Idealized Algol and its Specification Logic[1]

John C. Reynolds
School of Computer and Information Science
Syracuse University
Syracuse, New York

1. Introduction[2]

Specification logic is a new formal system for program proving that is applicable to programming languages, such as Algol, whose procedure mechanism can be described by the copy rule. The starting point of its development is the recognition that, in the presence of an Algol-like procedure mechanism, *specifications*, such as the Hoare triple $\{P\}\ S\ \{Q\}$ [1], must be regarded as predicates about environments (in the sense of Landin [2, 3]). The logic provides additional kinds of specifications describing an interference relation (#) between variables and other entities, and permits specifications to be compounded using the operations of implication ($\Rightarrow$), conjunction (&), and universal quantification ($\forall$). The result is a system in which one can infer *universal* specifications, i.e., specifications that hold in all environments.

To see why the introduction of procedures makes a specification depend upon an environment (which is a mapping from the free identifiers of the specification into appropriate meanings), it is enough to consider any nontrivial Hoare triple involving a procedure call. For example, the truth of

$$\{x = 3\}x := double(x)\{x = 6\}$$

clearly depends upon the meaning into which *double* is mapped by the relevant environment.

More surprisingly, since procedures introduce the phenomenon of interference (of which the simplest case is aliasing between variables), even specifications that do not refer to procedures overtly can still depend upon the environment. For example,

$$\{x = 3\}y := 4\{x = 3\}$$

(which is provable from Hoare's axiom of assignment and is universally true in the absence of a procedure mechanism) depends upon whether x and y are aliases, i.e., whether they are

[1]Work supported by National Science Foundation Grant MCS-8017577 and U.S. Army Contract DAAK80-80-C-0529. First appeared in D. Néel, editor, *Tools and Notions for Program Construction*, pages 121–161, Cambridge University Press 1982. Reprinted in *Algol-like Languages*, ed. P. W. O'Hearn and R. D. Tennent, vol. 1, pp. 125–156, Birkhäuser, 1997.

[2]The first part of this section was extended by the author in 1996.

mapped by the environment into the same storage location, so that assignment to one will change the value of the other.

In fact, to obtain universal specifications about commands in Algol-like languages, it is necessary to extend the language of specifications well beyond Hoare triples. A simple example of a universal specification is

$$y \# x \Rightarrow \{x = 3\} y := 4 \{x = 3\} \ ,$$

which specifies that, if the meaning of y does not *interfere* with the meaning of x, then assigning 4 to y will preserve the assertion $x = 3$. Another example is

$$\{(\forall n)\ \mathit{double}(n) = n + n\} \ \&\ \mathbf{gv}_{\mathrm{int}}(x) \Rightarrow \{x = 3\} x := \mathit{double}(x) \{x = 6\} \ ,$$

which specifies that, if the meaning of *double* is a function procedure that doubles any integer, and if the meaning of x is a "good variable" (a technical condition described in Section 7), then executing $x := \mathit{double}(x)$ in a state where $x = 3$ will produce a state where $x = 6$.

It is important to distinguish specifications from assertions, which are the first and last members of Hoare triples (i.e., the phrases enclosed in braces). While the truth of a specification depends only upon an environment, which maps identifiers into their meanings, the truth of an assertion depends upon both an environment and a storage state, which maps locations into values.

Although proof methods for procedures have been the subject of considerable research [4, 5, 6, 7], this work has focused on call by reference and call by value, and had led to extremely complex inference rules that are incapable of dealing with interference, call by name, statement parameters, or higher-order procedures. In contrast, specification logic is both simpler and more general. It assumes that the procedure mechanism is based upon a typed lambda calculus that encompasses the logic for program proving as well as the programming language.

In [8], a version of specification logic is given for the sublanguage of Algol W [9] that represents a refinement of Algol 60. In this report, we develop specification logic for a subset of the language described in [10], which is an idealization of Algol that makes the underlying (typed) lambda calculus explicit.

The advantage is that this language desugars well, i.e., many of its constructs can be defined as abbreviations in terms of a simpler "base" language. As a consequence, one can begin with simple inference rules for the base language and then derive more complex rules for the abbreviationally defined constructs. In particular, the complex aspects of inference that arise from binding can be encapsulated in the process of beta-reduction, which is applicable to specifications as well as programs.

In this report we will avoid formulating the semantics of specifications. Although a semiformal semantics is given in [8], there is some doubt that it is the "right" semantics. As we will see in Section 11, there are useful and apparently noncontradictory rules of inference that are not valid in this semantics.

It should be emphasized that this is a preliminary report of ongoing research. Both the programming language and logic described in it can be expected to change considerably in the future.

2. Syntax

The programming language we consider is basically a sublanguage of that described in [10], with the following restrictions:

(1) Recursive definition is permitted for commands, acceptors, and proper procedures, but not for expressions or function procedures.

This limitation is an inevitable (though regrettable) consequence of using the predicate calculus for assertions, along with the vital assumption that any expression of the programming language can occur as a term in such assertions. Recursive definition of expressions or function procedures would impose a domain structure upon their meanings that would be incompatible with the free use of quantifiers, because of their lack of continuity. In other words, we cannot deal with nonterminating expressions and function procedures because we cannot deal with nonterminating assertions.

Since nontermination is meaningless for certain phrase types, we must also abandon the phrase type **univ**(ersal).

(2) Multiple declarations (including mutual recursion), case statements, arrays, and labels are not treated.

These topics are simply omitted for brevity; none of them appear to raise fundamental difficulties in specification logic. (Arrays, labels, and also **for** statements are treated in [8].)

(3) Products and sums of data types are not treated.

Here our experience is too limited to say much; we have yet to explore the treatment of these topics in specification logic.

The following productions describe the various kinds of type we will use:

$$\langle \text{data type} \rangle ::= \textbf{integer} \mid \textbf{real} \mid \textbf{Boolean}$$

$$\begin{aligned}\langle \text{ordinary phrase type} \rangle ::= &\ \langle \text{data type} \rangle\ \textbf{exp} \\ &\mid \langle \text{data type} \rangle\ \textbf{acc} \\ &\mid \langle \text{data type} \rangle\ \langle \text{data type} \rangle\ \textbf{var} \mid \textbf{comm} \\ &\mid \langle \text{ordinary phrase type} \rangle \to \langle \text{ordinary phrase type} \rangle\end{aligned}$$

$\langle$recursive phrase type$\rangle$ ::= $\langle$data type$\rangle$ **acc** | **comm**
| $\langle$ordinary phrase type$\rangle \rightarrow \langle$recursive phrase type$\rangle$

$\langle$bindable phrase type$\rangle$ ::= $\langle$ordinary phrase type$\rangle$
| **commlike** | **explike**

$\langle$phrase type$\rangle$ ::= $\langle$bindable phrase type$\rangle$ | **spec**

A data type denotes a set of values appropriate for some kind of variable or expression, while a phrase type denotes a set of meanings appropriate for some kind of phrase. Among phrase types in general, bindable phrase types are those that can be denoted by identifiers, ordinary phrase types are those that can be denoted by identifiers that are bound by lambda expressions, and recursive phrase types are those for which recursive definition is permitted.

(Notice that variables are classified separately by the data types they accept and produce, i.e., $\tau_1\tau_2$ **var** means τ_1-accepting τ_2-producing variable. The rationale for this complication is given in [10].)

Throughout this paper we will use the following metavariables to range over various kinds of type:

τ: data types

ρ: recursive phrase types

θ: ordinary phrase types

ω: bindable phrase types

γ: phrase types.

For both data and phrase types, there is a partial ordering called the *subtype relation.* For data types, it is the least partial ordering such that **integer** $\leq$ **real**. For phrase types, it is the least partial ordering such that:

$\tau \leq \tau'$ implies τ **exp** $\leq \tau'$ **exp**

$\tau' \leq \tau$ implies τ **acc** $\leq \tau'$ **acc**

$\tau_1' \leq \tau_1$ and $\tau_2 \leq \tau_2'$ implies $\tau_1\tau_2$ **var** $\leq \tau_1'\tau_2'$ **var**

$\tau_1\tau_2$ **var** $\leq \tau_1$ **acc**

$\tau_1\tau_2$ **var** $\leq \tau_2$ **exp**

$\theta_1' \leq \theta_1$ and $\theta_2 \leq \theta_2'$ implies $\theta_1 \to \theta_2 \leq \theta_1' \to \theta_2'$

τ **acc** $\leq$ **commlike**

comm $\leq$ **commlike**

$\theta_2 \leq$ **commlike** implies $\theta_1 \to \theta_2 \leq$ **commlike**

τ **exp** $\leq$ **explike**

$\theta_2 \leq$ **explike** implies $\theta_1 \to \theta_2 \leq$ **explike**.

When $\gamma \leq \gamma'$ we say that γ is a *subtype* of γ'; the intent is that there is an implicit conversion from the meanings for γ to the meanings for γ', so that any phrase of type γ can be used in any context requiring a phrase of type γ'.

A *type assignment* is a function from a finite set of identifiers to bindable phrase types. Throughout this paper, we use the metavariable ι to range over identifiers and the metavariable π to range over type assignments.

The subtype relation is extended to type assignments as follows:

$$\pi \leq \pi' \text{ if and only if } \mathbf{dom}(\pi') \subseteq \mathbf{dom}(\pi) \text{ and } (\forall \iota \in \mathbf{dom}(\pi'))\ \pi(\iota) \leq \pi'(\iota)$$

where $\mathbf{dom}(\pi)$ denotes the domain of the function π.

To describe the syntax of both our programming language and the language of specifications, we will use phrase class names of the form $\langle \gamma, \pi \rangle$, where γ is a phrase type and π is a type assignment. The intent is that $\langle \gamma, \pi \rangle$ denotes the set of phrases P such that

(1) The identifiers occurring free in P belong to the domain of π.

(2) When its free identifiers are given the bindable phrase types indicated by π, P has phrase type γ.

We will only describe abstract syntax, ignoring considerations of parenthesization and precedence, and using production schemas containing the metavariables $\tau, \rho, \theta, \omega, \gamma, \iota$, and π (with occasional sub- and superscripts) whose ranges have been described above. First, we describe the "generic" part of the language, i.e., constructs that are applicable to a variety of phrase types:

$\langle \gamma', \pi \rangle$ ::= $\langle \gamma, \pi \rangle$ when $\gamma \leq \gamma'$

$\langle \omega, \pi \rangle$ ::= ι when $\iota \in \mathbf{dom}(\pi)$ and $\pi(\iota) = \omega$

$\langle \theta_2, \pi \rangle$ ::= $\langle \theta_1 \to \theta_2, \pi \rangle$ $(\langle \theta_1, \pi \rangle)$

$\langle \theta_1 \to \theta_2, \pi \rangle$ ::= $\lambda \iota{:}\, \theta_1 .\, \langle \theta_2, [\pi \mid \iota{:}\, \theta_1] \rangle$

$$\langle \rho, \pi \rangle \ ::= \mathbf{rec}\ \langle \rho \to \rho, \pi \rangle$$

$$\langle \rho, \pi \rangle \ ::= \perp_\rho$$

$$\langle \theta, \pi \rangle \ ::= \mathbf{if}\ \langle \text{Boolean exp},\ \pi \rangle\ \mathbf{then}\ \langle \theta, \pi \rangle\ \mathbf{else}\ \langle \theta, \pi \rangle$$

$$\langle \theta_2, \pi \rangle \ ::= \mathbf{let}\ \iota\ \mathbf{be}\ \langle \theta_1, \pi \rangle\ \mathbf{in}\ \langle \theta_2, [\pi \mid \iota{:}\, \theta_1] \rangle$$

$$\langle \theta, \pi \rangle \ ::= \mathbf{letrec}\ \iota{:}\, \rho\ \mathbf{be}\ \langle \rho, [\pi \mid \iota{:}\, \rho] \rangle\ \mathbf{in}\ \langle \theta, [\pi \mid \iota{:}\, \rho] \rangle$$

The first schema shows the purpose of the subtype relation. The second shows that an identifier assigned some bindable phrase type can be used as a phrase of that type.

The next two schemas describe procedure application (i.e., call) and lambda abstraction. We limit ourselves to procedures with a single parameter, since multiple parameters can be eliminated by Currying [10].

In the fourth schema $[\pi \mid \iota{:}\, \theta_1]$ denotes the type assignment such that

$$\mathbf{dom}([\pi \mid \iota{:}\, \theta_1]) = \mathbf{dom}(\pi) \cup \{\iota\}$$

$$[\pi \mid \iota{:}\, \theta_1]\,(\iota) = \theta_1$$

$$[\pi \mid \iota{:}\, \theta_1]\,(\iota') = \pi(\iota') \text{ when } \iota' \neq \iota.$$

Later we will use the following extensions of this notation:

$$[\pi \mid \iota_1{:}\, \theta_1 \mid \cdots \mid \iota_n{:}\, \theta_n] = [\cdots [\pi \mid \iota_1{:}\, \theta_1] \cdots \mid \iota_n{:}\, \theta_n]$$

$$[\iota_1{:}\, \theta_1 \mid \cdots \mid \iota_n{:}\, \theta_n] = [[\ ] \mid \iota_1{:}\, \theta_1 \mid \cdots \mid \iota_n{:}\, \theta_n],$$

where [ ] is the type assignment with empty domain.

The fifth and sixth schemas describe the least-fixed-point operator **rec** and the least-element constant $\perp$, both of which are limited to recursive phrase types.

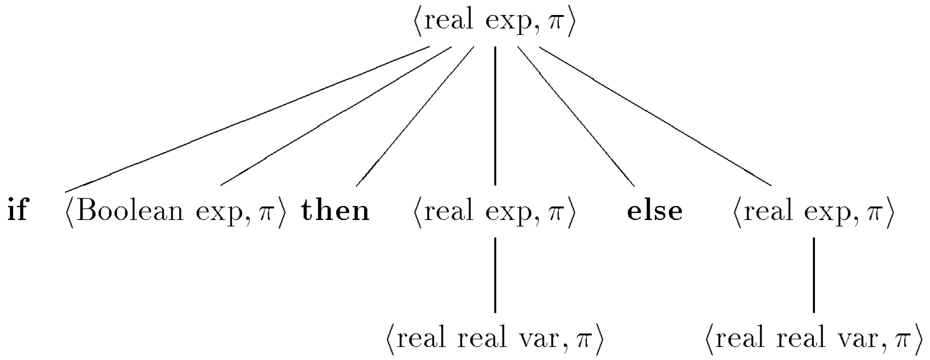
The seventh schema describes a generic conditional construction that is applicable to all ordinary phrase types.

The last two schemas introduce Landin's notation [2, 3] for nonrecursive and recursive declarations (limited to recursive phrase types in the latter case). For simplicity, we avoid the multiple declarations considered in [10].

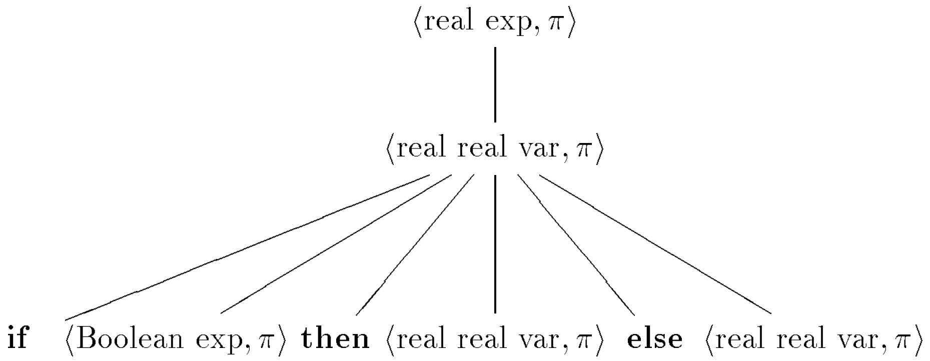
Notice that, quite aside from our neglect of parenthesization and precedence, this syntax is ambiguous with regard to phrase types. For example,

if $\langle$Boolean exp, $\pi\rangle$ **then** $\langle$real real var, $\pi\rangle$ **else** $\langle$real real var, $\pi\rangle$

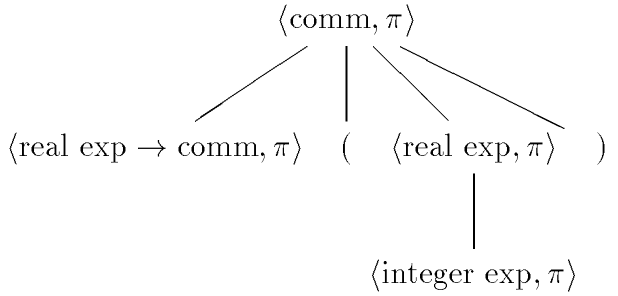
can be derived from $\langle$real exp, $\pi\rangle$ by either



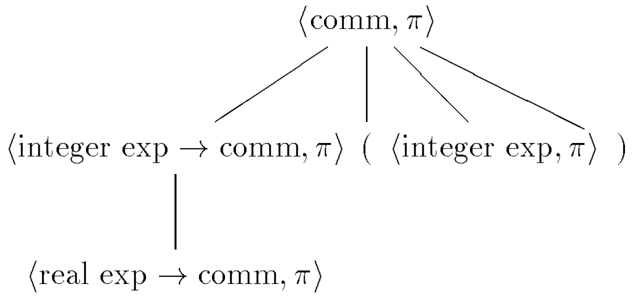
or



Similarly, $\langle$real exp $\to$ comm, $\pi\rangle(\langle$integer exp, $\pi\rangle)$ can be derived from $\langle$comm, $\pi\rangle$ by either



or



We assume that this kind of ambiguity does not lead to ambiguous meaning.

Next we give more specific production schemas describing expressions and commands. The schemas for expressions are obviously incomplete; the inclusion of additional constants and operations would be straightforward though uninteresting.

$$\langle \text{integer exp}, \pi\rangle \ ::= 0 \mid 1 \mid \langle \text{integer exp}, \pi\rangle + \langle \text{integer exp}, \pi\rangle$$

$$\langle \text{real exp}, \pi\rangle \ ::= 0.5 \mid \langle \text{real exp}, \pi\rangle + \langle \text{real exp}, \pi\rangle$$

$$\begin{aligned}\langle \text{Boolean exp}, \pi\rangle \ ::= {} & \mathbf{true} \mid \mathbf{false} \mid \langle \tau \text{ exp}, \pi\rangle = \langle \tau \text{ exp}, \pi\rangle \\ & \mid \langle \text{Boolean exp}, \pi\rangle \ \mathbf{and} \ \langle \text{Boolean exp}, \pi\rangle \\ & \mid (\forall \iota)\ \langle \text{Boolean exp},\ [\pi \mid \iota\colon \text{integer exp}]\rangle \\ & \mid (\exists \iota)\ \langle \text{Boolean exp},\ [\pi \mid \iota\colon \text{integer exp}]\rangle\end{aligned}$$

$$\begin{aligned}\langle \text{comm}, \pi\rangle \ ::= {} & \mathbf{skip} \mid \langle \tau \text{ acc}, \pi\rangle := \langle \tau \text{ exp}, \pi\rangle \\ & \mid \langle \text{comm}, \pi\rangle; \langle \text{comm}, \pi\rangle \\ & \mid \mathbf{while}\ \langle \text{Boolean exp}, \pi\rangle\ \mathbf{do}\ \langle \text{comm}, \pi\rangle \\ & \mid \mathbf{new}\ \tau\ \mathbf{var}\ \iota\ \mathbf{in}\ \langle \text{comm}, [\pi \mid \iota\colon \tau\ \tau \text{ var}]\rangle\end{aligned}$$

Notice that we permit Boolean expressions to contain quantifiers, at least over the set of integers. This will permit us to simplify our presentation by equating assertions with Boolean expressions. Of course, one cannot actually implement the evaluation of quantifiers, but semantically, quantified expressions have exactly the same kind of meaning as other logical expressions. It should be noted, however, that this view is tenable only because we have prohibited recursive functions and other constructs that would impose on the meaning of expressions a domain structure for which quantification would not be a continuous operation.

Finally, we give the syntax of specifications:

$$\begin{aligned}\langle \text{spec}, \pi\rangle \ ::= {} & \{\langle \text{Boolean exp}, \pi\rangle\}\langle \text{comm}, \pi\rangle\{\langle \text{Boolean exp}, \pi\rangle\} \\ & \mid \ \{\langle \text{Boolean exp}, \pi\rangle\} \\ & \mid \ \langle \text{commlike}, \pi\rangle \ \#\ \langle \text{explike}, \pi\rangle \\ & \mid \ \langle \text{spec}, \pi\rangle \ \&\ \langle \text{spec}, \pi\rangle \\ & \mid \ \langle \text{spec}, \pi\rangle \Rightarrow \langle \text{spec}, \pi\rangle \\ & \mid \ (\forall \iota\colon \omega)\langle \text{spec},\ [\pi \mid \iota\colon \omega]\rangle \\ & \mid \ \mathbf{gv}_\tau(\ \langle \tau\ \tau \text{ var}, \pi\rangle\ )\end{aligned}$$

A specification of the form $\{P\}\ C\ \{Q\}$ is true if and only if, for all states of the store in which P is true executing C either does not terminate or terminates with a state for which Q is true. This differs from Hoare's definition [1] in being implicitly quantified over states of the store but not over environments, since in our formalism $\{P\}\ C\ \{Q\}$, like any other kind of specification, is a predicate about environments.

A specification of the form $\{P\}$ is true if and only if P is true for all states of the store; in this case we say that P is a *static* assertion.

A specification of the form $C \# E$ is true if and only if C does not interfere with E. When C is a command and E is an expression this means that for any state of the store, executing C will not, at any time during the execution, affect the value of E. The meaning of this specification for other command-like and expression-like phrases will be clarified by equivalences in Sections 5 and 6.

The operators &, $\Rightarrow$, and $\forall$ denote conjunction, implication, and universal quantification of specifications. (In contrast to [8], we do not combine & and $\Rightarrow$ into a single operation.) Notice that quantification is over meanings of a particular phrase type (not values of a particular data type).

A specification of the form $\mathbf{gv}_\tau(V)$ means that V is a "good" variable, i.e., a variable that will always possess the last value assigned to it. This will be clarified by an equivalence in Section 7.

In conclusion, we state without proof a global property of our syntax. If $\pi' \leq \pi$ and $\gamma' \leq \gamma'$ then $\langle \gamma, \pi \rangle \subseteq \langle \gamma', \pi' \rangle$. (When a phrase class name $\langle \gamma, \pi \rangle$ is used as a set, it denotes the set of all phrases derivable from $\langle \gamma, \pi \rangle$.) When $\pi' = \pi$ this is an immediate consequence of the production schema describing the subtype relation; however, the general case requires a structural induction argument involving the entire syntax.

3. Equivalences

Basically, two phrases are equivalent if they possess the same meaning in every environment or, more abstractly, if they possess the same semantics. However, this concept is complicated by the possibility that a phrase may belong to more than one phrase class $\langle \gamma, \pi \rangle$, and possess a distinct semantics for each phrase class to which it belongs.

For this reason we must qualify equivalences with phrase class names. When P and Q belong to $\langle \gamma, \pi \rangle$ and have the same semantics for $\langle \gamma, \pi \rangle$, we write $P \equiv_{\gamma,\pi} Q$ and say that P and Q are *equivalent for* $\langle \gamma, \pi \rangle$.

Suppose $\pi' \leq \pi$ and $\gamma \leq \gamma'$. If $P \equiv_{\gamma,\pi} Q$ holds then $P \equiv_{\gamma',\pi'} Q$ will also hold. We will call this kind of deduction "type broadening."

Although types can be broadened, they cannot always be "narrowed." For example,

$$k > 4 \ \equiv_{\text{Boolean exp, } [k\text{: integer exp}]} \ k \geq 5$$

is true, but

$$k > 4 \ \equiv_{\text{Boolean exp, } [k\text{: real exp}]} \ k \geq 5$$

is false.

However, there is a special case in which type assignments can be narrowed: if $P \equiv_{\gamma,\pi} Q$ holds and π' is a restriction of π whose domain includes all identifiers that occur free in P or Q, then $P \equiv_{\gamma,\pi'} Q$.

We will usually describe sets of equivalences schematically. For example, to express the validity of beta reduction, suppose $P \in \langle \theta_2, [\pi \mid \iota{:}\,\theta_1] \rangle$ and $Q \in \langle \theta_1, \pi \rangle$. Then

$$(\lambda \iota{:}\,\theta_1.\, P)(Q) \;\equiv_{\theta_2,\pi}\; P|_{\iota \to Q} \;. \tag{3.1}$$

Here $P|_{\iota\to Q}$ denotes the result of substituting Q for the free occurrences of ι in P, with appropriate renaming of bound identifiers in P. More generally, if $\iota_1, \ldots, \iota_n$ are distinct identifiers, $P \in \langle \gamma, [\pi \mid \iota_1{:}\,\omega_1 \mid \cdots \mid \iota_n{:}\,\omega_n] \rangle$, and $Q_1 \in \langle \omega_1, \pi \rangle, \ldots, Q_n \in \langle \omega_n, \pi \rangle$, then

$$P|_{\iota_1,\ldots,\iota_n \to Q_1,\ldots,Q_n}$$

denotes the phrase in $\langle \gamma, \pi \rangle$ obtained by substituting each Q_i for the free occurrences of ι_i in P, with appropriate renaming.

To express the validity of eta reduction, suppose $P \in \langle \theta_1 \to \theta_2, \pi \rangle$ and $\iota \notin \mathbf{dom}(\pi)$ (so that ι does not occur free in P). Then

$$\lambda \iota{:}\,\theta_1.\, (P(\iota)) \equiv_{\theta_1 \to \theta_2,\pi} \; P \;. \tag{3.2}$$

Further equivalences describe the operator **rec**, the constant $\perp$, and the conditional construction. If $P \in \langle \rho \to \rho, \pi \rangle$ then

$$\mathbf{rec}\; P \;\equiv_{\rho,\pi}\; P(\mathbf{rec}\; P) \;. \tag{3.3}$$

If $P \in \langle \theta, \pi \rangle$ then

$$\perp_{\theta \to \rho} (P) \;\equiv_{\rho,\pi} \perp_\rho \;. \tag{3.4}$$

On the other hand,

$$\lambda \iota{:}\,\theta.\, \perp_\rho \,\equiv_{\theta \to \rho,\pi} \perp_{\theta \to \rho} \;. \tag{3.5}$$

If $P \in \langle \text{Boolean exp}, \pi \rangle$, $Q_1, Q_2 \in \langle \theta_1 \to \theta_2, \pi \rangle$, and $R \in \langle \theta_1, \pi \rangle$, then

$$(\mathbf{if}\; P \;\mathbf{then}\; Q_1 \;\mathbf{else}\; Q_2)(R) \;\equiv_{\theta_2,\pi}\; \mathbf{if}\; P \;\mathbf{then}\; Q_1(R) \;\mathbf{else}\; Q_2(R) \;. \tag{3.6}$$

On the other hand, if $P \in \langle \text{Boolean exp}, \pi \rangle$, Q_1, $Q_2 \in \langle \theta_2,\ [\pi \mid \iota{:}\,\theta_1] \rangle$ and $\iota \notin \mathbf{dom}(\pi)$, then

$$\lambda \iota{:}\,\theta_1.\, \mathbf{if}\; P \;\mathbf{then}\; Q_1 \;\mathbf{else}\; Q_2 \;\equiv_{\theta_1 \to \theta_2,\pi}\; \mathbf{if}\; P \;\mathbf{then}\; \lambda \iota{:}\,\theta_1.\, Q_1 \;\mathbf{else}\; \lambda \iota{:}\,\theta_1.\, Q_2 \;. \tag{3.7}$$

The **let** and **letrec** constructions are described by equivalences that can be regarded as definitions of these constructions in terms of application, abstraction, and **rec**. If $P \in \langle \theta_1, \pi \rangle$ and $Q \in \langle \theta_2,\ [\pi \mid \iota{:}\,\theta_1] \rangle$ then

$$\mathbf{let}\; \iota \;\mathbf{be}\; P \;\mathbf{in}\; Q \;\equiv_{\theta_2,\pi}\; (\lambda \iota{:}\,\theta_1.\, Q)(P) \;. \tag{3.8}$$

If $P \in \langle \rho, [\pi \mid \iota{:}\,\rho] \rangle$ and $Q \in \langle \theta, [\pi \mid \iota{:}\,\rho] \rangle$ then

$$\mathbf{letrec}\; \iota{:}\,\rho \;\mathbf{be}\; P \;\mathbf{in}\; Q \;\equiv_{\theta,\pi}\; (\lambda \iota{:}\,\rho.\, Q)(\mathbf{rec}\; \lambda \iota{:}\,\rho.\, P) \;. \tag{3.9}$$

Next, we give more specific equivalences describing commands. Suppose C, C', $C'' \in \langle\text{comm}, \pi\rangle$, $P \in \langle\text{Boolean exp}, \pi\rangle$, A, $A' \in \langle\tau \text{ acc}, \pi\rangle$, and E, $E' \in \langle\tau \text{ exp}, \pi\rangle$. Then

$$C;\ \textbf{skip}\ \equiv_{\text{comm},\pi}\ C \tag{3.10}$$

$$\textbf{skip};\ C\ \equiv_{\text{comm},\pi}\ C \tag{3.11}$$

$$C;(C'\ ;\ C'')\ \equiv_{\text{comm},\pi}\ (C\ ;\ C');\ C'' \tag{3.12}$$

$$A := \textbf{if}\ P\ \textbf{then}\ E\ \textbf{else}\ E'\ \equiv_{\text{comm},\pi}\ \textbf{if}\ P\ \textbf{then}\ A := E\ \textbf{else}\ A := E' \tag{3.13}$$

$$(\textbf{if}\ P\ \textbf{then}\ A\ \textbf{else}\ A') := E \equiv_{\text{comm},\pi} \textbf{if}\ P\ \textbf{then}\ A := E\ \textbf{else}\ A' := E \tag{3.14}$$

$$(\textbf{if}\ P\ \textbf{then}\ C\ \textbf{else}\ C');\ C''\ \equiv_{\text{comm},\pi}\ \textbf{if}\ P\ \textbf{then}\ (C;C'')\ \textbf{else}\ (C';C'') \tag{3.15}$$

$$\textbf{while}\ P\ \textbf{do}\ C\ \equiv_{\text{comm},\pi}\ \textbf{if}\ P\ \textbf{then}\ (C;\ \textbf{while}\ P\ \textbf{do}\ C)\ \textbf{else skip}. \tag{3.16}$$

There are also obvious equivalences describing the conjunction and implication of specifications. Suppose $S, S', S'' \in \langle\text{spec}, \pi\rangle$. Then

$$\{\textbf{true}\}\ \&\ S\ \equiv_{\text{spec},\pi}\ S \tag{3.17}$$

$$\{\textbf{false}\}\ \&\ S\ \equiv_{\text{spec},\pi}\ \{\textbf{false}\} \tag{3.18}$$

$$(S\ \&\ S')\ \&\ S''\ \equiv_{\text{spec},\pi}\ S\ \&\ (S'\ \&\ S'') \tag{3.19}$$

$$S\ \&\ S'\ \equiv_{\text{spec},\pi}\ S'\ \&\ S \tag{3.20}$$

$$S\ \&\ S\ \equiv_{\text{spec},\pi}\ S \tag{3.21}$$

$$\{\textbf{true}\} \Rightarrow S\ \equiv_{\text{spec},\pi}\ S \tag{3.22}$$

$$\{\textbf{false}\} \Rightarrow\ S\ \equiv_{\text{spec},\pi}\ \{\textbf{true}\} \tag{3.23}$$

$$S \Rightarrow \{\textbf{true}\}\ \equiv_{\text{spec},\pi}\ \{\textbf{true}\} \tag{3.24}$$

$$S \Rightarrow S\ \equiv_{\text{spec},\pi}\ \{\textbf{true}\} \tag{3.25}$$

$$S \Rightarrow (S' \Rightarrow S'')\ \equiv_{\text{spec},\pi}\ S\ \&\ S' \Rightarrow S'' \tag{3.26}$$

$$S \Rightarrow S'\ \&\ S''\ \equiv_{\text{spec},\pi}\ (S \Rightarrow S')\ \&\ (S \Rightarrow S''). \tag{3.27}$$

Finally, there are an indefinite number of equivalences of the form

$$P\ \equiv_{\text{Boolean exp},\pi}\ \textbf{true} \tag{3.28}$$

that describe the mathematics of data types. When the above equivalence holds, we say that $P \in \langle\text{Boolean exp}, \pi\rangle$ is a *mathematical fact* for π. For example, $k > 4$ **implies** $k \geq 5$ is a mathematical fact for [k: integer exp].

4. Universal Specifications and Their Inference

Roughly speaking, a specification is said to be *universal* if it is true for all environments. However, much as with equivalences, the situation is complicated by the fact that a specification can belong to $\langle\text{spec}, \pi\rangle$ for several type assignments π, and that its universality can depend upon π. To be precise, when $S \in \langle\text{spec}, \pi\rangle$ and the semantics of S for $\langle\text{spec}, \pi\rangle$ maps all environments in its domain into **true**, we write $S \;::\; \pi$ and say that S is *universal for* π.

For example,

$$\{k > 4\} \Rightarrow \{k \geq 5\} \;::\; [k\colon \text{integer exp}] \;,$$

but *not*

$$\{k > 4\} \Rightarrow \{k \geq 5\} \;::\; [k\colon \text{real exp}] \;.$$

Specification logic is a formal system for inferring universal specifications. Its rules of inference are schemas with the form "If $S_1 \;::\; \pi_1$ and $\cdots$ and $S_n \;::\; \pi_n$ then $S \;::\; \pi$," usually prefaced by restrictions on metavariables. The $n \geq 0$ specifications $S_1, \ldots, S_n$ are called *premisses* and S is called the *conclusion*.

Our first rule of inference captures the idea that a specification will remain universal when any subphrase is replaced by another subphrase to which it is equivalent. To state this precisely, we must characterize "subphrase" syntactically:

(4.1) Inference by Equivalence. Suppose $S \in \langle\text{spec}, \pi\rangle$, $P \equiv_{\gamma,\pi'} Q$, and there is a derivation of S from $\langle\text{spec}, \pi\rangle$ containing a subderivation of P from $\langle\gamma, \pi'\rangle$. Let S' be the specification derived from $\langle\text{spec}, \pi\rangle$ by the derivation obtained from the derivation of S by replacing the subderivation of P by a subderivation of Q. If

$$S \;::\; \pi$$

then

$$S' \;::\; \pi \;.$$

Next, we have two rules for broadening and narrowing type assignments in much the same way as for equivalences:

(4.2) Broadening Type Assignments. If $S \in \langle\text{spec}, \pi\rangle$ and $\pi' \leq \pi$ and

$$S \;::\; \pi$$

then

$$S \;::\; \pi' \;.$$

(4.3) Narrowing Type Assignments. If $S \in \langle \text{spec}, \pi \rangle$ is a restriction of π whose domain includes all identifiers occurring free in S, and

$$S \ :: \ \pi$$

then

$$S \ :: \ \pi' \ .$$

Three more rules are needed to capture the properties of & and $\Rightarrow$. In these rules we assume $S_1, S_2, S_3 \in \langle \text{spec}, \ \pi \rangle$:

(4.4) Truth.

$$\{\textbf{true}\} \ :: \ \pi \ .$$

(4.5) Adding Assumptions. If

$$S_2 \ :: \ \pi$$

then

$$S_1 \Rightarrow S_2 \ :: \ \pi \ .$$

(4.6) Modus Ponens. If

$$S_1 \Rightarrow S_2 \ :: \ \pi$$

and

$$S_2 \Rightarrow S_3 \ :: \ \pi$$

then

$$S_1 \Rightarrow S_3 \ :: \ \pi \ .$$

Since they are so similar to ordinary logic, we will often omit the details of applying these rules, as well as applying (4.1) for equivalences (3.17) to (3.27).

Two more rules deal with quantification:

(4.7) Quantifier Introduction. Suppose $S_1 \in \langle \text{spec}, \pi \rangle$, $S_2 \in \langle \text{spec}, [\pi \mid \iota{:}\,\omega] \rangle$ and $\iota \notin \textbf{dom}(\pi)$. If

$$S_1 \Rightarrow S_2 \ :: \ [\pi \mid \iota{:}\,\omega]$$

then

$$S_1 \Rightarrow (\forall \iota{:}\,\omega) S_2 \ :: \ \pi \ .$$

(4.8) Quantifier Removal. Suppose $S \in \langle \text{spec}, [\pi \mid \iota_1{:}\,\omega_1 \mid \cdots \mid \iota_n{:}\,\omega_n] \rangle$, and $P_1 \in \langle \omega_1, \pi \rangle$, ..., $P_n \in \langle \omega_n, \pi \rangle$, where $\iota_1, \ldots, \iota_n$ are distinct identifiers. Then

$$(\forall \iota_1{:}\,\omega_1) \cdots (\forall \iota_n{:}\,\omega_n) S \Rightarrow S|_{\iota_1,\ldots,\iota_n \longrightarrow P_1,\ldots,P_n} \ :: \ \pi \ .$$

From these rules we can derive the fact that universality is preserved by substitution:

(4.9) Free Substitution. Suppose $S \in \langle \text{spec}, [\pi \mid \iota_1 \!:\! \omega_1 \mid \cdots \mid \iota_n \!:\! \omega_n] \rangle$, $P_1 \in \langle \omega_1, \pi \rangle$, $\ldots$, $P_n \in \langle \omega_n, \pi \rangle$, $\iota_1, \ldots, \iota_n$ are distinct, and

$$S \ :: \ [\pi \mid \iota_1 \!:\! \omega_1 \mid \cdots \mid \iota_n \!:\! \omega_n] \ .$$

From (4.5),

$$\{\mathbf{true}\} \Rightarrow S \ :: \ [\pi \mid \iota_1 \!:\! \omega_1 \mid \cdots \mid \iota_n \!:\! \omega_n] \ ,$$

and by repeated application of (4.7),

$$\begin{aligned} &\{\mathbf{true}\} \Rightarrow (\forall \iota_n \!:\! \omega_n) S \ :: \ [\pi \mid \iota_1 \!:\! \omega_1 \mid \cdots \mid \iota_{n-1} \!:\! \omega_{n-1}] \\ &\qquad \vdots \\ &\{\mathbf{true}\} \Rightarrow (\forall \iota_1 \!:\! \omega_1) \cdots (\forall \iota_n \!:\! \omega_n) S \ :: \ \pi \ . \end{aligned}$$

Then (4.8), with modus ponens, gives

$$\{\mathbf{true}\} \Rightarrow S|_{\iota_1, \ldots, \iota_n \longrightarrow P_1, \ldots, P_n} \ :: \ \pi \ ,$$

and equivalence (3.22), via (4.1), gives the conclusion

$$S|_{\iota_1, \ldots, \iota_n \longrightarrow P_1, \ldots, P_n} \ :: \ \pi \ .$$

The derivation of this rule typifies the form of such derivations that will be used in most of this report: it simply consists of a statement of the rule with an appropriate argument inserted between premisses and conclusion.

Next, we derive a rule that permits any mathematical fact to be used as a static assertion:

(4.10) Mathematical Fact Introduction. Suppose P is a mathematical fact for π. Then applying equivalence (3.28) to $\{\mathbf{true}\} \ :: \ \pi$ gives the conclusion

$$\{P\} \ :: \ \pi \ .$$

Finally, we give a collection of rules which contain neither premisses nor metavariables, and are therefore simply *axioms*, i.e., particular specifications that are universal. The first of these rules serves to "lift" reasoning about assertions into reasoning about specifications. The next eight rules are reformulations of various aspects of Hoare's logic [1], and the final rule says that a command will preserve an assertion that it does not interfere with. For these rules we assume π is a type assignment mapping p, q, r, i, p_1, p_2, q_1, q_2 and ℓ into **Boolean exp**, and c, c_1, and c_2 into **comm**:

(4.11) Static Implication.

$$\{p\} \ \& \ \{p \ \mathbf{implies} \ q\} \Rightarrow \{q\} \ :: \ \pi \ .$$

(4.12) Command Composition.

$$\{p\} \ c_1 \ \{q\} \ \& \ \{q\} \ c_2 \ \{r\} \Rightarrow \{p\} \ c_1; \ c_2 \ \{r\} \ :: \ \pi \ .$$

(4.13) Strengthening Precedent.

$$\{p \textbf{ implies } q\} \ \& \ \{q\}\ c\ \{r\} \Rightarrow \{p\}\ c\ \{r\} \ :: \ \pi\ .$$

(4.14) Weakening Consequent.

$$\{p\}\ c\ \{q\} \ \& \ \{q \textbf{ implies } r\} \Rightarrow \{p\}\ c\ \{r\} \ :: \ \pi\ .$$

(4.15) **while** Command.

$$\{i \textbf{ and } \ell\}\ c\ \{i\} \Rightarrow \{i\} \textbf{ while } \ell \textbf{ do } c\ \{i \textbf{ and } \neg\ell\} \ :: \ \pi\ .$$

(4.16) Conditional Command.

$$\begin{aligned}&\{p \textbf{ and } \ell\}\ c_1\ \{q\} \ \& \ \{p \textbf{ and } \neg\ell\}\ c_2\ \{q\}\\ &\qquad \Rightarrow \{p\} \textbf{ if } \ell \textbf{ then } c_1 \textbf{ else } c_2\ \{q\} \ :: \ \pi\ .\end{aligned}$$

(4.17) **skip** Command.

$$\{p\} \textbf{ skip } \{p\} \ :: \ \pi\ .$$

(4.18) Specification Conjunction.

$$\{p_1\}\ c\ \{q_1\} \ \& \ \{p_2\}\ c\ \{q_2\} \Rightarrow \{p_1 \textbf{ and } p_2\}\ c\ \{q_1 \textbf{ and } q_2\} \ :: \ \pi\ .$$

(4.19) Specification Disjunction.

$$\{p_1\}\ c\ \{q_1\} \ \& \ \{p_2\}\ c\ \{q_2\} \Rightarrow \{p_1 \textbf{ or } p_2\}\ c\ \{q_1 \textbf{ or } q_2\} \ :: \ \pi\ .$$

(4.20) Constancy.

$$c \# p \ \& \ \{q\}\ c\ \{r\} \Rightarrow \{q \textbf{ and } p\}\ c\ \{r \textbf{ and } p\} \ :: \ \pi\ .$$

The occurrence of # in the rule of constancy raises the question of how one can infer noninterference specifications, which will be considered in the next two sections.

5. Leftside Noninterference Decomposition

In this section and the next, we develop equivalences and inference rules describing noninterference. The overall effect of these rules is to allow any noninterference specification $C \# E$ to be decomposed into noninterference specifications in which only identifiers occur on the leftside, or rightside, of #. In this section, we deal with leftside decomposition.

To begin, several equivalences serve to define noninterference for acceptors and proper procedures in terms of noninterference for commands. If $A \in \langle \tau \text{ acc}, \pi\rangle, E \in \langle \text{explike}, \pi\rangle$, and $\iota \notin \mathbf{dom}(\pi)$, then

$$A \# E \equiv_{\text{spec},\pi} (\forall \iota\colon \tau \textbf{ exp})(A := \iota \# E)\ . \tag{5.1}$$

If $\theta_1 \not\leq \mathbf{commlike}$, $\theta_2 \leq \mathbf{commlike}$, $H \in \langle \theta_1 \rightarrow \theta_2, \pi \rangle$, $E \in \langle \text{explike}, \pi \rangle$, and $\iota \notin \mathbf{dom}(\pi)$, then

$$H \ \# \ E \equiv_{\text{spec},\pi} (\forall \iota \colon \theta_1) \ H(\iota) \ \# \ E \ . \tag{5.2}$$

If $\theta_1 \leq \mathbf{commlike}$, $\theta_2 \leq \mathbf{commlike}$, $H \in \langle \theta_1 \rightarrow \theta_2, \pi \rangle$, $E \in \langle \text{explike}, \pi \rangle$, and $\iota \notin \mathbf{dom}(\pi)$, then

$$H \ \# \ E \equiv_{\text{spec},\pi} (\forall \iota \colon \theta_1)(\iota \ \# \ E \Rightarrow H(\iota) \ \# \ E) \ . \tag{5.3}$$

In addition, there are basic leftside decomposition rules for a variety of constructs in the language:

(5.4) If $P \in \langle \rho \rightarrow \rho, \pi \rangle$ and $E \in \langle \text{explike}, \pi \rangle$ then

$$P \ \# \ E \Rightarrow \mathbf{rec} \ P \ \# \ E \ :: \ \pi \ .$$

(5.5) If $E \in \langle \text{explike}, \pi \rangle$ then

$$\perp_\rho \ \# \ E \ :: \ \pi \ .$$

(Notice that $\rho \leq \mathbf{commlike}$ holds for all recursive phrase types.)

(5.6) If $\theta \leq \mathbf{commlike}$, $B \in \langle \text{Boolean exp}, \pi \rangle$, P_1, $P_2 \in \langle \theta, \pi \rangle$, and $E \in \langle \text{explike}, \pi \rangle$, then

$$P_1 \ \# \ E \ \& \ P_2 \ \# \ E \Rightarrow \mathbf{if} \ B \ \mathbf{then} \ P_1 \ \mathbf{else} \ P_2 \ \# \ E \ :: \ \pi \ .$$

(5.7) If $E \in \langle \text{explike}, \pi \rangle$ then

$$\mathbf{skip} \ \# \ E \ :: \ \pi \ .$$

(5.8) If $C_1, C_2 \in \langle \text{comm}, \pi \rangle$ and $E \in \langle \text{explike}, \pi \rangle$ then

$$C_1 \ \# \ E \ \& \ C_2 \ \# \ E \Rightarrow C_1; C_2 \ \# \ E \ :: \ \pi \ .$$

(5.9) If $B \in \langle \text{Boolean exp}, \pi \rangle$, $C \in \langle \text{comm}, \pi \rangle$, and $E \in \langle \text{explike}, \pi \rangle$ then

$$C \ \# \ E \Rightarrow \ \mathbf{while} \ B \ \mathbf{do} \ C \ \# \ E \ :: \ \pi \ .$$

(5.10) If $C \in \langle \text{comm}, [\pi \mid \iota \colon \tau \ \tau \ \text{var}] \rangle$ and $E \in \langle \text{explike}, \pi \rangle$ and $\iota \notin \mathbf{dom}(\pi)$ then

$$(\forall \iota \colon \tau \ \tau \ \text{var})(\iota \ \# \ E \Rightarrow C \ \# \ E) \ \Rightarrow \ (\mathbf{new} \ \tau \ \mathbf{var} \ \iota \ \mathbf{in} \ C) \ \# \ E \ :: \ \pi \ .$$

We now want to derive a general inference rule that subsumes all of the previous rules and equivalences for leftside noninterference decomposition. As a preliminary we must define, for any command-like phrase C, a subset $\mathcal{F}_{\text{commlike}}(C)$ of the identifiers occurring free in C. Let $C \in \langle \text{commlike}, \pi \rangle$. Then $\mathcal{F}_{\text{commlike}}(C) \subseteq \{\iota \mid \iota \in \ \mathbf{dom}(\pi) \text{ and } \pi(\iota) \leq \mathbf{commlike}\}$ is the set of identifiers such that:

(a) If C is an identifier ι then

$$\mathcal{F}_{\text{commlike}}(C) = \{\iota\}$$

(b) If C is $C_1(C_2)$ then there must be one or more pairs θ_1, θ_2 of phrase types such that $\theta_2 \leq$ **commlike**, $C_1 \in \langle \theta_1 \rightarrow \theta_2, \pi \rangle$, and $C_2 \in \langle \theta_1, \pi \rangle$.

(i) If there is any such θ_1 for which $\theta_1 \not\leq$ **commlike** then

$$\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_1)$$

(ii) Otherwise,

$$\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_1) \ \cup \ \mathcal{F}_{\text{commlike}}(C_2) \ .$$

(c) If C is $\lambda\iota{:}\,\theta.\,C_1$ then

$$\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_1) - \{\iota\} \ .$$

(d) If C is **rec** C_1 then

$$\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_1) \ .$$

(e) If C is $\perp_\rho$ then

$$\mathcal{F}_{\text{commlike}}(C) = \{\} \ .$$

(f) If C is **if** B **then** C_1 **else** C_2 then

$$\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_1) \ \cup \ \mathcal{F}_{\text{commlike}}(C_2) \ .$$

(g) If C is **let** ι **be** C_1 **in** C_2 then there must be at least one pair θ_1, θ_2 of phrase types such that $\theta_2 \leq$ **commlike**, $C_1 \in \langle \theta_1, \pi \rangle$, and $C_2 \in \langle \theta_2, \ [\pi \mid \iota{:}\,\theta_1] \rangle$.

(i) If there is any such θ_1 for which $\theta_1 \not\leq$ **commlike** then

$$\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_2) - \{\iota\} \ .$$

(ii) Otherwise,

$$\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_1) \ \cup \ (\mathcal{F}_{\text{commlike}}(C_2) - \{\iota\}) \ .$$

(h) If C is **letrec** $\iota{:}\,\rho$ **be** C_1 **in** C_2 then

$$\mathcal{F}_{\text{commlike}}(C) = (\mathcal{F}_{\text{commlike}}(C_1) \ \cup \ \mathcal{F}_{\text{commlike}}(C_2)) - \{\iota\} \ .$$

(i) If C is **skip** then

$$\mathcal{F}_{\text{commlike}}(C) = \{\} \ .$$

(j) If C is $C_1 := C_2$ then

$$\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_1) \ .$$

(k) If C is $C_1;\, C_2$ then

$$\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_1) \;\cup\; \mathcal{F}_{\text{commlike}}(C_2)\ .$$

(l) If C is **while** B **do** C_1 then

$$\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_1)\ .$$

(m) If C is **new** τ **var** ι **in** C_1 then

$$\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_1) - \{\iota\}\ .$$

This definition is the key to the following derived rule of inference:

(5.11) Leftside Noninterference Decomposition. Suppose that $C \in \langle \text{commlike}, \pi\rangle$, $E \in \langle \text{explike}, \pi\rangle$, and $\mathcal{F}_{\text{commlike}}(C) = \{\iota_1, \ldots, \iota_n\}$. Then

$$\iota_1 \ \#\ E \ \& \cdots \&\ \iota_n \ \#\ E \Rightarrow C \ \#\ E \ :: \ \pi\ .$$

The proof of this rule is considerably more complex than that of the derived rules encountered previously. To begin with, we will assume that no identifier bound in C belongs to $\mathbf{dom}(\pi)$; this restriction can always be met by renaming the bound identifiers of C, which changes neither the meaning of C nor $\mathcal{F}_{\text{commlike}}(C)$. With this restriction, the proof is by structural induction on C, with cases that parallel the definition of $\mathcal{F}_{\text{commlike}}$:

(a) If C is an identifier ι then $\mathcal{F}_{\text{commlike}}(C) = \{\iota\}$, so that the specification to be proved is

$$\iota \ \#\ E \Rightarrow \iota \ \#\ E \ :: \ \pi\ ,$$

which follows from the fact that any specification implies itself.

(b) If C is $C_1(C_2)$ then there must be one or more pairs θ_1, θ_2 of phrase types such that $\theta_2 \leq \textbf{commlike}$, $C_1 \in \langle \theta_1 \to \theta_2, \pi\rangle$, and $C_2 \in \langle \theta_1, \pi\rangle$.

(i) If there is any such θ_1 for which $\theta_1 \not\leq \textbf{commlike}$ then $\mathcal{F}_{\text{commlike}}(C_1) = \mathcal{F}_{\text{commlike}}(C)$, so that the induction hypothesis gives

$$\iota_1 \ \#\ E \ \& \cdots \&\ \iota_n \ \#\ E \Rightarrow C_1 \ \#\ E \ :: \ \pi\ .$$

But equivalence (5.2) gives

$$C_1 \ \#\ E \Rightarrow (\forall \iota\colon \theta_1) C_1(\iota) \ \#\ E \ :: \ \pi\ ,$$

where ι is some identifier not in $\mathbf{dom}(\pi)$, and quantifier removal gives

$$(\forall \iota\colon \theta_1) C_1(\iota) \ \#\ E \Rightarrow C_1(C_2) \ \#\ E \ :: \ \pi\ .$$

Thus by modus ponens,

$$\iota_1 \ \#\ E \ \& \cdots \&\ \iota_n \ \#\ E \Rightarrow C_1(C_2) \ \#\ E \ :: \ \pi\ .$$

(ii) Otherwise, $\mathcal{F}_{\text{commlike}}(C_1) \subseteq \mathcal{F}_{\text{commlike}}(C)$ and $\mathcal{F}_{\text{commlike}}(C_2) \subseteq \mathcal{F}_{\text{commlike}}(C)$, so that the induction hypothesis gives (with added assumptions)

$$\iota_1 \# E \,\&\cdots\&\, \iota_n \# E \Rightarrow C_1 \# E \;::\; \pi \;,$$

$$\iota_1 \# E \,\&\cdots\&\, \iota_n \# E \Rightarrow C_2 \# E \;::\; \pi \;.$$

But equivalence (5.3) gives

$$C_1 \# E \Rightarrow (\forall \iota\colon \theta_1)(\iota \# E \Rightarrow C_1(\iota) \# E) \;::\; \pi \;,$$

where ι is some identifier not in $\mathbf{dom}(\pi)$, and quantifier removal gives

$$(\forall \iota\colon \theta_1)(\iota \# E \Rightarrow C_1(\iota) \# E) \Rightarrow (C_2 \# E \Rightarrow C_1(C_2) \# E) \;::\; \pi \;.$$

Thus

$$\iota_1 \# E \,\&\cdots\&\, \iota_n \# E \Rightarrow C_1(C_2) \# E \;::\; \pi \;.$$

(c) If C is $\lambda\iota\colon\theta.\, C_1$ then

(i) If $\theta \not\leq$ **commlike** then $\mathcal{F}_{\text{commlike}}(C_1) = \mathcal{F}_{\text{commlike}}(C)$, since ι cannot belong to $\mathcal{F}_{\text{commlike}}(C_1)$, so that the induction hypothesis gives

$$\iota_1 \# E \,\&\cdots\&\, \iota_n \# E \Rightarrow C_1 \# E \;::\; [\pi \mid \iota\colon\theta] \;.$$

Then quantifier introduction and reverse beta reduction give

$$\iota_1 \# E \,\&\cdots\&\, \iota_n \# E \Rightarrow (\forall \iota\colon \theta)(\lambda\iota\colon\theta.\, C_1)(\iota) \# E \;::\; \pi$$

since ι occurs bound in C and therefore does not belong to $\mathbf{dom}(\pi)$. But equivalence (5.2) gives

$$(\forall \iota\colon \theta)(\lambda\iota\colon\theta.\, C_1)(\iota) \# E \;\Rightarrow (\lambda\iota\colon\theta.\, C_1) \# E \;::\; \pi \;.$$

Thus

$$\iota_1 \# E \,\&\cdots\&\, \iota_n \# E \Rightarrow (\lambda\iota\colon\theta.\; C_1) \# E \;::\; \pi \;.$$

(ii) If $\theta \leq$ **commlike** then $\mathcal{F}_{\text{commlike}}(C_1) \subseteq \mathcal{F}_{\text{commlike}}(C) \,\cup\, \{\iota\}$, so that the induction hypothesis gives

$$\iota_1 \# E \,\&\cdots\&\, \iota_n \# E \,\&\, \iota \# E \Rightarrow C_1 \# E \;::\; [\pi \mid \iota\colon\theta].$$

Then quantifier introduction and reverse beta reduction give

$$\iota_1 \# E \,\&\cdots\&\, \iota_n \# E \;\Rightarrow (\forall \iota\colon \theta)(\iota \# E \Rightarrow (\lambda\iota\colon\theta.\, C_1)(\iota) \# E) \;::\; \pi$$

since ι occurs bound in C and therefore does not belong to $\mathbf{dom}(\pi)$. But equivalence (5.3) gives

$$(\forall \iota\colon \theta)(\iota \# E \Rightarrow (\lambda\iota\colon\theta.\, C_1)(\iota) \# E) \Rightarrow (\lambda\iota\colon\theta.\, C_1) \# E \;::\; \pi \;.$$

Thus

$$\iota_1 \# E \,\&\cdots\&\, \iota_n \# E \Rightarrow (\lambda\iota\colon\theta.\, C_1) \# E \;::\; \pi \;.$$

(d) If C is **rec** C_1 then $\mathcal{F}_{\text{commlike}}(C_1) = \mathcal{F}_{\text{commlike}}(C)$, so that the induction hypothesis gives

$$\iota_1 \# E \;\&\cdots\&\; \iota_n \# E \Rightarrow C_1 \# E \;::\; \pi \;.$$

Then rule (5.4) gives

$$C_1 \# E \Rightarrow \textbf{rec}\ C_1 \# E \;::\; \pi \;,$$

and the result follows by modus ponens.

(e) If C is $\perp_\rho$ then $\mathcal{F}_{\text{commlike}}(C) = \{\}$, so that the specification to be proved is

$$\perp_\rho \ \# E \;::\; \pi \;,$$

which follows from rule (5.5).

(f) If C is **if** C_1 **then** C_2 **else** C_3 then $\mathcal{F}_{\text{commlike}}(C_2) \subseteq \mathcal{F}_{\text{commlike}}(C)$ and $\mathcal{F}_{\text{commlike}}(C_3) \subseteq \mathcal{F}_{\text{commlike}}(C)$, so that the induction hypothesis gives

$$\iota_1 \# E \;\&\cdots\&\; \iota_n \# E \Rightarrow C_2 \# E \;::\; \pi \;,$$

$$\iota_1 \# E \;\&\cdots\&\; \iota_n \# E \Rightarrow C_3 \# E \;::\; \pi \;.$$

Then rule (5.6) gives

$$C_2 \# E \;\&\; C_3 \# E \Rightarrow \textbf{if}\ C_1\ \textbf{then}\ C_2\ \textbf{else}\ C_3 \# E \;::\; \pi \;,$$

and the result follows by modus ponens.

(g) If C is **let** ι **be** C_1 **in** C_2 then there must be at least one pair θ_1, θ_2 of phrase types such that $\theta_2 \leq$ **commlike**, $C_1 \in \langle \theta_1, \pi \rangle$, and $C_2 \in \langle \theta_2, [\pi \mid \iota{:}\, \theta_1] \rangle$.

(i) If there is any such θ_1 for which $\theta_1 \not\leq$ **commlike**, then $\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_2) - \{\iota\} = \mathcal{F}_{\text{commlike}}(\lambda\iota{:}\, \theta_1.\, C_2) = \mathcal{F}_{\text{commlike}}((\lambda\iota{:}\, \theta_1.\, C_2)(C_1))$. Then the argument for (b) and (c) shows that

$$\iota_1 \# E \;\&\cdots\&\; \iota_n \# E \Rightarrow (\lambda\iota{:}\, \theta_1.\ C_2)(C_1) \# E \;::\; \pi \;,$$

and the result follows from the defining equivalence (3.8) for **let**.

(ii) Otherwise, $\mathcal{F}_{\text{commlike}}(C) = \mathcal{F}_{\text{commlike}}(C_1) \cup (\mathcal{F}_{\text{commlike}}(C_2) - \{\iota\}) = \mathcal{F}_{\text{commlike}}(C_1) \cup \mathcal{F}_{\text{commlike}}(\lambda\iota{:}\, \theta_1.\, C_2) = \mathcal{F}_{\text{commlike}}((\lambda\iota{:}\, \theta_1.\, C_2)(C_1))$. Again the argument for (b) and (c) shows that

$$\iota_1 \# E \;\&\cdots\&\; \iota_n \# E \Rightarrow (\lambda\iota{:}\, \theta_1.\ C_2)(C_1) \# E \;::\; \pi \;,$$

and the result follows from the defining equivalence (3.8) for **let**.

(h) If C is **letrec** $\iota{:}\, \rho$ **be** C_1 **in** C_2 then $\mathcal{F}_{\text{commlike}}(C) = (\mathcal{F}_{\text{commlike}}(C_1) \cup \mathcal{F}_{\text{commlike}}(C_2)) - \{\iota\} = \mathcal{F}_{\text{commlike}}(\lambda\iota{:}\, \rho.\, C_1) \cup \mathcal{F}_{\text{commlike}}(\lambda\iota{:}\, \rho.\, C_2) = \mathcal{F}_{\text{commlike}}((\lambda\iota{:}\, \rho.\, C_2)(\textbf{rec}\ \lambda\iota{:}\, \rho.\, C_1))$. Then the argument for (b), (c), and (d) shows that

$$\iota_1 \# E \;\&\cdots\&\; \iota_n \# E \Rightarrow (\lambda\iota{:}\, \rho.\ C_2)(\textbf{rec}\ \lambda\iota{:}\, \rho.\, C_1) \# E \;::\; \pi \;,$$

and the result follows from the defining equivalence (3.9) for **letrec**.

(i) If C is **skip**, then the result follows from rule (5.7) by an argument similar to (e).

(j) If C is $C_1 := C_2$ then $\mathcal{F}_{\text{commlike}}(C_1) = \mathcal{F}_{\text{commlike}}(C)$, so that the induction hypothesis gives

$$\iota_1 \# E \,\&\cdots\&\, \iota_n \# E \Rightarrow C_1 \# E \;::\; \pi \;.$$

But equivalence (5.1) gives

$$C_1 \# E \;\Rightarrow (\forall \iota{:}\, \tau\ \mathbf{exp})(C_1 := \iota \# E) \;::\; \pi \;,$$

where ι is some identifier not in $\mathbf{dom}(\pi)$, and quantifier removal gives

$$(\forall \iota{:}\, \tau\ \mathbf{exp})(C_1 := \iota \# E) \Rightarrow C_1 := C_2 \# E \;::\; \pi \;.$$

The result follows by modus ponens.

(k) If C is $C_1;\ C_2$ then the result follows from rule (5.8) by an argument similar to (f).

(l) If C is **while** C_1 **do** C_2 then the result follows from rule (5.9) by an argument similar to (d).

(m) If C is **new** τ **var** ι **in** C_1, then $\mathcal{F}_{\text{commlike}}(C_1) \subseteq \mathcal{F}_{\text{commlike}}(C) \cup \{\iota\}$, so that the induction hypothesis gives

$$\iota_1 \# E \,\&\cdots\&\, \iota_n \# E \,\&\, \iota \# E \Rightarrow C_1 \# E \;::\; [\pi \mid \iota{:}\, \tau\ \tau\ \mathbf{var}] \;.$$

Then quantifier introduction gives

$$\iota_1 \# E \,\&\cdots\&\, \iota_n \# E \Rightarrow (\forall \iota{:}\, \tau\ \tau\ \mathbf{var})(\iota \# E \Rightarrow C_1 \# E) \;::\; \pi$$

since ι occurs bound in C and therefore does not belong to $\mathbf{dom}(\pi)$. But rule (5.10) gives

$$(\forall \iota{:}\, \tau\ \tau\ \mathbf{var})(\iota \# E \Rightarrow C_1 \# E) \Rightarrow (\mathbf{new}\ \tau\ \mathbf{var}\ \iota\ \mathbf{in}\ C_1) \# E \;::\; \pi \;,$$

so that the result follows by modus ponens.

6. Rightside Noninterference Decomposition

The development of rightside decomposition parallels that of the previous section. We begin with equivalences that define noninterference for function procedures. If $\theta_1 \not\leq \mathbf{explike}$, $\theta_2 \leq \mathbf{explike}$, $C \in \langle \text{commlike}, \pi \rangle$, $H \in \langle \theta_1 \rightarrow \theta_2, \pi \rangle$, and $\iota \notin \mathbf{dom}(\pi)$, then

$$C \# H \;\equiv_{\text{spec},\pi}\; (\forall \iota{:}\, \theta_1) C \# H(\iota) \;. \tag{6.1}$$

On the other hand, if $\theta_1 \leq \mathbf{explike}$, $\theta_2 \leq \mathbf{explike}$, $C \in \langle \text{commlike}, \pi \rangle$, $H \in \langle \theta_1 \rightarrow \theta_2, \pi \rangle$, and $\iota \notin \mathbf{dom}(\pi)$, then

$$C \# H \;\equiv_{\text{spec},\pi}\; (\forall \iota{:}\, \theta_1)(C \# \iota \Rightarrow C \# H(\iota)) \;. \tag{6.2}$$

Next we give basic decomposition rules for a variety of constructs in the language:

(6.3) If $\theta \leq \textbf{explike}$, $C \in \langle \text{commlike}, \pi \rangle$, $B \in \langle \text{Boolean exp}, \pi \rangle$, and E_1, $E_2 \in \langle \theta, \pi \rangle$, then

$$C \# B \,\&\, C \# E_1 \,\&\, C \# E_2 \;\Rightarrow\; C \# \textbf{if } B \textbf{ then } E_1 \textbf{ else } E_2 \;::\; \pi \,.$$

(6.4) If $C \in \langle \text{commlike}, \pi \rangle$ then

$$C \# 0 \;::\; \pi \,,$$

$$C \# 1 \;::\; \pi \,,$$

$$C \# 0.5 \;::\; \pi \,,$$

$$C \# \textbf{true} \;::\; \pi \,.$$

(6.5) If $C \in \langle \text{commlike}, \pi \rangle$ and $E_1, E_2 \in \langle \text{integer exp}, \pi \rangle$ (or $\langle \text{real exp}, \pi \rangle$ or $\langle \text{Boolean exp}, \pi \rangle$) then

$$C \# E_1 \,\&\, C \# E_2 \Rightarrow C \# E_1 + E_2 \;::\; \pi \,,$$

$$C \# E_1 \,\&\, C \# E_2 \Rightarrow C \# E_1 = E_2 \;::\; \pi \,,$$

$$C \# E_1 \,\&\, C \# E_2 \Rightarrow C \# E_1 \textbf{ and } E_2 \;::\; \pi \,.$$

(6.6) If $C \in \langle \text{commlike}, \ \pi \rangle$, $E \in \langle \text{Boolean exp}, \ [\pi \mid \iota\text{: integer exp}] \rangle$, and $\iota \notin \textbf{dom}(\pi)$, then

$$(\forall \iota\text{: } \textbf{integer exp})(C \# \iota \Rightarrow C \# E) \;\Rightarrow\; C \# (\forall \iota)\, E \;::\; \pi \,.$$

Then we define, for any expression-like phrase E, a subset $\mathcal{F}_{\text{explike}}(E)$ of the identifiers occurring free in E. Let $E \in \langle \text{explike}, \pi \rangle$. Then

$$\mathcal{F}_{\text{explike}}(E) \subseteq \{\iota \mid \iota \in \textbf{dom}(\pi) \text{ and } \pi(\iota) \leq \textbf{explike}\}$$

is the set of identifiers such that:

(a) If E is an identifier ι then

$$\mathcal{F}_{\text{explike}}(E) = \{\iota\}.$$

(b) If E is $E_1(E_2)$ then there must be one or more pairs θ_1, θ_2 of phrase types such that $\theta_2 \leq \textbf{explike}$, $E_1 \in \langle \theta_1 \rightarrow \theta_2, \pi \rangle$, and $E_2 \in \langle \theta_1, \pi \rangle$.

(i) If there is any such θ_1 for which $\theta_1 \not\leq \textbf{explike}$ then

$$\mathcal{F}_{\text{explike}}(E) \;=\; \mathcal{F}_{\text{explike}}(E_1) \,.$$

(ii) Otherwise,

$$\mathcal{F}_{\text{explike}}(E) \;=\; \mathcal{F}_{\text{explike}}(E_1) \;\cup\; \mathcal{F}_{\text{explike}}(E_2) \,.$$

(c) If E is $\lambda\iota\colon\theta.\, E_1$ then

$$\mathcal{F}_{\text{explike}}(E) \;=\; \mathcal{F}_{\text{explike}}(E_1) \;-\; \{\iota\}\;.$$

(d) If E is **if** B **then** E_1 **else** E_2 then

$$\mathcal{F}_{\text{explike}}(E) \;=\; \mathcal{F}_{\text{explike}}(B) \;\cup\; \mathcal{F}_{\text{explike}}(E_1) \;\cup\; \mathcal{F}_{\text{explike}}(E_2)\;.$$

(e) If E is **let** ι **be** E_1 **in** E_2 then there must be at least one pair θ_1, θ_2 of phrase types such that $\theta_2 \le \textbf{explike}$, $E_1 \in \langle\theta_1, \pi\rangle$, and $E_2 \in \langle\theta_2,\ [\pi \mid \iota\colon\theta_1]\rangle$.

(i) If there is any such θ_1 for which $\theta_1 \not\le \textbf{explike}$ then

$$\mathcal{F}_{\text{explike}}(E) \;=\; \mathcal{F}_{\text{explike}}(E_2) \;-\; \{\iota\}\;.$$

(ii) Otherwise,

$$\mathcal{F}_{\text{explike}}(E) \;=\; \mathcal{F}_{\text{explike}}(E_1) \;\cup\; (\mathcal{F}_{\text{explike}}(E_2) \;-\; \{\iota\})\;.$$

(f) If E is 0, 1, 0.5, or **true**, then

$$\mathcal{F}_{\text{explike}}(E) \;=\; \{\}\;.$$

(g) If E is $E_1 + E_2$, $E_1 = E_2$, or E_1 **and** E_2, then

$$\mathcal{F}_{\text{explike}}(E) \;=\; \mathcal{F}_{\text{explike}}(E_1) \;\cup\; \mathcal{F}_{\text{explike}}(E_2)\;.$$

(h) If E is $(\forall\iota)E_1$ then

$$\mathcal{F}_{\text{explike}}(E) \;=\; \mathcal{F}_{\text{explike}}(E_1) \;-\; \{\iota\}\;.$$

Finally, we give a general derived rule for rightside decomposition:

(6.7) Rightside Noninterference Decomposition. Suppose $C \in \langle\text{commlike}, \pi\rangle$, $E \in \langle\text{explike}, \pi\rangle$, and $\mathcal{F}_{\text{explike}}(E) = \{\iota_1, \ldots, \iota_n\}$. Then

$$C \mathrel{\#} \iota_1 \,\&\cdots\&\, C \mathrel{\#} \iota_n \;\Rightarrow\; C \mathrel{\#} E \;::\; \pi\;.$$

We omit the proof, which is similar in style and tedium to that in the previous section.

7. Assignment and Simple Variable Declaration

To treat assignment, we begin with an equivalence that defines "good variable" specifications: If $V \in \langle\tau\ \tau\ \text{var}, \pi\rangle$ and ι and ι' are distinct identifiers not belonging to $\textbf{dom}(\pi)$, then

$$\begin{aligned}\textbf{gv}_\tau(V) \;\equiv_{\text{spec},\pi}\; & (\forall\iota\colon\tau\ \textbf{exp})(\forall\iota'\colon\tau\ \textbf{exp} \to \textbf{Boolean exp}) \\ & (V \mathrel{\#} \iota' \;\Rightarrow\; \{\iota'(\iota)\}\ V := \iota\ \{\iota'(V)\})\;.\end{aligned} \tag{7.1}$$

From this equivalence it is straightforward to derive an inference rule for assignment statements:

(7.2) Assignment. Let $V \in \langle \tau\ \tau\ \text{var},\ \pi\rangle$, $E \in \langle \tau\ \text{exp},\ \pi\rangle$, $P \in \langle \text{Boolean exp}, [\pi \mid \iota{:}\, \tau\ \text{exp}]\rangle$, and $\mathcal{F}_{\text{explike}}(P) - \{\iota\} = \{\iota_1, \ldots, \iota_n\}$. By quantifier removal, using the substitution $\iota, \iota' \to E, \lambda\iota{:}\, \tau\ \mathbf{exp}.\ P$, we have

$$\begin{aligned}&(\forall\iota{:}\, \tau\ \mathbf{exp})(\forall\iota'{:}\, \tau\ \mathbf{exp} \to \mathbf{Boolean\ exp})\\ &\qquad (V \# \iota' \ \Rightarrow\ \{\iota'(\iota)\}\ V := \iota\ \{\iota'(V)\})\\ &\Rightarrow\ (V \# \lambda\iota{:}\, \tau\ \mathbf{exp}.\ P\\ &\qquad \Rightarrow\ \{(\lambda\iota{:}\, \tau\ \mathbf{exp}.\ P)(E)\}\ \ V := E\ \{(\lambda\iota{:}\, \tau\ \mathbf{exp}.\ P)(V)\})\ ::\ \pi\ .\end{aligned}$$

Next beta reduction and rightside noninterference decomposition give

$$\begin{aligned}&(\forall\iota{:}\, \tau\ \mathbf{exp})(\forall\iota'{:}\, \tau\ \mathbf{exp} \to \mathbf{Boolean\ exp})\\ &\qquad (V \# \iota' \Rightarrow \{\iota'(\iota)\}\ V := \iota\ \{\iota'(V)\})\\ &\Rightarrow\ (V \# \iota_1 \,\&\cdots\&\, V \# \iota_n \Rightarrow \{P|_{\iota\to E}\}\ V := E\ \{P|_{\iota\to V}\})\ ::\ \pi\end{aligned}$$

since $\mathcal{F}_{\text{explike}}(\lambda\iota{:}\, \tau\ \mathbf{exp}.\ P) = \{\iota_1, \ldots, \iota_n\}$. Then equivalence (7.1) gives the conclusion

$$\mathbf{gv}_\tau(V)\ \&\ V \# \iota_1 \,\&\cdots\&\, V \# \iota_n\ \Rightarrow\ \{P|_{\iota\to E}\}\ V := E\ \{P|_{\iota\to V}\}\ ::\ \pi\ .$$

In the special case where $V = \iota$, this rule reduces to the simple assignment rule (R24) in [8], which says that Hoare-like reasoning about an assignment to ι requires ι to be a good variable that does not interfere with any other identifier with a free expression-like occurrence in P. In the simplest case, where ι is declared to be a simple variable, these assumptions will be assured by its declaration. Simple variable declarations are described by the following family of axioms:

(7.3) Simple Variable Declaration Axiom. Let π be a type assignment that maps $e_1, \ldots, e_m$ into **explike**, $c_1, \ldots, c_n$ into **commlike**, p, q into **Boolean exp**, and h into $\tau\ \tau$ **var** $\to$ **comm**. Then

$$\begin{aligned}&(\forall x{:}\, \tau\ \tau\ \mathbf{var})\ (\mathbf{gv}_\tau(x)\ \&\ x \# e_1 \,\&\cdots\&\, x \# e_m\ \&\ c_1 \# x \,\&\cdots\&\, c_n \# x\\ &\qquad\qquad\qquad \Rightarrow\ \{p\}\ h(x)\ \{q\})\\ &\Rightarrow \{p\}\ \mathbf{new}\ \tau\ \mathbf{var}\ x\ \mathbf{in}\ h(x)\ \{q\}\ ::\ \pi\ .\end{aligned}$$

From this axiom we can derive a less abstract rule akin to (R25) in [8]:

(7.4) Simple Variable Declaration Rule. Suppose $S \in \langle \text{spec},\ \pi\rangle$, $E_1, \ldots, E_m \in \langle \text{explike}, \pi\rangle$, $C_1, \ldots, C_n \in \langle \text{commlike}, \pi\rangle$, $P, Q \in \langle \text{Boolean exp}, \pi\rangle$, $B \in \langle \text{comm}, [\pi \mid \iota{:}\, \tau\ \tau\ \text{var}]\rangle$, and $\iota \notin \mathbf{dom}(\pi)$ are such that

$$\begin{aligned}&S\ \&\ \mathbf{gv}_\tau(\iota)\ \&\ \iota \# E_1 \,\&\cdots\&\, \iota \# E_m\ \&\ C_1 \# \iota \,\&\cdots\&\, C_n \# \iota\\ &\qquad \Rightarrow \{P\}\ B\ \{Q\}\ ::\ [\pi \mid \iota{:}\, \tau\ \tau\ \mathbf{var}]\ .\end{aligned}$$

By quantifier introduction we have

$$\begin{aligned}&S \Rightarrow (\forall\iota{:}\, \tau\ \tau\ \mathbf{var})\\ &\qquad \mathbf{gv}_\tau(\iota)\ \&\ \iota \# E_1 \,\&\cdots\&\, \iota \# E_m\ \&\ C_1 \# \iota \,\&\cdots\&\, C_n \# \iota\\ &\qquad\quad \Rightarrow \{P\}\ B\ \{Q\}\ ::\ \pi\ .\end{aligned}$$

In Axiom (7.3) we rename x to be ι (which can be assumed without loss of generality to be distinct from the identifiers occurring free in the axiom), and use the free substitution

$$e_1, \ldots, e_m \to E_1, \ldots, E_m$$

$$c_1, \ldots, c_n \to C_1, \ldots, C_n$$

$$p, q \to P, Q$$

$$h \to \lambda\iota{:}\, \tau\ \tau\ \mathbf{var}.\, B$$

to obtain

$$\begin{aligned}&(\forall\iota{:}\, \tau\ \tau\ \mathbf{var})\ (\mathbf{gv}_\tau(\iota)\ \&\ \iota \# E_1\ \& \cdots \&\ \iota \# E_m\ \&\ C_1 \# \iota\ \& \cdots \&\ C_n \# \iota \\ &\qquad\qquad\qquad \Rightarrow\ \{P\}\ (\lambda\iota{:}\, \tau\ \tau\ \mathbf{var}.\, B)(\iota)\ \{Q\}) \\ &\Rightarrow \{P\}\ \mathbf{new}\ \tau\ \mathbf{var}\ \iota\ \mathbf{in}\ (\lambda\iota{:}\, \tau\ \tau\ \mathbf{var}.\, B)(\iota)\ \{Q\}\ ::\ \pi\ .\end{aligned}$$

Then, after beta-reducing the two redexes and using modus ponens, we obtain the conclusion

$$S \Rightarrow \{P\}\ \mathbf{new}\ \tau\ \mathbf{var}\ \iota\ \mathbf{in}\ B\ \{Q\}\ ::\ \pi\ .$$

At this point, the reader may wonder if it is possible for a variable not to be good. An example is provided by the conditional variable

$$\mathbf{if}\ m = 1\ \mathbf{then}\ m\ \mathbf{else}\ n\ ,$$

where m and n are good integer variables. For a state where $m = 1$ and $n = 2$, the assignment

$$(\mathbf{if}\ m = 1\ \mathbf{then}\ m\ \mathbf{else}\ n) := 3$$

produces a state in which

$$(\mathbf{if}\ m = 1\ \mathbf{then}\ m\ \mathbf{else}\ n) = 2\ .$$

Thus $\mathbf{gv}_{\text{integer}}(\mathbf{if}\ m = 1\ \mathbf{then}\ m\ \mathbf{else}\ n)$ is false. (Another example of a bad variable, discussed in [8], is a self-subscripting array such as $X(X(1))$.)

The seriousness of the bad variable problem arises from the fact that such a variable can be used as an actual parameter in a procedure call, where call by name implies that its abnormal behavior is inherited by the corresponding formal parameter.

8. let Commands and Function Procedures

We now consider the **let** construction, which can be used to define nonrecursive procedures. First we derive a rule for commands of the form **let** ι **be** M **in** B.

(8.1) **let** Commands. Suppose

$$\iota \notin \mathbf{dom}(\pi)$$

$$S \in \langle \text{spec}, \pi \rangle$$

$$S_{\text{proc}} \in \langle \text{spec}, \ [\pi \mid \iota{:}\, \theta] \rangle$$

$$P, Q \in \langle \text{Boolean exp}, \pi \rangle$$

$$B \in \langle \text{comm}, \ [\pi \mid \iota{:}\, \theta] \rangle$$

$$M \in \langle \theta, \pi \rangle$$

are such that

$$S \Rightarrow (S_{\text{proc}}|_{\iota \to M}) \ :: \ \pi \ , \tag{a}$$

$$S \ \& \ S_{\text{proc}} \Rightarrow \{P\} \ B \ \{Q\} \ :: \ [\pi \mid \iota{:}\, \theta] \ . \tag{b}$$

From (b), reverse beta reduction gives

$$S \ \& \ S_{\text{proc}} \Rightarrow \{P\} \ (\lambda \iota{:}\, \theta.\ B)(\iota) \ \{Q\} \ :: \ [\pi \mid \iota{:}\, \theta] \ ,$$

and free substitution gives

$$S \,\&\, (S_{\text{proc}}|_{\iota \to M}) \Rightarrow \{P\} \ (\lambda \iota{:}\, \theta.\, B)(M) \ \{Q\} \ :: \ \pi$$

since ι does not occur free in S, P, Q, or the lambda expression. Then modus ponens with (a) gives

$$S \Rightarrow \{P\} \ (\lambda \iota{:}\, \theta.\, B)(M) \ \{Q\} \ :: \ \pi \ ,$$

and the equivalence (3.8) defining **let** gives the conclusion

$$S \Rightarrow \{P\} \ \mathbf{let} \ \iota \ \mathbf{be} \ M \ \mathbf{in} \ B \ \{Q\} \ :: \ \pi \ .$$

From this rule, one can derive more specific rules for nonrecursive proper and function procedure declarations. We limit our discussion to the latter, since the former is less interesting than the recursive case that will be considered later. For function procedures, however, only the nonrecursive case can be treated by specification logic. The following is a derivation of a rule similar to (R36′) in [8]. (The notation $\pi \rceil S$ denotes the restriction of π to S.)

(8.2) Nonrecursive Function Procedure Declaration. Suppose (for some $n \geq 0$)

$$\theta = \theta_1 \to (\cdots \to (\theta_n \to \tau\ \mathbf{exp})\cdots)$$

$\iota,\ \iota_1,\ \ldots,\ \iota_n$ are distinct identifiers

$\iota \notin \mathbf{dom}(\pi)$

$\pi' = \pi\rceil(\mathbf{dom}(\pi) - \{\iota_1, \ldots, \iota_n\})$

$\pi'' = [\pi' \mid \iota_1{:}\, \theta_1 \mid \cdots \mid \iota_n{:}\, \theta_n]$

$S \in \langle \mathrm{spec}, \pi\rangle$

$P, Q \in \langle \text{Boolean exp}, \pi\rangle$

$B_{\mathrm{proc}} \in \langle \tau\ \mathrm{exp}, \pi''\rangle$

$B \in \langle \mathrm{comm}, [\pi \mid \iota{:}\, \theta]\rangle$

$\{\iota''_1, \ldots, \iota''_m\} = \mathcal{F}_{\mathrm{explike}}(B_{\mathrm{proc}}) - \{\iota_1, \ldots, \iota_n\} \subseteq \mathbf{dom}(\pi')$

$\iota_{\mathrm{c}} \notin \{\iota\} \cup \mathbf{dom}(\pi')$

$$\begin{array}{rcl} S_{\mathrm{proc}} & = & (\forall \iota_1{:}\, \theta_1)\cdots(\forall \iota_n{:}\, \theta_n)\ \{\iota(\iota_1)\cdots(\iota_n) = B_{\mathrm{proc}}\} \\ & & \&\ (\forall \iota_{\mathrm{c}}{:}\, \mathbf{commlike})(\iota_{\mathrm{c}}\ \#\ \iota''_1\ \&\cdots\&\ \iota_{\mathrm{c}}\ \#\ \iota''_n \Rightarrow \iota_{\mathrm{c}}\ \#\ \iota) \\ & \in & \langle \mathrm{spec},\ [\pi' \mid \iota{:}\, \theta]\rangle\ , \end{array}$$

and that

$$S\ \&\ S_{\mathrm{proc}} \Rightarrow \{P\}\ B\ \{Q\}\ ::\ [\pi \mid \iota{:}\, \theta]\ . \tag{a}$$

Let M be $\lambda\iota_1{:}\, \theta_1. \ldots\ \lambda\iota_n{:}\, \theta_n.\, B_{\mathrm{proc}} \in \langle \theta, \pi'\rangle$. When M is substituted for ι in S_{proc}, there is no renaming since $\iota_1, \ldots, \iota_n$ and ι_{c} do not occur free in M, and B_{proc} and $\iota''_1, \ldots, \iota''_m$ remain unchanged since they do not contain free occurrences of ι. Moreover, $M(\iota_1)\cdots(\iota_n)$ beta-reduces to B_{proc}. Thus $S_{\mathrm{proc}}|_{\iota\to M} \in \langle \mathrm{spec}, \pi'\rangle$ is

$$\begin{array}{l} (\forall \iota_1{:}\, \theta_1)\cdots(\forall \iota_n{:}\, \theta_n)\ \{B_{\mathrm{proc}} = B_{\mathrm{proc}}\}\ \&\ (\forall \iota_{\mathrm{c}}{:}\ \mathbf{commlike}) \\ \qquad (\iota_{\mathrm{c}}\ \#\ \iota''_1\ \&\cdots\&\ \iota_{\mathrm{c}}\ \#\ \iota''_m \Rightarrow \iota_{\mathrm{c}}\ \#\ \lambda\iota_1{:}\, \theta_1.\ \ldots\ \lambda\iota_n{:}\, \theta_n.\, B_{\mathrm{proc}}), \end{array}$$

which is universal for π' since $B_{\mathrm{proc}} = B_{\mathrm{proc}}$ is a mathematical fact and

$$\begin{array}{l} \iota_{\mathrm{c}}\ \#\ \iota''_1\ \&\cdots\&\ \iota_{\mathrm{c}}\ \#\ \iota''_m\ \Rightarrow\ \iota_{\mathrm{c}}\ \#\ \lambda\iota_1{:}\, \theta_1.\ \ldots\ \lambda\iota_n{:}\, \theta_n.\, B_{\mathrm{proc}} \\ \qquad ::\ [\pi' \mid \iota_{\mathrm{c}}{:}\, \mathbf{commlike}] \end{array}$$

is a consequence of rightside noninterference decomposition along with the fact that $\mathcal{F}_{\mathrm{explike}}(\lambda\iota_1{:}\, \theta_1.\ \ldots\ \lambda\iota_n{:}\, \theta_n.\, B_{\mathrm{proc}}) = \{\iota''_1, \ldots, \iota''_m\}$. Broadening π' to π and adding the assumption S gives

$$S \Rightarrow (S_{\mathrm{proc}}|_{\iota\to M})\ ::\ \pi\ . \tag{b}$$

Then from (a) and (b) the rule (8.1) for **let** commands can be used to infer the conclusion

$$S \Rightarrow \{P\}\ \mathbf{let}\ \iota\ \mathbf{be}\ \lambda\iota_1{:}\, \theta_1.\ \ldots\ \lambda\iota_n{:}\, \theta_n.\, B_{\mathrm{proc}}\ \mathbf{in}\ B\ \{Q\}\ ::\ \pi\ .$$

9. Recursion

To encompass recursion, we must give basic rules for the operator **rec** and the constant $\perp$. The behavior of $\perp$ is described by

(9.1) Nontermination of $\perp$. If $P, Q \in \langle \text{Boolean exp}, \pi \rangle$ then

$$\{P\}\ \perp_{\text{comm}}\ \{Q\}\ ::\ \pi\ ,$$

along with equivalences (3.4) and (3.5), and inference rule (5.5).

Before giving a rule for the recursion operator **rec**, we must define, for any specification S, two subsets of the identifiers occurring free in S. If $S \in \langle \text{spec}, \pi \rangle$, we define $\mathcal{F}_-(S)$ and $\mathcal{F}_+(S) \subseteq \mathbf{dom}(\pi)$ by

$$\begin{array}{ll}
\mathcal{F}_-(\{P\}\ C\ \{Q\}) = \{\} & \mathcal{F}_+(\{P\}\ C\ \{Q\}) = \mathcal{F}_{\text{commlike}}(C) \\
\mathcal{F}_-(\{P\}) = \{\} & \mathcal{F}_+(\{P\}) = \{\} \\
\mathcal{F}_-(C \mathrel{\#} E) = \{\} & \mathcal{F}_+(C \mathrel{\#} E) = \mathcal{F}_{\text{commlike}}(C) \\
\mathcal{F}_-(S \mathbin{\&} S') = \mathcal{F}_-(S) \cup \mathcal{F}_-(S') & \mathcal{F}_+(S \mathbin{\&} S') = \mathcal{F}_+(S) \cup \mathcal{F}_+(S') \\
\mathcal{F}_-(S \Rightarrow S') = \mathcal{F}_+(S) \cup \mathcal{F}_-(S') & \mathcal{F}_+(S \Rightarrow S') = \mathcal{F}_-(S) \cup \mathcal{F}_+(S') \\
\mathcal{F}_-((\forall \iota\colon \theta)S) = \mathcal{F}_-(S) - \{\iota\} & \mathcal{F}_+((\forall \iota\colon \theta)S) = (\mathcal{F}_+(S) \cup \mathcal{F}_-(S)) - \{\iota\}
\end{array}$$

These sets reflect the semantic concept of continuity:

(1) If $\iota \notin \mathcal{F}_-(S)$ then the meaning of S, ordered by **true** $\sqsubseteq$ **false**, is continuous in the meaning of ι.

(2) If $\iota \notin \mathcal{F}_+(S)$ then the meaning of S, ordered by **false** $\sqsubseteq$ **true**, is continuous in the meaning of ι.

The asymmetric equations for quantified specifications reflect the fact that universal quantification preserves the first kind of continuity but not the second.

Consider $F \in \langle \rho \to \rho, \pi \rangle$. Suppose some property holds for $\perp_\rho$ and, whenever this property holds for ι, it holds for $F(\iota)$. Then the property holds for each member of the sequence $\perp_\rho,\ F(\perp_\rho),\ F(F(\perp_\rho)),\ \ldots$. Moreover, **rec** F can be regarded as the limit of this sequence, so that if the property is appropriately continuous—in the sense of (1) above—it will hold for **rec** F.

This reasoning is formalized by the following rule for **rec**, in which the property is expressed by a specification S regarded as a function of ι:

(9.2) Recursion. Suppose $F \in \langle \rho \to \rho, \pi \rangle$, $S \in \langle \text{spec}, [\pi \mid \iota\colon \rho] \rangle$, $\iota \notin \mathbf{dom}(\pi)$, and $\iota \notin \mathcal{F}_-(S)$. Then

$$(S|_{\iota \to \perp_\rho}) \mathbin{\&} (\forall \iota\colon \rho)(S \Rightarrow (S|_{\iota \to F(\iota)})) \Rightarrow (S|_{\iota \to \mathbf{rec}\, F})\ ::\ \pi\ .$$

It is illuminating to give an example showing the necessity of the condition $\iota \notin \mathcal{F}_{-}(S)$. Such an example can be developed from the fact that certain recursive procedures always terminate even though none of their "finite approximations" always terminate. Under the type assignment $[p\colon \mathbf{integer\ exp} \to \mathbf{comm} \mid x\colon \mathbf{integer\ exp}]$:

$\{\mathbf{true}\}p(x)\{\mathbf{false}\}$ specifies p does not terminate for x.

$\{\mathbf{true}\}p(x)\{\mathbf{false}\} \Rightarrow \{\mathbf{false}\}$ specifies that p terminates for x.

$(\forall x\colon \mathbf{integer\ exp})(\{\mathbf{true}\}p(x)\{\mathbf{false}\} \Rightarrow \{\mathbf{false}\})$ specifies that p always terminates.

$S = (\forall x\colon \mathbf{integer\ exp})(\{\mathbf{true}\}p(x)\{\mathbf{false}\} \Rightarrow \{\mathbf{false}\}) \Rightarrow \{\mathbf{false}\}$ specifies that p does not always terminate.

Even though S holds for all finite approximations to p, it may fail to hold for the limit. This is possible since $p \in \mathcal{F}_{-}(S)$ and, in fact, the meaning of S, ordered by $\mathbf{true} \sqsubseteq \mathbf{false}$, is not continuous in the meaning of p.

In formalizing this example, it will be convenient to use the following derived rule about the operation $\Rightarrow$:

(9.3) Reversal. Suppose S_1, S_2, $S_3 \in \langle \text{spec}, \pi \rangle$ and

$$S_1 \Rightarrow S_2 \ :: \ \pi \ .$$

From (3.25)

$$(S_2 \Rightarrow S_3) \Rightarrow (S_2 \Rightarrow S_3) \ :: \ \pi \ ,$$

so by (3.26) and (3.20)

$$S_2 \Rightarrow ((S_2 \Rightarrow S_3) \Rightarrow S_3) \ :: \ \pi \ ,$$

and by modus ponens

$$S_1 \Rightarrow ((S_2 \Rightarrow S_3) \Rightarrow S_3) \ :: \ \pi \ .$$

Then (3.26) and (3.20) give the conclusion

$$(S_2 \Rightarrow S_3) \Rightarrow (S_1 \Rightarrow S_3) \ :: \ \pi \ .$$

It will also simplify the argument to assume, for the duration of this example, that the values of integer expressions range over the natural numbers (excluding negative numbers). Thus $x + 1 = 0$ **implies false** is a mathematical fact for $[x\colon \mathbf{integer\ exp}]$.

Let

$$\begin{aligned} F = \ & \lambda p\colon \mathbf{integer\ exp} \to \mathbf{comm}.\, \lambda x\colon \mathbf{integer\ exp}. \\ & \quad \mathbf{if}\ x = 0\ \mathbf{then\ skip\ else}\ p(x-1) \end{aligned}$$

and

$$\pi = [p\colon \mathbf{integer\ exp} \to \mathbf{comm} \mid x\colon \mathbf{integer\ exp}] \ .$$

Then

$$\{\mathbf{false}\}\mathbf{skip}\{\mathbf{false}\} \ :: \ \pi \tag{4.17}$$

$$\{x+1=0\}\mathbf{skip}\{\mathbf{false}\} \ :: \ \pi \tag{4.13}$$

$$\{\mathbf{true}\}p(x)\{\mathbf{false}\} \Rightarrow \{\mathbf{true}\}p((x+1)-1)\{\mathbf{false}\} \ :: \ \pi \tag{4.1}$$

$$\{\mathbf{true}\}p(x)\{\mathbf{false}\} \Rightarrow \{\neg x+1=0\}\ p((x+1)-1)\{\mathbf{false}\} \ :: \ \pi \tag{4.13}$$

$$\{\mathbf{true}\}p(x)\{\mathbf{false}\} \Rightarrow \{\mathbf{true}\}F(p)(x+1)\{\mathbf{false}\} \ :: \ \pi \tag{4.16, 3.1}$$

$$\begin{aligned}&(\{\mathbf{true}\}F(p)(x+1)\{\mathbf{false}\} \Rightarrow \{\mathbf{false}\})\\ &\qquad \Rightarrow (\{\mathbf{true}\}p(x)\{\mathbf{false}\} \Rightarrow \{\mathbf{false}\}) \ :: \ \pi\end{aligned} \tag{9.3}$$

$$\begin{aligned}&(\forall x\colon \mathbf{integer\ exp})(\{\mathbf{true}\}F(p)(x)\{\mathbf{false}\} \Rightarrow \{\mathbf{false}\})\\ &\qquad \Rightarrow (\{\mathbf{true}\}F(p)(x+1)\{\mathbf{false}\} \Rightarrow \{\mathbf{false}\}) \ :: \ \pi\end{aligned} \tag{4.8}$$

$$\begin{aligned}&(\forall x\colon \mathbf{integer\ exp})(\{\mathbf{true}\}F(p)(x)\{\mathbf{false}\} \Rightarrow \{\mathbf{false}\})\\ &\qquad \Rightarrow (\{\mathbf{true}\}p(x)\{\mathbf{false}\} \Rightarrow \{\mathbf{false}\}) \ :: \ \pi\end{aligned} \tag{4.6}$$

$$\begin{aligned}&(\forall x\colon \mathbf{integer\ exp})(\{\mathbf{true}\}F(p)(x)\{\mathbf{false}\} \Rightarrow \{\mathbf{false}\})\\ &\qquad \Rightarrow (\forall x\colon \mathbf{integer\ exp})(\{\mathbf{true}\}p(x)\{\mathbf{false}\} \Rightarrow \{\mathbf{false}\})\\ &\qquad :: \ [p\colon \mathbf{integer\ exp} \to \mathbf{comm}]\end{aligned} \tag{4.7}$$

$$S \Rightarrow \left(S|_{p\to F(p)}\right) \ :: \ [p\colon \mathbf{integer\ exp} \to \mathbf{comm}] \tag{9.3}$$

$$(\forall p\colon \mathbf{integer\ exp} \to \mathbf{comm})(S \Rightarrow (S|_{p\to F(p)})) \ :: \ [\,] \tag{4.7}$$

$$\{\mathbf{true}\} \perp_{\mathrm{comm}} \mathbf{false}\} \ :: \ [\,] \tag{9.1}$$

$$\{\mathbf{true}\} \perp_{\mathrm{integer\ exp}\to\mathrm{comm}} (x)\{\mathbf{false}\} \ :: \ [x\colon \mathbf{integer\ exp}] \tag{3.4}$$

$$\begin{aligned}&(\forall x\colon \mathbf{integer\ exp})(\{\mathbf{true}\} \perp_{\mathrm{integer\ exp}\to\mathrm{comm}} (x)\{\mathbf{false}\} \Rightarrow \{\mathbf{false}\})\\ &\qquad \Rightarrow (\{\mathbf{true}\} \perp_{\mathrm{integer\ exp}\to\mathrm{comm}} (x)\,\{\mathbf{false}\} \Rightarrow \ \{\mathbf{false}\})\\ &\qquad :: \ [x\colon \mathbf{integer\ exp}]\end{aligned} \tag{4.8}$$

$$\begin{aligned}&\{\mathbf{true}\} \perp_{\mathrm{integer\ exp}\to\mathrm{comm}} (x)\{\mathbf{false}\} \Rightarrow (S|_{p\to\perp_{\mathrm{integer\ exp}\to\mathrm{comm}}})\\ &\qquad :: \ [x\colon \mathbf{integer\ exp}]\end{aligned} \tag{3.26, 3.20}$$

$$S|_{p\to\perp_{\mathrm{integer\ exp}\to\mathrm{comm}}} \ :: \ [x\colon \mathbf{integer\ exp}] \tag{4.6}$$

$$S|_{p\to\perp_{\mathrm{integer\ exp}\to\mathrm{comm}}} \ :: \ [\,] \tag{4.3}$$

Now consider the recursion rule, taking ρ to be **integer exp** $\to$ **comm**, π to be the empty type assignment $[\,]$, ι to be p, and S and F as above. In the absence of the restriction $p \notin \mathcal{F}_{-}(S)$, one could infer

$$S|_{p\to\mathbf{rec}\ F} \ :: \ [\,]$$

But this is false, since in fact **rec** F always terminates.

10. letrec Commands and Proper Procedures

From the rules in the previous section, we can derive a rule for **letrec** commands and a more specific rule for proper procedure declarations.

(10.1) **letrec** Commands. Suppose

$$\begin{aligned}
&\iota \notin \mathbf{dom}(\pi)\\
&S \in \langle \text{spec}, \pi \rangle\\
&S_{\text{proc}} \in \langle \text{spec}, [\pi \mid \iota\colon \rho] \rangle\\
&P, Q \in \langle \text{Boolean exp}, \pi \rangle\\
&B \in \langle \text{comm}, [\pi \mid \iota\colon \rho] \rangle\\
&M \in \langle \rho, [\pi \mid \iota\colon \rho] \rangle\\
&\iota \notin \mathcal{F}_{-}(S_{\text{proc}})
\end{aligned}$$

are such that

$$S \Rightarrow (S_{\text{proc}}|_{\iota \to \perp_\rho}) \;::\; \pi \tag{a}$$

$$S \;\&\; S_{\text{proc}} \Rightarrow (S_{\text{proc}}|_{\iota \to M}) \;::\; [\pi \mid \iota : \rho] \tag{b}$$

$$S \;\&\; S_{\text{proc}} \Rightarrow \{P\}\; B\; \{Q\} \;::\; [\pi \mid \iota : \rho]\;. \tag{c}$$

From (b), by backwards beta reduction, we have

$$S \;\&\; S_{\text{proc}} \;\Rightarrow\; \left(S_{\text{proc}}|_{\iota \to (\lambda\iota:\rho.M)(\iota)}\right) \;::\; [\pi \mid \iota : \rho]\;,$$

so that introducing a quantifier gives

$$S \Rightarrow (\forall \iota\colon \rho)(S_{\text{proc}} \;\Rightarrow\; (S_{\text{proc}}|_{\iota \to (\lambda\iota:\rho.M)(\iota)})) \;::\; \pi\;. \tag{d}$$

From the recursion rule (9.2), replacing S by S_{proc} and F by $\lambda\iota\colon\rho.\, M$, we get

$$\begin{aligned}
&(S_{\text{proc}}|_{\iota \to \perp_\rho}) \;\&\; (\forall \iota\colon \rho)(S_{\text{proc}} \Rightarrow (S_{\text{proc}}|_{\iota \to (\lambda\iota:\rho.M)(\iota)}))\\
&\qquad\qquad \Rightarrow (S_{\text{proc}}|_{\iota \to \mathbf{rec}\ \lambda\iota:\rho.M}) \;::\; \pi\;,
\end{aligned}$$

which, by modus ponens with (a) and (d), gives

$$S \Rightarrow (S_{\text{proc}}|_{\iota \to \mathbf{rec}\ \lambda\iota:\rho.M}) \;::\; \pi\;. \tag{e}$$

By backwards beta reduction in (c), we have

$$S \;\&\; S_{\text{proc}} \Rightarrow \{P\}\; (\lambda\iota\colon \rho.\, B)(\iota)\; \{Q\} \;::\; [\pi \mid \iota\colon \rho]\;.$$

Then, since ι does not occur free in S, P, $\lambda\iota\colon\rho.\, B$, or Q, free substitution gives

$$S \;\&\; (S_{\text{proc}}|_{\iota \to \mathbf{rec}\ \lambda\iota:\rho.M}) \\ \qquad \Rightarrow \{P\}\ (\lambda\iota\colon\rho.\ B)(\mathbf{rec}\ \lambda\iota\colon\rho.\ M)\ \{Q\}\ ::\ \pi\ ,$$

and modus ponens with (e), along with the equivalence (3.9) defining **letrec**, gives the conclusion

$$S \Rightarrow \{P\}\ \mathbf{letrec}\ \iota\colon\rho\ \mathbf{be}\ M\ \mathbf{in}\ B\ \{Q\}\ ::\ \pi\ .$$

From (10.1) we can derive the following more specific rule, which is similar to (R26) in [8]:

(10.2) Proper Procedure Declaration. Suppose that (for some $n, k \geq 0$)

$$\begin{aligned}
&\rho = (\theta_1 \to \cdots (\theta_n \to \mathbf{comm}) \cdots) \\
&\iota, \iota_1, \ldots, \iota_n, \iota'_1, \ldots, \iota'_k \text{ are distinct identifiers} \\
&\iota \notin \mathbf{dom}(\pi) \\
&\pi' = \pi\rceil\,(\mathbf{dom}(\pi) - \{\iota_1, \ldots \iota_n, \iota'_1, \ldots \iota'_k\}) \\
&\pi'' = [\pi' \mid \iota_1\colon \theta_1 \mid \cdots \mid \iota_n\colon \theta_n] \\
&\pi''' = [\pi'' \mid \iota'_1\colon \theta'_1 \mid \cdots \mid \iota'_k\colon \theta'_k] \\
&S_1 \in \langle \text{spec}, \pi' \rangle \\
&S_{\text{pa}} \in \langle \text{spec}, \pi''' \rangle \\
&S_2 \in \langle \text{spec}, \pi \rangle \\
&P_{\text{proc}}, Q_{\text{proc}} \in \langle \text{Boolean exp}, \pi''' \rangle \\
&P, Q \in \langle \text{Boolean exp}, \pi \rangle \\
&B_{\text{proc}} \in \langle \text{comm}, [\pi'' \mid \iota\colon \rho] \rangle \\
&B \in \langle \text{comm}, [\pi \mid \iota\colon \rho] \rangle \\
&\{\iota''_1, \ldots, \iota''_m\} = \mathcal{F}_{\text{commlike}}(B_{\text{proc}}) - \{\iota, \iota_1, \ldots, \iota_n\} \subseteq \mathbf{dom}(\pi') \\
&\iota_{\text{e}} \notin \{\iota\}\ \cup\ \mathbf{dom}(\pi') \\
&\begin{aligned}
S_{\text{proc}} \;\;&=\;\; (\forall\iota_1\colon \theta_1) \cdots (\forall\iota_n\colon \theta_n)(\forall\iota'_1\colon \theta'_1) \cdots (\forall\iota'_k\colon \theta_k) \\
&\qquad\qquad (S_{\text{pa}} \Rightarrow \{P_{\text{proc}}\}\ \iota(\iota_1) \cdots (\iota_n)\ \{Q_{\text{proc}}\}) \\
&\qquad \&\ (\forall\iota_{\text{e}}\colon\ \mathbf{explike})(\iota''_1 \# \iota_{\text{e}}\ \& \cdots \&\ \iota''_m \# \iota_{\text{e}} \Rightarrow \iota \# \iota_{\text{e}}) \\
&\in\;\; \langle \text{spec}, [\pi' \mid \iota\colon \rho] \rangle
\end{aligned}
\end{aligned}$$

and that

$$S_1 \;\&\; S_{\text{proc}} \;\&\; S_{\text{pa}} \Rightarrow \{P_{\text{proc}}\}\ B_{\text{proc}}\ \{Q_{\text{proc}}\}\ ::\ [\pi''' \mid \iota\colon \rho] \tag{a}$$

$$S_2 \;\&\; S_{\text{proc}}\ \Rightarrow \{P\}\ B\ \{Q\}\ ::\ [\pi \mid \iota\colon \rho]\ . \tag{b}$$

Let M be $\lambda\iota_1\colon \theta_1.\ \ldots\ \lambda\iota_n\colon \theta_n.\ B_{\text{proc}} \in \langle \rho, [\pi' \mid \iota\colon \rho] \rangle$. When M is substituted for ι in S_{proc}, there is no renaming since $\iota_1, \ldots, \iota_n$, $\iota'_1, \ldots, \iota'_k$, and ι_{e} do not occur free in M, and S_{pa}, P_{proc}, Q_{proc}, $\iota''_1, \ldots, \iota''_m$ remain unchanged since they do not contain free occurrences of ι. Moreover, $M(\iota_1) \cdots (\iota_n)$ beta-reduces to B_{proc}. Thus $S_{\text{proc}}|_{\iota \to M} \in \langle \text{spec}, [\pi' \mid \iota\colon \rho] \rangle$ is

$$\begin{aligned}&(\forall\iota_1{:}\,\theta_1)\cdots(\forall\iota_n{:}\,\theta_n)(\forall\iota'_1{:}\,\theta'_1)\cdots(\forall\iota'_k{:}\,\theta_k)\\&\qquad(S_{\text{pa}}\Rightarrow\{P_{\text{proc}}\}\ B_{\text{proc}}\ \{Q_{\text{proc}}\})\\&\ \&\ (\forall\iota_{\text{e}}{:}\ \mathbf{explike})\\&\qquad(\iota''_1\ \#\ \iota_{\text{e}}\ \&\cdots\&\ \iota''_m\ \#\ \iota_{\text{e}}\ \Rightarrow\ \lambda\iota_1{:}\,\theta_1.\ \ldots\ \lambda\iota_n{:}\,\theta_n.\,B_{\text{proc}}\ \#\ \iota_{\text{e}})\ .\end{aligned}$$

By introducing quantifiers into (a), we have

$$\begin{aligned}&S_1\ \&\ S_{\text{proc}}\Rightarrow(\forall\iota_1{:}\,\theta_1)\cdots(\forall\iota_n{:}\,\theta_n)(\forall\iota'_1{:}\,\theta'_1)\cdots(\forall\iota'_k{:}\,\theta'_k)\\&\qquad(S_{\text{pa}}\Rightarrow\{P_{\text{proc}}\}\ B_{\text{proc}}\ \{Q_{\text{proc}}\})\ ::\ [\pi'\mid\iota{:}\,\rho]\ .\end{aligned}\tag{c}$$

Since $\mathcal{F}_{\text{commlike}}(\lambda\iota_1{:}\,\theta_1.\ \ldots\ \lambda\iota_n{:}\,\theta_n.\,B_{\text{proc}})\subseteq\{\iota''_1,\ldots,\iota''_m,\iota\}$, leftside noninterference decomposition gives

$$\begin{aligned}&\iota''_1\ \#\ \iota_{\text{e}}\ \&\ \ldots\ \&\ \iota''_m\ \#\ \iota_{\text{e}}\ \&\ \iota\ \#\ \iota_{\text{e}}\\&\qquad\Rightarrow\lambda\iota_1{:}\,\theta_1.\ \ldots\ \lambda\iota_n{:}\,\theta_n.\,B_{\text{proc}}\ \#\ \iota_{\text{e}}\ ::\ [\pi'\mid\iota{:}\,\rho\mid\iota_{\text{e}}{:}\,\mathbf{explike}],\end{aligned}$$

but by quantifier removal

$$S_{\text{proc}}\Rightarrow(\iota''_1\ \#\ \iota_{\text{e}}\ \&\ \cdots\ \&\ \iota''_m\ \#\ \iota_{\text{e}}\Rightarrow\iota\ \#\ \iota_{\text{e}})\ ::\ [\pi'\mid\iota{:}\,\rho\mid\iota_{\text{e}}{:}\,\mathbf{explike}],$$

so that modus ponens gives

$$\begin{aligned}&S_{\text{proc}}\ \&\ \iota''_1\ \#\ \iota_{\text{e}}\ \&\ \cdots\ \&\ \iota''_m\ \#\ \iota_{\text{e}}\\&\qquad\Rightarrow\lambda\iota_1{:}\,\theta_1.\ \ldots\ \lambda\iota_n{:}\,\theta_n.\,B_{\text{proc}}\ \#\ \iota_{\text{e}}\ ::\ [\pi'\mid\iota{:}\,\rho\mid\iota_{\text{e}}{:}\,\mathbf{explike}],\end{aligned}$$

and quantifier introduction gives

$$\begin{aligned}&S_{\text{proc}}\ \Rightarrow\ (\forall\iota_{\text{e}}{:}\,\mathbf{explike})(\iota''_1\ \#\ \iota_{\text{e}}\ \&\cdots\&\ \iota''_m\ \#\ \iota_{\text{e}}\\&\qquad\qquad\Rightarrow\ \lambda\iota_1{:}\,\theta_1.\ \ldots\ \lambda\iota_n{:}\,\theta_n.\,B_{\text{proc}}\ \#\ \iota_{\text{e}})\ ::\ [\pi'\mid\iota{:}\,\rho]\ .\end{aligned}\tag{d}$$

Then modus ponens of (c) and (d) gives

$$S_1\ \&\ S_{\text{proc}}\Rightarrow(S_{\text{proc}}|_{\iota\to M})\ ::\ [\pi'\mid\iota{:}\,\rho]\ .\tag{e}$$

On the other hand, $S_{\text{proc}}|_{\iota\to\perp_\rho}\in\langle\text{spec},\pi'\rangle$ is

$$\begin{aligned}&(\forall\iota_1{:}\,\theta_1)\cdots(\forall\iota_n{:}\,\theta_n)(\forall\iota'_1{:}\,\theta'_1)\cdots(\forall\iota'_k{:}\,\theta_k)\\&\qquad(S_{\text{pa}}\Rightarrow\{P_{\text{proc}}\}\ \perp_\rho\ (\iota_1)\cdots(\iota_n)\ \{Q_{\text{proc}}\})\\&\ \&\ (\forall\iota_{\text{e}}{:}\ \mathbf{explike})(\iota''_1\ \#\ \iota_{\text{e}}\ \&\cdots\&\ \iota''_m\ \#\ \iota_{\text{e}}\Rightarrow\perp_\rho\ \#\ \iota_{\text{e}})\ ::\ \pi''\ ,\end{aligned}\tag{f}$$

whose universality can be inferred, by adding assumptions and introducing quantifiers, from

$$\{P_{\text{proc}}\}\ \perp_\rho(\iota_1)\cdots(\iota_n)\ \{Q_{\text{proc}}\}\ ::\ \pi'''$$

and

$$\perp_\rho\ \#\ \iota_{\text{e}}\ ::\ [\pi'\mid\iota_{\text{e}}{:}\,\mathbf{explike}]\ ,$$

which are consequences of (3.4), (9.1), and (5.5). Finally from (f), (e), and (b), broadening the type assignments to replace π' by π, taking S to be S_1 & S_2, and noting that $\iota \notin \mathcal{F}_{-}(S_{\text{proc}})$, rule (10.1) for **letrec** commands gives the conclusion

$$S_1 \mathbin{\&} S_2 \Rightarrow \{P\}\ \textbf{letrec}\ \iota{:}\,\rho\ \textbf{be}\ \lambda\iota_1{:}\,\theta_1.\ \ldots\ \lambda\iota_n{:}\,\theta_n.\,B_{\text{proc}}\ \textbf{in}\ B\ \{Q\}\ ::\ \pi\ .$$

11. Some Questionable Rules of Inference

We conclude by introducing two rules of inference whose status, at this writing, is still in question. Although we have not been able to deduce a contradiction from these rules, they are not valid for the kind of semantics discussed in [8], and the problem of constructing a semantics for which they are valid has not been solved. However, they are extremely useful; indeed they—or something like them—seem to be necessary for certain aspects of program proving.

The first is a stronger version of (4.20):

(11.1) Strong Constancy. Let π be a type assignment mapping c into **comm** and p, q, and r into **Boolean exp**. Then

$$c \mathbin{\#} p \mathbin{\&} (\{p\} \Rightarrow \{q\}\ c\ \{r\}) \Rightarrow \{q\ \textbf{and}\ p\}\ c\ \{r\ \textbf{and}\ p\}\ ::\ \pi\ .$$

Roughly speaking, if p holds before executing c and is not interfered with by c (at any time during its execution), one can assume that it is a static assertion in reasoning about c. The difficulty is that it is not really static, i.e., there are environments for which $c \mathbin{\#} p$ is true yet $\{p\}$ is false.

The second rule is a limited kind of converse to our rule for leftside noninterference decomposition:

(11.2) Leftside Noninterference Composition. Suppose that $C \in \langle\text{comm}, \pi\rangle$, $E \in \langle\text{explike}, \pi\rangle$, $P, Q \in \langle\text{Boolean exp}, \pi\rangle$, and $\mathcal{F}_{\text{commlike}}(C) = \{\iota_1, \ldots, \iota_n\}$. Then

$$(\iota_1 \mathbin{\#} E \mathbin{\&} \cdots \mathbin{\&} \iota_n \mathbin{\#} E \Rightarrow \{P\}\ C\ \{Q\})\ \Rightarrow\ (C \mathbin{\#} E \Rightarrow \{P\}\ C\ \{Q\})\ ::\ \pi\ .$$

To see the difficulty with this rule, take π to be $[k{:}\,\textbf{integer integer var}]$, C to be $(\lambda k{:}\,\textbf{integer integer var}.\,\textbf{skip})(k)$, E to be k, P to be **true**, and Q to be **false**. Then (11.2) gives, with beta reduction of C,

$$(k \mathbin{\#} k \Rightarrow \{\textbf{true}\}\ \textbf{skip}\ \{\textbf{false}\})\ \Rightarrow\ (\textbf{skip} \mathbin{\#} k \Rightarrow \{\textbf{true}\}\ \textbf{skip}\ \{\textbf{false}\})\ ::\ \pi\ ,$$

which in conjunction with (5.7) gives

$$(k \mathbin{\#} k \Rightarrow \{\textbf{true}\}\ \textbf{skip}\ \{\textbf{false}\})\ \Rightarrow\ \{\textbf{true}\}\ \textbf{skip}\ \{\textbf{false}\}\ ::\ \pi\ .$$

Clearly there are environments in which both $k \mathbin{\#} k$ and $\{\mathbf{true}\}\ \mathbf{skip}\ \{\mathbf{false}\}$ are false, so this specification is not universal in a conventional semantics.

On the other hand, the need for a rule of this sort is illustrated by an attempt to show that

$$
\begin{array}{l}
\mathbf{let}\ \mathit{iterate}\ \mathbf{be}\ \lambda p\colon \mathbf{integer\ exp} \to \mathbf{comm}.\\
\qquad \mathbf{new\ integer\ var}\ k\ \mathbf{in}\\
\qquad\qquad (k := 0;\ \mathbf{while}\ k < 100\ \mathbf{do}\ (k := k+1;\ p(k)))\\
\mathbf{in}\ (s := 0;\ \mathit{iterate}(\lambda k\colon \mathbf{integer\ exp}.\, s := s + f(k)))
\end{array}
$$

will set s to $\sum_{i=1}^{100} f(i)$.

It is a straightforward exercise to show

$$
\begin{array}{l}
(\forall k\colon \mathbf{integer\ exp})\\
\qquad (p(k) \mathbin{\#} k \Rightarrow \{w(k-1)\ \mathbf{and}\ 1 \leq k \leq 100\}\ p(k)\ \{w(k)\})\\
\Rightarrow \{w(0)\}\ \mathbf{new\ integer\ var}\ k\ \mathbf{in}\ \cdots\ \{w(100)\}\\
\qquad\qquad :: \ [p\colon \mathbf{integer\ exp} \to \mathbf{comm} \mid w\colon \mathbf{integer\ exp} \to \mathbf{Boolean\ exp}]
\end{array}
$$

which leads to an application of rule (10.2) with $S_{\mathrm{proc}} =$

$$
\begin{array}{l}
(\forall p\colon \mathbf{integer\ exp} \to \mathbf{comm})(\forall w\colon \mathbf{integer\ exp} \to \mathbf{Boolean\ exp})\\
\qquad ((\forall k\colon \mathbf{integer\ exp})\\
\qquad\qquad (p(k) \mathbin{\#} k\ \Rightarrow \{w(k-1)\ \mathbf{and}\ 1 \leq k \leq 100\}\ p(k)\ \{w(k)\})\\
\qquad \Rightarrow \{w(0)\}\ \mathit{iterate}(p)\ \{w(100)\})\\
\&\ (\forall e\colon \mathbf{explike})\ \mathit{iterate} \mathbin{\#} e\ .
\end{array}
$$

Then by removing quantifiers, with the substitution

$$p \to \lambda k\colon \mathbf{integer\ exp}.\, s := s + f(k)$$

$$w \to \lambda k\colon \mathbf{integer\ exp}.\, s = \textstyle\sum_{i=1}^{k} f(i)\ ,$$

we have (with appropriate beta reduction):

$$
\begin{array}{l}
S_{\mathrm{proc}}\ \&\ (\forall k\colon \mathbf{integer\ exp})(s := s + f(k) \mathbin{\#} k\\
\qquad\qquad \Rightarrow\ \{s = \displaystyle\sum_{i=1}^{k-1} f(i)\ \mathbf{and}\ 1 \leq k \leq 100\}\ s := s + f(k)\ \{s = \displaystyle\sum_{i=1}^{k} f(i)\})\\
\qquad \Rightarrow\ \{s = \displaystyle\sum_{i=1}^{0} f(i)\}\ \mathit{iterate}(\lambda k\colon \mathbf{integer\ exp}.\, s := s + f(k))\ \{s = \displaystyle\sum_{i=1}^{100} f(i)\}\\
\hfill :: \ \pi\ ,
\end{array}
$$

where π is a type assignment that maps s into **real real var**, f into **integer exp** $\rightarrow$ **real exp**, and *iterate* into (**integer exp** $\rightarrow$ **comm**) $\rightarrow$ **comm**.

By the rule for assignment and well-known mathematical facts about summation, we can obtain

$$\begin{aligned}&\mathbf{gv}(s)\ \&\ s \# f \Rightarrow\ (s \# k\\&\qquad \Rightarrow\ \{s = \sum_{i=1}^{k-1} f(i)\ \textbf{and}\ 1 \leq k \leq 100\}\ s := s + f(k)\ \{s = \sum_{i=1}^{k} f(i)\})\ ::\ \pi\ .\end{aligned}$$

The need for rule (11.2) arises from the discrepancy between $s \# k$ and $s := s + f(k) \# k$. If we can use (11.2), along with quantifier introduction, we get

$$\begin{aligned}&\mathbf{gv}(s)\ \&\ s \# f \Rightarrow (\forall k\colon \textbf{integer exp})(s := s + f(k) \# k\\&\qquad \Rightarrow\ \{s = \sum_{i=1}^{k-1} f(i)\ \textbf{and}\ 1 \leq k \leq 100\}\ s := s + f(k)\ \{s = \sum_{i=1}^{k} f(i)\})\ ::\ \pi\ ,\end{aligned}$$

and modus ponens gives the desired result:

$$\begin{aligned}&S_{\text{proc}}\ \&\ \mathbf{gv}(s)\ \&\ s \# f\ \Rightarrow\\&\qquad \{s = \sum_{i=1}^{0} f(i)\}\ \mathit{iterate}(\lambda k\colon \textbf{integer exp}.\, s := s + f(k))\ \{s = \sum_{i=1}^{100} f(i)\}\\&\qquad\qquad ::\ \pi\ .\end{aligned}$$

References

[1] Hoare, C. A. R. 1969. An axiomatic basis for computer programming. *Comm. ACM*, 12(10):576–580 and 583.

[2] Landin, P. J. 1966. A λ-calculus approach. In *Advances in Programming and Non-Numerical Computation*, L. Fox, editor, Oxford. Pergamon Press, pages 97–141.

[3] Landin, P. J. 1965. A correspondence between Algol 60 and Church's lambda-notation. *Comm. ACM*, 8(2,3):89–101 and 158–165.

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

[5] Hoare, C. A. R. and Wirth, N. 1973. An axiomatic definition of the programming language Pascal. *Acta Informatica*, 2(4):335–355.

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

[7] Gries, D. and Levin, G. 1980. Assignment and procedure call proof rules. *ACM Trans. on Programming Languages and Systems*, 2(4):564–579.

[8] Reynolds, J. C. 1981a. *The Craft of Programming*. Prentice-Hall International, London.

[9] Wirth, N. and Hoare, C. A. R. 1966. A contribution to the development of Algol. *Comm. ACM*, 9(6):413–432.

[10] Reynolds, J. C. 1981b. The essence of Algol. In *Algorithmic Languages, Proceedings of the International Symposium on Algorithmic Languages*, J. W. de Bakker and J. C. van Vliet, editors. North-Holland, Amsterdam, pages 345–372.