VIII—Floyd-Hoare Logic

We discuss here a rather syntactic approach to program *correctness* (or *program verification*), beginning with the austere 'language' **ATEN**, and then discussing more realistic programming languages. Later this will all be compared and contrasted with the approach of denotational semantics. In fact, a careful verification of the soundness of the proof system in the syntactic approach would often use a denotational specification of the semantics, although we shall use a more operational semantics in each case; for **ATEN**, just the one originally given. There is also a point of view which sees the F-H approach as an alternative form of specifying the semantics of a programming language.

8.1—Floyd-Hoare Logic for ATEN.

Floyd-Hoare logics are deductive systems for deriving statements of the form $F\{C\}G$, where F and G are $1^{\underline{\text{st}}}$order formulas (the 'pre- and post-conditions'), and C is a command from some imperative language. Here we'll choose that language to be **ATEN**. This choice will probably seem far too simplistic to any reader with experience of serious work in F-H logic, which exists mainly for more practical (and therefore complicated) languages, ones which include features such as procedure declarations and calls. See [**C**]. We shall study such a language in Subsection 8.3. In principle, of course, **ATEN** computes anything that is computable. In any case, commenting about soundness and adequacy (completeness) in the case of a simple language like **ATEN** serves to emphasize all the main general points, without having intricacies which obscure them.

But what do we 'mean' by a string $F\{C\}G$ of symbols? The idea is that this string makes an assertion. That assertion is this : *the formula G will be true for the state resulting from executing the command C, as long as we start from a state for which F is true and for which the execution of C does indeed terminate.* So it will come as no surprise that here, the $1^{\underline{\text{st}}}$order language from which F and G come is $1^{\underline{\text{st}}}$order number theory. In the next subsection, we shall discuss a more general situation with **ATEN**, as well as languages with more complicated features.

While **ATEN** is the command language used, it will also be no surprise that, in the semantics, we shall concentrate a great deal on **N** as the interpretation.

But to begin, it is better to consider an arbitrary interpretation I of the $1^{\underline{\text{st}}}$order number theory language. So I is a non-empty set, together with interpretations of the symbols $+$, $\times$, $<$, 0 and 1 over that set. We won't worry about how the elements of I (real numbers, for example) would be represented in some machine which we think of as implementing **ATEN** commands. Nor will we be concerned about implementing the operations which interpret the symbols $+$ and $\times$.

Definition. The F-H statement $F\{C\}G$ is *true in interpretation* I if and only if, for all $\underline{v}$ such that F is true at $\underline{v}$ and such that, when executed with initial state $\underline{v}$, the command C terminates (say, with state $\underline{w}$), we have that G is true at $\underline{w}$.

So this definition merely makes more formal the earlier explanation of the Floyd-Hoare statement $F\{C\}G$. Recall from [**LM**] that $\underline{v} = (v_0, v_1, v_2, \cdots)$, and also $\underline{w}$, are infinite sequences from the interpretation I. We 'store v_i in bin $\#i$ ', for the execution of C. Note also that F and G are full-blown formulas, not necessarily quantifier-free as with the formulas that are used 'for control' within whdo-commands.

The formulas F and G are also likely to have free variables, since if neither did, the definition above would say little about the behaviour of the command C. More precisely, it would say that, if C is a command with at least one input state where its execution terminates, then the sentence F being true in I implies that the sentence G is true in I (and it would put no restrictions at all on F and G if C 'loops' on all inputs). (There is a text which exposits $1^{\underline{\text{st}}}$order logic, religiously disallowing free variables in formulas, with considerable resulting awkwardness. It seems ironic, in view of the above, that the author is a computer scientist, and the intended audience consists of CSers.)

Note once again that termination of the program is part of the assumptions, not the conclusion in the definition. This is called *partial correctness* to contrast it with *total correctness*, where termination is part of the conclusion. Because of the results of Turing about the halting problem (see [**CM**]), thinking about systems for total correctness brings forward some serious limitations right at the beginning. There is a considerable literature on that topic, but we will only touch on it in the second addendum below. Expressed symbolically in terms from the beginning of [**CM**], a total correctness statement would be saying

$$F \text{ true at } s \implies ||C||(s) \neq err \ \wedge \ G \text{ true at } ||C||(s) \ .$$

The definition above for partial correctness says that truth of $F\{C\}G$ amounts to, for all s,

$$F \text{ true at } s \ \wedge \ ||C||(s) \neq err \quad \Longrightarrow \quad G \text{ true at } ||C||(s) \ .$$

Before getting into the proof system, possibly the reader is puzzled about one thing—surely the standard naive thought about correctness of programs is that the state after execution should be properly related to the state *before* execution? After all, we have emphasized, as most do at an elementary level, the idea that a command (or program) is a recipe for computing a function from states to states. But an F-H statement seems to be only asserting something about 'internal' relations involving the output state, not any relation of it to the input state. The way to deal with this question is to refer to snapshot variables (or history, or ghost, or auxiliary variables)—an example is the easiest way to explain this. Recall the **ATEN** command C whose purpose was to interchange x_1 and x_2. We used bin 3 as an intermediate storage, but the command didn't use any x_i for $i > 3$, so we'll employ x_4 and x_5 as our snapshot variables. An F-H statement whose truth should convince most people of the correctness of C would then be the following :

$$(x_4 \approx x_1 \ \wedge \ x_5 \approx x_2) \ \{C\} \ (x_4 \approx x_2 \ \wedge \ x_5 \approx x_1) \ .$$

Admittedly, this says nothing, strictly speaking, about what happens when we execute C without first making sure that bins 4 and 5 have identical contents to bins 1 and 2 respectively. But it seems that one simply has to *accept* as a kind of meta-semantic fact the obvious statement that the content of bin i is unchanged and irrelevant to the computation, if x_i does not occur in the command. Only believers in voodoo would waste time questioning that.

You can find a number of extra examples of specific F-H derivations in the first section of [**G**]. An exercise would be to deduce the completeness of the system there from that of the one below.

Definition. The deductive system we shall use is given symbolically as follows :

$$\text{(I)} \qquad \frac{F\{C\}G \ , \ G\{D\}H}{F\{(C;D)\}H}$$

(II) $$\frac{\text{empty}}{F^{[x\to t]}\{x \doteqdot t\}F}$$

(III) $$\frac{(F \wedge H)\{C\}F}{F\{\mathsf{whdo}(H)(C)\}(F \wedge \neg H)}$$

(IV) $$\frac{F \to F' \ , \ F'\{C\}G' \ , \ G' \to G}{F\{C\}G}$$

Comments. Each of these is to be regarded as a *rule of inference*, for use in *derivations*. The latter are finite sequences of F-H statements and of formulas $(A \to B)$, ending with an F-H statement. But the rule conclusions $(\frac{}{\text{under the line}})$ are all F-H statements, so we'll need some discussion (below) about how lines $(A \to B)$ can appear in such a derivation. In any case, only rule (IV) uses such formulas. Rule (III) contains a formula H which must be quantifier-free, since it occurs within a whdo-command. And rule (II) may be better regarded as a 'logical' axiom, since it has no premiss.

Definition. A rule $\frac{\alpha_1,\alpha_2,\cdots,\alpha_k}{\beta}$ is *valid* in I if and only if β is true in I whenever all α_i are true in I (as per the above definition for truth of F-H statements, or see [**LM**], p.212-214, for those α_i which are $1^{\underline{\text{st}}}$order formulas).

Definition. A system is *sound* for I if and only if all its rules are valid in I. (But the defined system is written with no dependence on any particular interpretation I, so it is simply *sound* (period!), as the theorem below states. In fact, it has no dependence even on which $1^{\underline{\text{st}}}$order language is used, as we discuss in the following subsection.)

Remark. There is surely no need to get formal about derivations, nor about the formulation, much less the proof, of the fact that all the F-H lines are themselves true in I, in a derivation using a sound system and only premisses that are true in I; in particular the *conclusion* of that derivation (its last line) is true in I. Most derivations have no premisses, certainly ones purporting to show that a particular program is correct. See also the second addendum below, where a discussion of including strings with propositional

connectives, and where we take the viewpoint that the unruly set of all true (in **N**) $1^{\underline{\text{st}}}$order formulas is actually the set of premisses.

Theorem 8.1. *The previous system is sound, as long as rule* (IV) *uses only formulas* $(A \to B)$ *which are true in the interpretation considered.*

Proof. Each rule is pretty obviously valid, but we'll give some details below. This is written out

(1) to give the reader some practice with the definitions; and

(2) since it is a matter of experience in this subject that unsound systems have been published several times, for the more complicated practical languages as in Subsection 8.3, and even for a system using **ATEN**, where one extends the F-H statements by using propositional connectives, as we do in the second addendum below, where these statements are called *assertions*.

See also the four quotes in the 4th subsection, just before the addenda.

Another reason to write this out is to avoid the need to do so later for analogous rules in a much more complicated system for a closer to practical, but messier, command language.

Note also that until we get more explicit about the formulas $(A \to B)$, and even then, one has a particular interpretation in mind when talking about derivations, which is often not the case in pure logic.

For rule (I), if F is true at $\underline{v}$ and $(C;D)$ is executed starting with $\underline{v}$, then we have the following possibilities.

(i) If C loops at $\underline{v}$, then so does $(C;D)$.

(ii) If not, let C output $\underline{w}$ when its input is $\underline{v}$. Then G is true at $\underline{w}$, since we are assuming that $F\{C\}G$ is true.

(iii) Now if D loops at $\underline{w}$, then so does $(C;D)$ at $\underline{v}$.

(iv) If not, let D output $\underline{z}$ when its input is $\underline{w}$. Then H is true at $\underline{z}$, since we are assuming that $G\{D\}H$ is true. But now, $(C;D)$ outputs $\underline{z}$ when its input is $\underline{v}$, so we have shown $F\{(C;D)\}H$ to be true, as required.

For rule (II), if $F^{[x_i \mapsto t]}$ is true at $\underline{v}$, then F is true at $(v_0, v_1, \cdots, v_{i-1}, t^{\underline{v}}, v_{i+1}, \cdots)$, that is, F is true at $\underline{w}$, where the latter is the output state when $x_i \doteqdot t$ is executed on $\underline{v}$, as required. See **[LM]**, pp.211, 226-7, if necessary, for $t^{\underline{v}}$.

For rule (III), assume that H is quantifier-free, that $(F \wedge H)\{C\}F$ is true, and that F is true at $\underline{v}$. Suppose also that $\mathsf{whdo}(H)(C)$ terminates with $\underline{w}$, when executed on $\underline{v}$. Then H is false at $\underline{w}$, so

(A) : $\neg H$ *is true there.*

Also H is true before each execution of C within the execution of $\mathsf{whdo}(H)(C)$.

Then, inductively, F is also true before each such execution, using the truth of $(F \wedge H)\{C\}F$ each time. Using it one last time,

(B) : *F is true after execution of* $\mathsf{whdo}(H)(C)$.

Thus, by (A) and (B), $F \wedge \neg H$ is true after execution of $\mathsf{whdo}(H)(C)$, as required.

Finally, for rule (IV), suppose that $\underline{v}$ is such that F is true there and C terminates when executed with $\underline{v}$ as input. Then F' is also true there, since we are assuming $F \to F'$ is true in I. But now, since we are assuming $F'\{C\}G'$ is true, we see that G' is true after executing C, and so, as required, G also is, since $G' \to G$ is true in I.

Now we shall begin to consider *adequacy* of the system, the reverse of soundness. Is it possible that any true F-H statement $F\{C\}G$ can be derived within the deductive system which we have given? Well, the system hasn't exactly been given, since the formulas $(A \to B)$ allowed in rule (IV) need to be specified. We certainly want them at least to be true in I, as in the soundness theorem. But a fundamental theme from post-1930 $1^{\underline{\text{st}}}$order logic, that *deducibility* can be much weaker than *truth*, has now to be considered.

For the sake of concreteness, from here on in this subsection we shall stick to $\mathbf{N}$ as the interpretation. First let us suppose that we tack on (to the given F-H proof system) some proof system for $1^{\underline{\text{st}}}$order number theory formulae which is sound with respect to the interpretation $\mathbf{N}$.

(For example, the formula $\neg 0 \approx x \ \to \ 0 < x$ might be derivable for use in rule (IV) of the F-H system, a formula which is true in $\mathbf{N}$, but certainly is not logically valid. Indeed, we might take it as one of the axioms for the system.) This $1^{\underline{\text{st}}}$order proof system is expected to be *decidable* or *axiomatic* in that there is, for example, an algorithm for recognizing whether a given formula is an axiom of the system.

Since the F-H system we gave is also decidable, it is a standard consequence that the F-H statements which can be derived using the combined system form a recursively enumerable set (with respect to some Gödel numbering of the set of such statements).

Now consider the derivable statements of the special type as follows :

$$0 \approx 0\{C\}\neg 0 \approx 0 \ .$$

Clearly one can 'automatically' recognize such formulas within an enumera-

tion as above, so we conclude that the derivable formulas of that type form a recursively enumerable set.

But what about the set of all such statements which happen to be true in **N**? It is clear, since $\neg t \approx t$ tends to be false in any interpretation of any $1^{\underline{\text{st}}}$order language (with equality), that $0 \approx 0\{C\}\neg 0 \approx 0$ is true for exactly those commands C which fail to terminate (or *halt*) no matter what the input state is.

For a contradiction, suppose now that every F-H statement which is true in **N** can be derived using our combined system. It follows that the set of C which always fail to halt must be recursively enumerable. But this directly contradicts a well-known theorem—see for example [**CM**], pp.106-7 where that set is shown not to be decidable. But the complement of that set is easily seen to be recursively enumerable (it can be semi-decided by just trying, computational step-by-step, all possible inputs for the command, until and if one is found where it terminates). So the set itself cannot be r.e., since, by [**CM**], **IV-5.3**, decidability is implied for an r.e. set whose complement is also r.e. This contradiction now shows that our combined system is never complete, whatever axiomatic proof system for number theory we use to supplement the four rules forming the F-H system.

For logicians, there is a rather more famous non-r.e. set, namely the set of formulas in $1^{\underline{\text{st}}}$order number theory which are true in **N**. (See, for example, Appendix LL in [**LM**].) One can base an alternative argument to the above on that non-r.e. set as follows. First fix a formula T which is always true, for example, $0 \approx 0$. Now fix a command, $NULL$, which does nothing other than terminate for every input. ('It computes the identity function.') An example might be $x_0 \leftarrow\!\!\div\ x_0$. Now consider the set of all F-H statements of the form $T\{NULL\}G$. Such a statement is true in **N** precisely when G is. So the set of such would be r.e. if we had a combined axiomatic proof system which was complete, and so no such system can exist.

Note that the above arguments depend only on both the F-H proof system for F-H statements and the proof system for $1^{\underline{\text{st}}}$order formulas being *axiomatic*, i.e., the rules and axioms form *decidable* sets. They have no dependence on the particular system defined above, or really even on the choice of **ATEN** as a 'Turing-equivalent' command language. Thus we have

Theorem 8.2. *For no combined axiomatic proof system, using a F-H system involving* $1^{\underline{\text{st}}}$*order formulas combined with a proof system producing formulas true in* **N**, *can we have adequacy (i.e. completeness). That is, some true F-H statements will necessarily be underivable by the combined system.*

So, to hope for completeness, we must have something like a 'non-axiomatic' system for deriving true (in **N**) formulas $(A \to B)$ as input for the final rule in our F-H system. I don't know of any such system which is fundamentally different from simply, in oracle-like fashion, assuming that all (true) such formulas can be used as required. So we'll call this the *oracular F-H system.* A bit surprising (at least to me) is the fact that we now do get completeness—somehow, rule (III), for whdo-commands, does say everything that needs to be said, though that seems to me rather unobvious.

In [**C**], this idea of a non-axiomatic system is expressed that way. However, it is probably best to just regard all derivations in the F-H system as having a set of premisses consisting of all $1^{\underline{\text{st}}}$order formulas which are true in **N**. So we imagine the oracle as being able to decide truth in **N**, not just being able to list all true formulas in **N**. Thus the set of all derivations becomes 'decidable relative to the set of all true formulas in **N**'. And so the set of all deducible F-H statements becomes 'recursively enumerable relative to the set of all true formulas in **N**'. This and the previous singly quoted phrase have precise meanings in recursion theory.

To prove completeness, we need a new concept, the *post relation*, and one good theorem about it, namely, its $1^{\underline{\text{st}}}$order definability.

Definition. Let F be a $1^{\underline{\text{st}}}$order number theory formula. and let C be a command from **ATEN**. Let $(y_1, \cdots, y_n)$ be a list of distinct variables which includes all the variables occurring in C and/or occurring freely in F. Let $Q_{F,C}^{(y_1,\cdots,y_n)}$ be the n-ary relation on natural numbers defined by

$$Q_{F,C}^{(y_1,\cdots,y_n)}(d_1, \cdots, d_n) \iff$$

for at least one input $(d'_1, \cdots, d'_n)$ for which F is true [with $y_i = d'_i$], the command C terminates with output $(d_1, \cdots, d_n)$.

Theorem 8.3. *For any F and C, the relation $Q_{F,C}^{(y_1,\cdots,y_n)}$ is '*$1^{\underline{\text{st}}}$*order definable', in that there is a formula $G = G_{F,C}$ with no free variables other than $y_1, \cdots, y_n$ such that G is true at $(d_1, \cdots, d_n)$ if and only if $Q_{F,C}^{(y_1,\cdots,y_n)}(d_1, \cdots, d_n)$.*

This follows using a major result, essentially due to Gödel, given in [**CM**]

as **V-2.3**, p.175. Take G to be

$$\exists z_1 \cdots \exists z_n \ (F \wedge H) \ ,$$

where $z_1, \cdots, z_n$ are distinct variables, disjoint from $\{y_1, \cdots, y_n\}$, and H is a formula with free variables from the y_i and z_i, such that

$$H^{[\vec{y} \to \vec{d} \ , \ \vec{z} \to \vec{d'}]} \text{ is true in } \mathbf{N} \qquad \Longleftrightarrow \qquad ||C||(\vec{d'}) = \vec{d} \, .$$

The relation on the right-hand side in the display (that is, with input $\vec{d'}$, command C terminates with output $\vec{d}$) is clearly semi-decidable, so Gödel's result assures us that H exists, i.e. the relation is $1^{\underline{\text{st}}}$order definable. It is straightforward to check that G has the required property. (Readers might give the details as an exercise—compare to the details we give for the expressivity result in the second addendum.)

In the next subsection, we'll begin to consider more general $1^{\underline{\text{st}}}$order languages, and associated command languages. We'll refer to such a setup as *expressive* when the statement in **8.3** holds. That implies, for each (F, C), the existence of a formula $G = G_{F,C}$ such that

(i) $F\{C\}G_{F,C}$ is true; and

(ii) for all formulas H, we have that $F\{C\}H$ is true implies that $G_{F,C} \to H$ is true.

More elegantly, this can be stated as

for all formulas H, we have [$F\{C\}H$ is true iff $G_{F,C} \to H$ is true].

(A formula with the properties of $G_{F,C}$ is often referred to as a "strongest postcondition". Pressburger arithmetic for $\mathbf{N}$ is an example of a non-expressive language—just drop the function symbol for multiplication from the language. A famous result says that truth in $\mathbf{N}$ then becomes decidable.)

Now we can state and prove the main result of this section.

Theorem 8.4. *The oracular F-H proof system is* complete*, in that every F-H statement which is true in* $\mathbf{N}$ *can be derived using that system.*

Remark. The statement $F\{C\}G$ is clearly true in each of the following cases. So the theorem guarantees the existence of a derivation. As an exercise, give a more direct argument for the derivation each time.

(i) G is logically valid; any F and C.

(ii) F is the negation of a logically valid formula; any G and C.

(iii) C 'loops' on every input; any F and G.

Actually, (iii) is needed later, and seems not completely obvious. The best I can come up with is this : By (IV), we need only find a derivation for $0 \approx 0\{C\}\neg 0 \approx 0$. Clearly C is not an assignment command. When C has the form $(D;E)$, go through that case of the proof below. When C is $\mathsf{whdo}(H)(D)$, it's certainly simpler than that case of the proof below. Firstly, H is necessarily true for all $\underline{v}$, as we use in the last sentence below. Now (i) gives a derivation for $0 \approx 0\{D\}H$, and hence for $(H \wedge H)\{D\}H$, by (IV), since $H \wedge H \rightarrow 0 \approx 0$ is true. So the while-rule gives one for $H\{\mathsf{whdo}(H)(D)\}(H \wedge \neg H)$. But $0 \approx 0 \rightarrow H$ and $H \wedge \neg H \rightarrow \neg 0 \approx 0$ are both true, so apply (IV) again to finish.

Proof. We proceed by structural induction on $C \in \mathbf{ATEN}$, assuming $F\{C\}G$ to be true in $\mathbf{N}$, and showing how to get a derivation of it. The initial and easier inductive cases will be done first to help orient the reader. The harder inductive case (namely when C is a whdo-command) is a substantial argument, due to Cook [**C**] (though it is far less intricate than his arguments when we have a more serious programming language with procedure calls, as Subsection 8.3 will explain).

Initial case, where $C = \ x \doteleftarrow t$: Assume that $F\{x \doteleftarrow t\}G$ is true in $\mathbf{N}$. Then "G with all free occurrences of x replaced by the term t" is true wherever F is true; that is, $F \rightarrow G^{[x \rightarrow t]}$ is true in $\mathbf{N}$. So the semantically complete (oracular) system for $\mathbf{N}$ gives a derivation of the latter formula. Now use rule (II) to get a derivation of $G^{[x \rightarrow t]}\{x \doteleftarrow t\}G$. To finish, apply rule (IV) as follows:

$$\frac{F \rightarrow G^{[x \rightarrow t]} \ , \ G^{[x \rightarrow t]}\{x \doteleftarrow t\}G \ , \ G \rightarrow G}{F\{x \doteleftarrow t\}G}$$

Inductive case, where $C = \ (D;E)$: Assume $F\{(D;E)\}H$ is true in $\mathbf{N}$. Let G express the post relation for (F,D) . By definition, $F\{D\}G$ is true in $\mathbf{N}$. We'll show $G\{E\}H$ is also true in $\mathbf{N}$, so, by the inductive assumption, they both have derivations within our oracular system, and then rule (I) gives the required result. Given a state $\vec{d}$ where G is true, its definition guarantees a state $\vec{d}'$ where: F is true and D 'produces' $\vec{d}$. But then, if E 'produces' $\vec{d}''$ with input $\vec{d}$, we must have that H is true at $\vec{d}''$, as required, since $(D;E)$ produces $\vec{d}''$ from input $\vec{d}'$, and $F\{(D;E)\}H$ is given to be true.

Inductive case, where $C = \mathsf{whdo}(H)(D)$: Let $y_1, \cdots, y_n$ be the list of all variables which occur in D and/or occur in H and/or occur freely in F or G. Let $z_1, \cdots, z_n$ be distinct variables, and disjoint from the y_i's. Define $L := \exists z_1 \cdots \exists z_n J$, where J is a formula defining the post relation for $(F, \mathsf{whdo}(K)(D))$, where

$$K := H \wedge (\neg y_1 \approx z_1 \vee \cdots \vee \neg y_n \approx z_n) \quad \text{(fortunately without quantifiers!)} .$$

The main property needed for L is that it is true for exactly those $(d_1, \cdots, d_n)$—values of $y_1, \cdots, y_n$ respectively—which satisfy the following property, which we shall refer to as $(*)$:

$(d_1, \cdots, d_n)$ arises as the output after D has been executed a finite (possibly zero) number of times, (1) starting with a state $(d'_1, \cdots, d'_n)$ where F is true; and (2) such that H is true immediately prior to each execution.

Let us assume this, and finish the main proof, then return to prove it.

We have two formulas and a F-H statement below which I claim are true in $\mathbf{N}$:

(i) $F \rightarrow L$: since, if F is true at $(d_1, \cdots, d_n)$, then so is L, by the case of zero executions in $(*)$.

(ii) $L \wedge \neg H \rightarrow G$: since, by $(*)$, a state where L is true and H is false is exactly a state arising after 'successfully' executing $\mathsf{whdo}(H)(D)$ starting from a state where F is true. But since $F\{\mathsf{whdo}(H)(D)\}G$ is here assumed to be true, it follows that G is true in such a state, as required.

(iii) $(L \wedge H)\{D\}L$: since, starting from a state where both L and H are true, executing D one more time as in $(*)$ still gives a state where L is true.

Now we can construct the required derivation of $F\{\mathsf{whdo}(H)(D)\}G$ as follows. By (iii) and the induction in the overall proof of this theorem, there is a derivation of $(L \wedge H)\{D\}L$. But we can then apply the two rules

$$\frac{(L \wedge H)\{D\}L}{L\{\mathsf{whdo}(H)(D)\}(L \wedge \neg H)}$$

$$\frac{F \rightarrow L \ , \ L\{\mathsf{whdo}(H)(D)\}(L \wedge \neg H) \ , \ (L \wedge \neg H) \rightarrow G}{F\{\mathsf{whdo}(H)(D)\}G}$$

to get the desired derivation. The second one displayed is indeed an instance of rule (IV) because of (i), (ii) and the fact that the oracle allows us to use any $(A \rightarrow B)$'s which are true in $\mathbf{N}$.

To actually complete the proof, here are the arguments both ways for the fact that $(*)$ holds exactly when L is true.

In one direction, let $\vec{d}$ be a state where $(*)$ holds. To show L is true in that state, let $\vec{d'}$ be a suitable input as in $(*)$. Now run $\mathsf{whdo}(K)(D)$ starting with state $(\vec{d'}, \vec{e}) = (\vec{d'}, \vec{d})$ for $(\vec{y}, \vec{z})$. This execution terminates with state $(\vec{d}, \vec{e}) = (\vec{d}, \vec{d})$ since the $\vec{e}$-component clearly doesn't change with D's execution, and, of course, $(\vec{d}, \vec{d})$ is a state where the

$$\text{“ } (\neg y_1 \approx z_1 \vee \cdots \vee \neg y_n \approx z_n)\text{-half of } K \text{ ”}$$

is false. Thus, $(\vec{d}, \vec{d})$ is a post state for $(F, \mathsf{whdo}(K)(D))$ since F holds for $(\vec{d'}, \vec{d})$, independently of the $\vec{d}$-half. Thus J is true at $(\vec{d}, \vec{d})$, and so L is true at $\vec{d}$, as required.

Conversely, suppose that L is true at $\vec{d}$, so that, for some $\vec{e}$, the formula J is true at $(\vec{d}, \vec{e})$. Thus there is a state $(\vec{d'}, \vec{e'})$ such that F is true at $\vec{d'}$ and executing $\mathsf{whdo}(K)(D))$ on $(\vec{d'}, \vec{e'})$ terminates with state $(\vec{d}, \vec{e})$. Therefore a finite number of executions of D exists, starting with $\vec{d'}$, where F is true at $\vec{d'}$, and terminating with $\vec{d}$. Also, for each state just prior to each execution, the formula K is true, and so H is also true. Thus $(*)$ holds for $\vec{d}$, as required.

Note how this proof of (relative) adequacy is in fact (relatively) effective, all relative to truth in $\mathbf{N}$ of course. Actually carrying out proofs of F-H statements is a matter of considerable interest, both by hand and mechanically.

8.2—Floyd-Hoare Logic for ATEN$_{\mathcal{L}}$.

Here we discuss generalizations of the theorems just proved. It is very nice that the F-H proof system is so simple, just one rule for each clause in the structural inductive definition of **ATEN**, plus one more rule to tie in with formal deductions in 1$^{\underline{\text{st}}}$order logic. Now, by the last (completeness) theorem, one expects to *prove* the following rules (which seem so fundamental that, a priori, one would have suspected that they (or something close) would need to be *part* of the basic system):

$$\frac{F\{C\}G\ ,\ F\{C\}H}{F\{C\}(G \wedge H)}$$

$$\frac{F\{C\}H\ ,\ G\{C\}H}{(F \vee G)\{C\}H}$$

However, after attempting these as an exercise, the reader may be even more disenchanted with this *oracular* F-H system. The only proofs I know would just go back to the inductive proof of **8.4** and keep pulling the oracle rabbit out of the hat. This hardly seems in the syntactic spirit of formal proof systems, much less in the spirit of something one would hope to automate.

But surely **8.4** has a good deal of syntactic content—we just need to generalize the situation of the last subsection to one where there are genuine axiomatic complete proof systems for truth in 1$^{\underline{\text{st}}}$order logic. One can do this merely by allowing *arbitrary* (in particular, *finite*) interpretations of the 1$^{\underline{\text{st}}}$order number theory language. But for later purposes, it is useful to discuss *arbitrary* 1$^{\underline{\text{st}}}$*order languages plus interpretations*, and the analogue of **ATEN** for them.

So let $\mathcal{L}$ be any 1$^{\underline{\text{st}}}$order language (with equality). Associated with it, one defines a command language **ATEN**$_{\mathcal{L}}$ in exact parallel to the original definition : The atomic commands are assignment commands using terms of the language. The inductive structure will use only whdo-commands and concatenation (;) for sequenced commands as before, with quantifier-free formulas (called "Boolean expressions" by CSers) from the language 'dictating the flow of control' in the former. (These are also used in ite$(H)(C)(D)$-commands, if the if-then-else command construction is added to the language, which is then often referred to as the 'while-language'—as we know, it's extra expressiveness in the informal sense does not increase the computational power, as long as the interpretation used

includes something equivalent to the set of all natural numbers.) For the semantics, one further fixes an interpretation of the language. Each of the usual "bins" contains an element of the interpretation. Then one can talk about F-H statements as before, with the so-called pre- and post- conditions being arbitrary formulas from the language. (Note that in [**C**], for example, the situation is generalized a bit further, with *two* $1^{\underline{\text{st}}}$order languages, one for each of the above uses, one being a sublanguage of the other. This seems not to be a major generalization, and will be avoided here for pedagogical reasons. Of course [**C**] also involves a more complicated command language, very similar to the one discussed in the next subsection.)

Now we note that, whatever the language, for any interpretation which is a *finite* set, truth is actually decidable, so there will certainly be a rather straightforward axiomatic complete proof system. (Expressed alternatively, the set of true formulas with respect to such an interpretation is a decidable set, so it's certainly a recursively enumerable set.) And so, the following theorem, whose proof is essentially the same as that of **8.4**, does have plenty of relevant examples.

To be convincing about that, here is the argument that $(\mathcal{L}, I)$ is expressive, as defined in the theorem just below, as long as I is a finite interpretation of $\mathcal{L}$. Let $x_1, \cdots, x_n$ be the variables occurring in C and/or free in F. Let $E \subset I^n$ be the set of values of those variables where $Q_{F,C}$ is true; that is outputs from C executed on some input where F is true. Now define G to be

$$\vee_{(a_1,\cdots,a_n)\in E}(x_1 \approx a_1 \ \wedge \cdots \wedge x_n \approx a_n) \quad .$$

This clearly does the job (and makes sense since E is a finite set!)

Theorem 8.5. *Assume the pair $\mathcal{L}, I$ is* expressive *: for any F and C there is a formula $G_{F,C}$ which is true at exactly those states which arise as the output from C applied to an input state where F is true.*

Then the combined F-H proof system for $\mathbf{ATEN}_{\mathcal{L}}$*, using the original set of four rules, plus an oracle, as discussed earlier, for (the $1^{\underline{\text{st}}}$order language $\mathcal{L}$ plus its given interpretation I), is itself* complete*, in that every F-H statement which is true in the given interpretation can be derived using that combined system.*

8.3—F-H Logic for more complicated (and realistic?) languages.

The next job is to describe a more 'realistic' version of this theorem, basically Cook's original theorem. Because the command language defined below includes procedure calls and variable declarations, the formulation of the system, and the proof of its soundness and adequacy, all require considerable care.

Now this language will not allow recursive programming, and its semantics uses what is called *dyamic scope*, making even it fairly unrealistic as an actual language in which to write lots of programs. The choice of Cook's language was made partly because of the apparent lack of a complete, painstakingly-careful treatment in the literature. Giving a smoother development would have been, say, a language allowing only *parameterless* procedures, even including recursive programming. Such a language is dealt with briefly in the third addendum below, and in reasonable detail in [**Apt**] (and also in [**deB**] except for disallowing nested procedures).

Cook refers somewhat unspecifically to instances in the literature of F-H systems which turned out inadvertantly to **not** be sound, including earlier versions of the system in [**C**]. The reader might like to look at four quotations from papers on the subject, easily found in the final subsection, before the addenda. These will help brace and motivate you for the painstaking technicalities at times here, though at first reading, maybe skipping a lot of those technicalities is desirable. On the other hand, they explain the need for the caveats given later which circumscribe the language **DTEN** we start with, producing a sublanguage **CTEN**. The language **DTEN** likely admits no complete F-H proof system, but the semantics given below are also probably 'wrong' anyway, without the restrictions to **CTEN**, or something similar. Giving rigorous semantics for complicated command languages has been the subject of some controversy over the years, including how necessary the use of denotational semantics is for this. At any rate, it seems to be only at the point where one has procedures both with parameters and allowing recursive programming that denotational semantics becomes particularly useful (or else if one insists on having GOTO commands, or declarations of functions, etc.) None of these three possibilities will be considered in this write-up. As mentioned above, in the third addendum, we do consider briefly a language with recursive programming, and give an F-H proof system for it, but, for that one, parameters are avoided. Function declarations appear briefly in the first addendum.

Before launching into this, it is worth mentioning a couple of more specific things about earlier instances of technical errors. As indicated below in the middle of heavy technicalities, we do fix one technical error in [**C**] which seems not to have been noticed before. But the statements of the main results of general interest in [**C**] are all correct, modulo the discussion in the next paragraph.

In [**C+**], there is a different correction to [**C**], related to the treatment of the declaration of variables. The two methods given there for fixing this seem a bit ad hoc, so below we have defined the semantics of the language a little differently than is done elsewhere. Rather than just updates to the 'store' or 'contents of the bins', we add also an update to

the function which associates variables to bins. So the semantics of a given command plus input, when defined, is not just a sequence of bin contents, but each term in the sequence has a second component, mapping variables injectively to bins. Actually, the only type of command which changes the latter is the variable declaration block command. This modification of the semantics seems to me a considerably more natural way to do things, and it does take care of the small problem in Cook's original published paper [**C**] for which [**C+**] gave the other fixes. Compare the definitions of *Comp* in [**C**] and [**Apt**] with the definition of *SSQ* below, to see more precisely what is being referred to here.

The two languages considered in this subsection are essentially fragments of ALGOL. We'll define **DTEN** by structural induction, but with a slight complication compared to earlier inductive definitions. There will be both a set of commands and a set of declarations. These are defined simultaneously by a mutual recursive method.

I suspect that a major reason for the popularity of BNF-notation in the CS literature (introduced more-or-less simultaneously with ALGOL around 1960) is that it largely obviates the need for conscious cogitation concerning this sort of structural inductive definition! The language **DTEN** has the "**D**" for "declaration" perhaps. The language **CTEN** is singled out from **DTEN** by a list of restrictions ('caveats') much later, and has "**C**" for caveat, or for Cook perhaps. Ours will be slightly simpler than Cook's language, but only by dropping the **if-thendo-elsedo**-command from **BTEN**, which is redundant in a sense, as mentioned in the last subsection. Dropping it just gets rid of an easy case in several lengthy definitions and proofs. A reader who wants that command constructor can easily add it in, along with its semantics, figure out what the extra rule of inference would need to be (see also Addendum 3 here), and then prove completeness for herself, checking her work against [**C**] if desired.

Definition of the command language.

We define two string sets, COM, of commands, and DEC, of declarations, by simultaneous structural induction. This will depend on a fixed $1^{\underline{\text{st}}}$order language whose name will be suppressed from the notation. The *command language* will of course be independent of which interpretation is used. (But the *semantics* of the command language will depend on the interpretation used for the $1^{\underline{\text{st}}}$order [assertion] language.) We'll define the whole thing below in the same style (a bastardization of BNF-notation) which was used earlier for the denotational semantics of **ATEN**. Recall that X^* is the set of finite strings of elements from some set X. Also we let $X^{*!}$ be the set

of finite strings of *distinct* elements from X.

$$x \in IDE\ ;\ p \in PRIDE\ ;\ \vec{u}, (\vec{x} : \vec{v}) \in IDE^{*!}\ ;\ t \in TRM\ ;\ \vec{t} \in TRM^*\ ;\ H \in FRM_{\text{free}}$$

where TRM is the set of terms from the $1^{\underline{\text{st}}}$order language, FRM_{free} its set of quantifier-free formulas, IDE is its set of variables (or *identifiers*), and $PRIDE$ is a set of identifiers for 'procedures', which might as well be disjoint from IDE. When paired as $(\vec{u}, \vec{t})$ below, we always require that "$\vec{u}$" be disjoint from "$\vec{t}$". The notation "$\vec{t}$" simply means the set of all variables which appear in the various terms t_i. That notation occurs often, to save words.

Here is the **Definition of DTEN**:

$$D, D_1, \cdots, D_k \in DEC\ :=\ \{\ \ \mathsf{new}\ x\ \ ||\ \ \mathsf{proc}\ p(\vec{x} : \vec{v}) \equiv C\ \mathsf{corp}\ \ \}$$

$$C, C_1, \cdots, C_n \in COM\ :=\ \{\ x \doteqdot t \mid \mathsf{call}\ p(\vec{u} : \vec{t})\ ||\ \mathsf{whdo}(H)(C)\ \mid \mathsf{begin}\ D_1; \cdots; D_k\ ;\ C_1; \cdots; C_n\ \mathsf{end}\ \}$$

The last construction allows k and/or n to be 0. And so, in inductive constructions and proofs for **DTEN** or **CTEN**, there will tend to be seven cases: the assignment and call (atomic) commands, the whdo-command, and then the cases $k = 0 = n$ (do-nothing command), $k = 0 < n$ (no declarations), and $k > 0$ (two cases, depending on which type of declaration is the leftmost, D_1, in the list). Actually, our distinction between declarations and commands is a bit moot, since the case $k = 1$ and $n = 0$ gives begin D end as a command for any declaration D.

Notice that a call subcommand can occur without any corresponding procedure declaration within the command; in fact the call command on its own is such a command. This is necessary in order to have inductive constructions and proofs, even if it would seldom occur in a useful program. Thus, below in the semantics, we need an extra component, π, which (sometimes) tells us which procedure (another command) to use when p is called. The object π itself is modified by the semantics to conform with whatever procedure declarations occur within the command being 'semantified'. Especially in the proof system, a version of π becomes more part of the syntax.

The functions s and δ, the other two extra semantic components besides π and the given interpretation are explained and used below. They are needed in order to make a separation between variables and the bins/store/memory,

which can occur in more practical ALGOL-like languages (as opposed to **ATEN/BTEN**, where no distinction is made). Treatments bypassing the bins, e.g. **[Cl1], [Old1]**, can be elegant and useful, but harder to motivate.

We need to modify our notation in **[LM]** for the semantics of $1^{\underline{\text{st}}}$order languages to conform with the above. How to do this will be fairly obvious. We consider functions m from a finite subset of identifiers (or variables) to the underlying set, I, of the interpretation. Further down m will always be $s \circ \delta$, so we frequently consider the 'value' $t^{s\circ\delta} \in I$, for a term t; and whether a formula F is true or not "at $s \circ \delta$".

N.B. "F is true at $s|\delta$" will mean exactly the same thing as "F is true at $s \circ \delta$". This will be convenient notation. But elsewhere, $s|\delta$ is just a convenient way to denote the ordered pair (s, δ). It is different than the function $s \circ \delta$.

To be semi-precise, the element t^m of I is undefined unless all variables in t are in the domain of m; and otherwise, it is what we used to call $t^{\underline{v}}$, when the identifiers are the variables x_i, and where the ith component, v_i, of $\underline{v}$ is $m(x_i)$ if that is defined; and v_i is uninteresting for the majority of i, those with x_i not in the domain of m.

Similarly, a formula being true or false at m can only hold when all free variables in the formula are in the domain of m, and then, at least when the identifiers are the variables x_i, it means the same as the formula being true at $\underline{v}$, which is related to m as in the paragraph above.

Next we discuss the objects π mentioned above. These will be sometimes written as functions

$$PRIDE \supset A \xrightarrow{\pi} COM \times IDE^{*!} \times IDE^{*!} \subset COM \times IDE^{*!} ,$$

so $\pi(p)$ has the form $(K, (\vec{x} : \vec{v}))$, where K is a command, and $(\vec{x} : \vec{v})$ is a list of distinct variables. The colon-divider between the 'x-part' and the 'v-part' will be later used to indicate for example that the v-part is not allowed to appear on the left-hand side of an assignment command, because we will substitute terms that might not be variables for it. The function π is defined for procedure identifiers p from a given **finite** subset, A, of $PRIDE$. This will say which procedure K in COM to activate when p is called. But, as we see below, in the definition of SSQ for the command

$$\mathsf{begin\ proc}\ p(\vec{x} : \vec{v}) \equiv K\ \mathsf{corp}\ ;\ D_1; \cdots ; D_k\ ;\ C_1; \cdots ; C_n\ \mathsf{end}\ ,$$

the value $\pi(p)$ will have been 'changed' to the correct thing if a procedure declaration for the name p has already occurred within the same begin—end-'block' before the call takes place. Interchangeably, we can write

$$\pi(p) \;=\; [\mathsf{proc}\ p(\vec{x} : \vec{v}) \equiv K\ \mathsf{corp}]\ .$$

Since π has been introduced somewhat as a semantic concept, and we wish to use it also syntactically, here is another alternative notation. Just as F, C and G are names for strings of 'meaningless' symbols, so too we shall take " / π " as short for a string

$$/\ p_1(\vec{x_1} : \vec{v_1}) \equiv K_1\ /\ p_2(\vec{x_2} : \vec{v_2}) \equiv K_2\ /\ \cdots\ /\ p_n(\vec{x_n} : \vec{v_n}) \equiv K_n\ \ .$$

We shall have given $PRIDE$ a fixed total order $\prec$, and the finite set $\{p_1 \prec p_2 \prec \cdots \prec p_n\} \subset PRIDE$ will be the domain of π with the inherited order, with $\pi(p_i) \;=\; (K_i, (\vec{x_i} : \vec{v_i}))$.

We shall also make minor use later of a total order on IDE, so assume that one is given to begin.

Though it seems not to appear in exactly this form in the literature, a more fundamental syntactic concept than a command C in a language with procedure calls seems to be a string C/π, where π is the sort of string just above. So these may still be conceived as syntactically generated strings of meaningless symbols. Allowing call-commands on their own, and introducing the $/\pi$-notation, are very convenient for theoretical considerations. An actual program for computation would presumably have a declaration for any call, and so "$/\pi$" would be redundant, and then we can think of it as the empty string (or function with empty domain) for such a program.

Definition of DTEN's semantics.

A preliminary definition needed is for the *'simultaneous' substitution* of terms for *free* variables in commands, as used below in defining SSQ of a call-command with $K^{[\ \vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}\ ]}$. As the author of **[C]** knows perfectly well (though some of his readers need lots of patience to see), defining, as on his pp.94-95, substitution in terms of free variables, then free variables in terms of substitution, is not circular, but can be made as a recursive definition in a sense. But this takes more than a tiny amount of thought, when, as in that paper, the data given by π is not made explicit. Another point motivating a more explicit approach is that substitution cannot always be done in a command; for example, substitution of a term which is *not* a variable for a variable x in a command which includes an assignment to x.

So we'll first define substitution for commands C by structural induction, then for strings π by induction on the length of the string. Let $\vec{z} = (z_1, \cdots, z_n)$ and $\vec{e} = (e_1, \cdots, e_n)$ be finite strings (of the same length) of variables z_i and terms e_i, with the variables distinct.

Simultaneous with the inductive definition below, we need the trivial proof by induction of the fact that $(\mathsf{begin}\ D_*; C_*\ \mathsf{end})^{[\vec{z}\to\vec{e}\ ]}$, if defined, always has the form $\mathsf{begin}\ D'_*; C'_*\ \mathsf{end}$, where the sequences D'_*, C'_* have the same lengths as D_*, C_* respectively.

The substitution $(x \div t)^{[\vec{z}\to\vec{e}\ ]}$ is defined when $x = z_i \Rightarrow e_i$ is a variable, and the answer is $x^{[\vec{z}\to\vec{e}\ ]} \div\ t^{[\vec{z}\to\vec{e}\ ]}$. The substitutions are the obvious substitution into terms as defined in **[CM]**.

The substitution $(\mathsf{call}\ p(\vec{u} : \vec{t}))^{[\vec{z}\to\vec{e}\ ]}$ is not defined when
$(\exists i, j,\ u_i = z_j$ and e_j is not a variable), and also when it would produce repeated variables left of the colon or the same variable occurring on both sides of the colon. When it is defined, the answer is

$$\mathsf{call}\ p(\vec{u}^{\ [\vec{z}\to\vec{e}\ ]} : \vec{t}^{\ [\vec{z}\to\vec{e}\ ]}) \ .$$

This notation means : "do the substitution in each of the terms in both strings".

Define

$$(\mathsf{whdo}(H)(C))^{[\vec{z}\to\vec{e}\ ]} \ := \ \mathsf{whdo}(H^{[\vec{z}\to\vec{e}\ ]})(C^{[\vec{z}\to\vec{e}\ ]})$$

where substitution into formulas is as defined in **[CM]**.

Define

$$(\mathsf{begin\ \ end})^{[\vec{z}\to\vec{e}\,]} \ := \ \mathsf{begin\ \ end}\ .$$

Define

$$(\mathsf{begin}\ C_1; C_*\ \mathsf{end})^{[\vec{z}\to\vec{e}\,]} \ := \ \mathsf{begin}\ C_1^{[\vec{z}\to\vec{e}\,]}; C'_*\ \mathsf{end}\ ,$$

where

$$(\mathsf{begin}\ C_*\ \mathsf{end})^{[\vec{z}\to\vec{e}\,]} \ = \ \mathsf{begin}\ C'_*\ \mathsf{end}\ .$$

To be straightforward about this last one

$$(\mathsf{begin}\ C_1; \cdots; C_k\ \mathsf{end})^{[\vec{z}\to\vec{e}\,]} \qquad \text{is simply} \qquad \mathsf{begin}\ C_1^{[\vec{z}\to\vec{e}\,]}; \cdots; C_k^{[\vec{z}\to\vec{e}\,]}\ \mathsf{end}\ .$$

Define

$$(\mathsf{begin\ new}\ x\ ;\ D_*\ ;\ C_*\ \mathsf{end})^{[\vec{z}\to\vec{e}\,]} \ := \ \mathsf{begin\ new}\ x'\ ;\ D'_*\ ;\ C'_*\ \mathsf{end}\ ,$$

where we let x' be the first variable 'beyond' all variables occurring in the string (including $\vec{z}$ and $\vec{e}$), then let D''_* and C''_* be obtained from D_* and C_* respectively by replacing all occurrences of x by x', and then define the right-hand side in the display by (using the induction on length)

$$\mathsf{begin}\ D'_*; C'_*\ \mathsf{end} \ := \ (\mathsf{begin}\ D''_*; C''_*\ \mathsf{end})^{[\vec{z}\to\vec{e}\,]}\ .$$

The reason that x is first replaced is because some of the e_i may have occurrences of x, and we don't want those x's to be governed by the "new" when they appear after substitution for the z's corresponding to those e's. A similar remark applies in the final case below. (One begins to appreciate why CSers tend to want to have substitution always defined, and tend to give a correspondingly involved definition when doing basic logic and λ-calculus. And we do need to fix a linear order beforehand on the set IDE, to make sense above of the "...first variable 'beyond' all variables ...".)

Finally define

$$(\mathsf{begin\ proc}\ p(\vec{x} : \vec{v}) \equiv K\ \mathsf{corp}\ ;\ D_*\ ;\ C_*\ \mathsf{end})^{[\vec{z}\to\vec{e}\,]} \ :=$$

$$\mathsf{begin\ proc}\ p(\vec{x}' : \vec{v}') \equiv K'\ \mathsf{corp}\ ;\ D'_*\ ;\ C'_*\ \mathsf{end}\ ,$$

where we let $\vec{x}'$ and $\vec{v}'$ be the first strings of distinct variables beyond all variables occurring in the string, then let K', D''_* and C''_* be obtained from K, D_* and C_* respectively by replacing all occurrences of x_i by x'_i and v_j by v'_j, and finally define the rest of the right-hand side in the display by

$$\mathsf{begin}\ D'_*; C'_*\ \mathsf{end}\ :=\ (\mathsf{begin}\ D''_*; C''_*\ \mathsf{end})^{[\vec{z}\to\vec{e}\ ]}\ .$$

For substituting into π itself, define

$$(\ p_1(\vec{x}_1 : \vec{v}_1) \equiv C_1\ /\ p_2(\vec{x}_2 : \vec{v}_2) \equiv C_2\ /\ \cdots\ /\ p_n(\vec{x}_n : \vec{v}_n) \equiv C_n)^{[\vec{z}\to\vec{e}\ ]}\ :=$$

$$(\ p_1(\vec{x}_1 : \vec{v}_1) \equiv C_1\ /\ \cdots\ /\ p_{n-1}(\vec{x}_{n-1} : \vec{v}_{n-1}) \equiv C_{n-1})^{[\vec{z}\to\vec{e}\ ]}\ /\ p_n(\vec{x_n} : \vec{v_n}) \equiv C_n^{[\vec{z'}\to\vec{e'}\ ]}\ ,$$

with $\vec{z'}$ being $\vec{z}$ with all the variables in "$\vec{x}_n$" $\cup$ "$\vec{v}_n$" removed, and $\vec{e'}$ being the terms in $\vec{e}$ corresponding to $\vec{z'}$. Start the induction in the obvious way, with $\emptyset^{[\vec{z}\to\vec{e}\ ]}\ :=\ \emptyset$, the empty string.

The definition of *free variable* or *global variable* in a command is itself inductive, and really it applies not to just a command, but a pair, *command* / π. (We can always take $\pi = \emptyset$.) The induction is on the cardinality of the domain of π, and then, for fixed such cardinality, structural induction on C. (This obviates the need to discuss "substituting the body of p for each call to p, over-and-over till no calls remain", which will only work because we shall be excluding recursive commands.)

$FREE(x \doteqdot t\ /\ \pi)$ consists of x and all variables occurring in t.

$FREE(\mathsf{call}\ p(\vec{u} : \vec{t})\ /\ \pi)$ consists of the u_i, the variables in the t_j, and,

(i) if $p \notin \mathrm{dom}(\pi)$, or if $p \in \mathrm{dom}(\pi)$ with $\pi(p) = (K, (\vec{x} : \vec{v}))$ but $K^{[\ \vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}\ ]}$ does not exist, then nothing else; whereas

(ii) if $p \in \mathrm{dom}(\pi)$ with $\pi(p) = (K, (\vec{x} : \vec{v}))$, and $K^{[\ \vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}\ ]}$ does exist, then include also the set $FREE(K^{[\ \vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}\ ]}\ /\ \pi')$, where π' is π, except that p is removed from its domain. (This is the only of the seven cases where the inductive hypothesis on the cardinality of π is used. N.B. We shall later be excluding recursive commands, and the use of π' rather than π here and later is justified by that.)

$FREE(\mathsf{whdo}(H)(C)\ /\ \pi)$ consists of $FREE(C/\pi)$ and the variables in H. (Of course, H is quantifier-free.)

$FREE(\mathsf{begin\ \ end}\ /\ \pi)$ is empty.

$FREE(\mathsf{begin}\ C_*\ \mathsf{end}\ /\ \pi)$ is the union of the $FREE(C_i/\pi)$—more properly, inductively

$$FREE(\mathsf{begin}\ C_1; C_*\ \mathsf{end}\ /\ \pi)\ :=\ FREE(C_1/\pi) \cup FREE(\mathsf{begin}\ C_*\ \mathsf{end}\ /\ \pi)\ .$$

Define $\quad FREE(\mathsf{begin\ new}\ x\ ;\ D_*\ ;\ C_*\ \mathsf{end}\ /\ \pi)\ :=$

$$FREE(\mathsf{begin}\ D_*\ ;\ C_*\ \mathsf{end}\ /\ \pi) \setminus \{x\}\ ;$$

that is, 'remove' the variable x .

Define $\quad FREE(\mathsf{begin\ proc}\ p(\vec{x} : \vec{v}) \equiv K\ \mathsf{corp}\ ;\ D_*\ ;\ C_*\ \mathsf{end}\ /\ \pi)\ :=$

$$[FREE(K/\pi) \setminus (\text{“}\vec{x}\text{ ”} \cup \text{“}\vec{v}\text{ ”})]\ \cup\ FREE(\mathsf{begin}\ D_*\ ;\ C_*\ \mathsf{end}\ /\ \pi)\ .$$

As indicated earlier, we shall use here (and many times below) notation such as “$\vec{x}$ ” ∪ “$\vec{v}$ ” and just “$\vec{x}$ ” in the obvious way, as the underlying set of variables in the list.

For the 'free' or 'so-called 'global' variables in a command C itself, just take π to be empty; that is, the set is $FREE(C/\emptyset)$.

The real 'meat' of the semantics will be a function $SSQ(C/\pi, s|\delta)$ of two (or three or four) 'variables', the 'State $Se$$Q$uence', as defined below. We shall define, inductively on n, the nth term, $SSQ_n(C/\pi, s|\delta)$ of that sequence, by structural induction on the command C. So it's a form of double induction.

The other two input components for SSQ are as follows, where we assume that an interpretation of our underlying $1^{\underline{\text{st}}}$order language has been fixed once-and-for-all.

We need an **injective** function

$$IDE\ \supset\ B\ \xrightarrow{\ \delta\ }\ \{\mathrm{bin}_i \mid i \geq 0\ \}\ ,$$

so δ 'locates' the variables from a **finite** subset B of variables as having values from suitable bins. Or, as we shall sometimes say, it 'associates' some variables with bins.

And we need

$$s \ : \ \{\mathrm{bin}_i \mid i \geq 0 \} \ \longrightarrow \ I \ ,$$

so s is the 'state', telling us which element of the interpretation I (which 'value') happens to be 'in' each bin. Sometimes the word "state" would rather be used for the composition $s \circ \delta$, or for the pair $s|\delta$.

Definition of the computational sequence SSQ.

The value, $SSQ(C/\pi, s|\delta)$, of the semantic function SSQ will be a non-empty (possibly infinite) sequence $\prec s|\delta \ , \ s'|\delta' \ , \ s''|\delta'' \ , \ \cdots \succ$ of pairs, which is what we envision as happening, step-by-step, to the bin contents and variable associations with bins, when the command C is executed beginning with 'environment' $[s|\delta, \pi]$. Note, from the definition below, that the second, 'δ', component remains constant in the semantics of any command which has no variable declaration block. The domain of SSQ is not *all* quadruples— as we see below, if C includes a call to p and $p \notin \mathrm{dom}(\pi)$, then $SSQ(C/\pi, s|\delta)$ *might* not be defined, but it normally will be if the call is within a block, begin $\cdots$ end, which earlier includes a declaration of p. Furthermore, if C includes an assignment command involving a variable on which δ is not defined, again SSQ might not be defined.

The seven cases above in defining freeness (and substitution) will occur in exactly that same order, each of the three times, immediately below in twice defining SSQ.

First we give the definition of SSQ_n in uncompromising detail; many can skip this and go to the second form of the definition just afterwards. When abort occurs below, that means that SSQ in that case is undefined. Whereas, if non-existent occurs, that means the term of the sequence being considered is undefined because the sequence is too short. Also we let

$$OUT(C/\pi, s|\delta) := \begin{cases} \text{last term in } SSQ(C/\pi, s|\delta) & \text{if the latter is a finite sequence;} \\ \text{indifferent} & \text{otherwise.} \end{cases}$$

Here is the doubly inductive definition of $SSQ_n(C/\pi, s|\delta)$:

When $n = 0$, the seven cases, in order, define $SSQ_0(C/\pi, s|\delta)$ to be, respectively:

(1) abort if some variable in t is not in the domain of δ ; otherwise
$[\mathrm{bin}_i \ \mapsto \ \mathsf{if}\ \delta(x) = \mathrm{bin}_i\ , \mathsf{then}\ t^{s\circ\delta}, \mathsf{else}\ s(\mathrm{bin}_i)\ ] \mid \delta$;

(2) abort if $p \notin \mathrm{dom}(\pi)$, or if $p \in \mathrm{dom}(\pi)$ with $\pi(p) = (K, (\vec{x} : \vec{v}))$ and $K^{[\ \vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}\ ]}$ does not exist ; otherwise $s|\delta$,

(3) abort if some variable in H is not in the domain of δ; otherwise $SSQ_0(C'/\pi, s|\delta)$ or $s|\delta$ depending on whether H is true or false at $s \circ \delta$;

(4) $s|\delta$,

(5) $SSQ_0(C_1/\pi, s|\delta)$,

(6) $s|\delta$,

(7) $s|\delta$.

For the inductive step on n, the seven cases, in order,
define $SSQ_{n+1}(C/\pi, s|\delta)$ to be, respectively:

(1) non-existent;

(2) abort if $p \notin \mathrm{dom}(\pi)$, or if $p \in \mathrm{dom}(\pi)$ with $\pi(p) = (K, (\vec{x} : \vec{v}))$ and $K^{[\ \vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}\ ]}$ does not exist ; otherwise
$SSQ_n(K^{[\ \vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}\ ]}\ /\pi, s|\delta)$ (which of course might itself be undefined) .

(3) abort, if $\delta(y)$ is not defined for some variable y occurring in H; otherwise non-existent, if H is false at $s \circ \delta$; otherwise
$SSQ_{n+1}(C'/\pi, s|\delta)$, if the latter is defined and H is true at $s \circ \delta$; otherwise
$SSQ_{n-k}(\mathsf{whdo}(H)(C')/\pi\ ,\ OUT(C'/\pi, s|\delta))$, if $SSQ(C'/\pi, s|\delta)$ has length $k+1$ with $k \le n$ and H is true at both $s|\delta$ and $OUT(C'/\pi, s|\delta)$.

(4) non-existent,

(5) $SSQ_{n+1}(C_1\ /\pi, s|\delta)$, if $SSQ(C_1\ /\pi, s|\delta)$ has length $k+1$ with $k > n$ (i.e. $SSQ_0, \cdots SSQ_k$ all exist but no others);
$SSQ_{n-k}(\mathsf{begin}\ C_*\ \mathsf{end}\ /\pi, OUT(C_1/\pi, s|\delta))$, if $SSQ(C_1/\pi, s|\delta)$ has length $k+1 \le n+1$ [so $OUT(C_1/\pi, s|\delta) = SSQ_k(C_1/\pi, s|\delta)$];

(6) Define

$$m \ = \ \min\{\ a \ : \ \ \delta(z) = \mathrm{bin}_b \implies b < a\ \}\ ;$$

$$\delta' \ := \ (y \mapsto [\textsf{if } y = x \ \textsf{ then } \mathrm{bin}_m \ \textsf{ else } \delta(y)]) \ ;$$

$$LOUT \text{ as the lefthand component of } OUT \ ;$$

and

$\delta''(z) \ := \ \mathrm{bin}_m$ if z is the first variable after $\mathrm{dom}(\delta)$, but $\delta'' = \delta$ otherwise ;

so $\mathrm{dom}(\delta'')$ =$\mathrm{dom}(\delta) \cup \{z\}$. (This uses the linear order on IDE.)

Then the answer is :
non-existent if $SSQ(\textsf{begin } D_* \ ; \ C_* \ \textsf{end}/\pi \ , \ s|\delta')$ has length less than n ;
$SSQ_n(\textsf{begin } D_* \ ; \ C_* \ \textsf{end}/\pi \ , \ s|\delta')$, if it has length greater than n ;
and
$LOUT(\textsf{begin } D_* \ ; \ C_* \ \textsf{end}/\pi \ , \ s|\delta') \mid \delta''$ if the length is n ; i.e. if
$OUT(\textsf{begin } D_* \ ; \ C_* \ \textsf{end}/\pi \ , \ s|\delta') = SSQ_{n-1}(\textsf{begin } D_* \ ; \ C_* \ \textsf{end}/\pi \ , \ s|\delta')$.

(7) $SSQ_n(\textsf{begin } D_* \ ; \ C_* \ \textsf{end}/\pi' \ , \ s|\delta)$, where π' is defined by

$$q \mapsto [\textsf{if } q = p \ \textsf{ then } (K, (\vec{x} : \vec{v})) \ \textsf{ else } \pi(q)] \ .$$

In 'delicate' cases later in the technical proofs, always refer back, if necessary, to this definition above; but now let us repeat the definition in a more readable form, but one where the 2nd and 3rd cases appear to be circular at first glance, because of the hidden induction on n. We have included a more 'CSer style' explanation, which may help neophytes reading similar things in the CS literature. So here's the definition of the sequence $SSQ(C/\pi, s|\delta)$ as a whole, with the seven cases in the usual order:

(1) $\underline{C \ = \ x \doteqdot t}$: The answer is

$$\prec s' \ : \ \mathrm{bin}_i \ \mapsto [\ \textsf{if } \delta(x) = \mathrm{bin}_i \ , \textsf{then } t^{s \circ \delta}, \textsf{else } s(\mathrm{bin}_i) \ ] \mid \delta \succ$$

(a sequence of length 1), and so $SSQ(\ x \doteqdot t \ / \ \pi, s|\delta)$ is undefined exactly when y is not in $\mathrm{dom}(\delta)$ for some variable y which occurs in the term t.

(2) $\underline{C \ = \ \textsf{call } p(\vec{u} : \vec{t} \ )}$: Here $SSQ(C/\pi, s|\delta)$ is undefined unless the lefthand side just following and $K^{[\ \vec{x} \to \vec{u} \ , \ \vec{v} \to \vec{t} \ ]}$ are both defined. Suppose that

$\pi(p) = (K, (\vec{x} : \vec{v}))$. Then the answer is:

$$\prec \ s|\delta \ , \ SSQ(K^{[\ \vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}\ ]}/\pi, s|\delta) \ \succ \ ,$$

where $\prec A, B \succ$ for two sequences means the first followed by the second, as long as A is finite, but just means A if the latter is an infinite sequence. So $SSQ(\ \mathsf{call}\ p(\vec{u} : \vec{t}\ )/\pi, s|\delta)$ is undefined exactly when p is not in $\mathrm{dom}(\pi)$ or $SSQ(K^{[\ \vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}\ ]}\ /\ \pi, s|\delta)$ is undefined (though that is a highly 'unoperational' criterion—see just below). The case that $K^{[\ \vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}\ ]}$ itself is undefined (perhaps because t_i is not merely a variable for some i such that v_i has an occurrence 'to the left of a colon') will not happen later, because the restrictions defining **CTEN** will forbid that possibility.

A remark is needed here. The command K might very well itself contain some call-subcommands (in fact, even perhaps calling p itself, though that would be excluded in the sublanguage **CTEN** carved out of **DTEN** later). In any case, this seems to destroy the inductive nature of this definition. (A similar remark occurs below for whdo.) But we are simply generating a sequence, by syntactic substitution ('call-by-name' if you like). So the definition above will certainly generate a unique sequence, though one which might be infinite. (Our previous pedantic definition, 'term-by-term', makes this explanation unnecessary.) The most extreme example of this would be where K itself is $\mathsf{call}\ p(\vec{x} : \vec{v}\ )$ and where we take $(\vec{u} : \vec{t}\ )$ to be $(\vec{x} : \vec{v}\ )$. There, the SSQ sequence S is given by the definition as $S = \prec s|\delta, S \succ$. It is a trivial induction on n to show that the nth term of any sequence S satisfying this must always be $s|\delta$ (and obviously S must be an infinite sequence). So S has been defined as the (expected) constant infinite sequence.

Now we resume giving the inductive definition of $SSQ(C/\pi, s|\delta)$.

(3) $\underline{C \ = \ \mathsf{whdo}(H)(C')}$: Recall that

$$OUT(C'/\pi, s|\delta) := \begin{cases} \text{last term in } SSQ(C'/\pi, s|\delta) & \text{if the latter is a finite sequence;} \\ \text{indifferent} & \text{otherwise.} \end{cases}$$

Then the answer is undefined if some variable in H is not in the domain of δ ; and otherwise is :

if H is false at $s \circ \delta$,then $\prec s|\delta \succ$, else

$$\prec SSQ(C'/\pi, s|\delta) \ , \ SSQ(\mathsf{whdo}(H)(C')/\pi, \ OUT(C'/\pi, s|\delta)) \succ \quad .$$

This is not a fixed point equation, in any subtle sense, despite the (non-appearing) left-hand side appearing more-or-less as part of the right-hand side. The given definition generates a state sequence, possibly infinite, in a perfectly straightforward manner. Here it is true that, when $SSQ(C'/\pi, s|\delta)$ is undefined and H is true at $s \circ \delta$, then $SSQ(\ \mathsf{whdo}(H)(C')/\pi, s|\delta)$ is undefined. But $SSQ(\ \mathsf{whdo}(H)(C')/\pi, s|\delta)$ may also be undefined for reasons related to repeated execution of C' possibly 'changing' δ and/or π.

(4) $\underline{C \ = \ \mathsf{begin \ \ end}}$: The answer is $\quad \prec s|\delta \succ \quad .$

(5) $\underline{C \ = \ \mathsf{begin}\ C_1; C_*\ \mathsf{end}}$: where C_* means $C_2; \cdots ; C_k$, including the case $k = 1$ where the latter is blank. Here the answer is:

$$\prec SSQ(C_1/\pi, s|\delta) \ , \ SSQ(\mathsf{begin}\ C_*\ \mathsf{end}/\pi, \ OUT(C_1/\pi, s|\delta)) \succ \ .$$

(6) $\underline{C \ = \ \mathsf{begin\ new}\ x \ ; \ D_* \ ; \ C_*\ \mathsf{end}}$: With m, δ', δ'' and $LOUT$ defined as in the more pedantic version of the definition of SSQ, the answer is:

$$\prec \ s|\delta \ , \ SSQ(\mathsf{begin}D_*; C_*\mathsf{end}/\pi, s|\delta') \ , \ LOUT(\mathsf{begin}D_*; C_*\mathsf{end}/\pi, s|\delta') \mid \delta'' \ \succ$$

The intended effect here is that, when executing this block, the variable x begins with whatever value is in the first of the ultimate consecutive string of 'unassigned to variables' bins, that is, the mth bin (presumably 'placed there as input'). But it is restored to the value, if any, it had before entry to the block, immediately upon exit from the block. Because of the injectivity of δ, the content of the bin, if any, 'to which x was assigned before execution of the block' is unaltered during that execution. In addition, a new variable is now associated with the mth bin after the above is done. This last requirement is the extra semantic requirement not occurring elsewhere. The variable-declaration is the only command construction in the definition of SSQ where

the righthand, 'δ', component of any term in the sequence gets changed from its previous value in the sequence. In other treatments, the computational sequence has only states s as terms, so δ never really changes. The interposing of bins in this definition is often avoided, but it does help at least one's intuition of what's going on.

(7) $\underline{C \ = \ \mathsf{begin\ proc}\ p(\vec{x} : \vec{v}) \equiv K\ \mathsf{corp}\ ;\ D_*\ ;\ C_*\ \mathsf{end}}$: Here the answer is

$$\prec \ s|\delta \ , \ SSQ(\mathsf{begin}\ D_*\ ;\ C_*\ \mathsf{end}/\pi', s|\delta) \ \succ$$

where

$$\pi'(q) \ := \ [\mathsf{if}\ q = p \ \ \mathsf{then}\ (K, (\vec{x} : \vec{v})) \ \ \mathsf{else}\ \pi(q)]) \ .$$

The intended effect here is that, when executing this block, any procedure call, to the procedure named p, will use K as that procedure, but the 'regime' given by π is restored immediately upon exit from the block.

This last four or five pages could all have been compressed into half a page. The wordiness above will hopefully help rather than irritate readers, especially non-CSers who take up the literature in the subject, which is occasionally closer to being machine-readable than human-readable! Always admirably brief, the choice seems to be between (readability and vagueness) or (unreadability and precision). By sacrificing brevity with a leisurely style, I hope to have largely avoided the other two sins.

Below in the proof system, we consider *'augmented' F-H statements*, which are strings of the form $F\{C/\pi\}G$, where π is, as above, a specification of which formal procedure is to be attached to each name from some finite set of procedure names (identifiers).

Here is the semantics of these F-H statements. We assume that the $1^{\underline{\text{st}}}$order (assertion) language has been fixed, so that **DTEN** and **CTEN** are meaningful, and also that an interpretation of the $1^{\underline{\text{st}}}$order language has been given, so that the semantic function SSQ can be defined.

Definition of truth of $F\{C/\pi\}H$:

In a fixed interpretation, say that $F\{C/\pi\}G$ $\mathsf{is\ true}$ if and only if, for any (s, δ) for which F is true at $s|\delta$, and for which $SSQ(C/\pi, s|\delta)$ is defined and is a finite sequence, we have that G is true at $OUT(C/\pi, s|\delta)$.

Note that an undefined $SSQ(C/\pi, s|\delta)$ is treated here exactly as if it were an infinite sequence, i.e. as if it produced an infinite loop.

Before listing the proof system's rules, here are some fundamental facts which one would intuitively expect to be true. In particular, part (b) below basically says that truth of a F-H statement is a property to do with states, s, (and not really to do with the functions, δ, which decide which variable should locate in which bin). Part (a) says that the semantics of a command really is a function of the map from variables to values in the interpretation, and doesn't essentially depend on the 'bins' as an intermediate stage in factoring that function. Much further along in separate places you will find parts (c), (d), (e) and (f). They have easy inductive proofs which we suppress, so they've been stated as close as possible to the location where they are needed (though it would be perfectly feasible to state and prove them here).

Theorem 8.6 (a) *Given* $C, \pi, \delta, s, \underline{\delta}, \underline{s}$ *, suppose that* $\underline{s} \circ \underline{\delta}(y) = s \circ \delta(y)$ *for all* $y \in FREE(C/\pi)$ *(in particular,* $\underline{\delta}$ *and* δ *are defined on all such* y*). Then, for all* i*, the term* $SSQ_i(C/\pi, s|\delta)$ *exists if and only if* $SSQ_i(C/\pi, \underline{s}|\underline{\delta})$ *exists and composing their two components, the two agree on all* $y \in FREE(C/\pi)$*.*

(b) *Given* $F\{C/\pi\}G$*, if there is a* $\underline{\delta}$ *whose domain includes all variables free in* F, G *and all of* $FREE(C/\pi)$*, and such that*

$$\forall \underline{s}\,,\ [F \text{ true at } \underline{s} \circ \underline{\delta} \text{ and } OUT(C/\pi, \underline{s}|\underline{\delta})] \text{ exists}] \implies G \text{ true at } OUT(C/\pi, \underline{s}|\underline{\delta}),$$

then, referring to δ*'s whose domain includes all free variables in* F, G *and all of* $FREE(C/\pi)$*, we have*

$$\forall \delta\, \forall s\,,\ [F \text{ true at } s \circ \delta \text{ and } OUT(C/\pi, s|\delta)] \text{ exists}] \implies G \text{ true at } OUT(C/\pi, s|\delta).$$

(This second display just says that $F\{C/\pi\}G$ is true in the interpretation.)

Proof. Part (a) is proved by induction on i, then for fixed i, by structural induction on C. The seven cases are quite straightforward from the definition of SSQ.

To deduce part (b) from (a), given $F\{C/\pi\}G$ and $\underline{\delta}$ as in the assumptions, let $s|\delta$ be a pair for which F is true at $s \circ \delta$ and $OUT(C/\pi, s|\delta)]$ exists. Now choose a state $\underline{s}$, requiring at least that $\underline{s}(\underline{\delta}(y)) = s(\delta(y))$ for all variables y free in F, G and in $FREE(C/\pi)$. Then part (a) gives us that the sequences

$$SSQ(C/\pi, \underline{s}|\underline{\delta}) \ = \ \prec \underline{s}_1|\underline{\delta}_1\ ,\ \underline{s}_2|\underline{\delta}_2\ ,\ \cdots\ ,\ \underline{s}_n|\underline{\delta}_n \succ$$

and

$$SSQ(C/\pi, s|\delta) \;=\; \prec s_1|\delta_1\ ,\ , s_2|\delta_2\ ,\ \cdots\ ,\ s_n|\delta_n \succ$$

have the same length, and, for all i, we have $\underline{s}_i \circ \underline{\delta}_i(y) = s_i \circ \delta_i(y)$ for all free variables y in F, G and C/π. But now, using this for $i = n$, which gives the OUT-state for the two sets of data, the truth of G at $\underline{s}_n \circ \underline{\delta}_n$ is the same as its truth at $s_n \circ \delta_n$, as required.

This concludes the specification of the semantics for **DTEN**, and so for the sublanguage **CTEN** singled out some paragraphs below. Because **DTEN** allows various possibilities which are forbidden to **CTEN**, it seems quite likely that the F-H proof system, given in the next few paragraphs, would be unsound as applied to the entire **DTEN**.

CTEN and Cook's complete F-H proof system for it.

Though perhaps useless, the system is meaningful for **DTEN**, so we'll continue to delay stating the caveats which delineate the sublanguage **CTEN**. The first four rules below match up exactly with the earlier ones for the language $\mathbf{ATEN}_{\mathcal{L}}$. As an exercise, the reader might give more exactly the connection; that is, show how to regard $\mathbf{ATEN}_{\mathcal{L}}$ plus its semantics as sitting inside **CTEN** plus *its* semantics.

Lines in a derivation using the system about to be defined will have the form $F\{C/\pi\}G$, (as well as some lines being $1^{\underline{\text{st}}}$order formulas). This at first might have seemed a bit fishy, since π had been introduced as a semantic concept. But just as F, C and G are names for strings of 'meaningless' symbols, so too, we shall take " / π " as short for a string

$$/\ p_1(\vec{x_1} : \vec{v_1}) \equiv C_1\ /\ p_2(\vec{x_1} : \vec{v_1}) \equiv C_2\ /\ \cdots\ /\ p_n(\vec{x_n} : \vec{v_n}) \equiv C_n\ \ ,$$

as explained earlier. So derivations can still be thought of as syntactically generated sequences of strings of meaningless symbols.

Validity of rules and soundness of a system of rules mean exactly the same as in the previous subsection.

The system of rules of inference :

(I) $$\frac{F\{C/\pi\}G \ , \ G\{\mathsf{begin}\ C_*\ \mathsf{end}\ /\ \pi\}H}{F\{\mathsf{begin}\ C; C_*\ \mathsf{end}\ /\ \pi\}H}$$

(II) $$\frac{\text{empty}}{F^{[x \to t]}\{x \doteqdot t\ /\ \pi\}F}$$

(III) $$\frac{(F \wedge H)\{C/\pi\}F}{F\{\mathsf{whdo}(H)(C)/\pi\}(F \wedge \neg H)}$$

(IV) $$\frac{F \to F' \ , \ F'\{C/\pi\}G' \ , \ G' \to G}{F\{C/\pi\}G}$$

(V) $$\frac{\text{empty}}{F\{\mathsf{begin}\ \ \mathsf{end}\ /\ \pi\}F}$$

(VI) $$\frac{F^{[x \to y]}\{\mathsf{begin}\ D_*; C_*\ \mathsf{end}\ /\ \pi\}G^{[x \to y]}}{F\{\mathsf{begin\ new}\ x; D_*; C_*\ \mathsf{end}\ /\ \pi\}G}$$

if y is not free in F or G and is not in $FREE(\mathsf{begin}\ D_*; C_*\ \mathsf{end}\ /\ \pi)$.

(VII) $$\frac{F\{\mathsf{begin}\ D_*; C_*\ \mathsf{end}\ /\ \pi'\}G}{F\{\mathsf{begin}\ D; D_*; C_*\ \mathsf{end}\ /\ \pi\}G}$$

if $D = \mathsf{proc}\ p(\vec{x} : \vec{v}) \equiv K\ \mathsf{corp}$, and where π and π' agree except that $\pi'(p)$ is defined and equals $(K, (\vec{x} : \vec{v}))$, whatever the status of p with respect to π.

(VIII) $$\frac{F\{C/\pi'\}G}{F\{\mathsf{call}\ p(\vec{x} : \vec{v})\ /\ \pi\}G}$$

where the procedure declaration $\mathsf{proc}\ p(\vec{x} : \vec{v}) \equiv C\ \mathsf{corp}$ is included in π ; that is, $p \in \mathrm{dom}(\pi)$ and $\pi(p) = (C, (\vec{x} : \vec{v}))$, and where π' agrees with π, except that p is not in the domain of π'. CAUTION: The use of π' here would be inappropriate, except that we shall be disallowing recursive commands.

(IX) $$\frac{F\{C/\pi\}G}{F^{[\ \vec{y}\to\vec{r}\ ]}\{C/\pi\}G^{[\ \vec{y}\to\vec{r}\ ]}}$$

if no variable y_i nor any variable in any term r_i is in $FREE(C/\pi)$.
REMARK: This rule is more general than needed for the proof of completeness. The system would still be complete if we only had call-commands for C and only had $\vec{r}$ as a string of *variables*.

(X) $$\frac{F\{\mathsf{call}\ p(\vec{y}:\vec{w})/\pi\}G}{F^{[\ \vec{y}\to\vec{u}\ ;\ \vec{w}\to\vec{t}\ ]}\{\mathsf{call}\ p(\vec{u}:\vec{t})/\pi\}G^{[\ \vec{y}\to\vec{u}\ ;\ \vec{w}\to\vec{t}\ ]}}$$

if no u_i, is free in F or G, where $(\vec{y}:\vec{w})$ is a list of distinct variables (not necessarily related to the formal parameters $(\vec{x}:\vec{v})$ which π might give to p). Assume also that "$\vec{y}$ " $\cup$ "$\vec{w}$ " is disjoint from "$\vec{u}$ " $\cup$ "$\vec{t}$ ".

(XI) $$\frac{F\{C/\pi'\}G}{F\{C/\pi\}G}$$

if $\pi \subset \pi'$, i.e. they agree except that π might have a smaller domain.

Rule (XI) doesn't occur in [**C**], but neither does the " / π"-notation, so I suppose (XI) *couldn't* occur. But it does seem to be needed. The replacement for the " / π"-notation are phrases such as "with the understanding that all calls to p are according to D", and presumably the effects of rule (XI) are built into that phraseology.

A version of rule (XI) is given backwards (i.e. $\pi' \subset \pi$) in [**Cl1**], B2b), p.138.

Theorem 8.7 *Fix a* $1^{\underline{\text{st}}}$*order language* $\mathcal{L}$*. Let* $\mathbf{CTEN}_{\mathcal{L}}$ *be the sublanguage of* $\mathbf{DTEN}_{\mathcal{L}}$ *given by the list of restrictions* CV_* *just below this theorem. Let* I *be an interpretation of* $\mathcal{L}$*. Consider the F-H proof system obtained by combining* (I) *to* (XI) *with a decision oracle for for* $\mathcal{L}, I$*, i.e. use the often undecidable set of all* F *true in* I *as the set of premisses for all derivations. Then*

(i) that F-H system is sound; and

(ii) as long as $\mathcal{L}, I$ *is expressive with respect to* $\mathbf{CTEN}_{\mathcal{L}}$ *(which it certainly will be when* I *is finite, or when* $\mathcal{L}$ *is number theory and* $I = \mathbf{N}$*) that F-H proof system is also adequate (complete).*

Restrictions defining CTEN as a sublanguage of DTEN.

Firstly recall that already in **DTEN**, the variables $(\vec{x} : \vec{v})$ in a procedure declaration are all different from each other, as are the variables $\vec{u}$ in a procedure call command. And the latter cannot appear as variables in the terms $\vec{t}$.

Next we define $SCOM(K/\pi)$, the *set of subcommands*, to be $\{K\} \cup PSCOM(K/\pi)$, where the set, $PSCOM(K/\pi)$, of *proper* subcommands is defined by induction on the length of π and structural induction:

the empty set for assignment and 'do-nothing commands',

$$SCOM(C_1/\pi) \cup SCOM(\mathsf{begin}\ C_*\ \mathsf{end}/\pi) \quad \text{for} \quad K = \mathsf{begin}\ C_1; C_*\ \mathsf{end}$$

$$SCOM(\mathsf{begin}\ D_*; C_*\ \mathsf{end}/\pi) \quad \text{for} \quad K = \mathsf{begin}\ D_1; D_*; C_*\ \mathsf{end}$$

$$SCOM(C/\pi) \quad \text{for} \quad K = \mathsf{whdo}(H)(C)$$

and

$$SCOM(L^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]}/\pi') \quad \text{for} \quad K = \mathsf{call}\ p(\vec{u} : \vec{t}) \quad ,$$

where $\pi(p) = (L, (\vec{x} : \vec{v}))$ and π' agrees with π, except that p is removed from the domain. There are no proper subcommands if $p \notin$domain(π). (Note that the call command has no proper substrings which are subcommands, but may non-the-less have plenty of subcommands—see **8.6**(c) later where the need for this slightly more subtle and inclusive definition can be motivated. In fact, block commands can also have subcommands which are not, strictly speaking, substrings.)

As with free variables, to get the subcommands of a command on its own, just take π to be empty; that is, use $SCOM(C/\emptyset)$.

Definition of 'indirect activation sequence' , a.k.a. 'i.a.s.'.
An i.a.s. for C/π is a sequence :

$$\mathsf{occ}\ p_1(\vec{u}^{(1)} : \vec{t}^{(\ 1)})/\ /\ K_1(\vec{x}^{(1)} : \vec{v}^{(1)})\ /\ /\ \mathsf{occ}\ p_2(\vec{u}^{(2)} : \vec{t}^{(2)})/\ /\ K_2(\vec{x}^{(2)} : \vec{v}^{(2)})$$
$$\cdots$$
$$\cdots$$
$$/\ /\ \mathsf{occ}\ p_\ell(\vec{u}^{(\ell)} : \vec{t}^{(\ell)})/\ /\ K_\ell(\vec{x}^{(\ell)} : \vec{v}^{(\ell)})\ /\ /\ \mathsf{occ}\ p_{\ell+1}(\vec{u}^{(\ell+1)} : \vec{t}^{(\ell+1)})/\ /\ \cdots$$
$$\cdots$$
$$\cdots$$
$$\mathsf{occ}\ p_n(\vec{u}^{(n)} : \vec{t}^{(n)})/\ /\ K_n(\vec{x}^{(n)} : \vec{v}^{(n)}) \ ,$$

(which may as well be finite)

such that the following hold.

(a) Its start term, $\mathsf{occ}\ p_1(\vec{u}^{(1)} : \vec{t}^{(1)})$, is an occurrence of $\mathsf{call}\ p_1(\vec{u}^{(1)} : \vec{t}^{(1)})$ as a subcommand in $SCOM(C/\emptyset)$.

(b) If $\mathsf{occ}\ p_1(\vec{u}^{(1)} : \vec{t}^{(1)})$ is an occurrence in a block of $SCOM(C/\emptyset)$ which includes a declaration of p_1, then that declaration is

$$\mathsf{proc}\ p_1(\vec{x}^{(1)} : \vec{v}^{(1)}) \equiv K_1\ \mathsf{corp}\ ;$$

otherwise $\pi(p_1) = K_1(\vec{x}^{(1)} : \vec{v}^{(1)})$.

(a+) For all $\ell \geq 1$, $\mathsf{occ}\ p_{\ell+1}(\vec{u}^{(\ell+1)} : \vec{t}^{(\ell+1)})$ is an occurrence of $\mathsf{call}\ p_{\ell+1}(\vec{u}^{(\ell+1)} : \vec{t}^{(\ell+1)})$ as a subcommand in $SCOM(K_\ell/\emptyset)$.

(b+) For all $\ell \geq 1$, if $\mathsf{occ}\ p_{\ell+1}(\vec{u}^{(\ell+1)} : \vec{t}^{(\ell+1)})$ is an occurrence in a block of $SCOM(K_\ell/\emptyset)$ which includes a declaration of $p_{\ell+1}$, then that declaration is

$$\mathsf{proc}\ p_{\ell+1}(\vec{x}^{(\ell+1)} : \vec{v}^{(\ell+1)}) \equiv K_{\ell+1}\ \mathsf{corp}\ ;$$

otherwise $\pi(p_{\ell+1}) = K_{\ell+1}(\vec{x}^{(\ell+1)} : \vec{v}^{(\ell+1)})$.

This is sometimes expressed informally by saying that calling the procedure p_1 *indirectly activates* all the p_ℓ for $\ell > 1$ (but perhaps activates other procedures as well, of course.)

Definition of CTEN/. Say that $C/\pi \in$ **CTEN/** if and only if the following six (really five) caveats hold :

$\underline{CV_1}$ A given procedure identifier p occurs in at most one procedure declaration within C together with all the indirect activation sequences for C/π.

REMARKS: It may occur as a declaration occurrence in a command which appears several times, once each in several i.a.s.'s for C/π . Also, a particular case of this caveat is that a given p is declared in C itself at most once, a sanitary requirement which simplifies the previous statement of the semantics, i.e. of SSQ. But also, in any i.a.s. for C/π, we have $p_n \neq p_1$. So this excludes 'recursive programming'.

In any indirect activation sequence for C/π, the caveats labelled with a "+" below hold :

$\underline{CV_2}$ If $\pi(p) = (K, (\vec{x} : \vec{v}))$ or if $\mathsf{proc}\ p(\vec{x} : \vec{v}) \equiv K\ \mathsf{corp}$ is a declaration in a subcommand in $SCOM(C/\emptyset)$, then the latter has no subcommands of the form $v_j \leftarrow t$ or $\mathsf{call}\ (\cdots v_j \cdots : \cdots)$; i.e. no v_j to the left of a colon.

$\underline{CV_2+}$ For $s \geq \ell \geq 1$, no subcommand in $SCOM(K_s/\emptyset)$, has the form $v_j^{(\ell)} \leftarrow t$, or the form $\mathsf{call}\ (\cdots v_j^{(\ell)} \cdots : \cdots)$.

$\underline{CV_3}$ If $\pi(p) = (K, (\vec{x} : \vec{v}))$ or if $\mathsf{proc}\ p(\vec{x} : \vec{v}) \equiv K\ \mathsf{corp}$ is a declaration in a subcommand in $SCOM(C/\emptyset)$, and if $\mathsf{call}\ (\vec{u} : \vec{t}) \in SCOM(C/\emptyset)$, then "$\vec{u}$ " $\cup$ "$\vec{t}$" is disjoint from $FREE(K/\pi)$.

$\underline{CV_3+}$ For $s \geq \ell \geq 1$, the set "$\vec{u}^{(\ell)}$ " $\cup$ "$\vec{t}^{(\ell)}$ " is disjoint from $FREE(K_s/\emptyset)$.

REMARK: The caveat CV_3 is just the case $s = 1 = \ell$ in CV_3+ (in one particular very short i.a.s.), but we state it separately because of the frequency to which it is referred later.

$\underline{CV_4+}$ For $s > \ell \geq 1$, the set "$\vec{x}^{(\ell)}$ " $\cup$ "$\vec{v}^{(\ell)}$ " is disjoint from $FREE(K_s/\emptyset)$.

REMARK: For $s > \ell = 1$, this says the formal parameters in the declaration of any called procedure are not free in any other procedure body which can be indirectly activated by the call.

REMARK: All the cases $\ell > 1$ above appear to be subsumed by simply saying that all K_s/π are in **CTEN/**; but that's a kind of circularity which is easy to hide when speaking glibly (non-technically)!

Proof of 8.7: Note that this proof occupies much of the remainder of this work. 'Logically', soundness should be proved first, but we'll start with adequacy, to make it clear why we need all those rules, before going to the trouble of checking their validity.

For adequacy, we proceed to show (in about 18 pages!) that if $F\{C/\pi\}G$ is true, then we can find a derivation for it. The proof is by induction on the cardinality of the domain of π, and then, for fixed such cardinality, structural induction on $C \in$ **CTEN**. (This takes the place of the need to discuss "substituting the body of p for each call to p, over-and-over till no calls remain", which depends on CV_1.) As with the definition of free variables, it's only when doing the hardest case, a call-command, that we need to appeal to the induction on the cardinality of the domain of π.

When C is an assignment command or whdo-command, or has the form $\mathsf{begin}\ C_1; C_*\ \mathsf{end}$, the arguments are virtually identical to the ones given in

the proof of **8.4**, modulo the somewhat different notation for semantics. So these cases will be omitted here, except for the following remarks. Note that this is where the "expressive" hypothesis in the theorem is needed, both for whdo-commands, and blocks without declarations (i.e. concatenation of commands). The required expressiveness is formulated and used as follows.

Given F and C/π, let

$$D_{F,C/\pi} := \{\, OUT(C/\pi, s|\delta) \;:\; F \text{ is true at } s|\delta \;\text{ and }\; SSQ(C/\pi, s|\delta) \text{ is finite} \,\}\,.$$

Then there is a formula $G = G_{F,C/\pi}$ such that G is true at $s' \circ \delta'$ if and only if $s'|\delta' \in D$. Thus, we get that $F\{C/\pi\}H$ is true iff $G \to H$ is true in the interpretation I.

To deal with begin $C_1; C_*$ end in this adequacy proof, use $C = C_1$ just above to produce an intermediate formula G, and proceed as in **8.4**.

REMARKS: It is interesting to analyze the counterexample given in [**C+**] with respect to where the proof in [**C**] goes wrong. In [**C+**], the two possible fixes are presented without any such analysis. This analysis here is a justification for inserting that last term in the sequence SSQ for variable-declaration-commands.

The example has the form $(C; D)$, where we simplify the concatenation notation with brackets instead of begin... end, and also drop all the $/\pi$, which are irrelevant to the point being made. Here, for a pair of distinct variables x and y,

$$C := \text{begin new } x \;;\; x \Leftarrow 1 \text{ end} \qquad \text{and} \qquad D := \text{begin new } x \;;\; y \Leftarrow x \text{ end} \;\;.$$

The semantics in [**C**] is the same as here with one exception: drop the last term in the defined SSQ here for variable-declaration-commands. In that semantics, δ never gets changed for the output, so SSQ is given simply as a sequence of s's. With that semantics (but not with ours, nor the fixes suggested in [**C+**]), it is easy to see that the F-H statement $Tr\{(C; D)\}y \approx 1$ is true. Here Tr is any logically valid formula. But that statement cannot be derived with the proof system. The completeness argument in [**C**] appears to break down for the case of concatenation, the difficulty turning on the following fine point concerning the existence of a strongest post-condition :

A re-wording of the fact we use above is

$$\forall F \forall C \forall \pi \exists G\; (\forall m\; G \text{ is true at } m \iff$$

$$\exists\, s|\delta \text{ with } F \text{ true at } s \circ \delta \text{ and } OUT(C/\pi, s|\delta) = s'|\delta' \text{ with } m = s' \circ \delta'.$$

In the semantics of [**C**] one can prove the corresponding

$$\forall F \forall C \forall \pi \forall \delta \exists G\; (\forall m\; G \text{ is true at } m \iff$$

$$\exists\, s \text{ with } F \text{ true at } s \circ \delta \text{ and } OUT(C/\pi, s|\delta) = s' \text{ with } m = s' \circ \delta.$$

But what is apparently needed for the completeness argument there is the more stringent requirement in which we just permute two quantifiers; that is, replace $\forall\delta\exists G$ with $\exists G\forall\delta$ just above. The example above shows by direct arguments that the more stringent requirement is false : On the one hand, the F-H statement $G\{D\}y \approx 1$ can only be true for a formula G which is never true (e.g. the negation of a logically valid formula). On the other hand, a formula G which satisfies the more stringent requirement, for C as in the example and $F = Tr$, is necessarily itself also logically valid. But, as in the proof of **8.4**, such a strongest postcondition G must make $G\{D\}H$ true, given that $Tr\{(C; D)\}H$ is true. So no strongest postcondition in the sense of the more stringent requirement can exist, for $F = Tr$ and C in the example.

Using our semantics here with H being $y \approx 1$, the statement $Tr\{(C; D)\}H$ simply isn't true, so the contradiction doesn't arise.

When $C =$ begin end, axiom (V) gives a derivation of $F\{C/\pi\}F$. The truth of $F\{C/\pi\}G$ shows that the formula $F \to G$ is true in I. Now just apply rule (IV) with numerator

$$F \to F \quad , \quad F\{C/\pi\}F \quad , \quad F \to G$$

to get the required derivation.

The next two of the final three cases are straightforward, but a little intricate in details, so we'll give them both carefully. We still won't need any of the caveats CV carving **CTEN** out as a subset of **DTEN** for these two. And the argument in each these cases rests on the one obviously relevant rule.

Suppose that $C =$ begin new $x; D_*; C_*$ end . Define $C_- :=$ begin $D_*; C_*$ end. Assume that $F\{C/\pi\}G$ is true. Fix any variable y different from x and not occurring in any of F or G or D_* or C_*. We shall prove that

$$(*) \qquad F^{[x\to y]}\{C_-/\pi\}G^{[x\to y]} \quad \text{is true.}$$

Then, by structural induction, the displayed F-H statement has a derivation. So the proof is completed by an application of rule (VI) to get the required derivation of $F\{C/\pi\}G$.

Let $\mathcal{F}$ be the set of variables which occur in F and/or G and/or $FREE(C/\pi)$. Choose δ' , with domain equal to $\mathcal{F} \cup \{x, y\}$, by first defining it arbitrarily (but injectively!) on $\mathcal{F}\setminus\{x, y\}$, then defining $\delta'(y) = \text{bin}_a$ and $\delta'(x) = \text{bin}_{a+1}$, for some a with $a > i$ for all i with $\text{bin}_i \in \underline{\delta}(\mathcal{F} \setminus \{x, y\})$.

To prove $(*)$, by **8.6**(b), it suffices to prove the following: given s such that $F^{[x \mapsto y]}$ is true at $s \circ \delta'$, and that $SSQ(C_-/\pi, s|\delta')$ is a finite sequence, we must show

$(**)$ $\qquad\qquad G^{[x \mapsto y]}$ is true at $OUT(C_-/\pi, s|\delta')$.

Define δ so that it is defined exactly on all variables other than y in the domain of δ', with $\delta(x) = \delta'(y) = \mathrm{bin}_a$, and δ agrees with δ' everywhere else on its domain.

Now δ' (except for being defined on y, which is irrelevant, as y doesn't occur in C) is obtained from δ as in the definition of SSQ for C in terms of SSQ for C_-. Thus $LOUT(C/\pi, s|\delta) = LOUT(C_-/\pi, s|\delta')$.

Since $s \circ \delta$ and $s \circ \delta'$ agree on all variables except that $s \circ \delta(x) = s \circ \delta'(y)$ (and except that $s \circ \underline{\delta}'(x)$ is defined but irrelevant), and since $F^{[x \mapsto y]}$ is true at $s \circ \delta'$, it follows that F is true at $s \circ \delta$.

Since $F\{C/\pi\}G$ is true, it now follows that G is true at $OUT(C/\pi, s|\delta)$.

Using this, we now obtain that $G^{[x \mapsto y]}$ is true at $OUT(C_-/\pi, s|\delta')$, which is $(**)$, as required. This last step is immediate because the corresponding $LOUT$s agree (noted above), and because the $ROUT$s agree except that

$$ROUT(C/\pi, s|\delta)(x) \;=\; \delta''(x) \;=\; \delta(x) \;=\; \delta'(y) \;=\; ROUT(C_-/\pi, s|\delta')(y)\ ,$$

and they may disagree on variables beyond those occurring free in G. (Recall that $ROUT(C/\pi, s|\delta)$ is the δ'' in the SSQ definition for variable-declaration-commands.)

For the penultimate case, suppose that $C = \mathsf{begin}\ D; D_*; C_*\ \mathsf{end}$ for some procedure declaration D. Define $C' = \mathsf{begin}\ D_*; C_*\ \mathsf{end}$. As usual, we are given that $F\{C/\pi\}G$ is true, and we want a derivation for it.

Define π' exactly the way π' and π are related in rule (VII); that is, they agree except that $\pi'(p)$ is defined and equals $(K, (\vec{x} : \vec{v}))$, whatever the status of p with respect to π, where D is $\mathsf{proc}\ p(\vec{x} : \vec{v}) \equiv K\ \mathsf{corp}$. By the definition of $SSQ(C/\pi, s|\delta)$ in terms of $SSQ(C'/\pi', s|\delta)$, it follows that

$$OUT(C/\pi, s|\delta) = OUT(C'/\pi', s|\delta)\ .$$

(Again, the two SSQ sequences agree, except for a spurious copy of $s|\delta$ at the beginning of one of them.)

From the display and the truth of $F\{C/\pi\}G$, it is immediate that $F\{C'/\pi'\}G$ is true. And so, by induction, it has a derivation. Now application of rule (VII) gives a derivation for $F\{C/\pi\}G$, as required.

Finally we come to the delicate case where C is a call-command. Here we apparently need the full force of the caveats which disallow many **DTEN**-commands. As far as I can determine, cases of languages closer to full ALGOL than **CTEN**, and proof systems which are sound and complete for the F-H statements corresponding to such languages, and where correct and complete mathematical proofs have been written out for those facts, are relatively thin on the ground, and **[Old1]** seems the best choice for getting at least some details. This is very technical work, and ordinary mortals cannot become experts overnight! We refer the reader again to the quotes in the last subsection of this paper. The proof below follows that in **[C]**, filling in details, and correcting a couple of errors, as noted earlier.

For the case of the structural inductive adequacy proof where C is a call-command, and also for later purposes, we need the following lemmas.

Lemma 8.8 *Let C/π be any command in* **DTEN/***. Assume that*

$$\delta_1|_{FREE(C/\pi)} \;=\; \delta_2|_{FREE(C/\pi)} \qquad \text{where } FREE(C/\pi) \subset \mathrm{dom}(\delta_1) \cap \mathrm{dom}(\delta_2)\ ,$$

and that the minimum m greater than all i for which bin_i is in the image of δ_1 is the same as that for δ_2. Then

$$LOUT(C/\pi, s|\delta_1) \;=\; LOUT(C/\pi, s|\delta_2)\ .$$

(Recall that $LOUT$ is the left-half, the 's-half', of OUT.)

Proof. This is an induction 'on SSQ' (very similar to the one in the proof of **8.6**(a)) showing that the left-halves of SSQ_n agree for all n. The case $C = \mathsf{begin\ new}\ x; D_*; C_*\ \mathsf{end}$ is the only one where the 'answer' uses a different δ than the starter. In that case, the conditions above, relating the starters δ_1 and δ_2, insure that the same conditions relate the two δ's used in calculating SSQ inductively. So the induction goes through. (This lemma is used only once below, non-essentially, so we won't flesh this proof out.)

Definition. We say that "$\underline{s}|\underline{\delta}$ is $(\vec{x}\mapsto \vec{u}\ ,\ \vec{v}\mapsto \vec{t})$—matched to $s|\delta$", if and only if the following four conditions hold:

(A) $[y \in \mathrm{dom}(\underline{\delta})\cap\mathrm{dom}(\delta)$ and $y \notin$ "$\vec{x}$"$\cup$"$\vec{v}$"$\cup$"$\vec{u}$"$]$ $\implies$ $\underline{s}\circ\underline{\delta}(y) = s\circ\delta(y)]$;

(B) $$\forall k\ ,\quad \underline{s}\circ\underline{\delta}(x_k)\ =\ s\circ\delta(u_k)\ ;$$

(C) $$\forall j\ ,\quad \underline{s}\circ\underline{\delta}(v_j)\ =\ t_j^{s\circ\delta}\ ;$$

(D) $\underline{s}(\mathrm{bin}_{\underline{i}}) = s(\mathrm{bin}_i)$ for $\underline{i} - \underline{m} = i - m \geq 0$ where $\underline{m}$ and m are the smallest subscripts of bins larger than the subscripts of all bins in the images of $\underline{\delta}$, δ respectively.

NOTE: Including $\cdots\cup$ "$\vec{u}$ " here in (A) disagrees with **[C]**, but his analogue of **8.10** below is false.

Lemma 8.9 *Assume given a formula F [and/or a term t], as well as $\underline{s}, \underline{\delta}, s, \delta, \vec{x}, \vec{u}, \vec{v}, \vec{t}$ such that :*

(i) $\underline{s}|\underline{\delta}$ is $(\vec{x}\mapsto \vec{u}\ ,\ \vec{v}\mapsto \vec{t})$—matched to $s|\delta$;

(ii) $\underline{\delta}$ and δ are defined for all variables free in the formulas F and $F^{[\vec{x}\mapsto\vec{u}\ ,\ \vec{v}\mapsto\vec{t}]}$ [and/or all variables in the terms e and $e^{[\vec{x}\mapsto\vec{u}\ ,\ \vec{v}\mapsto\vec{t}]}$] ;

(iii) the variables in "$\vec{u}$" do not appear in F and/or e.

Then

$$F^{[\vec{x}\mapsto\vec{u}\ ;\ \vec{v}\mapsto\vec{t}]} \text{ is true at } s\circ\delta \quad\Longleftrightarrow\quad F \text{ is true at } \underline{s}\circ\underline{\delta}\ ;$$

[and/or

$$(e^{[\vec{x}\mapsto\vec{u}\ ;\ \vec{v}\mapsto\vec{t}]})^{s\circ\delta}\ =\ e^{\underline{s}\circ\underline{\delta}}\ ]\ .$$

NOTE: Requiring *(iii)* here (because of the $\cdots\cup$"$\vec{u}$ " in (A) of the definition) weakens this compared to the corresponding Lemma 2 in **[C]**. But all five applications of this lemma below do in fact work alright. And Lemma 3 in **[C]** turns out to actually be false because $\cdots\cup$ "$\vec{u}$ " wasn't required in **[C]**—see the example at the end of the six page proof of **8.10** just ahead.

Proof. This is basic $1^{\underline{\mathrm{st}}}$order logic. It uses (A), (B) and (C) (but not (D) of course). The point is simply that substitutions into formulas and terms give answers whose semantics, relative to that prior to substitution, vary only with respect to the variables involved in that substitution.

Lemma 8.10 *Let* $[K, \pi, (\vec{x} : \vec{v}), (\vec{u} : \vec{t})]$ *be such that :*
(i) the usual disjointness of the variables in $(\vec{x} : \vec{v})$ *and in* $(\vec{u} : \vec{t})$ *holds ;*
(ii) $K/\pi \in$ **CTEN/** *;*
(iii) for any $p \in PRIDE$*, every i.a.s. for* call $p(\vec{u} : \vec{t})/\pi$ *which begins as* occ $p(\vec{u} : \vec{t})//K(\vec{x} : \vec{v})//\cdots$ *satisfies the caveats* CV_2+, CV_3+ *and* CV_4+ *in the definition of* **CTEN**.

[This is more-or-less saying that K is a procedure which can be "legally" called within **CTEN/**. The specification of the second term in the i.a.s. amounts to requiring that $\pi(p) = (K, (\vec{x} : \vec{v}))$.]
Assume that $\underline{s}|\underline{\delta}$ *is* $(\vec{x} \leftarrow \vec{u}\ ,\ \vec{v} \leftarrow \vec{t})$—matched to $s|\delta$. *Then*

$$OUT(K/\pi\ ,\ \underline{s}|\underline{\delta}) \text{ is } (\vec{x} \leftarrow \vec{u}\ ,\ \vec{v} \leftarrow \vec{t}) - \text{matched to } \quad OUT(K^{[\vec{x} \leftarrow \vec{u}\ ;\ \vec{v} \leftarrow \vec{t}]}/\pi\ ,\ s|\delta).$$

Its three applications. It is readily checked that *(i),(ii)* and particularly *(iii)* hold for $[K, \pi, (\vec{x} : \vec{v}), (\vec{u} : \vec{t})]$ in the following three examples, (α), (β) and (γ). These are the three situations [**8.11**, validity of (IX),(X)] where the lemma will be applied below. Checking these three may also help to familiarize the reader with the technicalities. In **[C]**, the entire set of assumptions is stated "where K is any statement such that $p(\vec{x} : \vec{v})$ proc K could be a legal declaration for a legal statement call $p(\vec{u} : \vec{t})$". But I could not find a less technical way to formulate this so that both its proof and its applications inspired reasonable confidence.

(α) For some p, call $p(\vec{u} : \vec{t})/\pi \in$**CTEN/** and $\pi(p) = (K, (\vec{x} : \vec{v}))$.

(β) $[C, \pi, (\emptyset : \vec{y}), (\emptyset : \vec{r})]$ where $C/\pi \in$**CTEN/** and "$\vec{y}$ " $\cup$ "$\vec{r}$ " is disjoint from $FREE(C/\pi)$.

(γ) $[L, \pi, (\vec{y} : \vec{w}), (\vec{u} : \vec{t})]$ where $(\vec{y} : \vec{w}) \in IDE^{*!}$, "$\vec{y}$ " $\cup$ "$\vec{w}$ " is disjoint from "$\vec{u}$ " $\cup$ "$\vec{t}$ ", and $L = K^{[\ \vec{x} \leftarrow \vec{y}\ ;\ \vec{v} \leftarrow \vec{w}\ ]}$ with $[K, \pi, (\vec{x} : \vec{v}), (\vec{u} : \vec{t})]$ as in (A).

Notation. Very occasionally, we shall need to separate $OUT(K/\pi, s|\delta)$ into its two halves, which we'll then specify by putting an L for "left" and an R for "right" in front of the OUT. That is,

$$OUT(K/\pi, s|\delta) \ =: \ LOUT(K/\pi, s|\delta) \mid ROUT(K/\pi, s|\delta)\ .$$

Proof of 8.10. This is an induction on the cardinality of $SCOM(K/\pi)$, and then on K, which has several non-obvious cases (of the usual seven cases), including

the cases of variable-declaration commands and assignment commands, where the caveats CV singling out **CTEN** from **DTEN** play a crucial role, and worse, if anything, the case of a call-command, where the $CV+$ are crucial. We'll leave the case of assignment commands to the last, for reasons indicated in the small print just after the proof. Unfortunately, this is where we (unavoidably it seems) reach an orgasm of technicalities. So some readers may prefer to skip the 9-page proof for now. On the other hand, this is really where the reasons for the restriction from **DTEN** to **CTEN** become manifest, at least if one cannot think of alternative lines of argument (or *alternate* lines of argument, for USers).

To help express the various cases of the proof readably, let us denote the displayed OUTputs in the lemma as

$$OUT(K/\pi\ ,\ \underline{s}|\underline{\delta}) = \underline{\sigma}_K(\underline{s}|\underline{\delta})$$

and

$$OUT(K^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]}/\pi\ ,\ s|\delta) = \sigma_K(s|\delta)\ .$$

And let's suppress the $\vec{x}\to\ \vec{u}\ ;\ \vec{v}\to\ \vec{t}$ from the notation. And finally let's shorten "—matched" to just "–m–"

So we must prove

$$\underline{s}|\underline{\delta}\ \text{–m–}\ s|\delta \quad\Longrightarrow\quad \underline{\sigma}_K(\underline{s}|\underline{\delta})\text{–m–}\sigma_K(s|\delta)\ .$$

Note before starting that condition (iii) on K in the lemma is inherited by any subcommand, so it is only the penultimate of the seven cases below (the call-command) where we need to check that the inductive assumption actually applies to the relevant command.

Case when $K = \mathsf{begin\ end.}$

This is beyond the pale of triviality.

Case when $K = \mathsf{begin}\ C_1; C_*\ \mathsf{end}.$

Applying the structural induction twice, $\underline{s}|\underline{\delta}\ \text{–m–}\ s|\delta$ implies that $\underline{\sigma}_{C_1}(\underline{s}|\underline{\delta})\text{–m–}\sigma_{C_1}(s|\delta)$. But then the latter implies that

$$\underline{\sigma}_{\mathsf{begin}\ C_*\ \mathsf{end}}(\underline{\sigma}_{C_1}(\underline{s}|\underline{\delta}))\ \text{–m–}\ \sigma_{\mathsf{begin}\ C_*\ \mathsf{end}}(\sigma_{C_1}(s|\delta)).$$

However, the definition of SSQ for K in terms of C_1 and C_* immediately shows that the last statement is identical with the required one, namely

$$\underline{\sigma}_K(\underline{s}|\underline{\delta})\text{–m–}\sigma_K(s|\delta)\ .$$

Case when $K = \mathsf{whdo}\ (H)(C)$.

Modulo one crucial observation, this is really all the instances of the previous case of the form $\mathsf{begin}\ C; C; \cdots; C\ \mathsf{end}$. But we also must observe that, in the relevant cases (where the whdo-command terminates), the formulas H and $H^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]}$ become false after exactly the same number of iterations in both cases, by **8.9** and induction. This uses **8.9** applied to the formula H, and our extra condition *(iii)* there holds, because no u_i can occur free in K, by CV_3+, $\ell = s = 1$, so in particular, in H. (The double-underlined is first in our recorded list of the assumptions about the given data which are needed to push the proof through.)

Case when $K = \mathsf{begin}\ D; D_*; C_*\ \mathsf{end}$, with D a procedure declaration.

Let $K' = \mathsf{begin}\ D_*; C_*\ \mathsf{end}$. Recall, from the definition of SSQ for this case, that we modified π to π' to express the state sequence for K in terms of that for K'. In fact, π' agrees with π, except that it maps the procedure name in D to its body and formal parameters, which need no notation here. We shall use π and π' in the subscripts on the σ's in the obvious way. Then the definition of SSQ just referred to gives

$$\underline{\sigma}_{K/\pi}(\underline{s}|\underline{\delta}) = \underline{\sigma}_{K'/\pi'}(\underline{s}|\underline{\delta}) \quad \text{and} \quad \sigma_{K/\pi}(s|\delta) = \sigma_{K'/\pi'}(s|\delta) \quad .$$

But now the inductive hypothesis applied to K' gives exactly the required statement about K.

Case when $K = \mathsf{begin\ new}\ x; D_*; C_*\ \mathsf{end}$.

Let $K' = \mathsf{begin}\ D_*; C_*\ \mathsf{end}$. As in the definition of SSQ for K in terms of K', define $\underline{\delta}', \delta'$ to agree with $\underline{\delta}, \delta$, except that they map the variable x to $\mathrm{bin}_{\underline{m}}$, bin_m, respectively. And define $\underline{\delta}'', \delta''$ to agree with $\underline{\delta}, \delta$, except that they map variables $\underline{z}, z$ to $\mathrm{bin}_{\underline{m}}$, bin_m, respectively, where $\underline{z}, z$ are the least variable after the domains of $\underline{\delta}, \delta$ respectively.

If x occurs in $(\vec{x} : \vec{v})$, consider the simultaneous substitution in which the one for x is omitted, but which is otherwise identical to $\vec{x}\to\ \vec{u}\ ;\ \vec{v}\to\ \vec{t}$. Use notation "–m*–" for the matching corresponding to that (possibly restricted) substitution. Direct from the definition of freeness, x is not free in K, so these two (possibly different) substitutions have the same effect on K

Now, direct from the definition of matching, that $\underline{s}|\underline{\delta}\ -\mathrm{m}-s|\delta$ gives that

$$\underline{s}|\underline{\delta}'\ -\mathrm{m}-s|\delta' \ .$$

The equations for the latter are identical with the equations for the former, except for the following : the equation $\underline{s}(\mathrm{bin}_{\underline{m}}) = s(\mathrm{bin}_m)$ from part (D) of the former isn't part of (D) for the latter, but is used rather for one new equation needed to verify part (A) or (B) or (C) of the latter, depending respectively on whether the variable x is not in "$\vec{x}$ " $\cup$ "$\vec{v}$ " $\cup$ "$\vec{u}$ ", or is in "$\vec{x}$ ", or is in "$\vec{v}$ ". (If it's in "$\vec{u}$ ", there is no extra equation to worry about.)

Then, applying the inductive hypothesis to K' and using the display immediately above,

$$\underline{\sigma}_{K'}(\underline{s}|\underline{\delta}') \text{ –m}^*\text{– } \sigma_{K'}(s|\delta').$$

Now the definition of SSQ for K in terms of K' shows the displayed claim just above to be the same as

$$(*) \qquad LOUT(K/\pi, \underline{s}|\underline{\delta}) \mid \underline{\delta}^* \text{ –m}^*\text{– } LOUT(K^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]}, s|\delta) \mid \delta^*,$$

for $\underline{\delta}^*, \delta^*$ defined by the right-hand sides of the first and third displays just below.

The claim about $(*)$ holds because

$$\underline{\sigma}_{K'}(\underline{s}|\underline{\delta}') = OUT(K'/\pi, \underline{s}|\underline{\delta}') \ = \ (\text{say})\ LOUT(K/\pi, \underline{s}|\underline{\delta})|\underline{\delta}^* \ ,$$

where

$$LOUT(K/\pi, \underline{s}|\underline{\delta})|\underline{\delta}'' = OUT(K/\pi, \underline{s}|\underline{\delta}) = \underline{\sigma}_K(\underline{s}|\underline{\delta}) \ ;$$

and because

$$\sigma_{K'}(s|\delta') = OUT(K'^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]}/\pi, s|\delta') \ = \ (\text{say})\ LOUT(K^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]}/\pi, s|\delta) \mid \delta^* \ ,$$

where

$$LOUT(K^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]}/\pi, s|\delta)|\delta'' = OUT(K^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]}/\pi, s|\delta) = \sigma_K(s|\delta) \ .$$

The middle equality in the last display depends on the fact that

$$K'^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]} = (K^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]})' \ .$$

By the second and fourth displays above, proving the required

$$(**) \qquad \underline{\sigma}_K(\underline{s}|\underline{\delta}) \text{ –m– } \sigma_K(s|\delta)$$

amounts to changing, in $(*)$, the $\underline{\delta}^*, \delta^*$ in $(**)$ to $\underline{\delta}'', \delta''$, and getting rid of the "*" on the matching. This where we get mildly painstaking. The crucial

observation in each case is that the execution of K using $\underline{\delta}$, because it starts with declaring "new x", has no effect on the contents of the bin $\underline{\delta}(x)$; that is, all the states in its SSQ-sequence map $\underline{\delta}(x)$ to the same value.

Here is a formal version of the general result needed for this observation, which is intuitively obvious, and can be proved very easily for the state SSQ_n by induction on n, then by structural induction :

8.6(c) *For any $(C/\pi, s|\delta)$ whose computation sequence exists, if*

$$SSQ(C/\pi, s|\delta) =\prec s_1|\delta_1\ ,\ s_2|\delta_2\ ,\ \cdots \succ\ ,$$

then, for all but finitely many bins b, we have $s_n(b) = s(b)$ for all n. In particular, this can fail to hold only for $b = \delta(z)$ for variables z for which some assignment command $z \leftarrow\!\!\div\ e$ is in $SCOM(C/\pi)$. In particular, this unaffectedness of $s \circ \delta(z)$ by execution of C/π holds for all variables z not in $FREE(C/\pi)$

$$[\ \text{since}\quad z \leftarrow\!\!\div\ e\ \in SCOM(C/\pi)\ \implies\ z \in FREE(C/\pi)\ ]\ .$$

Recall that $SCOM(K/\pi)$ is the set of subcommands, defined earlier, and it includes subcommands in all procedures that might get called and executed. In each case below, we are using $\underline{\sigma}$ [and σ respectively] as short for the left half of $\underline{\sigma}_K(\underline{s}|\underline{\delta})$ [and $\sigma_K(s|\delta)$ resp.]

Case (a) : $x \notin$ "$\vec{x}$ " $\cup$ "$\vec{v}$ " $\cup$ "$\vec{u}$ ".

Here the two substitutions are actually the same. With one switch, most equations for $(**)$ are the same as those for $(*)$. For (D) in $(**)$, we need the bottom case

$$\sigma(\text{bin}_m) = \sigma \circ \delta'(x) = \underline{\sigma} \circ \underline{\delta}'(x) = \underline{\sigma}(\text{bin}_{\underline{m}})\ \ ,$$

the middle equality being part of (A) for $(*)$. Also there is the additional equation in (A) for $(**)$, namely

$$\underline{\sigma} \circ \underline{\delta}(x) = \underline{s} \circ \underline{\delta}(x) = s \circ \delta(x) = \sigma \circ \delta(x)\ \ .$$

The two outside equalities are the principle (concerning the computation not affecting most bins) discussed just above; that is, use **8.6**(c). The middle one is the basic matching hypothesis in this lemma.

Case (b) : $x \in$ "$\vec{x}$ ".

With two exceptions (the displays below), the equations for $(**)$ are the same as those for $(*)$. For (D) we need the bottom case

$$\sigma(\mathrm{bin}_m) = \sigma \circ \delta'(x) = \underline{\sigma} \circ \underline{\delta}'(x) = \underline{\sigma}(\mathrm{bin}_{\underline{m}}) \quad ,$$

the middle equality being part of (A) for $(*)$. And, for (B), if $x_k = x$, we get

$$\underline{\sigma} \circ \underline{\delta}(x_k) = \sigma \circ \delta(u_k)$$

because the same holds with $s, \underline{s}$ in place of $\sigma, \underline{\sigma}$, and because the bins $\underline{\delta}(x_k)$ and $\delta(u_k)$ are unaffected by the execution of K and $K^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]}$ respectively. Note that u_k is not free in the latter command because it's not free in K, it doesn't get substituted for x because x is not free in K, and it's different from all the other u_i's and the variables in the t_j's (even if we were in **DTEN**).

Case (c) : $x \in$ "$\vec{v}$ ".

With two exceptions (the displays below), the equations for $(**)$ are the same as those for $(*)$. For (D) we need the bottom case

$$\sigma(\mathrm{bin}_m) = \sigma \circ \delta'(x) = \underline{\sigma} \circ \underline{\delta}'(x) = \underline{\sigma}(\mathrm{bin}_{\underline{m}}) \quad ,$$

the middle equality being part of (A) for $(*)$. And, for (C), if $v_j = x$, we get

$$\underline{\sigma} \circ \underline{\delta}(v_j) = t_j^{\sigma\circ\delta} \ ,$$

as required, because the same holds with $s, \underline{s}$ in place of $\sigma, \underline{\sigma}$, and because the bins $\delta(z)$ for all variables z in t_j (and $\underline{\delta}(v_j)$ respectively) are unaffected by the execution of $K^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]}$ (and K respectively). The argument for this last claim is as follows : These variables z *can* be free in $K^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]}$, but, because v_j is not free in K, so t_j doesn't get substituted for it, this can only happen with another v_i being replaced by some t_i, for $i \neq j$, where t_i and t_j have the variable z in common. But then no $z \mathrel{\div\!\!\leftarrow} e$ is in $SCOM(K^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]})$ because v_i cannot occur to the left of a " $\mathrel{\div\!\!\leftarrow}$ " in K, by CV_2, and so $\delta(z)$ is unaffected by executing $K^{[\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}]}$, as required.

Case (d) : $x \in$ "$\vec{u}$ ".

With one exception (the same old one!), the equations for $(**)$ are the same as those for $(*)$. For (D) we need the bottom case

$$\sigma(\mathrm{bin}_m) = \sigma \circ \delta'(x) = \underline{\sigma} \circ \underline{\delta}'(x) = \underline{\sigma}(\mathrm{bin}_{\underline{m}}) \quad ,$$

the middle equality being part of (A) for $(*)$.

This completes the case of a command which is a 'variable-declaration block'.

Case when $K = \mathsf{call}\ p'(\vec{u}' : \vec{t}')$.

If p' is not in the domain of π, the displayed OUT-states in the theorem do not exist and there's nothing to prove.

Otherwise, let $\pi(p') = (L, (\vec{x}' : \vec{v}'))$. Now, assuming that the inductive hypothesis applies to $L_1 := L^{[\vec{x}' \to \vec{u}' \ , \ \vec{v}' \to \vec{t}']}$, the proof is completed in this case as follows. We have, for some N, using the definition of SSQ for calls,

$$\underline{\sigma}_K(\underline{s}|\underline{\delta}) \ := \ OUT(K/\pi, \underline{s}|\underline{\delta}) \ = \ SSQ_N(K/\pi, \underline{s}|\underline{\delta}) \ =$$

$$SSQ_{N-1}(L_1/\pi, \underline{s}|\underline{\delta}) \ =: \ \underline{\sigma}_{L_1}(\underline{s}|\underline{\delta}) \ .$$

Define

$$u''_j := u'^{[\vec{x} \to \vec{u} \ , \ \vec{v} \to \vec{t}]}_j \quad \text{and} \quad t''_i := t'^{[\vec{x} \to \vec{u} \ , \ \vec{v} \to \vec{t}]}_i \quad .$$

Because "$\vec{x}$ " $\cup$ "$\vec{v}$ " is disjoint from $FREE(L/\pi)$ by CV_4+ , a direct elementary argument shows that

$$L^{[\vec{x}' \to \vec{u}'' \ , \ \vec{v}' \to \vec{t}'']} \ = \ L_1^{[\vec{x} \to \vec{u} \ , \ \vec{v} \to \vec{t}]} \ .$$

Thus

$$\sigma_K(s|\delta) \ := \ OUT(K^{[\vec{x} \to \vec{u} \ , \ \vec{v} \to \vec{t}]}/\pi, s|\delta) \ = \ OUT(\mathsf{call}\ p'(\vec{u}'' : \vec{t}'')\ /\pi, s|\delta) \ =$$

$$OUT(L^{[\vec{x}' \to \vec{u}'' \ , \ \vec{v}' \to \vec{t}'']}\ /\pi, s|\delta) \ = \ OUT(L_1^{[\vec{x} \to \vec{u} \ , \ \vec{v} \to \vec{t}]}\ /\pi, s|\delta) \ =: \ \sigma_{L_1}(s|\delta) \ .$$

So we have reduced the matching question from the required K to the command L_1.

The set $SCOM(L_1/\pi)$ is a proper subset of $SCOM(K/\pi)$, with cardinality one less. So the inductive hypothesis does indeed apply to L_1, completing the proof here, once we have argued that $[K, \pi, (\vec{x} : \vec{v}), (\vec{u} : \vec{t})]$ satisfying

(iii) implies that $[L_1, \pi, (\vec{x} : \vec{v}), (\vec{u} : \vec{t})]$ also satisfies *(iii)* (in the statement of **8.10**).

To do that, consider an i.a.s. for call $p(\vec{u} : \vec{t})/\pi$ as follows:

$$\mathsf{occ}\ p(\vec{u} : \vec{t})/\ /\ L_1(\vec{x} : \vec{v})/\ /\ \mathsf{occ}\ p_2(\vec{u}^{(2)} : \vec{t}^{(2)})/\ /\ \cdots$$

The third term is an occurrence of a call in $L_1 = L^{[\vec{x}' \to \vec{u}'\ ,\ \vec{v}' \to \vec{t}']}$. The corresponding call occurrence in L cannot involve the variables in "$\vec{x}'$ " $\cup$ "$\vec{v}'$ " because of CV_4+ for K, applied to i.a.s.'s which have K in their 2nd term and L in their 4th term. Thus the third term in the display is identical with its corresponding call occurrence in L. So all the conditions with $s > 1$ automatically hold in the case of L_1 because they hold for the above i.a.s.'s which have K in their 2nd term and L in their 4th term.

Thus what remains to check are the conditions on $[L_1, \pi, (\vec{x} : \vec{v}), (\vec{u} : \vec{t})]$ which are CV_2+ and CV_3+ for $s = \ell = 1$. For the latter, the set "$\vec{u}$ " $\cup$ "$\vec{t}$ " is disjoint from $FREE(L_1/\emptyset)$ because the same holds for L. For the former, no v_i occurs left of a colon in L_1 again because the same holds for L.

This completes the case of a call-command.

Case when K is the assignment command $z \div\!\!= e$.

Here there is no induction to rest on, and one must simply go back to the definition of matching and check the equations carefully.

Firstly, our assignment command, $z \div\!\!= e$, changes s only on the bin $\delta(z)$ (if it exists), so the bins not in the images of δ and $\underline{\delta}$ are unaffected. Thus condition (D) in the definition clearly holds for the output states.

Let $e_1 := e^{[\vec{x} \to \vec{u}\ ,\ \vec{v} \to \vec{t}]}$. Then, by **8.9**, we have $e_1^{s \circ \delta} = e^{\underline{s} \circ \underline{\delta}}$. For this, note that $u_i \notin$ "e" for all i, since those variables cannot be free in K by CV_3, so the new condition (iii) in **8.9** holds.

Also, letting

$$z_1 \ := \ \begin{cases} u_i & \text{if } z = x_i\ ; \\ z & \text{if } z \notin \text{"}\vec{x}\text{"}; \end{cases}$$

we get

$$(z \div\!\!= e)^{[\vec{x} \to \vec{u}\ ,\ \vec{v} \to \vec{t}]} \ = \ z_1 \div\!\!= e_1\ .$$

(By CV_2, the v_i cannot appear left of the colon in an assignment command in K, so the case $z = v_i$ does not arise above.)

Here we will again use shorter notation for the left-hand side of state-pairs, viz.

$$\underline{\sigma}_{z \leftarrow e}(\underline{s}|\underline{\delta}) = \underline{\sigma}|\underline{\delta} \quad \text{and} \quad \sigma_{z \leftarrow e}(s|\delta) = \sigma|\delta$$

defines $\underline{\sigma}$ and σ.

Thus

$$\sigma := LOUT(z_1 \leftarrow\!\!\div\, e_1/\pi, s|\delta) = \begin{cases} \delta(u_i) \mapsto e_1^{s\circ\delta} = e^{\underline{s}\circ\underline{\delta}}\,, & \text{and} \\ \text{other bin}_q \mapsto s(\text{bin}_q) & \text{if } z = x_i; \\ \delta(z) \mapsto e_1^{s\circ\delta} = e^{\underline{s}\circ\underline{\delta}}\,, & \text{and} \\ \text{other bin}_q \mapsto s(\text{bin}_q) & \text{if } z \notin \text{“}\vec{x}\text{ ”}; \end{cases}$$

whereas

$$\underline{\sigma} := LOUT(z \leftarrow\!\!\div\, e/\pi, \underline{s}|\underline{\delta}) = \begin{cases} \underline{\delta}(z) \mapsto e^{\underline{s}\circ\underline{\delta}}\,, & \text{and} \\ \text{other bin}_q \mapsto \underline{s}(\text{bin}_q) & . \end{cases}$$

To check condition (C) for σ and $\underline{\sigma}$ in the “matching”-definition, we have $\underline{\sigma}\circ\underline{\delta}(v_j) = \underline{s}\circ\underline{\delta}(v_j) = t_j^{s\circ\delta} = t_j^{\sigma\circ\delta}$ as required, for the following reasons :

The first equality is immediate from the calculation just above of $\underline{\sigma}$, since $z = v_j$ cannot occur because none from $\vec{v}$ can appear to the left of the “:” in an assignment command by CV_2, so $v_j \neq z$.

The second equality is condition (C) for the “matching” assumed in this lemma.

The final equality is clear as long as $\sigma \circ \delta(y) = s \circ \delta(y)$ for all variables y in the term t_j. But that is clear from the calculation just above of σ, since no variable in t_j can be a u_i, or be z by CV_3.

To check condition (B), immediately from the basic calculations of $\underline{\sigma}$ and σ in the above large displays, we get

$$\sigma \circ \delta(u_k) = \begin{cases} e^{\underline{s}\circ\underline{\delta}}\,, & \text{if } z \notin \text{“}\vec{x}\text{ ”} \text{ and } u_k = z\,; \\ s \circ \delta(u_k) & \text{if } z \notin \text{“}\vec{x}\text{ ”} \text{ and } u_k \neq z\,; \\ e^{\underline{s}\circ\underline{\delta}}\,, & \text{if } z = x_k\,; \\ s \circ \delta(u_k) & \text{if } z = x_i \text{ and } i \neq k\,; \end{cases}$$

whereas

$$\underline{\sigma} \circ \underline{\delta}(x_k) = \begin{cases} e^{\underline{s}\circ\underline{\delta}}\,, & \text{if } z = x_k\,; \\ \underline{s} \circ \underline{\delta}(x_k) & \text{if } z \neq x_k\,. \end{cases}$$

But the very top of the six right-hand sides simply doesn't occur here, because of CV_3, viz. no variable in $\vec{u}$ is free in K. It is immediate that the two left-hand sides agree, as required, from condition (B) for the "matching" assumed in this lemma.

Finally, to check condition (A), assume that $y \notin$ "$\vec{x} \cup \vec{v} \cup \vec{u}$ ". Immediately from the early calculations of $\underline{\sigma}$ and σ, we find

$$\sigma \circ \delta(y) = \begin{cases} e^{\underline{s} \circ \underline{\delta}} & \text{if } z = x_i \text{ and } u_i = y \ ; \\ s \circ \delta(y) & \text{if } z = x_i \text{ and } u_i \neq y, \text{ so } y \neq z \text{ since } y \notin \text{“}\vec{x}\text{”} \ ; \\ e^{\underline{s} \circ \underline{\delta}} \ , & \text{if } z \notin \text{“}\vec{x}\text{”} \text{ and } y = z \ ; \\ s \circ \delta(y) & \text{if } z \notin \text{“}\vec{x}\text{”} \text{ and } y \neq z \ ; \end{cases}$$

whereas

$$\underline{\sigma} \circ \underline{\delta}(y) = \begin{cases} e^{\underline{s} \circ \underline{\delta}} \ , & \text{if } y = z \ ; \\ \underline{s} \circ \underline{\delta}(y) & \text{if } y \neq z \ . \end{cases}$$

If the first of the six right-hand sides cannot happen, it is immediate that the two left-hand sides agree, as required, using condition (A) for the "matching" assumed in this lemma. But the top right-hand side cannot happen because we picked a variable y with $y \notin$ "$\vec{u}$ ".

This finally completes the proof of **8.10**. The condition that $y \notin$ "$\vec{u}$ " in the final line of that proof is essential, as the following example shows. This is a counterexample to Lemma 3 in [**C**], where the definition of matching does not include that condition in its part (A). (Fortunately) the rest of Cook's proof goes through (because I don't know any different proof!) with the weaker definition of matching and the consequential weaker version of his Lemma 2, our **8.9**. Note that both **8.9** and **8.10** are used crucially several times in the proof of adequacy *and* in the proof of soundness.

For the example, take the variables to be $w_1, w_2, \cdots$; take $\underline{\delta}$ and δ both to be defined on w_1, w_2, w_3 only, mapping w_i to bin_i, except that $\underline{\delta}$ maps w_1 to bin_2 and w_2 to bin_1; take $\underline{s} = s$ to be the state given by $0, 0, 1, 0, 0, 0, \cdots$; take K to be the assignment command $w_1 \Leftarrow w_3$; and finally take $\vec{v}$ and $\vec{t}$ to be empty, and $\vec{x} = (w_1)$ and $\vec{u} = (w_2)$, i.e. $u_1 = w_2$.

This satisfies the hypotheses of **8.10** even if we don't required $y \notin$ "$\vec{u}$ " in (A) of the definition of matching, i.e. if we'd used Cook's definition. Our definition requires $\underline{\sigma} \circ \underline{\delta}(y) = \sigma \circ \delta(y)$ only for $y = w_3$, whereas the stronger version of (A) requires it also for $y = w_2$. But both hold, w_2 giving the value 0, and w_3 the value 1.

A quick calculation gives $\underline{\sigma} = \sigma$ to be the state given by $0, 1, 1, 0, 0, 0, \cdots$. The substitution changes the command to $w_2 \Leftarrow w_3$. Then the matching conclusion of **8.10** of course holds for our 'matching definition'. But it fails for the other, because

$$0 = \underline{\sigma} \circ \underline{\delta}(w_2) \neq \sigma \circ \delta(w_2) = 1 \ .$$

Continuation of the proof of 8.7.
The final case for adequacy is for call commands.
Assume that $F\{\mathsf{call}\ p(\vec{u} : \vec{t})/\pi\}G$ is true, and we'll find a derivation.

The asinine subcase is that when p is not in the domain of π. (In that case, F and G could be any formulas.) Extend π to π', which agrees with π, and has only p added to its domain, with $\pi'(p) = B$, where B is some command which fails to terminate no matter what the input. By (XI), it suffices to find a derivation for $F\{\mathsf{call}\ p(\vec{u} : \vec{t})/\pi\}G$. By (VIII), it suffices to find a derivation for $F\{B/\pi'\}G$. The analogue was done just before the proof of **8.4** when we were in the child's world of the language **ATEN**. The solution here does require going through all seven cases in the structural inductive definition of commands in **DTEN**. But it is essentially elementary and will be left to the reader as an exercise. (Note that π might just as well be empty here.)

In the non-asinine subcase, let $\pi(p) = (K, (\vec{x} : \vec{v}))$. Let $\vec{z}$ be a list of all variables in $\vec{t}$. Let $\vec{x'}, \vec{v'}, \vec{x''}, \vec{v''}, \vec{z'}$ be a string of distinct variables, completely disjoint from any variables occurring so far in this case, and any variables in $FREE(\mathsf{call}\ p(\vec{u} : \vec{t})/\pi)$. (The individual string lengths can be guessed from the letters used, or deduced from the various substitutions below.)

Let $\vec{t'} := \vec{t}^{\,[\vec{z} \to \vec{z'}]}$, and define

$$F_1 \ := \ \vec{v} \approx \vec{t'} \ \wedge \ \left(F^{[(\vec{x},\vec{v}) \to (\vec{x''},\vec{v''})]}\right)^{[\vec{u} \to \vec{x}\ ,\ \vec{z} \to \vec{z'}]} .$$

The notation used should be fairly obvious. In particular, the 'vector' equality formula before the conjunction is really a *multiple conjunction* of equalities of variables with terms. Get G_1 from G by the same pair of two successive simultaneous substitutions that were applied to F in the display just above :

$$G_1 \ := \ \left(G^{[(\vec{x},\vec{v}) \to (\vec{x''},\vec{v''})]}\right)^{[\vec{u} \to \vec{x}\ ,\ \vec{z} \to \vec{z'}]} .$$

Lemma 8.11 *The F-H statement $F_1\{K/\pi'\}G_1$ is true, where π' agrees with π, except that p is not in the domain of π'.*

Delaying the proof, it now follows by the inductive hypothesis pertaining to π' being 'smaller' than π that the F-H statement in the lemma is derivable.

Immediately from rule (VIII), the statement

$$F_1\{\mathsf{call}\ p(\vec{x} : \vec{v})\ /\ \pi\}G_1$$

is derivable.

Now, in rule (X), take $\vec{y}, \vec{w}, \vec{u}, \vec{t}$ to be $\vec{x}, \vec{v}, \vec{x'}, \vec{v'}$ respectively. (So we are temporarily changing $\vec{u}, \vec{t}$ from its fixed meaning in this case.) Certainly no x'_i even occurs in F_1, so the rule applies. Looking at the definition of F_1, (and noting that $[\vec{u} \overset{(i)}{\longmapsto} \vec{x} \overset{(ii)}{\longmapsto} \vec{x'}$ and $\vec{v} \overset{(ii)}{\longmapsto} \vec{v'}$, where (ii) is the substitution in the lower line of rule (X), and (i) is in the superscript in the definition of F_1), we get the statement as follows to be derivable:

$$(\vec{v'} \approx \vec{t'} \wedge (F^{[(\vec{x},\vec{v})\mapsto(\vec{x''},\vec{v''})]})^{[\vec{u}\mapsto\vec{x'}\ ,\ \vec{z}\mapsto\vec{z'}]})\ \{\mathsf{call}\ p(\vec{x'} : \vec{v'})/\pi\}\ (G^{[(\vec{x},\vec{v})\mapsto(\vec{x''},\vec{v''})]})^{[\vec{u}\mapsto\vec{x'}\ ,\ \vec{z}\mapsto\vec{z'}]}\ .$$

Now easily apply rule (IX) with $\vec{y} = \vec{z'}$, $\vec{r} = \vec{z}$ and $C = \mathsf{call}\ p(\vec{x'} : \vec{v'})$ to get the same thing as above, except that the "$\vec{t'}$" becomes "$\vec{t}$" on the far left, and the ", $\vec{z}\mapsto \vec{z'}$ " are erased on the far right of the superscripts for both F and G.

Again, in rule (X), take $\vec{y}, \vec{w}, \vec{u}, \vec{t}$ to be $\vec{x'}, \vec{v'}, \vec{u}, \vec{t}$ respectively, (where we are now back to the fixed meaning of $\vec{u}, \vec{t}$). Certainly, no u_i occurs free in the pre- and post- conditions of the F-H statement because of the $\vec{u}\mapsto \vec{x'}$. So (X) does apply, and the substitutions in its lower line are $\vec{x'}\mapsto \vec{u}$, $\vec{v'}\mapsto \vec{t}$. So we get the statement as follows to be derivable:

$$(\vec{t} \approx \vec{t}\ \ \wedge\ \ F^{[(\vec{x},\vec{v})\mapsto(\vec{x''},\vec{v''})]})\ \{\mathsf{call}\ p(\vec{u} : \vec{t}\ )/\pi\}\ G^{[(\vec{x},\vec{v})\mapsto(\vec{x''},\vec{v''})]}\ .$$

Now we can just erase the "$\vec{t} \approx \vec{t}\ \wedge$" at the front, by using rule (IV) with the upper line

$$F_2\ \rightarrow\ \vec{t} \approx \vec{t}\ \wedge F_2\quad ,\quad (\vec{t} \approx \vec{t}\ \wedge F_2)\{C/\pi\}G_2\quad ,\quad G_2 \rightarrow G_2\quad ,$$

where the middle of the display is the F-H statement in the display second above.

Finally, in rule (IX), take $\vec{y} = (\vec{x''}, \vec{v''})$, $\vec{r} = (\vec{x}, \vec{v})$ and $C = \mathsf{call}\ p(\vec{u} : \vec{t})$ to obtain, as required, the derivability of $F\{\mathsf{call}\ p(\vec{u} : \vec{t}\ )/\pi\}G$. The rule is applicable, since variables in $(\vec{x''}, \vec{v''})$ are not in $FREE(\mathsf{call}\ p(\vec{u} : \vec{t}\ ))$ by the choice of those new variables disjoint from earlier ones, and those in $(\vec{x}, \vec{v})$ also do not appear in $FREE(\mathsf{call}\ p(\vec{u} : \vec{t}\ )/\pi)$ by its definition and our caveat CV_4+ .

This completes the proof of the adequacy half of **8.7**, modulo proving **8.11**, which we now proceed to do, before going on to proving the soundness half of **8.7**. Notice that the only two uses of rule (IX) above had $\vec{r}$ as a list of variables both times and also had the command as a call-command both times. So, at least to that extent, the proof system could be pared to a minimum by restricting (IX) to that situation.

Proof of Lemma 8.11.

By the definition of SSQ, the truth of $F\{\mathsf{call}\ p(\vec{u} : \vec{t}\,)/\pi\}G$ (assumed for the final case of adequacy) gives the truth of $F\{K^{[\vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}]}\ /\ \pi'\}G$. That in turn gives the truth of

$$F^{[(\vec{x},\vec{v})\to(\vec{x''},\vec{v''})]}\{K^{[\vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}]}\ /\ \pi'\}G^{[(\vec{x},\vec{v})\to(\vec{x''},\vec{v''})]} \qquad (*)$$

because none of the variables involved in the formula substitutions occur in the command.

Now assume F_1 to be true at $s_1 \circ \delta_1$, where δ_1 is defined at least on all free variables in F_1 and G_1 and K. Also assume that $SSQ(K/\pi', s_1|\delta_1)$ is a finite sequence. We must show that G_1 is true at $OUT(K/\pi', s_1|\delta_1)$.

Pick an $s|\delta$ so that the following four conditions hold:

$$s \circ \delta(y) \ = \ s_1 \circ \delta_1(y) \ \text{ for all } y \notin \vec{x} \cup \vec{v} \text{ such that } \delta_1(y) \text{ exists ;}$$

$$s \circ \delta(u_i) \ = \ s_1 \circ \delta_1(x_i) \ ;$$

$$s \circ \delta(z) \ = \ s_1 \circ \delta_1(z') \ ;$$

$$s_1(\mathrm{bin}_{i_1}) = s(\mathrm{bin}_i) \text{ for } i_1 - m_1 = i - m \geq 0 \text{ where } \ m_1 \text{ and } m$$

are the smallest subscripts of bins larger than the subscripts of all bins in the images of δ_1 , δ respectively.

The pair $s_1|\delta_1$ is then $(\vec{x}\to\vec{u}\ ;\ \vec{v}\to\vec{t}\,)$—matched to $s|\delta$. Checking condition(C) in the definition of "—matched" also uses the "$\vec{v} \approx \vec{t}$-part" of F_1 .

Because of the definition of (s, δ) on the variables in $F^{[(\vec{x},\vec{v})\to(\vec{x''},\vec{v''})]}$, that formula is true at $s \circ \delta$ because (the right-hand half of) F_1 is true at $s_1 \circ \delta_1$.

Combining this with $(*)$, we see that $G^{[(\vec{x},\vec{v})\to(\vec{x''},\vec{v''})]}$ is true at

$$OUT(K^{[\vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}]}/\pi', s|\delta) \ = \ OUT(K^{[\vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}]}/\pi, s|\delta) \ .$$

But, by **8.10**, the OUT in this display is $(\vec{x}\!\!\rightarrow \vec{u}\ ;\ \vec{v}\!\!\rightarrow \vec{t})$—matched to

$$OUT(K/\pi', s_1|\delta_1) \ := \ OUT(K/\pi, s_1|\delta_1)\ ,$$

Changing from π' to π in both OUT-displays above follows from **8.6**d) just below, since no calls to p can occur in the commands, by CV_1.

Combining these facts from the OUT-displays with **8.9**, we find that G_1 is true at $OUT(K/\pi', s_1|\delta_1)$, as required.

8.6(d) *Let* $C \in$**DTEN** *and let* π', π *and* p *be such that*
(i) $p \notin \mathrm{dom}(\pi')$ *;*
(ii) $\mathrm{dom}(\pi) = \{p\} \cup \mathrm{dom}(\pi')$ *and* π, π' *agree except that* $\pi(p)$ *is defined ;*
(iii) *The procedure identifier* p *is not in any indirect activation sequence arising from* C/π *.*
Then

$$SSQ(C/\pi, s|\delta) \ = \ SSQ(C/\pi', s|\delta) \quad \text{for any } s \text{ and } \delta\ .$$

The proof is a straightforward induction.

Completion of the proof 8.7 (i.e. proof of the soundness half).
The validity of rules (I) to (IV) is very straightforward, with proof similar to the arguments given in tedious detail in **8.1** for **ATEN**, so these four will be omitted here.

Rule (V) is beyond the pale of triviality.

Validity of rule (VI) :

Let $C = \mathsf{begin}\ D_*; C_*\ \mathsf{end}$ and $C_+ = \mathsf{begin\ new}\ x; D_*; C_*\ \mathsf{end}$. Assume the truth of the upper line in (VI), i.e.

$$F^{[x\rightarrow y]}\ \{C/\pi\}\ G^{[x\rightarrow y]} \qquad \text{is true} \qquad\qquad (*)$$

To show the truth of the lower line, i.e. $F\ \{C_+/\pi\}\ G$, assume that F is true at $s \circ \delta$, and that $SSQ(C_+/\pi, s|\delta)$ is a finite sequence.

We must prove that G is true at $OUT(C_+/\pi, s|\delta)$.

This comes from

F true at $s \circ \delta \Rightarrow F^{[x \to y]}$ true at $s \circ \delta^* \Rightarrow G^{[x \to y]}$ true at $OUT(C/\pi, s|\delta^*)$

$$\Rightarrow G \text{ true at } LOUT(C/\pi, s|\delta^*) \mid \delta'' \Rightarrow G \text{ true at } OUT(C_+/\pi, s|\delta) \,,$$

where we define δ' and δ'' as in the definition of SSQ for variable-declaration-commands, and define $\delta^* = \delta'$ except that $\delta^*(y) = \delta(x)$.

The first implication is because δ and δ^* agree, except that $\delta^*(y) = \delta(x)$ and the values $\delta^*(x)$ and $\delta(y)$ (if defined) are irrelevant, because y is not free in F, and x is not free in $F^{[x \to y]}$.

The second implication is immediate from $(*)$.

The third implication follows because δ'' and $ROUT(C/\pi, s|\delta^*)$ agree, except the first maps x to what the second maps y to, and their values on y and x respectively are irrelevant.

The fourth implication follows from

$$OUT(C_+/\pi, s|\delta) \;=\; LOUT(C/\pi, s|\delta') \mid \delta'' \;=\; LOUT(C/\pi, s|\delta^*) \mid \delta'' \,,$$

where the second equality holds because δ^* and δ' agree except on y, but y is not in $FREE(C/\pi)$. (This may be proved directly in this simple case, but it is also the one application for **8.8**.) The first equality is immediate from the definition of $SSQ(C_+/\pi, s|\delta)$ in terms of $SSQ(C/\pi, s|\delta')$.

Validity of rule (VII) :

Let $C = \mathsf{begin}\ D_*; C_*\ \mathsf{end}$ and $C_+ = \mathsf{begin}\ D; D_*; C_*\ \mathsf{end}$, where $D = \mathsf{proc}\ p(\vec{x} : \vec{v}) \equiv K\ \mathsf{corp}$. Assume the truth of the upper line in (VII), i.e. $F\ \{C/\pi'\}\ G$ is true . To show, as required, that $F\ \{C_+/\pi\}\ G$ is true , where $\pi = \pi'$ except that $\pi'(p) = (K, (\vec{x} : \vec{v}))$, suppose that F is true at $s \circ \delta$, and that $SSQ(C_+/\pi, s|\delta)$ is a finite sequence. We must check that G is true at $OUT(C_+/\pi, s|\delta)$ But it is true at $OUT(C/\pi', s|\delta)$, so we only need the equality of these "OUTs". And that follows because the sequences $SSQ(C_+/\pi, s|\delta)$ and $SSQ(C/\pi', s|\delta)$, by definiton of SSQ, only differ by a spurious initial term.

Validity of rule (VIII) :

Let $C_+ = \mathsf{call}\ p(\vec{x} : \vec{v})$ and suppose that $F\{C/\pi'\}G$, the upper line in this rule, is true, where $\pi(p) := (C, (\vec{x} : \vec{v}))$ adds p to the domain to give π with a strictly larger domain than π'. To show that $F\{C_+/\pi\}G$, the lower line, is true, suppose that F is true at $s \circ \delta$, and that $SSQ(C_+/\pi, s|\delta)$ is a finite sequence.

To show, as required, that G is true at $(OUT(C_+/\pi, s|\delta)$, note that G is true at $OUT(C/\pi', s|\delta)$, which exists because of the display below. We claim

$$SSQ(C_+/\pi, s|\delta) \ = \ \prec\ s\ ,\ SSQ(C^{[\ \vec{x}\to\vec{x}\ ,\ \vec{v}\to\vec{v}\ ]}\ /\ \pi, s|\delta)\ \succ$$

$$= \ \prec\ s\ ,\ SSQ(C/\pi, s|\delta)\ \succ \ = \ \prec\ s\ ,\ SSQ(C/\pi', s|\delta)\ \succ\ .$$

And so the two relevant OUTs agree, and we're done, as long as the equalities claimed above are correct. The first equality is just the definition of SSQ for C_+ in terms of that for C. The substitution is a 'non-starter', giving the second one. Using **8.6**(d), the last equality holds since CV_1 implies (iii) in **8.6**(d) for C/π.

Validity of rule (IX) :

Assume that $F\{C/\pi\}G$ is true, and that no variable y_i nor any variable in any term r_i is in $FREE(C/\pi)$. To show, as required, that $F^{[\ \vec{y}\to\vec{r}\ ]}\{C/\pi\}G^{[\ \vec{y}\to\vec{r}\ ]}$ is also true, assume that $F^{[\ \vec{y}\to\vec{r}\ ]}$ is true at $s \circ \delta$, and that $SSQ(C/\pi, s|\delta)$ is a finite sequence. We must prove that $G^{[\ \vec{y}\to\vec{r}\ ]}$ is true at $(OUT(C/\pi, s|\delta)$.

Determine s_1 by

$$s_1 \circ \delta(z) \ = \ s \circ \delta(z) \ \text{ for } z \notin \text{“}\vec{y}\text{”} \text{ with } z \in \mathrm{dom}(\delta)\ ;$$

$$s_1 \circ \delta(y_i) \ = \ r_i^{s\circ\delta} \ ;$$

$$s_1(\mathrm{bin}_n) = s(\mathrm{bin}_n) \text{ for } \mathrm{bin}_n \notin \mathrm{Image}(\delta)\ .$$

Then $s_1|\delta$ is (empty subn. , $\vec{y}\to\vec{r}$)—matched to $s|\delta$. In twice applying **8.9** below, hypothesis (iii) there holds vacuously.

By **8.9**, F is true at $s_1 \circ \delta$, and so, since $F\{C/\pi\}G$ is true, G is true at $OUT(C/\pi, s_1|\delta)$.

By **8.10**, $OUT(C/\pi, s_1|\delta)$ is (empty subn. , $\vec{y}\to\vec{r}$)—matched to $OUT(C^{[\ \vec{y}\to\vec{r}\ ]}/\pi, s|\delta)$.

Thus, by **8.9**, $G^{[\ \vec{y}\to\vec{r}\ ]}$ is true at $OUT(C^{[\ \vec{y}\to\vec{r}\ ]}/\pi, s|\delta)$.

Now the following general result gives that last OUT to agree with $OUT(C/\pi, s|\delta)$, so we're done.

8.6(e) *If no y_i nor any variable in r_i is in $FREE(C/\pi)$, then $C^{[\ \vec{y}\to\vec{r}\ ]} = C$*

.

The proof of this almost tautological fact is a straightforward induction.

Validity of rule (X) :

Let $C = \mathsf{call}\ p(\vec{y} : \vec{w})$ and $C_+ = \mathsf{call}\ p(\vec{u} : \vec{t})$, with conditions as in rule (X) holding.

If $\pi(p)$ is not defined, there is nothing to prove, since then, $F_1\{C_+/\pi\}G_1$ is true for any F_1 and G_1, because $SSQ(C_+/\pi, -|-)$ is undefined..

So let $\pi(p) = (K, (\vec{x}, \vec{v}))$. Suppose that $F\{C/\pi\}G$ is true. To show that

$$F^{[\ \vec{y}\to\vec{u}\ ,\ \vec{w}\to\vec{t}\ ]}\{C_+/\pi\}G^{[\ \vec{y}\to\vec{u}\ ,\ \vec{w}\to\vec{t}\ ]}$$

also is true, as required, assume that $F^{[\ \vec{y}\to\vec{u}\ ,\ \vec{w}\to\vec{t}\ ]}$ is true at $s \circ \delta$, and that $SSQ(C_+/\pi, s|\delta)$ is a finite sequence.

We must prove that $G^{[\ \vec{y}\to\vec{u}\ ,\ \vec{w}\to\vec{t}\ ]}$ is true at $OUT(C_+/\pi, s|\delta)$.

Pick some $s_1|\delta_1$ which is $(\vec{y}\to \vec{u}, \vec{w}\to \vec{t})$—matched to $s|\delta$. Then F is true at $s_1 \circ \delta_1$ by **8.9**. And so G is true at $OUT(C/\pi, s_1|\delta_1)$, since $F\{C/\pi\}G$ is true.

Because of the way SSQ is defined for call-commands,

$$OUT(C_+/\pi, s|\delta) \ = \ OUT(K^{[\ \vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}\ ]}/\pi\ ,\ s|\delta)\ .$$

Define $L \ := \ K^{[\ \vec{x}\to\vec{y}\ ,\ \vec{v}\to\vec{w}\ ]}$, so that

$$K^{[\ \vec{x}\to\vec{u}\ ,\ \vec{v}\to\vec{t}\ ]} \ = \ L^{[\ \vec{y}\to\vec{u}\ ,\ \vec{w}\to\vec{t}\ ]}\ .$$

But this last equality gives that

$$OUT(C_+/\pi, s|\delta) \ = \ OUT(L^{[\ \vec{y}\to\vec{u}\ ,\ \vec{w}\to\vec{t}\ ]}/\pi\ ,\ s|\delta)\ .$$

Now

$$\begin{aligned} OUT(C/\pi, s_1|\delta_1) \ &:= \ OUT(\mathsf{call}\ p(\vec{y} : \vec{w})/\pi, s_1|\delta_1) \\ &= \ OUT(K^{[\ \vec{x}\to\vec{y}\ ,\ \vec{v}\to\vec{w}\ ]}/\pi\ ,\ s_1|\delta_1) \ := \ OUT(L/\pi, s_1|\delta_1)\ . \end{aligned}$$

The non-definitional equality is immediate from the definition of SSQ for call-commands.

By **8.10**, the right-hand side OUTs of the previous two displays are matched, and so the left-hand sides also are; that is, $OUT(C/\pi, s_1|\delta_1)$ is $(\vec{y}\!\!\rightarrow \vec{u}, \vec{w}\!\!\rightarrow \vec{t})$—matched to $OUT(C_+/\pi, s|\delta)$.

Using **8.9**, and the truth of G at $OUT(C/\pi, s_1|\delta_1)$, we see that $G^{[\ \vec{y}\rightarrow\vec{u}\ ;\ \vec{w}\rightarrow\vec{t}\ ]}$ is true at $OUT(C_+/\pi, s|\delta)$, as required.

Validity of rule (XI) :

Here we use a final general result about the semantics, whose proof is yet another straightforward induction.

8.6(f) *If $\pi \subset \pi'$ and $SSQ(C/\pi, s|\delta)$ is a finite sequence, then so is $SSQ(C/\pi', s|\delta)$, and they have the same last term.*

Now assume that $F\{C/\pi'\}G$ is true, and that F is true at $s \circ \delta$. Suppose that $SSQ(C/\pi, s|\delta)$ is a finite sequence. The result above gives that $SSQ(C/\pi', s|\delta)$ is also a finite sequence, and that

$$OUT(C/\pi', s|\delta) \ = \ OUT(C/\pi, s|\delta) \ .$$

So G is true at $OUT(C/\pi, s|\delta)$, as required to show that $F\{C/\pi\}G$ is also true.

8.4—The non-existence of a complete F-H proof system for the full languages ALGOL and PASCAL.

The language **CTEN** is nowhere close to full ALGOL. Only a few of the caveats apply there. For example, certainly recursive procedures are allowed, indeed, very much encouraged! (But see Addendum 3 below.) And there are function declarations, GOTO-commands, etc. in ALGOL. So the following question arises: for which practical software languages do we have the existence of a complete F-H proof system in the sense of the theorems of the last two subsections?

In [**Cl1**] there are a number of cases more general than **CTEN** of this last theorem where such a proof system is claimed to exist, and a partial proof is given in one of these cases. The literature does abound with papers on this sort of question. But many of them also refer, more-or-less explicitly, to mistakes in earlier papers. The situation can be mildly confusing to a neophyte with mathematical interests but not a lot of knowledge of the CS jargon nor of which claims are to be regarded as reliable.

Here are a few of the negative quotes:

> In [**Cl1**] it was argued that if use of global variables was disallowed, then denesting of internal procedures would be possible. Thus, the proof system ... could be adapted.... This argument was shown to be incorrect by Olderog.
>
> Clarke 1984[**Hoare-Shepherdson eds**]p.98

An amusing sidenote here is that, in the title in Clarke's reference list for his own paper (our [**Cl1**]), the word "impossible" got changed to "possible"!

> We thank ... G.Plotkin, for his elegant counterexamples to the soundness of Clarke's procedure call axiom.
>
> Trakhtenbrot, Halpern,Mayer 1983[**Clarke-Kozen eds**]p.497

> Whether such systems are to be used for formal verification, by hand or automatically, or as a rigorous foundation for informal reasoning, it is essential that they be logically sound. Several popular rules in the Hoare logic are in fact not sound. These rules have been accepted because they have not been subjected to sufficiently strong standards of correctness.
>
> M.J. O'Donnell 1981[**Kozen-ed.**]p.349

> The rules for procedure call statements often (in fact usually) have technical bugs when stated in the literature, and the rules stated in earlier versions of the present paper are not exceptions.
>
> S. Cook 1978[**C**]p.70

One major motivation for the (excessive?) detail in the last subsection is just this queasy situation. I hope that I got it right! As mentioned earlier, the paper [**Old1**] gives fairly comprehensive results, and seems to be unquestioned as far as accuracy is concerned, though some disagree with the specification method for the semantics. It deals essentially with producing careful analysis of F-H proof systems for the five cases of command languages with procedure declarations plus all but one of the five language features displayed below in italics, where we explain briefly why such a system couldn't exist with all five features in the language. In the third addendum below, we at least give a system for a language with recursive programming.

Note also that O'Donnell's remarks quoted above concentrate on two features of programming languages which have not been discussed at all here yet—*declared functions* and *GOTO-commands*. It is interesting that 'Frege's Folly', AKA Russell's Paradox, was inadvertantly re-created 90 years after Frege in trying to invent proof rules for function declarations, as first observed by Ed Ashcroft, an occurrence which undoubtedly helped to get people serious about proving soundness of proof rules in this subject. See Addendum 1 below for a short discussion on function declarations and Ashcroft's observation, and see O'Donnell's paper quoted above for other details.

I don't know whether there is anything close to a rigorously proved complete F-H proof system for an ALGOL-style language such that any 'known' language feature which is missing cannot be added without making it impossible for such a proof system to exist. Most work after the mid-80's seems to concentrate on parallel and concurrent programming.

So we'll finish with a brief description of the major *negative* result in [**Cl1**], which apparently implies that there cannot be a complete F-H proof system for full ALGOL. It is natural to wonder whether, and to what extent, all this theory influenced the design of C, C++, C#, etc.

No attempt will be made to give exhaustive mathematical details here. The negative argument of the first subsection gives us the following. Suppose given a command language $\mathcal{C}$ which has the $1^{\underline{\text{st}}}$order language $\mathcal{L}$ as its basis. Fix an interpretation I of $\mathcal{L}$, so that $\mathcal{C}$ can be given a corresponding semantics. Assume this is done with sufficient rigour that one can prove:

(1) the halting problem for $\mathcal{C}$ relative to I to be undecidable; and

(2) there is a complete axiomatic proof system for $\mathcal{L}$ relative to I.

It follows from (1) that the set of commands C which loop on all inputs is not recursively enumerable. But a complete F-H proof system plus (2) would imply that set to be recursively enumerable.

Now (2) will certainly hold if I is finite. Clarke's major achievement here was to show how, if $\mathcal{C}$ had a certain list of features, namely:

(if you know the jargon) —*procedures as parameters of procedure calls, recursion, static scope, global variables, and internal procedures*—

then, for every I with more than one element (including of course *finite* I), the halting problem for $\mathcal{C}$ is undecidable, so (1) holds. And therefore the standard argument of Cook above applies : *there can be no (relatively) complete F-H proof system for such a language.*

It hardly needs saying that this surprising theorem of Clarke has been of huge importance in the field.

Addendum 1 : Function Declarations and Ashcroft's 'Paradox'.

We introduce a very simple function declaration string, to jazz up the language in Subsection 1. Funnily enough, it takes the form

$$\mathsf{fun}\ y \sim f(x) \Leftarrow K\ \mathsf{nuf}\quad ,$$

where K is a command (very likely containing at least the variable x). Beforehand, one has specified a set $FIDE$ of function identifiers with typical member f. The above y is then intuitively "the value of the function f at x", where y really refers to the value of that variable in the state *after* K has been executed, and x really to its own value in the state *before* K is executed.

A function declaration is not dissimilar to a procedure declaration. But one is, so to speak, only interested in the one value, y, within the output from K, not with the entire output state. And there is no direct function **call**—$f(\cdots)$ is simply allowed to be used wherever we formerly used terms from the $1^{\underline{\text{st}}}$order language, anywhere within the block starting with the function declaration, in assertions, commands and other declarations. So, for example,

$$\mathbf{whdo}(f(x) + z < w \times f(t + z))(C)$$

and

$$w \div f(x) + z$$

would be a new sorts of commands, usable within that block; and

$$\forall w\ (f(x) + z < w \times f(t + z))$$

would be a new sort of assertion, for any variables x, z, w and any term t.

Thus one would define three sets, say ASS, COM and DEC, of assertions ("formulas"), commands and declarations, respectively, by simultaneous structural induction. We won't go into details here. For the example below, you can imagine that DEC has only function (not variable nor procedure) declarations; that COM is the usual **ATEN** except for also using blocks beginning with a declaration, and allowing the use of f as just above in assignments and in the formulas for controlling 'while-do'-commands; and that ASS is just '$1^{\underline{\text{st}}}$order number theory souped up with functions', again using f as just above, and also in substitutions for variables.

One sanitary requirement would be that any $f \in FIDE$ occurs at most once in declarations within a block. Also, we might as well forbid recursive function declarations for our simple purposes here.

A proper treatment would give also a careful definition of the semantics, depending on a chosen interpretation of the $1^{\underline{\text{st}}}$order language. Again what's below is independent of the interpretation, but if definiteness is desired, take it to be **N**.

Now a rule which had been proposed, to go along with the four usual rules for **ATEN**, is

$$\frac{F\{K\}G}{F \to G^{[y \mapsto f(x)]}} \quad ,$$

for any F, G in ASS, and K in COM, where f is a function identifier appearing in a declaration as at the beginning of the addendum. (We can weaken it to stick to $1^{\underline{\text{st}}}$order formulas F and G without defined functions, but the Ashcroft example below still applies.)

The reader is urged not to think too much about this rule right now, other than to say quietly: *"That looks reasonable—when F is true then G with y replaced by $f(x)$ must be true, if f is computed using K, and the F-H statement in the numerator is true"*. If you feel a bit puzzled about this rule, after the Ashcroft contradiction just below, our discussion will hopefully explain away any puzzlement.

Now let K be the command $\mathbf{whdo}(0 \approx 0)(x \leftarrow\!\!\!\!\div\ x)$, which for these purposes could be any command which just loops forever on any input. Let f occur in the declaration as at the beginning with this particular K.

Now we get a derivation as follows :

$$\begin{array}{l} \quad\bullet \\ \quad\bullet \\ \quad\bullet \\ 0 \approx 0\{K\}\neg 0 \approx 0 \\ 0 \approx 0 \to (\neg 0 \approx 0)^{[y \mapsto f(x)]} \\ \quad\bullet \\ \quad\bullet \\ \neg 0 \approx 0 \end{array}$$

The first few lines would be a derivation fragment as discussed in (iii) after the statement of **8.4** ; because K loops, one could have any pre- and post-conditions here. The next line just applies the new rule. And the last

few lines merely give a $1^{\underline{\text{st}}}$order derivation, using the logical validity of $0 \approx 0$, and a presumed rule which at least allows us to derive H from $H^{[y \to f(x)]}$ when the formula H has no free y. Actually H and $H^{[y \to f(x)]}$ must be exactly the same string under these circumstances, once all the definitions have been fleshed out in detail.

Being able to derive $\neg 0 \approx 0$ seems undesirable, to say the least, and we'll just leave it at that—there seems little motivation at this point to try to find any kind of semantics which are reasonable, and for which the given rule is sound!

Additional Remarks. At first, having a rule whose conclusion is like a $1^{\underline{\text{st}}}$order formula seems puzzling. Heretofore, each rule had an F-H statement as its conclusion. After all, if we are working 'oracularly' as discussed in Subsection 1, we are **given** all true $1^{\underline{\text{st}}}$order formulas by the oracle, so who needs to derive them?

Also the rule's conclusion isn't actually in the $1^{\underline{\text{st}}}$order language because of the occurrence of f.

These two objections seem to cancel each other : the conclusion *is* in the enlarged $ASS \supset FORM$, and maybe we want to be able to derive 'formulas' in ASS of the form $H \to H'$, for use in a souped up version of the rule

$$\text{(IV)} \qquad \frac{F \to F' \ , \ F'\{C\}G' \ , \ G' \to G}{F\{C\}G}$$

This unfortunate 'rule' was originally stated with a universally quantified conclusion $\forall x \ (F \to G^{[y \to f(x)]})$. However, the presence or absence of the quantifier is moot, if one is actually talking about truth in an interpretation, as opposed to truth at a particular state.

But now one ponders it a bit more and puzzles concerning the restriction in the rule about f appearing in a certain declaration. This is getting stranger, since after all, given any two $1^{\underline{\text{st}}}$order formulas F and G, we can certainly find a command K already in the original **ATEN** for which the numerator of this rule can be derived. That's just the discussion of (iii) after the statement of **8.4** again. So the 'rule' would allow one to derive *any* assertion as in its denominator. And it seems to have no relevance at all to F-H logic as it has been presented here, since f doesn't really mean anything in particular.

One suspects that the intent of this rule was for producing intermediate $1^{\underline{\text{st}}}$order formulas somewhere 'in the middle of an execution of a program'. These formulas were presumably supposed to be true for the state produced at that stage. The intended rule is perhaps more like this :

$$\frac{F\{K\}G}{Tr\{\mathsf{begin\ fun}\ y \sim f(x) \Leftarrow K\ \mathsf{nuf}\ ;\ Null\ \mathsf{end}\}(F \rightarrow G^{[y \rightarrow f(x)]})} \quad ,$$

where Tr and $Null$ are any formula and command which are logically valid and 'do nothing', respectively.

In any case, Ashcroft's contradiction above applies just as well in that context, so it was "back to the drawing-board". General discussions, building up from 'while' to more complicated command-constructions, of F-H logic, such as [**Apt**] or [**deB**], seem to carefully avoid this topic, perhaps for good reason. Other than deBakker-Klop-Meyer, p.94 in [**Kozen-ed.**], I haven't been able to find any thorough analysis of function declarations for F-H logic as it is normally presented. That paper doesn't include detailed proofs of soundness and completeness. It is interesting that the set-up there is quite elaborate, involving a functional programming language to supplement the command language, and 'meaning' definitions which are given using denotational semantics.

Addendum 2 : Propositional Connectives applied to F-H Statements, and Total Correctness.

There are two aspects of the formulation of F-H logic earlier which some may find unnatural because of a lack of generality :

(1) The F-H statement $F\{C\}G$ (or $F\{C/\pi\}G$) is like a *closed* $1^{\underline{\text{st}}}$order formula in that it is either true or false in an interpretation: its truth value does not vary with different states from a given interpretation. It may seem more natural to introduce a syntactic notation $[F : C : G]$ whose semantics will say that it is true at a given state $\underline{v}$ exactly when [*if* F *is true at* $\underline{v}$ *and* $||C||(\underline{v}) \neq err$, *then* G *is true at* $||C||(\underline{v})$]. In this addendum, we shall always have C from a command language that is called the 'while' language, basically **ATEN** plus the if-then-else command construction (a halfway house between **ATEN** and **BTEN**). So the semantics is self-evident. One can think of the string $F\{C\}G$ as the (universal) closure of $[F : C : G]$.

(2) The other 'unnaturality' is in only dealing with F-H statements themselves, and not, say, with negations or conjunctions of them, or implications between them. As illustrated in the next addendum, when trying to give an F-H proof system for a command language with procedures in which *recursive* programs are allowed, it is almost *necessary* to deal not just with F-H statements, but also with implications and conjunctions iteratively applied to them, such as

$$F\{C\}G \;\&\; F'\{C'\}G' \qquad \text{or} \qquad (\; F\{C\}G \implies F'\{C'\}G' \;) \;\&\; F''\{C''\}G'' \quad .$$

Temporarily, we shall use

$$\neg \;\; ; \;\; \& \;\; ; \;\; \implies \;\; ; \;\; \iff \;\; ,$$

to keep these connectives distinct from the analogous connectives

$$\neg \;\; ; \;\; \wedge \;\; ; \;\; \longrightarrow \;\; ; \;\; \longleftrightarrow \;\; ,$$

which are parts of formulae from $\mathcal{L}$, the underlying $1^{\underline{\text{st}}}$order language. There are some good reasons for this distinction, despite it being patronizing—the sensitive reader can adopt the attitude that it's me, not him or her, who needs to keep from getting confused!

So below we consider what happens when both of these extra generalities are introduced. This is related to (but is done somewhat differently than in) **dynamic logic**. Certainly we'll be less general than the latter's most abstract form, which, for example, has non-deterministic constructs—the output sometimes not being unique for a given input to a terminating program.

First we remark on how this extra generality (perhaps unexpectedly) expresses much more than the F-H *partial* correctness from which we started. But the spirit here is merely intellectual interest—no claim is made about the languages below being just the right ones for all the important practical questions about programs.

Here is some notation, part of which is stolen from dynamic logic.

The language consisting of **assertions**, denoted $\mathcal{A}$, $\mathcal{B}$, etc., depending on a given $1^{\underline{\text{st}}}$order language plus its associated 'while' command language, will be as follows :

atomic assertions: $[F : C : G]$

for F, G in the underlying $1^{\underline{\text{st}}}$order language; C from its 'while' language;

general assertions: $\neg\mathcal{A}$ $(\mathcal{A}\&\mathcal{B})$ $\forall x\mathcal{A}$ $[\mathcal{A} : C : \mathcal{B}]$

The last of these is needed for simplifying the formulation of the proof system.

Now use the following abbreviations :

$(\mathcal{A} \Longrightarrow \mathcal{B})$ abbreviates the assertion $\neg(\mathcal{A}\&\neg\mathcal{B})$;

$(\mathcal{A} \Longleftrightarrow \mathcal{B})$ abbreviates the assertion $((\mathcal{A} \Longrightarrow \mathcal{B})\&(\mathcal{B} \Longrightarrow \mathcal{A}))$;

Bracket removal abbreviations will be natural ones as in **[LM]**, no memorizing precedence rules! For example, only the outside brackets should disappear on that last string. But "&" is 'stickier' than both "$\Longrightarrow$" and "$\Longleftrightarrow$", as everyone knows without needing to memorize it!

Tr (for "true") abbreviates the formula $x_0 \approx x_0$;

Nu (for "null") abbreviates the command $x_0 \div x_0$;

$[C]G$ abbreviates the assertion $[Tr : C : G]$;

'G' abbreviates the assertion $[Nu]G$, that is, $[Tr : Nu : G]$;

$[C]\mathcal{A}$ abbreviates the assertion $[`Tr` : C : \mathcal{A}]$;

$<C>G$ abbreviates the assertion $\neg[C]\neg G$;

and

$<C>\mathcal{A}$ abbreviates the assertion $\neg[C]\neg\mathcal{A}$.

These abbreviations each get their semantics from the basic semantics for $[F : C : G]$ given above, plus the usual semantics for $\neg$, $\&$ and $\forall x$. The semantics for $[\mathcal{A} : C : \mathcal{B}]$ is exactly as given at the beginning of the addendum for its dwarf cousin $[F : C : G]$. This can be re-worded to say that $[\mathcal{A} : C : \mathcal{B}]$ is true at a given $\underline{v}$ exactly when [*$\mathcal{A}$ is false at $\underline{v}$, or $||C||(\underline{v}) = err$, or $\mathcal{B}$ is true at $||C||(\underline{v})$*].

Recall also that $\forall x\mathcal{A}$ is true at $\underline{v}$ iff $\mathcal{A}$ is true at all states $\underline{w}$ which agree with $\underline{v}$ except possibly at the variable x. When $\underline{v}$ and $\underline{w}$ are related like this, we abbreviate this to $\underline{v} \ \approx_x \ \underline{w}$. This may also be written : $\underline{v}(y) = \underline{w}(y)$ for all variables y except possibly $y = x$.

Thus 'G' is true at $\underline{v}$ iff G is true at $\underline{v}$ (in the normal $1^{\underline{\text{st}}}$order logic sense). So the new language can be thought of as an extension of $1^{\underline{\text{st}}}$order logic, and many would regard maintaining the distinction between G and 'G' as unnecessary.

As above, the F-H statement $F\{C\}G$ can be taken to be any assertion

$$\forall y_1 \cdots \forall y_k \ [F : C : G]$$

where the y_i include all the variables in C and every free variable in F or G.

It is a simple exercise to show that $[F : C : G]$ and 'F' $\Longrightarrow [C]G$ have exactly the same semantics; that is, they are *truth equivalent*.

But now consider the semantics of

$$`F` \Longrightarrow <C>G \ .$$

This is true at $\underline{v}$ iff

[either 'F' is false at $\underline{v}$ or $<C>G$ is true at $\underline{v}$] iff

[either F is false at $\underline{v}$ or $\neg[C]\neg G$ is true at $\underline{v}$] iff

[either F is false at $\underline{v}$ or $[Tr : C : \neg G]$ is false at $\underline{v}$] iff

[either F is false at $\underline{v}$ or [$||C||(\underline{v}) \neq err$ and $\neg G$ is false at $||C||(\underline{v})$]] iff

[if F is true at $\underline{v}$ then [C terminates at $\underline{v}$ and G is true at $||C||(\underline{v})$]] .

So we see that 'F' $\Longrightarrow <C> G$ *being true at all* $\underline{v}$ *from an interpretation is exactly the statement asserting the* **total** *correctness of the command* C *for precondition* F *and postcondition* G !

This is the (at least to me) slightly surprising fact which more-or-less tells us that a 'complete' theory of *partial* correctness which includes propositional connectives will necessarily include a theory of *total* correctness. (For total correctness, there no standard notation such as $F\{C\}G$ or $\{F\}C\{G\}$, which are almost universally used by partial correctologists.)

Another way of viewing this, as the dynamic logicians do, is as a kind of duality between the [ ] and <> operators, analogous to the duality between $\forall$ and $\exists$, or, more to the point, the duality between "necessity" and "possibility" operators in modal logic. But recasting everything in that style, with Kripke semantics, etc., seems to be unnecessary for us. Note however that [ ] and <> play a big rôle in expressing the proof system below. And also the final rule of inference below is similar to rules that occur in the theory of total correctness, different from any that have so-far appeared here. (It seems to be even different in a minor but essential way from what appears *anywhere* else, having the advantage over others of being sound!—see later comments on this.)

The semantics of $[C]\mathcal{A}$ and $<C>\mathcal{A}$ on their own (and similarly with G replacing $\mathcal{A}$, but we might as well just take $\mathcal{A} =$ 'G') can be easily checked to be the following:

$[C]\mathcal{A}$ is true at $\underline{v}$ iff either $||C||(\underline{v}) = err$ or $\mathcal{A}$ is true at $||C||(\underline{v})$.

$<C>\mathcal{A}$ is true at $\underline{v}$ iff both $||C||(\underline{v}) \neq err$ and $\mathcal{A}$ is true at $||C||(\underline{v})$.

Now we present a proof system for this [language plus semantics]. We must back off from previous generality and assume that the underlying $1^{\underline{\text{st}}}$order language is number theory and that the interpretation is $\mathbf{N}$. (See **[Harel]**, p. 29, for a notion of *arithmetical universe* which generalizes this.) We shall continue to deal only with derivations in which the premiss set is the highly unruly set of all 'F' for which F is true in $\mathbf{N}$.

Here is the system which gives the completeness theorem below for this assertion language. The first seven are axioms, and the rest are rules with at least one premiss.

The System

$$\text{(I)}\mathcal{A} \Longrightarrow \mathcal{A}\&\mathcal{A} \quad \text{(II)}\mathcal{A}\&\mathcal{B} \Longrightarrow \mathcal{A} \quad \text{(III)}(\mathcal{A} \Longrightarrow \mathcal{B}) \Longrightarrow (\neg(\mathcal{B}\&\mathcal{C}) \Longrightarrow \neg(\mathcal{C}\&\mathcal{A}))$$

These three plus (MP), Rosser's system, can be replaced by any axioms which, with (MP), give a complete system for classical propositional logic, a so-called 'Hilbert-style proof system'. This is referred to below as "*propositional completeness*".

$$\text{(IV)} \qquad [F : C : G] \iff (\text{`}F\text{`} \Longrightarrow [C]G)$$

$$\text{(V)} \qquad [x \doteqdot t]F \iff \text{`}F^{[x \to t]}\text{`}$$

$$\text{(VI)} \qquad [(C_1; C_2)]F \iff [C_1][C_2]F$$

Notice that, on the right-hand side here, we have an assertion of the form $[C_1]\mathcal{A}$, not just $[C_1]H$ for a formula H from the underlying $1^{\underline{\text{st}}}$order language. This is really why we needed the language to include the assertion construction $[\mathcal{A} : C : \mathcal{B}]$, or at least the case of it where $\mathcal{A}$ is 'Tr'. However, expressivity will say that $[C_1][C_2]F$ 'could be replaced' by $[C_1]H$ for some $1^{\underline{\text{st}}}$order formula H.

$$\text{(VII)} \qquad [\mathsf{ite}(H)(C_1)(C_2)]G \iff (\text{`}H\text{`} \Longrightarrow [C_1]G)\&(\text{`}\neg H\text{`} \Longrightarrow [C_2]G)$$

$$\text{(VIII)} \qquad \frac{\mathcal{A}\ ,\ \mathcal{A} \Longrightarrow \mathcal{B}}{\mathcal{B}}\ , \qquad \text{i.e. modus ponens, or (MP)}$$

$$\text{(IX)} \qquad \frac{\mathcal{A} \Longrightarrow \mathcal{B}}{[C]\mathcal{A} \Longrightarrow [C]\mathcal{B}}$$

$$\text{(X)} \qquad \frac{\mathcal{A} \Longrightarrow \mathcal{B}}{\forall x\ \mathcal{A} \Longrightarrow \forall x\ \mathcal{B}}$$

$$\text{(XI)} \qquad \frac{[F \wedge G : C : F]}{[F : \mathsf{whdo}(G)(C) : F \wedge \neg G]}$$

$$\text{(XII)} \qquad \frac{\text{`}F^{[x \to 0]} \to \neg G\text{`} \quad , \quad \text{`}F^{[x \to x+1]} \to G\text{`} \quad , \quad < F^{[x \to x+1]} : C : F >}{< \exists x F : \mathsf{whdo}(G)(C) : F^{[x \to 0]} >}$$

for $x \notin C \cup G$, and defining $< J : C : K > \ := \ \text{`}J\text{`} \Longrightarrow < C > K$.

Then we get a completeness theorem by adopting methods due to Harel in dynamic logic, and of course inspired by Cook :

There is a derivation for $\mathcal{A}$ if and only if $\mathcal{A}$ is true at all states from $\mathbf{N}$.

Proving this is in detail will be somewhat lengthy.

Except for brevity in the instance of proving validity of the last rule, (XII), the soundness half of the proof is a sequence of brief and straightforward verifications of validity of each axiom and rule separately. Validity for axioms means "true in $\mathbf{N}$". Validity for rules means that, when each assertion in the numerator is true in $\mathbf{N}$, then so is the denominator. One is not claiming to be true in $\mathbf{N}$ a (stronger than the rule) axiom which says "conjunction of numerator assertions $\Longrightarrow$ denominator assertion". That would amount to changing the second previous sentence by using "true at $\underline{v}$" everywhere. The deduction lemma in general form will not be deducible in this system, which is analogous to **System** from [**LM**], rather than to **System***.

A version of that last rule, (XII), is unsoundly stated in all other sources which I have seen. More specifically, the statements in [**Apt**], Rule 6, p.441, and [**Harel**] , bottom of p.37, fail to have the restriction that the variable x does not occur in G. Here is a simple counterexample to the soundness of that stronger rule used by those much-cited papers. We now drop the silly single quotes that we've been putting around each $1^{\underline{\text{st}}}$order formula .

$$\frac{(x \approx y)^{[x \mapsto 0]} \to \neg x + 1 \approx y \ , \ (x \approx y)^{[x \mapsto x+1]} \to x + 1 \approx y \ , \ < x + 1 \approx y : \text{“}y \dot{\leftarrow} y - 1\text{”} : x \approx y >}{< \exists x \ x \approx y : \mathsf{whdo}(x + 1 \approx y)(\text{“}y \dot{\leftarrow} y - 1\text{”}) : (x \approx y)^{[x \mapsto 0]} >}$$

Explanation : This is an example of rule (XII), except that x *does* occur in G, which is taken to be $x + 1 \approx y$ for a pair of distinct variables x and y. The formula F is taken to be $x \approx y$. The command C, written "$y \dot{\leftarrow} y - 1$", is intended to be any command (easy to come by) which decreases the value of y by 1 when the input is any state where y has positive value; which may do almost anything you want when y has value zero, including perhaps looping; which never affects the value of x; and indeed, when written out properly as a 'while'-command, has no occurrence of x.

It is evident from the description of C that the total correctness assertion on the right in the numerator is true in $\mathbf{N}$. The middle formula is tautological.

The truth in **N** of the formula on the left is established without any major intellectual effort. (It's not true in **Z**, of course.)

However, the total correctness statement in the denominator is certainly not true in **N**. Just take any state $\underline{v}$ in which $\underline{v}(x) = 1 = \underline{v}(y)$. The formula on the left, $\exists x\ x \approx y$, is true there (in fact, at any state, since it's logically valid. But actually $x \approx y$ itself is true there, making this also a counterexample to validity for Harel's version, as discussed in the next paragraph.) At this state $\underline{v}$, the command $\mathsf{whdo}(G)(C)$ does no passes of C at all, since $x + 1 \approx y$ is false at $\underline{v}$. And so termination does hold, with $||\mathsf{whdo}(G)(C)||(\underline{v}) = \underline{v}$. But the formula on the right in the denominator, namely $0 \approx y$, is certainly false at $\underline{v}$, so the correctness aspect fails. Thus the assertion in the denominator is false in **N**, as required. (If desired, one can certainly give an example where the whdo-command does any pre-assigned finite number of passes—but counterexamples here are necessarily to correctness, not to termination.)

In [**Harel**], p.37, the rule we've named (XII) also has the $\exists x$ in the denominator missing. That weakens it, but the above counterexample still works to prove unsoundness. For that reason, and since we have propositional completeness, Harel's system as he states it may be used to derive any assertion whatsoever, and so one cannot claim that it is inadequate. However, if one puts our restriction that $x \notin G$ into his rule, but continues to omit the "$\exists x$", I don't know how to prove that system to be adequate, if it is, in fact, adequate. He provides all the fundamental ideas for the proof of completeness we give below, but scrimps on details which would be needed to verify by means of such a proof that his system is complete.

Example: The assertion $< C > J$ is clearly true in **N**, where

$$C \text{ is } \mathsf{whdo}(0 < y)(\text{“}y \dot{\leftarrow} y - 1\text{”}) \ , \text{ and } J \text{ is } y \approx 0 \ .$$

A derivation of it (within our system) is fairly easy to come by: Taking x to be any variable not occurring in C, rule (XII) yields a derivation of

$$\exists x\ y \approx x \ \rightarrow \ \ < C > J \ ,$$

since J is $(y \approx x)^{[x \mapsto 0]}$. The rule is applicable since the corresponding assertions in the numerator of that rule, namely

$$y \approx 0 \rightarrow \neg 0 < y \ , \ \ y \approx x{+}1 \rightarrow 0 < y \ , \ \text{and} \ \ < y \approx x{+}1 : \text{“}y \dot{\leftarrow} y{-}1\text{”} : y \approx x >$$

are clearly all derivable, the first two being premisses. The last would have an elementary proof, once the command is written out in detail, involving only the first eleven axioms and rules. But now $\exists x\ y \approx x$, being logically valid, is true in $\mathbf{N}$ and so derivable, and thus (MP) immediately yields the required result.

But this argument wouldn't work using the fixed-up rule from [**Harel**], since we have to drop the "$\exists x$", so logical validity evaporates. I suspect, and will leave it to the reader to attempt to verify, that $< C > J$ is not derivable in the latter system, which would therefore be incomplete. In fact, it seems likely that $\mathcal{A} \rightarrow < C > J$ is derivable in that system only for assertions $\mathcal{A}$ for which $\mathcal{A} \rightarrow y \approx x$ is true in $\mathbf{N}$.

In the proofs below, we shall point out where use is made of the two matters discussed above re rule (XII) (certainly essential use, in the case of $x \notin G$, and probably for $\exists x$).

Here then is a proof that the rule (XII), as we state it, is sound.

With the notation from that rule, assume that the three assertions in its numerator are true in $\mathbf{N}$.

Let $\underline{v}$ be such that $\exists x\ F$ is true at $\underline{v}$. Choose some $\underline{w}$ with $\underline{v} \underset{x}{\sim} \underline{w}$, and with F itself true at $\underline{w}$. Let $n := \underline{w}(x)$. For $0 \leq i \leq n$, define $\underline{w}^{(i)} := \underline{w}^{(x \mapsto n-i)}$, which maps x to $n - i$, but any other y to $\underline{w}(y)$. Thus $\underline{w}^{(0)} = \underline{w}$; $\underline{w}^{(n)}(x) = 0_{\mathbf{N}}$; and $\underline{w}^{(i)} \underset{x}{\sim} \underline{v}$ for all i. Now we have a \tiny diagram of implications, explained below, where $\mathbf{t}$ is short for "is true at". Of course, the \tiny $\Longrightarrow$'s and $\Downarrow$'s in this diagram are from the metalanguage of this writeup, *not* from the formal language of assertions!

$$
\begin{array}{ccccccc}
F\mathbf{t}\underline{w}^{(0)} & \Rightarrow & F^{[x \to x+1]}\mathbf{t}\underline{w}^{(1)} & \Rightarrow & F\mathbf{t}||C||(\underline{w}^{(1)}) & \Rightarrow & F^{[x \to x+1]}\mathbf{t}||C||(\underline{w}^{(2)}) \\
\Downarrow & & \Downarrow & & \Downarrow & & \Downarrow \\
G\mathbf{t}\underline{w}^{(1)} & & G\mathbf{t}||C||(\underline{w}^{(2)}) & & G\mathbf{t}||C||^2(\underline{w}^{(2)}) & & \\
\Downarrow & & \Downarrow & & & & \\
G\mathbf{t}\underline{v} & & G\mathbf{t}||C||(\underline{v}) & & & &
\end{array}
$$

$$
\begin{array}{ccccc}
\Rightarrow & \cdots & \Rightarrow & F\mathbf{t}||C||^n(\underline{w}^{(n)}) \\
\Downarrow & & \Downarrow & \Downarrow \\
G\mathbf{t}||C||^{n-1}(\underline{w}^{(n)}) & \cdots & G\mathbf{t}||C||^{n-1}(\underline{v}) & \neg G\mathbf{t}||C||^n(\underline{w}^{(n)}) \\
\Downarrow & & & \Downarrow \\
F^{[x \to 0]}\mathbf{t}||C||^n(\underline{w}^{(n)}) & & & \neg G\mathbf{t}||C||^n(\underline{v})
\end{array}
$$

On the top row, the 1st, 3rd, 5th, etc. $\Longrightarrow$'s are correct because, quite generally, F being true at $\underline{u}^{(x \mapsto i)}$ implies that $F^{[x \to x+1]}$ is true at $\underline{u}^{(x \mapsto i-1)}$. We need also to use that, because $x \notin C$, one has $||C||^j(\underline{w}^{(i)}) = ||C||^j(\underline{w})^{(i)}$.

On the top row, the 2nd, 4th, etc. $\Longrightarrow$'s follow because of the truth of $< F^{[x \to x+1]} : C : F >$ in $\mathbf{N}$ (the rightmost assertion inside the numerator of the rule). It gives us that, when $F^{[x \to x+1]}$ is true at $\underline{u}$, it follows that F is true at $||C||(\underline{u})$, which is $\neq err$ there. Part of the intended information in the diagram is that each state appearing there is $\neq err$, any questionable case following because of the $\Longrightarrow$ coming into the spot where that state appears.

As for the downward implications which are *not* in the rightmost column, those in the upper row are correct immediately from the truth of $F^{[x \to x+1]} \to G$ (the middle formula from the numerator of the rule). That those in the second row are correct comes from observing that $\underline{w}^{(1)} \ \mathrel{\not\asymp} \ \underline{v}$, then that $||C||(\underline{w}^{(2)}) \ \mathrel{\not\asymp} \ ||C||(\underline{v})$ because $\underline{w}^{(2)} \ \mathrel{\not\asymp} \ \underline{v}$ and $x \notin C$, then that $||C||^2(\underline{w}^{(3)}) \ \mathrel{\not\asymp} \ ||C||^2(\underline{v})$, etc. $\cdots$; and also using the quite general fact that $[\underline{u} \ \mathrel{\not\asymp} \ \underline{z}$ and $x \notin G]$ implies that G is true at $\underline{u}$ if and only if G is true at $\underline{z}$. (This is the one place where $x \notin G$ is used. When proving adequacy, we'll point out how the restriction is not a problem.)

As for the downward implications which *are* in the rightmost column :

The top one is correct because of the general fact that if $\underline{u}(x) = 0_{\mathbf{N}}$, then H is true at $\underline{u}$ iff $H^{[x \to 0]}$ is true there, and the special fact that because $\underline{w}^{(n)}(x) = 0_{\mathbf{N}}$ and $x \notin C$, we get $||C||^i(\underline{w}^{(n)})(x) = 0_{\mathbf{N}}$ for all i .

The middle one is correct simply because $F^{[x \to 0]} \to \neg G$ (the leftmost formula from the numerator of the rule) is true in $\mathbf{N}$.

And the bottom one is correct just as with the bottom downward implications in the other columns.

Now that you buy the correctness of the diagram, we can finish quite quickly the proof that the assertion from the denominator of the rule is true in $\mathbf{N}$. The statements at the bottoms of all the columns in the diagram immediately show that the 'while-do' command does terminate on input $\underline{v}$. More precisely

$$||\mathsf{whdo}(G)(C)||(\underline{v}) \ = \ ||C||^n(\underline{v}) \ \neq \ err \ .$$

But now the formula $F^{[x \to 0]}$ is true at that state, as required. That last claim is just what appears as the second-from-top statement in the last column of the diagram, except that $\underline{v}$ has been replaced by $\underline{w}^{(n)}$. But that's no problem,

as those two states are related by $\underset{x}{\sim}$, and the formula in question has no free x, as it has been substituted by 0.

The three underlined statements in this proof say exactly that the assertion from the denominator of the rule is true in **N**, as required.

Proof of adequacy (completeness) of the system.

We must prove, for assertions $\mathcal{A}$:

$(*)$ if $\mathcal{A}$ is true in **N**, then $\mathcal{A}$ is derivable.

It will be convenient (and of educational value) to pass over to a slightly different language. In this language, each $1^{\underline{\text{st}}}$order formula will actually *be* an assertion (so we get rid of those silly single quotes 'F'). The main change is to use directly, not as an abbreviation, the string $[C]\mathcal{A}$, and we get rid of the $[\mathcal{A} : C : \mathcal{B}]$, which will be replaced by $A \Longrightarrow [C]\mathcal{B}$. This language will really be a specialized form of a dynamic logic language.

The language is defined in the usual structural inductive fashion :
Each atom will be an *atomic* $1^{\underline{\text{st}}}$order formula . (These are the formulas $s \approx t$ and $s < t$, for any terms s and t, built up using variables and the symbols $+$, $\times$, 0 and 1.)
Compound assertions are built as

$$\mathcal{A} \mapsto \neg\mathcal{A} \;\; ; \;\; (\mathcal{A}, \mathcal{B}) \mapsto (\mathcal{A} \wedge \mathcal{B}) \;\; ; \;\; \mathcal{A} \mapsto \forall x \; \mathcal{A} \;\; ; \;\; \text{and} \;\; \mathcal{A} \mapsto [C]\mathcal{A} \; .$$

The first three of these four build any $1^{\underline{\text{st}}}$order formula of course. We'll revert to $\neg$, $\wedge$ and $\rightarrow$, to conform with the notation used in **[LM]**, our canonical reference for $1^{\underline{\text{st}}}$order logic. (That gives the reward of letting us use $\Longrightarrow$ in the metalanguage of this discussion without fussy comments!)

The new proof system will be obtained simply from the old one :
(1) drop the single quotes on each $1^{\underline{\text{st}}}$order formula ;
(2) omit axiom (IV), which is meaningless for the new language anyway;
(3) alter rules (XI) and (XII) to

$$\frac{F \wedge G \rightarrow [C]F}{F \;\rightarrow\; [\mathsf{whdo}(G)(C)](F \wedge \neg G)}$$

and

$$\frac{F^{[x \to 0]} \rightarrow \neg G \quad , \quad F^{[x \to x+1]} \;\rightarrow\; (G \wedge < C > F)}{\exists x F \rightarrow < \mathsf{whdo}(G)(C) > F^{[x \to 0]}} \qquad \text{for } x \notin C \cup G \; .$$

In this new language, as before, define $< C > F \ := \ \neg[C]\neg F$.

The semantics of the new language is exactly the same as that of the old. But what is a definition here for, say, the primitive string $[C]F$ in the new language, was a little proposition for that abbreviation in the old language.

Now I claim that proving $(*)$ above for the new language and proof system suffices to get it for the old. That seems fairly obvious, but let us at least give a sketch of how to check this.

Use $\mathcal{L}_{\rm old}$, $\mathcal{L}_{\rm new}$, $\rm sys_{old}$, $\rm sys_{new}$ in the obvious way for the old and new languages and proof systems. The original $(*)$ says

$$\mathbf{N} \models \mathcal{O} \quad \text{implies} \quad \rm sys_{old} \vdash \mathcal{O}$$

for any $\mathcal{O}$ in the old language. Below we shall prove

$$\mathbf{N} \models \mathcal{N} \quad \text{implies} \quad \rm sys_{new} \vdash \mathcal{N}$$

for any $\mathcal{N}$ in the new language. In both cases, the $\vdash$ means to use the system, and to take every $1^{\underline{\rm st}}$order formula which is true in $\mathbf{N}$ as a premiss.

Define, by induction on structure, a map

$$\varphi \ : \ \mathcal{L}_{\rm old} \ \rightarrow \ \mathcal{L}_{\rm new} \ ,$$

via

$$\varphi([F : C : G]) \ := \ F \rightarrow [C]G \ ;$$

$$\varphi(\neg\mathcal{A}) := \neg\varphi(\mathcal{A}) \ ; \ \varphi(\mathcal{A} \wedge \mathcal{B}) := \varphi(\mathcal{A}) \wedge \varphi(\mathcal{B}) \ \ ; \ \ \varphi(\forall x \ \mathcal{A}) := \forall x \ \varphi(\mathcal{A}) \ ;$$

and

$$\varphi([\mathcal{A} : C : \mathcal{B}]) \ := \ \varphi(\mathcal{A}) \rightarrow [C]\varphi(\mathcal{B}) \ .$$

Then it is mildly tedious checking to show

$$\mathbf{N} \models \mathcal{O} \quad \text{if and only if} \quad \mathbf{N} \models \varphi(\mathcal{O}) \ ,$$

and

$$\rm sys_{old} \vdash \mathcal{O} \quad \text{if and only if} \quad sys_{new} \vdash \varphi(\mathcal{O}) \ .$$

These obviously do the needed job (in fact, just the "only if " in the first and the "if " in the second). We'll leave it to the reader to go through the checking, proceeding by induction on $\mathcal{O}$, which carefully verifies the two displays just above.

If that's disagreeable, you can think of the completeness of the new system as being the main point of this addendum. And regard the old language as just motivation being used to facilitate the passage from classical Floyd-Hoare logic to this concrete subtheory of dynamic logic.

Before proving adequacy for the new system, we need an **expressiveness result**, namely :

For any assertion $\mathcal{A}$ in the new language, there is a $1^{\underline{\text{st}}}$*order formula A such that $A \leftrightarrow \mathcal{A}$ is true in* $\mathbf{N}$.

The initial and three of the four inductive cases (of the proof by structural induction on $\mathcal{A}$) are very easy. The other case, when $\mathcal{A}$ is $[C]\mathcal{B}$, can, as with the proof of **8.3**, be done easily with the help of Gödel's definability result for semi-decidable relations, as follows :

Firstly, it's clearly equivalent(and turns out to seem linguistically simpler) to deal with $< C > \mathcal{B}$ instead of $[C]\mathcal{B}$. Let all the variables in C and the free ones in $\mathcal{B}$ be from among the distinct variables $y_1, \cdots, y_n$. Let $x_1, \cdots, x_n$ be distinct variables, disjoint from the y_i's. For the fixed command C, define a $2n$-ary relation, R_C , on natural numbers by

$$R_C(a_1, \cdots, a_n, b_1, \cdots, b_n) \iff ||C||(\vec{a}) = \vec{b} \ .$$

This is semi-decidable (indeed, the archetype of such!), so, by Gödel, let H be a $1^{\underline{\text{st}}}$order formula with free variables from among the x_i and y_i, such that

$$R_C(\vec{a}, \vec{b}) \iff H^{[\vec{x} \to \vec{a} \ , \ \vec{y} \to \vec{b}]} \text{ is true in } \mathbf{N} \ .$$

By the inductive hypothesis, choose a $1^{\underline{\text{st}}}$order formula B with free variables from among the y_i, such that $B \leftrightarrow \mathcal{B}$ is true in $\mathbf{N}$.

Now define A to be $\exists y_1 \cdots \exists y_n \ (H \wedge B)$.
Then

$$A \text{ is true at } \vec{a} \iff \text{for some } \vec{b} \ , \ (H \wedge B)^{[\vec{y} \to \vec{b}]} \text{ true at } \vec{a} \iff$$

$$\text{for some } \vec{b} \ , \text{ both } H^{[\vec{x} \to \vec{a} \ , \ \vec{y} \to \vec{b}]} \text{ and } B^{[\vec{y} \to \vec{b}]} \text{ are true in } \mathbf{N} \iff$$

$$\text{for some } \vec{b} \ , \text{ both } R_C(\vec{a}, \vec{b}) \text{ holds and } B \text{ is true at } \vec{b} \iff$$

$$\text{for some } \vec{b} \ , \ ||C||(\vec{a}) = \vec{b} \text{ and } B \text{ is true at } ||C||(\vec{a}) \iff$$

$$||C||(\vec{a}) \neq err \text{ and } \mathcal{B} \text{ is true at } ||C||(\vec{a}) \iff < C > \mathcal{B} \text{ is true at } \vec{a} \ .$$

Thus, as required, $A \leftrightarrow \ < C > \mathcal{B}$ is true in $\mathbf{N}$.

Now let's proceed to the proof of adequacy for the new system. Accuse an occurrence of $\forall x$ in $\mathcal{A}$ of being dull when its scope is a $1^{\underline{\text{st}}}$order formula. Define, for $\mathcal{A}$ in the new language,

$$M_{\mathcal{A}} \ := \ \#\text{non-dull occurrences in } \mathcal{A} \text{ of } \forall x \text{ for various } x \ + \\ \#\text{occurrences in } \mathcal{A} \text{ of } [C] \text{ for various } C \ .$$

Then $\mathcal{A}$ is a $1^{\underline{\text{st}}}$order formula if and only if $M_{\mathcal{A}} = 0$. (As it happens, if #occurrences in $\mathcal{A}$ of $[C]$ is 0, then every occurrence of $\forall x$ is in fact dull.)

We shall prove $(*)$ (for $\mathcal{A}$ from the new language) by induction on $M_{\mathcal{A}}$. Since all $1^{\underline{\text{st}}}$order formulas are premisses, the start of the induction is trivial.

Here is the overall sketch of the lengthy inductive step, in reverse order :

First reduce it to proving $(*)$ for assertions $\mathcal{A} \rightarrow \mathsf{sym}\mathcal{B}$, in the four cases of the symbol string

$$\mathsf{sym} \ = \ [C] \ \text{ or } \ \neg[C] \ \text{ or } \ \forall x \ \text{ or } \ \neg\forall x \ ,$$

where the latter two cases involve non-dull occurrences (that is, $\mathcal{B}$ is not a $1^{\underline{\text{st}}}$order formula).

Further reduce it to proving $(*)$ for assertions of the form $F \longrightarrow [C]G$ and $F \longrightarrow \neg[C]G$ for $1^{\underline{\text{st}}}$order formulas F and G. (So we are down to two particular cases of when $M_{\mathcal{A}} = 1$.)

Finally give separate, somewhat parallel, lengthy arguments for those two cases.

Now we proceed to do these in reverse order.

Proof of $(*)$ for $\mathcal{A}$ of the form $F \longrightarrow [C]G$.

This proceeds by structural induction on C, simultaneously for all $1^{\underline{\text{st}}}$order formulas F and G.

When C is $x \dot{\leftarrow} t$: By (V), $G^{[x \rightarrow t]} \ \longrightarrow \ [x \ \dot{\leftarrow} \ t]G$ is derivable, so it remains to show $F \longrightarrow G^{[x \rightarrow t]}$ is derivable, and then to apply hypothetical syllogism from propositional completeness. But the $1^{\underline{\text{st}}}$order formula $F \longrightarrow G^{[x \rightarrow t]}$ is a premiss, since it is true in $\mathbf{N}$, because both $F \longrightarrow [x \dot{\leftarrow} t]G$ [by assumption] and $[x \dot{\leftarrow} t]G \ \longrightarrow \ G^{[x \rightarrow t]}$ [soundness of half-rule (V)] are true in $\mathbf{N}$.

When C is $\mathsf{ite}(H)(C_1)(C_2)$: Use (VII) more-or-less as we used (V) just above. We need only show

$$F \ \longrightarrow \ (H \longrightarrow [C_1]G) \wedge (\neg H \longrightarrow [C_2]G)$$

to be derivable. By propositional completeness, this reduces to the derivability separately of

$$F \longrightarrow (H \longrightarrow [C_1]G)) \quad \text{and} \quad F \longrightarrow (\neg H \longrightarrow [C_2]G) \ ,$$

or indeed

$$F \wedge H \longrightarrow [C_1]G) \quad \text{and} \quad F \wedge \neg H \longrightarrow [C_2]G \ .$$

The latter two are immediate, by the induction on C, since the truth in $\mathbf{N}$ of the latter formulae is clear from the truth of $F \longrightarrow [C]G$.

When C is $(C_1; C_2)$: Here axiom (VI) is of course crucial, but expressivity (not unexpectedly!) is also involved. By expressivity, choose a 1^{st}order formula H so that $H \longleftrightarrow [C_2]G$ is true in $\mathbf{N}$. By the structural induction on C, the assertion $H \longrightarrow [C_2]G$ is derivable. So, by (IX), the assertion $[C_1]H \longrightarrow [C_1][C_2]G$ is derivable. By hypothetical syllogism, and since $[C_1][C_2]G \longrightarrow [C]G$ is derivable, [i.e. 'is' half of axiom (VI)], it remains only to show that the assertion $F \longrightarrow [C_1]H$ is derivable. This follows from the inductive hypothesis as long as that assertion is true. But in

$$F \ \longrightarrow \ [(C_1; C_2)]G \ \longrightarrow \ [C_1][C_2]G \ \longrightarrow \ [C_1]H$$

we have that 'each $\longrightarrow$' is true, respectively, by assumption, by soundness of half-(VI), and essentially by soundness of half-(IX), knowing $[C_2]G \longrightarrow H$ to be true.

When C is $\mathsf{whdo}(H)(D)$: Using expressivity, choose a 1^{st}order formula J with $J \longleftrightarrow [C]G$ true in $\mathbf{N}$.

I claim that each of the following is derivable :

$$(1)\ F \to J \ ; \quad (2)\ J \to [C](J \wedge \neg H) \ ; \quad (3)\ J \wedge \neg H \to G \ .$$

If so, then (3) and axiom (IX) give $[C](J \wedge \neg H) \to [C]G$ to be derivable. So the double application of hypothetical syllogism to that assertion, (2) and (1) yields the desired derivation of $F \to [C]G$.

Verifying (1) : Both '$\to$'s in $F \to [C]G \to J$ are true (by assumption and choice of J), and so $F \to J$ is a premiss.

Verifying (2) : The new rule (XI) :

$$\frac{J \wedge H \to [D]J}{J \to [\mathsf{whdo}(H)(D)](J \wedge \neg H)} \quad ,$$

leaves us only to derive $J \wedge H \to [D]J$. That follows from its truth in **N** by the overall induction on C of this part of the proof. To see its truth : for a contradiction, suppose $\underline{v}$ is such that $J \wedge H$ is true at $\underline{v}$, $||D||(\underline{v}) \neq err$, and J is false at $||D||(\underline{v})$. From 'while-do' semantics and the truth of H at $\underline{v}$, we get

$$||C||(\underline{v}) \;=\; ||C||(||D||(\underline{v})) \;.$$

Thus J being false at $||D||(\underline{v})$ gives, by the specification of J, that $[C]G$ is false at $||D||(\underline{v})$. Thus the right-hand side of the display is $\neq err$, and G is false there.

So the left-hand side of the display is $\neq err$. But then, since J, therefore $[C]G$, is true at $\underline{v}$, we see that G is true at the left-hand side of the display.

The underlined statements contradict the display.

Verifying (3) : That $1^{\underline{\text{st}}}$order formula is a premiss, since its truth in **N** is easy to see : If $J \wedge \neg H$ is true at $\underline{v}$, then so are :

$\neg H$, giving $||C||(\underline{v}) = \underline{v}$, and

J , and so $[C]G$, giving either $||C||(\underline{v}) = err$ or G true at $||C||(\underline{v})$.

So, indeed, G is true at $\underline{v}$, as required.

Proof of $(*)$ for $\mathcal{A}$ of the form $F \longrightarrow \neg[C]G$.

Since $< C > G := \neg[C]\neg G$, derivability of $F \longrightarrow \neg[C]G_1$ (for all F and G_1 where it's true) is equivalent to derivability of $F \longrightarrow < C > G$ (for all F and G where *it* is true)—by 'taking G and G_1 to be negations of each other', so to speak. Proving the latter also proceeds by structural induction on C, simultaneously for all F and G, quite analogously to the previous proof, though the last case, of a 'while-do' command, is somewhat harder.

Indeed, the first three cases may be done almost word-for-word as before, changing [ ] to <> everywhere. For this, we need only show to be derivable the assertions $(\text{V})_{<>}$, $(\text{VI})_{<>}$, and $(\text{VII})_{<>}$, these being the corresponding axioms with [ ] changed to <> everywhere. To check them :

$$< x \div t > F \;=\; \neg[x \div t]\neg F \;\longleftrightarrow\; \neg(\neg F)^{[x \to t]} \;=\; \neg\neg F^{[x \to t]} \;\longleftrightarrow\; F^{[x \to t]} \;.$$

The first '$\longleftrightarrow$' is derivable using (V) and propositionality; and the second is just propositionality. For (VI), write down

$$< C_1 >< C_2 > F \;=\; \neg[C_1]\neg\neg[C_2]\neg F \;\longleftrightarrow\; \neg[C_1][C_2]\neg F$$
$$\longleftrightarrow\; \neg[(C_1;C_2)]\neg F \;=\; < (C_1;C_2) > F \;,$$

and say a few words. The last one even gives some minor propositional subtlety :

$$< \mathsf{ite}(H)(C_1)(C_2) > G \;=\; \neg[\mathsf{ite}(H)(C_1)(C_2)]\neg G$$
$$\longleftrightarrow\; \neg((H \longrightarrow [C_1]\neg G) \wedge (\neg H \longrightarrow [C_2]\neg G))$$
$$\longleftrightarrow\; \neg((H \longrightarrow \neg < C_1 > G) \wedge (\neg H \longrightarrow \neg < C_2 > G))$$
$$\longleftrightarrow\; \neg((< C_1 > G \longrightarrow \neg H) \wedge (< C_2 > G \longrightarrow H))$$
$$\longleftrightarrow\; (H \longrightarrow < C_1 > G) \wedge (\neg H \longrightarrow < C_2 > G) \;.$$

Each '$\longleftrightarrow$' is in fact derivable in the system, the last one being the mildly subtle fact that

$$\neg((J \longrightarrow \neg H) \wedge (K \longrightarrow H)) \qquad \text{and} \qquad (H \longrightarrow J) \wedge (\neg H \longrightarrow K)$$

are derivable from each other in propositional logic.

This leaves only one case in the structural inductive proof of the derivability of $F \rightarrow < C > G$ when that assertion is true, namely the case

When C is $\mathsf{whdo}(H)(D)$:
Pick a $1^{\underline{\text{st}}}$order formula J and a variable $x \notin D \cup G \cup H$ such that
J is true at $\underline{v}$ if and only if : with $n := \underline{v}(x)$ we have

(α) $||D||^n(\underline{v}) \neq err$ and H is true at $||D||^i(\underline{v})$ for $0 \leq i < n$;
and
(β) $G \wedge \neg H$ is true at $||D||^n(\underline{v})$.

To prove that J can be found, first define a command

$$D^{\#} \;:=\; \mathsf{whdo}(0 < x)(D \;;\text{“}x \Leftarrow x - 1\text{”}) \;.$$

Let $\vec{y} = (y_1, y_2, \cdots, y_r)$ be a list of distinct variables including all those in $D \cup G \cup H$. Here, as earlier, the command “$x \Leftarrow x - 1$” has the effect, when the value of x is positive, of decreasing it by 1 without altering any y_i (but it can behave arbitrarily otherwise.) We'll 'contract' $\underline{v}$ down to just writing the relevant variables. Then

$$||D^{\#}||(n, \vec{a}) \;=\; (0, ||D||^n(\vec{a})) \;,$$

where the first coordinate is the value of x and the vector coordinate is the value of $\vec{y}$ [and $(0, err)$ on the right-hand side would mean just err of course].

Now use expressivity for the assertion $<D^{\#}> H$ to find a $1^{\underline{\text{st}}}$order formula K such that

$$K^{[x\mapsto i\ ,\ \vec{y}\mapsto\vec{a}]} \text{ is true in } \mathbf{N} \iff ||D^{\#}||(i,\vec{a}) \neq err \text{ and } H \text{ is true there .}$$

Use expressivity again, this time for the assertion $<D^{\#}> (G \wedge \neg H)$ to find a $1^{\underline{\text{st}}}$order formula L such that

$$L^{[x\mapsto n\ ,\ \vec{y}\mapsto\vec{a}]} \text{ is true in } \mathbf{N} \iff ||D^{\#}||(n,\vec{a}) \neq err \text{ and } (G\wedge\neg H) \text{ is true there .}$$

Then let z be a 'new' variable, and define the required formula by

$$J \ := \ \forall z\ ((z < x \to K^{[x\mapsto z]}) \wedge (z \approx x \to L^{[x\mapsto z]}))\ .$$

Then

$$J \text{ is true at } (n,\vec{a}) \iff J^{[x\mapsto n\ ,\ \vec{y}\mapsto\vec{a}]} \text{ is true in } \mathbf{N} \iff$$

$$\forall z\ ((z < n \to K^{[x\mapsto z\ ,\ \vec{y}\mapsto\vec{a}]}) \wedge (z \approx n \to L^{[x\mapsto z\ ,\ \vec{y}\mapsto\vec{a}]})) \text{ is true in } \mathbf{N} \iff$$

$$\text{true in } \mathbf{N} \text{ are : } K^{[x\mapsto i\ ,\ \vec{y}\mapsto\vec{a}]} \text{ for } 0 \leq i < n, \text{ and } L^{[x\mapsto n\ ,\ \vec{y}\mapsto\vec{a}]} \iff$$

$$||D^{\#}||(i,\vec{a}) \neq err \text{ and } H \text{ is true there for } 0 \leq i < n \text{ and}$$

$$||D^{\#}||(n,\vec{a}) \neq err \text{ and } G \wedge \neg H \text{ is true there} \iff$$

$$||D||^n(\vec{a}) \neq err, \text{ the formula } G\wedge\neg H \text{ is true there, and } H \text{ is true at } ||D||^i(\vec{a}) \text{ for } 0 \leq i < n\ ,$$

as required.

I claim that the following are all true in $\mathbf{N}$.

$$(1)\ J^{[x\mapsto 0]} \to G \wedge \neg H\ ; \quad (2)\ J^{[x\mapsto x+1]} \to H \wedge <D> J\ ; \quad (3)\ F \to \exists x\ J\ .$$

Suspending disbelief in this for the moment, the following are then derivable :

$$J^{[x\mapsto 0]} \to \neg H \quad ; \quad J^{[x\mapsto x+1]} \to H \quad ; \quad J^{[x\mapsto x+1]} \to <D> J\ ;$$

the first two being premisses, and the latter because it is true and therefore derivable by the induction on C.

By rule (XII) and since $x \notin D \cup H$, we get a derivation of

$$\exists x\ J \ \to \ <C> J^{[x\mapsto 0]}\ .$$

But the $1^{\underline{\text{st}}}$order formula $J^{[x\to 0]} \to G$ is a premiss, and therefore the assertion $< C > J^{[x\to 0]} \to < C > G$ is derivable by a rule which is (IX) except with $[C]$ replaced by $< C >$, and which is easily verified from (IX) and the definition of $< C > F$. Finally $F \to \exists x\ J$ is true, so derivable. A double dose of hypothetical syllogism then shows $F \to < C > G$ to be derivable, as required.

It remains to check the truth of (1), (2) and (3), using the assumed truth of $F \to < \mathsf{whdo}(H)(D) > G$.

(1) Assume $J^{[x\to 0]}$ is true at $\underline{w}$. Then J itself is true at $\underline{v} := \underline{w}^{(x\mapsto 0)}$. So $n = 0$ in the specification defining J. From (β) we get that $G \wedge \neg H$ is true at $\underline{v}$. But since $\underline{v} \ \overset{x}{\sim}\ \underline{w}$ and $x \notin G \cup H$, we get, as required, that $G \wedge \neg H$ is true at $\underline{w}$.

(2) Assume that $J^{[x\to x+1]}$ is true at $\underline{w}$.

Then J itself is true at the state $\underline{v} := \underline{w}^{(x\mapsto \underline{w}(x)+1)}$. In the specification of J, we have $n = \underline{v}(x) = \underline{w}(x) + 1 > 0$. So $i = 0$ occurs in $(\alpha)_{\underline{v},n}$, and we get that H is true at $||D||^0(\underline{v}) = \underline{v}$. So H is true also at $\underline{w}$ (which is half the required), since $\underline{v} \ \overset{x}{\sim}\ \underline{w}$ and $x \notin H$.

But, since $n \geq 1$, condition $(\alpha)_{\underline{v},n}$ also gives $||D||(\underline{v}) \neq err$. And so $||D||(\underline{w}) \neq err$, since $\underline{v} \ \overset{x}{\sim}\ \underline{w}$ and $x \notin D$. Furthermore, J is true at $||D||(\underline{w})$: We check this using the definition of J as follows. Here

$$||D||(\underline{w})(x) = \underline{w}(x) = n - 1\ ,$$

so we must verify conditions $(\alpha)_{||D||(\underline{w}),n-1}$ and $(\beta)_{||D||(\underline{w}),n-1}$.

For the first, note that $||D||^{n-1}(||D||(\underline{w})) = ||D||^n(\underline{w}) \neq err$, since we have $||D||^n(\underline{v}) \neq err$. Also, for $0 \leq i < n - 1$, the formula H is true at $||D||^i||D||(\underline{w})) = ||D||^{i+1}(\underline{w})$, since again, $\underline{w}$ may be replaced by $\underline{v}$, and because $i + 1 < n$, making use of $(\alpha)_{\underline{v},n}$.

For $(\beta)_{||D||(\underline{w}),n-1}$, we again use the fact that $||D||^{n-1}||D||(\underline{w})) = ||D||^n(\underline{w})$, and $G \wedge \neg H$ is indeed true there, because it is true at $||D||^n(\underline{v})$.

The underlined above show that $J^{[x\to x+1]} \to < D > J$ is true in $\mathbf{N}$, completing the discussion of (2).

(3) Assume that F is true at $\underline{v}$. Our basic assumption that the assertion $F \to < \mathsf{whdo}(H)(D) > G$ is true in $\mathbf{N}$ yields that

$$||\mathsf{whdo}(H)(D)||(\underline{v}) \ = \ ||C||(\underline{v}) \ \neq \ err\ ,$$

and that G is true at $||C||(\underline{v})$. Thus, there is a (unique of course) $n \geq 0$ such that $(\alpha)_{\underline{v},n}$ and $(\beta)_{\underline{v},n}$ hold, by the semantics of 'whdo'. Now define $\underline{w}$ by $\underline{w} \ \aleph \ \underline{v}$ and $\underline{w}(x) = n$. By the definition of J, the latter formula is true at $\underline{w}$. Since $\underline{w} \ \aleph \ \underline{v}$, it is immediate from the basic (Tarski) definition of truth for an existential $1^{\underline{\text{st}}}$order formula that $\exists x \ J$ is true at $\underline{v}$, as required. (But not necessarily J itself, so more is needed to establish adequacy with a weakening of Harel's rule to make it sound, perhaps more than more!)

This finally completes the (structural induction on C) proof of $(*)$ for $\mathcal{A}$ of the form $F \longrightarrow < C > G$, and so for $\mathcal{A}$ of the form $F \longrightarrow \neg[C]G$.

To complete the (induction on $M_{\mathcal{A}}$) proof of $(*)$ for *all* $\mathcal{A}$ in the new language, let sym denote one of the symbol strings

$$[C] \quad \text{or} \quad \neg[C] \quad \text{or} \quad \forall x \quad \text{or} \quad \neg\forall x \quad .$$

Lemma. *For any $\mathcal{C}$ in the new language, there is a propositional derivation of an assertion of the form*

$$\mathcal{C} \iff \wedge_\alpha \mathcal{D}_\alpha \quad ,$$

where the finite conjunction on the right-hand side has each $\mathcal{D}_\alpha$ as a finite disjunction of assertions, each of which has the following form : either each is a $1^{\underline{\text{st}}}$order formula , or else at least one of them has the form sym$\mathcal{B}$ *for our four possible* sym*'s, where $\mathcal{B}$ is not a $1^{\underline{\text{st}}}$order formula when* sym *is either $\forall x$ or $\neg\forall x$.*

The proof of this proceeds by structural induction on $\mathcal{C}$, and is simplified by simultaneously proving the dual fact, where the prefixes "con" and "dis" trade places. The initial case is trivial. In two inductive cases, namely $\mathcal{A} \mapsto \forall x \ \mathcal{A}$; and $\mathcal{A} \mapsto [C]\mathcal{A}$, one simply has a single conjunct (resp. disjunct), which itself is a single disjunct (resp. conjunct). The inductive step for negations is easy because of proving the duals simultaneously : up to cancellation of double negations, deMorganization converts each expression from the right-hand side of the lemma to its dual. The symmetry breaks for the final case of conjuncting two assertions. For the actual lemma statement, the result is numbingly obvious. For the dual statement, it is run-of-the-mill obvious by using distributivity of $\wedge$ over $\vee$. (All we are really talking about here is conjunctive and disjunctive form.)

Since a conjunction is derivable (resp. true in $\mathbf{N}$) if and only if each conjunct has the same property (using propositionality of our system for the derivability), and since every true $1^{\underline{\text{st}}}$order formula is a premiss, the lemma reduces the question to proving $(*)$ only for assertions of the form $\mathcal{E} \vee \mathsf{sym}\mathcal{B}$. (One may use $\mathcal{E} = \ 0 \approx 1$ for any conjunct consisting of a single disjunct $\mathsf{sym}\mathcal{B}$.) Taking $\mathcal{A}$ as $\neg\mathcal{E}$,

we need only prove $(*)$ for assertions of the form $\mathcal{A} \to \mathsf{sym}\mathcal{B}$,

where, since $\mathcal{B}$ is not a $1^{\underline{\text{st}}}$order formula in two of four relevant cases, the inductive assumption applies to all assertions whose 'M-function' does not exceed that of one or the other of $\mathcal{A}$ or $\mathcal{B}$. All we are saying here is that

$$M_{\mathcal{A}\to\mathsf{sym}\mathcal{B}} \ = \ M_{\mathcal{A}} + M_{\mathcal{B}} + 1 \ .$$

By expressivity, choose A and B, each a $1^{\underline{\text{st}}}$order formula , such that both $A \longleftrightarrow \mathcal{A}$ and $B \longleftrightarrow \mathcal{B}$ are true in $\mathbf{N}$.

The proof will now be completed by considering each of the four possibilities for sym.

$\underline{\mathsf{sym} \text{ is } [C]}$: Use hypothetical syllogism after establishing the derivability of the three '$\to$' below :

$$\mathcal{A} \ \to \ A \ \to \ [C]B \ \to \ [C]\mathcal{B} \ .$$

$\mathcal{A} \to A$ is true, therefore derivable by the induction on the number M.
$B \to \mathcal{B}$ is true, therefore derivable by induction, and then so is the assertion $[C]B \to [C]\mathcal{B}$ derivable, using axiom (IX).
Derivability of $A \to [C]B$ follows from its truth and the first of the earlier agonizingly established $(*)$'s. Its truth is clear, since it 'factors into true assertions' :

$$A \ \to \ \mathcal{A} \ \to \ [C]\mathcal{B} \ \to \ [C]B \ .$$

$\underline{\mathsf{sym} \text{ is } \forall x}$: Just replace $[C]$ by $\forall x$ everywhere in the previous paragraph, and appeal to (X) rather than (IX), and get derivability of $A \to \forall x \ B$ much more easily, as a premiss.

$\underline{\mathsf{sym} \text{ is } \neg[C]}$: Just replace $[C]$ by $\neg[C]$ everywhere in the second previous paragraph, and make a few tiny wording changes :

Use $\mathcal{B} \rightarrow B$ true, so derivable by induction on the number M, to get $[C]\mathcal{B} \rightarrow [C]B$, and thence $\neg[C]B \rightarrow \neg[C]\mathcal{B}$ both derivable.

This case of course uses also the even more agonizingly established version of $(*)$ giving the derivability of $A \rightarrow \neg[C]B$.

sym is $\neg\forall x$: Write out the previous paragraph properly, then replace each $\neg[C]$ by $\neg\forall x$, again appealing simply to the 'premissiveness' of $A \rightarrow \neg\forall x\ B$, rather than needing the prickly proof of the derivability of $A \rightarrow \neg[C]\ B$.

So we're done : the proof system is indeed complete.

An exercise is to derive directly, from this system, all the rules from the first subsection of this chapter, as well as the rules at the beginning of the second subsection.

I don't yet know how to generalize this completeness to *arbitrary* underlying $1^{\underline{\text{st}}}$order languages and interpretations, nor to a system which merely extends the collection of F-H statements by adding in the propositional connectives—but these seem natural questions to consider. Nor do I understand why some denotational semanticizers require that one should use intuitionistic propositional logic here, rather than the classical logic as exemplified by modus ponens plus the first three axioms above.

Exercise. Going back to the old language beginning this addendum, show that $\neg F\{C\}\neg G$ is true in an interpretation iff there is at least one state $\underline{v}$ for which F is true at $\underline{v}$, and $||C||(\underline{v}) \neq err$, and G is true at $||C||(\underline{v})$.

It seems that one cannot get total correctness from partial correctness plus propositional connectives directly. One apparently needs one of the (interdefinable and slightly more refined) syntactic 'concepts' $[F : C : G]$ or $[C]G$ or $<C>G$ or $< F : C : G >$.

Addendum 3 : F-H Proof Systems where Recursive Programming Occurs.

Stephen Cook in [**C**] has at least the following fundamental accomplishments in this subject :

(1) isolating a notion of relative completeness which is *meaningful*, and generally (but not perhaps universally) regarded as significant;

(2) inventing the fundamental idea of the post relation (or *strongest post-condition*, as it is more popularly named) and its $1^{\underline{st}}$order definability (i.e. expressiveness), and using it to establish a benchmark completeness theorem for the **ATEN**-language (or, more accurately, the 'while-language') as in the first two subsections above;

(3) pretty much establishing the completeness and soundness of a proof system for the F-H statements concerning a command language with procedures which have parameters and global variables, as exposited at length in our third subsection.

One major sense in which this completeness work is 'incomplete' is that recursive procedures are forbidden. Here we give a command language which includes recursive programming, but vastly simplify by having no parameters. This is also simplified by avoiding declarations of variables. The semantics is given in the same style as the third section. Then we write down a proof system which is sound and complete, but don't include the proofs of the latter claims. At the end are some comments on what appear to this author to be the most (perhaps the only) reliable write-ups on systems for languages which get closer to 'real-live Algol', by having recursive declared procedures with parameters, and other features (but not so many as to contradict Clarke's famous theorem from the fourth subsection above).

The language, (operational) semantics and proof system below are closely related to those in Ch.5 of [**deB**]. He goes further in later chapters, to have both recursive programs and parameters, but avoids having nested procedures, even in Ch.5. We allow them, as only later chapters of [**deB**] apparently need to avoid nesting, or indeed really need to use denotational semantics, on which many words are devoted there. See also the comments near the end of this addendum. All the details for the soundness and adequacy of the system here have been checked, as we did in our third section, but it's quite lengthy, when done in enough detail to inspire confidence. This paper is getting far too verbose, so we'll not include the details here, risking

a crisis of confidence, not a novelty in this subject.

Actually, if we had followed Harel's wise comments quoted just below, this is where all the details would have been included, rather than in the tortuous 3rd subsection above!

> There is seemingly a drawback to our treatment in the fact that we do not provide tools for including any kinds of parameters in the programming language. The reason is our wanting to achieve a clarification of the mechanisms for reasoning about *pure* recursion. Our experience in digesting the literature on this subject indicates that in most of the cases the presentation of basic principles suffers from being obscured by rules for dealing with the parameters.
>
> David Harel [**FODL**], p. 44

The language RTEN.

As before, the set of procedure identifiers, disjoint from the variables, will have members with names like p, q, etc. And, quite similar to the beginning of Section 3 above, the set $DEC \cup COM$ of declarations and commands is defined by mutual structural induction as follows :

$$D, D_1, \cdots, D_k \in DEC \ := \ \{ \ - \ || \ [p : C] \ \}$$

$$C, C_1, \cdots, C_n \in COM \ := \ \{ \ x \Leftarrow t \mid \mathsf{call}\ p \ || \ \mathsf{ite}(H)(C_1)(C_2) \mid \mathsf{begin}\ D_1; \cdots; D_k \ ; \ C_1; \cdots; C_n \ \mathsf{end} \ \}$$

Intuitively, the declaration $[p : C]$ is just saying "the procedure to be associated with the identifier p is to be C ". In the block-command just above, if D_i is $[p_i : K_i]$, we require the identifiers $p_1, \cdots, p_k$ to be distinct. As before, the case $k = 0 = n$ gives the 'skip' or 'null' or 'do-nothing' command, namely $\mathsf{begin\ end}$, which really is a third atomic command. We have resurrected the **ite**-command (if-then-else) from **BTEN**, so H above is any quantifier-free formula from the underlying fixed $1^{\underline{\text{st}}}$order language $\mathcal{L}$ (with equality). But we have omitted the **whdo**-command, to shorten the treatment, with justification as follows :

Exercise. Using the semantics defined just below, show that the meaning of (i.e. the SSQ-sequence of)

$$\mathsf{begin}\ [p : \mathsf{ite}(H)(C; \mathsf{call}\ p)(\mathsf{begin\ end})] \ ; \ \mathsf{call}\ p \ \mathsf{end}$$

(a block-command with $k = 1 = n$) is the same as the SSQ-sequence of $\mathsf{whdo}(H)(C)$, as given in Section 3.

After reading in **[LM]**, Sect.IV-11: McSelfish Calculations how McCarthy's McSELF-language gives a completely general definition of *computability* without needing whdo, this exercise should come as no surprise. The command in the display above is a typical example of recursive programming, at least for the case where only *one* procedure identifier is involved. Mutual recursion with several declared procedures can also happen in interesting ways, of course.

Semantics of RTEN.

Here the extra semantics components s and δ are the same as in the third section; whereas π will be much simpler :

$$PRIDE \supset \text{ finite set } \xrightarrow{\pi} COM \quad .$$

We shall be writing strings C/π , where π is now more of a syntactic component, which could be identified with a string

$$[p_1 : K_1]\ [p_2 : K_2]\ \cdots [p_\ell : K_\ell]\ ,$$

where

$$\text{domain}(\pi) \ = \ \{p_1 \prec p_2 \prec \cdots \prec p_\ell\}$$

with respect to some fixed linear order "$\prec$" on $PRIDE$, and where $\pi(p_i) = K_i$ for $1 \leq i \leq \ell$.

When it is defined, the semantic object $SSQ(C/\pi, s|\delta)$ will be a (possibly infinite) sequence of states s—no need to include the "$|\delta$-half", as in Subsection 3 above). The definition will have six cases : the three types of atomic commands, the ite-command, and the two cases ($k = 0 < n$) and $k > 0,\ n \geq 0$) of the block-command.

$\underline{C = x \div t}$ or $\underline{\mathsf{begin\ end}}$ or $\underline{\mathsf{begin}\ C_*\ \mathsf{end.}}$ Defined the same as in Subsection 3, dropping all the $|\delta$'s.

$\underline{C = \mathsf{call}\ p.}$ The sequence $SSQ(\mathsf{call}\ p/\pi, s|\delta)$ is undefined unless p is in domain(π), in which case, with $\pi(p) = K$, it is

$$\prec\ s\ ,\ SSQ(K/\pi, s|\delta)\ \succ$$

(which might also be undefined, of course).

Remarks. Note that, just as in the third section, the definition, which we are halfway through giving, proceeds to define the nth term, SSQ_n, of the sequence by induction on n, and, for fixed n, by structural induction on C. The reader might prefer to re-write it in the stricter style we adopted at first in Section 3. Quite baffling to me are the sentence in **[deB]** on p. 150 :

> Note that, strictly speaking, the definition of $Comp(< E \mid P >)$ is not fully rigorous $\cdots$ it $\cdots$ may be made more precise (cf. the remark following Theorem A.27 of the appendix $\cdots$

and that remark :

> The non-trivial step in this approach to the definition of $Comp$ is the use of the recursion theorem $\cdots$

Note that (essentially) "$Comp$" is our SSQ, "E" is π, "P" is C, and "the recursion theorem" is one of Kleene's famous results from the theory of recursive functions (see **[CM]**, IV-7.2). Although we haven't bothered with so-called 'array variables', AKA 'subscripted variables', which are irrelevant here, by not forbidding nested procedures, we have a more general command language here. But the need for machinery from recursive function theory to define operational semantics seems out of the question. In any case, a definition is either rigorous or not; being "*strictly speaking*" not "*fully* rigorous" is an odd turn of phrase!

So let us complete the definition of SSQ.

$\underline{C = \text{an ite-command.}}$ Define

$$SSQ(\mathsf{ite}(H)(C_1)(C_2)/\pi, s|\delta) := \begin{cases} \text{undefined} & \text{if } \mathrm{free}(H) \not\subset \mathrm{dom}(\delta); \\ \prec s, SSQ(C_1/\pi, s|\delta) \succ & \text{if } H \text{ is true at } s \circ \delta; \\ \prec s, SSQ(C_2/\pi, s|\delta) \succ & \text{if } H \text{ is false at } s \circ \delta. \end{cases}$$

$\underline{C = \mathsf{begin}\ [p : K]; D_*; C_*\ \mathsf{end}\ .}$ Define

$$SSQ(C/\pi, s|\delta) \ := \ \prec \ s \ , \ SSQ(\mathsf{begin}\ D_*; C_*\ \mathsf{end}/\pi', s|\delta) \ \succ \ ,$$

where π' agrees with π, except that $\pi'(p) = K$, whatever the status of p with respect to π.

The assertion language and proof system.

We shall adopt the approach of the previous addendum. But here, as in Section 3 as opposed to Sections 1 and 2 and the last addendum, the basic F-H statement has the form $F\{C/\pi\}G$. That is, it includes π, as well as formulas F and G in $\mathcal{L}$ and a command C from the language **RTEN**.

So this language of assertions will start with atomic strings $[F : C/\pi : G]$ for F, G in the underlying $1^{\underline{\text{st}}}$order language; C from **RTEN**. Then it builds up compound assertions using

$$\neg\mathcal{A} \quad , \quad (\mathcal{A}\&\mathcal{B}) \quad , \quad \forall x\mathcal{A} \quad \text{and} \quad [\mathcal{A} : C/\pi : \mathcal{B}].$$

As before, the F-H statement $F\{C/\pi\}G$ is the (universal) closure of the assertion $[F : C/\pi : G]$.

We are using $\mathcal{A}$, $\mathcal{A}'$, $\mathcal{A}_1$, $\mathcal{B}$ etc. as names for assertions. And let $< F >$ be an abbreviation for $0 \approx 0\{\textbf{begin end}/\pi\}F$. Alternatively, $< F >$ can be 'F_1', where F_1 is any (universal) closure for F.

The proof system

IN THE EARLIER VERSION ON THIS WEB PAGE HAD SEVERAL DRAWBACKS which I realized after a lot of work on a dynamic logic version of a pure recursion language. So the reader should go to numbers 14, 15 and 16 on this web page for a great deal of detail on this, and should wait breathlessly while I try to improve what had been here, based on that experience.

Besides the proofs of soundness and adequacy, we have also left the reader with the preliminary job of giving careful definitions of the various sets of free variables, then of the substitution notation, both for variables and for procedure identifiers. As indicated via the expressiveness hypothesis in the theorem, one will also need to use a $1^{\underline{\text{st}}}$order definability result via Gödel numbering for the 'post relation/strongest postcondition' for **RTEN**, or alternatively, a 'weakest precondition' version.

The papers of Olderog **[Old1], [Old2]** give what seems to be definitive positive results regarding complete F-H proof systems for imperative languages with procedures, allowing nesting, recursion, parameters, free variables and so-called sharing. Note that he doesn't bother with a full propositional language of F-H assertions, but most of his rules begin $\mathcal{H} \Longrightarrow \cdots$

on both top and bottom. Going to a full propositional presentation, writing some rules as axioms, and using the propositional tautology

$$(\mathcal{A} \Longrightarrow \mathcal{B}) \implies [(\mathcal{H} \Longrightarrow \mathcal{A}) \Longrightarrow (\mathcal{H} \Longrightarrow \mathcal{B})] \ ,$$

these would simplify somewhat, but it's all a matter of taste.

Trakhtenbrot, Halpern and Meyer in **[Clarke-Kozen eds]** have some disagreement with the use of 'copy rules' in the above papers (rather than denotational semantics) for defining the semantics of the language. However I haven't succeeded in finding a detailed treatment of their approach.

References

[**Apt**] Apt, K.R., *Ten Years of Hoare's Logic: A Survey—Part 1.* ACM Trans. Prog. Lang. Syst. 3,4, Oct. 1981, 431-483.

[**deB**] deBakker, J.W., *Mathematical Theory of Program Correctness.* Prentice Hall, N.J., 1980.

[**Cl1**] Clarke, E.M. Jr., *Programming Language Constructs for which it is Impossible to Obtain Good Hoare-like Axioms.* J. ACM 26, 1979, 129-147.

[**Clarke-Kozen eds**] *Logics of Programs.* Lecture Notes in CS # 164, Springer, 1984.

[**C**] Cook, Stephen A., *Soundness and Completeness of an Axiom System for Program Verification.* SIAM. J. COMPUT. (7), 1978, 70-90.

[**C+**] Cook, Stephen A., *Corrigendum : etc.* SIAM. J. COMPUT. 10(3), 1981, 612.

[**Harel**] Harel, David, *First-Order Dynamic Logic.* Lecture Notes in CS # 68, Springer, 1979.

[**G**] Gordon, Michael J.C. *Programming Language Theory and its Implementation.* Prentice Hall, London, 1988.

[**CM**] Hoffman, P. *Computability for the Mathematical.* this website, 2005.

[**Hoare-Shepherdson eds**] *Mathematical Logic and Programming Languages.* Prentice Hall, N.J., 1985.

[**Kozen-ed.**] *Logics of Programs.* Lecture Notes in CS # 131, Springer, 1982.

[**LM**] Hoffman, P. *Logic for the Mathematical.* this website, 2003.

[**Old1**] Olderog, E-R. *A Characterization of Hoare-like Calculi Based on Copy Rules.* Acta Inf. 16, 1981, 161-197.

[**Old2**] Olderog, E-R. *Sound and Complete Hoare's Logic For Programs with Pascal-like Procedures.* Proc. 15th ACM Symp, Theory of Computing, 1983, 320-329.