Depth-Measure Semantics and Partial-Correctness Rules for Imperative Recursion

Peter Hoffman

Pure Mathematics, University of Waterloo

Abstact. *Please read the 5-page introduction.*

Much of this paper is expository rather than genuinely original. It comes from two or three different directions.

It is partly a follow-up to the purely pedagogical [**sem**], to point out how one may evade both fixed point considerations and syntactic unfoldings, and define the semantics of (parameterless) recursive commands using only a formalization of the idea of recursion depth. And then, motivated by that formalization, one can see how to 'totally unfold' a command to give an unfolding version of the semantics which works completely generally. The possible novelty here is that we use a depth measure which can vary from one recursion to another within a given command.

Heretofore, for the case of so-called *non-simple* nested recursion, I myself had not been able to find a *definition* of the semantics using simplistic unfolding, though this is closely related to the main result in [**Musk**], which has a very different, rather sophisticated, point of view. Also it seemed clear, even if it's missing from the literature, that the much more subtle work of Olderog [**Old**] (where the hornet's nest of interactions between recursive procedures with parameters, and declarations of 'new' (or 'local') variables, is studied to get more-or-less definitive completeness results for Floyd-Hoare partial correctness systems) could be specialized to the parameterless case, though the syntactic categories needed might appear somewhat excessive for that "toy" (parameterless) case. So we have done that in part II of the final section of this paper.

Neither the depth measure nor the total unfolding seem to be written up in the CS literature for any reasonably general form of this austere imperative recursion. Functional recursion of course has a huge literature, which perhaps implicitly contains, or can be translated into, what we've done here, for those to whom that literature is readable. Let me know about this if you can do the translation and know what to translate.

From another direction, the usual 'Floyd-Hoare' rule (essentially due to Dana Scott), for partial correctness of recursion, takes several forms. Few of these seem to have been given much in the way of a validity proof which is straightforwardly public (contrast those machine-proof descriptions from the school of ISABEL/HOLology [**Nip**] [**Sch**] [**vOh**] [**Kmn**]), nor really convincingly detailed (contrast that in [**Apt**]), the 'big machinery' approaches in [**Musk**] and the exhausting if not exhaustive [**deBa**] being perhaps exceptions. And again, the intricacies in [**Old**] (surrounding the copy-rule semantics used for languages much closer to being practical Algol-like programming languages) certainly include a sound soundness proof (or perhaps a valid validity proof!), as discussed in part II of the last section below.

So this validity question is elucidated here in the context of a very basic command language which allows nested recursion without simpleness restrictions. (We avoid getting into *mutual* recursion. Past experience does indicate that messiness, rather than the need for new ideas, will likely be the only obstruction to extending this work to cover the mutual case. But, having whined about cavalier claims of this sort in the past, I won't make any myself here.) By defining a precise version of semantics for 'recursion to depth $\leq n$', a second proof arises of the validity of the rule in [**sing**] for simple nested recursion, the new proof actually having no need for simpleness, so producing a result which is new to me. But by comparing to rules from elsewhere, and giving a bit of slack re proofs there, one expects that it is a new result only to the extent of the newness of my rule, so essentially only a new proof. And that may also be debatable: the work was stimulated by reading [**Nip**], where no nesting at all of recursion is allowed. The indicated proof there of the recursion rule's validity is not written out for human consumption, and seemed initially to a neophyte like me to be too austere to possibly work, not involving any mention of fixed points nor of syntactic unfolding. However, cleverly enough, just a small analysis of recursion depth is really all those people need to get that proof. Stealing the idea, whose provenance is unknown to me, our 'new' proof here has no particularly new idea, just using that basic trick of getting explicitly into recursion depth. However, because nested recursion is allowed here (more generally, calls are local, not necessarily global), our treatment of depth is more complicated than that in [**Nip**]. And depth analysis becomes an order of magnitude even more involved as a result of allowing **non-simpleness** [e.g. *one can have a recursive command on some variable, the body of which has a recursive subcommand on a different variable, the latter of which has a*

body which includes a free occurrence of (i.e. a call to) the earlier variable].

In Section 1, we recall the syntactic definition of $\mathcal{R}ec_{1+}$ from [**sem**]. That set of strings is the simplest deterministic Turing equivalent imperative language which includes recursion on one procedure variable, and for which no "non-nesting" nor "simpleness" restrictions are imposed. At the same time, a sublanguage important for Section 3, and called $\mathcal{R}ec_{1.01}$, is delineated. It consists of those commands where the 'bound' procedure variables have all been 're-named' so as to be distinct from one another. Thus, no computing power has been lost, just elegance. (Interestingly, a good deal of the complexities in [**Old**] revolve around the need for re-naming identifiers because of the ubiquity of substitution operations in 'doing unfoldings'.) Then, back in the more general $\mathcal{R}ec_{1+}$, we use the notational conventions from [**sem**], and define the 'depth $\underline{n}$ semantics'. Finally, a series of totally elementary proofs are given of its most basic properties, leading up to showing that, when $\underline{n} = \underline{\infty}$, the usual fixpoint semantics is recovered. The notation $\underline{n}$ refers to a whole bunch of natural numbers, used as depth bounds for various recursions within a command.

In Section 2, we give the total unfolding definition which parallels the depth measure above. The overall semantics of the $\underline{n}$th unfolding of a command C is easily seen to coincide with the $\underline{n}$-depth approximation semantics for C itself. Then it follows quickly that these unfoldings are all recursion-free commands, the union of whose semantic (i.e. input-output pairs of states) sets is the semantic set for the given command being unfolded.

In Section 3, we first discuss substitution for free occurrences of a procedure variable, say X. The usual complication (caused by *not wanting free occurrences of any procedure variable in the command which is to be substituted for X to become bound after the substitution*) has its usual solution via **re-naming bound variables**. Since a decent theory here seems to require a separate depth measure for each procedure variable, this messes things up a bit unless all the bound variables are distinct, so we stick to $\mathcal{R}ec_{1.01}$ here. And we make claims, concerning a command produced by substitution, only for those $\underline{n}$ taking the same value on a variable and all its re-namings; that is, all these must have the same depth bound. Since a separate depth bound seems most natural not just for each variable which occurs bound but actually for each separate recursion within the command, the use of the sublanguage $\mathcal{R}ec_{1.01}$ now has a second motivation. Then we recall the partial correctness rule for recursion from [**sing**], and soup it up to deal with the

possibility of non-simpleness. Finally, the proof of its validity now follows from a notion of '$\underline{n}$-truth', where $\underline{n}$ continues to measure recursion depth. The only tedium in the proof is going through all the more innocuous rules to check that a certain relation from [**sing**], called $\vdash'$, preserves $\underline{n}$-truth. One particular of the axioms here makes it clear why one of the issues discussed in the final section has relevance, as we describe in the final paragraph below of this introduction.

The various forms over the past 40 years of the partial correctness recursion rule involve an antecedent which is not a 'Floyd-Hoare assertion' (nor a statement in dynamic logic if that's the context), but rather an assertion of the existence of a derivation. (Harel's idiosyncratic system, [**Harel**], pp. 44-55, is an exception here.) Apt and others have called these "meta-rules", and a good deal of fuss with ad-hoc devices, especially conversion to Gentzen form, is always provided to *make sense* of soundness for these systems, much less being apparently needed for attempted *proofs* of soundness. Except for my rule in [**sub**], all these rules are 'self-referential', in that the above antecedent *uses* a conditional $\vdash$-derivation to formulate a rule which is one from the set of rules which *define* $\vdash$. Our rule uses a previously defined weaker system for the $\vdash'$ mentioned above. So I have for some time felt the need to compare and contrast the various rules, which is what Section 4 here finally does.

Sections 2, 3 and 4 are not far from being independent of each other.

Only one last point in this introduction : in $1^{\underline{\text{st}}}$order logic, there is a sometimes under-appreciated technical point concerning what might be termed 'Endertonian [**End**] versus Mendelsonian [**Men**]'. The question is whether derivations should relate semantically to 'truth at a state, i.e. truth of formulas at an assignment of variables', or to 'truth in an interpretation'. Because one is ultimately interested in *sentences* (no free variables), both types of system accomplish the same thing, a sound complete system for sentences, though usually 'passing through' formulas more generally. When dealing with recursion, analogous questions arise with respect to *procedure* variables (not merely the $1^{\underline{\text{st}}}$order or 'program' variables). So, to finish the paper, we discuss that issue in more than the zero detail I've seen in the past. There is an interesting paper by a philosopher of linguistics, [**Musk**], where recursive commands are given their semantics by syntactic translation into higher order (actually $2^{\underline{\text{nd}}}$order only needed) type-logic. We discuss the meaning and

status of validity/soundness in Muskens' sense, particularly for his "Scott rule" for recursion. This distinction concerning assignments (of procedure variables now) comes to the fore in trying to relate, to material elsewhere, both Muskens' and also deBakker's [**deBa**] treatments concerning rules for partial correctness of recursive commands. (Once again ignoring the literature on recursions with parameters, of which [**Old**] seems the most reliable) the previous two are the only ones of which I am aware that have the virtue of attempting to treat nested recursion in a direct way, even if the semantics in both have excruciatingly complicated formulations, and neither attempts, for nested recursion, to treat (Cook-)completeness of the systems, nor to discuss total correctness. (To be strictly correct further up, I should add that, in [**deBa**] , none of the various tortuous versions of the partial correctness recursion rule has $\vdash$ appearing in an antecedent. Rather they are all already translated into the ad-hoc "correctness phrases", where the symbol $\Longrightarrow$ is introduced in a peculiar way which studiously prevents one from stretching one's 'logical muscles' and actually having the flexibility of using all propositional connectives in the language. By the time it has all been defined and re-defined twice, the semantics takes 21 pages to define even at the stage where declarations are fixed and global, disallowing any nesting at all, and then a further 14 pages with nesting, but not followed by any analysis of completeness. I may be missing something, but I've tended to assume that the reluctance to introduce negation and get all the propositional connectives into the language is related to the desire to avoid having to treat **total** correctness, even implicitly. Readers are requested to sort me out on that if there is some other compelling reason.) So a third main motivation for this paper is to give a straightforward as possible discussion of this technicality for parameterless recursion, relating it to (1) the use of depth-measure alone for the semantics; (2) the validity and comparative strength of various rules for partial correctness of recursive commands; and (3) the relatively uncomplicated discussion needed when parameters don't occur.

1. **Depth-measure semantics.**

See [**sem**] for all the basic notation. For a command C, we shall frequently refer to its sets, $fr(C)$, of *free procedure variables*, and $bd(C)$, of *bound procedure variables*. So we now give the definition by structural induction of triples of objects $[C, fr(C), bd(C)]$ defining the triple

$$[\ \mathcal{R}ec_{1+}\ ,\ fr : \mathcal{R}ec_{1+} \to 2^{\mathcal{P}cv}\ ,\ bd : \mathcal{R}ec_{1+} \to 2^{\mathcal{P}cv}\ ].$$

(1) Assignment : $[\ y \leftarrow\!\!\div t\ \ ,\ \emptyset\ ,\ \emptyset\ ]$

(2) Call : $[\ callX\ ,\ \{X\}\ ,\ \emptyset\ ]$

(3) Sequencing : $[\ (C';C'')\ ,\ fr(C') \cup fr(C'')\ ,\ bd(C') \cup bd(C'')\ ]$

(4) if-then-else : $[\ ite(H)(C')(C'')\ ,\ fr(C') \cup fr(C'')\ ,\ bd(C') \cup bd(C'')\ ]$

(5) Recursive : $[\ \nabla X D\ ,\ fr(D) \setminus \{X\}\ ,\ bd(D) \cup \{X\}\ ]$

The sublanguage $\mathcal{R}ec_{1.01}$ (ignored till the 3rd section) is defined in the same way, with the added side conditions :
in (3) and (4), that $bd(C') \cap bd(C'') = \emptyset$;
in (5), that $X \notin bd(D)$.
Rather trivial induction on $C \in \mathcal{R}ec_{1+}$ shows that

$C \in \mathcal{R}ec_{1.01} \iff$

for all X, the consecutive pair ∇X appears at most once in the string C .

Of course, first one will have shown

$$X \in bd(C) \iff \text{the pair } \nabla X \text{ appears in the string } C\ .$$

'Clearly' it is also true that

$X \in fr(C) \iff$

the substring $callX$ appears in the string C , but not within the scope of a ∇X ;

but that requires a definition of *scope*, which isn't needed in the paper, so will be left as an exercise. Note that, for X to be a fully fledged member of $bd(C)$, the procedure variable X need not be actually "called" in any

C-subcommand $\nabla X D$, just that such a subcommand should exist, with D possibly being rather uninteresting in that it contains no 'free' $callX$.

Now let $\underline{n}$ denote a function $\mathcal{P}cv \to \mathbf{N} \cup \{\infty\}$. **Intuitively**, this will be used to measure recursion depth, as follows. An '(input,output) pair' (s, s') of states will be in the set $\mathcal{M}_{\underline{n}}(C, \Theta)$ when and only when, implementing the command C with the machine initially in state s *can* (actually, *will*, by determinisiticism), assuming the computation converges, produce the state s', where the implementation 'allows' , for all X and each subcommand $\nabla X D$ of C, the latter recursion to 'cycle' at most "$\underline{n}(X)$" times [and where, going back to the profligate discussion in [**sem**], the component Θ gives the 'phantom semantics' for ("is the assignment of") the free occurrences of procedure variables in C]. But this 'loosey-goosey' description plays no role whatsoever in the mathematics that follows, only in its motivation. The phrase "recursion depth" can be seen in plenty of loose descriptions, but seems to be more subtle in its possibilities (when dealing with nested, non-simple recursion) than it gets credit for. We'll be careful to verify everything strictly from the inductive definition three paragraphs below.

The really crucial part of that definition is the middle case of (5) below. It tells us what recursion depth really means here, whatever else one might have thought. There are several other possibilities for that definition, and none that I know will do what's needed in Section 3 of this paper, though several (e.g. making the depth bound fixed for all variables) can be used to give results analogous to the rest of this section.

Recall that Θ denotes a function from the countable set $\mathcal{P}cv$ of procedure variables to the set $\mathcal{B}sm$ of "basic semantic values", which is $2^{\mathcal{S}te \times \mathcal{S}te}$, the set of subsets of pairs of states. Also $\Theta_{X \mapsto \beta}$ agrees with Θ except for mapping X to β. Similarly $s_{y \mapsto k}$ differs from the state s on only one variable, a $1^{\underline{\text{st}}}$order variable y, and for natural numbers k (which form the only interpretation considered in this paper).

Definition. By induction on C, the five clauses just below define $\mathcal{M}_{\underline{n}}(C,\Theta)$:

(1) $\mathcal{M}_{\underline{n}}(y \doteqdot t,\Theta) \;:=\; \{(s, s_{y\mapsto s(t)}) \mid s \in \mathcal{S}te\}$ (independent of Θ and $\underline{n}$) .

(2) $\mathcal{M}_{\underline{n}}(callX,\Theta) \;:=\; \Theta(X)$ (independent of $\underline{n}$) .

(3) $\mathcal{M}_{\underline{n}}((C;D),\Theta) \;:=\; \mathcal{M}_{\underline{n}}(D,\Theta) \circ \mathcal{M}_{\underline{n}}(C,\Theta)$.

(4) $\mathcal{M}_{\underline{n}}(ite(H)(C)(D),\Theta) \;:=\; \{(s,s') \mid H\mathbf{tt}@s$ and $(s,s') \in \mathcal{M}_{\underline{n}}(C,\Theta)$; or $H\mathbf{ff}@s$ and $(s,s') \in \mathcal{M}_{\underline{n}}(D,\Theta)\}$.

(5) $\mathcal{M}_{\underline{n}}(\nabla XD,\Theta)$ is defined by (minimally transfinite) induction on $\underline{n}(X)$ via :

<u>if $\underline{n}(X) = 0$</u>, it is $\emptyset$;

<u>if $0 < \underline{n}(X) < \infty$</u>, it is $\mathcal{M}_{\underline{n}_{X-}}(D, \Theta_{X\mapsto \mathcal{M}_{\underline{n}_{X-}}(\nabla XD,\Theta)})$,

where we let $\underline{n}_{X-}$ agree with $\underline{n}$ except that $\underline{n}_{X-}(X) = \underline{n}(X) - 1$;

<u>if $\underline{n}(X) = \infty$</u>, it is $\bigcup_{\underline{\ell}}\{\, \mathcal{M}_{\underline{\ell}}(\nabla XD,\Theta) \mid \underline{\ell}$ agrees with $\underline{n}$, except $\underline{\ell}(X) < \infty \,\}$.

The first six of the following seven results are proved by induction on C; and the final case, $C = \nabla XD$, of those inductions is always proved by induction on $\underline{n}(X)$ [as might be expected from the definition above]. These proofs need no other ideas, so there's a lot of deja-vu going on. **Note that $\subset$ means *non-strict* inclusion here.** We'll use without mention elementary results from early in **[sem]** in several proofs below, for example that composition of relations preserves inclusions. The proofs, despite their pedestrianity, are given in considerable, perhaps excruciating, detail, this subject having historically suffered from more than its fair share of errors, including one claiming the soundness of an unsound Floyd-Hoare proof system for total correctness of recursive programming, in a paper referred to over 10 years in at least 100 journal articles before the error was discovered. (It has since occurred in not less than 1,000 bibliographies without any warning concerning the unsoundness of the total correctness system.)

Proposition 1.1 *If $\theta \sqsubset \Psi$* [meaning that, for all X, $\Theta(X) \subset \Psi(X)$], *then, for all $\underline{n}$ and C, we have*

$$\mathcal{M}_{\underline{n}}(C,\Theta) \;\subset\; \mathcal{M}_{\underline{n}}(C,\Psi) \;.$$

Proof. The required inclusion becomes $\Theta(X) \subset \Psi(X)$ when C is $callX$, and we get equality when C is an assignment command. When $C = (D; E)$,

$$\mathcal{M}_{\underline{n}}((D;E),\Theta) \;=\; \mathcal{M}_{\underline{n}}(E,\Theta)\circ\mathcal{M}_{\underline{n}}(D,\Theta) \;\subset\; \mathcal{M}_{\underline{n}}(E,\Psi)\circ\mathcal{M}_{\underline{n}}(D,\Psi) \;=\; \mathcal{M}_{\underline{n}}((D;E),\Psi)\,.$$

For $C = ite(H)(D)(E)$, supposing that $(s,s') \in \mathcal{M}_{\underline{n}}(ite(H)(D)(E),\Theta)$, we see that
if $H\mathbf{tt}@s$, then $(s,s') \in \mathcal{M}_{\underline{n}}(D,\Theta) \subset \mathcal{M}_{\underline{n}}(D,\Psi)$,
so $(s,s') \in \mathcal{M}_{\underline{n}}(ite(H)(D)(E),\Psi)$;
whereas if $H\mathbf{ff}@s$, then $(s,s') \in \mathcal{M}_{\underline{n}}(E,\Theta) \subset \mathcal{M}_{\underline{n}}(E,\Psi)$,
so $(s,s') \in \mathcal{M}_{\underline{n}}(ite(H)(D)(E),\Psi)$.

To finish the structural induction and the proof, for $C = \nabla XD$ we proceed by induction on $\underline{n}(X)$. When 0, both sides are $\emptyset$. When $0 < \underline{n}(X) < \infty$,

$$\begin{aligned}\mathcal{M}_{\underline{n}}(C,\Theta) \;&=\; \mathcal{M}_{\underline{n}_{X-}}(D,\Theta_{X\mapsto\mathcal{M}_{\underline{n}_{X-}}(\nabla XD,\Theta)})\\ &\subset\; \mathcal{M}_{\underline{n}_{X-}}(D,\Theta_{X\mapsto\mathcal{M}_{\underline{n}_{X-}}(\nabla XD,\Psi)})\\ &\subset\; \mathcal{M}_{\underline{n}_{X-}}(D,\Psi_{X\mapsto\mathcal{M}_{\underline{n}_{X-}}(\nabla XD,\Psi)}) \;=\; \mathcal{M}_{\underline{n}}(C,\Psi)\,.\end{aligned}$$

The second inclusion uses only the structural induction, but the first uses both it and the induction on $\underline{n}(X)$. Finally, when $\underline{n}(X) = \infty$,

$$\begin{aligned}\mathcal{M}_{\underline{n}}(\nabla XD,\Theta) \;&=\; \bigcup\{\, \mathcal{M}_{\underline{\ell}}(\nabla XD,\Theta) \mid \underline{\ell} \text{ agrees with } \underline{n}, \text{ except } \underline{\ell}(X) < \infty \,\}\\ &\subset\; \bigcup\{\, \mathcal{M}_{\underline{\ell}}(\nabla XD,\Psi) \mid \text{ as above } \,\} \;=\; \mathcal{M}_{\underline{n}}(\nabla XD,\Psi)\,.\end{aligned}$$

So the proof is complete.

Proposition 1.2 *If $\underline{n} \sqsubset \underline{m}$* [meaning that, for all X, $\underline{n}(X) \leq \underline{m}(X)$], *then, for all Θ and C, we have*

$$\mathcal{M}_{\underline{n}}(C,\Theta) \;\subset\; \mathcal{M}_{\underline{m}}(C,\Theta)\,.$$

Proof. Here the required inclusion becomes equality when C is either atomic command, where $\underline{n}$ is irrelevant. When $C = (D; E)$,

$$\mathcal{M}_{\underline{n}}((D;E),\Theta) \;=\; \mathcal{M}_{\underline{n}}(E,\Theta)\circ\mathcal{M}_{\underline{n}}(D,\Theta) \;\subset\; \mathcal{M}_{\underline{m}}(E,\Theta)\circ\mathcal{M}_{\underline{m}}(D,\Theta) \;=\; \mathcal{M}_{\underline{m}}((D;E),\Theta)\,.$$

For $C = ite(H)(D)(E)$, supposing that $(s,s') \in \mathcal{M}_{\underline{n}}(ite(H)(D)(E),\Theta)$, we see that
if $H\mathbf{tt}@s$, then $(s,s') \in \mathcal{M}_{\underline{n}}(D,\Theta) \subset \mathcal{M}_{\underline{m}}(D,\Theta)$,
so $(s,s') \in \mathcal{M}_{\underline{m}}(ite(H)(D)(E),\Theta)$;
whereas if $H\mathbf{ff}@s$, then $(s,s') \in \mathcal{M}_{\underline{n}}(E,\Theta) \subset \mathcal{M}_{\underline{m}}(E,\Theta)$,
so $(s,s') \in \mathcal{M}_{\underline{m}}(ite(H)(D)(E),\Theta)$.

To finish the structural induction and the proof, for $C = \nabla XD$ we proceed by induction on $\underline{n}(X)$. When 0, the left side is $\emptyset$. When $0 < \underline{n}(X) < \infty$,

$$\begin{aligned}\mathcal{M}_{\underline{n}}(C,\Theta) \;&=\; \mathcal{M}_{\underline{n}_{X-}}(D,\Theta_{X\mapsto\mathcal{M}_{\underline{n}_{X-}}(\nabla XD,\Theta)}) \\ &\subset\; \mathcal{M}_{\underline{n}_{X-}}(D,\Theta_{X\mapsto\mathcal{M}_{\underline{m}_{X-}}(\nabla XD,\Theta)}) \\ &\subset\; \mathcal{M}_{\underline{m}_{X-}}(D,\Theta_{X\mapsto\mathcal{M}_{\underline{m}_{X-}}(\nabla XD,\Theta)}) \;=\; \mathcal{M}_{\underline{m}}(C,\Theta)\,.\end{aligned}$$

The second inclusion uses the structural induction, and the first uses both **1.1** and the induction on $\underline{n}(X)$. Finally, when $\underline{n}(X) = \infty$ [and so is $\underline{m}(X)$],

$$\begin{aligned}\mathcal{M}_{\underline{n}}(\nabla XD,\Theta) \;&=\; \bigcup\{\, \mathcal{M}_{\underline{\ell}}(\nabla XD,\Theta) \mid \underline{\ell} \text{ agrees with } \underline{n},\ \text{except } \underline{\ell}(X) < \infty \,\} \\ &\subset\; \bigcup\{\, \mathcal{M}_{\underline{k}}(\nabla XD,\Theta) \mid \underline{k} \text{ agrees with } \underline{m},\ \text{except } \underline{k}(X) < \infty \,\} \;=\; \mathcal{M}_{\underline{m}}(\nabla XD,\Theta)\,.\end{aligned}$$

Proposition 1.3 *If C has no free occurrences of X* [i.e. $X \notin fr(C)$] *and $\Theta(Y) = \Psi(Y)$ for all $Y \neq X$, then we have*

$$\mathcal{M}_{\underline{n}}(C,\Theta) \;=\; \mathcal{M}_{\underline{n}}(C,\Psi)$$

for all $\underline{n}$.

It follows from an innocuous generalization of this (i.e. assume the sets $fr(C)$ and $\{X \mid \Theta(X) \neq \Psi(X)\}$ are disjoint) that Θ-dependence of $\mathcal{M}_{\underline{n}}(C,\Theta)$ is only on Θ's restriction to $fr(C)$.

Proof. The required equality becomes $\Theta(Y) = \Psi(Y)$ when C is *callY* for $Y \neq X$ (and C can't be *callX*!). We clearly get equality when C is an assignment command, where Θ and Ψ are irrelevant. When $C = (D;E)$,

$$\mathcal{M}_{\underline{n}}((D;E),\Theta) \;=\; \mathcal{M}_{\underline{n}}(E,\Theta)\circ\mathcal{M}_{\underline{n}}(D,\Theta) \;=\; \mathcal{M}_{\underline{n}}(E,\Psi)\circ\mathcal{M}_{\underline{n}}(D,\Psi) \;=\; \mathcal{M}_{\underline{n}}((D;E),\Psi)\,.$$

For $C = ite(H)(D)(E)$, supposing that $(s,s') \in \mathcal{M}_{\underline{n}}(ite(H)(D)(E),\Theta)$, we see that

if $H\mathbf{tt}@s$, then $(s,s') \in \mathcal{M}_{\underline{n}}(D,\Theta) = \mathcal{M}_{\underline{n}}(D,\Psi)$,
so $(s,s') \in \mathcal{M}_{\underline{n}}(ite(H)(D)(E),\Psi)$;

whereas if $H\mathbf{ff}@s$, then $(s,s') \in \mathcal{M}_{\underline{n}}(E,\Theta) = \mathcal{M}_{\underline{n}}(E,\Psi)$,
so $(s,s') \in \mathcal{M}_{\underline{n}}(ite(H)(D)(E),\Psi)$.

This gives inclusion one way, and the other way is immediate by the symmetry of the assumption concerning Θ and Ψ.

To finish the structural induction and the proof, for $C = \nabla Y D$ we proceed by induction on $\underline{n}(Y)$. When 0, both sides are $\emptyset$. When $0 < \underline{n}(Y) < \infty$,

$$\begin{aligned}\mathcal{M}_{\underline{n}}(C,\Theta) \;&=\; \mathcal{M}_{\underline{n}_{Y-}}(D,\Theta_{Y\mapsto\mathcal{M}_{\underline{n}_{Y-}}(\nabla Y D,\Theta)}) \\ &=\; \mathcal{M}_{\underline{n}_{Y-}}(D,\Theta_{Y\mapsto\mathcal{M}_{\underline{n}_{Y-}}(\nabla Y D,\Psi)}) \\ &=\; \mathcal{M}_{\underline{n}_{Y-}}(D,\Psi_{Y\mapsto\mathcal{M}_{\underline{n}_{Y-}}(\nabla Y D,\Psi)}) \;=\; \mathcal{M}_{\underline{n}}(C,\Psi)\,.\end{aligned}$$

The third equality uses the structural induction when $Y \neq X$ and is trivial when $Y = X$. The second equality uses the induction on $\underline{n}(Y)$. Finally, when $\underline{n}(Y) = \infty$,

$$\begin{aligned}\mathcal{M}_{\underline{n}}(\nabla Y D,\Theta) \;&=\; \bigcup\{\, \mathcal{M}_{\underline{\ell}}(\nabla Y D,\Theta) \mid \underline{\ell} \text{ agrees with } \underline{n}, \text{ except } \underline{\ell}(Y) < \infty \,\} \\ &=\; \bigcup\{\, \mathcal{M}_{\underline{\ell}}(\nabla Y D,\Psi) \mid \text{ as above } \,\} \;=\; \mathcal{M}_{\underline{n}}(\nabla Y D,\Psi)\,.\end{aligned}$$

So the proof is complete.

Proposition 1.4 *If C has no bound occurrences of X [i.e. $X \notin bd(C)$] and $\underline{n}(Y) = \underline{m}(Y)$ for all $Y \neq X$, then we have*

$$\mathcal{M}_{\underline{n}}(C,\Theta) \;=\; \mathcal{M}_{\underline{m}}(C,\Theta)$$

for all Θ.

An easy generalization (i.e. assume $bd(C) \cap \{X \mid \underline{m}(X) \neq \underline{n}(X)\} = \emptyset$) implies that, for each C, the dependence of $\mathcal{M}_{\underline{n}}(C,\Theta)$ on $\underline{n}$ is only on its restriction to $bd(C)$.

Proof. The result is obvious when C is either atomic command, where $\underline{n}$ is irrelevant. When $C = (D;E)$,

$$\mathcal{M}_{\underline{n}}((D;E),\Theta) \;=\; \mathcal{M}_{\underline{n}}(E,\Theta)\circ\mathcal{M}_{\underline{n}}(D,\Theta) \;=\; \mathcal{M}_{\underline{m}}(E,\Theta)\circ\mathcal{M}_{\underline{m}}(D,\Theta) \;=\; \mathcal{M}_{\underline{m}}((D;E),\Theta)\,.$$

For $C = ite(H)(D)(E)$, supposing that $(s,s') \in \mathcal{M}_{\underline{n}}(ite(H)(D)(E),\Theta)$, we see that
if $H\mathbf{tt}@s$, then $(s,s') \in \mathcal{M}_{\underline{n}}(D,\Theta) = \mathcal{M}_{\underline{m}}(D,\Psi)$,
so $(s,s') \in \mathcal{M}_{\underline{m}}(ite(H)(D)(E),\Theta)$;
whereas if $H\mathbf{ff}@s$, then $(s,s') \in \mathcal{M}_{\underline{n}}(E,\Theta) = \mathcal{M}_{\underline{m}}(E,\Theta)$,
so $(s,s') \in \mathcal{M}_{\underline{m}}(ite(H)(D)(E),\Theta)$.
This proves inclusion one way, and the other way is immediate from the symmetry of the assumption concerning $\underline{n}$ and $\underline{m}$.

To finish the structural induction and the proof, for $C = \nabla Y D$ we proceed by induction on $\underline{n}(Y)$. Note that $Y \neq X$, so $\underline{n}(Y) = \underline{m}(Y)$. When 0, both sides are $\emptyset$. When $0 < \underline{n}(Y) < \infty$,

$$\begin{aligned}\mathcal{M}_{\underline{n}}(C,\Theta) \;&=\; \mathcal{M}_{\underline{n}_{Y-}}(D,\Theta_{Y\mapsto\mathcal{M}_{\underline{n}_{Y-}}(\nabla Y D,\Theta)}) \\ &=\; \mathcal{M}_{\underline{n}_{Y-}}(D,\Theta_{Y\mapsto\mathcal{M}_{\underline{m}_{Y-}}(\nabla Y D,\Theta)}) \\ &=\; \mathcal{M}_{\underline{m}_{Y-}}(D,\Theta_{Y\mapsto\mathcal{M}_{\underline{m}_{Y-}}(\nabla Y D,\Theta)}) \;=\; \mathcal{M}_{\underline{m}}(C,\Theta)\;.\end{aligned}$$

The second equality uses the structural induction, and the first uses the induction on $\underline{n}(Y)$. Finally, when $\underline{n}(Y) = \infty$ (and so is $\underline{m}(Y)$),

$$\begin{aligned}\mathcal{M}_{\underline{n}}(\nabla Y D,\Theta) \;&=\; \bigcup\{\, \mathcal{M}_{\underline{\ell}}(\nabla Y D,\Theta) \mid \underline{\ell} \text{ agrees with } \underline{n},\ \text{except } \underline{\ell}(Y) < \infty \,\} \\ &=\; \bigcup\{\, \mathcal{M}_{\underline{k}}(\nabla Y D,\Theta) \mid \underline{k} \text{ agrees with } \underline{m},\ \text{except } \underline{k}(Y) < \infty \,\} \;=\; \mathcal{M}_{\underline{m}}(\nabla Y D,\Theta)\;.\end{aligned}$$

Proposition 1.5 *The function* $[\beta \mapsto \mathcal{M}_{\underline{n}}(C,\Theta_{Y\mapsto\beta}\,]\;:\;\mathcal{B}sm \to \mathcal{B}sm$ *is continuous, for each* $\underline{n}, C, \Theta$, *and* Y. *More generally, if* $\Theta_0 \sqsubset \Theta_1 \sqsubset \Theta_2 \sqsubset \cdots$, *then we have that, for any* $\underline{n}, C$ *and* Θ,

$$\mathcal{M}_{\underline{n}}(C,\bigsqcup_i \Theta_i) \;=\; \bigcup_i \mathcal{M}_{\underline{n}}(C,\Theta_i)\;.$$

By definition, $(\bigsqcup_i \Theta_i)(X) := \bigcup_i(\Theta_i(X))$.

Proof. That last definition proves it when $C = callX$. The result is clear for assignment commands, where $\mathcal{M}_{\underline{n}}(C,\Theta)$ is independent of Θ.

When $C = (C'; C'')$, we have

$$\mathcal{M}_{\underline{n}}((C'; C''), \bigsqcup_i \Theta_i) \;=\; \mathcal{M}_{\underline{n}}(C'', \bigsqcup_i \Theta_i) \circ \mathcal{M}_{\underline{n}}(C', \bigsqcup_i \Theta_i) \;=$$

$$\bigcup_i \mathcal{M}_{\underline{n}}(C'', \Theta_i) \circ \bigcup_i \mathcal{M}_{\underline{n}}(C', \Theta_i) \;=$$

$$\bigcup_i (\mathcal{M}_{\underline{n}}(C'', \Theta_i) \circ \mathcal{M}(C', \Theta_i)) \;=\; \bigcup_i \mathcal{M}_{\underline{n}}((C'; C''), \Theta_i) \; .$$

For $C = ite(H)(C')(C'')$,

$$(s, s') \in \mathcal{M}_{\underline{n}}(ite(H)(C')(C''), \bigsqcup_i \Theta_i) \iff$$

$$[H\mathbf{tt}@s \text{ and } (s, s') \in \mathcal{M}_{\underline{n}}(C', \bigsqcup_i \Theta_i) \quad \text{or} \quad H\mathbf{ff}@s \text{ and } (s, s') \in \mathcal{M}_{\underline{n}}(C'', \bigsqcup_i \Theta_i)] \iff$$

$$[H\mathbf{tt}@s \text{ and } (s, s') \in \bigcup_i \mathcal{M}_{\underline{n}}(C', \Theta_i) \quad \text{or} \quad H\mathbf{ff}@s \text{ and } (s, s') \in \bigcup_i \mathcal{M}_{\underline{n}}(C'', \Theta_i)] \iff$$

$$\exists i \, [H\mathbf{tt}@s \text{ and } (s, s') \in \mathcal{M}_{\underline{n}}(C', \Theta_i) \quad \text{or} \quad H\mathbf{ff}@s \text{ and } (s, s') \in \mathcal{M}_{\underline{n}}(C'', \Theta_i)] \iff$$

$$\exists i \, [(s, s') \in \mathcal{M}_{\underline{n}}(ite(H)(C')(C''), \Theta_i)] \; .$$

When $C = \nabla X D$, we proceed by induction on $\underline{n}(X)$. First note that

$$(\bigsqcup_i \Theta_i)_{X \mapsto \cup_i \beta_i} \;=\; \bigsqcup_i ((\Theta_i)_{X \mapsto \beta_i}) \; ,$$

since both map X to $\cup_i \beta_i$, and any other Y to $\cup_i \Theta_i(Y)$. Now when $\underline{n}(X)$ is 0, both sides of the identity to be proved are $\emptyset$. When $0 < \underline{n}(X) < \infty$, we get

$$\mathcal{M}_{\underline{n}}(\nabla X D, \bigsqcup_i \Theta_i) \;=\; \mathcal{M}_{\underline{n}_{X-}}(D, (\bigsqcup_i \Theta_i)_{X \mapsto \mathcal{M}_{\underline{n}_{X-}}(\nabla X D, \bigsqcup_i \Theta_i)})$$

$$= \; \mathcal{M}_{\underline{n}_{X-}}(D, (\bigsqcup_i \Theta_i)_{X \mapsto \cup_i \mathcal{M}_{\underline{n}_{X-}}(\nabla X D, \Theta_i)})$$

$$= \; \mathcal{M}_{\underline{n}_{X-}}(D, \bigsqcup_i (\Theta_i)_{X \mapsto \mathcal{M}_{\underline{n}_{X-}}(\nabla X D, \Theta_i)})$$

$$= \; \bigcup_i \mathcal{M}_{\underline{n}_{X-}}(D, (\Theta_i)_{X \mapsto \mathcal{M}_{\underline{n}_{X-}}(\nabla X D, \Theta_i)}) \;=\; \bigcup_i \mathcal{M}_{\underline{n}}(\nabla X D, \Theta_i) \; .$$

Successively, the equalities come from : definition; induction on $\underline{n}(X)$; the display just above this big one; induction on C; and definition, respectively. Note that the correct 'square-inclusions' hold for the penultimate equality, using **1.1**. Finally, for $\underline{n}(X) = \infty$, we get

$$\mathcal{M}_{\underline{n}}(\nabla XD, \bigsqcup_i \Theta_i) \;=\; \bigcup_{\underline{\ell}} \{ \mathcal{M}_{\underline{\ell}}(\nabla XD, \bigsqcup_i \Theta_i) \mid \underline{\ell} \text{ agrees with } \underline{n}, \text{ except } \underline{\ell}(X) < \infty \}$$

$$= \bigcup_{\underline{\ell}} \{ \bigcup_i \mathcal{M}_{\underline{\ell}}(\nabla XD, \Theta_i) \mid \underline{\ell} \text{ agrees with } \underline{n}, \text{ except } \underline{\ell}(X) < \infty \} \;=\; \bigcup_i \mathcal{M}_{\underline{n}}(\nabla XD, \Theta_i) \,,$$

where we unionize in the other order for the last step.

Proposition 1.6 *If $\underline{\infty}$ means $\underline{\infty}(X) = \infty$ for all X, then, for all C, we have*

$$\mathcal{M}_{\underline{\infty}}(C, \Theta) \;=\; \bigcup_{\underline{n}} \{ \mathcal{M}_{\underline{n}}(C, \Theta) \mid \underline{n}(X) < \infty \text{ for all } X \}) \,.$$

More generally, for any $\underline{k}$,

$$\mathcal{M}_{\underline{k}}(C, \Theta) \;=\; \bigcup_{\underline{n}} \{ \; \mathcal{M}_{\underline{n}}(C, \Theta) \mid \text{ for all } X, \; \underline{n}(X) < \infty \text{ and } \; \underline{n}(X) \leq \underline{k}(X) \; \} \,.$$

Proof. Once again, because of independence from $\underline{n}$, the cases of C being atomic are trivial. Let "$COND(\underline{n})$" be short for the phrase
" for all X, $\underline{n}(X) < \infty$ and $\underline{n}(X) \leq \underline{k}(X)$".
When $C = (C'; C'')$, we get

$$\mathcal{M}_{\underline{k}}(C, \Theta) \;:=\; \mathcal{M}_{\underline{k}}(C'', \Theta) \circ \mathcal{M}_{\underline{k}}(C', \Theta)$$

$$= \; (\bigcup_{\underline{a}} \{ \mathcal{M}_{\underline{a}}(C'', \Theta) \mid COND(\underline{a}) \; \}) \circ (\bigcup_{\underline{b}} \{ \mathcal{M}_{\underline{b}}(C', \Theta) \mid COND(\underline{b}) \; \})$$

$$= \; \bigcup_{\underline{a},\underline{b}} \{ \mathcal{M}_{\underline{a}}(C'', \Theta) \circ \mathcal{M}_{\underline{b}}(C', \Theta) \mid COND(\underline{a}) \text{ and } COND(\underline{b}) \}$$

$$= \; \bigcup_{\underline{n}} \{ \mathcal{M}_{\underline{n}}(C'', \Theta) \circ \mathcal{M}_{\underline{n}}(C', \Theta) \mid COND(\underline{n}) \; \}$$

$$= \; \bigcup_{\underline{n}} \{ \mathcal{M}_{\underline{n}}(C, \Theta) \mid COND(\underline{n}) \; \} \,.$$

When $C = ite(H)(C')(C'')$, we get

$(s,s') \in \mathcal{M}_{\underline{k}}(ite(H)(C')(C''),\Theta) \iff$

$[H\mathbf{tt}@s \text{ and } (s,s') \in \mathcal{M}_{\underline{k}}(C',\Theta) \quad \text{or} \quad H\mathbf{ff}@s \text{ and } (s,s') \in \mathcal{M}_{\underline{k}}(C'',\Theta)] \iff$

$[H\mathbf{tt}@s \text{ and } (s,s') \in \bigcup_{\underline{n}}\{\mathcal{M}_{\underline{n}}(C',\Theta) \mid COND(\underline{n})\ \}) \quad \text{or}$

$H\mathbf{ff}@s \text{ and } (s,s') \in \bigcup_{\underline{n}}\{\mathcal{M}_{\underline{n}}(C'',\Theta) \mid COND(\underline{n})\ \})] \iff$

$\exists \underline{n}$ satisfying $COND(\underline{n})$ with

$[H\mathbf{tt}@s \text{ and } (s,s') \in \mathcal{M}_{\underline{n}}(C',\Theta) \quad \text{or} \quad H\mathbf{ff}@s \text{ and } (s,s') \in \mathcal{M}_{\underline{n}}(C'',\Theta)] \iff$

$\exists \underline{n}$ satisfying $COND(\underline{n})$ with $(s,s') \in \mathcal{M}_{\underline{n}}(ite(H)(C')(C''),\Theta)$,

as required.

And now, for one final time, when $C = \nabla XD$, we proceed by induction on $\underline{k}(X)$. When 0, each $\underline{n}(X)$ on the right-hand side is also 0, and the result reduces to the empty set being a union of empty sets. When $0 < \underline{k}(X) < \infty$, we get, letting $COND_-$ be the $COND$ corresponding to $\underline{k}_{X-}$ rather than $\underline{k}$,

$$\begin{aligned}
\mathcal{M}_{\underline{k}}(\nabla XD,\Theta) &= \mathcal{M}_{\underline{k}_{X-}}(D,\Theta_{X\mapsto\mathcal{M}_{\underline{k}_{X-}}(\nabla XD,\Theta)}) \\
&= \mathcal{M}_{\underline{k}_{X-}}(D,\Theta_{X\mapsto\bigcup_{\underline{a}}\{\mathcal{M}_{\underline{a}}(\nabla XD,\Theta) \mid COND_-(\underline{a})\ \}}) \\
&= \bigcup_{\underline{a}}\{\ \mathcal{M}_{\underline{k}_{X-}}(D,\Theta_{X\mapsto\mathcal{M}_{\underline{a}}(\nabla XD,\Theta)}) \mid COND_-(\underline{a})\ \})) \\
= \bigcup_{\underline{a}}\bigcup_{\underline{b}}\{\ \mathcal{M}_{\underline{b}}(D,\Theta_{X\mapsto\mathcal{M}_{\underline{a}}(\nabla XD,\Theta)}) &\mid COND_-(\underline{a}) \text{ and } COND_-(\underline{b})\ \} \\
&= \bigcup_{\underline{e}}\{\ \mathcal{M}_{\underline{e}}(D,\Theta_{X\mapsto\mathcal{M}_{\underline{e}}(\nabla XD,\Theta)} \mid COND_-(\underline{e})\ \} \\
&= \bigcup_{\underline{n}}\{\ \mathcal{M}_{\underline{n}}(\nabla XD,\Theta) \mid COND(\underline{n}) \text{ and } \underline{n}(X) \neq 0\}\ . \\
&= \bigcup_{\underline{n}}\{\ \mathcal{M}_{\underline{n}}(\nabla XD,\Theta) \mid COND(\underline{n})\ \}\ .
\end{aligned}$$

The successive equalities come from : definition ; induction on $\underline{k}(X)$; **1.5** ; induction on C ; easy argument using **1.2** ; definition and taking $\underline{e}$ to

be $\underline{n}_{X-}$; add a $\emptyset$ to the union , respectively. Finally, when $\underline{k}(X) = \infty$, we get, letting $COND_{\underline{\ell}}$ be the $COND$ corresponding to $\underline{\ell}$ rather than $\underline{k}$,

$$\mathcal{M}_{\underline{k}}(\nabla XD, \Theta) \;=\; \bigcup_{\underline{\ell}} \{\; \mathcal{M}_{\underline{\ell}}(\nabla XD, \Theta) \mid \underline{\ell} \text{ agrees with } \underline{k}, \text{ except } \underline{\ell}(X) < \infty \;\}$$

$$= \bigcup_{\underline{\ell}} \bigcup_{\underline{n}} \{\; \mathcal{M}_{\underline{n}}(\nabla XD, \Theta) \mid COND_{\underline{\ell}}(\underline{n}) \text{ holds and } \underline{\ell} \text{ agrees with } \underline{k}, \text{ except } \underline{\ell}(X) < \infty \;\}$$

$$= \bigcup_{\underline{n}} \{\; \mathcal{M}_{\underline{n}}(\nabla XD, \Theta) \mid COND(\underline{n}) \text{ holds } \} \;,$$

as it is easy to verify that $\underline{n}$ satisfies $COND(\underline{n})$ [which is $COND_{\underline{k}}(\underline{n})$ and, in this case, simply says that $\underline{n}(X) < \infty$] if and only if there is an $\underline{\ell}$ agreeing with $\underline{k}$, except $\underline{\ell}(X) < \infty$, and such that $COND_{\underline{\ell}}(\underline{n})$ holds .

Theorem 1.7

$$\mathcal{M}_{\underline{\infty}} \;=\; \mathcal{M}_{\ell ix} \;;$$

that is, $\mathcal{M}_{\underline{\infty}}$ simply reproduces the fixpoint semantics defined in Section 1+ *of* [**sem**], which is also the union of the semantics of certain unfoldings when doing the only interesting case, a recursive command, by a main result exposited there—but those particular standard unfoldings apparently cannot be used to write a **definition** of the semantics as a union of 'sorta finite' things, except when simpleness is assumed (but now we know that depth semantics can). See also the next section here. We'll just use $\mathcal{M}$ for the overall semantics after the following proof, dropping the subscript $\underline{\infty}$ or lix.

The definition of fixpoint semantics is fairly explicit in the following proof, so the reader need not go back to [**sem**].

Proof. The two semantics are defined in the same way on all but '∇-commands', so we need only deal with that case.

To show that $\mathcal{M}_{\underline{\infty}}(\nabla YD, \Theta)$ is a fixed point of $[\beta \mapsto \mathcal{M}_{\underline{\infty}}(D, \Theta_{Y \mapsto \beta}) \;]$, temporarily abbreviate $\mathcal{M}_{\underline{k}}$ to $\mathcal{N}_k$ when $\underline{\underline{k}}(X) = \infty$ for all X except that $\underline{\underline{k}}(Y) = k \in \mathbf{N}$. [So the '$k$' within '$\underline{\underline{k}}$ actually means something here—it doesn't in the 'single-underlined $\underline{k}$'.] Calculate as follows.

$$\mathcal{M}_{\underline{\infty}}(\nabla Y D, \Theta) \;=\; \bigcup_{k<\infty} \mathcal{N}_k(\nabla Y D, \Theta) \;=\; \bigcup_{k<\infty} \mathcal{N}_{k+1}(\nabla Y D, \Theta)$$

$$= \bigcup_{k<\infty} \mathcal{N}_k(D, \Theta_{Y \mapsto \mathcal{N}_k(\nabla Y D, \Theta)}) \;=\; \bigcup_{a,b<\infty} \mathcal{N}_a(D, \Theta_{Y \mapsto \mathcal{N}_b(\nabla Y D, \Theta)})$$

$$= \bigcup_{a<\infty} \mathcal{N}_a(D, \Theta_{Y \mapsto \bigcup_{b<\infty} \mathcal{N}_b(\nabla Y D, \Theta)}) \;=\; \bigcup_{a<\infty} \mathcal{N}_a(D, \Theta_{Y \mapsto \mathcal{M}_{\underline{\infty}}(\nabla Y D, \Theta)})$$

$$= \mathcal{M}_{\underline{\infty}}(D, \Theta_{Y \mapsto \mathcal{M}_{\underline{\infty}}(\nabla Y D, \Theta)}) \;,$$

as required. The equalities, respectively, use : $(**)$ i.e. (**1.2** and **1.6**) ; **1.2** ; definition ; **1.2** and elementary considerations ; **1.5** ; $(**)$ again ; and $(**)$ yet again.

To show that $\mathcal{M}_{\underline{\infty}}(\nabla Y D, \Theta)$ is contained in every other fixed point, we show (superficially more generally) that

$$\mathcal{M}_{\underline{\infty}}(D, \Theta_{Y \mapsto \beta}) \subset \beta \implies \mathcal{M}_{\underline{\infty}}(\nabla Y D, \Theta) \subset \beta \;.$$

It suffices to deduce, by induction on k, that $\mathcal{N}_k(\nabla Y D, \Theta) \subset \beta$ for all k. When $k = 0$, on the left is $\emptyset$ by definition. For the inductive step,

$$\mathcal{N}_{k+1}(\nabla Y D, \Theta) \;=\; \mathcal{N}_k(D, \Theta_{Y \mapsto \mathcal{N}_k(\nabla Y D, \Theta)}) \;\subset\; \mathcal{N}_k(D, \Theta_{Y \mapsto \beta}) \;\subset\; \beta \;,$$

as required, respectively by definition; by induction and by continuity (**1.5**) which implies monotony; and finally by the left-hand side of the penultimate display and **1.2**.

2. **'Total' unfolding—a variant on Muskens' Theorem.**

Motivated by the definition of $\mathcal{M}_{\underline{n}}$ in the previous section, but only for those $\underline{n}$ with $\underline{n}(X) < \infty$ for all X, define a command $C^{\underline{n}}$ by induction on commands C as follows, where $gigabug := \nabla Y_0 call Y_0$ for some particular Y_0, a command which 'always loops', i.e. its semantics set, $\mathcal{M}(gigabug, \Theta)$ is empty for all Θ . Recall that $\underline{n}_{X-}$ agrees with $\underline{n}$, except its value on X is one less.

Definition. If C is atomic, $C^{\underline{n}} := C$;
$(C'; C'')^{\underline{n}} := (C'^{\underline{n}} ; C''^{\underline{n}})$;
$ite(H)(C')(C'')^{\underline{n}} := ite(H)(C'^{\underline{n}})(C''^{\underline{n}})$;
when $C = \nabla XD$, proceed by induction on $\underline{n}(X)$:
when $\underline{n}(X) = 0$, let $(\nabla XD)^{\underline{n}} := gigabug$;
when $0 < \underline{n}(X) < \infty$, let $(\nabla XD)^{\underline{n}} := |D^{\underline{n}_{X-}}|^{X \mapsto (\nabla XD)^{\underline{n}_{X-}}}$.

The last line uses substitution notation from [**sem**]. We shall get somewhat more involved with this in the next section. But the definition and results in [**sem**] are sufficient for this short section.

Proposition 2.1. *If being " ∇-free" for a command means that any occurrence of " ∇" is immediately followed by "$Y_0 call Y_0$"* (i.e. the only 'recursions' in the command, if any, are *gigabug*s), *then $C^{\underline{n}}$ is always ∇-free.*

Proof. This is immediate by induction: firstly on commands; then on $\underline{n}(X)$, when the command is ∇XD.

Theorem 2.2. *For any C and $\underline{n}$ with $\underline{n}(X) < \infty$ for all X, we have*

$$\mathcal{M}_{\underline{n}}(C, \Theta) \;=\; \mathcal{M}(C^{\underline{n}}, \Theta) \; .$$

Proof. All but the final case of the induction on C are very straightforward from definitions. When the command is ∇XD, proceed by induction on $\underline{n}(X)$, with the initial case immediate. When $0 < \underline{n}(X) < \infty$, we have

$$\mathcal{M}((\nabla XD)^{\underline{n}}, \Theta) \;=\; \mathcal{M}(|D^{\underline{n}_{X-}}|^{X \mapsto (\nabla XD)^{\underline{n}_{X-}}}, \Theta)$$

$$= \; \mathcal{M}(D^{\underline{n}_{X-}}, \Theta_{X \mapsto \mathcal{M}((\nabla XD)^{\underline{n}_{X-}}, \Theta)}) \;=\; \mathcal{M}_{\underline{n}_{X-}}(D, \Theta_{X \mapsto \mathcal{M}((\nabla XD)^{\underline{n}_{X-}}, \Theta)})$$

$$= \; \mathcal{M}_{\underline{n}_{X-}}(D, \Theta_{X \mapsto \mathcal{M}_{\underline{n}_{X-}}(\nabla XD, \Theta)}) \;=\; \mathcal{M}_{\underline{n}}(\nabla XD, \Theta) \; ,$$

as required. The equalities, respectively, are justified by : definition ; substitution property for $\mathcal{M}$—see $\mathbf{1.2}_+$ in [**sem**], somewhat less difficult than what is to come in the next section for $\mathcal{M}_{\underline{n}}$; the induction on commands ; the induction on $\underline{n}(X)$; and finally, by definition.

Corollary 2.3. (to **2.2**, **1.6** and **1.7**) *For any C, we have*

$$\mathcal{M}(C,\Theta) \;=\; \bigcup_{\underline{n}}\{\mathcal{M}(C^{\underline{n}},\Theta) \mid \underline{n}(X) < \infty \text{ for all } X \;\}) \;.$$

The proof is immediate. So, for each C, we have produced a collection of ∇-free commands in a canonical inductive way, which I'd call its *total unfoldings*, so that the semantics of C is the union of those of its total unfoldings. This is exactly the form of the main theorem in [**Musk**].

More precisely, on page 4 of that paper, the main theorem refers to "all linear programs L such that $L \vdash_{\mathcal{C}} P$ ", and says that "the denotation of P ... is exactly the union of the denotations of all such L". This is his Theorem 23, page 29. Similar to the above, these linear programs are free of recursions. But these L are produced using a little formal system, rather differently than our $C^{\underline{n}}$, perhaps somewhat less direct. Their 'construction' also depends crucially on the command language having the canonical non-deterministic 'union' command constructor, more-or-less in place of our *ite* (which can be conveniently obtained from the union operator). All our work here and in [**sem**] can be re-done using that non-deterministic command language instead, but I prefer to be more down-to-earth. Muskens' main theorem depends on pretty well the entire machinery using higher-order logic and his analogues of some basic results of the denotational semanticizers. I find this very interesting, and believe it likely to be stimulating for inventing and proving generalizations. However, our pedestrian approach to everything has advantages too, of conceptual simplicity and straightforwardness; and in any case, the corollary above is not quite exactly the same result as Muskens'. (Also, Muskens does everything for full-blown *mutual* recursion—as mentioned in the introduction, I expect [but don't claim!] the present paper generalized to the mutual case to be a matter of messier notation and no new ideas.)

Note that, among the main facts exposited in [**sem**] is the fact that one can write the semantics as a union of semantics of 'partial unfoldings', unfoldings which would always differ from the above when C has more than one ∇-subcommand. But, except for *simple* recursion, the latter formulae

seem not to be useable as *definitions*, whereas the formula in the corollary just above could be used as the main part of the definition, having first given the rather trivial inductive definition for ∇-free C.

3. **Substitution formula and recursion rule validity.**

We wish to prove (a correct analogue for) the following (ill-defined and incorrect, but intuitively appealing) formula for the 'depth-semantics' of the command, denoted here as $|C|^{X\to E}$, which is roughly what results from substituting the command E for each free occurrence of $callX$ in the command C :

$$\mathcal{M}_{\underline{n}}(|C|^{X\to E},\Theta) \;\stackrel{?}{=}\; \mathcal{M}_{\underline{n}}(C,\Theta_{X\mapsto\mathcal{M}_{\underline{n}}(E,\Theta)}) \ .$$

As a crucial step in the previous section, we used the analogue of this where there are no 'depth subscripts'. A proof of that easier fact may be found in **[sem], 1.2+**, p.33.

What else could one expect on the right-hand side, other than, as above, treating $|C|^{X\to E}$'s semantics as that of C, but with each free occurrence of X having the semantics of E imposed on it, by modifying the value of the 'Θ-component'? Besides wanting to be careful that 'the depth is right', the only really problematic thing is that normally there is some 're-naming of bound variables' built into the definition of substitution, and usually (elsewhere) there is no need to be explicit about the names of the new variables. But, as yet here, no connection exists between the value of the depth measure $\underline{n}$ on such a bound variable and on any of its new 'versions', after re-naming. So that's basically the only thing which needs to be fixed.

As explained roughly in the introduction, we will work strictly in $\mathcal{R}ec_{1.01}$; i.e. commands contain no repeated bound variable. That will constitute one extra requirement for re-naming, convenient, harmless and possibly dispensible with. The official notation for substitution will have an extra subscript 'h' which carries along re-naming information which ensures that substituting for a free variable which occurs more than once will not result in repeating bound variables from what is substituted, when substituting in for these different occurrences. But the earlier 'non-capture of free variables' motivation for re-naming will also be operative in the definition of $|C|_h^{X\to E}$, but will not be part of the notation, so there will still be that indeterminacy in the meaning of such a notation. In any case, the basic result, **3.5** below, will be the formula above, except with (1) the extra subscript h; (2) the proviso that the formula only applies to commands in $\mathcal{R}ec_{1.01}$; (3) $bd(C)\cap bd(E)=\emptyset$ helping to ensure that $|C|_h^{X\to E}$ *is* in $\mathcal{R}ec_{1.01}$; and, most important, (4) the formula only being asserted for those $\underline{n}$ which are constant on each orbit consisting of all procedure variables related under the equivalence relation generated by re-naming.

After that, we deal with the validity question of the partial correctness recursion rule, as discussed in the introduction. So let's stop the gab and get on with the technicalities.

Definition. The *disjoint union*, $A \sqcup B$, of two sets A and B will not be defined explicitly, but rather characterized by some of its properties. By definition, $\sqcup$ is a binary operator on sets that comes with two additional functions,

$$\alpha : A \to A \sqcup B \quad \text{and} \quad \beta : B \to A \sqcup B \ ,$$

such that :

(a) the operator $\sqcup$ is associative and commutative, with $\emptyset$ as a two-sided 'zero';

(b) for any set C, there is a bijection between the set of functions from $A \sqcup B$ to C, and the set of pairs of functions, from A to C and from B to C, a bijection defined by the obvious composition using the fixed functions α and β.

This is the category-theoretic notion of *coproduct*, applied to the category of sets, and is all we'll need here. In fact, '(a) up to canonical bijection' follows from (b).

Definition. Denoting by $foc(X, C)$ *the set of free occurrences of the procedure variable* X *in the command* C [again more handily characterized abstractly, rather than specified explicitly—no sane mathematician would doubt its existence] gives a collection of sets with the following properties [which are far more convenient and perspicuous for our purposes than any explicit choices for $foc(X, C)$] :

(i) $foc(X, y \leftarrow t)$ is empty ;

(ii) $foc(X, callX)$ is a singleton, but $foc(X, callY)$ is empty for $Y \neq X$;

(iii) $foc(X, (C'; C'')) \ = \ foc(X, C') \sqcup foc(X, C'')$;

(iv) $foc(X, ite(H)(C')(C'')) \ = \ foc(X, C') \sqcup foc(X, C'')$;

(v) $foc(X, \nabla X D) \ = \ \emptyset$, whereas $foc(X, \nabla Y D) = foc(X, D)$ if $Y \neq X$.

Proposition 3.1. *The set* $foc(X, C)$ *is non-empty if and only if* X *is in* $fr(C)$.

The proof, by induction on the structure of C, is so trivial this time that we shall omit it. Supplement that exercise by proving that $foc(X, C)$ is a *finite* set.

Definition. For a pair of procedure variables Y and Z, and D in $\mathcal{R}ec_{1.01}$, define $|D|^{Y\mapsto callZ} \in \mathcal{R}ec_{1.01}$ below by induction on *the length of* the string D. [This is a special case of the definition after next, with D here in place of C there, and with Z in place of E ; and ignoring h there, because its domain here is empty, so it is unique, simplifying considerably that complicated later definition. This definition is a needed stopgap, occurring as part of the next two definitions, the second of which generalizes it. Also we need to prove **3.2** below, and use it in establishing properties of the 're-naming' definition further down.]

$(1)'$ $\quad |y \leftarrow\!\!\div t|^{Y\mapsto callZ} \;:=\; y \leftarrow\!\!\div t$;

(2) $\quad |callY|^{Y\mapsto callZ} \;:=\; callZ$;
$(2)'$ When $X \neq Y$, $\; |callX|^{Y\mapsto callZ} \;:=\; callX$;

(3) $\quad |(C';C'')|^{Y\mapsto callZ} \;:=\; (|C'|^{Y\mapsto callZ} \;;\; |C''|^{Y\mapsto callZ})$;

(4) $\quad |ite(H)(C')(C'')|^{Y\mapsto callZ} \;:=\; ite(H)((|C'|^{Y\mapsto callZ})(|C''|^{Y\mapsto callZ})$;

(5) $\quad |\nabla YC|^{Y\mapsto callZ} \;:=\; \nabla YC$;
$(5)'$ When $X \neq Y$, $\quad |\nabla XC|^{Y\mapsto callZ} \;:=\; \nabla X'|C'|^{Y\mapsto callZ}$, where, for some choice of $\quad X' \notin \{X\} \cup \{Y\} \cup \{Z\} \cup C$, we take $C' := |C|^{X\mapsto callX'}$.
As it happens, only the case $X = Z$ needs this last elaborate definition, but, as written, it will be easier to see it as a special case of the even more elaborate definition just before **3.4**. The reason for not using induction on D itself is to make this last definition work, i.e. the $|C'|^{Y\mapsto callZ}$-part. The lengths of C and C' are both two less than that of ∇XC.

Lemma 3.2. *For any $\underline{k}, D, \Psi, Y$ and α, and where Z is chosen outside $\{Y\} \cup D$, we have*

$$\mathcal{M}_{\underline{k}}(|D|^{Y\mapsto callZ}, \Psi) \;=\; \mathcal{M}_{\underline{k}}(D, \Psi_{Y\mapsto\Psi(Z)}) \;.$$

Note that this lemma is a particular case of the formula we are seeking (and given precisely in **3.5**), but it seems logically simpler to give a completely separate proof, even if we are duplicating a few things from **3.5**'s proof.

Proof. Proceed by induction on **the length of the string** D. (N.B.—again, **not** structural induction on D itself).

For assignment commands D, both sides calculate the semantics of D itself, and the "Θ" is irrelevant, so this is trivial.

When $D = callX$ for $X \neq Y$, both sides again calculate the semantics of D itself, and give $\Psi(X)$.

When $D = callY$, the right-hand side gives $\Psi(Z)$. The left-hand side becomes

$$\mathcal{M}_{\underline{k}}(|callY|^{Y \mapsto callZ}, \Psi) \;=\; \mathcal{M}_{\underline{k}}(callZ, \Psi) \;=\; \Psi(Z) \;,$$

as required.

When $D = (C'; C'')$, the left-hand side is

$$\mathcal{M}_{\underline{k}}((|C'|^{Y \mapsto callZ} \,;\, |C''|^{Y \mapsto callZ}) \,,\, \Psi) \;=\; \mathcal{M}_{\underline{k}}(|C''|^{Y \mapsto callZ} \,,\, \Psi) \circ \mathcal{M}_{\underline{k}}(|C'|^{Y \mapsto callZ} \,,\, \Psi) \;=$$

$$\mathcal{M}_{\underline{k}}(C'', \Psi_{Y \mapsto \Psi(Z)}) \;\circ\; \mathcal{M}_{\underline{k}}(C', \Psi_{Y \mapsto \Psi(Z)}) \;=\; \mathcal{M}_{\underline{k}}((C'; C''), \Psi_{Y \mapsto \Psi(Z)}) \;,$$

as required.

When $D = ite(H)(C')(C'')$, suppose (s, s') is a member of the left-hand side, which is $\mathcal{M}_{\underline{k}}(ite(H)(|C'|^{Y \mapsto callZ})(|C''|^{Y \mapsto callZ}) \,,\, \Psi)$. First assume $H\mathtt{tt}@s$. Then

$$(s, s') \in \mathcal{M}_{\underline{k}}(|C'|^{Y \mapsto callZ}), \Psi) \;=\; \mathcal{M}_{\underline{k}}(C', \Psi_{Y \mapsto \Psi(Z)}) \;,$$

so $(s, s') \in \mathcal{M}_{\underline{k}}(ite(H)(C')(C''), \Psi_{Y \mapsto \Psi(Z)})$ as required. On the other hand, if $H\mathtt{ff}@s$, then essentially the same argument applies; just change C' to C''.

When $D = \nabla YC$, we have $|D|^{Y \mapsto callZ} \;=\; D$, so the left-hand side is $\mathcal{M}_{\underline{k}}(D, \Psi)$. But $Y \notin fr(\nabla YC)$, so by **1.3**, the right-hand side is also $\mathcal{M}_{\underline{k}}(\nabla YC, \Psi)$.

When $D = \nabla XC$ where $X \neq Y$, choose any $X' \notin \{X\} \cup \{Y\} \cup \{Z\} \cup D$, let $C' := |C|^{X \mapsto callX'}$. So $\nabla X'|C'|^{Y \mapsto callZ}$ is a typical choice for $|D|^{Y \mapsto callZ}$, and we must prove

$$\mathcal{M}_{\underline{k}}(\nabla X'|C'|^{Y \mapsto callZ}, \Psi) \;=\; \mathcal{M}_{\underline{k}}(\nabla XC, \Psi_{Y \mapsto \Psi(Z)}) \; .$$

Proceed by induction on $\underline{k}(X)$. We have $\underline{k}(X) = \underline{k}(X')$. Note that $Z \neq Y$, since $Z \notin \{Y\}$; and $Z \neq X$, since X is in $bd(D)$ but Z isn't.

When $\underline{k}(X) = 0$, both sides are the empty set.

When $0 < \underline{k}(X) < \infty$, let $\underline{\ell} := (\underline{k}_{X'-})_{X-}$. The left-hand side is

$$\mathcal{M}_{\underline{k}_{X'-}}(|C'|^{Y \mapsto callZ}, \Psi_{X' \mapsto \mathcal{M}_{\underline{k}_{X'-}}(\nabla X'|C'|^{Y \mapsto callZ}, \Psi)})$$

$$= \; \mathcal{M}_{\underline{\ell}}(|C'|^{Y \mapsto callZ}, \Psi_{X' \mapsto \mathcal{M}_{\underline{\ell}}(\nabla X'|C'|^{Y \mapsto callZ}, \Psi)})$$

$$= \; \mathcal{M}_{\underline{\ell}}(|C'|^{Y \mapsto callZ}, \Psi_{X' \mapsto \mathcal{M}_{\underline{\ell}}(\nabla XC, \Psi_{Y \mapsto \Psi(Z)})})$$

$$= \; \mathcal{M}_{\underline{\ell}}(C', (\Psi_{X' \mapsto \mathcal{M}_{\underline{\ell}}(\nabla XC, \Psi_{Y \mapsto \Psi(Z)})})_{Y \mapsto \Psi(Z)})$$

$$= \; \mathcal{M}_{\underline{\ell}}(C', (\Psi_{Y \mapsto \Psi(Z)})_{X' \mapsto \mathcal{M}_{\underline{\ell}}(\nabla XC, \Psi_{Y \mapsto \Psi(Z)})})$$

$$= \; \mathcal{M}_{\underline{\ell}}(C, (\Psi_{Y \mapsto \Psi(Z)})_{X \mapsto \mathcal{M}_{\underline{\ell}}(\nabla XC, \Psi_{Y \mapsto \Psi(Z)})}) \; [= \gamma, \text{say}] \; .$$

The successive equalities are justified by : definition ; $X' \neq X \notin |C'|^{Y \mapsto callZ}$ and **1.4** ; the inductive hypothesis on $\underline{k}(X)$; the inductive hypothesis on the length of D ; fairly obvious, since $X' \neq Y$; and again the inductive hypothesis on the length of D, since $C' := |C|^{X \mapsto callX'}$ and $X' \notin C$; respectively.

On the other hand, the right-hand side, by definition, is

$$\mathcal{M}_{\underline{k}_{X-}}(C, (\Psi_{Y \mapsto \Psi(Z)})_{X \mapsto \mathcal{M}_{\underline{k}_{X-}}(\nabla XC, \Psi_{Y \mapsto \Psi(Z)})}) \; ,$$

which reduces to γ, as required, by changing $\underline{k}_{X-}$ to $\underline{\ell}$ in both places using **1.4** since X' is in neither $bd(\nabla XC)$ nor in $bd(C)$.

When $\underline{k}(X) = \infty$, the result is immediate from the previous cases and **1.6**.

So the proof is complete (and you can imagine **3.5**'s will be even uglier).

Let f be any bijection from $\mathcal{P}cv$ to itself. Below is a formal inductive definition of $[)C(]_f \in \mathcal{R}ec_{1+}$, which is supposed to denote the result of re-naming, within $C \in \mathcal{R}ec_{1+}$, each bound occurrence of Y to the 'new' variable $f(Y)$, for all $Y \in bd(C)$. [That is, the unique bound occurrence $\nabla Y \cdots callY \cdots$ (if any) in C is replaced by $\nabla f(Y) \cdots callf(Y) \cdots$.] We shall only define this when

$$[bd(C) \cup fr(C)] \cap f(bd(C)) = \emptyset \ ;$$

that is, none of the new bound variables already occurs in C, bound or free.

Definition.

(1) $[)y \leftarrow\!\!\!\!\div t(]_f \ := \ y \leftarrow\!\!\!\!\div t$;

(2) $[)callY(]_f \ := \ callY$;

(3) $[)(C';C'')(]_f \ := \ ([)C'(]_f \ ; \ [)C''(]_f) \ \ ;$

(4) $[)ite(H)(C')(C'')(]_f \ := \ ite(H)(([)C'(]_f)([)C''(]_f)$;

(5) $[)\nabla X D(]_f \ := \ \nabla f(X)|[)D(]_f|^{X \mapsto callf(X)}$.

Here we use the previous definition for $|-|^{X \mapsto callf(X)}$.

Immediately from the definition, $C \in \mathcal{R}ec_{1.01} \implies [)C(]_f \in \mathcal{R}ec_{1.01}$

The following is easily proved by induction : *If f and f' agree on $bd(C)$, then $[)C(]_f = [)C(]_{f'}$.*

Applications.

(i) Given injective $g : bd(C) \to \mathcal{P}cv$ whose image is disjoint from $bd(C)$, extend it arbitrarily (bijectively) to f and define $[)C(]_g := [)C(]_f$.

(ii) By taking f' as the identity map, we get $[)C(]_f = C$ if $bd(C) = \emptyset$.

The formula $bd([)C(]_f) = f(bd(C))$ is also proved by induction, using $f(A \cup B) = f(A) \cup f(B)$ for the inductive cases of (3) and (4).

The most crucial property is again proved by an induction on commands E :

Proposition 3.3. *For f such that $[)E(]_f$ is defined, we have*

$$\mathcal{M}_{\underline{n}}([)E(]_f, \Theta) \;=\; \mathcal{M}_{\underline{n}}(E, \Theta) \;,$$

as long as $\underline{n}$ takes the same values on each variable and its re-naming, i.e. $\underline{n}(X) = \underline{n}(f(X))$ for all $X \in bd(E)$.

Proof. Proceed by induction on E. The cases (1) to (4) are easy. To do case (5), with $E = \nabla XD$, proceed by an induction on $\underline{n}(X)$ which is trivial when the latter is 0 or ∞. Then the inductive step reduces to the following, by applying the definition of $\mathcal{M}_{\underline{n}}$ to each side:

$$\mathcal{M}_{\underline{n}_{f(X)-}}(|[)D(]_f|^{X \mapsto call f(X)}, \Theta_{f(X) \mapsto \mathcal{M}_{\underline{n}_{f(X)-}}(\nabla f(X) | [)D(]_f |^{X \mapsto call f(X)}, \Theta)}) \stackrel{?}{=}$$

$$\mathcal{M}_{\underline{n}_{X-}}(D, \Theta_{X \mapsto \mathcal{M}_{\underline{n}_{X-}}(\nabla XD, \Theta)}) \quad ?$$

By **3.2** and the proposition's assumption about $\underline{n}$, the left-hand side of the latter display reduces to

$$\mathcal{M}_{\underline{n}_{X-}}([)D(]_f, \Psi_{X \mapsto \Psi(f(X))}) \qquad\qquad (*)$$

where

$$\Psi \;=\; \Theta_{f(X) \mapsto \mathcal{M}_{\underline{n}_{X-}}(\nabla f(X) | [)D(]_f |^{X \mapsto call f(X)}, \Theta)} \;.$$

Applying the definition of $[)\nabla XD(]_f$ and then the induction on $\underline{n}(X)$,

$$\Psi(f(X)) \;=\; \mathcal{M}_{\underline{n}_{X-}}([)\nabla XD(]_f, \Theta) \;=\; \mathcal{M}_{\underline{n}_{X-}}(\nabla XD, \Theta) \;.$$

This yields

$$\Psi_{X \mapsto \Psi(f(X))} \;=\; (\Theta_{X \mapsto \mathcal{M}_{\underline{n}_{X-}}(\nabla XD, \Theta)})_{f(X) \mapsto \mathcal{M}_{\underline{n}_{X-}}(\nabla XD, \Theta)} \;.$$

And so, also applying the induction on E to change $[)D(]_f$ to simply D in display $(*)$, the left-hand side of the display with the question mark does reduce to its right-hand side, since the last subscript just above specifying the value on $f(X)$ is immaterial by **1.3**, that variable not being free in D, because of $[)\nabla XD(]_f$ being defined. This completes the proof.

Definition. Assume given C, D and E in $\mathcal{R}ec_{1.01}$, and $X \in \mathcal{P}cv$. For each injective function

$$h \ : \ bd(E) \times foc(X, C) \to \mathcal{P}cv$$

whose image is disjoint from $bd(E)$, define $|C|_h^{X \mapsto E}$ by induction on C as follows : [The idea is that if γ is a free occurrence of X in C, that occurrence is to be replaced by E'', say, which is the result of first producing E' by changing, for each procedure variable Y, the unique bound occurrence $\nabla Y \cdots callY \cdots$ (if any) in E to $\nabla h(Y, \gamma) \cdots callh(Y, \gamma) \cdots$. So the E' produced for the different occurrences of X in C continue, as with E, to have all their bound variables distinct, but also entirely disjoint from one another, as we vary the free occurrence of $callX$. Then E'' is obtained from E' by changing, for each procedure variable Y, the unique bound occurrence $\nabla Y \cdots callY \cdots$ (if any) in E' to $\nabla Z \cdots callZ \cdots$ for Z's chosen different from each other and from every procedure variable within a million miles. This second step is the one which makes sure that none of the free variables in E get captured, so to speak, by ∇'s in C. It is the one where we don't need to get more specific about the choices of Z for each Y, at least not in the notation. But it should be clear by now that an inductive definition is absolutely necessary, so as to have a chance of giving (mechanical, but somewhat tortuous) proofs of properties.] The definition before **3.2**, as noted there, does agree with this definition, it being the considerably simpler case of this definition when E happens to be a procedure variable.

(1) $\quad |y \leftarrow\!\!\!\!\cdot\, t|_h^{X \mapsto E} \ := \ y \leftarrow\!\!\!\!\cdot\, t$;

(2) $\quad$ for $Y \neq X$, $\ |callY|_h^{X \mapsto E} \ := \ callY$;

whereas, $|callX|_h^{X \mapsto E} \ := \ [\,)E(\,]_{Y \mapsto h[Y, elt]}$, where elt is the unique element in $foc(X, callX)$, and noting that $bd([\,)E(\,]_g) = g(bd(E))$;

(3) $\quad |(C'; C'')|_h^{X \mapsto E} \ := \ (|C'|_{h\mathrm{restr}}^{X \mapsto E} \ ; \ |C''|_{h\mathrm{restr}}^{X \mapsto E})$,

where, in each case, "$h$restr" denotes h restricted to what the domain must be in that context, and where we are definitely working in $\mathcal{R}ec_{1.01}$, so this definition can only be 'legally' applied when the data is such that

$$bd(|C'|_{h\mathrm{restr}}^{X \mapsto E}) \cap bd(|C''|_{h\mathrm{restr}}^{X \mapsto E}) = \emptyset \ ;$$

(4) $\quad |ite(H)(C')(C'')|_h^{X\to E} \;:=\; ite(H)(|C'|_{h\mathrm{restr}}^{X\to E})(|C''|_{h\mathrm{restr}}^{X\to E})$,
where the same remarks as in (3) apply ;

(5) $\quad |\nabla XD|_h^{X\to E} \;:=\; \nabla XD$;

whereas, for $Y \neq X$, picking some

$$Z \notin X \cup Y \cup D \cup E \cup h[bd(E) \times foc(X, \nabla YD)] \ ,$$

define $|\nabla YD|_h^{X\to E} \;:=\; \nabla Z|B|_h^{X\to E}$, where, using the definition two paragraphs back *or* the induction, $B \;:=\; |D|^{Y\to callZ}$. Note that the length of B is two less than that of ∇YD, so $|B|_h^{X\to E}$ has already been defined. In this definition, we can use the same h on both sides because $foc(X, \nabla YD) = foc(X, B)$. This requires a minor inductive argument, to see the 'obvious' fact that $foc(X, |D|^{Y\to callZ})$ and $foc(X, D)$ are the same.

Note that, for any C' and C'' in (3) and (4), there are always choices of h so that the substituted formula (being defined) *is* actually defined. The extreme case might be where $C' = C''$, in which case the 'image under h' of the two halves would need to be disjoint from each other. Even though the two sets in the displayed part of the condition then look to be identical, they are not, because the two "$h$restr"s are quite different, to make them disjoint.

Proposition 3.4. *For data as in the definition, in particular, any choices of the 'Z' as in the last clause occurring as we build inductively, the string* $|C|_h^{X\to E}$ *is a command in* $\mathcal{R}ec_{1.01}$.

[The number of such choices is the size of $bd(C)$.]

Sketch Proof. Each right-hand side in the definition is certainly a command; it's the distinctness of the bound procedure variables that needs comment. For this proof, we'll use that $\mathcal{R}ec_{1.01}$ consists of exactly those commands in $\mathcal{R}ec_{1+}$ for which, for any X, the consecutive symbols ∇X occur at most once. There are no bound procedure variables in (1). The injectiveness of h plus the fact that E has mutually distinct bound variables are the relevant facts for (2). For (3) and (4), the disjointness of $foc(X, C')$ and $foc(X, C'')$ within $foc(X, C)$ is the crucial fact. For (5), the induction plus the fact that Z is chosen disjoint from $h[bd(E) \times foc(X, \nabla YD)]$ and from C are what is used. In every case, note that $bd(|C|_h^{X\to E})$ consists of $h[bd(E) \times foc(X, C)]$ together with all the Z's in the 'Y-to-Z-choices' occurring in the course of building $|C|_h^{X\to E}$ inductively.

Fortunately, we have no need to introduce a notation to specifically record those choices.

We shall say that $\underline{n}$ *is compatible with the re-namings in the substitution-definition of* $|C|_h^{X\mapsto E}$ when
(i) for each Y, $\underline{n}(h(Y,*))$ is independent of $*$; and
(ii) for all Y-to-Z-renamings in building up that definition, $\underline{n}(Z) = \underline{n}(Y)$.

Theorem 3.5. *For any C and E in $\mathcal{R}ec_{1.01}$ with $bd(C) \cap bd(E) = \emptyset$, and for all Θ, X and h, and any $\underline{n}$ compatible with the re-namings in the substitution-definition of $|C|_h^{X\mapsto E}$, we have*

$$\mathcal{M}_{\underline{n}}(|C|_h^{X\mapsto E}, \Theta) \;=\; \mathcal{M}_{\underline{n}}(C, \Theta_{X\mapsto \mathcal{M}_{\underline{n}}(E,\Theta)}) \;.$$

Note that this result agrees with **3.2**, which is its special case when E is a procedure variable.

Proof. Proceed by induction on **the length of the string** C.

For assignment commands C, both sides calculate the semantics of C itself, and the "Θ-component" is irrelevant, so this is trivial.

When $C = callY$ for $X \neq Y$, both sides again calculate the semantics of C itself, and give $\Theta(Y)$.

When $C = callX$, the right-hand side gives $\mathcal{M}_{\underline{n}}(E, \Theta)$. The left-hand side becomes, for some g,

$$\mathcal{M}_{\underline{n}}(|callX|^{X\mapsto E}, \Theta) \;=\; \mathcal{M}_{\underline{n}}([)E(]_g, \Theta) \;=\; \mathcal{M}_{\underline{n}}(E, \Theta) \;,$$

as required, where the 2nd equality comes from **3.3**, useable because $\underline{n}$ takes the same values on the new bound variables as it did on the old.

When $C = (C'; C'')$, the left-hand side is

$$\mathcal{M}_{\underline{n}}((|C'|^{X \mapsto E}_{h\text{restr}}\ ;\ |C''|^{Y \mapsto E}_{h\text{restr}})\ ,\ \Theta)\ =\ \mathcal{M}_{\underline{n}}(|C''|^{X \mapsto E}_{h\text{restr}}\ ,\ \Theta) \circ \mathcal{M}_{\underline{n}}(|C'|^{X \mapsto E}_{h\text{restr}}\ ,\ \Theta)\ =$$

$$\mathcal{M}_{\underline{n}}(C'', \Theta_{X \mapsto \mathcal{M}_{\underline{n}}(E,\Theta)}) \circ \mathcal{M}_{\underline{n}}(C', \Theta_{X \mapsto \mathcal{M}_{\underline{n}}(E,\Theta)})\ =\ \mathcal{M}_{\underline{n}}((C'; C''), \Theta_{X \mapsto \mathcal{M}_{\underline{n}}(E,\Theta)})\ ,$$

as required.

When $C = ite(H)(C')(C'')$, suppose (s, s') is a member of the left-hand side, which is $\mathcal{M}_{\underline{n}}(ite(H)(|C'|^{X \mapsto E}_{h\text{restr}})(|C''|^{X \mapsto E}_{h\text{restr}})\ ,\ \Theta)$. First assume $H\mathsf{tt}@s$. Then

$$(s, s') \in \mathcal{M}_{\underline{n}}(|C'|^{X \mapsto E}_{h\text{restr}}), \Theta)\ =\ \mathcal{M}_{\underline{n}}(C', \Theta_{X \mapsto \mathcal{M}_{\underline{n}}(E,\Theta)})\ ,$$

so $(s, s') \in \mathcal{M}_{\underline{n}}(ite(H)(C')(C''), \Theta_{X \mapsto \mathcal{M}_{\underline{n}}(E,\Theta)})$ as required. On the other hand, if $H\mathsf{ff}@s$, then essentially the same argument applies; just change C' to C''.

When $C = \nabla XD$, we must show

$$\mathcal{M}_{\underline{n}}(\nabla XD, \Theta)\ =\ \mathcal{M}_{\underline{n}}(\nabla XD, \Theta_{X \mapsto \mathcal{M}_{\underline{n}}(E,\Theta)})\ ,$$

which is immediate from **1.3**, since $X \notin fr(\nabla XD)$.

When $C = \nabla Y D$ where $Y \neq X$, let $B := |D|^{Y \mapsto callZ}$, after choosing some $Z \notin \{X\} \cup \{Y\} \cup C \cup D \cup E \cup image(h)$. So $\nabla Z|B|_h^{X \to E}$ is a typical choice for $|C|_h^{X \to E}$, and we must prove

$$\mathcal{M}_{\underline{n}}(\nabla Z|B|_h^{X \to E}, \Theta) \;=\; \mathcal{M}_{\underline{n}}(\nabla Y D, \Theta_{X \mapsto \mathcal{M}_{\underline{n}}(E,\Theta)}) \;.$$

Proceed by induction on $\underline{n}(Y)$, which equals $\underline{n}(Z)$.

When $\underline{n}(Y) = 0$, both sides are the empty set.

When $0 < \underline{n}(Y) < \infty$, let $\underline{\ell} := (\underline{n}_{Y-})_{Z-}$, recalling that compatibility of $\underline{n}$ implies that $\underline{n}(Y) = \underline{n}(Z)$, and therefore implies compatibility of $\underline{\ell}$ as well. By definition of the semantics of '∇-commands', the right-hand side is

$$\mathcal{M}_{\underline{n}_{Y-}}(D, (\Theta_{X \mapsto \mathcal{M}_{\underline{n}}(E,\Theta)})_{Y \mapsto \mathcal{M}_{\underline{n}_{Y-}}(\nabla Y D, \Theta_{X \mapsto \mathcal{M}_{\underline{n}}(E,\Theta)})})$$

$$= \; \mathcal{M}_{\underline{\ell}}(D, \Psi_{Y \mapsto \mathcal{M}_{\underline{\ell}}(\nabla Y D, \Theta_{X \mapsto \mathcal{M}_{\underline{\ell}}(E,\Theta)})})$$

where $\Psi := \Theta_{X \mapsto \mathcal{M}_{\underline{n}}(E,\Theta)}$. That last equality, inserting $\underline{\ell}$ in three places, follows from **1.4** because Z is a not a bound procedure variable in any of D or E or $\nabla Y D$, and also $\underline{Y \notin bd(E)}$. Now, by the induction on $\underline{n}(Y)$,

$$\mathcal{M}_{\underline{\ell}}(\nabla Z|B|_h^{X \to E}, \Theta) \;=\; \mathcal{M}_{\underline{\ell}}(\nabla Y D, \Theta_{X \mapsto \mathcal{M}_{\underline{\ell}}(E,\Theta)}) \; [= \; \alpha \; , \text{say}] \;,$$

so the right-hand side is $\mathcal{M}_{\underline{\ell}}(D, \Psi_{Y \mapsto \alpha})$.

The left-hand side is

$\mathcal{M}_{\underline{n}_{Z-}}(|B|_h^{X \to E}, \Theta_{Z \mapsto \mathcal{M}_{\underline{n}_{Z-}}(\nabla Z|B|_h^{Y \to E}, \Theta)})$ [[definition of semantics of $\nabla-$commands]]

$= \; \mathcal{M}_{\underline{\ell}}(|B|_h^{X \to E}, \Theta_{Z \mapsto \alpha})$ [[use **1.4** to change both $\underline{n}_{Z-}$ to $\underline{\ell}$: $Y \notin bd(|B|_h^{X \to E}) \cup bd(\nabla Z|B|_h^{X \to E})$]]
since it′s not bound in D because $\nabla Y D \in \mathcal{R}ec_{1.01}$]]

$= \; \mathcal{M}_{\underline{\ell}}(B, (\Theta_{Z \mapsto \alpha})_{X \mapsto \mathcal{M}_{\underline{\ell}}(E,\Theta)})$ [[induction on length of C]]

$= \; \mathcal{M}_{\underline{\ell}}(B, (\Theta_{X \mapsto \mathcal{M}_{\underline{n}}(E,\Theta)})_{Z \mapsto \alpha})$ [[by **1.4** since $\underline{Y \notin bd(E)}$ and $Z \notin bd(E)$]]

$= \; \mathcal{M}_{\underline{\ell}}(B, \Psi_{Z \mapsto \alpha}) \; = \; \mathcal{M}_{\underline{\ell}}(|D|^{Y \mapsto callZ}, \Psi_{Z \mapsto \alpha}) \; = \; \mathcal{M}_{\underline{\ell}}(D, (\Psi_{Z \mapsto \alpha})_{Y \mapsto \Psi_{Z \mapsto \alpha}(Z)}) \; = \; \mathcal{M}_{\underline{\ell}}(D, \Psi_{Y \mapsto \alpha}) \;,$

as required, using, for the last two equalities respectively: a previous lemma, namely **3.2**; and **1.3** combined with the fact that $Z \notin fr(D)$.

When $\underline{n}(Y) = \infty$, the result is immediate from **1.6**, using the previous cases.

So that does it, finally.

Validity of a rule of inference.

The main reference for this is [**sing**], where a proof system for a kind of *Floyd-Hoare logic* is presented (the *dynamic logic* kind, that is). Here we wish to show how to give a different proof of the validity of the rule for partial correctness of a recursive command. For simplicity, we will specialize it somewhat. The actual rule in [**sing**] can be proved the same way. The advantage here is that we can see now how to get this for so-called *non-simple* recursion, which was excluded in [**sing**].

That system actually is set up to deal with 'wfd's' ('well-formed dynamos') in dynamic logic. See [**sem**] for notation concerning wfd's.

We need to refine the notion of 'truth', called "tt", in [**sing**] to a notion "$\mathsf{tt}\underline{n}$". The semantic assertion "$\mathcal{A}\ \mathsf{tt}\underline{n}@s$" is defined by structural induction on the wfd $\mathcal{A}$, with all but the last clause identical with those in defining just "$\mathcal{A}\ \mathsf{tt}@s$" (standard Tarski $1^{\underline{\text{st}}}$order definition). That final clause is

$$< C > \mathcal{A}\ \ \mathsf{tt}\underline{n}@s \iff \exists (s, s') \in \mathcal{M}_{\underline{n}}(C, \Phi) \text{ with } \mathcal{A}\ \mathsf{tt}\underline{n}@s' \ .$$

Here Φ maps all procedure variables to the empty set (of pairs of states!).

Then "$\mathcal{A}\ \mathsf{tt}\underline{n}\mathbf{N}$" means "$\mathcal{A}\ \mathsf{tt}\underline{n}@s$ for all s".

It is immediate from **1.7** here and this definition that tt coincides with $\mathsf{tt}\underline{\infty}$.

It is easily deduced that $[C]G\ \mathsf{tt}@s$ if and only if, for some X, we have $[C]G\ \mathsf{tt}\underline{\underline{n}}@s$ for all $n \in \mathbf{N}$, where we define $\underline{\underline{n}}$ to denote that object mapping X to n and all other Y to ∞.

We wish to prove the following theorem, where $fr(\mathcal{A})$ is the union of the $fr(D)$ for which D is a subcommand of $\mathcal{A}$.

Theorem 3.6. *Assume that $\vdash'$ is a transitive binary relation on wfd's such that $\vdash'$ preserves $\mathsf{tt}\underline{n}\mathbf{N}$ for all $\underline{n}$. Suppose given C, f, X and $\mathcal{A}$ with the properties $\nabla XC \in \mathcal{R}ec_{1.01}$, $X \notin fr(\mathcal{A})$ and*

$$[\nabla XC]\mathcal{A} \ \vdash' \ [|C_1|_f^{X \mapsto \nabla XC}]\mathcal{A} \ ,$$

where C_1 denotes C with all its bound variables re-named to be distinct from each other and from those of ∇XC. Then $[\nabla XC]\mathcal{A} \ \ \mathsf{tt}\mathbf{N}$.

Corollary 3.7. *The rule of inference, with antecedent $[\nabla XC]\mathcal{A} \ \vdash' \ [|C_1|_f^{X \mapsto \nabla XC}]\mathcal{A}$ and consequent $[\nabla XC]\mathcal{A}$, is valid.*

The corollary refers to the $\vdash'$ in **[sing]**, which is given by a bunch of rules, not including the rule in the corollary itself. These rules are listed at the start of the appendix to Section 4 at the end of this paper, but aren't needed specifically below in the proof. The rule in the corollary is a sufficiently general special case, of the general partial correctness recursion rule in the proof system of **[sing]** that proving the latter's validity by this method should present no difficulty to the assiduous reader. Two interesting points here for me are that :

(i) we are now working in $\mathcal{R}ec_{1+}$, whereas the reference above refers strictly to $\mathcal{R}ec_{1-}$, where non-simple recursions, as explained in the introduction to this paper, are excluded ; and

(ii) the proof of the corollary uses the fact that we specialized Θ to Φ in the definition of 'truth'. The question unspecializing and adding "for all Θ" gives an interesting avenue for extensive comment in the next section. (See TT versus tt there.)

The $\vdash'$ in **[sing]** is of course a relation from *sets* of wfd's to wfd's, so the theorem is just referring to the case where that set is a singleton. The corollary's proof then consists in checking the truth preservation assumption in the theorem for the $\vdash'$ in **[sing]**. This is tedious but elementary, being a wade through the rules defining $\vdash'$ in **[sing]**, but one which is virtually identical with the corresponding proof in that reference for just 'ordinary truth', tt, itself. The point in (ii) above comes up in checking $\mathsf{tt}\underline{n}\mathbf{N}$-validity of the axiom $\neg < callX > \mathcal{A}$ for all X and $\mathcal{A}$; equivalently, the $\mathsf{tt}\underline{n}\mathbf{N}$-truth of the wfd $[callX]\mathcal{A}$.

Proof of 3.6. To show, as required, that $[\nabla XC]\mathcal{A}\ \mathtt{tt}\underline{\infty}\mathbf{N}$, we show by ordinary induction that, for each $n \in \mathbf{N}$, we have $[\nabla XC]\mathcal{A}\ \mathtt{tt}\underline{\underline{n}}\mathbf{N}$.

Here we are temporarily changing the normally meaningless " n"-part of the notation " $\underline{n}$" to being an integer, and defining $\underline{\underline{n}}$ to denote that object mapping X to n and all other Y to ∞, where X is the procedure variable explicitly involved in the theorem being proved.

Recall from dynamic logic that $[D]\mathcal{A}$ is defined to be an abbreviation for $\neg < D > \neg\mathcal{A}$ [in a sense giving a 'modal duality' between total and partial correctness]. It follows easily that

$$[D]\mathcal{A}\ \mathtt{tt}\underline{k}@s \iff \forall (s,s') \in \mathcal{M}_{\underline{k}}(D,\Phi) \text{ we have } \mathcal{A}\ \mathtt{tt}\underline{k}@s' \ .$$

Thus, $[D]\mathcal{A}\ \mathtt{tt}\underline{0}@s$ for all D and $\mathcal{A}$ for which $\mathcal{M}_{\underline{0}}(D,\Phi) = \emptyset$. But $\underline{\underline{0}} \neq \underline{0}$. It is only when D has the form ∇XC that we can use the same trivial argument, namely that $\mathcal{M}_{\underline{\underline{0}}}(\nabla XC,\Phi) = \emptyset$ [because $\underline{\underline{0}}(X) = 0$] to see that $[\nabla XC]\mathcal{A}\ \mathtt{tt}\underline{\underline{0}}s$ for all C and $\mathcal{A}$. So the initial case of our induction is trivially true.

To 'get from n to $n+1$', the displayed hypothesis in **3.5**, plus the 'truth preservation' of $\vdash'$, immediately give that $[|C_1|_f^{X\mapsto\nabla XC}]\mathcal{A}\ \ \mathtt{tt}\underline{\underline{n}}\mathbf{N}$. From this, we shall deduce that $[\nabla XC]\mathcal{A}\ \ \mathtt{tt}\underline{\underline{(n+1)}}\mathbf{N}$, as required.

So suppose that $(s,s') \in \mathcal{M}_{\underline{\underline{n+1}}}(\nabla XC,\Phi)$. The latter set of ordered pairs of states is, by definition, $\mathcal{M}_{\underline{\underline{n}}}(C,\Phi_{X\mapsto\mathcal{M}_{\underline{\underline{n}}}(\nabla XC,\Phi)})$, i.e. $\underline{\underline{(n+1)}}_{X-} = \underline{\underline{n}}$, obviously.

We need to show that $\mathcal{A}\ \ \mathtt{tt}\underline{\underline{(n+1)}}@s'$. Because $X \notin fr(\mathcal{A})$, and only on X do $\underline{\underline{(n+1)}}$ and $\underline{\underline{n}}$ differ, we see that

$$\mathcal{A}\ \ \mathtt{tt}\underline{\underline{(n+1)}}@s' \iff \mathcal{A}\ \ \mathtt{tt}\underline{\underline{n}}@s' \ .$$

[A very easy induction on $\mathcal{A}$ proves the obvious general result here, dependent on, and analogous to, **1.2**.] Because $[|C_1|_f^{X\mapsto\nabla XC}]\mathcal{A}\ \ \mathtt{tt}\underline{\underline{n}}\mathbf{N}$, it now suffices to show that $(s,s') \in \mathcal{M}_{\underline{\underline{n}}}(|C_1|_f^{X\mapsto\nabla XC},\Phi)$. The latter set of pairs of states, by our hard-earned **3.5**, is the same as $\mathcal{M}_{\underline{\underline{n}}}(C_1,\Phi_{X\mapsto\mathcal{M}_{\underline{\underline{n}}}(\nabla XC,\Phi)})$, since the extra assumptions in **3.5** hold. But in this last set of pairs of states, we can change the C_1 to C, since $\underline{\underline{n}}$ is constant (at the value ∞) on all variables other than X, in particular, it takes the same value on any two that are re-namings of one another to get C_1 from C, as needed. (This uses the main result in the following appendix.) But (s,s') *is* in that last set, as noted in the previous paragraph.

So we're done.

4. Zoology of partial correctness recursion rules.

There are two aspects of this topic which are "all over the map", so to speak, in various places in the literature. Both have come up earlier here.

One is the definition used for the semantics, not just fixpoint versus unfolding versus recursion depth versus whatever, but also (as mentioned in the introduction) the matter concerning truth in an interpretation (all assignments versus one particular 'empty' assignment, of procedure variables; perhaps also... versus truth at one assignment 'at a time' in validity of rules). In those more subtle treatments of partial correctness, particularly **[Old]**, where the gnarly matters of procedures with parameters interacting with declarations of variables is the main topic, this is dealt with by making a kind of syntactic rather than semantic assignment of variables; see 'units' in part II below.

The other aspect alluded to above is reflected in the syntax. (We're not talking merely of notation, which does vary quite spectacularly, as expected whenever logicians are involved!) Here the variation is in which, if any, restrictions from the most general 1-variable recursion are employed, namely, whether nesting is disallowed, and if allowed, whether the recursion body can have free variables other than those being 'recursed upon'. If yes to the latter, then we have what we take here to be the most general possibility for parameterless recursion. Mutual recursion is of course more general, but presents the same triple of alternatives. And the differences between those three seem more challenging (conceptually if not notationally) than the step from recursion on a single variable to mutual recursion.

So this section will go through a list of recursion rules for partial correctness (avoiding mutual recursion, where analogous comments would likely apply), and try to relate them to each other, discussing chiefly validity and the two matters above.

I. Rigid body recursion, global declaration only, so no nesting.

Here the command language, $\mathcal{R}ec_{nn}$, stripped to a bare minimum, could be

$$gig \mid y \div t \mid X_0 \;\; || \;\; (C'; C'') \mid ite(H)(C')(C'') \quad .$$

The command gig always diverges. There is only one procedure variable, and the semantics of X_0 is given by singling out a fixed command, say B_0, as the "body", which is implemented recursively each time X_0 is 'stumbled upon'. (Semantics given intelligently below!) Such a language is of course not Turing equivalent. Often some iteration construct is thrown in, e.g. 'while-do', which does allow the language to compute anything computable, but we agree with [**Harel**] that separating out recursion from iteration allows clearer analysis—the two never have a bad interaction anyway. No nesting of recursions can occur in $\mathcal{R}ec_{nn}$. The so-called meta-rule for partial correctness of a recursive command, translated from standard Floyd-Hoare language to dynamic logic notation is

$$(\ (F \to [X_0]G) \ \vdash \ (F \to [B_0]G)\ )\ /\ (F \to [X_0]G)\ ,$$

where F and G are $1^{\underline{\text{st}}}$order formulas; or (untranslated) :

$$(\ \{F\}X_0\{G\} \ \vdash \ \{F\}B_0\{G\}\ )\ /\ \{F\}X_0\{G\}\ .$$

I say: why bother with the sqiggly brackets when you're doing mathematics, as opposed to training less 'theoretically-interested' software engineers (who can't decide which convention to use for the sqiggly brackets anyway)? Knowing a priori that it consists of a formula, then a command, then a formula, the doubly concatenated string FCG, by unique readability, is completely unambiguous without throwing in the sqiggly brackets—where the formula F ends, and then where the command C ends, are simply the first symbol which gives an actual formula, respectively, command—as long as you don't start messing around with so-called abstract syntax and thereby leaving out brackets which are, in fact, needed in formulae and/or commands!

Taking $F = 1 \approx 1$ and simplifying propositionally, we get a derived rule somewhat reminiscent of the rule discussed in **3.5/3.6**,

$$(\ [X_0]G \ \vdash \ [B_0]G\ )\ /\ [X_0]G\ .$$

The next several pages will, among other things, contain a sketch proof of the validity of (a weakening of) this last rule, one which would be close to the validity proof in [**Nip**] after easily generalizing to the first rule above. We shall include a lot of discussion, in particular embedding the work in the earlier more general language $\mathcal{R}ec_{1+}$ where nesting could occur. The embedding will depend on the previous choice of the command B_0, which now gets to be part of the syntactic discussion, not just a parameter for setting up the semantics of $\mathcal{R}ec_{nn}$. From that more general viewpoint, we'll also see why the rule 'looks wrong' (without necessarily *being* wrong, i.e. invalid).

The language $\mathcal{R}ec_{nn}$ here is not exactly a sublanguage of $\mathcal{R}ec_{1+}$. (If we tried to identify X_0 with $callX_0$ or alternatively with $\nabla X_0 B_0$, the semantics is slightly askew either way, using the standard semantics of $\mathcal{R}ec_{nn}$ below.) So define an embedding

$$\Gamma \;=\; \Gamma_{B_0} \;:\; \mathcal{R}ec_{nn} \;\rightarrow\; \mathcal{R}ec_{1+}$$

by first fixing some procedure variable $Y_0 \in \mathcal{P}cv$ (which could be X_0 if the latter was in $\mathcal{P}cv$, but it's less confusing to have a separate notation), and then defining

$$\Gamma(S) \;:=\; |S|^{X_0 \rightarrow \nabla Y_0 C_0} \quad \text{where} \quad C_0 \;:=\; |B_0|^{X_0 \rightarrow callY_0} \;.$$

Here, $\Gamma(gig) = gig = gigabug$, in $\mathcal{R}ec_{1+}$, can be taken to be $\nabla Y_0 callY_0$, which does indeed universally diverge. Notice how X_0 from the simple language $\mathcal{R}ec_{nn}$ is connected to both the *call*- and the ∇-commands from the fancier language $\mathcal{R}ec_{1+}$ here. Also the replacement operation for X_0 in both these is perfectly straightforward, no re-naming worries, so we'll leave the reader to write down the inductive definition.

Next, here is the so-called *approximation* semantics for $\mathcal{R}ec_{nn}$, as given in [**Apt**] (really the same idea as what we called an *unfolding* semantics earlier here). In the course of the discussion below, we'll show how a very simple *recursion depth* semantics, from [**Nip**] and perhaps elsewhere, agrees with it—but to be convincing in some later discussion of [**Apt**], we need to be sure the semantics here agrees with what's there. Define

$$m \;:\; \mathcal{R}ec_{nn} \;\rightarrow \mathcal{B}sm$$

by firstly defining $m(S)$ by induction on the number of occurrences of X_0 in S, and then, at each such inductive step, by structural induction on $S \in \mathcal{R}ec_{nn}$,

where assignments, sequencing, and 'if-the-else' get the same formulas as always, and where $m(gig) := \emptyset$ of course, leaving only one case, when $S = X_0$:

$$m(X_0) \;:=\; \bigcup_{n\geq 0} m(B_0^{(n)}) \quad \text{with} \quad B_0^{(0)} \;:=\; gig \quad \text{and} \quad B_0^{(n+1)} \;:=\; |B_0|^{X_0 \to B_0^{(n)}} \;.$$

Note that $B_0^{(n)}$ has no X_0, making that first inductive step okay. It follows easily that, for any $S \in \mathcal{R}ec_{nn}$, we have

$$m(S) \;:=\; \bigcup_{n\geq 0} m(|S|^{X_0 \to B_0^{(n)}}) \;.$$

Again, this means straightforward substitution for X_0. The right-hand side is a union of the semantics of 'X_0-free' commands. Prove this formula by a 'double induction' exactly as in the definition.)

Theorem 4.1. *The map Γ preserves semantics; that is, for all S,*

$$\mathcal{M}(\Gamma(S), \Theta) \;=\; m(S) \;.$$

In particular (unsurprisingly) the left-hand side is independent of Θ, since, after all, $\Gamma(S)$ has no free assignment variables.

Proof. Again proceed by a 'double induction' exactly as in the definition; and find that all cases are simple and mechanical, except when the number of occurrences of X_0 is 1, and the command S is just X_0 itself. The required equality is then

$$\mathcal{M}(\nabla Y_0 C_0 \;, \Theta) \;=\; \bigcup_{n\geq 0} m(B_0^{(n)}) \;.$$

But the left-hand side here, by the work in Section 2, is

$$\bigcup_{n\geq 0} \mathcal{M}((\nabla Y_0 C_0)^{\underline{\underline{n}}} \;, \Theta) \;, \quad \text{where} \;\; \underline{\underline{n}}(Y_0) = n \;.$$

(The value of $\underline{\underline{n}}$ on other procedure variables is irrelevant.) Applying the already proved case re zero occurrences of X_0, it suffices to show that

$$\Gamma(B_0^{(n)}) \;=\; (\nabla Y_0 C_0)^{\underline{\underline{n}}} \;,$$

where the left-hand side simplifies to just $B_0^{(n)}$ itself, because of the latter being X_0-free. Now one can prove that (simplified) displayed equality by an

induction on $n \geq 0$, whose initial case gives *gigabug* on both sides. For the inductive step, the right-hand side is

$$(\nabla Y_0 C_0)^{\underline{\underline{n+1}}} \;=\; |C_0^{\underline{\underline{n}}}|^{Y_0 \to (\nabla Y_0 C_0)^{\underline{\underline{n}}}} \;=\; |C_0^{\underline{\underline{n}}}|^{Y_0 \to B_0^{(n)}} \;=\; |C_0|^{Y_0 \to B_0^{(n)}}$$

$$=\; ||B_0|^{X_0 \to call Y_0}|^{Y_0 \to B_0^{(n)}} \;=\; |B_0|^{X_0 \to B_0^{(n)}} \;=\; B_0^{(n+1)} \;,$$

as required to complete the proof. (The 3rd equality follows because C_0 is ∇-free.)

Applying Γ to each of the pieces in the 3rd (derived) rule previous naively gives a 'rule'

$$\text{(INVAL)} \qquad (\; [\nabla Y_0 C_0]G \;\vdash\; [|C_0|^{Y_0 \to \nabla Y_0 C_0}]G \;) \;/\; [\nabla Y_0 C_0]G \;.$$

Note that this would just be the rule in **[sing]**, except that we've used $\vdash$ rather than $\vdash'$. Even more naively, because of the theorem, if the rule to which we just applied Γ is valid, surely this one is too.

But it isn't, except for very special C_0—at least not if we are talking about working in the entire language $\mathcal{R}ec_{1+}$ and about $\vdash$ meaning the system in **[sing]** (rather than its little brother called $\vdash'$ and used in an antecedent of the partial correctness rule of that system for $\vdash$ [but not part of the system for $\vdash'$], as discussed several times later in this section). Invalidity is easy to see—the point is that the antecedent is trivially always true, given directly by one of the other axioms of the system for $\vdash$ (but not $\vdash'$). However, most instances of the consequent are false; one could just take G to be $1 \approx 0$, and any C_0 for which $\nabla Y_0 C_0$ converged for at least one input state.

The reason this is not a contradiction of world-shaking significance is that the rule before applying Γ is certainly valid (below well reproduce something along the lines of **[Nip]**s proof), its domain of application being the rather pathetic (in my view!) command language $\mathcal{R}ec_{nn}$, with $\vdash$ coming from a fairly small set of rules. One could undoubtedly formulate, in terms of the tiny image language under Γ of $\mathcal{R}ec_{nn}$ within $\mathcal{R}ec_{1+}$, and an appropriate set of rules to define the meaning of $\vdash$ in its antecedent, a scenario making (INVAL) just above valid.

This is an example of a phenomenon that O'Donnell **[O'D]** points out rather forcefully : a rule (or "metarule" if you wish) which, as above, uses an antecedent with the $\vdash$ for the whole system, is unstable in that adding

additional rules at a later date (even if it only amounts to expanding the language so that earlier rules cover a larger number of instances) could convert a sound system into an unsound one. There are a few ways to avoid this danger, two of them being :

(1) the use of a subsidiary $\vdash'$ or $\vdash^?$ as discussed later in this part, referring to a set of rules which are to remain fixed whatever else you later decide to do with the entire system when perhaps expanding the language; or

(2) using some 'Gentzenizing' or so-called 'correctness phrases', the syntax then getting somewhat baroque, as discussed in part II just ahead.

We shall finish this part of this final section by discussing the recursion depth semantics for $\mathcal{R}ec_{nn}$ from **[Nip]** and perhaps earlier somewhere, showing it agrees with the approximation semantics, and that it can be used in a validity proof, analogously to what is done in Sections 2 and 3 here, but much easier. And finally we comment critically on a few versions of what is in the literature on attempts to describe validity proofs for the recursion rule at issue.

Definition. Define $m_n : \mathcal{R}ec_{nn} \to \mathcal{B}sm$ by induction on n, then induction on structure, using all the usual formulas for assignments, sequencing and if-then-else, plus the requirement that

$$m_0(X_0) := \emptyset \quad \text{and} \quad m_{n+1}(X_0) := m_n(B_0) \ .$$

So that's dead simple, isn't it? Just an index shift.

Theorem 4.1. $$m(S) \ = \ \bigcup_n m_n(S) \ .$$

Proof. More precisely, and as suffices, we have, for all n,

$$m_n(S) \ = \ m(|S|^{X_0 \to B_0^{(n)}}) \ .$$

Induction on n, then, for each n, induction on S, proves this. All cases except $S = X_0$ are mechanical. The latter requires one to show

$$m_n(X_0) \ = \ m(B_0^{(n)}) \ .$$

The case $n = 0$ is immediate from definitions. The inductive step is

$$m_{n+1}(X_0) \ = \ m_n(B_0) \ = \ m(|B_0|^{X_0 \to B_0^{(n)}}) \ = \ m(B_0^{(n+1)}) \ .$$

The middle equality is from the induction on n and the other two are definitions.

Now let $\vdash^?$ be the system in [**Apt**] for the present language, except that the rule for recursion is omitted. If you inspect and accept his proof for (Cook-)completeness of his system, you will notice that the rule for recursion, given at the start of this part, can be weakened by using only $\vdash^?$ in its antecedent, rather than the full $\vdash$, and completeness still holds. This is analogous to my use of the weaker $\vdash'$ in [**sing**], and is also in concord with Apt's somewhat ambiguous phrase "using the **other** rules and axioms" ([**Apt**], p.444) when describing this rule. (But then he doesn't actually do that in the mathematics!)

Specializing the resulting rule somewhat as we did at the beginning of this section gives

$$(\ [X_0]G \ \vdash^? \ [B_0]G \ ) \ / \ [X_0]G \ .$$

This is the rule for which we can now describe a rather nice clean proof of validity (not a new one), using the just defined depth description of the semantics. It's pleasant not to need any messing with substitution. As discussed later, if one goes back from $\vdash^?$ to the 'full' $\vdash$,
(i) the corresponding proof alluded to in [**Nip**] is probably a slightly more sophisticated induction; whereas
(ii) I'm not entirely sure about the nature of the proof in [**Apt**].
(Going back to the unspecialized rule with "$F \rightarrow ...$" in front of all three wfds is a trivial matter.)

For that validity proof, we'll just deduce that $[X_0]G \ \mathtt{tt}n\mathbf{N}$ for all $n \geq 0$, by induction on n, where $\mathcal{A} \ \mathtt{tt}n@s$ is defined inductively on the dynamic logic string $\mathcal{A}$ exactly as the classical Tarski 'definition of truth', except for the case

$$[C]\mathcal{B} \ \mathtt{tt}n@s \quad \Longleftrightarrow \quad \text{for every } (s,s') \in m_n(C) \text{ we have } \mathcal{B} \ \mathtt{tt}n@s' \ .$$

Actually, that's a consequence of what probably should be the main part of the definition, namely

$$< C > \mathcal{B} \ \mathtt{tt}n@s \quad \Longleftrightarrow \quad \text{there is an } (s,s') \in m_n(C) \text{ with } \mathcal{B} \ \mathtt{tt}n@s' \ .$$

(But the latter also follows from the former—there is a kind of duality between total and partial correctness, which dynamic logic brings some clarity to by seeing it as the usual duality in modal logic.)

It follows, as implicitly suggested above, that $[C]G$ $\mathtt{tt}@s$ if and only if $[C]G$ $\mathtt{tt}n@s$ for all n. And, intuitively, n is of course measuring recursion depth. (Note that the corresponding condition for $< C > G$ is different.)

The fundamental fact needed now is that $\vdash^?$ preserves "$\mathtt{tt}n$ $\mathbf{N}$" (which as usual is defined to mean "$\mathtt{tt}n@s$ for all s"). Exceptionally straightforward checking rule-by-rule proves this. But it isn't so straightforward, in fact it almost seems circular, if the recursion rule itself needs to be checked, which would be the case with the original (customary) rule, where $\vdash$, not $\vdash^?$, is in that famous antecedent. See the paragraph after next.

Anyway, the proof is now similar to (but simpler than) what we did in Section 3. The preservation of truth, and the antecedent, get us from $[X_0]G$ $\mathtt{tt}n\mathbf{N}$ to $[B_0]G$ $\mathtt{tt}n\mathbf{N}$. The latter and definitions now get us to $[X_0]G$ $\mathtt{tt}(n+1)\mathbf{N}$, completing the induction, whose initial case is trivial. The very last part of the argument of course depends on the fact that recursion depth has no bearing on truth relating to a $1^{\underline{\mathrm{st}}}$order formula such as G.

Except for the changing of $\vdash$ to $\vdash^?$, this must be the proof not given in [**Nip**]. I have not got my hands on their ISABEL/HOL output. But maybe the best way to understand why the unnecessarily (for completeness) stronger rule is also valid (and the *only* way to prove its validity that I am convinced is solid), is to simultaneously, in the induction on n, prove the fact that that rule preserves "$\mathtt{tt}n$". Just amplify the wording in the previous paragraph a little. Unless that's what ISABEL/HOL does, I really don't understand the proof alluded to in [**Nip**], and, for that matter, certainly not the one in [**Apt**], as discussed a few paragraphs below.

As for the 'replacing' of derivations (in an amusing but not-too-serious attempt to deduce validity of the stronger rule from that of the weaker), suppose one has a derivation witnessing $\Omega \vdash \mathcal{A}$, where the dynamic logic strings ("wfds") are based on $\mathcal{R}ec_{nn}$, and $\vdash$ means the system in [**Apt**], whose other rules need not be spelled out explicitly here. Though he converts to a Gentzen-style system in a somewhat mysterious manner (see below), that's not needed to (and it's not all that difficult to) say what such a derivation of length ℓ could be taken to be in such a way that the meaning of the rule is preserved. For $\ell = 1$, it is the single wfd $\mathcal{A}$, which must either be in Ω or be a system axiom. And, as a temporary technical definition, the wfd $\mathcal{A}$ is the only one which *adheres to* this '1-line derivation'. Then, inductively on ℓ, such a derivation is an ordered pair $(D, \mathcal{A})$, where D forms a derivation of length "$\ell - 1$" witnessing either

(i) $\Omega \vdash$ some $\mathcal{B}$, where $\mathcal{A}$ is a consequent of one of the rules other than the recursion rule based on antecedent(s) which are wfd(s) *adhering to* D (and the strings adhering to

this [length ℓ]-derivation are defined to be those adhering to D plus $\mathcal{A}$ itself); or else

(ii) $\{F \to [X_0]G\} \vdash F \to [B_0]G$, an instance of the antecedent in the recursion rule, and where $\mathcal{A} = F \to [X_0]G$ (and the only string adhering to this [length ℓ]-derivation is $\mathcal{A}$ itself).

If we change from $\vdash$ to $\vdash^?$, change the definition by requiring in (ii) that its derivation be free of any of type (ii). Now (as an exercise??) the reader might try to show how to modify the former sort of derivation to one of the latter. Proceed by induction on ℓ, and try your best. Success would give another, quite different, proof for the soundness of the unnecessarily strong rule, based on that for the weak one.

The next two short subsections are there only to partially justify all this effort being expended on discussing versions of partial correctness rules for recursive commands, and on attempted proofs of their validity, or, more importantly, of the soundness of the systems they are embedded in.

Aptology.

What about [**Apt**]'s proof of validity for the recursion rule, of course relating to *partial* correctness? (For total correctness, [**AdB**] had to modify its 'proof', more accurately, modify the system, and so, as well, re-do the proof of relative completeness, after discovering the fact that Apt's system had been unsound for the previous 10 years of its frequent bibliographic appearances.) In [**Apt**], bottom p.448, when launching his description of a possible soundness proof for the system for partial correctness (the one, as mentioned above, whose analog didn't quite work for total correctness—top p.453 says "arithmetical soundness of G_0 can be proved in a way analogous to the way soundness of G was proved"), the author says "It is... not clear... in what sense the recursion rule is to be proved sound." The system referred to is called G. A Gentzen-style system called K is then introduced, and on p.449, after a few more definitions referring to K (not to G where even the objective is apparently unclear), we find

Claim 1. *If the proof system K is good, then the proof system G is sound.*

The proof of this is not given in anything but the sketchiest form. In any case, it is moderately confusing how, when one is unclear about the meaning of soundness for one of the rules in a system, one would go about proving that such a system is sound, either as a consequence of some other system having some property ("goodness"), or simply in an absolute sense. In any case, the sketch proof is based on the claim that G can derive ϕ from a set T of F-H statements if and only if K can derive ϕ from T, together with

the assertion that ϕ is I-true if and only if it is I-good. Here I is a general interpretation, and the object ϕ has been identified with a suitable syntactic object with which the system K deals. That syntax, the system K itself, and the various definitions leading up to the notion of *goodness* take about a page to provide. So, besides the vagueness about what the original proof system called G is really requiring in the way of validity/soundness, this approach appears to be somewhat byzantine.

And that vagueness makes evaluating the proof of soundness in [**Apt**] pretty well impossible without a lot of guesswork. One attempts a few guesses, or just makes a temporary suspension of disbelief, and tries to follow the proof that K is good, which consists of four further statements, a. to d., p.450. Only c. is discussed, but the others do seem straightforward. The proof of c. looks rather like our use earlier of depth semantics to prove validity of a weaker (but adequate) recursion rule. So my mediocre brainpower just cannot really 'get' this proof.

Without basically [**Nip**]'s proof above, and the long-term acceptance of the rule, my present state of befuddlement about other proofs would make it perfectly conceivable to me for the (stronger) rule not to be valid. The weaker one certainly is valid, and I seriously doubt whether any actual use of these rules in program verification has ever gone beyond the weaker rule. By (Cook-)completeness, there would never be a need to go beyond it. In any case, it *is* perfectly clear to *me* what soundness means for both.

I have really been overly critical above, in that one certainly can write out this byzantine stuff in an acceptable form, even a version of it which applies to recursion with nesting. This seems to be due to Olderog, and can be found in part II coming up next.

Encyclopedicity.

Psasinspc might nose around and find what appears to be a simple explanation of the meaning/soundness/validity/... of a system consisting of a collection $\mathcal{R}$ of ordinary rules (including some axioms) together with a single "meta rule", $(H \vdash H')/H''$, similar to what we've been pondering. He might say that

"$H_1, H_2, \cdots, H_n$ *is a formal proof of* H_n" in such a system if and only if

"$H, H_1, H_2, \cdots, H_n$ *is a formal proof of* H_n *from* H *in the system* $\mathcal{R} \cup \{H'/H''\}$",

the latter being a 'usual' (Hilbertian) system of rules.

However, that reduction to a more familiar style of proof system is just a bit too simplistic, it seems. For example, the following sequence of partial correctness assertions (using $\mathcal{R}ec_{nn}$ and dropping both the earlier used subscripts 0 and the squiggly brackets I had complained about in small print),

$$FXG \quad , \quad Gx \leftarrow\!\!\!\!\cdot\, xG \quad , \quad F(X; x \leftarrow\!\!\!\!\cdot\, x)G \quad ,$$

is a perfectly good derivation of the third assertion from the first, where the rules include

at least the usual sequencing rule and assignment axiom, plus the rule (unused in that derivation) FBG/FXG, the latter coming via the 2nd quoted line above from the metarule $(FXG \vdash FBG)/FXG$ we have been considering. But surely

$$Gx \leftarrow\!\!\!\!\!\!- xG \quad , \quad F(X; x \leftarrow\!\!\!\!\!\!- x)G$$

is **not** a valid derivation in the system including that metarule.

However, **[Cst]** p.934, gives exactly the same interpretation as given in the second previous paragraph. I've quoted almost directly, the phrase *"from H"* being the only addition. Actually there is surprisingly little other than that on imperative recursion in this encyclopedic work. He spends a good deal of time on more primitive imperative stuff, but passes to functional languages for giving any deeper results on recursion, much as is done in several standard textbooks on semantics of software languages. The rule we've been talking about is the occasion for the above attempt in **[Cst]** to explain "metarules". Furthermore, on validity of the partial correctness recursion rule, nothing is attempted beyond references to **[Apt]** and **[AdB]**, the latter itself merely referring to the former (when the question of validity of the metarule for *partial* correctness came up).

The validity of rules for *total* correctness of recursive commands (oddly enough) seems a much less subtle matter (despite the error in **[Apt]**), and details on that *are* provided by **[AdB]**.

II. The 'same' rule used, but where nested recursion can occur.

Here we severely simplify what I take to be a fairly definitive work (which doesn't even get a mention in the extensive reference list in **[Ten]**, perhaps because of being from an heretical 'semantics religion'!) on partial correctness related to "Algol-like" languages, namely **[Old]**. Despite its extreme technicality, it is certainly a paper done to the taste of a pure mathematician, with little left vague, and, refreshingly, not one where subsequent work has turned up errors in anything basic, AFAIK. It is however a paper, if ever there was one, which could use a separate list of all the notation, repeating the definition in each case, or at least referring to the location of that definition in the main text! We strip away all the difficult stuff related to the interaction between recursion (*with* parameters) and declaration of variables, to see how the recursion rule used there is given its semantics and proof of validity in the resulting much easier setting.

One apparently needs several syntactic categories—the above reference has all the following: *statements* (I'll change the word to "commands"), *blocks*, *environments*, ALGOL-like *programs*, *units*, *atomic Hoare formulas* of three distinct types, *sets of the latter*, and finally, *proof lines*. In our simplified setting, several of these can be omitted. (But it's still rather baroque, mainly due to avoiding the language of dynamic logic, IMHO. Perhaps by redoing **[Old]**, but within dynamic logic, one could reproduce all his partial correctness stuff, do the job for total correctness and much more, and possibly explain the Clark incompleteness result by having, other than the oracle for 'true' $1^{\underline{\text{st}}}$order formulas, all rules except the one for partial correctness of recursion being decidable, as they are supposed to be, but the latter only decidable when one of the required language features in Clarke's theorem is missing.)

Not for the language, but at least for the rule for partial correctness of recursion, we shall also shrink from mutual to single recursion, in which case, with the other vast simplification mentioned above, **[Old]**'s rule (8) reduces to something close to the sort of rule looked at in the last part of this section, namely (obliterating sqiggly brackets)

$$(H \cup \{FD|callXG\} \longrightarrow FD|BG) \;/\; (H \longrightarrow FD|callXG) \quad ,$$

where $D|callX \rightsquigarrow D|B$. A good deal of explanation is needed here.

We begin with the basic language: it has *commands* C given almost

exactly by what was $\mathcal{R}ec_{\infty+}$ in **[sem]**

$$y \leftarrow t \mid callX \mid\mid (C';C'') \mid [\ ite(H)(C')(C'') \mid \nabla DC \quad .$$

And it has *declarations* D (involved just above in the structural inductive definition of commands) given by

$$emptystring \mid procX;C;\ \mid\mid D'D'' \quad .$$

(All the latter says is that every declaration has the form

$$procX_1;C_1;procX_2;C_2;\cdots procX_k;C_k; \quad ,$$

where possibly $k = 0$.) But *the X_i must be distinct*, an extra requirement.

So this is a 'mutual' structural inductive definition, popularly given by BNF-notation in the CS literature. For the basic symbols, see **[sem]**. We will avoid stuff peripheral to our intent here, by forgetting about anything other than the $1^{\underline{\text{st}}}$order language of number theory and its canonical interpretation consisting of the natural numbers with their usual ordering relation, and addition and multiplication operations.

Next, a 'Floyd-Hoare assertion' here has the form $FD|CG$ (more conventionally, $\{F\}D|C\{G\}$, or sometimes with the sqiggly brackets around the $D|C$, a syntactic construct which Olderog calls a *unit*) when every free $callX$ in the command C has its procedure variable X appearing (necessarily uniquely) in the declaration D. *This last assumption is required in the previous rule.*

We still need to explain the H and the $\rightarrow$ in the rule, as well as the $\rightsquigarrow$ in its condition; and then give the semantics and the proof of validity.

A finite set containing $1^{\underline{\text{st}}}$order formulas and/or Floyd-Hoare assertions, including, with obvious notational conventions, both the empty set case and singleton set case, is denoted H or H', etc. And a proof line always has the form $H \rightarrow H'$ (with the studious avoidance of regarding the arrow as a propositional connective, or, horror of horrors, introducing a negation symbol). So the previous rule now makes sense syntactically. Its extra condition with the $\rightsquigarrow$ merely means that the occurrence of X (necessarily only once) in D is in a connected subsequence $\cdots X;B\cdots$; that is, it is followed immediately by "$;B$".

Now we need to give the semantics for everything in sight :

A rule being *sound* means:

if all antecedents $H \to H'$'s are $\mathbf{N}$*-valid,*
then so is the consequent $H \to H'$;

where : to say

$H \to H'$ is $\mathbf{N}$-*valid* means : $\forall n \geq 0, [H \mathtt{tt} n\mathbf{N} \implies H' \mathtt{tt} n\mathbf{N}]$;

where :

$H\mathtt{tt}n\mathbf{N}$ means : $\forall h \in H,\ h\mathtt{tt}n\mathbf{N}$;

where :

$h\mathtt{tt}n\mathbf{N}$ means :

when $h = F$, a $1^{\underline{\text{st}}}$order formula, just that $F\mathtt{tt}\mathbf{N}$; but
when $h = FD|CG$, an F-H assertion, that, for all states s and s',
[$F\mathtt{tt}@s$ and $(s, s') \in m_n(D|C)$]implies that $G\mathtt{tt}@s'$;

where :

$m_n(D|C)$ is defined by induction on $n \geq 0$, and for each n by structural induction on C, as follows. (Note that these are repeats of all the standard definitions, except perhaps the recursion one, which is the simple index shift from part I here. Olderog gets this as a consequence, not a definition, which for his hornet's nest of difficulties with declared variables interacting peculiarly with (parametered) recursive commands, requires a more nuanced definition. That definition expresses it basically as unfoldings in which the 'copying' of the "copy rule" from the title is referring to how the body is to be substituted for the procedure variable. But it is the parameters and declared variables that cause all the trouble, and we are avoiding that here.)

Definition of $m_n(D|C)$:

(1) $m_n(D|y \leftarrow t) \ := \ \{(s, s_{y \mapsto s(t)}) \mid s \in \mathcal{S}te\}$ (independent of D and n) .

(2) $m_0(D|callX) \ := \ \emptyset$ and, for $n \geq 1$, define $m_n(D|callX) \ := \ m_{n-1}(D|B)$ where $D|callX \rightsquigarrow D|B$ —(recall the condition on $D|C$'s, so B exists.)

(3) $m_n(D|(C'; C'')) \ := \ m_n(D|C'') \circ m_n(D|C')$.

(4) $m_n(D|ite(H)(C')(C'')) \ := \ \{(s, s') \mid H\mathtt{tt}@s$ and $(s, s') \in m_n(D|C')$; or
$H\mathtt{ff}@s$ and $(s, s') \in m_n(D|C'')\}$.

(5) $m_n(D|\nabla D_0 C) \ := \ m_n(D_1|C)$, where D_1 is gotten by 'adding' D_0 to what you get by erasing from D all 1-variable sub-declarations with procedure variable occurring in D_0. (So 'what a procedure variable is declared to be in D_0 takes precedence over what it is declared to be in D'. This and (2) are simple but, to me anyway, subtle.)

Proof of validity of the recursion rule:

Assume, in hot pursuit of a contradiction, that the consequent is not **N**-valid, despite the antecedent being so. In particular, looking at the consequent, for some $n \geq 0$, $H\mathtt{tt}n\mathbf{N}$, but $FD|callXG \not\mathtt{tt}n\mathbf{N}$. We may assume that n is smallest possible for that to hold. So choose a pair (s, s') of states which is in $m_n(D|callX)$ (so $n > 0$) with $F\mathtt{tt}@s$, but $G\mathtt{ff}@s'$. We thus have that $(s, s') \in m_{n-1}(D|B)$, where $D|callX \rightsquigarrow D|B$. Returning now to the antecedent, its 'truth in **N**' and the fact that $FD|BG \not\mathtt{tt}(n-1)\mathbf{N}$ (from three statements just above), we must have the union set at the left-hand end of the rule being $\not\mathtt{tt}(n-1)\mathbf{N}$. But because $H\mathtt{tt}n\mathbf{N}$, we also have $H\mathtt{tt}(n-1)\mathbf{N}$ [immediate from $m_{n-1}(D|C) \subset m_n(D|C)$ for all C and D, an easy induction on C from the definition of m_n], and so we get $FD|callXG \not\mathtt{tt}(n-1)\mathbf{N}$. This plus $H\mathtt{tt}(n-1)\mathbf{N}$ contradicts the minimality of n, completing the proof.

So this is really quite easy, once it has been set up correctly; it's pretty much identical to the argument in part I above for the "rigid body" simplification. See the paragraph after next concerning a possible analogue for *total* correctness.

We should note the fact that these "correctness phrases" do give what 'program correctologists' really want : Assume that C has no free $callX$, so that $\emptyset|C$ is a legal unit (in Olderog's terminology). His entire proof system and proof of Cook-completeness assumes this 'legality' for all units considered. Then the semantics of $F\emptyset|CG$ is easily seen to agree with that of the more usual Floyd-Hoare assertion FCG, which itself is the same as the universal closure of the wfd $F \rightarrow [C]G$ from dynamic logic. And that's all we need to know, as long as the semantics of ∇ and *call* agree with your preconceived notions—which I cannot know!

By 1/5th of [Clarke's incompletability theorem plus the five examples showing its assumptions cannot be weakened], one has, for a suitably complicated command language with "fixed body" recursion, a Cook-complete proof system for partial correctness such that no such system can exist if nested recursion is instead allowed. So it seems that being careful about casual claims of generalizability in this direction—fixed body to nested—is a reasonable attitude. On the other hand, it is possible, even likely, that the intentions, in **[Apt]**, **[AdB]** and **Nip]**, were that the treatment of *total* correctness was supposed to be 'pumpable up' in a way that would give something for total correctness similar to the above vast simplification of **[Old]** for partial correctness. So that those treatments, which had fixed declarations and definitely no nesting of recursion, would then generalize to give nesting. This may very well be doable for the latter two. (As said earlier, the first of

those three is in any case an unsound system for total correctness.) This pumping up is not an entirely trivial process, introducing "units" etc., and is in any case not my job, but rather the authors', if that is what was intended. So my 'priority' claim in [**mutu**] seems an acceptable one—unless some kindly reader can find for me another reference where that job had been done. That claim was to have presented the first correct and complete system for *total* correctness of a self-contained (to exclude [**Harel**], as the others do, and, in any case, his system isn't quite Cook-complete, cf. [**while**], Appendix) language and reasonable semantics for recursive commands *with nesting*. ["Self contained" also seems a good idea, since subsets of recursively enumerable sets don't generally share that property (as most readers will know, so I better stop preaching). Note that this question of decidability is not a problem with Harel's setup.]

III. Other partial correctness recursion rules for $\mathcal{R}ec_{1\pm}$.

Here we deal with syntax which is equivalent to what sometimes is called the "μ-calculus". My choice to use '∇' in place of 'μ' (flippantly perhaps) is because (1) the latter is more often associated with fixed point semantics, and separating semantic and syntactic aspects helps (my limited intellect at least) to stay on track; and (2) what could be mistaken for an upside-down 'delta' seems to go well with the backward 'E' and the upside-down 'A' for this syntax, where ∇ does behave in many ways like a quantifier ('acting on' procedure variables, not $1^{\underline{\text{st}}}$order variables).

In any case, we are trying for a reasonably simple syntax, in contrast to the last part, avoiding separate declarations, and more particularly, avoiding the "units" occurring in the proof rules in [**Old**].

With that desideratum in mind, except for the ingenious, idiosyncratic (and generally unimitated because it needs to use a huge panoply of extra non-deterministic commands—for Cook-completeness it also needs a correction to the rule for total correctness of recursive commands, as alluded to above) system in [**Harel**] (which we won't comment any more on here), for a serious look at correctness properties of recursion *with local declarations* (in particular admitting nested recursion and in contradistinction to the syntax/semantics of part I of the present section), [**Musk**] and [**deBa**] seem to be the only sources besides my recent work (and [**Old**] and similar for strictly partial correctness, as discussed in part II). Neither [**Musk**] nor [**deBa**] deals at all with total correctness, nor with the completeness (á la Cook) of the systems they present for partial correctness. Earlier in [**deBa**], Cook-completeness of a system for recursion *without nesting* is discussed. It may be worth pointing out now that those systems (resp: Section 8 on p.24, and results 7.12 and 7.16) have no hope of such a completeness property in the normal sense: Neither has a 'rule for dealing with $\{F\}X\{G\}$'. So a simple (non-sentence) Floyd-Hoare assertion such as $\{1 \approx 1\}X\{1 \approx 1\}$ can be seen by a straightforward syntactic analysis not to be derivable, despite being true in the "TT-semantics" (see below) of those papers (and also in the "tt-semantics", for that matter). Here 'just-plain X' is really the same as *call*X in [**sing**], and "non-sentence" above refers to the freeness of the procedure variable X in the command inside the Floyd-Hoare assertion. (It is within the realm of possibility that some sort of completeness holds for deriving *strictly sentences* in the above sense, but that would appear to need serious work, if

indeed it's feasible.) In any case, there seems a close tie-up between the lack of a completeness discussion for these systems, and the distinction several paragraphs below between "tt-semantics" and "TT-semantics", where it is only on non-sentences that that distinctness is detectable. (This also relates to the requirement in **[Old]** that the left half of a unit include a declaration for any variable which occurs free in its right half.)

We mainly want to study the partial correctness rule for recursive commands which both of these authors use, and compare it to the rule in **[sing]**. Here is the rule as we find it in **[Musk]** :

$$(\mathsf{M})_1 \qquad \frac{F \to [|D|^{X \mapsto gigabug}]G \ \ , \ \ (F \to [D]G) \ \vdash \ (F \to [|D|^{X \mapsto C}]G)}{F \to [\nabla C/XD]G} \ .$$

where the newly minted string $\nabla C/XD$ has C as the body declared for X, but D as the command 'performed recursively'. We can get a version of that command, so to speak, simply by substituting ∇XC for free occurrences of $callX$ in D. So, by letting $D = C$, the rule above essentially specializes to

$$(\mathsf{M}) \qquad \frac{F \to [|C|^{X \mapsto gigabug}]G \ \ , \ \ (F \to [C]G) \ \vdash \ (F \to [|C|^{X \mapsto C}]G)}{F \to [\nabla XC]G} \ .$$

More accurately, the above is what you get from **[Musk]** by retracting from mutual recursion to single procedure variable recursion (and by using my notation and expressing things in the language of dynamic logic). Also, as indicated earlier, the command language in **[Musk]** is the one with the standard union-constructor (more-or-less in place of if-then-else), the usual non-deterministic behaviour of $\cup$ being the intended semantics. These variations make no substantial difference to what follows, nor does simplifying (M) to its (more-or-less) special case

$$\frac{[|C|^{X \mapsto gigabug}]G \ \ , \ \ ([C]G \ \vdash \ [|C|^{X \mapsto C}]G)}{[\nabla XC]G} \ .$$

No discussion of (Cook-)completeness is attempted by **[Musk]**. He produces the semantics by translating everything into a form of higher order type-logic, rather sophisticated.

The previous paragraph suffers from a giant gap. A rule of inference such as those above is meaningless (or at least its validity is) in isolation from the

system of rules it is meant to be part of, since the symbol "$\vdash$" appears with a wfd to its left in an antecedent. As long as we are referring directly to his paper, the system defining $\vdash$ overall is of course just the one in [**Musk**], including the mutual recursion generalization of the earlier rule above (so there's some self-reference). And really, the rules should be translated back into partial correctness Floyd-Hoare notation. We won't need to write out his other rules (Section 8 on p.24) for what is below. Soon we'll consider changing the symbol "$\vdash$" in the 2nd antecedent of the above rule to the symbol "$\vdash'$" as defined in [**sing**] (and also changing it in various other ways), in order to compare versions of Muskens' rule with the rule in [**sing**], keeping all other rules the same for the comparison, and also in order to consider the question of validity.

In the book [**deBa**], early on we find a lengthy chapter on the severely restricted form of recursion (i.e. essentially global declarations, or at least no nesting) studied in part I of this section. Later there is a partial correctness proof system for a language as in [**Musk**] above, but with much less analysis than for the restricted command language. No discussion of (Cook-) completeness is attempted in the later chapter. The semantics is cumulative, and presented in a rather exhausting way, spread out over two lengthy sets of consecutive pages in the earlier chapter (pp. 131-161, 189-197), then supplemented by yet a third (pp. 261-270)in the later one where local declarations and nesting are permitted. There the basic rule for partial correctness (total correctness is discussed very little) of recursive commands is pretty much the same as $(\mathsf{M})_1$ above. However, it is expressed in terms of the "correctness phrases" mentioned earlier, so one would need a careful analysis of those 30 to 40 pages of semantics definitions to satisfy oneself completely about the essential sameness of [**deBa**] and [**Musk**] in these respects. I'd prefer to leave that to other times or other people, satisfied for now at least that the analysis below is largely relevant to [**deBa**], not just [**Musk**]. So no further reference to [**deBa**] will be made here, though the semantics there does also seem to have the interesting aspect we now want to analyze.

We shall now discuss a distinction, in defining truth for a wfd, which seems to me of some interest. The new regime will be denoted by TT (versus the previous tt). The crucial clause for the latter was

$$< C > \mathcal{A} \;\; \mathsf{tt}@s \iff \exists s' \;\text{ with } (s, s') \in \mathcal{M}(C, \Phi) \text{ and } \mathcal{A}\ \mathsf{tt}@s' \ .$$

Here Φ maps all procedure variables to the empty set (of pairs of states!).

Contrast that with the definition

$$< C > \mathcal{A} \;\; \mathsf{TT}@s \iff \forall \Theta\ \exists s' \;\text{ with } (s, s') \in \mathcal{M}(C, \Theta) \text{ and } \mathcal{A}\ \mathsf{TT}@s' \ .$$

For the standard Tarski phrases applying to the other parts of the syntactic construction of wfds, there is no difference in the definitions of $\mathsf{TT}@s$ and $\mathsf{tt}@s$. And, as with tt, by the assertion "$\mathcal{A}\ \mathsf{TTN}$", we simply mean "$\mathcal{A}\ \mathsf{TT}@s$ for all states s". Then a crucial point here is to say that a rule of inference being tt-valid will be defined to mean exactly the same as before, whereas (analogously) the rule is TT-valid if and only if, when all its ordinary antecedents are TTN, and its 'conditional $\vdash$-antecedent' (if any) is actually the case, then its consequent is TTN.

Note that $\mathsf{TT}@s$ and $\mathsf{tt}@s$ agree on wfds $\mathcal{A}$ whose subcommands have no free procedure variables. Prove it by an induction on $\mathcal{A}$ whose only non-trivial case, $\mathcal{A} =< C > \mathcal{B}$, follows inductively from the definition and the fact that $\mathcal{M}(C, \Theta) = \mathcal{M}(C, \Phi)$ because Θ and Φ agree on all the (non-existing!) free procedure variables in C. See **1.3**.

Note that the "units" in part II are in some sense a way of getting round this semantic distinction, more-or-less forcing everything considered to be a sentence. The position I'm taking here is that maybe it's preferable to simplify the semantics and "bite the bullet" about deciding to give every command a semantics, i.e. ones with free procedure variables as well.

Before going further, here are a couple of trivial examples to show that, in general, neither form of validity implies the other form :

$$[callX]1 \approx 0 \;/\; [bugga]1 \approx 0 \quad \text{is } \mathsf{TT}\text{-valid, but not } \mathsf{tt}\text{-valid .}$$

$$[bugga]1 \approx 1 \;/\; [callX]1 \approx 0 \quad \text{is } \mathsf{tt}\text{-valid, but not } \mathsf{TT}\text{-valid .}$$

I find it odd that other authors don't point out this distinction between two semantic possibilities, especially those who use 'TT-semantics', such as [**Musk**] and [**deBa**]. It seems rather obvious to me that tt-semantics is the one which relates to the 'real world'

of program verification; but I'm not a computer scientist, so I may very well be missing some crucial point here. On the other hand, the above possibility for a partial correctness recursion rule can perhaps only be compared with the one in [**sing**] with the use of $\vdash'$, rather than the full $\vdash$ so to speak, in the main antecedent, and the use of dynamic logic language, with, as indicated above, only the tt milieu making sense for the semantics. But the tables below seem to preclude much comparison at all.

Let us now also simplify the discussion by avoiding the most inane subtleties to do with substitution for procedure variables. The technicalities in Section 3 of this paper could be transferred here, but that would obscure the points we are making. So, in effect, we are backing off from $\mathcal{R}ec_{1+}$ to $\mathcal{R}ec_{1-}$, which means allowing only simple recursion. That's what's in [**sing**] anyway.

Below is a pair of tables which tabulate information on mainly the validity of the following rule, essentially from [**Musk**] and [**deBa**], not quite identical to (M) because of the occurrence of the symbol $\vdash^{??}$:

$$\frac{F \to [|C|^{X \to gigabug}]G \quad , \quad (F \to [C]G) \ \vdash^{??} \ (F \to [|C|^{X \to C}]G)}{F \to [\nabla XC]G} \ ,$$

or of its Floyd-Hoare partial correctness notation version

$$\frac{\{F\}|C|^{X \to gigabug}\{G\} \quad , \quad \{F\}C\{G\} \ \vdash^{??} \ \{F\}|C|^{X \to C}\{G\}}{\{F\}\nabla XC\{G\}} \ ,$$

in the eight cases which now arise naturally. The 'derivability verb' $\vdash^{??}$ in these gives one parameter (the rows), the bottom row referring to the full system's verb, the row above it being as indicated. The other parameter within the tables (the columns) has the two choices of semantics described a few paragraphs above. The 3rd parameter (the two tables) has the first table using the dynamic logic language in [**sing**] plus all the other rules there. And instead, Floyd-Hoare partial correctness notation in [**Musk**] and the remainder of his system for defining $\vdash^{??}$ is operative in the lower table.

Dynamic logic system for $\vdash$ and $\vdash'$ in [**sing**] (except its rule for a typical [*recursive command*], i.e. $[\nabla XC]$, is replaced by the first one just above, which is what the table comments on)

	tt	TT
$\vdash'$	invalid, e.g. below	silly, a $\vdash'$ axiom is not **TTN**
$\vdash$	invalid, same e.g.	silly, same $\vdash$ axiom not **TTN**

Floyd-Hoare system for $\vdash$ in [**Musk**] (and the verb $\vdash^!$ means "is derivable by all except that rule", that rule being the second one above, which is what the table comments on)

	tt	TT
$\vdash^!$	likely valid, but incomplete ; (invalid with callrule)	valid, see immediately below
$\vdash$	likely valid, but incomplete ; (invalid with callrule)	valid [**Musk**], system incomplete

So all eight entries give one pause.

First let us explain the right-hand column in the upper table, and the conjectured column to the left in the lower table. In [**sing**], $\neg < callX > \mathcal{A}$ is an axiom in the definition of the system for $\vdash'$ and also for $\vdash$. It is not valid in the TT-sense. For example $\neg < callX > 1 \approx 1$ is not **TTN**. So TT-validity in the top right of the upper table is essentially meaningless, or at least uninteresting, since having a rule in the system for $\vdash'$ which is TT-invalid makes having a recursion rule with that as part of its antecedents rather pointless. And similarly for the box immediately below it.

As for the lower table, once we give some sort of elementary depth-style proof of the facts in the rightmost column of the lower table, the validity but incompleteness in its left column most probably follows similarly. The validity claim on the bottom-right of the lower table is simply imported from [**Musk**], though, as suggested above, it ought to be deducible in a more elementary way. The incompleteness was explained near the beginning of this part. The top-right of the lower table follows from the bottom-right.

Referring back to the 2nd previous paragraph, $[callX]\mathcal{A}$ would be an equivalent form of the offending axiom there. Then an analogous, obviously tt-valid, Floyd-Hoare partial correctness axiom applicable to the left-hand column of the lower table is $\{F\}callX\{G\}$ for any F, X, G. Adding that to Muskens' system might get us a good ways towards completeness, but unfortunately validity would then slip out of our grasp, using an almost

identical example to the one just below for explaining the left column of the upper table.

Let us now turn to that example. Let x, y and z be three distinct program (i.e. $1^{\underline{\text{st}}}$order) variables. Let $D := |C|^{X \to C}$ where

$$C \ := \ ite(y \approx 0)(bugga)(\text{“}y \dot{\leftarrow} y \dot{-} 1\text{”} \ ; \ x \dot{\leftarrow} x + 1 \ ; \ callX) \ .$$

The quoted subcommand is an abbreviation for a ‘proper decrementation of y’-command (that is, fix 0 and decrease all else by 1), and we've left out two pairs of brackets around some sequencing, which can be inserted either way. The following three facts are intuitively obvious (leaving out a few more pairs of brackets) :

$$(\ x \approx z \ \to \ [C] \ x < z + 1 \ ) \ \ \mathsf{tt}\mathbf{N} \ ;$$

$$(\ x \approx z \ \to \ [D] \ x < z + 1 + 1 \ ) \ \ \mathsf{tt}\mathbf{N} \ ;$$

$$(\ x \approx z \ \to \ [\nabla XC] \ x < z + 1 + 1 \ ) \quad \not{\mathsf{tt}}\mathbf{N} \ ;$$

As to the first two, it seems reasonable to expect (and we'll prove in considerable detail in the appendix to this section) that we have even more, so to speak :

$$\vdash' \ (\ x \approx z \ \to \ [C] \ x < z + 1 \ ) \ ;$$

$$\vdash' \ (\ x \approx z \ \to \ [D] \ x < z + 1 + 1 \ ) \ .$$

Taken together with the proof in the appendix of the third line of the penultimate display, plus the truth in $\mathbf{N}$ of $x < z + 1 \ \to \ x < z + 1 + 1$, and the fact that C and $|C|^{X \to gigabug}$ have the same tt-semantics, we get a counterexample, as required, to the tt-validity of the rule (M)

$$\frac{F \to [|C|^{X \to gigabug}]G \ \ , \ \ (F \to [C]G) \ \vdash' \ (F \to [|C|^{X \to C}]G)}{F \to [\nabla XC]G} \ ,$$

(and so to the rule in [**Musk**]) by taking $F = x \approx z$ and $G = x < z + 1 + 1$.

To finish, here are a few miscellaneous facts.

(a) As noted earlier for tt, without losing validity in either sense above (and so losing any interest whatsoever in the rule), in **[sing]** the relation $\vdash'$ cannot be changed to just $\vdash$ in its rule for partial correctness of recursive commands, a simplified version of the rule, analogous to (M), being

$$(\mathsf{S}) \qquad \frac{(F \to [\nabla XC]G) \;\vdash'\; (F \to [|C|^{X \to \nabla XC}]G)}{F \to [\nabla XC]G} \quad .$$

(b) The first antecedent in Muskens' rule (M) cannot be dropped. This is clear from any example where X does not occur in C, and where $[C]G$ is not TTN.

(c) Note that, for the (Cook-) completeness of the system in **[sing]**, we must actually use [souped-up, generalized (S)]=:$(\mathsf{S}+)$ below. So, for any hope of some kind of completeness theorem using Muskens' stuff and applying to wfds with no free procedure variables (as alluded to earlier), one would possibly need a [souped-up, generalized (M)]=:$(\mathsf{M}+)$. Here are the "souped up" versions:

$$(\mathsf{M}+) \quad \frac{\vdash \mathcal{C}\ \forall \mathcal{C} \in \Omega \;\;,\;\; F \to [|C|^{X \to gigabug}]G \;\;,\;\; \Omega \cup \{F \to [C]G\} \;\vdash'\; F \to [|C|^{X \to C}]G)}{F \to [\nabla XC]G}$$

$$(\mathsf{S}+) \qquad \frac{\vdash \mathcal{C}\ \forall \mathcal{C} \in \Omega \;\;,\;\; \Omega \cup \{F \to [\nabla XC]G\} \;\vdash'\; F \to [|C|^{X \to \nabla XC}]G}{\vdash F \to [\nabla XC]G}$$

In these, Ω is a finite set of wfd's $\mathcal{C}$ which don't have ∇XC in them. Actually F and G can more generally be wfd's (not merely $1^{\underline{\text{st}}}$order formulas), again as long as they don't have ∇XC in them, but that stronger rule is overkill from the completeness point-of-view. One of the longer loose ends in this paper is to work out whether such a completeness result holds.

(d) Except for the unfortunate unsoundness, one can see how, in some sense, to derive my rule from Muskens'. To super-simplify, spill the soup and even dump the "$F \to$" in front of everything in both rules. Then proceed as follows: Assume the antecedent in 'my super-simplified' rule, i.e. $[\nabla XC]G \;\vdash'\; [|C|^{X \to \nabla XC}]G$. Then, from **[sing]**, Theorem 11, we have the result $[E]G \;\vdash'\; [|C|^{X \to E}]G$ for any command E whose program variables are

already ones in C. (See the appendix.) Now $[gig]G$ is derivable, and therefore, so is Muskens' first antecedent $[|C|^{X \mapsto gig}]G$ by letting $E = gig$. And so is the second antecedent, $[C]G \ \vdash' \ [|C|^{X \mapsto C}]G$, by letting $E = C$. 'Unsuper-simplifying', the same argument works to derive the antecedents in Muskens' from mine, and thereby, to 'justify' mine from his. However, the invalidity of Muskens' for the relevant semantics of **[sing]** spoils everything it seems, since in the language of dynamic logic, we are back to normal non-awkward logic (as opposed to the Floyd-Hoare straightjacket), so with normal propositional connectives etc., an unsound rule will quickly derive *anything*.

(e) Conversely, rule (M+) would be derivable within the system in **[sing]** since that system is complete [in the tt-sense, as proved in **[sing]**—perhaps one could say "derivable from rule (S+)"]—if and only if the former rule is tt-valid, which, unfortunately, it isn't. Actually, if we *could* derive rule (M+) from rule (S+) directly, that would be also a proof of the former's tt-validity, so we *can't*.

(f) As a side issue, the use of Ω in 'souping-up' the rule (S) above is something one first suspects to be redundant, as some have tried to point out. But note that it is the full $\vdash$ applying to elements of Ω, but only $\vdash'$ in the main antecedent, so the naive argument that we can dispense with the "souping-up" is not going to work. Indeed, in many of the examples from the second section of **[sing]**, it is exactly that fact which is crucial. I doubt that one would have (Cook-)completeness if the Ω were deleted (equivalently, empty), but, in **[while]**, I have already done my bit in sweating out a syntactic non-completeness proof to counter a naive referee, and have no desire for a second banishment to the sweat-shop.

Appendix to Section 5: The system from [sing] for $\vdash'$ (and a few derivations and non-derivations).

Here are the $\vdash'$-rules, mostly axioms:

$$(\mathsf{ORACLE}) \quad \emptyset \ / \text{ any } 1^{\underline{\text{st}}} \text{ order formula that is true in } \mathbf{N}$$

$$(\mathsf{MP}) \qquad \mathcal{A}\ ,\ \mathcal{A} \to \mathcal{B} \ / \ \mathcal{B}$$

$$(\mathsf{TAUT}) \qquad \emptyset \ / \ \mathcal{A} \qquad \text{for every (propositional) tautology } \mathcal{A}.$$

$$(\mathsf{AX})_{<\leftrightarrow>} \qquad \emptyset \ / \ (< x \doteq t > F \ \leftrightarrow \ |F|^{x \to t})$$

$$(\mathsf{AX})_{<;>} \qquad \emptyset \ / \ (< (C;D) > \mathcal{A} \ \leftrightarrow \ < C >< D > \mathcal{A})$$

$$(\mathsf{AX})_{<ite>} \qquad \emptyset \ / \ (< ite(G)(C)(D) > \mathcal{A} \ \leftrightarrow \ (G \to < C > \mathcal{A}) \wedge (\neg G \to < D > \mathcal{A}))$$

$$(\mathsf{AX})_{[ite]} \qquad \emptyset \ / \ ([ite(G)(C)(D)]\mathcal{A} \ \leftrightarrow \ (G \to [C]\mathcal{A}) \wedge (\neg G \to [D]\mathcal{A}))$$

$$(\mathsf{UNAR})_{\forall} \qquad (\mathcal{A} \to \mathcal{B}) \ / \ (\forall x\ \mathcal{A} \to \forall x\ \mathcal{B})$$

$$(\mathsf{UNAR})_{< >} \quad (\mathcal{A} \to \mathcal{B}) \ / \ (< C > \mathcal{A} \to < C > \mathcal{B})$$

plus any subset of the following rules which happens to allow one to derive the ones left out from those selected plus those just above :

$$(\mathsf{AX})_{<call>} \qquad \emptyset \ / \ \neg < callX > \mathcal{A}$$

$$(\mathsf{AX})_{disj} \qquad \emptyset \ / \ (\mathcal{A} \to [D]\mathcal{A})$$

if $\mathcal{A}$ and D have no common variable.

$$(\mathsf{AND}) \quad \mathcal{A} \to [D]\mathcal{B}\ ,\ \mathcal{A}' \to [D]\mathcal{B}' \ / \ (\mathcal{A} \wedge \mathcal{A}' \to [D](\mathcal{B} \wedge \mathcal{B}'))$$

$$(\mathsf{PRE}) \qquad (\mathcal{A} \to \mathcal{B}) \ / \ ((\exists z \mathcal{A}) \to \mathcal{B})$$

where the variable z is not in $\mathcal{B}$.

$$(\mathsf{SUB})_1 \quad \mathcal{A} \ / \ |\mathcal{A}|^{u \to t}$$

where the variable u does not occur as a program variable in any command within $\mathcal{A}$.

$$(\mathsf{SUB})_2 \quad F \to \mathcal{B} \ / \ (|F|^{z \to x} \to \mathcal{B})$$

where the variable z is not in $\mathcal{B}$.

$$(\mathsf{AX})_{<\to>} \qquad \emptyset \ / \ ((\mathcal{A} \to < C > \mathcal{B}) \ \to \ (\mathcal{A} \wedge \mathcal{D} \to < C > (\mathcal{B} \wedge \mathcal{D})))$$

when $\mathcal{D}$ and C have no variables in common.

A 'rule' $(\mathsf{AX})_{[\rightarrow]}$ (use [ ] in place of < > in the rule just above) is valid, and can be easily derived from $(\mathsf{AX})_{disj}$ and (AND).

Here is a propositionally derivable rule, hypothetical syllogism, [i.e. derivable simply from (MP) and (TAUT)] to which we shall have occasion to refer:

$$(\mathsf{HYSY}) \qquad \mathcal{A} \rightarrow \mathcal{B} \ , \ \mathcal{B} \rightarrow \mathcal{C} \ / \ \mathcal{A} \rightarrow \mathcal{C} \ .$$

The following are also easily derivable from the (UNAR)-rules and others:

$$(\mathsf{AX})_{[call]} \qquad \emptyset \ / \ [callX]\mathcal{A}$$

$$(\mathsf{UNAR})_{\exists} \qquad (\mathcal{A} \rightarrow \mathcal{B}) \ / \ (\exists x \ \mathcal{A} \rightarrow \exists x \ \mathcal{B})$$

$$(\mathsf{UNAR})_{[\ ]} \qquad (\mathcal{A} \rightarrow \mathcal{B}) \ / \ ([C]\mathcal{A} \rightarrow [C]\mathcal{B})$$

Finally, each of $(\mathsf{AX})_{<;>}$ and $(\mathsf{AX})_{<\leftrightarrow>}$ rather easily implies the analogous rule, $(\mathsf{AX})_{[;]}$ or $(\mathsf{AX})_{[\leftrightarrow]}$, which are obtained by just changing all occurrences of < > to [ ] . The same holds for *ite*, but maybe without the "rather easily" for beginners, so we've written it down explicitly as part of the system.

One result from [**sing**] used in a minor way a few pages back is

Theorem 11. *Suppose* $\Gamma \vdash' \mathcal{A}$. *Then* $|\Gamma|^{\nabla XC \rightarrow E} \vdash' |\mathcal{A}|^{\nabla XC \rightarrow E}$ *for any commands* C *and* E *for which all the variables in* E *already occur in* C.
By the set $|\Gamma|^{\nabla XC \rightarrow E}$ here, we mean $\{ \ |\mathcal{B}|^{\nabla XC \rightarrow E} \ : \ \mathcal{B} \in \Gamma \ \}$. See [**sing**] for defining $|\mathcal{B}|^{\nabla XC \rightarrow E}$.

The proof is simply to take any derivation witnessing $\Gamma \vdash' \mathcal{A}$, and replace each line $\mathcal{B}$ by $|\mathcal{B}|^{\nabla XC \rightarrow E}$, noting that the $|\ |^{\nabla XC \rightarrow E}$-operation carries $\vdash'$-rules to $\vdash'$-rules, and so the replacement is indeed a $\vdash'$-derivation. The proof depends strongly on the fact that no 'explicit rules for ∇X' are included in the system for $\vdash'$. Thus, one certainly cannot generalize by replacing ∇XC by a more general command. The condition on the variables in E is needed so as not to run into a problem with those rules which have a restriction placed on variables. The detailed proof of **11** is in Section 3 of [**sing**].

The rest of this appendix (and the paper) consists of the proofs promised on page 48.

A simple but messy derivation. Our first job here is to 'give' a derivation witnessing :

$$\vdash' \ (\ x \approx z \ \rightarrow \ [C] \ x < z+1 \ )$$

where

$$C \ := \ ite(y \approx 0)(bugga)(\text{“}y \dot{\leftarrow} y \dot{-} 1\text{”} \ ; \ x \dot{\leftarrow} x+1 \ ; \ callX) \ .$$

By $(\mathsf{AX})_{[ite]}$ and (TAUT)/(MP) (appeals to the latter just phrased "propositionally" henceforth), this reduces to derivations for

$$(1) \qquad \vdash' \ y \approx 0 \ \wedge \ \ x \approx z \ \rightarrow \ [bugga] \ x < z+1 \ .$$

and

$$(2) \qquad \vdash' \ \neg y \approx 0 \ \wedge \ x \approx z \ \rightarrow \ [\text{“}y \dot{\leftarrow} y \dot{-} 1\text{”} \ ; \ x \dot{\leftarrow} x+1 \ ; \ callX] \ x < z+1 \ ;$$

The argument in (2) gets rather grundgy, though very trivial, so it may seem a hard way merely to establish the truth in **N** of the wfs. But we'll use this to short-circuit the corresponding argument in the next example, and that's one where a $\vdash'$-derivation really does need to be proved to exist.

As for (1), take *bugga* to be $x_0 \dot{\leftarrow} x_0$, and use lots of (HYSY) applied to

$$y \approx 0 \wedge \ x \approx z \ \rightarrow \ x \approx z \ \rightarrow \ x < z+1 \ = \ |x < z+1|^{x_0 \rightarrow x_0} \ \rightarrow \ [x_0 \dot{\leftarrow} x_0] \ x < z+1 \ .$$

That is, each 'arrow' is derivable, using (ORACLE) twice and $(\mathsf{AX})_{[\dot{\leftarrow}]}$ once.

To do (2) more elegantly, one could simply claim that the following holds:

$$\vdash' \quad [(C \ ; \ callX)] \ G$$

for any C, X and G. But that isn't quite true: one wants C not to involve ∇ to get it for $\vdash'$, not merely $\vdash$. So let's just do it in the particular case at hand.

Start with the 'axiom' $[callX]x < z+1$. Now here's a sequence of wfds which can be $\vdash'$-derived. The lines not justified by the word "propositionally" will be explained below.

$$1 \approx 1 \to [callX]x < z+1 \quad ; \quad [x \leftarrow\!\div x+1]1 \approx 1 \to [x \leftarrow\!\div x+1][callX]x < z+1 \quad ;$$

$$1 \approx 1 \to [x \leftarrow\!\div x+1]1 \approx 1 \quad ; \quad 1 \approx 1 \to [x \leftarrow\!\div x+1][callX]x < z+1 \quad ;$$

$$[x \leftarrow\!\div x+1][callX]x < z+1 \to [(x \leftarrow\!\div x+1 \; ; \; callX)]x < z+1 \quad ;$$

$$1 \approx 1 \to [(x \leftarrow\!\div x+1 \; ; \; callX)]x < z+1 \quad ;$$

$$[\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}]1 \approx 1 \to [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}][(x \leftarrow\!\div x+1 \; ; \; callX)]x < z+1 \quad ;$$

$$1 \approx 1 \;\wedge\; 0 < y \;\to\; [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}]\; 1 \approx 1 \quad ;$$

$$1 \approx 1 \;\wedge\; 0 \approx y \;\to\; [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}]\; 1 \approx 1 \quad ;$$

$$1 \approx 1 \quad \longrightarrow \quad (1 \approx 1 \;\wedge\; 0 < y) \;\vee\; (1 \approx 1 \;\wedge\; 0 \approx y) \quad ;$$

$$1 \approx 1 \to [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}]\; 1 \approx 1 \quad ;$$

$$1 \approx 1 \to [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}][(x \leftarrow\!\div x+1 \; ; \; callX)]x < z+1 \quad ;$$

$$[\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}][(x \leftarrow\!\div x+1 \; ; \; callX)]x < z+1 \to [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”} \; ; \; x \leftarrow\!\div x+1 \; ; \; callX]\; x < z+1 \quad ;$$

$$1 \approx 1 \to [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”} \; ; \; x \leftarrow\!\div x+1 \; ; \; callX]\; x < z+1 \quad ; \quad 1 \approx 1$$

$$[\text{“}y \leftarrow\!\div y \dot{-} 1\text{”} \; ; \; x \leftarrow\!\div x+1 \; ; \; callX]\; x < z+1 \; ,$$

from which the result is propositionally derived very quickly.

As for non-propositional justifications of the mess above:
2nd, 7th wfds from $(\mathsf{UNAR})_{[\;]}$;
3rd wfd from $(\mathsf{AX})_{[\leftarrow\!\div]}$;
5th, 13th wfds from $(\mathsf{AX})_{[;]}$;
8th wfd from page 46 in [**sing**] ;
9th wfd could have been done even more easily there ;
10th, 15th wfds from (ORACLE) .
(Sorry for the unenlightening enumerating labour.)

A less simple but still messy derivation. The next job is to 'witness' [writing the symbol 2 for the string (1+1)] ,

$$\vdash' \ (\ x \approx z \ \rightarrow \ [D] \ x < z + 2 \ ) \ ,$$

where

$$D \ := \ |C|^{X \to C} \ = \ ite(y \approx 0)(bugga)(\text{“}y \leftarrow\!\div y \dot{-} 1\text{”} \ ; \ x \leftarrow\!\div x + 1 \ ; \ C) \ \ .$$

By $(\mathsf{AX})_{[ite]}$ and "propositionally", this reduces to derivations for both

$$(1)' \qquad \vdash' \ y \approx 0 \ \wedge \ \ x \approx z \ \rightarrow \ [bugga] \ x < z + 2 \qquad \text{and}$$

$$(2)' \qquad \vdash' \ \neg y \approx 0 \ \wedge \ \ x \approx z \ \rightarrow \ [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”} \ ; \ x \leftarrow\!\div x + 1 \ ; \ C] \ x < z + 2 \ \ .$$

In view of (1) in the previous example, no more need be said about $(1)'$. By that example, i.e. just change z to $z + 1$ in it, we have

$$\vdash' \quad x \approx z + 1 \ \rightarrow \ [C] \ x < z + 2 \ \ .$$

So clearly, for $(2)'$, using $(\mathsf{AX})_{[;]}$, it remains only to $\vdash'$-derive

$$\neg y \approx 0 \ \wedge \ \ x \approx z \ \rightarrow \ [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”} \ ; \ x \leftarrow\!\div x + 1] \ x \approx z + 1 \ \ .$$

Similar to the previous example, here are six wfds which can be $\vdash'$-derived:

$$\neg y \approx 0 \ \wedge \ \ x \approx z \ \rightarrow \ [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}] \ x \approx z \quad \text{since} \quad |x \approx z|^{y \to y+1} = x \approx z \ \ ;$$

$$x \approx z \ \rightarrow \ x + 1 \approx z + 1 \ ;$$

$$[\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}] \ x \approx z \ \rightarrow \ [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}] \ x + 1 \approx z + 1 \ ;$$

$$x + 1 \approx z + 1 \ \rightarrow \ [x \leftarrow\!\div x + 1] \ x \approx z + 1 \ \ ;$$

$$[\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}] \ x + 1 \approx z + 1 \ \rightarrow [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}] \ [x \leftarrow\!\div x + 1] \ x \approx z + 1 \ \ ;$$

$$[\text{“}y \leftarrow\!\div y \dot{-} 1\text{”}] \ [x \leftarrow\!\div x + 1] \ x \approx z + 1 \ \rightarrow \ [\text{“}y \leftarrow\!\div y \dot{-} 1\text{”} \ ; \ x \leftarrow\!\div x + 1] \ x \approx z + 1 \ \ .$$

A triple application of (HYSY), using the 1st, 3rd, 5th and 6th lines finishes the job. The justifications of the six lines are, respectively: page 46 in **[sing]** ; (ORACLE) ; $(\mathsf{UNAR})_{[\ ]}$; $(\mathsf{AX})_{[\leftarrow\!\div]}$; $(\mathsf{UNAR})_{[\ ]}$; and $(\mathsf{AX})_{[;]}$.

A non-derivable. The final job is to prove

$$(\ x \approx z \ \rightarrow \ [\nabla XC] \ x < z + 2 \ ) \quad \not\vdash \not\mathsf{tt}\mathbf{N} \ .$$

Let s be a state where the values of x, y and z are 0, 2 and 0 respectively. Let s' be the same state, except the values are 2, 0 and 0 respectively. Then we'll leave it as an exercise for the reader to look up the semantics of ∇-commands and prove rigorously the intuitively obvious fact that $(s, s') \in m(\nabla XC)$. (In fact that pair is in "m" of the 3rd unfolding, ['one unfolding beyond the previous pair of examples'], if unfoldings is how you decide to look at the semantics.) Clearly $x \approx z \ \ \mathsf{tt}@s$, whereas $\ x < z + 2 \ \ \mathsf{ff}@s'$, so that finishes the job.

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.

[**AdB**] America, Pierre and de Boer, Frank *Proving Total Correctness of Recursive Procedures.* Information and Computation 84(2), 1990, 129-162.

[**book**] Hoffman, P. *Logic for the Mathematical.*
papers #9 and #12 on : http://www.math.uwaterloo.ca/PM(underscore) Dept/Homepages/Hoffman/hoffman.shtml .

[**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.

[**Cst**] Cousot, Patrick *Methods and Logics for Proving Programs.* pp. 841-993 in: *Handbook of Theoretical Computer Science*, ed. J. van Leeuwen, Elsevier Science Publishers B. V., 1990.

[**deBa**] deBaaker, Jaco *Mathematical Theory of Program Correctness.* Prentice/Hall International, 1980.

[**End**] Enderton, Herbert A. *A Mathematical Introduction to Logic.* Harcourt/Academic Press, 2001/ 1972.

[**Gor**] Gorelick, Gerald Arthur *A Complete Axiomatic System for Proving Asser- tions about Recursive and Non-recursive Programs.* Tech. Report 75, Dept. of Computer Science, Univ. of Toronto, 1975.

[**H-K-T**] Harel, D., Kozen, D. and Tiuryn, J. *Dynamic Logic.* MIT Press, 2000.

[**Harel**] Harel, David *First-Order Dynamic Logic.* Lecture Notes in CS # 68, Springer, 1979. See also *Correctness of regular deterministic programs.* Theor. Comp. Sci. 12(1980) 61-81

[**Hom**] Homeier et al *USPatent #5,963,739* Slogan: Making Formal Methods into Normal Methods. (As yet, no record of SuperBowl TV commercial.)

[**infocom**] Hoffman, P. *A Proof System for Recursive Programming with Local Declarations.*
paper #16 on : http://www.math.uwaterloo.ca/PM(underscore)
Dept/Homepages/Hoffman/hoffman.shtml

[**Kmn**] Kliemann, Thomas *Hoare Logic and Auxiliary Variables.* Formal Aspects of Computing, 11(1999) 541-566.

[**Men**] Mendelson, Elliott *Introduction to Mathematical Logic.* Chapman & Hall, 1964.

[**Musk**] Muskens, Reinhard *Program Semantics and Classical Logic.* website : http://let.uvt.nl/general/people/rmuskens/ .

[**mutu**] Hoffman, P. *Imperative Mutual Simple Recursion Proof Systems.*
paper #15 on : http://www.math.uwaterloo.ca/PM(underscore)
Dept/Homepages/Hoffman/hoffman.shtml

[**Nip**] Nipkow, Tobias *Hoare Logics for Recursive Procedures and Unbounded Nondeterminism.* in: *Computer Science Logic (CSL 2002).* pp. 103-119 of Lecture Notes in CS # 2471, Springer, 2002.

[**Nip2**] Nipkow, Tobias *Winskel is (almost) right.* pp. 180-192 of : *Foundations of Software Technology and Theoretical Computer Science.* eds. V. Chandru and V. Vinay, Lecture Notes in CS # 1180, Springer, 1996.

[**O'D**] O'Donnell, Michael J. *A Critique of the Foundations of Hoare-Style Programming Logics.* in: [Kozen-ed.] *Logics of Programs.* pp. 349-374 of Lecture Notes in CS # 131, Springer, 1982.

[**Old**] Olderog, E-R. *Sound and Complete Hoare-like Calculi Based on Copy Rules.* Acta Inf. 16, 1981, 161-197.

[**Sch**] Schreiber, Thomas *Auxiliary Variables and Recursive Procedures.* in: *TAPSOFT'97: Theory and Practice of Software Development* pp. 697-711 of Lecture Notes in CS # 1214, Springer, 1997.

[**sem**] Hoffman, P. *Detailed Semantics of Toy Command Languages for Iteration and Recursion.*
paper #17 on : http://www.math.uwaterloo.ca/PM(underscore)Dept/Homepages/Hoffman/hoffman.shtml

[**sing**] Hoffman, P. *Deterministic Dynamic Logic Imperative Recursive Programming Proof Systems.*
paper #14 on : http://www.math.uwaterloo.ca/PM(underscore)Dept/Homepages/Hoffman/hoffman.shtml

[**Ten**] Tennant, R.D. *Denotational Semantics.* Ch.2, pp.170-322 in Vol. 3 of *Handbook of Logic in Computer Science.* Eds. S Abramsky, Dov M. Gabbay and T.S.E. Maibaum, Oxford U. Press, 1994.

[**vOh**] van Oheimb, David *Hoare Logic for Mutual Recursion and Local Variables.* in : *Foundations of Software Technology and Theoretical Computer Science* (eds. C. Pandu Rangan, V. Raman and R. Ramanujam) pp.168-180 of Lecture Notes in CS # 1783, Springer, 1999.

[**W**] Winskel, Glynn *The Formal Semantics of Programming Languages.* MIT Press, 1993.

[**while**] Hoffman, P. *Systems for 'while'-Total Correctness with Quantifiers and Connectives.*
paper #13 on : http://www.math.uwaterloo.ca/PM(underscore)Dept/Homepages/Hoffman/hoffman.shtml

[**xmpl**] Hoffman, P. *Challenge Examples and a Conjecture re Correctness of Imperative Recursive Programs.*
paper #19 on : http://www.math.uwaterloo.ca/PM(underscore)Dept/Homepages/Hoffman/hoffman.shtml