CMU-CS-85-106

On the Axiomatic Treatment of Concurrency

Stephen D. Brookes
Carnegie-Mellon University
Department of Computer Science
Schenley Park
Pittsburgh
PA 15213
USA

To appear in: Proc. 1984 NSF-SERC Seminar on Concurrency, Springer LNCS (1985).

The research reported in this paper was supported in part by funds from the Computer Science Department of Carnegie-Mellon University, and by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539. The views and conclusions contained in it are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

ON THE AXIOMATIC TREATMENT OF CONCURRENCY

Stephen D. Brookes
Carnegie-Mellon University
Department of Computer Science
Schenley Park
Pittsburgh

0. Abstract.

This paper describes a semantically–based axiomatic treatment of a simple parallel programming language. We consider an imperative language with shared variable concurrency and a critical region construct. After giving a structural operational semantics for the language we use the semantic structure to suggest a class of assertions for expressing semantic properties of commands. The structure of the assertions reflects the structure of the semantic representation of a command. We then define syntactic operations on assertions which correspond precisely to the corresponding syntactic constructs of the programming language; in particular, we define sequential and parallel composition of assertions. This enables us to design a truly compositional proof system for program properties. Our proof system is sound and relatively complete. We examine the relationship between our proof system and the Owicki-Gries proof system for the same language, and we see how Owicki's parallel proof rule can be reformulated in our setting. Our assertions are more expressive than Owicki's, and her *proof outlines* correspond roughly to a special subset of our assertion language. Owicki's parallel rule can be thought of as being based on a slightly different form of parallel composition of assertions; our form does not require *interference-freedom*, and our proof system is relatively complete without the need for auxiliary variables. Connections with the "Generalized Hoare Logic" of Lamport and Schneider, and with the Transition Logic of Gerth, are discussed briefly, and we indicate how to extend our ideas to include some more programming constructs, including conditional commands, conditional critical regions, and loops.

1. Introduction.

It is widely accepted that formal reasoning about program properties is desirable. Hoare's paper [12] has led to attempts to give axiomatic treatments for a wide variety of programming languages. Hoare's paper treated partial correctness properties of commands

in a sequential programming language, using simple assertions based on pre- and post-conditions; the axiom system given in that paper is sound and relatively complete [8]. The proof system was *syntax-directed*, in that axioms or rules were given for each syntactic construct. The assertions chosen by Hoare are admirably suited to the task: they are concise in structure and have a clear correlation with a natural state transformation semantics for the programming language; this means that fairly straightforward proofs of the soundness and completeness of Hoare's proof system can be given [1,8].

When we consider more complicated programming languages the picture is not so simple. Many existing axiomatic treatments of programming languages have turned out to be either unsound or incomplete [25]. The task of establishing soundness and completeness of proof systems for program properties can be complicated by an excessive amount of detail used in the semantic description of the programming language. This point seems to be quite well known, and is made, for instance in [1]. Similar problems can be caused by the use of an excessively intricate or poorly structured assertion language, or by overly complicated proof rules. Certainly for sequential languages with state-transformation semantics the usual Hoare-style assertions with pre- and post-conditions are suitable. But for more complicated languages which require more sophisticated semantic treatment we believe that it is inappropriate to try to force assertions to fit into the pre- and post-condition mould; such an attempt tends to lead to pre- and post-conditions with a rather complex structure, when it could be simpler to use a class of assertions with a different structure which more accurately corresponds to the semantics. The potential benefits of basing an axiomatic treatment directly on a well chosen semantics has been argued, for instance, in [7], where an axiomatic treatment of aliasing was given. Parallel programming languages certainly require a more sophisticated semantic model than sequential languages, and this paper attempts to construct a more sophisticated axiomatic treatment based on the *resumption* model of Hennessy and Plotkin [22].

Proof systems for reasoning about various forms of parallelism have been proposed by several authors, notably [2,3,4,11,15,16,17,18,19,20,21]. Owicki and Gries [20,21] gave a Hoare-style axiom system for a simple parallel programming language in which parallel commands can interact through their effects on shared variables. Their proof rule for parallel composition involved a notion of *interference-freedom* and used *proof outlines* for parallel processes, rather than the usual Hoare-style assertions. In order to obtain a complete proof system Owicki found it necessary to use *auxiliary variables* and to add proof rules for dealing with them. These features have been the subject of considerable discussion in the literature, such as [5,16]. Our approach is to begin with an appropriate semantic model, chosen to allow compositional reasoning about program properties. We use the structure of this model more directly than is usual in the design of an assertion language for program properties, and this leads to proof rules with a very simple structure, although (or rather, because) our assertions are more powerful than conventional Hoare-style assertions; Owicki's proof outlines emerge as special cases of our assertions. The

soundness and completeness of our proof system are arguably less difficult to establish, as the proof system is closely based on the semantics and the semantics has been chosen to embody as little complication as possible while still supporting formal reasoning about the desired properties of programs.

The programming language discussed here is a subset of the language considered by Owicki [20,21], and by Hennessy and Plotkin [22]. Adopting the structural operational semantics of [22,26] for this language, we design a class of assertions for expressing semantic properties of commands. We then define *syntactic* operations on assertions which correspond to the *semantics* of the various syntactic constructs in the programming language; in particular, we define sequential and parallel composition for assertions. This leads naturally to *compositional*, or syntax-directed, proof rules for the syntactic constructs. We do not need an interference-freedom condition in our rule for parallel composition, in contrast to Owicki's system. Similarly, we do not need an auxiliary variables rule in order to obtain completeness. We show how to construct Owicki's rule for parallel composition and the need for her interference-freedom condition, using our methods. Essentially, Owicki's system uses a restricted subset of our assertions and a variant form of parallel composition of assertions.

We compare our work briefly with that of some other authors in this field, discuss some of its present limitations, and the paper ends with a few suggestions for further research and some conclusions. In particular, we indicate that our ideas can be extended to cover features omitted from the body of the paper, such as conditional critical regions, loops and conditionals. We also believe that with a few modifications in the assertion language we will be able to incorporate guarded commands [9,10], and with an appropriate definition of parallel composition for assertions we will be able to treat CSP-like parallel composition [13], in which processes do not share variables but instead interact solely by means of synchronized communication.

2. A Parallel Programming Language.

We begin with a simple programming language containing assignment and sequential composition, together with a simple form of parallel composition, and a "critical region" construct. Parallel commands interact solely through their effects on shared variables. For simplicity of presentation we omit conditionals and loops, at least for the present, as we want to focus on the problems caused by parallelism. We will return briefly to these features later. As usual for imperative languages, we distinguish the syntactic categories of identifiers, expressions, and commands. The abstract syntax for expressions and identifiers will be taken for granted.

Syntax.

$$I \in \mathbf{Ide} \quad \text{identifiers},$$
$$E \in \mathbf{Exp} \quad \text{expressions},$$
$$\Gamma \in \mathbf{Com} \quad \text{commands},$$
$$\Gamma ::= \mathbf{skip} \mid I{:=}E \mid \Gamma_1 ; \Gamma_2 \mid [\Gamma_1 \,\|\, \Gamma_2] \mid \langle \Gamma \rangle.$$

The notation is fairly standard. The command **skip** is an atomic action having no effect on program variables. An assignment, denoted $I{:=}E$, is also an atomic action; it sets the value of I to the (execution-time) value of E. Sequential composition is represented by $\Gamma_1; \Gamma_2$. A parallel composition $[\Gamma_1 \,\|\, \Gamma_2]$ is executed by interleaving the atomic actions of the component commands Γ_1 and Γ_2. A command of the form $\langle \Gamma \rangle$ is a *critical region*; this construct converts a command into an atomic action, and corresponds to a special case of an *await* statement in [20], where the notation **await true do** Γ would have been used.

In describing the semantics of this language, we will focus mainly on commands. The set S of *states* consists simply of the (partial) functions from identifiers to values:

$$S = [\mathbf{Ide} \to_p V],$$

where V is some set of expression values (typically containing integers and truth values). We use s to range over states, and we write $s + [I \mapsto v]$ for the state which agrees with s except that it gives identifier I the value v. As usual, the value denoted by an expression may depend on the values of its free identifiers. Thus, we assume the existence of a semantic function

$$\mathcal{E} : \mathbf{Exp} \to [S \to V].$$

We specify the semantics of commands in the structural operational style [26], and our presentation follows that of [22], where identical program constructs were considered. We define first an abstract machine which specifies the computations of a command. The abstract machine is given by a *labelled transition system*

$$\langle \mathbf{Conf}, \mathbf{Lab}, \to \rangle,$$

where **Conf** is a set of *configurations*, **Lab** is a set of *labels* (ranged over by α, β and γ), and $\to$ is a family

$$\{ \xrightarrow{\alpha} \mid \alpha \in \mathbf{Lab} \}$$

of *transition relations* $\xrightarrow{\alpha} \subseteq \mathbf{Conf} \times \mathbf{Conf}$ indexed by elements of **Lab**. An atomic action is either an assignment, or **skip**, or a critical region. We use labels for atomic actions, and assume from now on that all atomic actions of a command have labels: in other words, we deal with *labelled commands*. For precision, we give the following syntax for labelled

commands, in which α ranges over **Lab**:

$$\Gamma ::= \alpha{:}\mathbf{skip} \mid \alpha{:}I{:=}E \mid \Gamma_1;\Gamma_2 \mid [\Gamma_1 \parallel \Gamma_2] \mid \alpha{:}\langle\Gamma\rangle.$$

For convenience we introduce a term **null** to represent termination, and we specify (purely for notational convenience) that

$$[\mathbf{null} \parallel \Gamma] = [\Gamma \parallel \mathbf{null}] = \Gamma,$$
$$\mathbf{null}; \Gamma = \Gamma.$$

We will use **Com′** for the set containing all labelled commands and **null**. The set of configurations is $\mathbf{Conf} = \mathbf{Com}' \times S$. A configuration of the form $\langle \Gamma, s\rangle$ will represent a stage in a computation at which the remaining command to be executed is Γ, and the current state is s. A configuration of the form $\langle \mathbf{null}, s\rangle$ represents termination in the given state. A *transition* of the form

$$\langle \Gamma, s\rangle \xrightarrow{\alpha} \langle \Gamma', s'\rangle$$

represents a step in a computation in which the state and remaining command change as indicated, and in which the atomic action labelled α occurs. We write $\langle \Gamma, s\rangle \rightarrow \langle \Gamma', s'\rangle$ when there is an α for which $\langle \Gamma, s\rangle \xrightarrow{\alpha} \langle \Gamma', s'\rangle$. And we use the notation $\rightarrow^*$ for the reflexive transitive closure of this relation. Thus $\langle \Gamma, s\rangle \rightarrow^* \langle \Gamma', s'\rangle$ iff there is a sequence of atomic actions from the first configuration to the second.

The transition relations are defined by the following syntax-directed transition rules; the transition relations are to be the smallest satisfying these laws. This means that a transition is possible if and only if it can be deduced from the rules.

Transition Rules

$$\langle \alpha{:}\mathbf{skip}, s\rangle \xrightarrow{\alpha} \langle \mathbf{null}, s\rangle \tag{A1}$$

$$\langle \alpha{:}I{:=}E, s\rangle \xrightarrow{\alpha} \langle \mathbf{null}, s + [I \mapsto \mathcal{E}[\![E]\!]s]\rangle \tag{A2}$$

$$\frac{\langle \Gamma_1, s\rangle \xrightarrow{\alpha} \langle \Gamma_1', s'\rangle}{\langle \Gamma_1;\Gamma_2, s\rangle \xrightarrow{\alpha} \langle \Gamma_1';\Gamma_2, s'\rangle} \tag{A3}$$

$$\frac{\langle \Gamma_1, s\rangle \xrightarrow{\alpha} \langle \Gamma_1', s'\rangle}{\langle [\Gamma_1 \parallel \Gamma_2], s\rangle \xrightarrow{\alpha} \langle [\Gamma_1' \parallel \Gamma_2], s'\rangle} \tag{A4}$$

$$\frac{\langle \Gamma_2, s\rangle \xrightarrow{\alpha} \langle \Gamma_2', s'\rangle}{\langle [\Gamma_1 \parallel \Gamma_2], s\rangle \xrightarrow{\alpha} \langle [\Gamma_1 \parallel \Gamma_2'], s'\rangle} \tag{A5}$$

$$\frac{\langle \Gamma, s\rangle \rightarrow^* \langle \mathbf{null}, s'\rangle}{\langle \alpha{:}\langle\Gamma\rangle, s\rangle \xrightarrow{\alpha} \langle \mathbf{null}, s'\rangle} \tag{A6}$$

From our definition of the transition system, we see that we have specified that a parallel composition terminates only when both components have terminated. This is because of our conventions about null: we have $\langle[\Gamma_1 \parallel \Gamma_2], s\rangle \xrightarrow{\alpha} \langle\Gamma_2, s'\rangle$ whenever $\langle\Gamma_1, s\rangle \xrightarrow{\alpha} \langle\mathbf{null}, s'\rangle$, for instance. It is also clear from the definitions that all computations eventually terminate in this transition system, and that no computation gets "stuck": the only configurations in which no further action is possible are the terminal configurations. These properties would not hold if we add guarded commands or loops to the language. This point will be mentioned again later; for now we will concentrate on the language as it stands.

Examples.

Example 1. Let s be a state and let $s_i = s + [x \mapsto i]$ for $i \geq 0$. Let Γ be the labelled command

$$[\alpha : x{:=}x + 1 \parallel \beta : x{:=}x + 1].$$

Then we have

$$\langle\Gamma, s_0\rangle \xrightarrow{\alpha} \langle\beta : x{:=}x + 1, s_1\rangle \xrightarrow{\beta} \langle\mathbf{null}, s_2\rangle,$$

and a similar sequence in which the order of the two actions is reversed:

$$\langle\Gamma, s_0\rangle \xrightarrow{\beta} \langle\alpha : x{:=}x + 1, s_1\rangle \xrightarrow{\alpha} \langle\mathbf{null}, s_2\rangle.$$

These are the only possible computations from this initial configuration. ∎

Example 2. Let Γ be the command $[\alpha : x{:=}2 \parallel (\beta : x{:=}1; \gamma : x{:=}x + 1)]$. Using the s_i notation of the previous example, we have:

$$\langle\Gamma, s\rangle \xrightarrow{\alpha} \langle\beta : x{:=}1; \gamma : x{:=}x + 1, s_2\rangle \xrightarrow{\beta} \langle\gamma : x{:=}x + 1, s_1\rangle \xrightarrow{\gamma} \langle\mathbf{null}, s_2\rangle,$$
$$\langle\Gamma, s\rangle \xrightarrow{\beta} \langle[\alpha : x{:=}2 \parallel \gamma : x{:=}x + 1], s_1\rangle \xrightarrow{\alpha} \langle\gamma : x{:=}x + 1, s_2\rangle \xrightarrow{\gamma} \langle\mathbf{null}, s_3\rangle,$$
$$\langle\Gamma, s\rangle \xrightarrow{\beta} \langle[\alpha : x{:=}2 \parallel \gamma : x{:=}x + 1], s_1\rangle \xrightarrow{\gamma} \langle\alpha : x{:=}2, s_2\rangle \xrightarrow{\alpha} \langle\mathbf{null}, s_2\rangle.$$

This command sets x to 2 or 3, depending on the order in which its atomic actions are executed. ∎

Example 3. Let Γ be the command $[\alpha : x{:=}1 \parallel \beta : y{:=}1]$. Then we have:

$$\langle\Gamma, s\rangle \xrightarrow{\alpha} \langle\beta : y{:=}1, s + [x \mapsto 1]\rangle \xrightarrow{\beta} \langle\mathbf{null}, s + [x \mapsto 1, y \mapsto 1]\rangle,$$
$$\langle\Gamma, s\rangle \xrightarrow{\beta} \langle\alpha : y{:=}1, s + [y \mapsto 1]\rangle \xrightarrow{\alpha} \langle\mathbf{null}, s + [x \mapsto 1, y \mapsto 1]\rangle.$$

This command sets both x and y to 1. ∎

Semantics.

Using the transition system we may now extract a semantics. For a partial correctness semantics, we should examine the (terminating) computations of a command and extract the initial and final states. Of course, in the present language there is no need to distinguish between total and partial correctness because all computations terminate, but this issue will arise in treatments of an extended language containing loops (for example). For uniformity, we still refer to partial correctness, as the definition we give adapts even to the extended language and does then correspond to partial correctness.

Definition 1. The semantic function $\mathcal{M} : \mathbf{Com} \to [S \to \mathcal{P}(S)]$ is

$$\mathcal{M}[\![\Gamma]\!]s = \{ s' \mid \langle \Gamma, s \rangle \to^* \langle \mathbf{null}, s' \rangle \}. \quad \blacksquare$$

Examples. We have already seen that

1. $\mathcal{M}[\![\alpha : x{:=}x+1 \parallel \beta : x{:=}x+1]\!]s_i = \{ s_{i+2} \}$,
2. $\mathcal{M}[\![\alpha : x{:=}2 \parallel (\beta : x{:=}1; \gamma : x{:=}x+1)]\!]s = \{ s_2, s_3 \}$,
3. $\mathcal{M}[\![\alpha : x{:=}1 \parallel \beta : y{:=}1]\!]s = \{ s + [x \mapsto 1, y \mapsto 1] \}$.

Reasoning about commands.

In conventional Hoare logics for sequential imperative programs, assertions of the form

$$\{ P \}\Gamma\{ Q \}$$

are used, with P and Q being called the pre- and post-condition. These conditions are typically drawn from a simple first order language, and are interpreted as predicates of the state. Given a satisfaction relation $\models$ on conditions and states, we say that $\{ P \}\Gamma\{ Q \}$ is *valid*, written $\models \{ P \}\Gamma\{ Q \}$, iff

$$\forall s, s'[s \models P \ \& \ s' \in \mathcal{M}[\![\Gamma]\!]s \quad \Rightarrow \quad s' \models Q].$$

In other words, a Hoare assertion of this type describes the relationship between an initial state and the possible final states of a computation of a command. However, it is well known [20,22,23] that in a language involving parallel composition it is not possible to reason about partial correctness properties of a command in isolation: account must be taken of the context in which the command is to be run. This is exemplified by the commands

$$x{:=}2, \qquad \text{and} \qquad x{:=}1; x{:=}x+1,$$

which clearly have the same partial correctness properties in isolation, *i.e.*

$$\mathcal{M}[\![x{:=}2]\!] = \mathcal{M}[\![x{:=}1; x{:=}x+1]\!],$$

but which exhibit different partial correctness properties in some programming language contexts; for instance, the commands

$$[x{:=}2 \parallel x{:=}2] \qquad \text{and} \qquad [(x{:=}1; x{:=}x+1) \parallel x{:=}2]$$

do not have the same partial correctness properties, as the latter command may set x to 3. Thus, the $\mathcal{M}$ semantics does not always distinguish between pairs of commands if there is a program context in which they exhibit different partial correctness behaviour. Technically, the relational semantics $\mathcal{M}$ fails to be *fully abstract* [22,23] with respect to partial correctness; it makes too few distinctions between commands, and is therefore "too abstract". In order to reason about the correctness of a parallel combination of commands in a manner independent of the context in which the command appears, we need to know more about the individual commands than simply their relational semantics $\mathcal{M}$. Similarly, we cannot axiomatize partial correctness of commands solely on the basis of partial correctness properties of components: conventional pre- and post-condition assertions are not going to suffice.

Hennessy and Plotkin [22] showed that the transition system above can be used to define a semantics which will distinguish between terms if there is a context in which they can exhibit different partial correctness properties. This semantics uses the notion of a *resumption*. For our subset of the language, we may adapt these ideas slightly to define the following semantics for labelled commands:

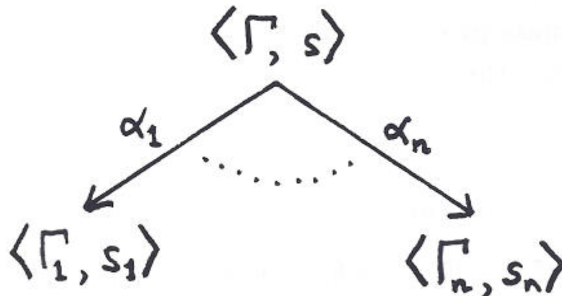
$$\mathcal{R} : \mathbf{Com}' \to R,$$
$$R = [S \to \mathcal{P}(\mathbf{Lab} \times R \times S)],$$

with the definition being

$$\mathcal{R}[\![\Gamma]\!]s = \{\, \langle \alpha, \mathcal{R}[\![\Gamma']\!], s' \rangle \mid \langle \Gamma, s \rangle \xrightarrow{\alpha} \langle \Gamma', s' \rangle \,\}.$$

Justification for this use of a recursively defined domain R of resumptions can be given if we interpret $\mathcal{P}$ as a powerdomain construct, and the interested reader should consult [22] for details.

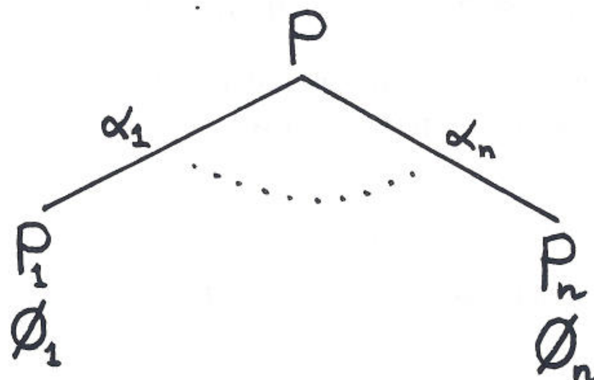
Note that according to this definition we have $\mathcal{R}[\![\mathbf{null}]\!]s = \emptyset$ for all states s. Note also that for any state s, $\mathcal{R}[\![\Gamma]\!]s$ will be a finite set. This can be represented as a tree structure as follows, with a branch for each member of the set, labelled by the corresponding atomic action label, with a son consisting of a resumption-state pair.



The tree structure suggests a class of assertions with components representing the branch structure of trees. We therefore introduce a class of assertions of the form

$$\phi ::= P\sum_{i=1}^{n} \alpha_i P_i \phi_i,$$

where as before P and the P_i are drawn from some *condition* language, and where the α_i are labels. This notation obviously corresponds with Milner's linear notation for *synchronization trees* [24]; in addition to labelling the arcs with action labels, we also incorporate conditions at nodes. We make no distinction between assertions which differ only in the order in which their branches are written. A tree representation of such a ϕ will often be preferable to the linear notation; for example, the assertion $P\sum_{i=1}^{n} \alpha_i P_i \phi_i$ may be represented as:



We will feel free to use set braces to delimit conditions as an aid to the eye, and we use NIL for the tree with no branches (this corresponds to termination, since in this language inability to perform any action coincides with termination). Thus, an assertion in which $n = 0$ will be written $\{P\}$NIL; we also introduce the special notation $\bullet$ to stand for the assertion $\{\mathbf{true}\}$NIL. Finally, it will be convenient to adopt the convention that $\{P\}\alpha\{Q\}$ (which does not conform to the syntax above) abbreviates the assertion $\{P\}\alpha\{Q\}\{Q\}$NIL (which does).

Note that there is an obvious definition of the *depth* of an assertion ϕ, and that all assertions have finite depth. The terminal assertions are those with zero depth.

In order to express the property that a command Γ *satisfies* an assertion ϕ we write

$$\Gamma\ \mathbf{sat}\ \phi.$$

This type of formal property will be the subject of our proof system, and we will see later that we have a generalization of conventional Hoare-style assertions.

When ϕ is the assertion $P \sum_{i=1}^{n} \alpha_i P_i \phi_i$ we interpret Γ **sat** ϕ in the following way. If the command is started in a state satisfying P, then its initial action must be an α_i drawn from the set of initial labels of the assertion, and these labels are precisely the initial actions possible for the command. If the command starts with an α_i action it reaches a state where P_i is true and where the remaining command satisfies ϕ_i. Specifically, we write

$$\models \Gamma \ \mathbf{sat}\ \phi$$

to indicate that Γ satisfies ϕ. This means that, with the above notation,

$$\forall s \forall \alpha. (s \models P \ \& \ \langle \Gamma, s\rangle \xrightarrow{\alpha} \langle \Gamma', s'\rangle \quad \Rightarrow \quad \exists i \leq n. \alpha = \alpha_i \ \& \ s' \models P_i \ \& \ \Gamma' \models \phi_i), \qquad (1)$$

and, in addition, that

$$\forall s. \forall i. (s \models P \ \Rightarrow \ \exists \Gamma_i, s_i. \langle \Gamma, s\rangle \xrightarrow{\alpha_i} \langle \Gamma_i, s_i\rangle), \qquad (2)$$

so that all of the actions specified in ϕ are indeed possible for Γ when the initial state satisfies P. These definitions can be rephrased in terms of the semantic function $\mathcal{R}$.

Note that we always have

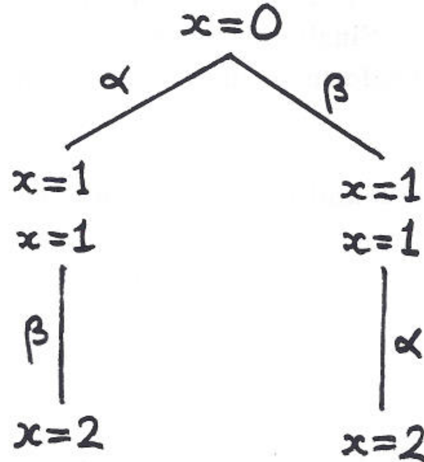
$$\models \mathbf{null\ sat} \bullet,$$

and indeed (non-trivial) terminal assertions can only be satisfied by **null**.

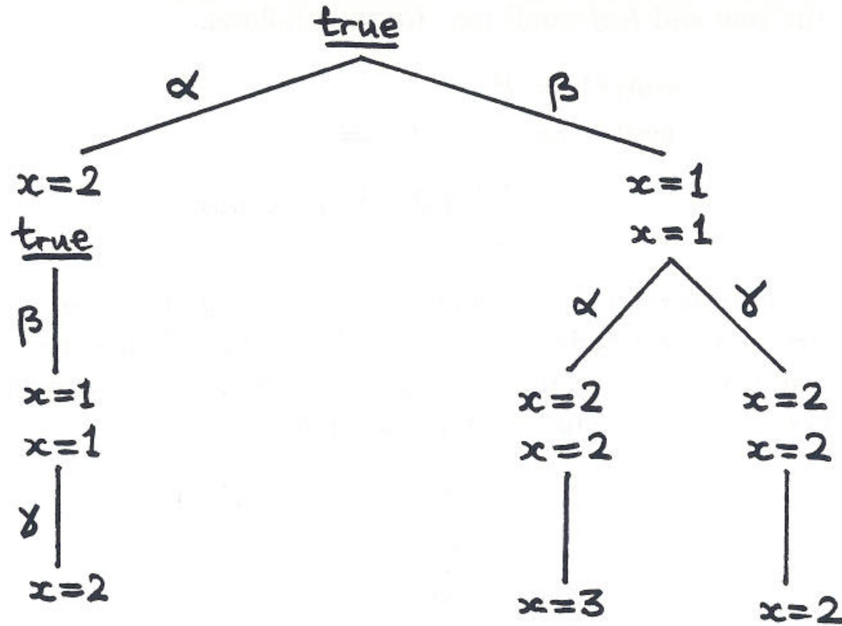
Examples..

Example 1. The command $[\alpha : x := x + 1 \parallel \beta : x := x + 1]$ satisfies the assertion

$$\{x = 0\}(\alpha\{x = 1\}\{x = 1\}\beta\{x = 2\}$$
$$+\beta\{x = 1\}\{x = 1\}\alpha\{x = 2\}).$$



Example 2. The command $[\alpha : x{:=}2 \parallel (\beta : x{:=}1; \gamma : x{:=}x+1)]$ satisfies the assertion



Example 3. The command $[\alpha : x{:=}1 \parallel \beta : y{:=}1]$ satisfies the assertion

$$\{\,\mathbf{true}\,\}(\alpha\{\,x = 1\,\}\{\,x = 1\,\}\beta\{\,x = 1\,\&\,y = 1\,\}$$
$$+\beta\{\,y = 1\,\}\{\,y = 1\,\}\alpha\{\,x = 1\,\&\,y = 1\,\}).$$

Note also that the command does *not* satisfy the assertion

$$\{\,\mathbf{true}\,\}(\alpha\{\,x = 1\,\}\{\,\mathbf{true}\,\}\beta\{\,x = 1\,\&\,y = 1\,\}$$
$$+\beta\{\,y = 1\,\}\{\,\mathbf{true}\,\}\alpha\{\,x = 1\,\&\,y = 1\,\}).$$

Example 4. The assertion

$$\{\,x = 0\,\}\alpha\{\,x = 1\,\}\{\,x = 1\,\}\beta\{\,x = 2\,\}$$

is satisfied by the labelled command

$$\alpha : x{:=}x + 1; \beta : x{:=}x + 1,$$

and so is the assertion

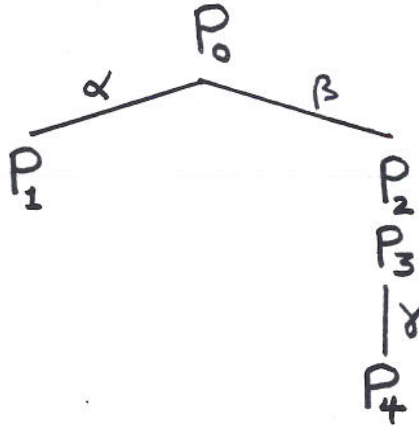
$$\{\,x = 0\,\}\alpha\{\,x = 1\,\}\{\,x = 99\,\}\beta\{\,x = 100\,\}. \quad \blacksquare$$

Let ϕ be the assertion $P\sum_{i=1}^{n}\alpha_i P_i\phi_i$. Tree structure suggests the use of the following notation. Define the *root* and *leaf* conditions for ϕ as follows:

$$\begin{aligned}\text{root}(\phi) &= P,\\ \text{leaf}(\phi) &= P \qquad \text{if } n=0,\\ &= \bigvee_{i=1}^{n}\text{leaf}(\phi_i) \quad \text{otherwise.}\end{aligned}$$

The root condition characterizes the state at the root of a computation tree, and the leaf condition characterizes the leaf nodes, i.e. the terminal states. This is just the disjunction of the conditions at the leaves of the assertion. Using the conventional abbreviations introduced earlier, we see for example that the assertion

$$\{P_0\}(\alpha\{P_1\}+\beta\{P_2\}\{P_3\}\gamma\{P_4\})$$



has leaf condition $P_1 \vee P_4$. We also have

$$\begin{aligned}\text{leaf}(\{P\}\alpha\{Q\}) &= Q,\\ \text{leaf}(\{P\}\alpha\{Q\}\bullet) &= \mathbf{true}.\end{aligned}$$

Note that in the syntactic definition of the class of assertions, we have not required that any logical connection exist between adjacent "intermediate" conditions inside an assertion. Although in Example 4 the condition $x = 1$ appears as an intermediate condition, we do not insist that the "following" condition $x = 99$ be a logical consequence. Assertions in which this constraint is satisfied correspond very closely with computation trees and proof outlines. There are good semantic reasons for not making this constraint on the syntax of our assertion language, since assertions satisfying the constraint describe the behaviour of a command in isolation and we know that in general this information is insufficient to characterize the behaviour of a command in all parallel contexts.

Proof System.

Now that we have designed an assertion language for our programming language, let us build a proof system. We will find that we can give a set of syntax-directed proof rules, by constructing syntactic operations on assertions to correspond to the syntactic operations of the programming language. The important point is that we are going to use the semantics directly to suggest how to design our rules.

Atomic assertions.

A terminal assertion $\{P\}\mathrm{NIL}$ represents termination. An *atomic assertion* has the form $\{P\}\alpha\{Q\}\{R\}\mathrm{NIL}$, and the special abbreviated forms $\{P\}\alpha\{Q\}$ and $\{P\}\alpha\{Q\}\bullet$ are thus atomic. Atomic commands satisfy atomic assertions, and the axioms expressing this fact for **skip** and assignment are simple:

$$\alpha\!:\!\textbf{skip}\ \textbf{sat}\ \{P\}\alpha\{P\}\bullet \tag{B1}$$

$$\alpha\!:\!I{:=}E\ \textbf{sat}\ \{[E\backslash I]P\}\alpha\{P\}\bullet. \tag{B2}$$

We use the notation $[E\backslash I]P$ for the result of replacing every free occurrence of I in P by E, with suitable name changes to avoid clashes.

A *critical region* also creates an atomic action out of a command. In order to axiomatize this construct we need to single out a class of assertions which state properties of a command when run in isolation as an indivisible atomic action, since the effect of the critical region construct is to run a command without allowing interruption. Define safe(ϕ) for ϕ of the form $P\sum_{i=1}^{n}\alpha_i P_i \phi_i$ by

$$\mathrm{safe}(\phi) \quad\Leftrightarrow\quad \bigwedge_{i=1}^{n}(P_i \Rightarrow \mathrm{root}(\phi_i))\ \&\ \bigwedge_{i=1}^{n}\mathrm{safe}(\phi_i).$$

This is precisely the constraint mentioned earlier: at each node of the tree the post-condition established by the previous atomic action is required to imply the root condition of the remaining subtree. When $n = 0$ this is trivially true, and the two abbreviated forms of atomic assertion $\{P\}\alpha\{Q\}$ and $\{P\}\alpha\{Q\}\bullet$ are always safe.

Intuitively, if Γ satisfies ϕ and ϕ is safe, then ϕ describes a possible execution of Γ in which no non-trivial interruption is allowed or assumed. Thus, a safe assertion gives information about the command's behaviour in isolation. We can therefore use safe assertions in the proof rule for critical regions:

$$\frac{\Gamma\ \textbf{sat}\ \phi,\quad \mathrm{safe}(\phi)}{\alpha\!:\!\langle\Gamma\rangle\ \textbf{sat}\ \{\mathrm{root}(\phi)\}\alpha\{\mathrm{leaf}(\phi)\}\bullet}. \tag{B3}$$

The soundness of this rule is easy to establish.

Parallel composition.

It is possible to define a parallel composition for assertions. The definition is given inductively. For the base case, when one of the assertions has zero depth, we specify that

$$[\{P\}\mathrm{NIL} \parallel Q\sum_{j=1}^{m}\beta_j Q_j \psi_j] \;=\; \{P \,\&\, Q\}\sum_{j=1}^{m}\beta_j Q_j \psi_j,$$

and similarly when the two terms are exchanged. In particular, it follows that

$$[\bullet \parallel \psi] \;=\; [\psi \parallel \bullet] \;=\; \psi.$$

(Strictly speaking, these are logical equivalences rather than syntactic identities). The inductive clause is an extension of the well known *interleaving* operation on synchronization trees [6,24,28] which handles the node conditions in an appropriate manner. For assertions ϕ and ψ of the form

$$\phi \;=\; P(\sum_{i=1}^{n}\alpha_i P_i \phi_i),$$
$$\psi \;=\; Q(\sum_{j=1}^{m}\beta_j Q_j \psi_j),$$

we define

$$[\phi \parallel \psi] \;=\; \{P \,\&\, Q\}(\sum_{i=1}^{n}\alpha_i P_i[\phi_i \parallel \psi] + \sum_{j=1}^{m}\beta_j Q_j[\phi \parallel \psi_j]).$$

Note that as far as the action sequences are concerned the operation corresponds to the interleaving of trees.

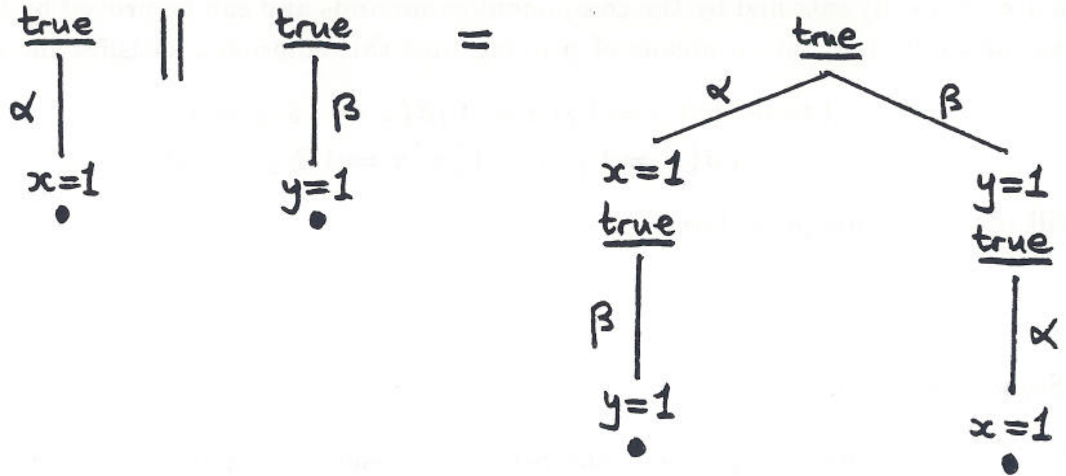
For example, if ϕ and ψ are the atomic assertions

$$\{\,\mathbf{true}\,\}\alpha\{\,x = 1\,\}\bullet, \quad \{\,\mathbf{true}\,\}\beta\{\,y = 1\,\}\bullet,$$

we get

$$\begin{aligned}[\phi \parallel \psi] \;=\; \{\,\mathbf{true}\,\}(&\alpha\{\,x=1\,\}\{\,\mathbf{true}\,\}\beta\{\,y=1\,\}\bullet \\ &+\beta\{\,y=1\,\}\{\,\mathbf{true}\,\}\alpha\{\,x=1\,\}\bullet \\ &)\end{aligned}$$

In tree form, this is represented as follows:



In general, for the abbreviated form of atomic assertion we have:

$$[\{P\}\alpha\{P'\} \,||\, \{Q\}\beta\{Q'\}] = \{P \,\&\, Q\}(\alpha\{P'\}\{P' \,\&\, Q\}\beta\{Q'\} + \beta\{Q'\}\{P \,\&\, Q'\}\alpha\{P'\})$$

$$[\{P\}\alpha\{P'\}\bullet \,||\, \{Q\}\beta\{Q'\}\bullet] = \{P \,\&\, Q\}(\alpha\{P'\}\{Q\}\beta\{Q'\}\bullet + \beta\{Q'\}\{P\}\alpha\{P'\}\bullet).$$

Of course, this composition is not guaranteed to produce *safe* assertions, even if the component assertions are safe. Nevertheless, the following result shows that parallel composition of assertions does indeed have the correct effect: if Γ_1 satisfies ϕ and Γ_2 satisfies ψ then $[\Gamma_1 \,||\, \Gamma_2]$ satisfies $[\phi \,||\, \psi]$. This is true regardless of the structure of ϕ and ψ.

Theorem 1. If $\models \Gamma_1 \textbf{ sat } \phi$ and $\models \Gamma_2 \textbf{ sat } \psi$ then $\models [\Gamma_1 \,||\, \Gamma_2] \textbf{ sat } [\phi \,||\, \psi]$. ∎

Thus we are led to the proof rule:

$$\frac{\Gamma_1 \textbf{ sat } \phi \qquad \Gamma_2 \textbf{ sat } \psi}{[\Gamma_1 \,||\, \Gamma_2] \textbf{ sat } [\phi \,||\, \psi]}. \tag{B4}$$

As an example, we can show that the command $[\alpha : x{:=}1 \,||\, \beta : y{:=}1]$ satisfies the assertion

$$\{\textbf{true}\}(\alpha\{x = 1\}\{\textbf{true}\}\beta\{y = 1\}\bullet + \beta\{y = 1\}\{\textbf{true}\}\alpha\{x = 1\}\bullet),$$

by forming the parallel composition of the assertions

$$\{\textbf{true}\}\alpha\{x = 1\}\bullet, \qquad \{\textbf{true}\}\beta\{y = 1\}\bullet,$$

which are obviously satisfied by the component commands and can be proved by (B2). Note that so far we do not have a means of proving that this command satisfies the assertion

$$\{\,\mathbf{true}\,\}(\alpha\{\,x=1\,\}\{\,x=1\,\}\beta\{\,x=1\,\&\,y=1\,\}$$
$$+\beta\{\,y=1\,\}\{\,y=1\,\}\alpha\{\,x=1\,\&\,y=1\,\}).$$

We will return to this point later.

Sequential composition.

We may also define a sequential composition for assertions. The definition is straightforward, again by induction on depth. The operation grafts ψ on to the leaf nodes of the tree corresponding to ϕ. In the base case, we put

$$(\{\,P\,\}\mathrm{NIL});Q\sum_{j=1}^{m}\beta_jQ_j\psi_j \;=\; \{\,P\,\&\,Q\,\}\sum_{j=1}^{m}\beta_jQ_j\psi_j,$$

so that $\bullet;\psi \;=\; \psi$. When ϕ is $P(\sum_{i=1}^{n}\alpha_iP_i\phi_i)$ and $n>0$ we put

$$\phi;\psi \;=\; P\sum_{i=1}^{n}\alpha_iP_i(\phi_i;\psi).$$

Again we can show that the operation has the desired effect: if Γ_1 satisfies ϕ and Γ_2 satisfies ψ then $\Gamma_1;\Gamma_2$ satisfies $\phi;\psi$.

Theorem 2. If $\models\Gamma_1\ \mathbf{sat}\ \phi$ and $\models\Gamma_2\ \mathbf{sat}\ \psi$ then $\models(\Gamma_1;\Gamma_2)\ \mathbf{sat}\ (\phi;\psi)$. ∎

This suggests the proof rule:

$$\frac{\Gamma_1\ \mathbf{sat}\ \phi \qquad \Gamma_2\ \mathbf{sat}\ \psi}{(\Gamma_1;\Gamma_2)\ \mathbf{sat}\ (\phi;\psi)}. \tag{B5}$$

As an example, we can now prove that the command $\alpha\!:x\!:=\!x+1;\beta\!:x\!:=\!x+1$ satisfies the assertion

$$\{\,x=0\,\}\alpha\{\,x=1\,\}\{\,x=99\,\}\beta\{\,x=100\,\},$$

by forming the sequential composition of the assertions

$$\{\,x=0\,\}\alpha\{\,x=1\,\},\qquad \{\,x=99\,\}\beta\{\,100\,\}.$$

In summary, the rules so far introduced are:

PROOF RULES

$$\alpha\!:\!\textbf{skip sat}\ \{\,P\,\}\alpha\{\,P\,\}\ \bullet \tag{B1}$$

$$\alpha\!:\!I\!:=\!E\ \textbf{sat}\ \{\,[E\backslash I]P\,\}\alpha\{\,P\,\}\ \bullet \tag{B2}$$

$$\frac{\Gamma\ \textbf{sat}\ \phi \quad \mathrm{safe}(\phi)}{\alpha\!:\!\langle\Gamma\rangle\ \textbf{sat}\ \{\,\mathrm{root}(\phi)\,\}\alpha\{\,\mathrm{leaf}(\phi)\,\}\ \bullet} \tag{B3}$$

$$\frac{\Gamma_1\ \textbf{sat}\ \phi_1 \quad \Gamma_2\ \textbf{sat}\ \phi_2}{[\Gamma_1 \parallel \Gamma_2]\ \textbf{sat}\ [\phi_1 \parallel \phi_2]} \tag{B4}$$

$$\frac{\Gamma_1\ \textbf{sat}\ \phi_1 \quad \Gamma_2\ \textbf{sat}\ \phi_2}{(\Gamma_1 ; \Gamma_2)\ \textbf{sat}\ (\phi_1 ; \phi_2)} \tag{B5}$$

The system presented above is sound but not complete. One reason for incompleteness is rather trivial: every command satisfies an assertion ϕ whose root is **false**, but we have no way of proving this from the above rules. One solution is to add a rule to this effect:

$$\frac{\neg\mathrm{root}(\phi)}{\Gamma\ \textbf{sat}\ \phi} \tag{B0}$$

Even this does not guarantee completeness by itself. We saw earlier (Examples 2 and 3) that we were unable to prove some assertions about parallel commands. Example 2, for instance, showed that there is no proof from these rules alone that the command

$$[\alpha\!:\!x\!:=\!x+1 \parallel \beta\!:\!x\!:=\!x+1]$$

satisfies the assertion

$$\{\,x=0\,\}(\alpha\{\,x=1\,\}\{\,x=1\,\}\beta\{\,x=2\,\}\ +\ \beta\{\,x=1\,\}\{\,x=1\,\}\alpha\{\,x=2\,\}).$$

Rule (B0) does not help in these examples. Essentially, the reason for this is that we really need to use *two* assertions about each component command here: we need to be able to say

that $x{:=}x+1$ will change the value of x from 0 to 1, and that it will equally well change the value of x from 1 to 2. Of course, in general the number of separate assertions required may be more than two. We will therefore allow *conjunction* of assertions and include a natural rule which expresses an appropriate notion of *implication* for our assertions. For conjunction we simply add to the syntax of our assertion language the clause

$$\phi ::= (\phi_1 \oplus \phi_2).$$

We use $\oplus$ rather than & merely to keep a distinction between conjunction at this level and conjunction in the condition language. The interpretation is simple:

$$\models \Gamma\,\mathbf{sat}\,(\phi_1 \oplus \phi_2) \quad \Leftrightarrow \quad \models \Gamma\,\mathbf{sat}\,\phi_1 \quad \& \quad \models \Gamma\,\mathbf{sat}\,\phi_2.$$

Conjunction is clearly associative, and we may therefore omit parentheses and write

$$\phi_1 \oplus \phi_2 \oplus \phi_3,$$

for example. We then extend the definitions of our syntactic operations to cover conjunctions. The definition of parallel composition of assertions is a straightforward generalization of the earlier definition. When ϕ and ψ are conjunctions, $[\phi \parallel \psi]$ is defined to be a conjunction: for each conjunct $P\sum_{i=1}^{n}\alpha_i P_i\phi_i$ of ϕ and each conjunct $Q\sum_{j=1}^{m}\beta_j Q_j\psi_j$ of ψ we include in $[\phi \parallel \psi]$ a conjunct of the form:

$$\{\,P\,\&\,Q\,\}(\sum_{i=1}^{n}\alpha_i P_i[\phi_i \parallel \psi] \;+\; \sum_{j=1}^{m}\beta_j Q_j[\phi \parallel \psi_j]).$$

When ϕ and ψ are simple assertions this is exactly the same definition as before. For an example, when ϕ and ψ are the assertions

$$\begin{aligned}\phi &= (\{\,x=0\,\}\alpha\{\,x=1\,\}) \oplus (\{\,x=1\,\}\alpha\{\,x=2\,\}),\\ \psi &= (\{\,x=0\,\}\beta\{\,x=1\,\}) \oplus (\{\,x=1\,\}\beta\{\,x=2\,\}),\end{aligned}$$

the parallel composition has four conjuncts:

$$\begin{aligned}&\{\,x=0\,\}(\alpha\{\,x=1\,\}\psi \;+\; \beta\{\,x=1\,\}\phi),\\ &\{\,x=1\,\}(\alpha\{\,x=2\,\}\psi \;+\; \beta\{\,x=2\,\}\phi),\\ &\{\,\mathbf{false}\,\}(\alpha\{\,x=1\,\}\psi \;+\; \beta\{\,x=2\,\}\phi),\\ &\{\,\mathbf{false}\,\}(\alpha\{\,x=2\,\}\psi \;+\; \beta\{\,x=1\,\}\phi).\end{aligned}$$

For sequential composition we merely put $(\phi_1 \oplus \phi_2);\psi \;=\; (\phi_1;\psi) \oplus (\phi_2;\psi)$ and similarly when we have a conjunction in the second place: in other words, sequential composition distributes over conjunction. With these additions, the axioms and rules given earlier remain sound, with (B3) applicable for conjunction-free assertions as we have not specified a definition of safe(ϕ) when ϕ is a conjunction.

We add rules for conjunction introduction and elimination:

$$\frac{\Gamma\ \mathbf{sat}\ \phi \qquad \Gamma\ \mathbf{sat}\ \psi}{\Gamma\ \mathbf{sat}\ (\phi \oplus \psi)} \tag{B6}$$

$$\frac{\Gamma\ \mathbf{sat}\ (\phi \oplus \psi)}{\Gamma\ \mathbf{sat}\ \phi, \quad \Gamma\ \mathbf{sat}\ \psi} \tag{B7}$$

Implication between assertions is defined as follows for simple assertions without conjunction; the definition extends in the obvious way to conjunctions: we certainly want to have $(\phi \oplus \psi) \Rightarrow \phi$ and $(\phi \oplus \psi) \Rightarrow \psi$ for example. For

$$\begin{aligned} \phi &= P(\sum_{i=1}^{n} \alpha_i P_i \phi_i), \\ \psi &= Q(\sum_{i=1}^{n} \alpha_i Q_i \psi_i), \\ (\phi \Rightarrow \psi) \quad &\Leftrightarrow \quad (Q \Rightarrow P) \ \& \ \bigwedge_{i=1}^{n} (P_i \Rightarrow Q_i) \ \& \ \bigwedge_{i=1}^{n} (\phi_i \Rightarrow \psi_i). \end{aligned}$$

In the case when $n = 0$ this merely requires that $Q \Rightarrow P$. Also, when ϕ is $\{P\}\alpha\{Q\}$ and ψ is $\{P'\}\alpha\{Q'\}$ we have $\phi \Rightarrow \psi$ iff $P' \Rightarrow P$ and $Q \Rightarrow Q'$; this is analogous to the usual Rule of Consequence of conventional Hoare logic [1,12]:

$$\frac{P' \Rightarrow P, \quad \{P\}\Gamma\{Q\}, \quad Q \Rightarrow Q'}{\{P'\}\Gamma\{Q'\}} \tag{C}$$

Our rule for implication is a form of *modus ponens*:

$$\frac{\Gamma\ \mathbf{sat}\ \phi \quad \phi \Rightarrow \psi}{\Gamma\ \mathbf{sat}\ \psi}. \tag{B8}$$

From the definitions above it follows, for example, that

$$\{P\}\alpha\{Q\}\bullet \quad \Rightarrow \quad \{P\}\alpha\{Q\},$$

because $Q \Rightarrow \mathbf{true}$. This means, in particular, that we may derive the following assertion schemas for assignment and skip, by using the axioms (B1) and (B2) together with (B8):

$$\alpha{:}\mathbf{skip}\ \mathbf{sat}\ \{P\}\alpha\{P\}, \tag{B1$'$}$$

$$\alpha{:}I{:=}E\ \mathbf{sat}\ \{[E\backslash I]P\}\alpha\{P\}. \tag{B2$'$}$$

These forms resemble the usual Hoare axioms for these constructs [12].

Examples.

Consider again the problematic examples introduced earlier.

Example 1. We wish to prove that Γ **sat** θ, where

$$\Gamma = [\alpha : x{:=}x+1 \parallel \beta : x{:=}x+1],$$
$$\theta = \{\, x=0 \,\}(\alpha\{\, x=1 \,\}\{\, x=1 \,\}\beta\{\, x=2 \,\} \;+\; \beta\{\, x=1 \,\}\{\, x=1 \,\}\alpha\{\, x=2 \,\}).$$

We have the following assertions (by rules B2 and B6):

$$\alpha : x{:=}x+1 \text{ \textbf{sat} } \phi$$
$$\phi = (\{\, x=0 \,\}\alpha\{\, x=1 \,\}) \oplus (\{\, x=1 \,\}\alpha\{\, x=2 \,\}),$$
$$\beta : x{:=}x+1 \text{ \textbf{sat} } \psi$$
$$\psi = (\{\, x=0 \,\}\beta\{\, x=1 \,\}) \oplus (\{\, x=1 \,\}\beta\{\, x=2 \,\}).$$

We have already seen that $[\phi \parallel \psi]$ is a conjunction of four terms, one of which is

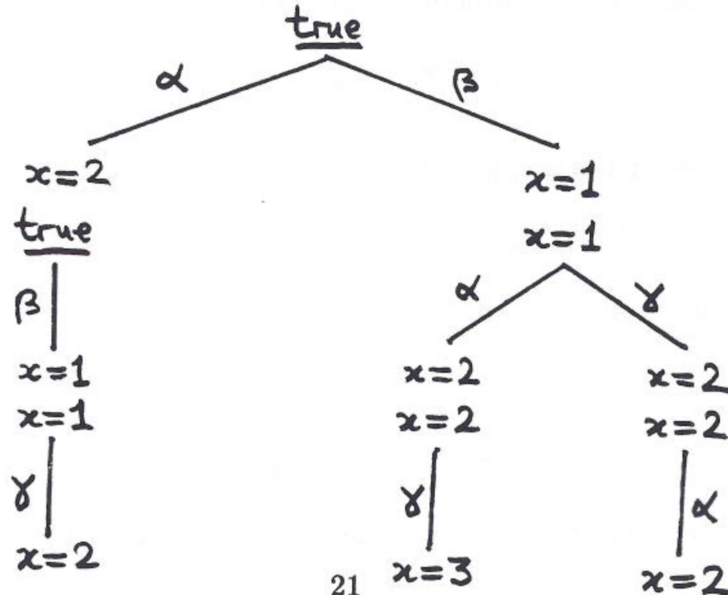
$$\{\, x=0 \,\}(\alpha\{\, x=1 \,\}\psi \;+\; \beta\{\, x=1 \,\}\phi).$$

But $\psi \Rightarrow \{\, x=1 \,\}\beta\{\, x=2 \,\}$ and $\phi \Rightarrow \{\, x=1 \,\}\alpha\{\, x=2 \,\}$. Hence, $[\phi \parallel \psi] \Rightarrow \theta$, and the result follows by (B4) and (B8). ∎

Example 2. In the composition $[\alpha : x{:=}2 \parallel (\beta : x{:=}1 ; \gamma : x{:=}x+1)]$ the two component commands satisfy the assertions

$$\{\, \mathbf{true} \,\}\alpha\{\, x=2 \,\},$$
$$\{\, \mathbf{true} \,\}\beta\{\, x=1 \,\}((\{\, x=1 \,\}\gamma\{\, x=2 \,\}) \oplus (\{\, x=2 \,\}\gamma\{\, x=3 \,\})).$$

The parallel composition of these assertions implies the desired assertion:



Example 3. Let $\Gamma = [\alpha : x{:=}1 \parallel \beta : y{:=}1]$. We wish to prove that $\Gamma\ \mathbf{sat}\ \theta$, where

$$\begin{aligned}\theta = \{\,\mathbf{true}\,\}(\alpha\{\,x=1\,\}\{\,x=1\,\}\beta\{\,x=1\ \&\ y=1\,\}\\ +\beta\{\,y=1\,\}\{\,y=1\,\}\alpha\{\,x=1\ \&\ y=1\,\}).\end{aligned}$$

To this end, let ϕ and ψ be the following assertions:

$$\begin{aligned}\phi &= \{\,\mathbf{true}\,\}\alpha\{\,x=1\,\} \oplus \{\,y=1\,\}\alpha\{\,x=1\ \&\ y=1\,\},\\ \psi &= \{\,\mathbf{true}\,\}\beta\{\,y=1\,\} \oplus \{\,x=1\,\}\beta\{\,x=1\ \&\ y=1\,\}.\end{aligned}$$

Then we have $\alpha : x{:=}1\ \mathbf{sat}\ \phi$ and $\beta : y{:=}1\ \mathbf{sat}\ \psi$. And

$$[\phi \parallel \psi] \Rightarrow \{\,\mathbf{true}\,\}(\alpha\{\,x=1\,\}\psi + \beta\{\,y=1\,\}\phi)$$

By choosing the appropriate conjuncts in ϕ and ψ we see that this assertion implies θ. That completes the proof. ∎

Soundness and Completeness.

Although we do not provide a proof in this paper, the proof system formed by (B0)–(B8) is sound: all provable assertions are valid. The system is also relatively complete in the sense of Cook [8]: every true assertion of the form $\Gamma\ \mathbf{sat}\ \phi$ is provable, given that we can prove all of the conditions necessary in applications of the critical region rule and of *modus ponens.* Both of these rules require assumptions which take the form of implications between conditions. Let **Th** be the set of valid conditions (including implications between conditions). Write $\mathbf{Th} \vdash \Gamma\ \mathbf{sat}\ \phi$ if this can be proved from (B0)–(B8) using assumptions from **Th**. The soundness result is:

Theorem 3. If $\mathbf{Th} \vdash \Gamma\ \mathbf{sat}\ \phi$ then $\models \Gamma\ \mathbf{sat}\ \phi$. ∎

Relative completeness is expressed as follows:

Theorem 4. If $\models \Gamma\ \mathbf{sat}\ \phi$ then $\mathbf{Th} \vdash \Gamma\ \mathbf{sat}\ \phi$. ∎

We omit the proof of this result.

3. Deriving Owicki's proof rules.

In Owicki's proof system, conventional Hoare-style assertions of the form

$$\{P\}\Gamma\{Q\}$$

are used, although the parallel composition rule requires the use of a *proof outline* above the inference line. A proof outline is a command text annotated with conditions, one before and one after each syntactic occurrence of an atomic action. At least for sequential commands, *safe* assertions in our assertion language correspond precisely with such proof outlines because computations of sequential commands follow the syntactic structure of the command. The analogy can be extended to parallel commands too, although the syntactic structure of a proof outline is no longer so close to that of the corresponding safe assertion. The following proof rule forms a connection between our proof system and that of Owicki. Above the line, we have a safe assertion of our form, and below we have a Hoare-style partial correctness assertion. The rule states that a safe assertion implies the partial correctness of the command with respect to its root and leaf conditions. The rule is:

$$\frac{\Gamma \text{ sat } \phi, \qquad \text{safe}(\phi)}{\{\text{root}(\phi)\}\Gamma\{\text{leaf}(\phi)\}} \tag{R}$$

To see why Owicki's proof rule for parallel composition required an extra constraint, that of *interference-freedom*, let us see how to model her rule in our notation.

Owicki's parallel composition rule essentially corresponds to a slightly different form of parallel composition of assertions. This may be defined as follows. For $\phi = P\sum_{i=1}^{n} \alpha_i P_i \phi_i$ and $\psi = Q\sum_{j=1}^{m} \beta_j Q_j \psi_j$ with n and m non-zero, we put

$$[\phi \,\|_O\, \psi] = \{P \,\&\, Q\}(\sum_{i=1}^{n} \alpha_i\{P_i \,\&\, Q\}[\phi_i \,\|_O\, \psi] + \sum_{j=1}^{m} \beta_j\{P \,\&\, Q_j\}\beta_j[\phi \,\|_O\, \psi_j]).$$

We also specify that

$$[\{P\}\text{NIL} \,\|_O\, Q\sum_{j=1}^{m} \beta_j Q_j \psi_j] = \{P \,\&\, Q\}\sum_{j=1}^{m} \beta_j\{P \,\&\, Q_j\}[\{P\}\text{NIL} \,\|_O\, \psi_j],$$

and a similar definition when the terms are exchanged. In particular,

$$[\bullet \,\|_O\, \psi] = [\psi \,\|_O\, \bullet] = \psi.$$

The essential difference between this operation and our earlier one is that this one carries pre-conditions through into post-conditions. For example,

$$[\{P\}\alpha\{P'\} \,\|_O\, \{Q\}\beta\{Q'\}] = \{P \,\&\, Q\}(\alpha\{P' \,\&\, Q\}\{P' \,\&\, Q\}\beta\{P' \,\&\, Q'\} \\ +\beta\{P \,\&\, Q'\}\{P \,\&\, Q'\}\alpha\{P' \,\&\, Q'\}).$$

Unfortunately, this form of composition does not always produce an assertion which correctly describes the behaviour of a parallel composition of commands. We need the notion of *interference-freedom* to guarantee this.

Define the set atoms(ϕ) of *atomic sub-assertions* of ϕ by induction on the depth of ϕ. For the assertion $\phi = P\sum_{i=1}^{n} \alpha_i P_i \phi_i$ we put

$$\text{atoms}(\phi) = \{\, \{P\}\alpha_i\{P_i\} \mid 1 \leq i \leq n \,\} \cup \bigcup_{i=1}^{n} \text{atoms}(\phi_i).$$

A terminal assertion $\{P\}$NIL has no atomic sub-assertions. The interference-free condition is defined as follows:

Definition 2. Two assertions ϕ and ψ are interference-free, written int-free(ϕ, ψ), iff for every pair of atomic assertions

$$\{p\}\alpha\{p'\} \in \text{atoms}(\phi), \qquad \{q\}\beta\{q'\} \in \text{atoms}(\psi),$$

the (ordinary Hoare-style assertions)

$$\begin{gathered} \{p \,\&\, q\}\alpha\{q\} \\ \{p \,\&\, q\}\beta\{p\} \\ \{p \,\&\, q'\}\alpha\{q'\} \\ \{p' \,\&\, q\}\beta\{p'\} \end{gathered}$$

are valid. ∎

Theorem 5. If ϕ and ψ are interference-free then

$$\models \Gamma_1 \,\mathbf{sat}\, \phi, \quad \models \Gamma_2 \,\mathbf{sat}\, \psi \quad \Rightarrow \quad \models [\Gamma_1 \parallel \Gamma_2] \,\mathbf{sat}\, [\phi \parallel_O \psi]. \quad ∎$$

In view of the above theorem we may include the following rule in our system:

$$\frac{\Gamma_1 \,\mathbf{sat}\, \phi, \quad \Gamma_2 \,\mathbf{sat}\, \psi, \quad \text{int-free}(\phi, \psi)}{[\Gamma_1 \parallel \Gamma_2] \,\mathbf{sat}\, [\phi \parallel_O \psi]}. \tag{B9}$$

Note that this theorem and the proof rule are stated in a form applicable to *all* assertions, not just to safe assertions. This can, therefore, be regarded as a slight extension of Owicki's ideas to encompass a more expressive assertion language. The following result shows that interference-feedom guarantees the preservation of safeness.

Theorem 6. If ϕ and ψ are safe and interference-free, then $[\phi \parallel_O \psi]$ is safe. ∎

The root and leaf conditions of this form of parallel composition satisfy the following logical equivalences:

$$\text{root}(\phi \parallel_O \psi) \equiv \text{root}(\phi) \,\&\, \text{root}(\psi)$$
$$\text{leaf}(\phi \parallel_O \psi) \equiv \text{leaf}(\phi) \,\&\, \text{leaf}(\psi).$$

This may be shown by an inductive argument. The fact that roots and leaves fit together in this composition simply by conjunction provides us with an obvious link with Owicki's proof rule for parallel composition. This rule, taken from [20], is:

$$\frac{\text{proofs of } \{P_1\}\Gamma_1\{Q_1\},\ \{P_2\}\Gamma_2\{Q_2\} \text{ interference-free}}{\{P_1 \,\&\, P_2\}[\Gamma_1 \parallel \Gamma_2]\{Q_1 \,\&\, Q_2\}} \qquad \text{(O)}$$

Now proof outlines for the Hoare assertions $\{P_i\}\Gamma_i\{Q_i\}$ correspond to safe assertions ϕ_i such that Γ_i sat ϕ_i, with $\text{root}(\phi_i) = P_i$ and $\text{leaf}(\phi_i) = Q_i$. The interference-freedom of these proof outlines corresponds to interference-freedom of ϕ_1 and ϕ_2. Then $[\phi_1 \parallel_O \phi_2]$ is a safe assertion satisfied by $[\Gamma_1 \parallel \Gamma_2]$, and has root $P_1 \,\&\, P_2$ and leaf $Q_1 \,\&\, Q_2$. Thus, a proof using Owicki's rule can be represented in our system, if we allow the use of (R) and (B9).

Interestingly, the analogy between safe assertions and proof outlines also yields some other connections with conventional Hoare logic. For instance, the sequential composition rule (B5) together with the following property can be used to derive Hoare's rule for sequential composition [12]:

Theorem 7. If ϕ and ψ are safe and $(\text{leaf}(\phi) \Rightarrow \text{root}(\psi))$ then $\phi;\psi$ is safe. ∎

Hoare's rule was:

$$\frac{\{P\}\Gamma_1\{Q\} \qquad \{Q\}\Gamma_2\{R\}}{\{P\}\Gamma_1;\Gamma_2\{R\}}.$$

The derivation relies on the facts that for non-trivial ϕ and ψ we have

$$\text{root}(\phi;\psi) \equiv \text{root}(\phi), \qquad \text{leaf}(\phi;\psi) \equiv \text{leaf}(\psi).$$

Auxiliary variables and auxiliary critical regions.

It is well known [20] that the proof system based on (B0), (B1), (B2), (B3), (C), (B9), (B5) and (R) is not complete for partial correctness assertions. As a simple example, it is impossible even to prove the obviously valid assertion

$$\{x = 0\}[x{:=}x + 1 \parallel x{:=}x + 1]\{x = 2\}$$

using these rules alone. We chose to avoid this problem by introducing conjunctions and implication. This particular assertion, for instance, can be proved by using rule (R) on

the assertion discussed in Example 1 earlier. Owicki achieved completeness by adding "auxiliary variables" to programs and adding new proof rules to allow their use. We can formalise this as follows. We say that a set X of identifiers is *auxiliary* for a command Γ if all free occurrences of identifiers from this set in Γ are inside assignments to identifiers also in X. Thus, for instance, for the command

$$x{:=}x+1;\ y{:=}z;\ a{:=}x$$

the sets $\{y\}, \{y,z\}, \{a,x\}$ and $\{x,y,z,a\}$ are auxiliary, but $\{x\}$ is not. Let us write

$$\Gamma\ \mathrm{aux}\ X$$

when X is an auxiliary set of identifiers for Γ. Given any set X of identifiers and any command Γ, we can define a command $\Gamma\backslash X$ resulting from the deletion in Γ of all assignments to identifiers in X. The definition is syntax-directed:

$$\begin{aligned}
\mathbf{skip}\backslash X &= \mathbf{skip} \\
(I{:=}E)\backslash X &= \mathbf{skip} \qquad \text{if } I \in X \\
&= (I{:=}E) \quad \text{otherwise} \\
(\Gamma_1;\Gamma_2)\backslash X &= (\Gamma_1\backslash X);(\Gamma_2\backslash X) \\
[\Gamma_1 \parallel \Gamma_2]\backslash X &= [(\Gamma_1\backslash X) \parallel (\Gamma_2\backslash X)] \\
\langle\Gamma\rangle\backslash X &= \langle\Gamma\backslash X\rangle.
\end{aligned}$$

With this definition, it is clear (and provable) that if X is auxiliary for Γ then $\Gamma\backslash X$ has the same partial correctness effect on identifiers outside X as Γ does, and $\Gamma\backslash X$ leaves the values of all identifiers in X fixed.

Let free$[\![P,Q]\!]$ stand for the set of identifiers having a free occurrence in either P or Q. Owicki's auxiliary variables rule is:

$$\frac{\{P\}\Gamma\{Q\} \qquad \Gamma\ \mathrm{aux}\ X \qquad \mathrm{free}[\![P,Q]\!] \cap X = \emptyset}{\{P\}\Gamma\backslash X\{Q\}}. \tag{AV}$$

In addition to this rule, for completeness of the Owicki proof system we also need a rule for eliminating "unnecessary" critical regions and irrelevant atomic actions which have been inserted merely to cope with auxiliary variables. The following command equivalences are valid with respect to partial correctness in all contexts:

$$\begin{aligned}
\mathbf{skip};\Gamma &\equiv \Gamma \\
\Gamma;\mathbf{skip} &\equiv \Gamma \\
(\Gamma_1;\Gamma_2);\Gamma_3 &\equiv \Gamma_1;(\Gamma_2;\Gamma_3) \\
\langle\langle\Gamma\rangle\rangle &\equiv \langle\Gamma\rangle \\
\langle\mathbf{skip}\rangle &\equiv \mathbf{skip} \\
\langle I{:=}E\rangle &\equiv I{:=}E \\
[\mathbf{skip} \parallel \Gamma] &\equiv \Gamma \\
[\Gamma \parallel \mathbf{skip}] &\equiv \Gamma
\end{aligned}$$

Owicki's proof system uses a rule based on these equivalences, which we may formalise as follows:

$$\frac{\{P\}\Gamma\{Q\} \quad \Gamma \equiv \Gamma'}{\{P\}\Gamma'\{Q\}}. \tag{EQ}$$

As an example, we can now prove (as in [20]) the assertion

$$\{x = 0\}[x{:=}x + 1 \parallel x{:=}x + 1]\{x = 2\}$$

by first introducing auxiliary variables a and b to tag the two assignments and establishing the assertion

$$\{x = 0\}a{:=}0;\, b{:=}0;\, [\langle a{:=}1;\, x{:=}x + 1\rangle \parallel \langle b{:=}1;\, x{:=}x + 1\rangle]\{x = 2\}.$$

Then we eliminate the auxiliary variables and the extra critical regions. This augmented assertion can be proved by first proving the following assertions for the two parallel components:

$$\begin{aligned}
&\{P_a\}\langle a{:=}1;\, x{:=}x + 1\rangle\{Q_a\},\\
P_a &= (b = 0 \,\&\, x = 0) \vee (b = 1 \,\&\, x = 1),\\
Q_a &= (b = 0 \,\&\, x = 1 \,\&\, a = 1) \vee (b = 1 \,\&\, x = 2 \,\&\, a = 1),\\
&\{P_b\}\langle b{:=}1;\, x{:=}x + 1\rangle\{Q_b\},\\
P_b &= (a = 0 \,\&\, x = 0) \vee (a = 1 \,\&\, x = 1),\\
Q_b &= (a = 0 \,\&\, x = 1 \,\&\, b = 1) \vee (a = 1 \,\&\, x = 2 \,\&\, b = 1).
\end{aligned}$$

These two proof outlines are interference-free (this requires the verification of four conditions), and their use in the parallel rule enables us to conclude

$$\{P_a \,\&\, P_b\}[\langle a{:=}1;\, x{:=}x + 1\rangle \parallel \langle b{:=}1;\, x{:=}x = 1\rangle]\{Q_a \,\&\, Q_b\}.$$

Since we have

$$\begin{aligned}
&\{x = 0\}a{:=}0;\, b{:=}0\{x = 0 \,\&\, a = 0 \,\&\, b = 0\},\\
&x = 0 \,\&\, a = 0 \,\&\, b = 0 \;\Rightarrow\; P_a \,\&\, Q_a,\\
&Q_a \,\&\, Q_b \;\Rightarrow\; x = 2,
\end{aligned}$$

the desired result follows by the usual Hoare rules for sequential composition and the Rule of Consequence.

The Owicki–Gries proof system can, then, be thought of as built from the rules (B0), (B1), (B2), (B3), (B9), (B5), (AV), (EQ) and (C). It is arguable whether or not our proof system, which does not require the use of auxiliary variables in proofs, is preferable to Owicki's. The reader might like to compare the styles of proof in the two systems for the example above. Just as it is necessary to exercise skill in the choice and use of auxiliary variables in Owicki's system, our system requires a judicious choice of conjunctions. However, the details of auxiliary variables and reasoning about their values can be ignored in our system. At least we are able to demonstrate that there are alternatives to the earlier proof rules of [18,19] which do not explicitly require the manipulation of variables purely for proof-theoretical purposes and which do not require a notion of interference-freedom to guarantee soundness.

4. Extensions.

In this section we discuss briefly some effects of extending the programming language. We add the *await* statement (conditional critical region), conditional command, and *while* loop, thus bringing the language more fully into line with the programming language covered in [20].

The syntax for the new constructs will be:

$$\Gamma ::= \textbf{await}\, \beta{:}B\, \textbf{do}\, \gamma{:}\Gamma \ \mid\ \textbf{while}\, \beta{:}B\, \textbf{do}\, \Gamma \ \mid\ \textbf{if}\, \beta{:}B\, \textbf{then}\, \Gamma_1\, \textbf{else}\, \Gamma_2,$$

with B drawn from a syntactic category **BExp** of *boolean expressions* whose syntax will be ignored. We have inserted labels to indicate that the tests are regarded as atomic actions, as is the body of an await statement.

For the semantics of these constructs we add to the transition system the following rules:

$$\frac{s \models \neg B}{\langle \textbf{await}\, \beta{:}B\, \textbf{do}\, \gamma{:}\Gamma,\, s\rangle \xrightarrow{\beta} \langle \textbf{await}\, \beta{:}B\, \textbf{do}\, \gamma{:}\Gamma,\, s\rangle} \quad \text{(A7)}$$

$$\frac{s \models B, \quad \langle \Gamma, s\rangle \to^{*} \langle \textbf{null}, s'\rangle}{\langle \textbf{await}\, \beta{:}B\, \textbf{do}\, \gamma{:}\Gamma,\, s\rangle \xrightarrow{\gamma} \langle \textbf{null}, s'\rangle} \quad \text{(A8)}$$

$$\frac{s \models B}{\langle \textbf{if}\, \beta{:}B\, \textbf{then}\, \Gamma_1\, \textbf{else}\, \Gamma_2,\, s\rangle \xrightarrow{\beta} \langle \Gamma_1, s\rangle} \quad \text{(A9)}$$

$$\frac{s \models \neg B}{\langle \textbf{if}\, \beta{:}B\, \textbf{then}\, \Gamma_1\, \textbf{else}\, \Gamma_2,\, s\rangle \xrightarrow{\beta} \langle \Gamma_2, s\rangle} \quad \text{(A10)}$$

$$\frac{s \models B}{\langle \textbf{while}\, \beta{:}B\, \textbf{do}\, \Gamma,\, s\rangle \xrightarrow{\beta} \langle \Gamma; \textbf{while}\, \beta{:}B\, \textbf{do}\, \Gamma,\, s\rangle} \quad \text{(A11)}$$

$$\frac{s \models \neg B}{\langle \textbf{while}\, \beta{:}B\, \textbf{do}\, \Gamma,\, s\rangle \xrightarrow{\beta} \langle \textbf{null}, s\rangle} \quad \text{(A12)}$$

We assume given the semantics of boolean expressions, so that a satisfaction relation $\models \subseteq S \times \textbf{BExp}$ is known.

Note that these definitions give loops and conditionals the ability to be interrupted after evaluation of the test and before beginning the selected command. Later we will give a non-interruptible version.

With the extended transition system formed by (A1)–(A12), the old proof rules (B0)–(B8) are still valid. The following proof rules correspond to the new constructs, and are closely connected with the new semantic clauses.

$$\frac{\Gamma \textbf{ sat } \phi, \quad \text{safe}(\phi)}{\textbf{await } \beta : B \textbf{ do } \gamma : \Gamma \textbf{ sat } \{\, \text{root}(\phi) \,\&\, B \,\}\gamma\{\, \text{leaf}(\phi) \,\} \bullet} \tag{C7}$$

$$\frac{\textbf{await } \beta : B \textbf{ do } \gamma : \Gamma \textbf{ sat } \phi}{\textbf{await } \beta : B \textbf{ do } \gamma : \Gamma \textbf{ sat } \{\, P \,\&\, \neg B \,\}\beta\{\, P \,\}\phi} \tag{C8}$$

$$\frac{\Gamma_1 \textbf{ sat } \phi_1}{\textbf{if } \beta : B \textbf{ do } \Gamma_1 \textbf{ else } \Gamma_2 \textbf{ sat } \{\, P \,\&\, B \,\}\beta\{\, P \,\}\phi_1} \tag{C9}$$

$$\frac{\Gamma_2 \textbf{ sat } \phi_2}{\textbf{if } \beta : B \textbf{ do } \Gamma_1 \textbf{ else } \Gamma_2 \textbf{ sat } \{\, P \,\&\, \neg B \,\}\beta\{\, P \,\}\phi_2} \tag{C10}$$

$$\textbf{while } \beta : B \textbf{ do } \Gamma \textbf{ sat } \{\, P \,\&\, \neg B \,\}\beta\{\, P \,\} \bullet \tag{C11}$$

$$\frac{\textbf{while } \beta : B \textbf{ do } \Gamma \textbf{ sat } \theta, \quad \Gamma \textbf{ sat } \phi}{\textbf{while } \beta : B \textbf{ do } \Gamma \textbf{ sat } \{\, P \,\&\, B \,\}\beta\{\, P \,\}(\phi; \theta)} \tag{C12}$$

The soundness of these rules is easy to establish. Note the fact that our earlier rule for an unconditional critical region $\langle \Gamma \rangle$ can be derived from the new rule by making the test **true**.

For a non-interruptible version of loops and conditional, in which there is no interruption point between test and body, we change the semantics as follows:

$$\frac{s \models B, \quad \langle \Gamma_1, s \rangle \xrightarrow{\alpha} \langle \Gamma', s' \rangle}{\langle \textbf{if } \beta : B \textbf{ then } \Gamma_1 \textbf{ else } \Gamma_2, s \rangle \xrightarrow{\alpha} \langle \Gamma', s' \rangle} \tag{A9'}$$

$$\frac{s \models \neg B, \quad \langle \Gamma_2, s \rangle \xrightarrow{\alpha} \langle \Gamma', s' \rangle}{\langle \textbf{if } \beta : B \textbf{ then } \Gamma_1 \textbf{ else } \Gamma_2, s \rangle \xrightarrow{\alpha} \langle \Gamma', s' \rangle} \tag{A10'}$$

$$\frac{s \models B, \quad \langle \Gamma, s \rangle \xrightarrow{\alpha} \langle \Gamma', s' \rangle}{\langle \textbf{while } \beta : B \textbf{ do } \Gamma, s \rangle \xrightarrow{\alpha} \langle \Gamma'; \textbf{while } \beta : B \textbf{ do } \Gamma, s' \rangle} \tag{A11'}$$

$$\frac{s \models \neg B}{\langle \textbf{while } \beta : B \textbf{ do } \Gamma, s \rangle \xrightarrow{\beta} \langle \textbf{null}, s \rangle} \tag{A12'}$$

The appropriate proof rules are:

$$\frac{\Gamma_1 \ \mathbf{sat}\ P \sum_{i=1}^{n} \alpha_i P_i \phi_i}{\mathbf{if}\ \beta : B\ \mathbf{do}\ \Gamma_1\ \mathbf{else}\ \Gamma_2\ \mathbf{sat}\ \{ P \ \&\ B \} \sum_{i=1}^{n} \alpha_i P_i \phi_i} \qquad \text{(C9}')$$

$$\frac{\Gamma_2 \ \mathbf{sat}\ P \sum_{i=1}^{n} \alpha_i P_i \phi_i}{\mathbf{if}\ \beta : B\ \mathbf{do}\ \Gamma_1\ \mathbf{else}\ \Gamma_2\ \mathbf{sat}\ \{ P \ \&\ \neg B \} \sum_{i=1}^{n} \alpha_i P_i \phi_i} \qquad \text{(C10}')$$

$$\mathbf{while}\ \beta : B\ \mathbf{do}\ \Gamma\ \mathbf{sat}\ \{ P \ \&\ \neg B \} \beta \{ P \} \bullet \qquad \text{(C11)}$$

$$\frac{\mathbf{while}\ \beta : B\ \mathbf{do}\ \Gamma\ \mathbf{sat}\ \theta, \quad \Gamma\ \mathbf{sat}\ P \sum_{i=1}^{n} \alpha_i P_i \phi_i}{\mathbf{while}\ \beta : B\ \mathbf{do}\ \Gamma\ \mathbf{sat}\ \{ P \ \&\ B \} \sum_{i=1}^{n} \alpha_i P_i (\phi_i ; \theta)} \qquad \text{(C12}')$$

We believe that even with these extensions to the programming language the proof system remains sound and relatively complete. Soundness is straightforward, since the proof rules are based so closely on the operational semantics.

5. Conclusions.

We have described a syntax-directed proof system for semantic properties of commands in a simple parallel programming language. The assertions were chosen to correspond in form to the semantic structure, which itself was chosen to be powerful enough to allow reasoning about partial correctness properties to be carried out by manipulating assertions in a context-independent manner. We discussed some connections with more conventional logics such as the Owicki-Gries proof system.

Various proof systems for concurrent languages proposed by Lamport and others can also be related to our work. Lamport [16] proposed using assertions of the form $\{ P \} \Gamma \{ Q \}$ with the interpretation that in every execution which starts somewhere inside Γ with P true, P remains true until Γ terminates, when Q will be true. Such an assertion corresponds to one of our assertions $P \sum_{i=1}^{n} \alpha_i P_i \phi_i$ in which each P_i (and all other intermediate conditions) are identical to P and all leaf conditions are identical to Q. The proof rule for parallel composition given in [16] was:

$$\frac{\{ P \} \Gamma_1 \{ Q \} \quad \{ P \} \Gamma_2 \{ Q \}}{\{ P \} [\Gamma_1 \parallel \Gamma_2] \{ Q \}}.$$

But our definition of parallel composition of assertions preserves this uniformity property: the parallel composition of (the assertions representing) $\{ P \} \Gamma_1 \{ Q \}$ and $\{ P \} \Gamma_2 \{ Q \}$ will again have leaf Q and each intermediate condition will be P. (In fact, this uniformity

property is preserved both by our $\|$ and by the other form $\|_O$). Thus, either our proof rule (B4) or (O) suffices to derive Lamport's rule. Instead of adding auxiliary variables, Lamport suggested the addition of program labels and simple assertions about them. He suggested using labels λ_i for the control points (or interruption points) of a program, and including in the condition language expressions of the form at(λ), inside(λ), after(λ). Lamport's system requires reasoning about control points and the relationship between them. Since in a Lamport-style assertion the same P has to represent more than one control point at a time, the conditions can get rather large. Indeed, it can be argued that since the same P is serving a multitude of purposes it is more natural to split it up into its components and to attach these components to the control points at which they are intended to hold; this is more in line with our notation, with control points corresponding to nodes in a tree.

The Generalized Hoare Logic of Lamport and Schneider [17] used a similar type of assertion to those of [16], except that they insisted that the post-condition coincide with the pre-condition: they used invariant assertions $\{P\}\Gamma\{P\}$. The interpretation is as before, that whenever an execution begins somewhere inside Γ with P true, P will remain true until termination. Again, their proof rule for parallel composition (essentially, a special case of the one from [16], given above) is representable in our system. Again, control conditions are used inside invariants, so that an invariant is really serving a multitude of purposes and could profitably be split up and distributed to the separate control points.

The *Transition Logic* of Gerth [11] is also has some connection with our work. Gerth's assertions, written $[P]\Gamma[Q]$, are interpreted: every transition that begins somewhere in Γ from a state satisfying P ends in a state satisfying Q. Again, the conditions may involve control assertions. Gerth's rule for parallel composition is:

$$\frac{[P]\Gamma_1[Q] \quad [P]\Gamma_2[Q]}{[P][\Gamma_1 \| \Gamma_2][Q]}.$$

But the assertion $[P]\Gamma[Q]$ can again be rendered in our assertion language as an assertion with a simple structure (alternating P and Q along each branch), and again our parallel composition of assertions has the required effect, producing an assertion representating $[P][\Gamma_1 \| \Gamma_2][Q]$ from representations of $[P]\Gamma_1[Q]$ and $[P]\Gamma_2[Q]$. This again means that Gerth's rule can be derived in our system.

The proof methodology and program development method advocated by Jones [14] uses *rely* and *guarantee* conditions in addition to pre- and post-conditions. Although we have not yet investigated the connection in any detail, it appears that these ideas are somewhat related to ours; roughly speaking, a rely condition might correspond to a pre-condition assumed by every atomic action in an assertion, and a guarantee condition would then be implied by all post-conditions of atomic actions.

Other authors have proposed compositional proof systems for concurrent programs in which the underlying assertions are temporal in nature. In particular, we refer to [4] and [19]. In contrast to these methods, we have avoided temporal assertions at the expense of using conjunction and implication as operations on more highly structured assertions built from conventional pre- and post-conditions. We still obtained a compositional proof system. In fact, our assertions do have some similarity with temporal logic in the sense that an assertion has built into it a specification of the possible atomic actions and the behaviour of the command after each of them, so that one might be able to represent one of our assertions ϕ in a more conventional temporal or dynamic logic.

We also believe that similar ideas to those used in this paper may be adopted in an axiomatic treatment of other forms of parallel programming. In particular, CSP [13] may be axiomatized if we modify the class of assertions to represent the potential for communication and if we design a suitable parallel composition of assertions. In CSP, the inclusion of guarded commands will necessitate a distinction between deadlock (a stuck configuration) and successful termination, but this may be handled by an appropriate choice of assertion language. We plan to investigate this topic in a future paper, and we hope that some connections with earlier work [2,18,27] will become apparent when this is done.

Another possibility for future development is to investigate an appropriate generalization of predicate transformers, weakest pre-conditions and strongest post-conditions (see [10], for example) for parallel commands, using our more general assertions instead of Hoare-style assertions. For instance, there is a reasonable notion of *strongest safe assertion* for a (labelled) command and an initial condition, provided we have *strongest post-conditions* of conventional type for atomic actions. If $\mathbf{sp}[\![\alpha]\!](P)$ is the strongest post-condition of atomic action α with respect to the pre-condition P, we may build a safe assertion $\Phi(\Gamma, P)$ as follows. If the initial actions for Γ (from states satisfying P) are $\{\alpha_1, \ldots, \alpha_n\}$, and if Γ_i is the remaining command after α_i, we put

$$\begin{aligned} \Phi(\Gamma, P) &= P \sum_{i=1}^{n} \alpha_i P_i \phi_i, \\ \text{where} \quad P_i &= \mathbf{sp}[\![\alpha_i]\!](P), \\ \phi_i &= \Phi(\Gamma_i, P_i). \end{aligned}$$

For convenience we put $\Phi(\text{null}, P) = P$. For example, the assertion built in this way from the command $[\alpha\!:x\!:=\!x+1 \,||\, \beta\!:x\!:=\!x+1]$ and the initial condition $x = 0$ is:

$$\{x=0\}(\alpha\{x=1\}\{x=1\}\beta\{x=2\} + \beta\{x=1\}\{x=1\}\alpha\{x=2\}).$$

Of course, when we include loops and conditionals we should be more careful in our definitions, but at least for finite commands this type of strongest safe assertion seems to be of interest. We plan to investigate this topic further.

Another point we should mention is that our assertions described above are all finite, and have been given a rather rigid interpretation: they not only describe the potential computations of a command as beginning with one of a given set of actions, but also specify that each of the actions mentioned in the assertion is indeed possible. It is, of course, possible to relax this interpretation; we are not sure if there would be any benefit to doing so, but it may be worth investigating. Similarly, we would like to try the effect of a different choice of assertions. It is clearly possible to model infinite computations by using recursively defined assertions, perhaps with a version of the μ notation often used for this purpose. Thus, if θ is a variable understood to range over assertions, we might write $\mu\theta.[\{ P \,\&\, \neg B \}\beta\{ P \}\theta]$ for an assertion which would be satisfied by the command **await** $\beta : B$ **do** $\gamma : \Gamma$. Recursive assertions could then be used to build proof rules for loops and conditional critical regions.

Acknowledgements. The author is grateful for discussions with Eike Best, Ed Clarke, Rob Gerth, Jay Misra, and Glynn Winskel.

6. References.

[1] Apt, K. R., Ten Years of Hoare's Logic: A Survey, ACM TOPLAS, vol. 3 no. 4 (October 1981) 431–483.

[2] Apt, K. R., Francez, N., and de Roever, W. P., A proof system for communicating sequential processes, ACM TOPLAS, vol. 2 no. 3 (July 1980), 359-385.

[3] Ashcroft, E. A., Proving assertions about parallel programs, J. Comput. Syst. Sci. 10 (Jan. 1975), 110-135.

[4] Barringer, H., Kuiper, R., and Pnueli, A., Now You May Compose Temporal Logic Assertions, Proc. 16th ACM Symposium on Theory of Computing, Washington, May 1984.

[5] Best, E., A relational framework for concurrent programs using atomic actions, Proc. IFIP TC2 Conference (1982).

[6] Brookes, S. D., On the Relationship of CCS and CSP, Proc. ICALP 83, Springer LNCS (1983).

[7] Brookes, S. D., A Fully Abstract Semantics and Proof System for An ALGOL-like Language with Sharing, CMU Technical Report (1984).

[8] Cook, S., Soundness and Completeness of an Axiom System for Program Verfification, SIAM J. Comput. vol 7. no. 1 (February 1978) 70–90.

[9] Dijkstra, E. W., Cooperating Sequential Processes, in: Programming Languages, F. Genuys (Ed.), Academic Press, NY (1968) 43-112.

[10] Dijkstra, E. W., A Discipline of Programming, Prentice-Hall, New Jersey (1976).

[11] Gerth, R., Transition Logic, Proceedings of the 16th ACM STOC Conference, 1983.

[12] Hoare, C. A. R., An axiomatic basis for computer programming, CACM 12, 10 (Oct. 1969), 576-580.

[13] Hoare, C. A. R., Communicating Sequential Processes, CACM 21, 8 (Aug. 1978), 666-677.

[14] Jones, C. B., Tentative Steps Towards a Development Method for Interfering Programs, ACM TOPLAS vol. 5 no. 4, (October 1983) 596–619.

[15] Keller, R. M., Formal verification of parallel programs, CACM 19,7 (July 1976), 371-384.

[16] Lamport, L., The 'Hoare Logic' of concurrent programs, Acta Informatica 14 (1980), 21-37.

[17] Lamport, L., and Schneider, F., The "Hoare Logic" of CSP, and All That, ACM TOPLAS 6, 2 (April 1984), 281-296.

[18] Levin, G. M., and Gries, D., A proof technique for communicating sequential processes, Acta Informatica 15 (1981), 281-302.

[19] Manna, Z., and Pnueli, A., Verification of Concurrent Programs: The Temporal Framework, in: "The Correctness Problem in Computer Science", ed. R. S. Boyer and J. S. Moore, Academic Press, London (1982).

[20] Owicki, S. S., and Gries, D., An Axiomatic proof technique for parallel programs, Acta Informatica 6 (1976), 319-340.

[21] Owicki, S. S., Axiomatic proof techniques for parallel programs, Ph. D. dissertation, Cornell University (Aug. 1975).

[22] Hennessy, M., and Plotkin, G. D., Full Abstraction for a Simple Parallel Programming Language, Proc. MFCS 1979, Springer LNCS vol. 74, pp. 108–120.

[23] Milner, R., Fully Abstract Models of Typed Lambda-Calculi, Theoretical Computer Science (1977).

[24] Milner, R., A Calculus of Communicating Systems, Springer LNCS vol. 92 (1980).

[25] O' Donnell, M., A Critique of the Foundations of Hoare-Style Programming Logic, CACM vol. 25 no. 12 (December 1982) 927–934.

[26] Plotkin, G. D., A Structural Approach to Operational Semantics, DAIMI Report FN-19, Aarhus University (1981).

[27] Plotkin, G. D., An Operational Semantics for CSP, Proceedings of the W. G. 2.2 Conference, 1982.

[28] Winskel, G., Synchronisation Trees, Proc. ICALP 1983, Springer LNCS vol. 154. (1983).