc}\, p(\bar{x}:\bar{q}); B; \in E_i$ and $B' = B[\bar{y}, \bar{r}/\bar{x}, \bar{q}]$. Thus $E_{i+1} = E_i$ and $S_{i+1} = \underline{\mathscr{C}}(B', \mathrm{idf}(E_i|S_i))$. By (a_i) $S_i' = \sigma(p(\bar{y}:\bar{r}))$ and $\sigma(\mathbf{proc}\, p(\bar{x}:\bar{q}); B;) \in E_i'$. Therefore we choose $E_{i+1}' = E_i'$ and $S_{i+1}' = \underline{\mathscr{C}}(B'', \mathrm{idf}(E_i'|S_i'))$ where $B'' = \sigma(B)\,[\sigma(\bar{y}, \bar{r})/\sigma(\bar{x}, \bar{q})]$. Note that $B'' = \sigma(B')$ holds by (1). Since property (b_{i+1}) holds trivially, (a_{i+1}) remains to be shown: $S_{i+1}' = \underline{\mathscr{C}}(B'', \mathrm{idf}(E_i'|S_i')) = \underline{\mathscr{C}}(B'', \mathrm{idf}(E_i'|S_i') \cup I')$ [by (i)] $= \underline{\mathscr{C}}(B'', \mathrm{idf}(E_i|S_i) \cup I')$ [by (a_i) and (b_i)] $= \underline{\mathscr{C}}(B'', \mathrm{idf}(E_i|S_i))$ [by (i)] $= \sigma(\underline{\mathscr{C}}(B', \mathrm{idf}(E_i|S_i))$ [by (ii)] $= \sigma(S_{i+1})$ which implies $E_{i+1}|S_{i+1\,\sigma} \cong E_{i+1}'|S_{i+1}'$ [by (a_i)].

Thus (2) holds. An analogous argument proves (2'). Using (2) and (2') we can conclude that for every $j \geqq 0$

$$\sigma(\mathscr{Copy}_{\underline{\mathscr{C}},j}(E|S)) = \mathscr{Copy}_{\underline{\mathscr{C}},j}(E'|S')$$

holds. Now the semantical equation $(*)$ of the lemma follows by employing (1), the definition of $\Sigma^{\mathscr{I}}_{\mathscr{C},j}$ and Lemma 2, b). □

Remark 1. By Lemma 4, $\Sigma^{\mathscr{I}}_{\mathscr{C},j}(E|S)$ is a program function on $\mathrm{free}(\min(E|S)) \subseteq \mathrm{free}(E|S)$.

6. How to Deal with Sharing?

Starting from Hoare [19] the so-called *sharing* (or *aliasing*) *of variables* has been disallowed in most subsequent papers (e.g. [10, 13, 9, 32, 12, 34]). In case of the copy rule $\mathscr{C} = \mathscr{C}_{60}$ this restriction can be formulated as follows:

A distinguished program π is *sharing free* if for every procedure call $p(\bar{x}:\bar{q})$ with $\emptyset|\pi \xrightarrow[\mathscr{C}]{*} E|p(\bar{x}:\bar{q})$ the actual variables in $\bar{x}$ are all distinct and different from global (i.e. free) variables in E.

The motivation for disallowing sharing was of technical nature, namely to obtain the following simple substitution rule

$$(\mathrm{Sub}_0)\quad \begin{array}{ll} \text{If} & \Sigma_{\mathscr{C}}^{\mathscr{I}}(E|p(\bar{x}:\bar{q}))(P)\subseteq Q \quad \text{and} \quad \{\bar{y}\}\cap \mathrm{free}(P\vee Q)=\emptyset \\ \text{then} & \Sigma_{\mathscr{C}}^{\mathscr{I}}(E|p(\bar{y}:\bar{q}))(P[\bar{y}/\bar{x}])\subseteq Q[\bar{y}/\bar{x}]. \end{array}$$

By Lemma 1 and 4 this rule is sound if both units are "generated" by a sharing free distinguished program π, i.e. if $\emptyset|\pi \xrightarrow[\mathscr{C}]{*} \mathrm{E}|p(\bar{x}:\bar{q})$ and $\emptyset|\pi \xrightarrow[\mathscr{C}]{*} E|p(\bar{y}:\bar{q})$ holds. Of course, (Sub_0) is not sound in general as simple examples show (see e.g. Hoare [19]).

But looking at real programs, the sharing restriction is very unnatural. Why should a procedure call $p(x, y:\bar{q})$ be more reasonable than $p(u, u:\bar{q})$?(See also [5] and [40].) Moreover, we easily run into conceptual difficulties because it is undecidable whether an ALGOL-like program is sharing free. This follows from a result by Langmaack [26] stating that the *formal reachability* of procedures is undecidable for L_{Algol} when the copy rule $\mathscr{C}_{60}$ is applied. Thus admitting only sharing free programs leads in general to an undecidable subset L of L_{Algol} which contradicts the concept of syntax (see Sect. 4). If on the other hand we allow arbitrary programs, but restrict the application of (Sub_0) to sharing free programs, then this rule in turn becomes undecidable which contradicts the notion of proof rule (see Sect. 7).

Summarizing, we find the only satisfactory solution is here to formulate a rule which can deal with sharing. Fortunately the concept of substitutional equivalence together with the Substitution Lemma leads very easily to such a refined substitution rule, namely

$$(\mathrm{Sub}_1)\quad \begin{array}{ll} \text{If} & \Sigma_{\mathscr{C}}^{\mathscr{I}}(E|p(\bar{x}:\bar{q}))(P)\subseteq Q, \\ & E|p(\bar{x}:\bar{q})\cong E|p(\bar{y}:\bar{q}) \quad \text{and} \quad \{\bar{y}\}\cap \mathrm{free}(P\vee Q)=\emptyset \\ \text{then} & \Sigma_{\mathscr{C}}^{\mathscr{I}}(E|p(\bar{y}:\bar{q}))(P[\bar{y}/\bar{x}])\subseteq Q[\bar{y}/\bar{x}]. \end{array}$$

which is sound without implicit assumptions.[7]

Intuitively speaking, this rule divides the set $\mathrm{call}(E, p, \bar{q})$ of all correct calls of the procedure p in E with fixed actual procedure identifiers $\bar{q}$ into *sharing classes*, i.e. equivalence classes w.r.t. the following relation $\sim$:

$$p(\bar{x}:\bar{q})\sim p(\bar{y}:\bar{q}) \qquad \text{iff } E|p(\bar{x}:\bar{q})\cong E|p(\bar{y}:\bar{q}).$$

For example $p(x, y:\bar{q})\sim p(a, b:\bar{q})$, but $p(x, y:\bar{q})\not\sim p(u, u:\bar{q})$. The rule (Sub_1) guarantees that a correctness result proved for a procedure call $p(\bar{x}:\bar{q})$ can be transferred only to those calls $p(\bar{y}:\bar{q})$ which are in the same sharing class.

Observe that there are only *finitely many* sharing classes in $\mathrm{call}(E, p, \bar{q})$. This fact guarantees completeness results even in the presence of sharing (cf. Sect. 9).

[7] Actually we use a more general substitution rule in $\mathscr{H}(\mathscr{C})$, but to discuss the issue of sharing (Sub_1) is sufficient

Note that in this terminology the restriction to sharing free programs can be understood as allowing *only one* sharing class. In this paper we confine ourselves to identifiers called *by name* as actual parameters of procedure calls, but the basic observation that only finitely many sharing classes arise remains valid when expressions called *by value* or array variables called *by reference* are allowed as actual parameters.

Related approaches to deal with sharing can be found in [3, 5, 8, 15] and [40, 41]. Proofs of soundness and completeness are outlined in [8] only.

7. The Hoare-like Calculi $\mathscr{H}(\mathscr{C})$

Hoare-like systems dealing with recursive procedures have been presented in two different ways: Either in the style of Gentzen's systems of natural deduction (e.g. in [9, 12, 20]) or in the style of Gentzen's calculi of sequents (e.g. in [1, 6, 17]) (see [38] for the logical notions.) In this paper we prefer calculi rather than deduction systems because the notion of formal proof is simpler for calculi. This leads – together with our particular interpretation of proof lines – to a fairly simply soundness proof in Sect. 8 and further on simplifies the proofs in Sect. 11.

On the other hand, deduction systems are more convenient for practical applications because they allow structured proofs with assumptions. This is, however, no objection against our calculi since both approaches of presenting a proof system are equivalent and each system of natural deduction can easily be transformed into a calculus of sequents. (For the case of Hoare-like systems such a transformation is described in [2].)

Before defining a Hoare-like calculus $\mathscr{H}(\mathscr{C})$ for each copy rule $\mathscr{C}$ we have to fix some notation. An *atomic Hoare formula* is a formula P or a construct of the form $\{P\}E|S\{Q\}$ or $\{P\}\pi\{Q\}$. By a *Hoare formula* we mean a finite set of atomic Hoare formulas. We take h as a typical atomic Hoare formula and H as a typical Hoare formula. A *proof line* ℓ is a construct of the form $H\rightarrow H'$; proof lines $H\rightarrow\{h\}$ and $\emptyset\rightarrow\{h\}$ are abbreviated by $H\rightarrow h$ and h respectively. A Hoare-like calculus is given by so-called *proof rules* which are decidable $(n+1)$-place relations ρ over the set of proof lines ($n\in\mathbb{N}_0$). Instead of the set-theoretic notation $\rho=\{(\ell_1,\ldots,\ell_n,\ell_{n+1})\,|\,\text{where}\ldots\}$ we take the usual notation for proof rules

$$\frac{\ell_1,\ldots,\ell_n}{\ell_{n+1}}\quad \text{where}\ \ldots$$

In the case "$n=0$" ρ is called an *axiom schema* or simply *axiom* and the notation shortens in an obvious way. Proof rules are *applied* to finite sequences ξ of proof lines in the following way: If a proof rule ρ fits $\ell_1,\ldots,\ell_n,\ell_{n+1}$ and $\ell_1,\ldots,\ell_n$ occur in ξ, then ℓ_{n+1} may be added at the end of ξ.

Definition 6. Let $\mathscr{C}$ be a copy rule. Then the Hoare-like calculus $\mathscr{H}(\mathscr{C})$ is defined as follows:

(1) Axiom of Assumption

$$H\rightarrow H$$

(2) Axiom of the Error Statement

$$\{P\}E|\mathbf{error}\,\{Q\}$$

(3) Axiom of Assignment Statements

$$\{P[e/x]\}E|x:=e\{P\}$$

(4) Axiom of Incorrect Procedure Calls

$$\{P\}E|p(\bar{x}:\bar{q})\{Q\}$$

where the procedure call $p(\bar{x}:\bar{q})$ is incorrect w.r.t. E.

(5) Rule of Composition

$$\frac{H\rightarrow\{\{P\}E|S_1\{R\},\ \{R\}E|S_2\{Q\}\}}{H\rightarrow\{P\}E|S_1;\ S_2\{Q\}}$$

(6) Rule of Conditional Statements

$$\frac{H\rightarrow\{\{P\wedge b\{E|S_1\{Q\},\ \{P\wedge\neg b\}E|S_2\{Q\}\}}{H\rightarrow\{P\}E|\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\{Q\}}$$

(7) Rule of Blocks[8]

$$\frac{H\rightarrow\{P[y/x]\wedge x=\omega\}E_1|S_1\{Q[y/x]\}}{H\rightarrow\{P\}E|\mathbf{begin\ var}\ x;\ E_0S_1\ \mathbf{end}\{Q\}}$$

where $y\notin\text{free}(P\vee Q)\cup\text{free}(\min(E_1|S_1))$ and $E_1=\text{add}(E, E_0)$.

(8) Rule of Recursive Procedure Calls

$$\frac{H\cup\bigcup_{i=1}^{m}\{\{P_i\}E_i|p_i(\bar{y}_i:\bar{r}_i)\{Q_i\}\}\rightarrow\bigcup_{i=1}^{m}\{\{P_i\}E_i|B_{i\mathscr{C}}\{Q_i\}\}}{H\rightarrow\bigcup_{i=1}^{m}\{\{P_i\}E_i|p_i(\bar{y}_i:\bar{r}_i)\{Q_i\}\}}$$

where $E_i|p_i(\bar{y}_i:\bar{r}_i)\xrightarrow[\mathscr{C}]{}E_i|B_{i\mathscr{C}}$ holds for $i=1,\ldots,m$.

(9) Rule of Substitution

$$\frac{H\rightarrow\{P\}E|S\{Q\}}{H\rightarrow\{P\sigma\}E'|S'\{Q\sigma\}}$$

where σ is a substitution which is injective on $\text{free}(P\vee Q)\cup\text{idf}(\min(E|S))$ and $E|S_\sigma\cong E'|S'$ holds.

[8] The case that the variable declaration is missing is trivially subsumed under this rule

(10) Rule of Variable Substitution

$$\frac{H \to \{P\}E|S\{Q\}}{H \to \{P\sigma\}E|S\{Q\sigma\}}$$

where $\sigma = [x_1, \ldots, x_n/y_1, \ldots, y_n]$ is a substitution satisfying for all $k = 1, \ldots, n$ i) $y_k \notin \text{free}(\min(E|S))$ and ii) $x_k \notin \text{free}(\min(E|S))$ if $y_k \in \text{free}(Q)$.

(11) Rule of Invariance

$$\frac{H \to \{P\}E|S\{Q\}}{H \to \{P \wedge R\}E|S\{Q \wedge R\}}$$

where $\text{free}(R) \cap \text{free}(\min(E|S)) = \emptyset$.

(12) Rule of Connection

$$\frac{H_1 \to H_2 \cup H_3,\ H_1' \to H_2' \cup H_3'}{H_1 \cup H_1' \to H_2 \cup H_2'}$$

(13) Rule of Consequence

$$\frac{P_1 \to P,\ H \to \{P\}E|S\{Q\},\ Q \to Q_1}{H \to \{P_1\}E|S\{Q_1\}}$$

(14) Rule of Programs

$$\frac{H \to \{P\}\emptyset|\pi_d\{Q\}}{H \to \{P\}\pi\{Q\}}$$

where $\pi_d \approx \pi$ is distinguished.

In order to apply rule (13) in a meaningful way, $\mathscr{H}(\mathscr{C})$ must be augmented with an *oracle* $\mathcal{O} \subseteq LF$ for logical formulas, e.g. $\mathcal{O} = Th(\mathscr{I})$. Formally $\mathcal{O}$ is treated as an additional axiom:

(15) P where $P \in \mathcal{O}$.

We denote the extended calculus by $\mathscr{H}(\mathscr{C}), \mathcal{O}$. Note that - strictly speaking - (15) need not be a "proof rule" in the sense of our definition since $\mathcal{O} \subseteq LF$ may be undecidable.

A *formal proof* (of $H \to H'$) in $\mathscr{H}(\mathscr{C}), \mathcal{O}$ is defined to be a sequence $\ell_1 \ldots \ell_n$ of proof lines (with $\ell_n = H \to H'$) such that for all $i = 1, \ldots, n$ ℓ_i can be added to the sequence $\ell_i \ldots \ell_{i-1}$ by applying one of the rules (1)-(15). Then $H \to H'$ is called *formally provable* or *deducible* in $\mathscr{H}(\mathscr{C}), \mathcal{O}$ ($\vdash_{\mathscr{H}(\mathscr{C}),\mathcal{O}} H \to H'$ for short). Observe that the property whether a given sequence of proof lines is a formal proof in $\mathscr{H}(\mathscr{C}), \mathcal{O}$ is decidable relative to $\mathcal{O}$.

Some comments on the proof rules of $\mathscr{H}(\mathscr{C})$ may be helpful. Rules of the type (1) and (12) occur if Hoare-like deduction systems are transformed into Hoare-like sequent calculi (cf. [2]). Rule (7) combines subrules for variable resp. procedure declarations introduced in [32] resp. [9]. As renamings of identifiers are only done within the formulas P and Q, it might seem that this rule leads to dynamic scope. A closer look at $\mathscr{H}(\mathscr{C})$ reveals, however, that all renamings needed to handle scope problems are done here in rule (8) and rule (14). This is in agreement with our copy rule approach to semantics explained in Sect. 4 where the necessary renamings are done firstly by switching from π to $\emptyset|\pi_d$ (here rule (14)) and secondly whenever the copy rule is applied (here rule (14)).

The recursion rule (8) is the only rule which depends on the chosen copy rule $\mathscr{C}$. Since its syntactical form is similar to the corresponding rule of [9], rule (8) is a generalization of a first version of this rule presented in [19]. Rule (9), constructed according to the Substitution Lemma, is new and essential for our completeness results in Sect. 9 and 10. Note that the environment extension rule of [3] and [9] is subsumed under this rule. The rules (10) and (11) are due to [13] and [12] respectively.

8. The Soundness Theorem

In this section we want to show that the calculi $\mathscr{H}(\mathscr{C})$ are "sound". To this end, we present a semantical *interpretation of proof lines* and show that the proof rules "respect" this interpretation. One might think of employing the full semantics. $\Sigma_{\mathscr{C}}^{\mathscr{I}} = \bigcup_{j \geq 0} \Sigma_{\mathscr{C},j}^{\mathscr{I}}$ to interpret proof lines. However, the presentation in [1] and [6] shows that this idea, which seems to be quite natural when denotational semantics methods as in [1, 6] are employed, can lead to a rather complicated interpretation, and consequently, to a rather tedious soundness proof.

We instead use the *approximating semantics* $\Sigma_{\mathscr{C},j}^{\mathscr{I}}$ to interpret proof lines because this is in a way simpler than $\Sigma_{\mathscr{C}}^{\mathscr{I}}$. Starting with copy rules to define the semantics of programs this is a very natural idea, and it leads in fact to a rather simple soundness proof. Although a number of papers (e.g. [1, 9, 13, 20]) have employed approximating semantics in order to prove the soundness of their recursion rule, they have not used it as a means to interpret proof lines. Apt [2] has independently detected this advantage of approximating semantics when trying to simplify his proofs in [1].

Definition 7. Let $\mathscr{C}, \mathscr{C}'$ be copy rules and $j \geq 0$.

a) We write $\models_{\mathscr{I},\mathscr{C},j} P$ if $\models_{\mathscr{I}} P$ holds.

b) We write $\models_{\mathscr{I},\mathscr{C},j} \{P\} E|S \{Q\}$ if $\Sigma_{\mathscr{C},j}^{\mathscr{I}}(E|S)(P) \subseteq Q$ holds.[9]

c) We write $\models_{\mathscr{I},\mathscr{C},j} \{P\} \pi \{Q\}$ if $\Sigma_{\mathscr{C},j}^{\mathscr{I}}(\pi)(P) \subseteq Q$ holds.

d) We write $\models_{\mathscr{I},\mathscr{C},j} H$ if $\models_{\mathscr{I},\mathscr{C},j} h$ holds for every $h \in H$.

e) A proof line $H \rightarrow H'$ is *valid* w.r.t. $\mathscr{I}$ and $\mathscr{C}$ if for every $j \geq 0$ $\models_{\mathscr{I},\mathscr{C},j} H$ implies $\models_{\mathscr{I},\mathscr{C},j} H'$.

f) A proof rule ρ is *$\mathscr{C}$-sound* if ρ preserves the validity w.r.t. $\mathscr{I}$ and $\mathscr{C}$ for every interpretation $\mathscr{I}$.

g) The Hoare-like calculus $\mathscr{H}(\mathscr{C})$ is *sound* for a programming language $\Pi = (L, \Sigma_{\mathscr{C}'})$ if for every interpretation $\mathscr{I}$, $\pi \in L$ and $P, Q \in LF$ whenever $\{P\} \pi \{Q\}$ is formally provable in $\mathscr{H}(\mathscr{C})$, $Th(\mathscr{I})$ then $\{P\} \pi \{Q\}$ is valid w.r.t. $\mathscr{I}$ and $\mathscr{C}'$.

9 Recall the notational convention preceding Lemma 1

Note that $\{P\}\pi\{Q\}$ is valid w.r.t. $\mathscr{I}$ and $\mathscr{C}$ iff $\Sigma_{\mathscr{C}}^{\mathscr{I}}(\pi)(P)\subseteq Q$ holds, i.e. for programs π our notion of validity coincides with the usual concept of partial correctness.

Theorem 1 (Soundness Theorem). *All proof rules of $\mathscr{H}(\mathscr{C})$ are $\mathscr{C}$-sound.*

Proof. In detail we study the $\mathscr{C}$-soundness of the recursion rule (8). [The $\mathscr{C}$-soundness of the remaining rules are – unless trivial – immediate consequences of Lemma 1 (for the easy rules depending on properties of program functions only) and Lemma 4 (Rule of Substitution) together with Corollary 2 and Remark 1.]

Let $H(j)$, $p(j)$ and $B(j)$ be abbreviations for the assertions "$\models_{\mathscr{I},\mathscr{C},j} H$", "$\Sigma_{\mathscr{C},j}^{\mathscr{I}}(E_i|p_i(\bar{y}_i:\bar{r}_i))(P_i)\subseteq Q_i$ holds for all $i=1,\dots,m$" and "$\Sigma_{\mathscr{C},j}^{\mathscr{I}}(E_i|B_{i\mathscr{C}})(P_i)\subseteq Q_i$ holds for all $i=1,\dots,m$" respectively. Now suppose that the premise of rule (8) is valid w.r.t. $\mathscr{I}$ and $\mathscr{C}$. We have to show that the conclusion of rule (8) is also valid w.r.t. $\mathscr{I}$ and $\mathscr{C}$, i.e. that

$$H(j) \text{ implies } p(j) \qquad \text{for all } j\geq 0.$$

We proceed by induction on j. The case "$j=0$" is trivial because $p(0)$ is vacuously true. Let us now consider the induction step "$j\to j+1$", i.e. assume that $H(j+1)$ holds. Then $H(j)$ [because of $\Sigma_{\mathscr{C},j}^{\mathscr{I}}\subseteq\Sigma_{\mathscr{C},j+1}^{\mathscr{I}}$] and therefore $p(j)$ hold. This implies $B(j)$ [premise of rule (8)]. By Corollary 2, we conclude $p(j+1)$ what had to be proved. □

Corollary 3. *$\mathscr{H}(\mathscr{C})$ is sound for $\Pi=(L_{\text{Algol}},\Sigma_{\mathscr{C}})$.*

Finally, let us note that the provision of a semantical interpretation for proof lines, especially for the symbol "$\to$", ensures that the soundness of $\mathscr{H}(\mathscr{C})$ cannot be destroyed by introducing additional sound proof rules. This is not necessarily true for Hoare-like systems which employ the provability sign $\vdash$ in their rules (e.g. in [9]) as observed in [36].

9. The Completeness Theorem

For two reasons we cannot expect that all Hoare formulas $\{P\}\pi\{Q\}$ which are valid w.r.t. $\mathscr{I}$ and $\mathscr{C}$ can be formally proved in $\mathscr{H}(\mathscr{C})$, even if we admit the full theory $Th(\mathscr{I})$ as an oracle.

Firstly, certain sets X of states may not be expressible by a (first order) formula, i.e. there may be no formula P in LF with $\mathscr{S}t^{\mathscr{I}}(P)=X$. This phenomenon is due to Hoare's approach to describe the correctness of programs by formulas and arises already in the context of while-programs. Examples are given in [10, 44] and [7]. To overcome this difficulty Cook [10] proposed to study completeness *relative* to the assumption of expressibility.

Definition 8. Let $\mathscr{C}$ and $\mathscr{C}'$ be copy rules.

a) *FOL* is *expressive* w.r.t. $\mathscr{I}$ and $\mathscr{C}$ if for every unit $E|S$ and every formula P there exists a formula Q

$$\Sigma_{\mathscr{C}}^{\mathscr{I}}(E|S)(\mathscr{S}t^{\mathscr{I}}(P))=\mathscr{S}t^{\mathscr{I}}(Q).$$

b) $\mathcal{H}(\mathcal{C})$ is *relatively complete* (or *complete in the sense of Cook*) for $\Pi=(L,\Sigma_{\mathcal{C}'})$ if for every interpretation $\mathcal{I}$ such that FOL is expressive w.r.t. $\mathcal{I}$ and $\mathcal{C}'$ the following holds: Whenever $\{P\}\pi\{Q\}$ with $\pi\in L$ and $P, Q\in LF$ is valid w.r.t. $\mathcal{I}$ and $\mathcal{C}'$ then $\{P\}\pi\{Q\}$ is formally provable in $\mathcal{H}(\mathcal{C})$, $Th(\mathcal{I})$.

An additional source of incompleteness is the unrestricted use of the procedure concept in L_{Algol}. This was first observed by Clarke [9] who showed that it is impossible to obtain a sound and relatively complete Hoare-like calculus for L_{Algol} with static scope semantics $\Sigma_{\mathcal{C}_{60}}$. More precisely Clarke showed that this incompleteness result holds already for the subset $L_{\text{Pas}}\subseteq L_{\text{Algol}}$ of all *PASCAL-like programs* where procedures which occur as actual procedure parameters are disallowed to have own formal procedures as parameters [21]. (Incompleteness for the full set L_{Algol} follows already from the undecidability of the formal reachability of procedures proved in Langmaack [26].) As crucial step Clarke proves that even for all finite interpretations $\mathcal{I}$ with $|\mathcal{D}|\geq 2$ it is undecidable whether programs $\pi\in L_{\text{Pas}}$ are *divergent* w.r.t. $\Sigma^{\mathcal{I}}_{\mathcal{C}_{60}}$, i.e. whether $\Sigma^{\mathcal{I}}_{\mathcal{C}_{60}}(\pi)=\emptyset$ holds.[10] (See [31] for a fuller discussion of this issue.)

Thus the Hoare-like calculi $\mathcal{H}(\mathcal{C})$ can be relatively complete only for *restricted* languages $\Pi=(L,\Sigma_{\mathcal{C}})$ where either $\Sigma_{\mathcal{C}}$ is a simplified semantics as compared with the static scope semantics $\Sigma_{\mathcal{C}_{60}}$ or L is a proper sublanguage of L_{Algol}. To handle such restrictions in a uniform way we introduce the concept of $\mathcal{C}$-boundedness and show in the Completeness Theorem that $\mathcal{H}(\mathcal{C})$ is relatively complete for $\mathcal{C}$-bounded programs. The development of this theorem takes the rest of this section and is divided into four steps.

Step 1. This step is due to Gorelick [13]. We show that in order to deduce an arbitrary valid Hoare formula $\{P\}E|S\{Q\}$ in $\mathcal{H}(\mathcal{C})$, $Th(\mathcal{I})$ it suffices to deduce a so-called most general formula [13] $\{G\}E|S\{G'\}$ where G and G' are special formulas which are independent of P and Q.

Definition 9. Let FOL be expressive w.r.t. $\mathcal{I}$ and $\mathcal{C}$. A Hoare formula $\{G\}E|S\{G'\}$ is called a *most general formula* w.r.t. $\mathcal{I}$ and $\mathcal{C}$ if the following holds:

$$G=(\bar{x}=\bar{y})\quad\text{and}\quad \Sigma^{\mathcal{I}}_{\mathcal{C}}(E|S)(\mathcal{S}t^{\mathcal{I}}(G))=\mathcal{S}t^{\mathcal{I}}(G')$$

where $\bar{x}$ and $\bar{y}$ are disjoint lists of distinct variables with $|\bar{x}|=|\bar{y}|$ and $\text{free}(\min(E|S))\subseteq\{\bar{x}\}$. (By $(\bar{x}=\bar{y})$ we denote the formula $x_1=y_1\wedge\ldots\wedge x_n=y_n$ provided $\bar{x}=x_1,\ldots,x_n$ and $\bar{y}=y_1,\ldots,y_n$.)

Intuitively G' describes the final values of the "active" variables $\bar{x}$ of $E|S$ in terms of the initial values represented by $\bar{y}$.

Lemma 5. *Assume that FOL is expressive w.r.t. $\mathcal{I}$ and $\mathcal{C}$. Let $\{G\}E|S\{G'\}$ be a most general formula w.r.t. $\mathcal{I}$ and $\mathcal{C}$.*
If $H\to\{G\}E|S\{G'\}$ *is formally provable in* $\mathcal{H}(\mathcal{C})$, $Th(\mathcal{I})$
and $\{P\}E|S\{Q\}$ *is valid w.r.t.* $\mathcal{I}$ *and* $\mathcal{C}$
then $H\to\{P\}E|S\{Q\}$ *is also formally provable in* $\mathcal{H}(\mathcal{C})$, $Th(\mathcal{I})$.

[10] For $|\mathcal{D}|=1$ the divergence problem is still decidable due to Langmaack [28]

Proof. The case that $\mathrm{free}(P \vee Q) \cap \{\bar{y}\} = \emptyset$ is treated in Lemma 5 of [13]. (Thereby the rules (10), (11), (13) and (15) are needed.) The case that $\mathrm{free}(P \vee Q) \cap \{\bar{y}\} \neq \emptyset$ can easily be reduced to the first case by using a suitable substitution σ, Lemma 1, c) and rule (10). □

Step 2. In this step we reduce the problem of deducing an arbitrary Hoare formula $\{P\}E|S\{Q\}$ in $\mathscr{H}(\mathscr{I}), Th(\mathscr{C})$ to deducing certain most general formulas $\{G\}E'|S'\{G'\}$ where S' is a *procedure call.*

Definition 10. Let $\mathscr{C}$ be an arbitrarily fixed copy rule. We write $E_0|S_0 \xrightarrow{\mathscr{P}} E|p(\bar{x}:\bar{q})$ if there exists a path $\not{p}$: $E_0|S_0 \xrightarrow[\mathscr{C}]{} \dots \xrightarrow[\mathscr{C}]{} E|p(\bar{x}:\bar{q})$ such that $p(\bar{x}:\bar{q})$ is correct w.r.t. E and $E|p(\bar{x}:\bar{q})$ is the only procedure call unit in $\not{p}$.

Note that this definition is independent of the particular $\mathscr{C}$. Informally $E_0|S_0 \xrightarrow{\mathscr{P}} E|p(\bar{x}:\bar{q})$ holds iff $p(\bar{x}:\bar{q})$ is a correct procedure call in the main part of S_0 equipped with the appropriate environment E. Hence there are only finitely many units $E|p(\bar{x}:\bar{q})$ with $E_0|S_0 \xrightarrow{\mathscr{P}} E|p(\bar{x}:\bar{q})$.

Lemma 6. *Let FOL be expressive w.r.t.* $\mathscr{I}$ *and* $\mathscr{C}$ *and a unit* $E|S$ *be given. Let* $E_i|p_i(\dots)$, $i=1,\dots,m$, $m \in \mathbb{N}_0$, *be exactly those units with* $E|S \xrightarrow{\mathscr{P}} E_i|p_i(\dots)$ *and* $\{G_i\}E_i|p_i(\dots)\{G_i'\}$ *be most general formulas w.r.t.* $\mathscr{I}$ *and* $\mathscr{C}$, $i=1,\dots,m$.

If $\quad H \to \bigcup_{i=1}^{m} \{\{G_i\}E_i|p_i(\dots)\{G_i'\}\}$ is formally provable in

$\mathscr{H}(\mathscr{C}), Th(\mathscr{I})$ and $\{P\}E|S\{Q\}$ is calid w.r.t. $\mathscr{I}$ and $\mathscr{C}$

then $H \to \{P\}E|S\{Q\}$ is formally provable in $\mathscr{H}(\mathscr{C}), Th(\mathscr{I})$.

Proof. We proceed by induction on the structure of S.

Case 1. $S =$ **error** or $S = x := e$

In this case $m = 0$ holds. The assertion of the lemma is proved by using Corollary 2, Lemma 1, e), f) and the rules (2), (3), (12), (13) and (15).

Case 2. $S = p(\bar{x}:\bar{q})$

If $p(\bar{x}:\bar{q})$ is an incorrect procedure call, $m = 0$ holds and we use the rules (4) and (12). Otherwise $m = 1$ and $E_1|p_1(\dots) = E|p(\dots)$. Then we apply Lemma 5.

Case 3. $S =$ **begin var** x; $E_0 S_1$ **end**

Let $E_1' = \mathrm{add}(E, E_0)$. By Corollary 2 and Lemma 1, i)

$$\{P[y/x] \wedge x = \omega\}E_1'|S_1\{Q[y/x]\}$$

is valid w.r.t. $\mathscr{I}$ and $\mathscr{C}$ (where $y \notin \mathrm{free}(P \vee Q) \cup \mathrm{free}(\min(E_1'|S_1))$). Observe that whenever $E_1'|S_1 \xrightarrow{\mathscr{P}} E'|p'(\dots)$ holds then $E'|p'(\dots) = E_i|p_i(\dots)$ for some $i \in \{1,\dots,m\}$. Since S_1 is a proper substatement of S, we conclude by induction hypothesis and rule (12) that $\vdash_{\mathscr{H}(\mathscr{C}), Th(\mathscr{I})} H \to \{P[y/x] \wedge x = \omega\}E_1'|S_1\{Q[y/x]\}$ holds. Rule (7) yields $\vdash_{\mathscr{H}(\mathscr{C}), Th(\mathscr{I})} H \to \{P\}E|S\{Q\}$.

Case 4. $S = S_1; S_2$ or $S =$ **if** b **then** S_1 **else** S_2 **fi**

The argumentation is similar to that of case 3. □

Let us summarize Step 2 as follows: $\mathscr{H}(\mathscr{C})$ is relatively complete for programming languages $\Pi=(L, \Sigma_{\mathscr{C}})$ if $\mathscr{H}(\mathscr{C})$ is able to "master" all procedure calls induced by programs in L.

Step 3. In this step we present two (equivalent) conditions for programs which are sufficient for this "mastering" of procedure calls, one is more convenient for proving the Completeness Theorem in Step 4 (finite $\mathscr{C}$-index) and the other is directed towards the applications of this theorem in the next section ($\mathscr{C}$-boundedness).

Consider a unit $E|S$. By a *reference chain* of length n in $E|S$ we mean a sequence of non-formal procedure identifier occurrences

$$(i_1, p_1) \rightarrow (i_2, p_2) \rightarrow \dots \rightarrow (i_n, p_n)$$

in **begin** ES **end** such that the declarations of $(i_1, p_1), \dots, (i_n, p_n)$ are all distinct, (i_1, p_1) occurs in S and, for $j=2, \dots, n$, $(i_{j-1}, p_{j-1}) \rightarrow (i_j, p_j)$ denotes that (i_j, p_j) occurs freely in the procedure body belonging to (i_{j-1}, p_{j-1}).

Definition 11. A program π is called *$\mathscr{C}$-bounded*, $\mathscr{C}$ a copy rule, if there exists a constant $k \in \mathbb{N}$ such that whenever $\emptyset|\pi_d \overset{*}{\underset{\mathscr{C}}{\rightarrow}} E|S$ holds and S is a w.r.t. E correct procedure call then the lengths of the reference chains in $E|S$ are bounded by k.

Intuitively speaking a program is $\mathscr{C}$-bounded if applications of the copy rule $\mathscr{C}$ do not lead to units with reference chains of arbitrary length.

Definition 12. A program π is said to have a *finite $\mathscr{C}$-index*, $\mathscr{C}$ a copy rule, if the substitutional equivalence $\cong$ induces only finitely many equivalence classes in the set

$$\mathscr{R}_{\mathscr{C}}(\pi) = \{E|S \mid \text{where } \emptyset|\pi_d \overset{*}{\underset{\mathscr{C}}{\rightarrow}} E|S \text{ and } S \text{ is a w.r.t. } E \text{ correct procedure call}\}.$$

Thus for programs π with a finite $\mathscr{C}$-index we can split the in general infinite set $\mathscr{R}_{\mathscr{C}}(\pi)$ into finitely many classes of procedure call units which differ only "unessentially". This ability of reducing the infinity of $\mathscr{R}_{\mathscr{C}}(\pi)$ is crucial for obtaining finite correctness proofs for π, i.e. for proving the desired completeness result. The basic idea of this proof technique appeared already in [13] and [9] where (the equivalent to) $\mathscr{R}_{\mathscr{C}}(\pi)$ is called *recursive cycle* resp. *range*; it is also present - in a rather hidden way - in [6] (see Lemma 9.26 in Chap. 9). How powerful the completeness results will be depends on how the word "unessential" is interpreted. We interpret it as "substitutional equivalent". In particular, this allows us to deal with the problem of sharing as explained in Sect. 6. In contrast Gorelick and Clarke essentially required that $\mathscr{R}_{\mathscr{C}}(\pi)$ *itself* is finite. For ALGOL-like programs π such an approach is appropriate only if we restrict ourselves to the naive copy rule $\mathscr{C}=\mathscr{C}_n$, i.e. to dynamic scope.

Lemma 7. *A program π is $\mathscr{C}$-bounded iff π has a finite $\mathscr{C}$-index.*

Proof. For both implications we need the following relationship between reference chains in a unit $E|S$ and the minimal unit $\min(E|S)$:

(*) If $E_0|S=\min(E|S)$ then E_0 consists exactly of all procedures **proc** $p(\dots);\dots;$ of E such that p is reachable by a reference chain $(i_1, p_1) \rightarrow \dots \rightarrow (i_n, p_n)$ with $p_n = p$ in $E|S$.

"*if*": By $(*)$ the following holds for all units $E|S$ and $E'|S'$. If there is a reference chain of length k in $E|S$ and $E|S \cong E'|S'$ holds then there is also a reference chain of length k in $E'|S'$. Thus whenever π has a finite $\mathscr{C}$-index there exists a constant $k \in \mathbb{N}$ such that for every unit $E|S$ in $\mathscr{R}_{\mathscr{C}}(\pi)$ the reference chains in $E|S$ have a length of at most k, i.e. π is $\mathscr{C}$-bounded.

"*only if*": For $j \in \mathbb{N}$ let $R_j = \mathscr{R}_{\mathscr{C}}(\pi) \cap \{E|S \mid \text{where } |\min(E|S)| = j\}$. ($|\min(E|S)|$ denotes the length of $\min(E|S)$ considered as a string over T.) Observe that for every program π and every $j \in \mathbb{N}$ the substitutional equivalence $\cong$ induces only finitely many equivalence classes in R_j. Now let π be $\mathscr{C}$-bounded. By $(*)$ there exists a constant $k \in \mathbb{N}$ such that $|\min(E|S)| \leqq k$ holds for every $E|S$ in $\mathscr{R}_{\mathscr{C}}(\pi)$. Thus $\mathscr{R}_{\mathscr{C}}(\pi) = R_1 \cup \ldots \cup R_k$, but this implies that π has a finite $\mathscr{C}$-index. $\square$

We remark that it is in general undecidable whether an ALGOL-like program π is $\mathscr{C}$-bounded resp. has a finite $\mathscr{C}$-index. For $\mathscr{C} = \mathscr{C}_{60}$ this follows from the undecidability of the *macro program problem* proved in [27]. Hence the set of all $\mathscr{C}_{60}$-bounded programs would not form a proper syntax in the sense of Sect. 4.

Step 4. We are now prepared to prove

Theorem 2 (Completeness Theorem). *Assume that FOL is expressive w.r.t.* $\mathscr{I}$ *and* $\mathscr{C}$. *Let* $\pi \in L_{\text{Algol}}$ *be* $\mathscr{C}$*-bonded and* $\{P\}\pi\{Q\}$ *is formally provable in* $\mathscr{H}(\mathscr{C})$, $Th(\mathscr{I})$.

Proof. By Lemma 7, π has a finite $\mathscr{C}$-index. Thus we can choose a system $\mathfrak{M} = \{E_i|p_i(\ldots) \mid i = 1, \ldots, m\}$ of representatives for the finitely many equivalence classes w.r.t. $\cong$ in $\mathscr{R}_{\mathscr{C}}(\pi)$ such that $\emptyset|\pi_d \xrightarrow{\mathscr{I}} E|p(\ldots)$ implies $E|p(\ldots) \in \mathfrak{M}$. Let $\underline{\mathscr{C}}$ be a specimen of $\mathscr{C}$ and define

$$\mathfrak{N} = \mathfrak{M} \cup \left\{ E|p(\ldots) \;\middle|\; \begin{array}{l} \text{where } E_i|p_i(\ldots) \xrightarrow[\underline{\mathscr{C}}]{} E_i|B_{i\underline{\mathscr{C}}} \xrightarrow{\mathscr{I}} E|p(\ldots) \\ \text{holds for some } i \text{ with } 1 \leqq i \leqq m \end{array} \right\}$$

We take the following numeration:

$$\mathfrak{N} \setminus \mathfrak{M} = \{E_i|p_i(\ldots) \mid i = m+1, \ldots, m+n\} \quad \text{with} \quad n \in \mathbb{N}_0.$$

Since π has a finite $\mathscr{C}$-index, we may choose most general formulas $\{G_i\}E_i|p_i(\ldots)$ $\{G_i'\}$ w.r.t. $\mathscr{I}$ and $\mathscr{C}$, $i = 1, \ldots, m+n$, such that for every $j \in \{m+1, \ldots, m+n\}$ there exists an $i \in \{1, \ldots, n\}$ and a substitution σ which is injective on $\text{free}(G_i \vee G_i') \cup \text{idf}(\min(E_i|p_i(\ldots))$ and satisfies $E_i|p_i(\ldots)_\sigma \cong E_j|p_j(\ldots)$ and $G_i\sigma = G_j$. Lemma 1, a) and b), implies that $\models_{\mathscr{I}} G_i'\sigma \leftrightarrow G_j'$ holds. Using the rules (9), (13) and (15) we get the following auxiliary assertion:

$(*)$ For all $j \in \{m+1, \ldots, m+n\}$ there exists an $i \in \{1, \ldots, m\}$
such that $\vdash_{\mathscr{H}(\mathscr{C}), Th(\mathscr{I})} H \rightarrow \{G_i\}E_i|p_i(\ldots)\{G_i'\}$
implies $\vdash_{\mathscr{H}(\mathscr{C}), Th(\mathscr{I})} H \rightarrow \{G_j\}E_j|p_j(\ldots)\{G_j'\}$.

We are now ready to show the formal provability of $\{P\}\pi\{Q\}$:

(1) $\vdash_{\mathscr{H}(\mathscr{C}), Th(\mathscr{I})} \bigcup_{i=1}^{m} \{\{G_i\}E_i|p_i(\ldots)\{G_i'\}\} \rightarrow \bigcup_{i=1}^{m} \{\{G_i\}E_i|p_i(\ldots)\{G_i'\}\}$ by rule (1).

(2) $\vdash_{\mathscr{H}(\mathscr{C}), Th(\mathscr{I})} \bigcup_{i=1}^{m} \{\{G_i\}E_i|p_i(\ldots)\{G_i'\}\} \to \bigcup_{i=1}^{m+n} \{\{G_i\}E_i|p_i(\ldots)\{G_i'\}\}$
by $(*)$ and rule (12).

Define $B_{i\underline{\mathscr{C}}}$ by $E_i|p_i(\ldots) \xrightarrow[\mathscr{C}]{} E_i|B_{i\underline{\mathscr{C}}}$, $i=1, \ldots, m+n$. By Corollary 2 and the definition of G_i and G_i', $\{G_i\}E_i|B_{i\mathscr{C}}\{G_i'\}$ is valid w.r.t. $\mathscr{I}$ and $\mathscr{C}$ for $i=1, \ldots, m+n$. By the definition of $\mathfrak{M}$, an application of Lemma 6 and rule (12) leads to

(3) $\vdash_{\mathscr{H}(\mathscr{C}), Th(\mathscr{I})} \bigcup_{i=1}^{m} \{\{G_i\}E_i|p_i(\ldots)\{G_i'\}\} \to \bigcup_{i=1}^{m} \{\{G_i\}E_i|B_{i\underline{\mathscr{C}}}\{G_i'\}\}$.

(4) $\vdash_{\mathscr{H}(\mathscr{C}), Th(\mathscr{I})} \bigcup_{i=1}^{m} \{\{G_i\}E_i|p_i(\ldots)\{G_i'\}\}$ by rule (8).

By the definition of $\mathfrak{M}$, a further application of Lemma 6 and rule (12) yields

(5) $\vdash_{\mathscr{H}(\mathscr{C}), Th(\mathscr{I})} \{P\}\pi\{Q\}$ what was to be proved. $\square$

An equivalent formulation of Theorem 2 is

Corollary 4. *Let L be a decidable subset of L_{Algol} such that every program $\pi \in L$ is $\mathscr{C}$-bounded. Then $\mathscr{H}(\mathscr{C})$ is relatively complete for $\Pi=(L, \Sigma_{\mathscr{C}})$.*

The next question is now for which languages L resp. which copy rules $\mathscr{C}$ the property of $\mathscr{C}$-boundedness is fulfilled.

10. Applications of the Completeness Theorem

In this section we answer the last question by presenting several corollaries to Theorem 2 and relating them to results and claims published in the literature. We start with the naive copy rule $\mathscr{C}_n$.

Corollary 5. *$\mathscr{H}(\mathscr{C}_n)$ is relatively complete for $\Pi=(L_{\text{Algol}}, \Sigma_{\mathscr{C}_n})$.*

Proof. Let $\pi \in L_{\text{Algol}}$. Since $\mathscr{C}_n$ does not rename procedure identifiers (in fact not even variables), there is a universal bound k for the number of procedures in E whenever $\emptyset|\pi_d \xrightarrow[\mathscr{C}_n]{*} E|S$ holds: Simply take k to be the number of procedures in π_d. This fact implies the $\mathscr{C}_n$-boundedness of π. $\square$

As mentioned in Sect. 4 the naive copy rule is closely related to the dynamic scope semantics Σ_{dyn}: $\Sigma_{\text{dyn}}^{\mathscr{I}}(\pi) = \Sigma_{\mathscr{C}_n}^{\mathscr{I}}(\emptyset|\pi)$, thus $\Sigma_{\mathscr{C}_n}^{\mathscr{I}}(\pi) = \Sigma_{\text{dyn}}^{\mathscr{I}}(\pi_d)$. Adopting this little change in rule (14) of $\mathscr{H}(\mathscr{C}_n)$, we obtain a relatively complete Hoare-like calculus for $\Pi=(L_{\text{Algol}}, \Sigma_{\text{dyn}})$ by Corollary 5.

Dynamic scope semantics has been studied by several authors. Sound and relatively complete Hoare-like (deduction) systems for dynamic scope are presented in [10, 13] and [9]. In all papers sharing is disallowed; Cook and Gorelick don't consider procedures as parameters. Additionally, Cook does not admit recursive procedures.

Sound and relatively complete rules which deal with sharing are presented in [8]. Due to some restrictions about procedures (e.g. no procedures as parameters) static scope coincides with dynamic scope in this paper. On the other hand, arrays are considered as an additional language feature.

We now switch to the "most recent" copy rule $\mathscr{C}_{mr}$.

Corollary 6. *$\mathscr{H}(\mathscr{C}_{mr})$ is relatively complete for $\Pi=(L_{\text{Algol}}, \Sigma_{\mathscr{C}_{mr}})$.*

The proof of Corollary 5 remains valid here because also $\mathscr{C}_{mr}$ does not rename procedure identifiers. The "most recent" copy rule will be used to deal with programs with the formal "most recent" property (see Corollary 8).

Let us summarize the last results as follows: For simplified semantics such as Σ_{dyn}, $\Sigma_{\mathscr{C}_n}$ or $\Sigma_{\mathscr{C}_{mr}}$ it is no problem to find a Hoare-like calculus which is relatively complete for the full ALGOL-like language L_{Algol}. As mentioned in Sect. 9 this is not longer true for the ALGOL 60 copy rule $\mathscr{C}_{60}$ inducing the static scope semantics $\Sigma_{\mathscr{C}_{60}}$. For $\mathscr{C}_{60}$ we shall present two sublanguages $L\subseteq L_{\text{Algol}}$ which have a relatively complete Hoare-like calculus due to Theorem 2.

First we consider the set L_{gf} of all programs *without global formal procedure identifiers.* Thereby an identifier β of a program π is called *global formal* if there is an occurrence (i, β) in π which is free in some extended procedure body in π and defined as a formal parameter. Of course L_{gf} is a decidable subset of L_{Algol}.

Corollary 7. *$\mathscr{H}(\mathscr{C}_{60})$ is relatively complete for $\Pi=(L_{gf}, \Sigma_{\mathscr{C}_{60}})$.*

Proof. Let $\pi\in L_{gf}$ and N be the number of procedures in π_d. To prove $\mathscr{C}_{60}$-boundedness of π we show by induction on k that for all paths

$$\not{p}_k\colon\ \emptyset|\pi_d=E_1|S_1 \xrightarrow[\mathscr{C}_{60}]{} \dots \xrightarrow[\mathscr{C}_{60}]{} E_k|S_k$$

the lengths of the reference chains in $E_k|S_k$ are bounded by N. (Notice that there may be no universal bound depending on π_d only for the number of procedures in E_k.)

This is clearly true for $k=1$. Consider now the induction step "$k\to k+1$" and assume that the assertion holds already for all $j\leqq k$. The only non-trivial situation is that S_k is a procedure call $S_k=p(\dots)$. Let **proc** $p(\dots); B; \in E_k$ and $B'=S_{k+1}$. Then there is a unique index $j\leqq k$ such that **proc** $p(\dots); B$; is in E_{j+1}, but not in E_j. Consider now an arbitrary reference chain $(i'_1, p'_1)\to\dots\to(i'_n, p'_n)$ in $E_{k+1}|S_{k+1}=E_{k+1}|B'$.

Case 1. All $(i'_1, p'_1), \dots, (i'_n, p'_n)$ are defined in B'. Then there is a corresponding reference chain $(i_1, p_1)\to\dots\to(i_n, p_n)$ in $E_j|S_j$ where all $(i_1, p_1), \dots, (i_n, p_n)$ are defined in B.

Case 2. There exists an $m\leqq n$ such that (i'_m, p'_m) is *free* in B'.

Subcase 2.1. (i'_m, p'_m) occurs in the main part of B'. (Thus $m=1$.) Then there is a corresponding reference chain $(i_1, p'_n)\to\dots\to(i_n, p'_n)$ either in $E_j|S_j$ or in $E_k|S_k$ depending on whether p'_1 was already present in B or has just been inserted as an actual parameter of $S_k=p(\dots)$.

Subcase 2.2. (i'_m, p'_m) occurs in the body of a procedure in B',
e.g. $B'=$ **begin**...**proc** $q'(\dots)$; **begin**...p'_m...**end**...**end.**
Thus $B=$ **begin**...**proc** $q(\dots)$; **begin**...r...**end**...**end**
where q and q' (r and p'_m) occur at the same position in B and B' respectively. Since $\pi\in L_{gf}$, we conclude $r=p'_m$. Thus we can find a corresponding reference chain

$$(i_1, p_1)\to\dots\to(i_{m-1}, p_{m-1})\to(i_m, p'_m)\to\dots\to(i_n, p'_n) \quad \text{in } E_j|S_j$$

where $(i_1, p_1), \dots, (i_{m-1}, p_{m-1})$ are defined in B.

In both cases 1 and 2 there exists a reference chain of length n in $E_j|S_j$ or $E_k|S_k$. By induction hypothesis $n \leqq N$ holds. □

The second sublanguage we investigate in connection with the copy rule $\mathscr{C}_{60}$ is the set L_{mr} of all programs satisfying the *formal "most recent" property* [35, 22]. This property is more complex to define than the ones considered so far. It originates from the study of *run time systems* supporting the execution of ALGOL-like programs with recursive procedures. It is well-known such systems work with a run time stack of so-called *activation records* and use so-called *static chains* to access the current activation record of an identifier in this stack [11, 39, 14, 35].

Informally speaking a program has the "most recent" property if at run time the static chain pointer of a procedure p always points to the most recent, not yet completed, activation record of that procedure q which lexicographically encloses p (see [11], p. 318). Such programs are of interest because run time systems can apply a simple "most recent search strategy" for them (see [35]). We formalize now this property following [22]. The basic idea is that every state of the run time stack corresponds to a certain path $\not{p}$: $\pi_1 \vdash_{\mathscr{C}_{60}} \ldots \vdash_{\mathscr{C}_{60}} \pi_k$ of programs where the single programs π_i correspond to the single activation records in the stack.

Consider a program $\pi \in L_{\text{Algol}}$. First we define the notion of an *associated procedure identifier:* Let $\pi_d \overset{*}{\vdash}_{\mathscr{C}_{60}} \pi'$ and (i, p') be an occurrence of a procedure identifier p' in π'. Then the declaration **proc** $p'(\ldots); \ldots;$ of p' in π' is a copy of exactly one procedure **proc** $p(\ldots); \ldots;$ in π_d. We call p the associated procedure identifier of (i, p') and write $p = \text{ass}(i, p')$. Consider now a path $\not{p}$: $\pi_d \overset{*}{\vdash}_{\mathscr{C}_{60}} \pi' \vdash_{\mathscr{C}_{60}} \pi''$ and let $\text{stm}(\pi', \pi'') = (i, p'(\ldots))$. (Recall that by Definition 2 $\text{stm}(\pi', \pi'')$ is the unique occurrence of a procedure call $p'(\ldots)$ in π' which generates π''.) Then we call $p = \text{ass}(i, p')$ the associated procedure identifier of π'' and write $p = \text{ass}(\pi'')$. Intuitively p tells us which procedure in π_d generated the innermost modified procedure body $B = blk(\pi', \pi'')$ in π''.

Consider now a path

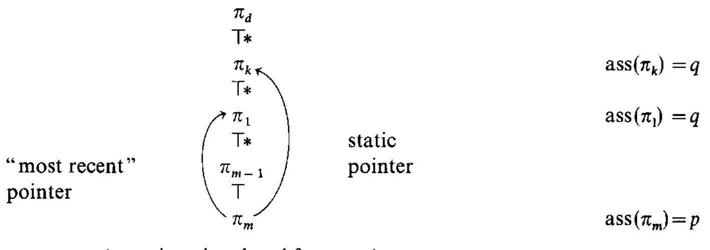
$$\not{p}:\ \pi_d = \pi_0 \vdash_{\mathscr{C}_{60}} \ldots \vdash_{\mathscr{C}_{60}} \pi_k \vdash_{\mathscr{C}_{60}} \ldots \vdash_{\mathscr{C}_{60}} \pi_l \vdash_{\mathscr{C}_{60}} \ldots \vdash_{\mathscr{C}_{60}} \pi_{m-1} \vdash_{\mathscr{C}_{60}} \pi_m$$

with $m \geqq 1$ and $\text{stm}(\pi_{m-1}, \pi_m) = (i, p'(\ldots))$. Then there is a unique index k with $0 \leqq k < m$ such that **proc** $p'(\ldots); \ldots;$ is defined in all programs $\pi_k, \ldots, \pi_m$, but not in π_{k-1}. π_k is called the *static* predecessor of π_m and $\pi_m \rightarrow \pi_k$ is the *static pointer* of π_m: we write $\pi_k = \text{static}(\pi_m)$. If $p = \text{ass}(\pi_m)$ and $q = \text{ass}(\pi_k)$ holds the program π_d has the following structure:

```
π_d = begin...proc q(...); begin
                              ...
                              proc p(...); ...;
                              ...
                           end; ...
      end
```

Thus procedure q lexicographically encloses p. Further on there is a greatest index l with $k \leqq l < m$ such that $\text{ass}(\pi_l) = \text{ass}(\text{static}(\pi_m))$. π_l is called the *"most recent" predecessor* of π_m and $\pi_m \rightarrow \pi_l$ is the *"most recent" pointer* of π_m: we write

$\pi_l = mr(\pi_m)$. Examples in [35] and [22] show that in general $k \neq l$, i.e. $\text{static}(\pi_m) \neq mr(\pi_m)$ holds: see Fig. 2.



($\vdash$ is a shorthand for $\vdash_{\mathscr{C}_{60}}$.)

Fig. 2

Now a program π has the *formal "most recent" property* if for every path $\not p$: $\pi_d \overset{*}{\vdash}_{\mathscr{C}_{60}} \pi' \vdash_{\mathscr{C}_{60}} \pi''$ the static and "most recent" predecessors of π'' coincide: $\text{static}(\pi'') = mr(\pi'')$. Due to Kandzia [22] L_{mr} is a decidable subset of L_{Algol}.

When applying Theorem 2 to L_{mr} we have to face the difficulty that programs in L_{mr} need not be $\mathscr{C}_{60}$-bounded. An example is

```
π = begin proc p(:f); begin
                          proc q(:r); begin f(:r) end; p(:q)
                      end;
      p(:p)
    end
```

where applications of the copy rule $\mathscr{C}_{60}$ lead to units with arbitrarily long reference chains due to the global formal procedure identifier f in procedure q. Notice, however, that q is "unessential", i.e. it is never called. We solve this difficulty by switching to the simpler copy rule $\mathscr{C}_{mr}$.

Lemma 8. *For programs π with the formal "most recent" property $\Sigma^{\mathscr{J}}_{\mathscr{C}_{60}}(\pi) = \Sigma^{\mathscr{J}}_{\mathscr{C}_{mr}}(\pi)$ holds.*

Proof. Let $\pi \in L_{mr}$. Then the following holds:

(1) For every path $\not p_k$: $\pi_d = \pi_1 \vdash_{\mathscr{C}_{60}} \dots \vdash_{\mathscr{C}_{60}} \pi_k$ there exists a corresponding path $\not p'_k$: $\pi_d = \pi'_1 \vdash_{\mathscr{C}_{mr}} \dots \vdash_{\mathscr{C}_{mr}} \pi'_k$ such that $\pi'_k = \text{ass}(\pi_k)$ holds, i.e. π'_k is obtained from π_k by replacing every procedure identifier p occurring at some position i in π_k by $\text{ass}(i, p)$.

(*Corresponding* means that for every $j \in \{1, \dots, k\}$ π_j and π'_j have the same structure and in π_j and π'_j the copy rule $\mathscr{C}_{60}$ resp. $\mathscr{C}_{mr}$ is applied at the same position.)

(1') Vice versa: For every path $\not p'_k$: ... there exists a corresponding path $\not p_k$: ... with $\pi'_k = \text{ass}(\pi_k)$.

Properties (1) and (1') are shown by induction on the length k of path $\not p_k$ resp. $\not p'_k$. Together with Lemma 3 they imply $\Sigma^{\mathscr{J}}_{\mathscr{C}_{60}}(\pi) = \Sigma^{\mathscr{J}}_{\mathscr{C}_{mr}}(\pi)$. □

Intuitively the lemma states that for programs $\pi \in L_{mr}$ no renaming of *procedure identifiers* is necessary when applying the copy rule once we have done the preprocessing step $\pi \to \pi_d$.

Corollary 8. *$\mathscr{H}(\mathscr{C}_{mr})$ is (sound and) relatively complete for $\Pi = (L_{mr}, \Sigma_{\mathscr{C}_{60}})$.*

Proof. By Lemma 8 it suffices to show that $\mathscr{H}(\mathscr{C}_{mr})$ is sound and relatively complete for $\Pi' = (L_{mr}, \Sigma_{\mathscr{C}_{mr}})$. But this is an immediate consequence of Theorem 1 and Corollary 6. □

Remark 2. Corollary 8 is independent of Corollary 7 since there are programs in $L_{mr} \setminus L_{gf}$ (e.g. π above) and others in $L_{gf} \setminus L_{mr}$ [22].

Let us now present some special applications of the last two corollaries: Define L_{pnes}, L_{par}, L_{pp} and L_{rp} to be the sets of all programs
- without *p*rocedure *nes*tings
- with *par*ameterless procedures only
- without *p*rocedure identifiers as *p*arameters
- in which formally *r*ecursive procedures are disallowed to have *p*rocedure identifiers as formal parameters

respectively. (A procedure **proc** $q(\dots); \dots;$ in a program π_d is called *formally recursive* if there exists a path

$$\wp\colon\ \pi_d \vdash\!\!\frac{*}{\mathscr{C}_{60}}\!\!\; \pi_1 \frac{+}{\mathscr{C}_{60}} \pi_2 \quad \text{with} \quad q = \operatorname{ass}(\pi_1) = \operatorname{ass}(\pi_2).)$$

By a theorem proved in [23] (see also [22], Theorem 1) we conclude that programs in L_{pnes}, L_{par} and L_{pp} have the so-called *strong* "most recent" property [22, 23] which implies the formal "most recent" property defined above. A closer investigation of the proof in [23] shows that also the programs in L_{rp} have the strong "most recent" property. Thus

$$(*) \qquad L_{pnes}, L_{par}, L_{pp}, L_{rp} \subseteq L_{mr}.$$

Moreover, L_{rp} is a decidable subset of L_{Algol}. This is true because it is decidable whether a procedure in a program with the strong "most recent" property is formally recursive [24] and whether a program has the strong "most recent" property [22].

By (*) $\mathscr{H}(\mathscr{C}_{mr})$ is (sound and) relatively complete for $\Pi_{pnes} = (L_{pnes}, \Sigma_{\mathscr{C}_{60}})$, $\Pi_{par} = (L_{par}, \Sigma_{\mathscr{C}_{60}})$, $\Pi_{pp} = (L_{pp}, \Sigma_{\mathscr{C}_{60}})$ and $\Pi_{rp} = (L_{rp}, \Sigma_{\mathscr{C}_{60}})$. By Corollary 7 also $\mathscr{H}(\mathscr{C}_{60})$ is relatively complete for Π_{pnes}, Π_{par} and Π_{pp}. Additionally, $\mathscr{H}(\mathscr{C}_{60})$ is relatively complete for Π_{rp} since it can be shown that every $\pi \in L_{rp}$ is $\mathscr{C}_{60}$-bounded. For Π_{par} even $\mathscr{H}(\mathscr{C}_n)$ is sound and relatively complete since $\Sigma^{\mathscr{I}}_{\mathscr{C}_{60}}(\pi) = \Sigma^{\mathscr{I}}_{\mathscr{C}_n}(\pi)$ holds for every $\pi \in L_{par}$. (I.e. apart from the preprocessing step $\pi \to \pi_d$ no further renaming of variables and procedure identifiers is needed here. Cf. Lemma 8.)

We conclude this section by comparing our results with other approaches dealing with static scope semantics. In [1] and [43] sound and relatively complete Hoare-like calculi are presented for the case of *parameterless* procedures. In [17] and ([6], Chapter 9) sound and relatively complete Hoare-like calculi for programs *without procedure nestings* and *without procedures as parameters* are given. Both Apt [1] and de Bakker [6] allow array variables as an additional language feature. The calculus in [17] can be used also to prove other properties than partial correctness, for example equivalence of programs.

Clarke [9] considers ALGOL-like languages $\Pi = ((L, \Sigma_{\mathscr{C}_{60}})$ with the general restrictions "no sharing" and "no self-application" for L. A program π has *no self-application* if all procedures in π have finite modes in the sense of ALGOL 68 [45]. In particular no procedure calls of the form $p(\dots p \dots)$ are allowed in such programs. Let L_{sa} denote the set of all programs without self-application ($L_{sa} \subseteq L_{\text{Algol}}$ is decidable.). Note that $L_{Pas} \subseteq L_{sa}$ holds since in PASCAL-like programs (cf. Sect. 9) every procedure has a finite mode of depth ≤ 2.

Clarke [9] claims without proof that relative completeness holds for this type of programming languages in the cases that additionally *procedure nestings* or *formally recursive procedures* or *procedure identifiers as parameters* or *global variables* are disallowed. The first three cases are covered by the completeness results on Π_{pnes}, Π_{pp} and Π_{rp}. The case "no global variables", however, is considerably more complicated and will be discussed later in Sect. 12.

11. The Characterization Theorem

In this section we investigate the problem whether the converse of the Completeness Theorem holds, i.e. whether formal provability in $\mathscr{H}(\mathscr{C})$ implies $\mathscr{C}$-boundedness. We show in the Characterization Theorem that this is true. So the calculus $\mathscr{H}(\mathscr{C})$ can deal *exactly* with $\mathscr{C}$-bounded programs. To prove this theorem we study the structure of the formal proofs ξ in $\mathscr{H}(\mathscr{C})$ with oracle $Th(\mathscr{I})$ more closely.

First we show that formal proofs can be transformed into standard proofs. By a *standard proof* we mean a formal proof in $\mathscr{H}(\mathscr{C})$, $Th(\mathscr{I})$ in which the recursion rule (8) is applied at most once. Recall that a standard proof was constructed in order to show Theorem 2. (A different method of constructing a formal proof ξ for a valid Hoare formula $\{P\}\pi\{Q\}$ can be found in Apt's completeness proof [1]: There the number of applications of the recursion rule in ξ depends on the structure of π and may be >1. By the following lemma this method is not farer reaching than the standard proof technique used in our paper.)

Lemma 9. *For every proof line $H_1 \to H_2$ which is formally provable in $\mathscr{H}(\mathscr{C})$, $Th(\mathscr{I})$ there exists also a standard proof.*

Proof. Let ξ be a formal proof of $H_1 \to H_2$ in $\mathscr{H}(\mathscr{C})$, $Th(\mathscr{I})$. By induction on the length of ξ it suffices to consider the case where rule (8) is applied *twice* in ξ. Using an obvious symbolic notation we may assume that ξ has the following structure:

$$\begin{array}{lll}
\xi_1\left\{\boxed{\begin{matrix}\dots\\ H\cup H_P\to H_B\end{matrix}}\right. & & \\
(*)\ H\to H_P & & \text{(rule (8))} \\
\xi_4\left\{\begin{array}{l}
\xi_2\left\{\boxed{\begin{matrix}\dots\\ H'\cup H'_P\to H'_B\end{matrix}}\right. \\
(**)\ H'\to H'_P \\
\xi_3\left\{\boxed{\begin{matrix}\dots\\ H_1\to H_2\end{matrix}}\right.
\end{array}\right. & & \begin{array}{l} \\ \text{(rule (8))} \\ \\ \end{array}
\end{array}$$

Further on we assume: (1) No proof line in ξ is superfluous for proving $H_1 \to H_2$, (2) in ξ_4 no proof rule is applied to a proof line in ξ_1, (3) in ξ_3 no proof rule is applied to a proof line in ξ_2 and (4) $H_P \cap H = \emptyset = H'_P \cap H'$.

By (2) and (3) the sequence ξ_3 is a formal proof of $H_1 \to H_2$ without application of rule (8), but with the assumptions (*) and (**). Let $\tilde{H} = (H \backslash H'_P) \cup H'$. Because of $\tilde{H} \subseteq H_1$ (by (1)) we can modify ξ_3 into a formal proof ξ'_3 of $H_1 \to H_2$ without application of rule (8), but with the assumption $\tilde{H} \to H_P \cup H'_P$ instead of (*) and (**). Now we can construct a formal proof ξ' of $H_1 \to H_2$ with just *one* application of rule (8) which has the following structure:

ξ_1		$\dots$	
	(o)	$H \cup H_P \to H_B$	
		$H \cup H_P \to H \cup H_P$	(rule (1))
	(+)	$H \cup H_P \to H_P$	(rule (12))
ξ'_2		$\dots$	
		$H' \cup H'_P \to H'_B$ *or*	(see comment below)
	(oo)	$H' \cup H_P \cup H'_P \to H'_B$	
		$H \cup H' \cup H_P \cup H'_P \to H_B \cup H'_B$	(rule (12) applied to (o) and (oo))
		$\tilde{H} \to H_P \cup H'_P$	(rule (8))
ξ'_3		$\dots$	
		$H_1 \to H_2$	

Comment: In ξ'_2 the same rules as in ξ_2 are applied, but all references to (*) in ξ_2 are replaced by references to (+). Thus the proof lines in ξ'_2 are those of ξ_2 with possibly H_P as additional set in their antecedents.
This proves the lemma. □

Next we show in Lemma 11 that a standard proof of a Hoare formula $\{P\}\pi\{Q\}$ contains already - up to substitutional equivalence - all information about the procedure call units generated by π. As auxiliary tools for this lemma we need Lemma 10 and Corollary 9.

Lemma 10. *Consider an arbitrary formal proof ξ of a proof line $H_1 \to \{\{P\}E|S\{Q\}\} \cup H_2$ where all atomic Hoare formulas $h \in H_1$ deal with procedure calls and S is no procedure call. Assume that $E|S \xrightarrow[\mathscr{C}]{} E''|S''$ holds. Then there exists a proof line $H'_1 \to \{\{P'\}E'|S'\{Q'\}\} \cup H'_2$ in ξ where $E'|S'$ and $E''|S''$ are substitutionally equivalent.*

Proof. Let the proof lines in ξ be marked by natural numbers. Then there is a least $j \in \mathbb{N}$ such that the j-th proof line in ξ is of the form $\tilde{H}_1 \to \{\{\tilde{P}\}\tilde{E}|\tilde{S}\{\tilde{Q}\}\} \cup \tilde{H}_2$ where $E|S \cong \tilde{E}|\tilde{S}$ and $\tilde{H}_1 \subseteq H_1$ hold. The proof line j cannot be an axiom. Hence it is the result of applying one of the proof rules to some of the preceeding proof lines. Since only the rules (5)-(7) have to be considered, there is a proof line i with $i<j$ of the form $H'_1 \to \{\{P'\}E'|S'\{Q'\}\} \cup H'_2$ where $\tilde{E}|\tilde{S} \xrightarrow[\mathscr{C}]{} E'|S'$ holds and $\xrightarrow[\mathscr{C}]{}$ is

applied at the same position as in $E|S \xrightarrow[\mathscr{C}]{} E''|S''$. Since $E|S \cong \tilde{E}|\tilde{S}$ holds, we conclude $E'|S' \cong E''|S''$ what was to be proved. □

Corollary 9. *Consider an arbitrary formal proof ξ of a proof line $H_1 \to \{\{P\}E|S\{Q\}\} \cup H_2$ where all atomic Hoare formulas $h \in H_1$ deal with procedure calls. Assume that $E|S \xrightarrow{\mathscr{P}} E''|S''$ holds. Then there exists a proof line $\{\{P'\}E'|S'\{Q'\}\} \cup H_1' \to H_2'$ in ξ with $E'|S' \cong E''|S''$.*

Proof. Note that the assertion of Lemma 10 remains true if S is an arbitrary statement and $E|S \xrightarrow[\mathscr{C}]{} E''|S''$ is replaced by $E|S \xrightarrow{\mathscr{P}} E''|S''$. The assertion of the corollary follows by observing that atomic Hoare formulas about correct procedure calls can – apart from substitutional equivalence – only be introduced in a formal proof by rule (1). □

Lemma 11. *Let ξ be a standard proof of $\{P\}\pi\{Q\}$ in which rule* (8) *is applied as follows:*

$$\begin{array}{l} \vdots \\ (k)\ \bigcup_{i=1}^{m} \{\{P_i\}E_i|p_i(\ldots)\{Q_i\}\} \to \bigcup_{i=1}^{m} \{\{P_i\}E_i|B_{i\mathscr{C}}\{Q_i\}\} \\ \vdots \\ (l)\ \bigcup_{i=1}^{m} \{\{P_i\}E_i|p_i(\ldots)\{Q_i\}\} \\ \vdots \end{array}$$

Then for every path $\not{p}$: $\emptyset|\pi_d \xrightarrow[\mathscr{C}]{} \ldots \xrightarrow[\mathscr{C}]{} E|q(\ldots)$ where $q(\ldots)$ is correct w.r.t. E there is an $i \in \{1, \ldots, m\}$ such that $E_i|p_i(\ldots) \cong E|q(\ldots)$ holds. In other words: π has a finite $\mathscr{C}$-index and $\{E_1|p_1(\ldots), \ldots, E_m|p_m(\ldots)\}$ is a system of representatives for the finitely many equivalence classes w.r.t. $\cong$ in $\mathscr{R}_{\mathscr{C}}(\pi)$.

Proof. We may assume that ξ has minimal length. The proof of the lemma is by induction on the *weight* w of $\not{p}$ which is the number of correct procedure call units occurring in $\not{p}$. In the case "$w = 1$" we have $\emptyset|\pi_d \xrightarrow{\mathscr{P}} E|q(\ldots)$. An application of Corollary 9 to the particular situation in ξ yields the assertion of the lemma. Let us now consider the induction step "$w \to w+1$". Then $\not{p}$ is of the form

$$\not{p}\colon\ \emptyset|\pi_d \xrightarrow[\mathscr{C}]{} \ldots \xrightarrow[\mathscr{C}]{} E'|q'(\ldots) \xrightarrow[\mathscr{C}]{} E'|B' \xrightarrow[\mathscr{C}]{} \ldots \xrightarrow[\mathscr{C}]{} E|q(\ldots)$$

where

$$\not{p}_w\colon\ \emptyset|\pi_d \xrightarrow[\mathscr{C}]{} \ldots \xrightarrow[\mathscr{C}]{} E'|q'(\ldots)$$

and

$$\not{p}_1\colon\ E'|B' \xrightarrow[\mathscr{C}]{} \ldots \xrightarrow[\mathscr{C}]{} E|q(\ldots)$$

are paths of weight w and 1 respectively. By induction hypothesis $E_j|p_j(\ldots) \cong E'|q'(\ldots)$ holds for some $j \in \{1, \ldots, m\}$.

Let $\tilde{\not{p}}_1$: $E_j|B_{j\mathscr{C}} \xrightarrow[\mathscr{C}]{} \ldots \xrightarrow[\mathscr{C}]{} E''|q''(\ldots)$ correspond to $\not{p}_1$. The argument employed in the case "$w = 1$" yields

$$E_i|p_i(\ldots) \cong E''|q''(\ldots) \qquad \text{for some } i \in \{1, \ldots, m\}.$$

The assertion of the lemma follows by observing that $E''|q''(\ldots) \cong E|q(\ldots)$ holds. □

We are now in a position to prove the main result of our paper: a complete characterization of the power of the calculi $\mathscr{H}(\mathscr{C})$.

Theorem 3 (Characterization Theorem). *Let FOL be expressive w.r.t. $\mathscr{I}$ and $\mathscr{C}$. Then the following assertions are equivalent:*

(1) *$\{P\}\pi\{Q\}$ is formally provable in $\mathscr{H}(\mathscr{C})$, $Th(\mathscr{I})$.*
(2) *π has a finite $\mathscr{C}$-index and $\{P\}\pi\{Q\}$ is valid w.r.t. $\mathscr{I}$ and $\mathscr{C}$.*
(3) *π is $\mathscr{C}$-bounded and $\{P\}\pi\{Q\}$ is valid w.r.t. $\mathscr{I}$ and $\mathscr{C}$.*

Proof. "(1)→(2)": The validity of $\{P\}\pi\{Q\}$ follows from Corollary 3 of Theorem 1. Lemma 9 guarantees the existence of a standard proof for $\{P\}\pi\{Q\}$. Now Lemma 11 states that π has a finite $\mathscr{C}$-index. "(2)→(3)": By Lemma 7. "(3)→(1)": By Theorem 2. □

12. Concluding Remarks

In this section we discuss two issues related to the Characterization Theorem and indicate further directions for research. The first issue concerns the notion of Hoare logic and the second one addresses Clarke's completeness claim for the case "no global variables" (see Sect. 10).

Hoare Logic. The Characterization Theorem leads to the following remark:

For a given program $\pi \in L_{\text{Algol}}$ *all* valid Hoare formulas $\{P\}\pi\{Q\}$ are provable in $\mathscr{H}(\mathscr{C})$, $Th(\mathscr{I})$ provided $\mathscr{H}(\mathscr{C})$, $Th(\mathscr{I})$ is able to prove *just one* Hoare formula $\{P_0\}\pi\{Q_0\}$.

Thus $\mathscr{H}(\mathscr{C})$ does not care about the particular situation of $\{P\}\pi\{Q\}$, but represents a general mechanism designed to deal with program schemas rather than with specific programs. In this light the notion "formal proof" has a double meaning: On the one hand "formal" refers to the strict rules of the calculus and on the other hand "formal" means that such a proof is independent of the actually arising computations.

This suggests that a notion of Hoare logic should be much more than just a (relative to $Th(\mathscr{I})$) recursively enumerable set of Hoare formulas $\{P\}\pi\{Q\}$ (Lipton's definition [33, 30]). It should also be considered as a sort of acceptor for structures of formal computations generated by programs. In our future work we intend to explore the relationships between program classes, structures of formal computations and basic proof principles needed to understand these structures.

"No Global Variables". Let L_{sh}, L_{sa} and L_{gv} denote the sets of programs without sharing, self-application and global variables respectively. Then it was Clarke's claim [9] that there exists a sound and relatively complete Hoare-like calculus for $\Pi_{Cl}=(L_{sh}\cap L_{sa}\cap L_{gv}, \Sigma_{\mathscr{C}_{60}})$. (We remark that $L_{sh}\cap L_{sa}\cap L_{gv}$ is a decidable subset of L_{Algol} though $L_{sh}\subseteq L_{\text{Algol}}$ itself is undecidable for $\mathscr{C}_{60}$). The difficulty of this claim depends on how we interpret the notion "global variable".

The interpretation we had in mind when reading Clarke's paper was that a global variable of a program π is a *variable* x which occurs freely in some extended procedure body of π. In particular nothing is said about procedure identifiers p in π. For a better understanding of the consequences of this interpretation let us look at the following example.

```
$\pi_0$ = begin
        var count;
        proc $p(x_1, x_2 : f)$;
           begin
              proc $q(y_1, y_2 :)$; begin $y_1 := y_1 + 1; f(y_1, y_2 :)$ end;
              $x_1 := x_1 + 1$; if $x_1 \leqq x_2$ then $p(x_1, x_2 : q)$ else $f(x_1, x_2 :)$ fi
           end;
        proc $r(z_1, z_2 :)$; begin $z_2 := z_1$ end;
        count := 0; p(count, out : r)
     end
```

Of course $\pi_0 \in L_{sh} \cap L_{sa} \cap L_{gv}$ holds because the global formal procedure identifier f does not matter. (Note that π_0 os in fact a PASCAL-like program.) If the copy rule $\mathscr{C}_{60}$ is applied π_0 terminates and the Hoare formula $\{\text{out} = \text{in}\}\, \pi_0 \,\{\text{out} = 2 * \text{in} + 1\}$ is valid w.r.t. the standard interpretation $\mathscr{N}$ of natural numbers and $\mathscr{C}_{60}$. On the other hand, both $\mathscr{C}_{mr}$ and $\mathscr{C}_n$ lead to non-termination of π_0. Moreover, π_0 is constructed in such a way that for all copy rules $\mathscr{C}$ for which π_0 is $\mathscr{C}$-bounded the Hoare formula $\{\textbf{true}\}\, \pi_0 \,\{\textbf{false}\}$ is valid w.r.t. $\mathscr{N}$ and $\mathscr{C}$, i.e. π_0 never terminates. This leads to the following corollary of Theorem 3.

Corollary 11. *There is no copy rule $\mathscr{C}$ such that $\mathscr{H}(\mathscr{C})$ is sound and relatively complete for $\Pi_{Cl} = (L_{sh} \cap L_{sa} \cap L_{gv}, \Sigma_{\mathscr{C}_{60}})$.*

Proof. Assume that $\mathscr{H}(\mathscr{C})$ is relatively complete for Π_{Cl}. Choose the interpretation $\mathscr{I} = \mathscr{N}$. Since FOL is expressive w.r.t. $\mathscr{N}$ and $\mathscr{C}_{60}$ [10], Theorem 3 implies that every program π in $L_{sh} \cap L_{sa} \cap L_{gv}$ is $\mathscr{C}$-bounded. Considering the sample program π_0 we conclude that $\mathscr{H}(\mathscr{C})$ cannot be sound for Π_{Cl}. $\square$

Clarke's original idea (private communication) to verify the claim on language Π_{Cl} was to eliminate procedure nestings by a transformation

$$T\colon\ L_{sh} \cap L_{sa} \cap L_{gv} \to L_{pnes}$$

such that π and $T(\pi)$ are *schematically equivalent*, i.e. $\Sigma^{\mathscr{I}}_{\mathscr{C}_{60}}(\pi) = \Sigma^{\mathscr{I}}_{\mathscr{C}_{60}}(T(\pi))$ holds for every interpretation $\mathscr{I}$. Then by the completeness result on $\Pi_{p_{nes}}$ the calculus $\mathscr{H}(\mathscr{C}_{60})$, augmented with T as an additional proof rule, would be sound and relatively complete for Π_{Cl}. But it is rather doubtful whether such a transformation exists in light of the following fact (which can be proved by methods developed in [25] resp. [27]):

There is no program π in $L_{p_{nes}}$ which is *formally equivalent* [27] to the program π_0 above.

(Formal equivalence is a stronger property than schematical equivalence). Together with Corollary 11 this fact leads to the following conclusion (see [31]):

Presently known Hoare-style proof techniques for ALGOL-like programs - which we think are essentially represented in the calculi $\mathscr{H}(\mathscr{C})$ - are not sophisticated enough to deal with the language Π_{Cl}.

All these difficulties do not arise if we use the interpretation Clarke had in mind (private communication) when writing "no global variables", namely that - in our terminology - both global procedure identifiers and global variables are disallowed. The $L_{sh} \cap L_{sa} \cap L_{gv} \subseteq L_{gf}$ holds. Now there exists a transformation

T: $L_{gf} \rightarrow L_{pnes}$ eliminating procedure nestings [27]. Independent of this transformation $\mathscr{H}(\mathscr{C}_{60})$ is now sound and relatively complete for Π_{Cl} due to Corollary 7.

Let us summarize these observations as follows: Interpreted in the strict sense Clarke's claim leads to a challenging problem, namely to find a Hoare-like proof system for Π_{Cl}. At present only first steps towards a solution are known. In particular [29] has shown that for finite interpretations $\mathscr{I}$ it is uniformly decidable whether a program $\pi \in L_{sa} \cap L_{gv}$ is divergent w.r.t. $\Sigma^{\mathscr{I}}_{\mathscr{C}_{60}}$. (Note that the sharing restriction is not needed here.) According to a theorem by Lipton [33, 30] this result implies the existence of a sound and relatively complete Hoare logic for $\Pi = (L_{sa} \cap L_{gv}, \Sigma_{\mathscr{C}_{60}})$. Of course from Lipton's notion of Hoare logic (see above) we don't get any information how a natural syntax-directed Hoare-like system would look like, but it can be understood as a first step in this direction. Hence the following

Conjecture. [31] There is a sound and relatively complete Hoare-like system $\mathscr{H}$ for $\Pi = (L_{sa} \cap L_{gv}, \Sigma_{\mathscr{C}_{60}})$.

We hope that the decidability proof in [29] will give some hints to attack this problem.

Appendix: Diagram of Languages

The following languages have been discussed in connection with static scope semantics $\Sigma_{\mathscr{C}_{60}}$.

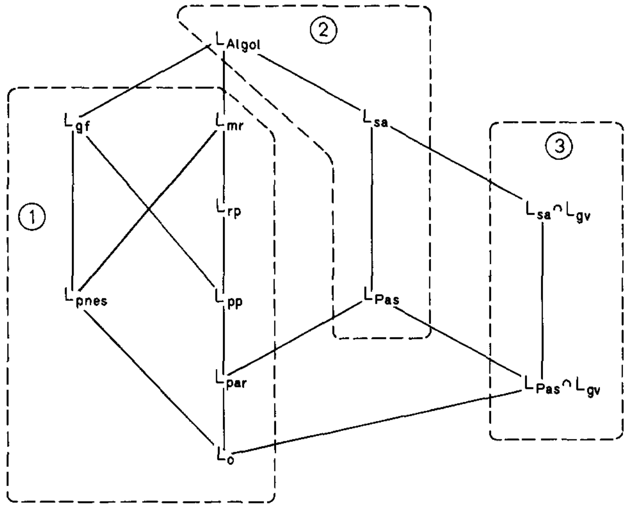


Diagram 1

(L_0 denotes the set of all programs without procedures.)

① $\mathscr{H}(\mathscr{C}_{60})$ resp. $\mathscr{H}(\mathscr{C}_{mr})$ sound and relatively complete for these languages.

② *No* sound and relatively complete Hoare-like calculus [9].

③ Only sound and relatively complete Hoare *logic* in the sense of Lipton, but a concrete calculus is not yet known.

Acknowledgements. I am indepted to Professor H. Langmaack for introducing me to the area of programming languages and for many stimulating discussions on the subject of this paper. Also I would like to thank K.R. Apt for his comments on the first version of this paper which helped to improve the presentation in various places.

References

1. Apt, K.R.: A sound and complete Hoare-like system for a fragment of PASCAL. Report IW 96/78. Mathematisch Centrum, 1978
2. Apt, K.R.: Ten years of Hoare's logic, a survey, part I. Tech. Report, Faculty of Economics, Erasmus Univ., Rotterdam, April 1979 ACM TOPLAS (in press, 1981)
3. Apt, K.R., de Bakker, J.W.: Semantics and proof theory of PASCAL procedures. In: Proc. 4th Coll. Automata, Languages and Programming (A. Salomaa, M. Steinby, eds.). Lecture Notes in Computer Science Vol. 52, pp. 30–44. Berlin-Heidelberg-New York: Springer 1977
4. Backus, J.W., Bauer, F.L., Green, J., Katz, C., McCarthy, J., Naur, P., Perlis, A.J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J.H., van Wijngaarden, A., Woodger, M.: Revised report on the algorithmic language ALGOL 60 (P. Naur, ed.) Numer. Math. **4**, 420–453 (1963)
5. de Bakker, J.W.: Semantics and the foundation of program proving. In: Proc. IFIP Congress 1977, Toronto, pp. 279–284. Amsterdam: North Holland 1977
6. de Bakker, J.W.: Mathematical theory of programm correctness. London: Prentice Hall International 1980
7. Bergstra, J.A., Tucker, J.V.: Some natural structures which fail to possess a sound and decidable Hoare-like logic for their while-programs. Theor. Comput. Sci. (in press, 1981)
8. Cartwright, R., Oppen, D.: Unrestricted procedure calls in Hoare's logic. In: Proc. of the Fith Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, pp. 131–140. New York: ACM 1978
9. Clarke, E.M.: Programming language constructs for which it is impossible to obtain good Hoare-like axioms. J. Assoc. Comput. Mach. **26**, 129–147 (1979)
10. Cook, S.A.: Soundness and completeness of an axiom system for program verification. SIAM J. Comput. **7**, 70–90 (1978)
11. Dijkstra, E.W.: Recursive programming. Numer. Math. **2**, 312–318 (1960)
12. Donahue, J.E.: Complementary definitions of programming language semantics. Lecture Notes in Computer Science Vol. 42. Berlin-Heidelberg-New York: Springer 1976
13. Gorelick, G.A.: A complete axiomatic system for proving assertions about recursive and non-recursive programs. Tech. Rep. 75, Dept. of Computer Sci., Univ. of Toronto (1975)
14. Grau, A.A., Hill, U., Langmaack, H.: Translation of ALGOL 60. Handbook for Automatic Computation, Vol. I, Part b. (Grundlehren der mathematischen Wissenschaften, Band 137) (K. Samelson, ed.) Berlin-Heidelberg-New York: Springer 1967
15. Gries, D., Levin, G.: Assignment and procedure call proof rules. ACM TOPLAS **2**, 564–579 (1980)
16. Guttag, J.V., Horning, J.J., London, R.L.: A proof rule for EUCLID procedures. In: Formal description of Programming Concepts (E.J. Neuhold, ed.), pp. 211–220. Amsterdam: North Holland 1978
17. Harel, D., Pnueli, A., Stavi, J.: A complete axiomatic system for proving deductions about recursive programs. In: Proc. of the 9th ACM Symposium on Theory of Computing, pp. 249–260. New York: ACM 1977
18. Hoare, C.A.R.: An axiomatic basis for computer programming. Comm. ACM **12**, 576–580, 583 (1969)
19. Hoare, C.A.R.: Procedures and parameters: An axiomatic approach. In: Symposium on semantics of algorithmic languages (E. Engeler, ed.). Lecture Notes in Mathematics 188, pp. 102–116. Berlin-Heidelberg-New York: Springer 1971
20. Igarashi, S., London, R.L., Luckham, D.C.: Automatic program verfication I: A logical basis and its implementation. Acta Informat. **4**, 145–182 (1975)
21. Jensen, K., Wirth, N.: PASCAL user manual and report. Berlin-Heidelberg-New York: Springer 1975
22. Kandzia, P.: On the "most recent" property of ALGOL-like programs. In: Proc. 2nd Coll. Automata, Languages and Programming (J. Loeckx, ed.). Lecture Notes in Computer Science Vol. 14, pp. 97–111. Berlin-Heidelberg-New York: Springer 1974

23. Kandzia, P., Langmaack, H.: On a theorem of McGowan concerning the "most recent" property of programs. Bericht Nr. A 74/07 des Fachbereichs Angew. Math. und Inform., Univ. d. Saarlandes (1974)
24. Klein, H.-J.: Untersuchungen im Zusammenhang mit der "most recent" Eigenschaft von Pro-Programmen. Bericht Nr. 8/77 des Inst. f. Inform. u. Prakt. Math., Christian-Albrechts-Univ., Kiel 1977
25. Klein, H.-J.: Zur Charakterisierung von Programmen mit endlichen Arten. Dissertation, Christian-Albrechts-Univ., Kiel 1980
26. Langmaack, H.: On correct procedure parameter transmission in higher programming languages. Acta Informat. **2**, 110–142 (1973)
27. Langmaack, H.: On procedures as open subroutines I, II. Acta Informat. **2**, 311–333 (1973) and Acta Informat. **3**, 227–241 (1974)
28. Langmaack, H.: On a theory of decision problems in programming languages. In: Proc. of the Int. Conf. on Math. Studies of Inform. Processing, Kyoto (E.K. Blum, S. Takasu, eds.). Lecture Notes in Computer Science Vol. 75, pp. 538–558. Berlin-Heidelberg-New York: Springer 1969
29. Langmaack, H.: On termination problems for finitely interpreted ALGOL-like programs. Bericht Nr. 7904 des Inst. f. Inform. u. Prakt. Math., Christian-Albrechts-Univ., Kiel, 1979
30. Langmaack, H.: A proof of a theorem of Lipton on Hoare logic and applications. Bericht Nr. 8003 des Inst. f. Inform. u. Prakt. Math., Christian-Albrechts-Univ., Kiel, 1980
31. Langmaack, H., Olderog, E.-R.: Present-day Hoare-like systems for programming languages with procedures: power, limits and most likely extensions. In: Proc. 7th Coll. Automata, Languages and Programming (J.W. de Bakker, J. van Leeuwen, eds.). Lecture Notes in Computer Science Vol. 85, pp. 363–373. Berlin-Heidelberg-New York: Springer 1980
32. Lauer, P.E.: Consistent formal theories of the semantics of programming languages. Tech. Rep. TR 25.121, IBM Laboratory, Vienna 1971
33. Lipton, R.J.: A necessary and sufficient condition for the existence of Hoare logics. In: 18th IEEE Symposium on Foundations of Computer Science, Providence, Rhode Island, pp. 1–6. New York: IEEE 1977
34. London, R.L., Guttag, J.V., Horning, J.J., Lampson, B.W., Mitchell, J.G., Popek, G.J.: Proof rules for the programming language EUCLID. Acta Informat. **10**, 1–26 (1978)
35. McGowan, C.L.: The "most recent" error: its causes and correction. SIGPLAN Notices **7**, 191–202 (1972)
36. O'Donnell, M.J.: A critique of the foundations of Hoare-style programming logic. Tech. Report, Computer Sci. Dept., Purdue Univ., West-Lafayette, 1980
37. Olderog, E.-R.: Korrektheits- und Vollständigkeitsaussagen über Hoaresche Ableitungskalküle. Diplomarbeit, Christian-Albrechts-Univ., Kiel 1979
38. Prawitz, D.: Natural deduction – a proof-theoretic study. Stockholm: Almqvist and Wiksell 1965
39. Randell, B., Russell, L.J.: ALGOL 60 implementation. London: Academic Press, 1964
40. Schwartz, R.L.: An axiomatic definition of ALGOL 68. Comp. Sci. Dept., UCLA-34P214-75, Univ. of California, Los Angeles, 1978
41. Schwartz, R.L.: On axiomatizability as a language design tool. Dept. of Applied Maths., Weizmann Inst. of Sci., Rehovot, January 1979
42. Schwarz, J.: Generic commands – a tool for partial correctness formalism. Comput. J. **20**, 151–155 (1977)
43. Sieber, K.: A new Hoare-calculus for programs with recursive parameterless procedures. Bericht A 81/02 des Fachbereichs Angew. Math. u. Inform., Univ. d. Saarlandes, Saarbrücken 1981
44. Wand, M.: A new incompleteness result for Hoare's system. J. Assoc. Comput. Mach. **25**, 168–175 (1978)
45. van Wijngaarden, A., Mailloux, B.J., Peck, J.E.L., Koster, C.H.A., Sintzoff, M., Lindsey, C.H., Meertens, L.G.L.T., Fisker, R.G. (eds.): Revised report on the algorithmic language ALGOL 68. Acta Informat. **5**, 1–236 (1975)

Received November 5, 1979 / June 21, 1981