Deterministic Dynamic Logic Imperative Recursive Programming Proof Systems

Peter Hoffman

Pure Mathematics, University of Waterloo

Abstact. *A proof system for program verification is studied from a mathematical point of view. The command language is one with 'pure' recursion, where, in contradistinction to all recent work, procedure declarations can be local.*

The pluralized noun at the end of that adjectivally-gluttonous title is misleading—we really spend all our effort on one particular system, commenting only peripherally on minor variations and predecessors. The parsing of the title is $(D(DL))((I(RP))(PS))$.

In view of the chequered careers of closely related subjects—subtly unsound/mildly incomplete systems and slightly vague/sometimes unsubstantiated claims—the present introduction will be brief. All the definitions/results are given (without much motivational or braggadocio material) immediately in the long first section, in a form which should at least avoid any odour of vagueness, or of syntax vs. semantics conflation, or indeed of *in*flation. And these central facts are accompanied by a sequence of needed technical results together with some proof descriptions. That will hopefully be useful to the time-pressed expert reader. It may even generate some faith in the results' substantiatability.

However, Sections 3, 4 and 5 provide a quite painstaking checking of details. Absolute certainty is an elusive goal (except to the several hundred million religious/ideological crazies). But left to verify ("fill in between the lines") are hopefully just matters of routine mechanical manipulations for most readers, none needing more than 5 or 6 lines. (We do leave out some annoyingly necessary syntactic matters—unique readability, occurrences of 'sub-wfs's, substitution well-definition, etc. At least that fact has now been mentioned, in contradistinction to some of the related literature.) Though relatively recent papers bring to light individuals who actually have

more faith in a proof of soundness or completeness if it has been machine-generated/checked, I'm presumably not the only one who begs to differ on this point. But perhaps I misunderstand, and the author does it himself (without publishing the details for posterity) before or after the machine does it (a machine whose own program may have no guarantee of correctness).

The 2nd section provides a leisurely discussion of: some history of the subject; credit for the major ideas (none of which are mine), particularly to Stephen Cook, David Harel and Dana Scott (and of course Floyd and Hoare); description of what (little) we claim to be new here; comparison to existing literature; etc.....

1. Definitions, Main Results and some Technicalities.

The version here of the language of $1^{\underline{\text{st}}}$order number theory (with equality—the symbol $\approx$ being used here in the syntax half of discussions) has constant symbols 0 and 1, function symbols $+$ and $\times$, and relation symbol $<$. To avoid peripheral/orthogonal issues, we will never consider any interpretation other than $\mathbf{N}$, with $0_{\mathbf{N}}$, $1_{\mathbf{N}}$, $+_{\mathbf{N}}$, $\times_{\mathbf{N}}$, $<_{\mathbf{N}}$, the usual structure. That over-elaborate notation is occasionally used (at risk of being patronizing) in the semantic half, in order to help keep an arms-length between syntax and semantics (with $=$ [not $\equiv$] used in the semantics, i.e. metalanguage, for equality of objects, such as strings of symbols from the syntax).

Command syntax. Now we define, by structural induction, (use BNF notation if you must) some strings of symbols: the set $\mathcal{R}ec$ of *commands*, depending on having already defined strings $t, t_1, t_2, \ldots$ of *terms*, and $F, G, \ldots$ of *formulas*, as usual in the $1^{\underline{\text{st}}}$order language above. Additional symbols for $\mathcal{R}ec$ will be

$$ite \mid \ \div \mid \ ; \mid X \mid \nabla \mid call \quad .$$

Atomic commands are : $callX$, and $x \div t$ for every variable x and term t of the $1^{\underline{\text{st}}}$order language.

The structural induction generates, for commands C, D and quantifier-free $1^{\underline{\text{st}}}$order formulas G, the new commands

$$(C; D) \quad | \quad ite(G)(C)(D) \quad | \quad \nabla XC \quad .$$

Command semantics. Intuitively we are thinking of the usual meanings for these constructs : $x \leftarrow\!\!\div t$ for assignment, $(C; D)$ for sequencing, ite for if-then-else, and ∇XC for something like an Algol-style block which might be loosely written

$$\textsf{begin} \;\; declare\ X\ to\ be\ C\ ;\ callX \;\; \textsf{end} \quad .$$

(The command $D_{\nabla XC}$ defined just below simulates the superficially more general

$$\textsf{begin} \;\; declare\ X\ to\ be\ C\ ;\ D \;\; \textsf{end} \quad .)$$

Note also that we have omitted *while-do*. We agree with Harel that clarity demands, at least initially, isolating questions about recursion from the corresponding questions about iteration (which are 'orthogonal' [and easier] anyway), and both of them from questions about parameter-passing, which is a hornet's nest of complications. The command language $\mathcal{R}ec$ will of course compute any computable (i.e. partial recursive) function. So, from the viewpoint of computability theory and the interpretation $\mathbf{N}$, the language $\mathcal{R}ec$ is 'as general as needed'. If *while-do* were included, then *if-then-else* could be dropped, as far as computability is concerned. But then so could ∇ and *call* for that matter. In a future project, I would hope to go through the details of this paper, but enriching $\mathcal{R}ec$ with both iteration (*while-do*) and *variable declarations*. But I shall not make the canonical claim that this will just amount to extra tedia and messy notation. Another project would replace the assignment command by a generic 'do' command, hopefully generalizing all this to the *unbounded non-deterministic* context. A third one, the least ambitious, would allow arbitrarily many distinct procedure variables, thereby allowing for *simultaneous recursion*. See many comments in Section 2.

Let $\mathcal{S}te$ be the set of states $s : \mathcal{V}ar \to \mathbf{N}$, where $\mathcal{V}ar$ is the set of variables in the $1^{\underline{\text{st}}}$order language, those variables usually denoted $x, y, z, \ldots$. The semantics is a function (defined in the following paragraphs)

$$m \;:\; \mathcal{R}ec \;\to\; 2^{\mathcal{S}te \times \mathcal{S}te} \quad ,$$

where $2^{\mathcal{S}te \times \mathcal{S}te}$ is the set of all subsets of $\mathcal{S}te \times \mathcal{S}te$. It will satisfy the following easy theorem.

Theorem 1. *$\mathcal{R}ec$ is a deterministic language; i.e. for each $C \in \mathcal{R}ec$, the set $m(C)$ is the graph of a* (partial) *function; i.e. if (s, s') and (s, s'') are both in $m(C)$, then $s' = s''$.*

We shall define $m(C)$ by induction on

$\mathrm{nab}(C) :=$ the number of distinct D such that $\nabla X D$ occurs as a subcommand of C .

For fixed $\mathrm{nab}(C)$, it will then be defined by structural induction on C.

Definition of much of m.

$$m(x \leftarrow\!\!\div t) \;:=\; \{(s, s_{x\mapsto s(t)}) \mid s \in \mathcal{S}te \;\} \quad ,$$

where, for $k \in \mathbf{N}$,

$$s_{x\mapsto k}(y) \;=\; \{ \begin{array}{cl} s(y) & \text{if } y \neq x \ ; \\ k & \text{if } y = x \ ; \end{array}$$

and where $s(t) \in \mathbf{N}$ is the 'value' of the term t at state s, i.e. using the function s's usual extension to acting on terms, not just variables.

$$m(\mathit{call}X) \;:=\; \emptyset \ , \text{ the empty set .}$$

$$m[(C;D)] \;:=\; \{(s,s'') \mid \exists s' \text{ with } (s,s') \in m(C) \text{ and } (s',s'') \in m(D) \;\} \quad .$$

$$m[\mathit{ite}(G)(C)(D)] \;:=\; \left\{ (s,s') \mid \begin{array}{llll} G\ \mathsf{tt}@s & \text{and} & (s,s') \in m(C) & \text{or} \\ G\ \mathsf{ff}@s & \text{and} & (s,s') \in m(D) & \end{array} \right\} \quad .$$

Here and below, we shorten "...is true at state s" to $\mathsf{tt}@s$, and "...is true in $\mathbf{N}$" to $\mathsf{tt}\mathbf{N}$. And similarly $\mathsf{ff}@s$ for "is false at s ".

This much of the definition deals with the initial case when nab $= 0$, and also does all the inductive cases except for dealing with defining $m(\nabla X C)$.

Diversion. Here we make two definitions by induction, where C, D and E may be any commands.

Firstly, $|D|^{\nabla X C \to E}$ is defined by induction on D as follows:

$$|x \leftarrow\!\!\div t|^{\nabla X C \to E} \;:=\; x \leftarrow\!\!\div t \ ;$$

$$|\mathit{call}X|^{\nabla X C \to E} \;:=\; \mathit{call}X \ ;$$

$$|(C;D)|^{\nabla XG\to E} \;:=\; (|C|^{\nabla XG\to E};|D|^{\nabla XG\to E})\ ;$$
$$|ite(G)(C)(D)|^{\nabla XG\to E} \;:=\; ite(G)(|C|^{\nabla XG\to E})(|D|^{\nabla XG\to E})\ ;$$
$$\text{When } B\neq C,\quad |\nabla XB|^{\nabla XG\to E} \;:=\; \nabla X|B|^{\nabla XG\to E}\ .$$
$$|\nabla XC|^{\nabla XG\to E} \;:=\; E\ .$$

That just says: "Replace ∇XC by E, everywhere in the command".

Proposition 2. *We do get a command* $|D|^{\nabla XG\to E}$.

We'll see somewhat later (**Theorem 11** and the proof rule $(\mathsf{RCS})_{[\ ]\to}$) why ∇XC is one type of command for which we want to do a replacement. More immediately, replacing 'free' occurrences of $callX$ is central to the completion of the definition for the semantics of commands. This is $|C|^{callX\to D}$, to be denoted C_D, and defined by induction on C as follows:

$$|x \mathrel{:=} t|^{callX\to D} \;:=\; x \mathrel{:=} t\ ;$$
$$|callX|^{callX\to D} \;:=\; D\ ;$$
$$|(C;C')|^{callX\to D} \;:=\; (|C|^{callX\to D};|C'|^{callX\to D})\ ;$$
$$|ite(G)(C)(C')|^{callX\to D} \;:=\; ite(G)(|C|^{callX\to D})(|C'|^{callX\to D})\ ;$$
$$|\nabla XC|^{callX\to D} \;:=\; \nabla XC\ .$$

This is the obvious definition of replacing only *free* occurrences of $callX$ by D, with the last clause indicating that none of those occurrences in ∇XC is free. Some may be free as occurrences in C, but the ∇X 'binds them' so they are not free as occurrences in ∇XC. The ∇-symbol behaves somewhat like a quantifier acting on the 'procedure variable' X.

Proposition 3. *We do get a command* $C_D \;=\; |C|^{callX\to D}$.

Finally, for a command C, define a command $C^{<n>}$ by induction on $n\in\mathbf{N}$:

$$C^{<0>} \;:=\; callX \quad\text{and}\quad C^{<n+1>} \;:=\; C_{C^{<n>}} \quad\text{i.e. take } D=C^{<n>} \text{ just above}\ .$$

Proposition 4. *We do get a command* $C^{<n>}$*; and for all* a,b *in* $\mathbf{N}$ *we have*

$$C^{<a+b>} \;=\; [C^{<a>}]_{C^{<b>}} \;=\; |C^{<a>}|^{callX\to C^{<b>}}\ .$$

Proposition 5. *For all $n > 0$ and C we have*

$$\mathrm{nab}(C^{<n>}) \;=\; \mathrm{nab}(C) \;=\; \mathrm{nab}(\nabla X C) - 1 \;.$$

And so the following completes the (doubly-inductive) definition of the function m :

Definition of m completed.

$$m(\nabla X C) \;:=\; \bigcup_{n=0}^{\infty} m(C^{<n>}) \;.$$

Proposition 6. *For all n and C we have $m(C^{<n>}) \subset m(C^{<n+1>})$. Also $m(C^{<n>})$ is the graph of a function.* [Note that $\subset$ denotes *not-necessarily-proper* inclusion.]

The proof of **Theorem 1** is now quite clear; but (as threatened), details of all this will be included in the 3rd section.

Syntax of Dynamic Logic. Define by structural induction another set of strings (called here *wfs's—well-formed strings*) denoted $\mathcal{A}, \mathcal{B}, \ldots$:
An *atomic wfs* is just the same as an atomic formula in the $1^{\underline{\mathrm{st}}}$order language, that is,

$$t_1 \approx t_2 \quad \text{or} \quad t_1 < t_2 \quad \text{for terms } t_1, t_2 \text{ of the } 1^{\underline{\mathrm{st}}}\text{order language .}$$

The *inductively defined wfs*'s are given by

$$\neg\mathcal{A} \;\mid\; (\mathcal{A} \wedge \mathcal{B}) \;\mid\; \forall x \mathcal{A} \;\mid\; < C > \mathcal{A}$$

for wfs's $\mathcal{A}, \mathcal{B}$, with $\neg$, $\wedge$, $\forall$ the same symbols as in $1^{\underline{\mathrm{st}}}$order, for x any variable in $1^{\underline{\mathrm{st}}}$order, and for C any command.

Abbreviated wfs's are used as usual : Removal of outside brackets and those allowed by the priority rule making $\rightarrow$ and $\leftrightarrow$ 'less sticky' than $\vee$ and $\wedge$;

$\mathcal{A} \vee \mathcal{B}$ is $\neg(\neg\mathcal{A} \wedge \neg\mathcal{B})$; $\mathcal{A} \rightarrow \mathcal{B}$ is $\neg(\mathcal{A} \wedge \neg\mathcal{B})$; $\mathcal{A} \leftrightarrow \mathcal{B}$ is $(\mathcal{A} \rightarrow \mathcal{B}) \wedge (\mathcal{B} \rightarrow \mathcal{A})$;

$\exists x \mathcal{A}$ is $\neg\forall x \neg\mathcal{A}$; $\qquad [C]\mathcal{A}$ is $\neg < C > \neg\mathcal{A}$.

Note that the $1^{\underline{\mathrm{st}}}$order formulas all are wfs's.

Semantics of wfs's. Define $\mathcal{A}$ tt**N** to mean $\mathcal{A}$ tt@s for all states s (experts have permission to use the overused $\models$ here twice if preferred), where the latter is defined by induction on $\mathcal{A}$:

The atoms are 1st order with obvious semantics:

$$t_1 \approx t_2 \text{ tt@}s \iff s(t_1) = s(t_2) \ ;$$

$$t_1 < t_2 \text{ tt@}s \iff s(t_1) <_{\mathbf{N}} s(t_2) \ .$$

Then, inductively

$$\neg\mathcal{A} \text{ tt@}s \iff \text{it is not the case that } \mathcal{A} \text{ tt@}s \ ; \ \text{i.e. } \mathcal{A} \text{ ff@}s \ ;$$

$$\mathcal{A} \wedge \mathcal{B} \text{ tt@}s \iff \text{both } \mathcal{A} \text{ tt@}s \text{ and } \mathcal{B} \text{ tt@}s \ ;$$

$\forall x\mathcal{A}$ tt@s $\iff$ for every s' agreeing with s except perhaps at x, we have $\mathcal{A}$ tt@s' ;

$< C > \mathcal{A}$ tt@s $\iff$ there is some s' with $(s, s') \in m(C)$ and $\mathcal{A}$ tt@s' .

(By determinacy, the s' just above is unique. All but this last clause constitute what is often called the "standard Tarski definition of truth in 1st order logic". The last clause seems to be the essence of dynamic logic.)

It follows that the propositional and $\exists$ abbreviations get their usual semantics. More interestingly :

Proposition 7. *We have*

$$[C]\mathcal{A} \text{ tt@}s \iff \text{for all } s', \text{ if } (s, s') \in m(C), \text{ then we have } \mathcal{A} \text{ tt@}s' \ .$$

(Again, such an s' would be unique relative to the given s, for the command language in this [ardently deterministic] paper.)

Immediately from these definitions, we also get

Proposition 8. *The universal closure of the wfs $F \rightarrow < C > G$, for 1st order formulas F and G, has exactly the same meaning as the* total *correctness version of $F\{C\}G$ (sometimes $\{F\}C\{G\}$) in Floyd-Hoare logic. More generally,*

$$(\mathcal{A} \rightarrow < C > \mathcal{B}) \text{ ttN} \iff \text{for all } s \text{ such that } \mathcal{A} \text{ tt@}s,$$

$$\text{we have that } (s, s') \in m(C) \text{ for some } s', \text{ and } \mathcal{B} \text{ tt@}s' \ .$$

Proposition 9. *The universal closure of the wfs $F \to [C]G$ has exactly the same meaning as the* partial *correctness version of $F\{C\}G$ in Floyd-Hoare logic. More generally,*

$$(\mathcal{A} \to [C]\mathcal{B}) \;\; \mathtt{ttN} \quad \Longleftrightarrow \quad \text{for all } s \text{ such that } \; \mathcal{A} \; \mathtt{tt@}s,$$
$$\text{if } (s, s') \in m(C), \text{ then } \mathcal{B} \; \mathtt{tt@}s' \quad .$$

Substitution. For substituting

terms for variables in a term;

terms for free variables in a $1^{\underline{\text{st}}}$*order formula*;

terms for non − program free variables in a wfs;

we'll use notation similar to what has occurred earlier:

$$|t_1|^{x \to t}; \qquad |F|^{x \to t}; \qquad |\mathcal{A}|^{y \to t};$$

respectively. For the 1st two, there should be no need to write down the familiar inductive definitions. For the 3rd, first ignore the remark about program variables and let t be a variable z. We need to define $|C|^{y \to z}$ for commands C :

$$|x \div t|^{y \to z} \;:=\; |x|^{y \to z} \div |t|^{y \to z}$$
$$|callX|^{y \to z} \;:=\; callX$$
$$|(C; D)|^{y \to z} \;:=\; (|C|^{y \to z}; |D|^{y \to z})$$
$$|ite(H)(C)(D)|^{y \to z} \;:=\; ite(|H|^{y \to z})(|C|^{y \to z})(|D|^{y \to z})$$
$$|\nabla X C|^{y \to z} \;:=\; \nabla X |C|^{y \to z} \;\; ;$$

and then the only inductive part for wfs's not identical to that for $1^{\underline{\text{st}}}$order formulas:

$$| < C > \mathcal{A}|^{y \to z} \;:=\; < |C|^{y \to z} > |\mathcal{A}|^{y \to z} \; .$$

This definition will never be applied to any $\mathcal{A}$ in this paper if y is a program variable in any of its subcommands. A program variable is one which occurs to the left of an assignment symbol "$\div$ ", this definition being easy to formalize by induction on commands. Under this proviso, it then makes sense also to

substitute a term for free occurrences of a variable, so the notation $|\mathcal{A}|^{y\mapsto t}$ makes sense and will be occurring below.

Often below we shall encounter lists $(y_1, \cdots, y_k)$ of mutually distinct variables, denoted $\vec{y}$. A *matching disjoint list* $\vec{z}$ will of course mean $(z_1, \cdots, z_k)$, i.e. same length, again mutually distinct and with no z_j the same as any y_i. The notation $|O|^{\vec{y}\mapsto\vec{z}}$, for various objects O, will mostly only be used under these circumstances of matching lengths, of no non-variable terms, and of disjointness/distinctness, so the distinction between successive and simultaneous substitution disappears and there is no ambiguity. For example,

$$|\mathcal{A}|^{(y_1,y_2)\mapsto(z_1,z_2)} \quad \text{might as well be} \quad ||\mathcal{A}|^{y_1\mapsto z_1}|^{y_2\mapsto z_2} \ .$$

In any case, we'll always mean *simultaneous* substitution.

The following well-known and easily proved semantic facts will be used often without specific referral back here:

$$|F|^{\vec{x}\mapsto\vec{t}}\ \mathsf{tt}@s \quad \Longleftrightarrow \quad F\ \mathsf{tt}@s_{\vec{x}\mapsto s(\vec{t})} \ \ ; \ \ \text{and}$$

$$F\ \mathsf{tt}@s \quad \Longleftrightarrow \quad F\ \mathsf{tt}@s' \ \text{ if } s(x) = s'(x)\ \forall x \in F \ .$$

In the latter, "$x \in F$" is just sloppy notation for saying "the variable x has a free occurrence in F". In the former, we use $s(\vec{t})$ for the string $s(t_1), \cdots, s(t_k)$, using the function s's extension to acting on terms. We also use the obvious vectorial extension of the notation $s_{x\mapsto k}$ defined earlier.

Both displays extend to hold with F changed to any wfs $\mathcal{A}$, where it is important to recall the proviso a page back—nowhere in this paper is there a need to substitute for a variable which occurs as a program variable in a command which is part of $\mathcal{A}$. This condition saves a lot of fuss and helps us avoid making statements involving substitution which are simply false because they are too general. From these extensions, it follows that, if all variables in $\mathcal{A}$ are from $\vec{y}$, then

$$s(\vec{y}) = s'(\vec{z}) \quad \text{implies that} \quad [\ \mathcal{A}\ \mathsf{tt}@s \quad \Longleftrightarrow \quad |\mathcal{A}|^{\vec{y}\mapsto\vec{z}}\ \mathsf{tt}@s'\ ] \ .$$

The painstaking reader will find that every tiny missing line in any of the tortuous semantic arguments in the final three sections will turn out to be a special case of something in the last few paragraphs, or a quick consequence of the following intutively obvious result:

Proposition 10. *If $(s, s') \in m(C)$ and the variable x is not a program variable in C, then $s(x) = s'(x)$.*

The proof is by induction on C. See Section 3

The system for $\vdash'$. This will allow us to define $\Gamma \vdash' \mathcal{A}$ for sets Γ of wfs's and for wfs's $\mathcal{A}$. We shall be concerned with soundness for $\vdash'$. The validity of the rules of inference will be of interest since those rules are part of the main system below, for $\vdash$. It should be noted that what is meant here by validity of a rule of inference is analogous to the convention in Mendelson's book [**Men**], not the one in Enderton's book [**End**], to mention a couple of excellent texts on elementary logic, much loved by Ph.D. students in mathematical logic, some of whom have been known to become confused on this point. To explain, for validity, one merely must check that the truth in $\mathbf{N}$ (for us—in a 'general interpretation', for them—) of the consequent follows from the truth in $\mathbf{N}$ of all the antecedents, *not* the stronger statement that the truth $@s$ of the consequent follows from that of all the antecedents. Thus, for example, though the validity of an 'axiom-rule' of the form $\emptyset/(\mathcal{A} \to \mathcal{B})$ implies the validity of $\mathcal{A}/\mathcal{B}$, the converse is not true. For example, the wfs $\forall x\mathcal{A} \to \mathcal{A}$ is valid, from which, using (MP), one immediately derives the rule $\forall x\mathcal{A}/\mathcal{A}$. However, the wfs $\mathcal{A} \to \forall x\mathcal{A}$ is *not* valid, despite us having the rule $\mathcal{A}/\forall x\mathcal{A}$, which is valid (but would not be in Enderton's conventions without adding provisos), and is derivable from the system for $\vdash'$.

Completeness questions for $\vdash'$ will be of no interest, since there is no expectation for any useful form of completeness to hold.

The relation $\vdash'$ will be defined in exactly the usual 'Hilbert-style derivations/proof system' way, with respect to the following rules. (The first of these indicates that our eventual concern is with so-called Cook-completeness, or completeness relative to an oracle for truth in $1^{\underline{\text{st}}}$order number theory. And so the system is not 'axiomatic', i.e. the rules are not decidable, or even recursively enumerable, though the rest of them, after banishing the oracle, form a decidable system.)

$$\begin{array}{cc} (\mathsf{ORACLE}) & \emptyset \ / \text{ any } 1^{\underline{\text{st}}} \text{ order formula that is true in } \mathbf{N} \\ (\mathsf{MP}) & \mathcal{A}\ ,\ \mathcal{A} \to \mathcal{B}\ /\ \mathcal{B} \\ (\mathsf{TAUT}) & \emptyset\ /\ \mathcal{A} \\ & \text{for every (propositional) tautology } \mathcal{A}. \end{array}$$

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

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

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

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

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

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

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 (We have simply included every rule explicitly referred to in the proofs in Sections 3 and 4. See also the paragraph before **Theorem 11**):

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

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

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

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

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

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

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

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

$$(\mathsf{SUB})_2 \qquad F \rightarrow \mathcal{B} \ / \ (|F|^{z \mapsto x} \rightarrow \mathcal{B})$$

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

(The use of a 1$^{\underline{\text{st}}}$order F here rather than general wfs is not necessary for validity of the rule [in fact the more general form is derivable from $(\mathsf{SUB})_1$], but seems to be a helpful simplification ultimately for soundness of the final system for $\vdash$.)

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

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

A rule $(\mathsf{AX})_{[\rightarrow]}$ (use [ ] in place of $<\quad>$ in the rule just above) is valid, and can be easily derived from $(\mathsf{AX})_{disj}$ and (AND). It seems to be a matter of indifference whether $(\mathsf{AX})_{<\rightarrow>}$ is made part of the $\vdash'$-system, or reserved for the $\vdash''$-system, to be defined below. But there is a break in the symmetry here, since the analogue of $(\mathsf{AX})_{disj}$ where we use $<\quad>$ in place of [ ] is clearly invalid.

In the justifications for lines of derivations, a notation like $(\mathsf{SUB})_i^{\vec{y}\rightarrow\vec{x}}$ sometimes occurs. This just means several applications of $(\mathsf{SUB})_i$, using the variables indicated in the exponent. So actually there are several separate lines; and the order in which it is done is irrelevant. Similar remarks apply to (PRE).

Here are a few derivable propositional rules [i.e. derivable simply from (MP) and (TAUT)] to which we shall have occasion to refer:

$$\begin{array}{cc}
(\mathsf{P})_1 & \mathcal{A} \wedge \mathcal{B} \rightarrow \mathcal{C} \;/\; \mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{C}) \\
(\mathsf{P})_2 & \mathcal{A} \rightarrow \mathcal{B}\;,\; \mathcal{A} \rightarrow \mathcal{C} \;/\; \mathcal{A} \rightarrow \mathcal{B} \wedge \mathcal{C} \\
(\mathsf{P})_3 & \neg\mathcal{A} \;/\; \mathcal{A} \rightarrow \mathcal{B} \\
(\mathsf{P})_4 & \mathcal{B} \;/\; \mathcal{A} \rightarrow \mathcal{B} \\
(\leftrightarrow) & \mathcal{A} \leftrightarrow \mathcal{B} \;/\; \mathcal{A} \rightarrow \mathcal{B} \\
(\mathsf{HYSY}) & \mathcal{A} \rightarrow \mathcal{B}\;,\; \mathcal{B} \rightarrow \mathcal{C} \;/\; \mathcal{A} \rightarrow \mathcal{C}
\end{array}$$

The following are also easily derivable (see Section 3) from the corresponding (UNAR)-rules and a few others:

$$\begin{array}{cc}
(\mathsf{AX})_{[call]} & \emptyset \;/\; [callX]\mathcal{A} \\
(\mathsf{UNAR})_{\exists} & (\mathcal{A} \rightarrow \mathcal{B}) \;/\; (\exists x\; \mathcal{A} \rightarrow \exists x\; \mathcal{B}) \\
(\mathsf{UNAR})_{[\;\;]} & (\mathcal{A} \rightarrow \mathcal{B}) \;/\; ([C]\mathcal{A} \rightarrow [C]\mathcal{B})
\end{array}$$

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 $<\quad>$ to $[\quad]$. Again see Section 3. 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, to avoid diversions into side issues.

Later, in Section 2, we produce a reasonably small and more memorizable system (viz. the immediately previous sentence). My experience in the subject is insufficient to give me lease to express an opinion as to whether squashing the system down to tiny size is a good or a bad thing from the viewpoint of 'practical' program verification. (Of course, it *does* make proving soundness less tedious.) In any case, that is a matter quite separate from actually writing down and checking all the details for a system as described by the title of this paper. And the time is past due for that job to be done (correctly!) without any hand-waving claims about how it is easy to extend something less general (e.g. fixed declaration) to such a system; or about how proofs have been machine-checked, but are too long for inclusion in the write-up; or about how the case of partial (total) correctness is analogous to the total (partial) case, and so is left to the reader. Such hand-waving becomes unconvincing, at least when taken as a whole.

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

The proof is simply to take any derivation witnessing $\Gamma \vdash' \mathcal{A}$, and replace each line $\mathcal{B}$ by $|\mathcal{B}|^{\nabla X C \to E}$, noting that the $|\ \ |^{\nabla X C \to E}$-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 $\nabla X C$ 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. See Section 3 for the detailed proof of **11**.

The 'partial' rule, $(\mathsf{RCS})_{[\ ]\to}$, for recursion.

Now we want to write out some propositions necessary to check that a generalization, of a certain rule (or 'meta-rule') basically due to Dana Scott, will maintain a sound system for $\vdash$, once one knows the $\vdash'$-system to be sound. The relation $\vdash$ is defined below by building on $\vdash'$, with three new

rules in all. The rule in question here, written with "turnstyles" included, is

$$(\mathsf{RCS})_{[\,]\to} \qquad \frac{\vdash \mathcal{C} \text{ for all } \mathcal{C} \in \Omega \quad , \quad \Omega \ \cup \ \{\mathcal{A} \to [\nabla XC]\mathcal{B}\} \ \vdash' \ (\mathcal{A} \to [C_{\nabla XC}]\mathcal{B})}{\vdash \quad (\mathcal{A} \to [\nabla XC]\mathcal{B})} \ ,$$

where :
(1) *the set* Ω *is a finite set of wfs's* ;
(2) *the command* C *is such that* ∇XC *is not a subcommand of any command occurring within a wfs in* Ω ;
(3) *the wfs's* $\mathcal{A}$ *and* $\mathcal{B}$ *also have no occurrences of* ∇XC *in them.*

(As it happens, elements of Ω can be assumed to be wfs's which use only subcommands of C, but that, and countless other refinements and restrictions possible for our rules, are irrelevant side issues, to be avoided here for the sake of concentrating on the main issue.) The braces just indicate a singleton set—no relation to Floyd-Hoare notation.

Briefly before doing this, here is a simple special case, and quick run-through of the corresponding steps in showing its validity:

$$(\mathsf{RCS})_{[\,]} \qquad\qquad (\ \{[\nabla XC]\mathcal{B}\} \ \vdash' \ [C_{\nabla XC}]\mathcal{B} \ ) \ / \ [\nabla XC]\mathcal{B} \ .$$

This is almost just the special case of $(\mathsf{RCS})_{[\,]\to}$ where Ω is empty, and where $\mathcal{A}$ is (say) the exciting $1^{\underline{\text{st}}}$order formula $1 \approx 1$. We're assuming that $\mathcal{B}$ has no occurrences of ∇XC.

Firstly, by **11**, the antecedant just above yields

$$\{[E]\mathcal{B}\} \ \vdash' \ [C_E]\mathcal{B} \quad \text{for } E \text{ whose variables are all in } C.$$

Taking $E = C^{<n>}$ specializes it to

$$\{[C^{<n>}]\mathcal{B}\} \ \vdash' \ [C^{<n+1>}]\mathcal{B} \quad \text{for all } n \geq 0 \ .$$

But then the soundness of the $\vdash'$-proof system shows that $[C^{<n>}]\mathcal{B}$ ttN for all $n \geq 0$, by induction on n. 'Clearly', the latter implies $[\nabla XC]\mathcal{B}$ ttN .

That's the required validity.

The following propositions are here to give the reader 95% of the proof for the validity of $(\mathsf{RCS})_{[\,]\to}$. In Section 3, **Corollary 17** will be used crucially when we establish the soundness of the system as a whole. These propositions are written out in perhaps overly-stringent detail for experts' tastes.

For **12** to **17** we assume the following:
(1) The set Ω is always a finite set of wfs's.

(2) The command C is such that ∇XC is not a subcommand of any command occurring within a wfs in Ω.
(3) The wfs's $\mathcal{A}$ and $\mathcal{B}$ also have no occurrences of ∇XC in them.

Corollary 12.

$$\Omega \ \cup \ \{\mathcal{A} \to [\nabla XC]\mathcal{B}\} \ \vdash' \ \mathcal{A} \to [C_{\nabla XC}]\mathcal{B} \ \textit{implies that}$$

$\Omega \cup \{\mathcal{A} \to [E]\mathcal{B}\} \ \vdash' \ \mathcal{A} \to [C_E]\mathcal{B}$ *for all* E all of whose variables are also in C .

Proof. The conditions just above make it clear that there is only the one (very visible) occurrence of ∇XC on each side of $\vdash'$ for which to substitute the command E, so this is immediate from **11**.

Corollary 13.

$$\Omega \ \cup \ \{\mathcal{A} \to [\nabla XC]\mathcal{B}\} \ \vdash' \ \mathcal{A} \to [C_{\nabla XC}]\mathcal{B} \ \textit{implies that}$$

$$\Omega \ \cup \ \{\mathcal{A} \to [C^{<n>}]\mathcal{B}\} \ \vdash' \ \mathcal{A} \to [C^{<n+1>}]\mathcal{B} \quad \textit{for all } n \geq 0 \ .$$

Proof. Take $E = C^{<n>}$ in **12** , noting that $C_{C^{<n>}} = C^{<n+1>}$ and that $C^{<n>}$ has the same variables as C for $n > 0$.

Theorem 14. *The $\vdash'$-proof system is sound. That is, if $\Gamma \vdash' \mathcal{A}$, and if, for each $\mathcal{B} \in \Gamma$, we have $\mathcal{B}$ $\mathtt{ttN}$, then $\mathcal{A}$ $\mathtt{ttN}$.*

The proof is the usual checking that each rule is valid with respect to "$\mathtt{ttN}$". It is laid out in fearsome detail in Section 3, and in glib slickness elsewhere in the literature.

Corollary 15. *With the conditions on $\mathcal{A}$, $\mathcal{B}$ and Ω stated above, if*

$$\Omega \ \cup \ \{\mathcal{A} \to [\nabla XC]\mathcal{B}\} \ \vdash' \ \mathcal{A} \to [C_{\nabla XC}]\mathcal{B}$$

where all wfs's in Ω are $\mathtt{ttN}$, then $(\mathcal{A} \to [C^{<n>}]\mathcal{B})$ $\mathtt{ttN}$ for all $n \geq 0$.

Proof. Proceed by induction on n. Since $C^{<0>} = callX$ which never converges, clearly $(\mathcal{A} \to [C^{<0>}]\mathcal{B})$ $\mathtt{ttN}$ for *any* $\mathcal{A}$ and $\mathcal{B}$. For the inductive step, simply use **13**, and the soundness of $\vdash'$ from **14**.

Lemma 16. *If, for all $n \geq 0$, $\mathcal{A} \to [C^{<n>}]\mathcal{B}$ $\mathtt{ttN}$, then $(\mathcal{A} \to [\nabla XC]\mathcal{B})$ $\mathtt{ttN}$.*

The proof is easy from the definitions of $m(\nabla XC)$, of $\mathbf{ttN}$, and of $[D]\mathcal{B}$. See Section 3.

Corollary 17. *If*

$$\Omega \ \cup \ \{\mathcal{A} \to [\nabla XC]\mathcal{B}\} \ \vdash' \ \mathcal{A} \to [C_{\nabla XC}]\mathcal{B}$$

for some finite set Ω *of wfs's, all* $\mathbf{ttN}$*, then* $\mathcal{A} \to [\nabla XC]\mathcal{B}$ $\mathbf{ttN}$.

The proof is immediate from **15** and **16**.

This is the corollary used for part of the $\vdash$-system soundness proof in Section 3. Earlier (and in the straightjacket of both not going to full dynamic logic and of restricting the language so that X has a fixed meaning declared), the progenitor of the rule we are discussing has often been written as a version of

$$(\mathsf{RCSS})_{[\,]} \qquad\qquad (\ F \to [X]G) \ \vdash' \ (F \to [C_X]G\ ) \ / \ (F \to [X]G)\ ;$$

that is, in the Floyd-Hoare notational straightjacket,

$$(\mathsf{RCSSS})_{[\,]} \qquad\qquad F\{X\}G \ \vdash \ F\{bodyX\}G \ / \ F\{X\}G\ ,$$

neither of which really mean anything here, and where the distinction between $\vdash'$ and $\vdash$ (or even whether there is one) is not always made entirely clear. In the antecedent of $(\mathsf{RCS})_{[\,]\to}$, the rule we use, the $\vdash$ cannot be replaced by $\vdash'$, if we want completeness, it seems. And the $\vdash'$ cannot be replaced by $\vdash$, without some discussion to make it meaningful—and in any case, doing that will likely destroy soundness. Furthermore, a rule such as just above is apparently not quite general enough (viz. the set Ω) to get completeness, at least when one doesn't have the awkwardness of allowing only a single fixed procedure declaration. See Section 2 for more remarks on this.

The system for $\vdash''$, definition of $\vdash^w$, and 'total' recursion rule, $(\mathsf{RCS})_{<\ >}$.

The system defining $\vdash''$ is again a straightforward 'Hilbert system', where we merely add one rule to all the earlier ones for $\vdash'$:

$$(\mathsf{AX})_{<\nabla>} \qquad \emptyset \ / \ (< C_{\nabla XC} > \mathcal{A} \ \longleftrightarrow \ < \nabla XC > \mathcal{A})\ .$$

There is an exceptionally simple proof that $(\mathsf{AX})_{[\nabla]}$ (you know what I mean!) is derivable.

Now define, for any variable w, any set Γ of wfs's, and any wfs $\mathcal{A}$, the relation

$$\Gamma \ \vdash^w \ \mathcal{A}$$

to mean that there is a derivation witnessing $\Gamma \ \vdash'' \ \mathcal{A}$ in which justifications using the following five rules have extra restrictions as follows. (Basically we just require the variable w to not be involved).

$(\mathsf{UNAR})_\forall$ $x \neq w$; $(\mathsf{UNAR})_{< >}$ $w \notin C$;
$(\mathsf{SUB})_1$ $u \neq w$; $(\mathsf{SUB})_2$ $z \neq w$;
(PRE) $z \neq w$.

The crucial property of $\vdash^w$ will be that it is halfway between 'Mendelsonian' and 'Endertonian' (see the paragraphs just before the rules for $\vdash'$), in the sense that it preserves 'truth at the set of states with a fixed value of w'.

The rule referred to in the title is the following, again with the "turnstyles" included.

$$(\mathsf{RCS})_{< >} \quad \frac{\vdash \neg|\mathcal{A}|^{w\to 0} \ , \ \vdash \mathcal{C} \ \forall \mathcal{C} \in \Omega \ , \ \Omega \cup \{ \ \mathcal{A} \to \mathcal{B}\} \ \vdash^w \ (|\mathcal{A}|^{w\to w+1} \to \mathcal{B} \ )}{\vdash \ (\mathcal{A} \to \mathcal{B})} \ ,$$

for all variables w, all wfs's $\mathcal{A}$ (recall that, because of the substitution, w cannot be a program variable in any of $\mathcal{A}$'s subcommands), all finite sets Ω of wfs's, and all wfs's $\mathcal{B}$ in which w does not occur.

This has as its progenitor a rule often attributed to Sokolowski [**Sok**] and very well known. One slightly phoney aspect of our present generality is that, in the past, $\mathcal{A}$ would be a $1^{\underline{\text{st}}}$order formula and $\mathcal{B}$ would be the latter 2/3 of a total correctness Floyd-Hoare statement, $< \nabla XC > G$ in our case. We will only use it in that case, and actually the proof of completeness shows that it would be needed at most once (in a derivation of a wfs $\mathcal{D}$) for each ∇XC which occurs as a command within $\mathcal{D}$. A second aspect of generality is the occurrence of Ω. When empty, the rule simplifies to something looking more like the progenitor. But non-empty Ω definitely seem to be needed once one allows oneself an escape from the awkwardness of a fixed procedure for the entire language. Finally, the consequent usually has an $\exists w$ in front of F, but this is redundant in the presence of the rule (PRE).

Note that there seems to be no need for a derived rule whose consequent is simply $< \nabla XC > G$, so we don't bother with the notation $(\mathsf{RCS}\ )_{< >\to}$.

Tradition asks us to use the letter "n" in this rule as the variable, in place of "w". But our weak intellect finds that makes it too easy for us to conflate syntax—n as a variable—with semantics—n as a natural number; so we answer "no" to tradition.

The system for $\vdash$.

The system for $\vdash$ will **informally** consist of
(1)″ All the rules for $\vdash''$;
(2)″ the (so-called meta-)rule $(\mathsf{RCS})_{[\ ]\rightarrow}$ further up;
(3)″ the (so-called meta-)rule $(\mathsf{RCS})_{<\ >}$ just above.

The point of view of dynamic logic allows avoiding all the fuss, in the straightjacket of ordinary Floyd-Hoare logic, as discovered by the authors of, and exposited in, the papers **[AdB]** **[Apt]** **[Kmn]** **[N]** **[Sc]** **[vOh]**, about *counting variables, auxiliary variables, adaptation rules, contexts, correctness phrases, conversion of Hilbert systems to Gentzen systems to make sense of soundness,* etc. (none of which occur here).

It could be argued that, because of the "meta" nature of (2)″ and (3)″, we have actually gotten away from dynamic logic. There is more on this in Section 2, where we conjecture that this is unavoidable if we wish to remain deterministic. Harel has essentially shown that it isn't if we don't and we're willing to extend the command language—see **[Harel]** pp. 44-55, and comments in our Section 2.

Because of the slightly 'bizarre' forms of the rules in (2)″ and (3)″, we need to formulate $\vdash$ in a more stringent manner. Let Γ be any set of wfs's. Here, to define the meaning of " $\Gamma \vdash \mathcal{D}$ ", we shall very temporarily re-write the latter as " $\mathcal{D} \in \Gamma^{\vdash}$ " *and officially define* $\Gamma^{\vdash}$ *to be the smallest set of wfs's such that*

(0)′ $\Gamma \subset \Gamma^{\vdash}$;
(1)′ if $\Lambda \vdash'' \mathcal{D}$ for some finite set Λ of wfs's, and $\Gamma \vdash \mathcal{B}$ for all $\mathcal{B} \in \Lambda$, then $\Gamma \vdash \mathcal{D}$;

(2)′ if $\Omega \cup \{\mathcal{A} \rightarrow [\nabla XC]\mathcal{B}\} \vdash' \mathcal{A} \rightarrow [C_{\nabla XC}]\mathcal{B}$, for some finite set Ω of wfs's, and $\Gamma \vdash \mathcal{C}$ for all $\mathcal{C} \in \Omega$, then $\Gamma \vdash \mathcal{A} \rightarrow [\nabla XC]\mathcal{B}$ as long as ∇XC does not occur as a command within $\mathcal{A}$ or $\mathcal{B}$ or any $\mathcal{C} \in \Omega$; and

(3)′ if the variable w does not occur in the wfs $\mathcal{B}$, then $\Gamma \vdash \neg|\mathcal{A}|^{w \rightarrow 0}$ plus $\Gamma \vdash \mathcal{C}$ for all $\mathcal{C} \in \Omega$ plus $\Omega \cup \{\ \mathcal{A} \rightarrow \mathcal{B}\} \vdash^{w} (|\mathcal{A}|^{w \rightarrow w+1} \rightarrow \mathcal{B}\ )$ imply that

$\Gamma \vdash \ \mathcal{A} \to \mathcal{B}$, where $\mathcal{A}$ is any wfs, w is any variable, and Ω is any finite set of wfs's (and because of the substitutions, it is assumed implicitly that w is not a program variable in any command within $\mathcal{A}$).

Theorem 18. ($\vdash$-derivation theorem) *The assertion* $\Gamma \vdash \mathcal{D}$ *holds if and only if there is a sequence*

$$(\mathcal{A}_1, \mathcal{J}_1) \ , \ (\mathcal{A}_2, \mathcal{J}_2) \ , \ \cdots\cdots\cdots \ , \ (\mathcal{A}_k, \mathcal{J}_k)$$

(" $\mathcal{J}_i$ is the justification for appending $\mathcal{A}_i$ to $\mathcal{A}_1, \cdots, \mathcal{A}_{i-1}$ ")
with $\mathcal{A}_k = \mathcal{D}$ *, and such that each* $\mathcal{J}_i$ *is one of :*

(0) *the observation that* $\mathcal{A}_i \in \Gamma$*; or*

(1) *a* $\vdash''$*-justification for* $\mathcal{A}_i$ *, referring to the lines* $\mathcal{A}_1, \cdots \mathcal{A}_{i-1}$ *preceding it; or*

(2) *a justification based on* (2)′*: that is,* $\mathcal{A}_i$ *has the form* $\mathcal{A} \to [\nabla XC]\mathcal{B}$ *and* $\mathcal{J}_i$ *is a* $\vdash'$*-derivation witnessing*

$$\Omega \ \cup \ \{\mathcal{A} \to [\nabla XC]\mathcal{B}\} \ \vdash' \ \mathcal{A} \to [C_{\nabla XC}]\mathcal{B} \ ,$$

with all wfs's in Ω *occurring as lines above* $\mathcal{A}_i$*, and the command* ∇XC *not occurring in any of* $\mathcal{A}$ *or* $\mathcal{B}$ *or any* $\mathcal{C} \in \Omega$*; or finally*

(3) *a justification based on* (3)′*: that is,* $\mathcal{A}_i$ *has the form* $\mathcal{A} \to \mathcal{B}$ *, with the variable* w *not occurring in the wfs* $\mathcal{B}$*, one of the preceding* $\mathcal{A}_j$*'s is* $\neg|\mathcal{A}|^{w \to 0}$ *, all wfs's in* Ω *occur as lines above* $\mathcal{A}_i$*, and* $\mathcal{J}_i$ *is a* $\vdash''$*-derivation witnessing*

$$\Omega \cup \{ \ \mathcal{A} \to \mathcal{B}\} \quad \vdash^w \quad (|\mathcal{A}|^{w \to w+1} \to \mathcal{B}) \ .$$

Actually, only $\vdash \mathcal{D}$ occurs below, not $\Gamma \vdash \mathcal{D}$ for nonempty Γ, in contradistinction to $\vdash'$ and $\vdash^w$. But, in the case of non-empty Γ, the theorem makes it clear that in making a $\vdash'$-derivation or $\vdash^w$-derivation to check the main antecedent in one of the two recursion rules, one is *not* to use wfs's from Γ. This can be a fuzzy point in the literature—see Section 2.

The last theorem is here partly to reinforce the definition—it won't be proved in detail since it is just an amplification of the "smallest set" part of the $\vdash$ definition—and partly to simplify the discussion and proof of soundness for this system in Section 3. Here is a brief explanation for the mildly unhappy need for four different "turnstyles". Just to formulate the two recursion rules, we need preliminary systems. (See Section 2 for some discussion

about whether earlier, rather less formally given, recursion rules were doing this explicitly or implicitly.) The extra rule to get $\vdash''$ seem to definitely be needed for completeness. On the other hand the rule $(\mathsf{AX})_{<\nabla>}$ must be excluded in the definition of $\vdash'$ because it explicitly involves ∇XC, and we have an apparent need, at one point in the soundness/completeness proofs in Sections 3 to 5, to substitute for ∇XC and preserve $\vdash'$-derivability, viz. **11**. Finally, the restriction, in the main antecedent of rule $(\mathsf{RCS})_{< >}$, to using $\vdash^w$ seems to be necessary for it to be a valid rule.

The next two theorems are the main results.

Theorem 19. (Soundness of the $\vdash$-system) *If* $\vdash \mathcal{A}$, *then* $\mathcal{A}$ $\mathbf{ttN}$.
The same proof shows that if $\Gamma \vdash \mathcal{A}$ and all $\mathcal{B} \in \Gamma$ are $\mathbf{ttN}$, then $\mathcal{A}$ $\mathbf{ttN}$, though that is unneeded in this paper.

We'll need to prove **14** and validity of $(\mathsf{AX})_{<\nabla>}$, discuss $(\mathsf{RCS})_{[\,]\to}$, and deal with validity of $(\mathsf{RCS})_{< >}$. See Section 3 for the proof, and Section 2 for more discussion.

Theorem 20. (Completeness of the $\vdash$-system) *If* $\mathcal{A}$ $\mathbf{ttN}$, *then* $\vdash \mathcal{A}$.

See the results just below for the strategy, Section 2 for discussion, and Sections 4 and 5 for the proofs. Note that Sections 3, 4 and 5 are essentially independent of each other; they can be read in any order. 'Logically', 3 comes first, since proving completeness for an unsound system would seem a waste of time. Probably 5 comes next, since 4 refers to objects whose existence is established in 5.

Theorem 21. (essentially in [**Harel**]) *To prove* **20**, *it suffices to prove the following two special cases, for* $1^{\underline{\text{st}}}$*order* F *and* G, *and any command* D :
(i) $(F \to [D]G)$ $\mathbf{ttN}$ $\implies$ $\vdash (F \to [D]G)$; *and*
(ii) $(F \to < D > G)$ $\mathbf{ttN}$ $\implies$ $\vdash (F \to < D > G)$.

This is proved in Section 4.

Theorem 22. *For all* F, D *and* G, *statement* **21***(i) holds.*

Theorem 23. *For all* F, D *and* G, *statement* **21***(ii) holds.*

These two, and the following technical results, (plus deductions of **22** and **23** from them) are placed here rather than in the 4th and 5th sections (where the technical results are proved in detail) for ease of reference in the next

section's critique of the past literature and its relation to this paper.

The technical results below are numbered in the form **x.y**, where :
x is **22** or **23**, and, fixing it, the **x.y**'s are the main components of the proof of **Theorem x**.
y lies between **0** and **5**.
x.0 is needed to make sense of the rest, and is proved independently in Section 5.
The **x.y** for **y** between **1** and **5** are proved in reverse order to the size of the numbers **y**. For value **4** and **5**, they are proved directly ; **x.3** uses **x.4** ; **x.2** uses **x.3** and **x.5** ; **x.1** uses **x.2**, **x.3** and **x.4** ; **Theorem x** uses **x.1** and **x.2** [and also **x.4** when **x**=23]. This information may help time-challenged expert readers avoid some of Sections 4 and 5, where exceedingly fine detail is given, relative to the usual conventions of the subject.

By comparing the coarse details of this section, the reader will hopefully see something of an analogy, actually more like a duality, relating **22** and **23**, and between the overall form of the proofs of those two theorems. But it's not sufficiently precise as to be mathematically formulated, at least not yet. Not surprisingly, **23** is more laborious to prove.

The next result, **22.0**, and its mate, **23.0**, assert the existence of certain $1^{\underline{\text{st}}}$order formulas, A and B, defined by their semantics. And so these are not unique as formulas, only unique 'up to truth equivalence'. But every time they occur, that is the only important thing mathematically, i.e. true for one implies true for all, so this will not be dwelt upon each time. A few examples of these formulas are produced in the appendix to Section 2.

Lemma 22.0.

(a) *For each command C, and $1^{\underline{\text{st}}}$order formula F, there is a $1^{\underline{\text{st}}}$order formula $A(C,F)$—("A" for "after" rather than "post")—such that*

$$A(C,F)\ \mathtt{tt}@s' \quad \Longleftrightarrow \quad \exists s \text{ with } F\ \mathtt{tt}@s \text{ and } (s,s') \in m(C)\ .$$

And so, $A(C,F)$ is the 'universal' A for which $(F \to [C]A)$ $\mathtt{ttN}$. That is, for all $1^{\underline{\text{st}}}$order K,

$$F \to [C]K\ \ \mathtt{ttN} \quad \Longleftrightarrow \quad A(C,F) \to K\ \ \mathtt{ttN}\ .$$

(b) *For any wfs $\mathcal{H}$, there is a $1^{\underline{\text{st}}}$order formula H, whose free variables are from the variables occurring in $\mathcal{H}$, with $H \leftrightarrow \mathcal{H}$ true in $\mathbf{N}$.*

Given a desire to derive $\mathcal{H}$, one of course need only discover a corresponding H as in (b) of the lemma (and hopefully true in $\mathbf{N}$!), derive $H \to \mathcal{H}$, and then appeal to (ORACLE) and (MP). But the first two steps are more easily said than done. In particular, the second is the obstruction to an easy proof of completeness, though in some sense, the whole enterprise of Cook-completeness has to do with reduction from dynamic to ('static') $1^{\underline{\text{st}}}$order logic. Dynamic logic makes it very easy to formalize interesting statements which are not exactly correctness statements, for example statements about 'equivalence' of commands, so **22.0**(b) says rather more than one might initially appreciate.

Lemma 22.1. *For all commands C and matching disjoint lists $\vec{x}$ of the form $(x_1, \cdots, x_k)$ and $\vec{z}$ of pairwise distinct variables such that all variables from C are in $\vec{x}$, there is a finite set Ω of wfs's which are*

(i) all ttN*; and*

(ii) involve only commands which occur as subcommands of C;

such that we have

$$\Omega \cup \{\vec{x} \approx \vec{z} \ \to \ [\nabla XC]A(\nabla XC, \vec{x} \approx \vec{z})\} \ \vdash' \ (\vec{x} \approx \vec{z} \ \to \ [C_{\nabla XC}]A(\nabla XC, \vec{x} \approx \vec{z})) \, .$$

In fact, Ω can be taken as the set of all $\vec{x} \approx \vec{z} \ \to \ [\nabla XE]A(\nabla XE, \vec{x} \approx \vec{z})$ as E ranges over all commands for which ∇XE is a subcommand of C.

Here $\vec{x} \approx \vec{z}$ is short for $x_1 \approx z_1 \ \wedge \ \cdots \ \wedge \ x_k \approx z_k$.

To dramatize the contrast between $\vdash'$ and $\vdash''$, note that $(\mathsf{AX})_{[\nabla]}$ gives a completely trivial derivation to witness

$$\{\vec{x} \approx \vec{z} \ \to \ [\nabla XC]A(\nabla XC, \vec{x} \approx \vec{z})\} \ \vdash'' \ (\vec{x} \approx \vec{z} \ \to \ [C_{\nabla XC}]A(\nabla XC, \vec{x} \approx \vec{z})) \, ,$$

and lots more.

Lemma 22.2. *Suppose given a command D, and $1^{\underline{\text{st}}}$order formulas F and G, such that $(F \to [D]G)$* ttN*. For each command E such that ∇XE is a sub-command of D, choose a pair of matching disjoint lists $\vec{x}$ and $\vec{z}$ of pairwise distinct variables, such that all the variables in E are in $\vec{x}$. Let Γ_D be the* (finite!) *set of all formulas (one for each such E)*

$$\vec{x} \approx \vec{z} \ \to \ [\nabla XE]A(\nabla XE, \vec{x} \approx \vec{z}) \, .$$

Then $\Gamma_D \ \vdash' \ (F \to [D]G)$.

Deduction of Theorem 22 from 22.1 and 22.2. Proceed by induction on D to show $\vdash (F \to [D]G)$ from its truth in $\mathbf{N}$. Since $\vdash'$ is stronger than $\vdash$, we have $\Gamma_D \vdash (F \to [D]G)$ for the set Γ_D in **22.2**, from that lemma's conclusion. It remains to prove that $\vdash \mathcal{D}$ for each $\mathcal{D} \in \Gamma_D$; which will give $\vdash (F \to [D]G)$, as required.

The initial cases are immediate, since Γ_D is empty then.

For the inductive cases, all $\mathcal{D} \in \Gamma_D$ are $\mathsf{tt}\mathbf{N}$, and they all have the form $F_0 \to [D_0]G_0$ in which D_0 is a subcommand of D. Only when D has the form ∇XB and $D_0 = D$ is it not a *proper* subcommand, so the inductive hypothesis shows $\vdash \mathcal{D}$ as required in all but that one case.

In that case,

$$\mathcal{D} = (\vec{x} \approx \vec{z} \ \to \ [\nabla XB]A(\nabla XB, \vec{x} \approx \vec{z})) \ .$$

Here we apply the instance of $(\mathsf{RCS})_{[\,]\to}$ for which
$\mathcal{A} = \vec{x} \approx \vec{z}$, $C = B$, $\mathcal{B} = A(\nabla XB, \vec{x} \approx \vec{z})$ and
$\Omega = \{\ \vec{x} \approx \vec{z} \ \to \ [\nabla XE]A(\nabla XE, \vec{x} \approx \vec{z}) \mid \nabla XE$ is a subcommand of $B\ \}$.
Then all $\mathcal{C} \in \Omega$ also are true in $\mathbf{N}$ and have the form $F_0 \to [D_0]G_0$ in which D_0 is a proper subcommand of D. And so we have $\vdash \mathcal{C}$ for all $\mathcal{C} \in \Omega$ by the inductive hypothesis. To verify the other antecedent in $(\mathsf{RCS})_{[\,]\to}$, just use **22.1**, with $C = B$. The set Ω above is exactly the correct one.

Lemma 22.3. *For any F, G and C', and matching disjoint lists $\vec{x}$ and $\vec{z}$ of pairwise distinct variables such that all variables from C' are in $\vec{x}$, if $(F \to [C']G)$ $\mathsf{tt}\mathbf{N}$, then*

$$\{\vec{x} \approx \vec{z} \ \to \ [C']A(C', \vec{x} \approx \vec{z})\} \quad \vdash' \quad (F \to [C']G) \ .$$

Lemma 22.4. *With hypothesis as in* **22.3**, *we have*

$$(A(C', \vec{x} \approx \vec{z}) \ \wedge \ |F|^{\vec{x} \to \vec{z}} \ \to \ G) \quad \mathsf{tt}\mathbf{N} \ .$$

Lemma 22.5. *If D is a command, and Γ is a set of wfs's, define the phrase "Γ der $[D]$" to mean*

$\mathsf{for\ all}\ 1^{\underline{\mathrm{st}}}\ \mathsf{order\ formulas}\ F\ \mathsf{and}\ G, \quad (F \to [D]G)\ \mathsf{tt}\mathbf{N} \implies \Gamma \vdash' (F \to [D]G)$.

Assume Γ_1 der $[D_1]$ and Γ_2 der $[D_2]$. Then $\Gamma_1 \cup \Gamma_2$ der $[(D_1; D_2)]$; and similarly, with H any quantifier-free $1^{\underline{\mathrm{st}}}$order formula, we also have

$$\Gamma_1 \cup \Gamma_2 \ \ \mathsf{der} \ \ [ite(H)((D_1)(D_2)] \ .$$

The proof is essentially the same as one of the easier parts of Cook's seminal contributions to the subject. See Section 4.

Lemma 23.0. *For all commands C and D, $1^{\underline{\text{st}}}$order formulas G, and variables $w \notin C \cup D \cup G$, there is a $1^{\underline{\text{st}}}$order formula $B(w, C, D, G)$—("B" for "before" rather than "pre")—such that*

$$B(w, C, D, G)\ \mathsf{tt}@s \quad \Longleftrightarrow \quad < D_{C^{<s(w)>}} > G\ \mathsf{tt}@s\ .$$

We don't here use the notation $F(t)$, elsewhere denoting F with all free occurrences of some-or-other variable replaced by the term t. The "w" in $B(w, C, D, G)$ is there for a different reason: to tell us which variable is the 'special one', playing a 'counting role' in its semantics, as defined by the display in the lemma.

It follows from that semantics (though not directly utilized here) that

$$\exists w B(w, C, D, G)\ \rightarrow\ < D_{\nabla XC} > G\quad \mathsf{ttN}\ .$$

In fact, $\exists w B(w, C, D, G)$ is the 'universal' $1^{\underline{\text{st}}}$order formula J for which we have $(J \rightarrow < D_{\nabla XC} > G)\ \mathsf{ttN}$, in the sense that $(J \rightarrow \exists w B(w, C, D, G))\ \mathsf{ttN}$ for any such J. Summarizing, for all J,

$$J \rightarrow < D_{\nabla XC} > G\ \ \mathsf{ttN} \quad \Longleftrightarrow \quad J \rightarrow \exists w B(w, C, D, G)\ \ \mathsf{ttN}\ .$$

In each of **23.1** to **23.4** below, a command C, a variable w, and variable strings $\vec{x}$, $\vec{z}$ are involved.

Let us just use $B(C)$ to denote $B(w, C, callX, \vec{x} \approx \vec{z})$.

This notation is only used with $\vec{x}$ and $\vec{z}$ disjoint strings of distinct variables, with all variables in C from $\vec{x}$, and with one more new variable w.

Remark. The definition of $B(C)$ and the semantics of $B(w, C, D, G)$ in general yield

$B(C)\ \mathsf{tt}@s \iff$ for some s', we have $(s, s') \in m(C^{<s(w)>})$ and $s'(\vec{x}) = s'(\vec{z})$.

Thus, from the semantics of ∇XC,

$\exists w\ B(C)\ \mathsf{tt}@s \iff$ for some s', we have $(s, s') \in m(\nabla XC)$ and $s'(\vec{x}) = s'(\vec{z})$.

So the following is true in **N**:

$$\exists w\ B(C) \ \longleftrightarrow\ \ < \nabla XC > \vec{x} \approx \vec{z}\ .$$

Deriving the '$\rightarrow$-half' of that wfs looms large just ahead.

Lemma 23.1. *For each command C, pair of matching disjoint lists $\vec{x} = (x_1, \cdots, x_k)$ and $\vec{z}$ of pairwise distinct variables such that all variables from C are in $\vec{x}$, and variable $w \notin \vec{x} \cup \vec{z}$, the finite set Ω of wfs's below, which are all* ttN*, is such that, for all $1^{\underline{\text{st}}}$order formulas G and subcommands D of C, we have*

$$\Omega\ \cup\ \{B(C) \rightarrow < \nabla XC > \vec{x} \approx \vec{z}\}\ \ \vdash^w\ \ B(w, C, D, G) \rightarrow < D_{\nabla XC} > G\ .$$

The Ω which usefully does the trick is the set of all wfs's $B(*, *, \nabla XE, \vec{x} \approx \vec{z}) \rightarrow\ \ < \nabla XE > \vec{x} \approx \vec{z}$ as E ranges over all commands for which ∇XE is a subcommand of C.

Why the two $*$'s in $B(*, *, \nabla XE, G)$ just above, and also below in **23.4**(i)(c)? Since ∇XE has no free *call*X, we get $(\nabla XE)_D = \nabla XE$ for any D, and so $B(w, C, \nabla XE, G)$ is 'the same' for all C and w; i.e. they are all tt for the same set of states.

The lemma above will only be used in the case $D = C$ and $G = \vec{x} \approx \vec{z}$; but the general case seems needed for seeing how to prove it—namely, by induction on D, and with one inductive case really needing general G.

Lemma 23.2. *Suppose given a command D_1, and $1^{\underline{\text{st}}}$order formulas F and G, such that $(F \rightarrow < D_1 > G)$* ttN*. For each command C such that ∇XC is a sub-command of D_1, choose a pair of matching disjoint lists $\vec{x}$ and $\vec{z}$ of pairwise distinct variables, and variable $w \notin \vec{x} \cup \vec{z}$, such that all the variables in C are in $\vec{x}$. Let the set Λ_{D_1} consist of all formulas (one for each such C)*

$$\exists w B(C)\ \ \rightarrow\ \ \ < \nabla XC > \vec{x} \approx \vec{z}\ .$$

Then $\Lambda_{D_1}\ \vdash^w\ (F \rightarrow < D_1 > G)$.

Lemma 23.3. *For any F, G and C, and matching disjoint lists $\vec{x}$ and $\vec{z}$ of pairwise distinct variables such that all variables from C are in $\vec{x}$, if $(F \rightarrow < \nabla XC > G)$* ttN *, then*

$$\{(\exists w\ B(C)) \rightarrow < \nabla XC > \vec{x} \approx \vec{z}\}\ \ \vdash^w\ \ (F \rightarrow < \nabla XC > G)\ .$$

Lemma 23.4. *Under the conditions listed below, each of the following five* $1^{\underline{\text{st}}}$*order formulas is true in* $\mathbf{N}$.

(i)(a) $F \ \rightarrow \ \exists \vec{z}((\exists w B(C)) \wedge |G|^{\vec{x}\rightarrow\vec{z}})$;

(b) $B(w, C, callX, G) \ \rightarrow \ \exists \vec{z}(B(C) \wedge |G|^{\vec{x}\rightarrow\vec{z}})$;

(c) $B(*, *, \nabla XE, G) \ \rightarrow \ \exists \vec{z}(B(*, *, \nabla XE, \vec{x} \approx \vec{z}) \wedge |G|^{\vec{x}\rightarrow\vec{z}})$;

(ii) $\neg |B(C)|^{w\rightarrow 0}$;

(iii) $|B(C)|^{w\rightarrow w+1} \ \rightarrow \ B(w, C, C, \vec{x} \approx \vec{z})$.

Assumptions: *We are given a command* C*, and matching disjoint lists* $\vec{x}$ *and* $\vec{z}$ *of pairwise distinct variables such that all variables in* C *are from* $\vec{x}$*, plus a variable* w *not from* $\vec{x} \cup \vec{z}$*. Recall the formula* $B(C)$ *from just after* **23.0**. *In* (i)(a)*, assume* $F \rightarrow < \nabla XC > G$ *is true in* $\mathbf{N}$*, with no variables from* $\vec{z}$ *in the* $1^{\underline{\text{st}}}$*order formulas* F *and* G *in all three parts of* (i). *Finally, in* (i)(c)*, assume the command* E *also has all variables from* $\vec{x}$.

Deduction of Theorem 23 from 23.1, 23.2 and (ii) **,** (iii) **of 23.4**. Proceed by induction on D_1 to show $\vdash (F \rightarrow < D_1 > G)$ from its truth in $\mathbf{N}$. By **23.2** it suffices to establish, for any C, w, $\vec{x}$ and $\vec{z}$ as in **23.2**, that

$$\vdash \quad \exists w B(C) \ \rightarrow \ < \nabla XC > \vec{x} \approx \vec{z} \, .$$

To do this, look at the case $D = C$ and $G = \vec{x} \approx \vec{z}$ of **23.1**, and with Ω as in its 2nd sentence. Firstly, each $\mathcal{C} \in \Omega$ is true in $\mathbf{N}$, and has the form $F_1 \rightarrow < \nabla XE > G_1$ with ∇XE a proper subcommand of D_1. So, by the inductive hypothesis, we have

$$\vdash \mathcal{C} \ \text{ for all } \mathcal{C} \in \Omega \qquad (1)$$

Next, using (ORAC) and **23.4**(ii),

$$\vdash \quad \neg |B(C)|^{w\rightarrow 0} \qquad (2)$$

The conclusion of **23.1** yields

$$\Omega \cup \{B(C) \rightarrow < \nabla XC > \vec{x} \approx \vec{z}\} \ \vdash^w \ B(w, C, C, \vec{x} \approx \vec{z}) \rightarrow < C_{\nabla XC} > \vec{x} \approx \vec{z} \, .$$

By $(\mathsf{AX})_{<\nabla>}$ and $(\leftrightarrow)$, we have

$$\vdash^w \quad < C_{\nabla XC} > \vec{x} \approx \vec{z} \ \rightarrow \ < \nabla XC > \vec{x} \approx \vec{z} \, .$$

Using (ORAC) and **23.4**(iii), we have

$$\vdash^w \quad |B(C)|^{w \mapsto w+1} \to B(w, C, C, \vec{x} \approx \vec{z}) \ .$$

Now two applications of (HYSY) yield

$$\Omega \cup \{ \ B(C) \to < \nabla XC > \vec{x} \approx \vec{z} \ \} \ \vdash^w \ (|B(C)|^{w \mapsto w+1} \to < \nabla XC > \vec{x} \approx \vec{z}) \qquad (3)$$

So (1), (2) and (3) give us the antecedents in that instance of $(\mathsf{RCS})_{< >}$ with $\mathcal{A} = B(C)$ and $\mathcal{B} =< \nabla XC > \vec{x} \approx \vec{z}$. And so, using that recursion rule, we conclude

$$\vdash \quad B(C) \to < \nabla XC > \vec{x} \approx \vec{z} \ .$$

Finally, a single application of (PRE) gives the required

$$\vdash \quad \exists w B(C) \ \to \ < \nabla XC > \vec{x} \approx \vec{z} \ .$$

Lemma 23.5. *This is identical to* **22.5** *except for replacing* $[C]$ *by* $< C >$ *for each command* C*, and replacing* $\vdash'$ *by* $\vdash^w$*.*

The proof is virtually identical also.

2. What may be new here, and what isn't; plus lots of examples.

Having included little of this material earlier, now we attempt in quite considerable detail to explain where the present paper seems to represent some small advances on previous knowledge. To summarize, it produces a sound, complete proof system, *entirely within a deterministic command language and its associated dynamic logic*, where the command language has recursion, and a given command is able to have *any number of procedures declared (locally) as the (temporary) 'value' of the single procedure variable.* At a more particular level, it is introducing two proof rules for recursion (for partial and for total correctness), very much derivative of earlier major ideas, but which do have a couple of novel aspects. There are a number of rather vague claims in papers on this subject, which need discussion. Thus, a more detailed comparison to the previous literature is required, so here goes.

From my (perhaps naive) point of view, it is worth distinguishing four levels for recursion in imperative programming languages, as in the following diagram, **often referred to in the rest of this section**.

single procedure variable; *fixed declaration*	*single procedure variable;* *variable (i.e. local) declaration*
many procedure variables; *fixed declarations*	*many procedure variables;* *variable (i.e. local) declarations*

Note that, as a single language with a fixed declared program (often denoted S_0), with only assignments, sequencing and 'if-thendo-elsedo' in addition to recursion, a language as in the top-left corner would not be 'Turing equivalent' (over $\mathbf{N}$)—there are of course lots of computable functions it cannot compute—whereas the other three corners are Turing equivalent, though the bottom left would be rather unwieldy, to say the least. (As is well-known, if iteration, i.e. 'while-do', is included, recursion becomes redundant from the viewpoint of a computability theorist, and the top-left becomes able to compute anything that any other deterministic language can.)

Before going further, here is a simple example of this, an example which **[Nip]** uses to illustrate something he regards as fundamental. We shall come back to give some

considerable detail on it. See **Ex. 10** and **11** and several other examples in the appendix to this section. The wfs in the first display below is expressed in the language of Section 1, and we claim it to be true in **N**, and so it is derivable within our system of that section, by completeness. When we come back to this in the appendix, we shall write down a derivation. On the other hand, command languages as indicated above on the top-left cannot express the command, because of the two distinct ∇XC-commands. The wfs just below is saying that the command inside the [ ] is fixing the value of x whenever it converges. In fact it remains true even with [ ] changed to $<\ \ >$, and so the command is really equivalent to *bugga*, all the more since it certainly cannot move any other variable, by **Proposition 10** or $(\mathsf{AX})_{disj}$:

$$x \approx z \ \rightarrow\ [\ \nabla Xite(0 < x)(\text{“}x \leftarrow\!\!\!\!\cdot\, x - 1\text{”}; callX; x \leftarrow\!\!\!\!\cdot\, x + 1)(bugga)\ ]\ x \approx z\ ,$$

where *bugga* is some fixed command which ‘does nothing’, for example $x \leftarrow\!\!\!\!\cdot\, x$, and

$$\text{“}x \leftarrow\!\!\!\!\cdot\, x - 1\text{”}\ :=\ y \leftarrow\!\!\!\!\cdot\, 0\ ;\ \nabla Xite(y + 1 < x)(y \leftarrow\!\!\!\!\cdot\, y + 1; callX)(bugga)\ ;\ x \leftarrow\!\!\!\!\cdot\, y$$

Of course x, y and z are three distinct variables, and we iterate the sequencing operator, “;”, sometimes without putting in the brackets. Admittedly, the need for the command in the first display to have two distinct ∇XC-subcommands is an artifice of the fact that our language for $1^{\underline{\text{st}}}$order number theory has not been equipped with a ‘*proper subtraction*’ symbol from the start. But a paper in theoretical CS isn't supposed to necessarily produce examples which are pregnant with money-making significance!

In this paper, we are in the top-right corner of the diagram at the beginning of this section. Recursion being such a basic construct, it initially came as a surprise to see that more than a handful of papers on program verification for ‘pure’ recursion had appeared in major CS journals within the past 15 years or so (about 40% of the ‘Floyd-Hoare era’), given the hectic rush to new frontiers that seems to pervade both ‘practical’ and theoretical CS. *These papers all deal with what I'd term the least general type of recursion, namely, the top-left above.* Both [**Nip**] and the earlier [**deBa**] treat also the bottom-left case, the latter only for partial correctness except for a few cursory remarks. We should also remark that Nipkow has solved a problem about unbounded non-determinism in that paper, a problem concerning which we hope the methods of this paper will also give some new information eventually. (The difficulty is with the analogue of Section 5 below.)

Many of these papers, including as well the earlier and very influential [**Apt**], seem to claim that all their results are generalizable by easy but tedious extensions of the ideas presented. Here are some such quotes:

[**AdB**] p.132 “Just for the sake of convenience we shall restrict ourselves in this paper to programs of the form $< P \leftarrow S_0 \mid S >$. $\cdots\cdots$ It is a straightforward, but tedious task to generalize the results of this paper to programs with more than one procedure.”

[Apt] p.443 "To simplify the discussion, we restrict our attention to the case of one procedure declaration. All the results of this section can be straightforwardly generalized to the case of more than one procedure declaration."

[Sch] p.6 "We restrict our attention to the case of a single procedure declaration. We expect no difficulties in generalizing the results of this paper to mutually recursive procedures. In the sequel, S_0 : prog denotes the body of the procedure."

[Kmn] p.17 "We restrict our attention to a single parameterless procedure. ..... An extension to multiple procedures is straightforward [ref]." (Reference to Ph.D. thesis)

[Nip] p.3 "There is only one parameterless procedure in the program. Hence $CALL$ does not even need to mention the procedure name." (Later he *does* give some details on a generalization, as indicated below.)

To my untrained eye, all but the last of these remarks are less than precise as to which of the other three corners the generalizability refers. *However, if it is either of the two on the right-hand side of the diagram, I would have to admit to some degree of skepticism* (at least until details have become available, which they haven't heretofore). This is based on some of the extra small difficulties in this paper (and other contrasts with works on the top-left type of recursion).

So it seems likely that the generalization claimed refers to going from languages on the top-left to those on the bottom-left corner. The two sources mentioned earlier which *have* given detail concerning generalization say the following (just to make it clear that it *is* the fixed declaration situation to which they refer) :

[deBa] p.128 "Note that we do not allow the procedure bodies S_i themselves to contain procedure declarations. (A limited treatment of this feature will be given in chapter 7.)"

Then Ch. 7, p.260 "Observe that there is no section on completeness. We expect that the techniques of section 5.5 can be modified appropriately to deal with the μ-calculus, but we have not explored this possibility at the moment of writing this." (μ is essentially our ∇, and de Bakker refers to Harel's work at that point, as discussed below.)

[Nip] p.12 "...this set up assumes we have a procedure body for each procedure name."

So the main object here with the possibility of being something new is this:

(1) a *sound, complete* system, *entirely within a deterministic language*, where *any number of distinct 'declarations'*, of which procedure is to corre-

spond to the procedure variable, can occur anywhere in a command.

Two other, minor, possibly new things are:

(2) proof system rules for recursion, one each for partial and for total correctness, which go somewhat beyond earlier ones;

(3) use of dynamic logic in this context (of both *variable* declarations and of *not* extending the language to an artificial non-deterministic command language).

I shall argue below that (3) is desirable:

(a) not so much for the other types of statements besides Floyd-Hoare correctness statements which can be expressed and derived within dynamic logic (though that surely is of interest—for example, statements that two commands are equivalent);

(b) but equally or more for the fact that all the complications in the papers **[AdB]** **[Apt]** **[Kmn]** **[Nip]** **[Sch]** **[vOh]**, about *counting variables, auxiliary variables, adaptation rules, contexts, correctness phrases, conversion of Hilbert systems to Gentzen systems to make sense of soundness,* etc. (none of which occur here) associated with avoiding 'Apt's unsoundness' are (in my opinion) caused by not doing the natural thing and going to dynamic logic, but insisting on remaining in the straightjacket of the usual Floyd-Hoare statements plus $1^{\underline{st}}$order formulas, plus whatever ad-hoc devices, as above, are needed for progress.

At this point we should discuss Harel's interesting contribution, which occurred before all the references above. It does everything we have done, i.e. versions of (1) to (3) above, with the version of (2) seeming to be completely original, as opposed to our rules, which are largely derivative from others' major ideas. There is just the one exception, viz. "entirely within a deterministic language". Harel finds it necessary to make a considerable extension of the command language, in order to produce a proof system which is (he claims and sketches a proof of) sound and complete. His proof rules for recursion heavily involve the new, non-deterministic objects in this extended language. Although their system really only involves a 'top-left corner language', the authors of **[AdB]** do feel they have improved on this aspect of Harel, despite Harel's language being 'top-right corner', as we read in **[AdB]**, p.131 : "...in our system it is not necessary to *artificially* extend the programming language". (my emphasis) So it is presumably of at least some interest to see here that one can avoid extending and in particular can avoid non-deterministic command languages; and actually do it for the more general situation which

Harel presents (i.e. top-right) and America-deBoer don't.

A minor side issue is that there is likely the need to add one more rule like our (PRE) in order for Harel's system to actually be complete. In [**Hoff**] there is a detailed syntactic proof that some such addition is needed to rescue his earlier system (for iteration rather than recursion) from incompleteness. Brief remarks at the end of that paper indicate why the same minor problem likely occurs with Harel's system above for recursion—but I have no desire at all to sweat out another one of those syntactic proofs! That syntactic proof took some effort, but was motivated only by the desire to explain (in no uncertain terms) to a certain referee why he/she was dead-wrong.

As we mentioned earlier, there is a claim in [**AdB**] for the existence of a dynamic logic system for doing what Harel did but without an unnatural non-deterministic language extension, whereas it seems in fact to be rather less than Harel did, i.e. top-left versus top-right. In any case, further down in this section, we discuss what we take this system to be. (They never quite say what it is, and there are a few other peculiarities about their presentation). However, the recursion rule which they use for *total* correctness *is* very interesting, but we'll leave the discussion of it to a general discussion below of rules (and so-called *meta-rules*) for recursion. The recursion rule which they intend to use for *partial* correctness never shows up in that last section of [**AdB**].

Another point which seems to arise in this context would tend to imply that one has no need to pass to the right-hand side of our diagram, since any command in a language from the top or bottom right can be simulated by a command from the bottom-left, as long as that language is chosen with reasonable care. On the one hand, this is just the remark further up that all the corners except the top-left correspond to Turing-equivalent languages. Those who take that statement as a serious criticism presumably have no interest in any Floyd-Hoare logic beyond writing down the basic system of Hoare for the simplest *while*-language, and proving it sound and (Cook)-complete, and then going home to cut the grass. On the other hand, what would one need to do to actually specify a Turing-equivalent language on the bottom-left? After writing down a BNF (i.e. inductive) definition of the set of commands, presumably something like

$$x \leftarrow\!\!\div t \quad | \quad X_1, X_2, X_3, \cdots \quad || \quad (C; D) \quad | \quad ite(G)(C)(D) \ ,$$

you'd need to specify an effective list of all (or nearly all) commands in the language and then say that the ith command is declared to be the procedure called by the ith procedure variable X_i. Anyone who finds this an attractive

command language to study (as compared to those we talk about here, corresponding to the right-hand side of the diagram from several pages back) is probably not going to find an advocate in me. The need for that listing of procedures seems somewhat beyond the pale as a way of specifying a programming language. The names of those programs would have to be part of the rules in the Floyd-Hoare system.

One suspects that many CSers who have given it a thought might feel that the simulation of the right-hand corners (especially the top-right) by the bottom-left is so simple as to make worrying about the procedure variables actually *varying* (i.e. right-side corners) a waste of time. As well as the relative mathematical ugliness of the fixed declaration languages on the left, and also the earlier quoted remark in **[deBa]** concerning the μ-calculus, the reality of such simulation appears to be not that simple. To do it, one would presumably define, by induction on C, for each command C on the top-right, a command $L(C)$ on the bottom-left as follows: *L would fix assignment commands, and preserve the sequencing* $(;)$ *and ite operators. Atomic (and so free occurrences of) call$X$ would go to some fixed,* always diverging *command (often denoted Ω; we prefer 'gigabug'). For defining $L(\nabla XC)$, one would simply write down X_i, the ith procedure variable, after looking up in the list for which i the procedure variable X_i is declared to be procedure $L(C)$ (already known inductively).* That sounds simple, but this last sentence seems far from the paragon of simplicity. The rejoinder to that might be: "I'm not interested in producing a Turing-equivalent language on the bottom-left beforehand. I'd just look at the various C for which ∇XC occurs as a subcommand in the command for which some verifying needed doing; say they are $C_1, \cdots, C_k$. Then I'd use $X_1, \cdots, X_k$ to be declared as $L(C_1), \cdots, L(C_k)$, respectively, and not lose any sleep over X_i for $i > k$." And to that one might reply: "Since L is defined inductively, it's still no simpler. Furthermore, inventing a 'new' language (and proof system in a sense) for each command to be verified strikes one as not really in the spirit of the subject. Surely one has an ideal of defining a Turing-equivalent language, with the command-construction features for which one wishes to give a sound, Cook-complete system, by a structural inductive definition; and then of writing down a single comprehensive such system."

In the realm of practical program verification, one wonders how awkward a practical programming language, but constrained with respect to recursion as on the bottom-left, would become for verification in the following scenario:

Programmer A writes program α containing lots of separate procedure variables with fixed declarations. Having thereby made his fortune, he retires to Monaco. Then programmer B wishes to write program β, also with lots of separate procedure variables, including one which declares itself to be the previous program α. The resulting tangled web, especially after iterating this to programmer C, etc., would not have happened if the original command language worked as on the bottom-right.

Before getting just a bit more technical in this discursive section, we should discuss the question of 'getting to' the bottom-right corner. This is clearly desirable: although the command language of this paper allows arbitrary local declarations, having only one procedure variable, it does *not* allow for *mutual recursion*. As mentioned several times already, this does not limit it at all as far as computability is concerned, nor does our omission of *while-do* i.e. *iteration*. Harel omits the latter also, and eloquently explains the desire to isolate recursion from 'orthogonal' issues such as iteration, and such as (the hornets' nest of) parameter passing. He also does a nice job of explaining parallels between iteration and recursion. Harel indicates how to extend his results to the bottom-right, of course still needing to expand the command language (in a way that doesn't appeal to the authors of **[AdB]** at least!) So I would like to sometime try for an extension of the results in this paper from the top-right to the bottom-right corner, with no language expansion to non-deterministic (or other) territory. [1] But I shall refrain from making any remark at all (about "easy but tedious") along the lines of the first set of quotes above—not before actually doing it, writing down all the details, and checking them many times (maybe even *multiple* times, to quote the army of adjectival inflators). As well as the rather thorough details of 7 or 8 examples at the end of this section, for my own peace of mind (in pontificating on a subject which is notorious for the ease with which errors can arise), the last three sections here really go overboard with checking details in the proofs, including proofs of several standard propositions for the case of the present command language.

Perhaps I can get well enough educated to be able to re-check these details using a machine, as **[Nip], [Sch]/[Kmn] [vOh]** seem to think desirable, even preferable. (The latter reference appears to be mainly about so-called *local*

[1] See **[mutu]** for this, now done.

variables—with no bound on the size of localities, it would seem that, strictly speaking, every variable is one, so *non-global* might be a better term; these are not procedure variables—and probably entirely about *partial* correctness, though one has to work very hard to figure that out.) But surely we are not talking about something akin to the proof of the 4-colour theorem, which apparently is *not* humanly checkable! Here's another quote, this from [**Sch**] p.1 : "Most published verification calculi for imperative programs dealing with recursive procedures are known to be either unsound or incomplete, despite authors backing up their claims with "proofs" [ref]. No such proof attempts would have been accepted by a mechanical proof checker." One cannot resist adding: "Nor would they have been accepted by a competent referee for a reputable mathematics journal!"

Some history of the subject, starting with Turing and Von Neumann informally, then Floyd, Naur and Hoare in the late 60's, and credit for the major ideas, can be found in many places, for example [**Apt**] and [**Harel**]. It seems that the fundamental idea behind all these recursion rules for partial correctness is due to Dana Scott, though first appearing in a journal article of Hoare. The idea for the recursion rules for total correctness is usually credited to Sokolowski, but Clarke's thesis a year before [**Sok**] mentions O'Donnell as the originator of such a rule. Pratt invented dynamic logic, and [**Harel**] made it into an even more beautiful thing, a minor aspect of his work being the invention of those new, non-deterministic rules for recursive commands to which we alluded. Apt discovered several fundamental things, and, as we discuss below, the paper [**AdB**] has also been important for me for this work. And it should be evident that [**Harel**] has been absolutely essential to this.

The remainder of the section consists of a very long appendix with the examples, preceded by several shorter, more technical, subsections, with no proofs, and not crucial at all for the non-stop proving of the main results (which constitutes the rest of the paper after that). The appendix has examples of deriving wfs's within this paper's system, illustrating a number of the points that come up in the proofs of the last three sections.

A language for (hopefully) extending to the bottom-right.

Continuing from the 4th previous paragraph, (especially for someone else who might like to attempt the job), here is a good notation for a command language when one has a countable infinity of procedure variables.

Amplify X to a sequence of 'procedure variables' $X_1, X_2, \cdots$.

Amplify $callX$ to $callX_1, callX_2, \cdots$.

Replace ∇XC (really the more general $D_{\nabla XC}$) by strings of the form $\nabla C_1 Y_1 C_2 Y_2 \cdots C_k Y_k D$, for various $k \geq 1$, and distinct procedure variables Y_i. This last string is meant to convey

$$\textsf{begin } declare\ Y_1\ to\ be\ C_1\ ;\ \cdots\ ;\ declare\ Y_k\ to\ be\ C_k\ ;\ doD\ \textsf{ end}\ .$$

(The old ∇XC becomes ∇CXC, or at least $C_{\nabla XC}$ does. More generally, the old $D_{\nabla XC}$ becomes ∇CXD.) Because of unique readability, there is no theoretical need for brackets or 'begin-end's in the earlier string. Note also that this would not preclude occurrences of $callX_j$, within D or any of the C_i, which are 'free' when regarded as occurrences in the entire command. Here, "free" would mean the X_j not being one of the Y_i. So the language is given by structural induction (it's BNFable!) as follows

$$x \leftarrow t \ \mid\ callY \ \mid\mid\ (C;D) \ \mid\ ite(G)(C)(D) \ \mid\ \nabla C_1 Y_1 C_2 Y_2 \cdots C_k Y_k D\ ,$$

where Y may be any X_j, and where $Y_1, Y_2, \cdots, Y_k$ are mutually distinct X_j's.

As for the semantics, the function m is defined much as in this paper, with all the free $callX_i$'s as universally diverging, and with that last command having its m-function again defined as a union:

$$m(\nabla C_1 Y_1 C_2 Y_2 \cdots C_k Y_k D) \ := \ \bigcup_{n>0} m(D^{<n>}_{\vec{C}/\vec{Y}})\ ,$$

where, inductively on n,

$$D^{<1>}_{\vec{C}/\vec{Y}} \ := \ D \quad \text{and} \quad D^{<n+1>}_{\vec{C}/\vec{Y}} \ := \ |D|^{callY_1 \mapsto C_1^{<n>}_{\vec{C}/\vec{Y}}\ ,\ callY_2 \mapsto C_2^{<n>}_{\vec{C}/\vec{Y}}\ ,\ -,-,-}$$

We trust (but don't assert) that a fairly minor modification to the system of Chapter 1 will give a sound, Cook-complete system for the dynamic logic which uses this command language and semantics in place of $\mathcal{R}ec$.

There is quite an array of semantic definitions in this subject area. Once one gets into programming with function-declaring, parameter-passing, GOTO's etc., denotational semantics is the trick, or so it seems. Our only extensive reference where the bottom-left type of language with mutual recursion is

used, namely the book [**deBa**], does both a 'step-by-step' operational semantics, and also a denotational semantics, working its way towards command languages with more complications. So we find 15 full pages (pp.146-161) to give the semantics of a language of essentially pure recursion (many procedure variables as on the bottom-left—and admittedly the extra complication of "array variables", but that's not the reason for the lengthy definition of the semantics). But then, in the middle of the following section on the proof system, an axiom as simple (and obviously true under any cogent semantics) as our $(\mathsf{AX})_{disj}$ turns out to be invalid in that choice of semantics, and a lengthy modification must be added to make it true at all states—(see pp.188-196).

This can be a thorny subject. Readability isn't helped by notation which would wish to write (using three copies of "=" with three different meanings and bragging about it) as "$\mathcal{W}(t_1 = t_2)(s) = (\mathcal{V}(t_1)(s) = \mathcal{V}(t_2)(s))$" our "$t_1 \approx t_2\ \mathsf{tt}@s \iff s(t_1) = s(t_2)$".

The dynamic logic system in [AdB].

As mentioned earlier, in the next subsection we shall discuss their interesting recursion rule for '$< C >$' (not for '$[C]$', since that one is not given, for whatever reason). The main qualitative aspect which needs to be noticed here about their rule is that [**AdB**]'s two antecedents are both just plain wfs's—they do not involve the 'derivability verb $\vdash$' or any subsidiary one, such as our '$\vdash^w$', which all earlier such rules exhibit, other than Harel's non-deterministic ones. Although the main contrast between the [**AdB**] purported dynamic logic system and ours is the 'top-left versus top-right' business discussed above, a quite fundamental matter is that of whether or not what one uses as a recursion rule is, as we do, something called by Apt and others a *metarule*. Although unsuccessful in discovering such a rule, I assume the corresponding recursion rule for $[C]$, hinted at but not given in [**AdB**], is similarly a *non*-meta-rule, just like their rule for $< C >$, since they say (p.156): "...we only give axioms and rules dealing with the $< \cdot >$ operator. The axioms and rules which formalize reasoning about the $[\cdot]$ operator are similar." This is in contrast with an earlier statement, when working with the usual (in my opinion awkward) Floyd-Hoare language (p.131) : "...we may conclude that reasoning about total correctness differs from partial correctness in a substantial way which has not been recognized till now." But perhaps this is just another illustration of the mathematically more perspicuous dynamic logic, for which I am trying

to make a case.

There are some related puzzling aspects of their write-up on the dynamic logic system, likely caused by the need for brevity in a research journal. The operator '[ ]' is given as a primitive symbol, and nowhere is there mentioned its duality with the operator '< >'. Presumably this would be done with an additional axiom, making quite redundant most of the "...rules which formalize reasoning about the [·] operator ..." from above. In the same vein, referring to their THEOREM 6.9, which is essentially the same as our **23** in their context, they say (p.158) "Completeness follows from the following theorem, see (Harel, 1979)." That seems to make any hard work on an analogue of our **22** unnecessary, something which I find rather doubtful, both from my own efforts and from reading Harel's excellent write-up.

A final small quibble is that, though (MP) is included in their system, the axioms (TAUT) are not, presumably an oversight, since they refer to "propositional reasoning" no less than six times in later sketchy descriptions of derivations.

Leaving aside the matter about the [ ]-operator, it is still somewhat difficult to determine what the rules and axioms of their dynamic logic system actually are. They suddenly say, after writing down a bowdlerized version of Harel's system, and a few newer rules (p.158) : "This new system can be proved to be sound by a straightforward induction on the length of the derivation. The soundness of the Recursion Rule is established in a similar way as the corresponding rule of the system manipulating correctness phrases ..." So I will take their "new system" to (roughly!) consist of all the axioms and rules written down in that section, except that the rules labeled "Assertion" and "Recursion" (p.157) are removed (since they involve Harel's non-deterministic construct) with some indeterminate axiom(s) and/or rule(s) added related to $[C]$, i.e. to partial correctness. The recursion rule referred to in the quote is the later one (p.158), an analogue of which we discuss below. In any case, their rules and axioms unrelated to recursion have many similarities to our list, differing mainly because of this curious non-treatment of partial correctness. I have cadged plenty of small ideas and inspiration from [**AdB**], without which the present paper would not exist.

Various recursion rules, what (one sometimes guesses) is intended, and a conjecture.

As given in **Theorem 18** of Section 1, what is intended by $\vdash$ for the system in the present paper is quite straightforward. Because the 'proto' versions, $\vdash'$ and $\vdash^w$, used in the two recursion rules, correspond to uncomplicated Hilbert-type derivability, and are distinct from $\vdash$ itself (avoiding any sort of "self-reference"), what is meant by deriving in our system doesn't need to be thought about too deeply.

Except for [**AdB**]'s mentioned above and discussed in the next paragraph, all other deterministic recursion rules that I have seen, both for partial and for total correctness, seem to use the overall $\vdash$ itself in one of the antecedents. An example would be $(\mathsf{RCSSS})_{[\,]}$ mentioned in the previous section. The self-referential aspect is this: rules which are supposed to define the relation $\vdash$ themselves refer to a conditional derivation in the $\vdash$-sense. So quite a rigmarole of conversion to a Gentzen system, introduction of new concepts such as *contexts* and *correctness phrases* is apparently necessary, just to explain what is meant by $\vdash$ and to formulate a proof of soundness. The phrase "seem to" above indicates some uncertainty on my part—in [**Apt**], p.444, we read "infer $\{p\}P\{q\}$ from the fact that $\{p\}S_0\{q\}$ can be proved **(using the other rules and axioms)** from the assumption $\{p\}P\{q\}$."(my emphasis) Nowhere else in that paper or any I've seen has the "other" word occurred, so I suspect that this was not intended as part of the definition, just an observation that that's the way it usually happens. Sometimes informality with a fair amount of syntax/semantics conflation leads to difficulty of interpretation. Another example of ambiguity is that, if one is using a system with such a recursion rule to give a derivation witnessing $\Gamma \vdash \mathcal{S}$ where $\Gamma \neq \emptyset$, is one allowed to use the objects in Γ when doing an auxiliary derivation to verify the antecedent in some instance of the recursion rule? Probably not, I'd guess.

Now I shall write down a recursion rule for total correctness which is very close to the dynamic logic one at the end of [**AdB**], not having any turnstyles at all in its antecedents. It differs because our language is for the more general type of recursion from the top-right corner of the diagram early in this section (and the slightly bizarre universal quantifier notation should not be blamed on them!) We have singled out a variable w, and the notation $\forall\ \not{w}$ means $\forall y_1 \forall y_2 \cdots \forall y_k$ for some sequence $\vec{y}$ of variables which does *not* include w, but *does* include every other variable occurring free in the wfs's

immediately following (the 'scopes'). Here is that (non-meta-)rule:

$$\frac{\neg F^{[w\to 0]} \ , \ \forall\not{w}(\ F \to < \nabla XC > G) \ \to \ \forall\not{w}(F^{[w\to w+1]} \to < \nabla XC > G\ )}{(\exists w F) \to < \nabla XC > G}$$

This seems interesting for several reasons:
it is certainly valid;
it does *not* seem to be enough (even after being generalized by including our Ω), as a replacement for our $(\mathsf{RCS})_{< >}$, to produce a complete system (this may be an indication of the extra subtleness of the top-right vs. top-left, i.e. us vs. [**AdB**]);
the version of this in which a "$\vdash^w$" is placed in front of the first $\forall\not{w}$ can be deduced as a derived rule from our $(\mathsf{RCS})_{< >}$;
similarly the rule displayed above can be derived from what you get by replacing $\vdash^w$ by just-plain $\vdash$ in $(\mathsf{RCS})_{< >}$;
yet I suspect that the latter rule is **invalid**!

The last comment could be misinterpreted as an expression of skepticism about the soundness of all the rules for total correctness earlier (except, of course, the 'non-meta-' ones, Harel's and the dynamic logic one in [**AdB**]). However, their soundness (often undiscussed, and when it is, almost never proved) and the above one's unsoundness could again just be a reflection of the top-left vs. top-right differences.

Conjecture. *There is no sound and Cook-complete proof system for our dynamical logic language* (semantics in **N** or in arbitrary arithmetical interpretations—see [**Harel**]), *each of whose rules has just a* (finite) *set of wfs's as antecedents* (e.g. no conditional derivations using $\vdash'$ or $\vdash^w$, etc.), *and whose non-oracular rules form a decidable system.*
If true and [**AdB**]'s system really does have that form, one would have a dramatic contrast here between the left and right sides of the early diagram from this section. It would also serve to emphasize Harel's cleverness in extending the language.

A more compact version of our system.

First consider

$$(\mathsf{AX})_{\forall} \qquad\qquad \emptyset \ / \ \forall x\mathcal{A} \ \longrightarrow \ |\mathcal{A}|^{x\to t} \ .$$

This axiom is an obvious extension of a very common one in plain $1^{\underline{\text{st}}}$order logic. Recall our substitution convention that **no command in $\mathcal{A}$ should have x as a program variable**. Validity of this axiom is then easy to establish. Now $(\mathsf{AX})_{\forall}$ can replace $(\mathsf{SUB})_1$ in our system, using the derived rule $\mathcal{A}/\forall x\mathcal{A}$ to derive $(\mathsf{SUB})_1$. And $(\mathsf{SUB})_2$ can be derived from $(\mathsf{SUB})_1$.

Generally speaking, one prefers axioms to other rules as often as possible: truth at states is preserved if $\mathcal{A} \to \mathcal{B}$ is valid, but only 'in $\mathbf{N}$' by the corresponding valid rule $\mathcal{A}/\mathcal{B}$. So we beef up (AND) to

$$(\mathsf{AX})_{[AND]} \qquad \emptyset \,/\, (\mathcal{A} \to [D]\mathcal{B}) \wedge (\mathcal{A}' \to [D]\mathcal{B}') \;\to\; (\mathcal{A}\wedge\mathcal{A}' \to [D](\mathcal{B}\wedge\mathcal{B}')) \,.$$

Removing also the derivable $(\mathsf{AX})_{[ite]}$, and sorted in a semi-rational fashion, here is our more compact version of the system of this paper:

(ORACLE)

(TAUT) (MP)

$(\mathsf{AX})_{<\leftrightarrow>}$ $(\mathsf{AX})_{<;>}$ $(\mathsf{AX})_{<ite>}$

$(\mathsf{UNAR})_{\forall}$ $(\mathsf{UNAR})_{<\,>}$ (PRE)

$(\mathsf{AX})_{\forall}$ $(\mathsf{AX})_{disj}$ $(\mathsf{AX})_{[AND]}$ $(\mathsf{AX})_{<call>}$ $(\mathsf{AX})_{<\to>}$

$(\mathsf{AX})_{<\nabla>}$

$(\mathsf{RCS})_{[\,]\to}$ $(\mathsf{RCS})_{<\,>}$

All but the last two lines is the system defining $\vdash'$.
All but the last line is the system defining $\vdash''$.
The latter system, with added restrictions for the three non-axiom rules on the 4th line as to the non-occurrence of w, is the system defining $\vdash^w$. (It seems like cheating that 'now' we have no need to worry about checking uses of $(\mathsf{SUB})_1$ and $(\mathsf{SUB})_2$ to be sure that a derivation shows $\vdash^w$, not merely $\vdash''$. But that's an illusion—any case of a derived rule must be carefully vetted; in this instance, a use of $(\mathsf{UNAR})_{\forall}$ to derive $\mathcal{A}/\forall x\mathcal{A}$, to then derive $(\mathsf{SUB})_1$ and $(\mathsf{SUB})_2$, must have $x \neq w$, for $\vdash^w$ to hold!)
The second line gives 'propositionality'.
The third line consists of the 'Hoare's innocuous command rules'.
The last line consists of the 'not-so-innocuous command rules' for recursion.
The 4th line has the only non-axioms (other than the recursion rules). They definitely don't preserve truth 'at a state', only 'in $\mathbf{N}$'.

The penultimate line is the first which has explicit reference to ∇XC, so must be excluded in defining $\vdash'$.

I will leave it to senility to try paring this system down further, and to the CS experts as to any practicality aspects for actually verifying programs. (My interest seems to be strictly correctness and 'good taste' from the pure mathematics point of view.) By looking through the proofs of **22** and **23** respectively, one can extract what part of the above system is needed for partial correctness, and, respectively, what part for total correctness. My impression is that there is a great desire for exactly one rule to apply for each command construct, so that the only creativity left is selecting the relevant $1^{\underline{\text{st}}}$order formulas ("intermediate assertions"), especially when one is engaged in the activity of simultaneously creating and verifying software.

Adaptation Rules in [Nip] and [Sch].

The grandaddy of such rules in the usual Floyd-Hoare language is

$$\frac{F' \to F \ , \ F\{C\}G \ , \ G \to G'}{F'\{C\}G'} \ .$$

Rewriting in dynamic logic language will see this as merely propositional—a double application of hypothetical syllogism, (HYSY). So here are three propositional rules, easily checked by truth tables for validity, the first being essentially a way of expressing 'double-(HYSY)' as above:

$$\frac{\mathcal{A} \to \mathcal{B} \ \ , \ \ (\mathcal{A}' \to \mathcal{A}) \wedge (\mathcal{B} \to \mathcal{B}')}{\mathcal{A}' \to \mathcal{B}'} \ ;$$

$$\frac{\mathcal{A} \to \mathcal{B} \ \ , \ \ \mathcal{A}' \to \mathcal{A} \wedge (\mathcal{B} \to \mathcal{B}')}{\mathcal{A}' \to \mathcal{B}'} \ ;$$

$$\frac{\mathcal{A} \to \mathcal{B} \ \ , \ \ (\mathcal{A} \to \mathcal{B}) \to (\mathcal{A}' \to \mathcal{B}')}{\mathcal{A}' \to \mathcal{B}'} \ .$$

(Actually, this last is just some instances of (MP), isn't it?)

In language which is somewhat opaque to non-experts like me, the papers **[Sch]/[Kmn], [Nip]** write out rules based on the 2nd and 3rd above respectively, but with certain additional quantifiers and forms of substitution, and using a kind of mixture of semantics and syntax which is challenging for someone not embedded right in that group of machine-assisted theorem-provers.

The new rule in each paper is asserted to suffice to replace not only the earlier adaptation rule, but to also be adequate to replace three substitution rules introduced by [**Apt**] in order to get a complete system for 'top-left corner' recursion, recast with restrictions by [**AdB**] in order to repair the unsoundness thereby introduced.

And so the systems of [**Sch**]/[**Kmn**], [**Nip**] are claimed to be much simpler to use in 'practical' program verification, I think because they are closer to the desideratum vaguely proffered in the final sentence of the previous section. But experts may not agree and I'd like to hear.

In any case, it may be possible, by stealing these ideas from the papers [**Sch**]/[**Kmn**], [**Nip**], to simplify our compact system of the previous subsection some more. But a purely mathematical treatment can differ from practical program verification, and one would hope that the former style would have value in attempts to extend results to new territory. I'm more interested right now in the problem of extending, from the 'bottom-left' to the 'bottom-right', the accomplishment of [**Nip**] in giving a good system that works for a command language with unbounded non-determinism.

Our diagram at the beginning of this section, about a dozen pages back, has served us well.

APPENDIX: Examples of derivations.

We shall do both partial and total correctness derivations, suppressing very few details, for the simplest cases we can find to illustrate various points in the lemmas related to **22** and **23** in the 1st section, and to their proofs in the 4th section. There are also a few general examples.

The first example just below is a command used by a nervous idiot who just wants to change the value of y from 0 to 1, and do nothing if it isn't 0 to begin with. But after doing it, he nervously and unnecessarily goes back and checks that it really isn't 0 anymore, using, of all things, a recursive command to achieve all this. This example is supposed to illustrate:

(1) just how tedious writing out a derivation can be, considering how utterly trivial this command is;

(2) more-or-less how to proceed with a $\nabla X(ite\text{-command})$;

and also

(3) how, at the recursion rule antecedent verifying stage in these deriva-

tions, the string ∇XC plays the role really of just one 'abstract' symbol—in the partial, i.e. $[\nabla XC]$, case this seems clear from the discussion of $(\mathsf{RCS})_{[\ ]\rightarrow}$, but the same happens for the total, i.e. $<\nabla XC>$, case.

These points hold for all our examples. In later ones, we'll use our experience here to suppress some details. Here we shall give all the lines in the derivation, except three times where there is a 2nd justification $(\leftrightarrow)$. Which lines are referred back to in the justifications is pretty obvious, except that the line $(\mathsf{P})_2$ uses both lines $(\mathsf{P})_1$.

Recall the *bugga*-command from the beginning of this section, a command which does nothing, by assigning some variable to itself.

Example 1. With

$$C \ := \ ite(y \approx 0)(y \dot{\leftarrow} y + 1 \ ; \ callX)(bugga) \ ,$$

and

$$A \ := \ (0 \approx z \wedge y \approx 1) \vee (0 < z \wedge y \approx z) \ ,$$

we show that $y \approx z \ \rightarrow \ [\nabla XC]A$ can be $\vdash$-derived, i.e. derived by our overall system. It's a wfs whose truth in $\mathbf{N}$ is easy to check, and, of course, it seems to be just what the idiot wants, at least if we re-did it for total correctness. As it happens, the formula A 'is' in fact, $A(\nabla XC, y \approx z)$, so the derivation below illustrates **22.1** and its proof.

Using $(\mathsf{RCS})_{[\ ]\rightarrow}$ with Ω empty, the required derivation is immediate from a derivation witnessing

$$\{y \approx z \ \rightarrow \ [\nabla XC]A\} \ \vdash' \ y \approx z \ \rightarrow \ [C_{\nabla XC}]A \ .$$

Just below is that derivation. For its penultimate line, note that

$$C_{\nabla XC} \ = \ ite(y \approx 0)(y \dot{\leftarrow} y + 1 \ ; \ \nabla XC)(bugga) \ .$$

That 2nd last line should be read first, noticing how the earlier part is just two separate threads aiming respectively for the two lines justified by $(\mathsf{P})_1$. The first thread is quick and trivial, whereas the second is not-so-quick, but essentially trivial. In any case, a bunch of the rules get somewhat illustrated.

After reading a few more examples, the reader may enjoy the exercise of returning to this one, but deriving the corresponding *total* correctness fact.

$A \;\rightarrow\; [bugga]A$ $\quad$ $(\mathsf{AX})_{\dot{\leftarrow}}$

$y \approx z \wedge \neg y \approx 0 \;\rightarrow\; A$ $\quad$ (ORACLE)

$y \approx z \wedge \neg y \approx 0 \;\rightarrow\; [bugga]A$ $\quad$ (HYSY)

$y \approx z \rightarrow (\neg y \approx 0 \;\rightarrow\; [bugga]A)$ $\quad$ $(\mathsf{P})_1$

$y \approx z \;\rightarrow\; [\nabla XC]A$ $\quad$ (premiss)

$y \approx z+1 \;\rightarrow\; [\nabla XC]((0 \approx z+1 \wedge y \approx 1) \vee (0 < z+1 \wedge y \approx z+1))$ $(\mathsf{SUB})_1^{z \mapsto z+1}$

$(0 \approx z+1 \wedge y \approx 1) \vee (0 < z+1 \wedge y \approx z+1) \;\rightarrow\; y \approx z+1$ (ORACLE)

$[\nabla XC]((0 \approx z+1 \wedge y \approx 1) \vee (0 < z+1 \wedge y \approx z+1)) \rightarrow [\nabla XC]y \approx z+1$ $(\mathsf{UNAR})_{[\,]}$

$y \approx z+1 \;\rightarrow\; [\nabla XC]y \approx z+1$ (HYSY)

$[y \dot{\leftarrow} y+1]y \approx z+1 \;\rightarrow\; [y \dot{\leftarrow} y+1][\nabla XC]y \approx z+1$ $(\mathsf{UNAR})_{[\,]}$

$y \approx z \;\rightarrow\; [y \dot{\leftarrow} y+1]y \approx z+1$ $\quad$ $(\mathsf{AX})_{\dot{\leftarrow}}$, $(\leftrightarrow)$, (ORACLE) and (HYSY)

$y \approx z \;\rightarrow\; [y \dot{\leftarrow} y+1][\nabla XC]y \approx z+1$ (HYSY)

$[y \dot{\leftarrow} y+1][\nabla XC]y \approx z+1 \;\rightarrow\; [(y \dot{\leftarrow} y+1 \;;\; \nabla XC)]y \approx z+1$ $(\mathsf{AX})_{[;]}$ $(\leftrightarrow)$
$y \approx z \;\rightarrow\; [(y \dot{\leftarrow} y+1 \;;\; \nabla XC)]y \approx z+1$ (HYSY)

$y \approx z \wedge z \approx 0 \;\rightarrow\; [(y \dot{\leftarrow} y+1 \;;\; \nabla XC)](y \approx z+1 \wedge z \approx 0)$ $(\mathsf{AX})_{[\rightarrow]}$

$y \approx z+1 \wedge z \approx 0 \;\rightarrow\; A$ (ORAC)

$[(y \dot{\leftarrow} y+1 \;;\; \nabla XC)](y \approx z+1 \wedge z \approx 0) \;\rightarrow\; [(y \dot{\leftarrow} y+1 \;;\; \nabla XC)]A$ $(\mathsf{UNAR})_{[\,]}$

$y \approx z \wedge z \approx 0 \;\rightarrow\; [(y \dot{\leftarrow} y+1 \;;\; \nabla XC)]A$ (HYSY)

$y \approx z \wedge y \approx 0 \;\rightarrow\; y \approx z \wedge z \approx 0$ (ORACLE)

$y \approx z \wedge y \approx 0 \;\rightarrow\; [(y \dot{\leftarrow} y+1 \;;\; \nabla XC)]A$ (HYSY)

$y \approx z \rightarrow (y \approx 0 \;\rightarrow\; [(y \dot{\leftarrow} y+1 \;;\; \nabla XC)]A)$ $\quad$ $(\mathsf{P})_1$

$y \approx z \rightarrow (y \approx 0 \rightarrow [(y \dot{\leftarrow} y+1 \;;\; \nabla XC)]A) \wedge (\neg y \approx 0 \rightarrow [bugga]A)$ $\quad$ $(\mathsf{P})_2$

$(y \approx 0 \rightarrow [(y \dot{\leftarrow} y+1 \;;\; \nabla XC)]A) \wedge (\neg y \approx 0 \rightarrow [bugga]A) \;\rightarrow\; [C_{\nabla XC}]A$ $(\mathsf{AX})_{ite}$ $(\leftrightarrow)$

$y \approx z \;\rightarrow\; [C_{\nabla XC}]A$ (HYSY)

Next we shall verify, separately for partial and for total correctness, the fundamental facts about the 'proper subtraction' command mentioned early in this section. This is semi-general, perhaps helping to expose what are really the issues in both derivations. The definition is, with $x \neq y$,

$$\text{`}y \leftarrow\!\div y - 1\text{'} \;:=\; x \leftarrow\!\div 0 \;;\; \nabla XC \;;\; y \leftarrow\!\div x \;,$$

where

$$C \;:=\; ite(x + 1 < y)(x \leftarrow\!\div x + 1; callX)(bugga) \;.$$

And (the semi-generality is that for *any* G in which the x used to produce '$y \leftarrow\!\div y - 1$' does not appear) we shall derive both

$$(\mathbf{2}) \qquad G \wedge 0 < y \;\rightarrow\; [\text{`}y \leftarrow\!\div y - 1\text{'}]|G|^{y \rightarrowtail y+1}$$

and

$$(\mathbf{3}) \qquad G \wedge 0 < y \;\rightarrow\; <\text{`}y \leftarrow\!\div y - 1\text{'}> |G|^{y \rightarrowtail y+1} \;.$$

Ex. 2. As for the first, partial correctness, verification, directly from $(\mathsf{AX})_{\leftarrow\!\div}$, we can derive

$$|G|^{y \rightarrowtail x+1} \;\rightarrow\; [y \leftarrow\!\div x]|G|^{y \rightarrowtail y+1} \;,$$

and therefore, by $(\mathsf{UNAR})_{[\ ]}$ and $(\mathsf{AX})_{[;]}$,

$$[x \leftarrow\!\div 0][\nabla XC]|G|^{y \rightarrowtail x+1} \;\rightarrow\; [\text{`}y \leftarrow\!\div y - 1\text{'}]|G|^{y \rightarrowtail y+1} \;.$$

From $(\mathsf{AX})_{\leftarrow\!\div}$, we can derive

$$G \wedge 0 < y \;\rightarrow\; [x \leftarrow\!\div 0]G \wedge x < y \;.$$

So it remains to derive

$$G \wedge x < y \;\rightarrow\; [\nabla XC]|G|^{y \rightarrowtail x+1} \;.$$

Note that x *is*, whereas y *isn't*, a program variable in ∇XC, whereas, in '$y \leftarrow\!\div y - 1$', both are. Our experience with the previous example tells us that we should use $(\mathsf{RCS})_{[\ ]\rightarrow}$ to deal with $[\nabla X(\text{this } ite \text{ command})]$. And, since $C_{\nabla XC}$ is an *ite*-command, we can reduce the required job, of getting a derivation witnessing

$$\{G \wedge x < y \;\rightarrow\; [\nabla XC]|G|^{y \rightarrowtail x+1}\} \;\vdash'\; G \wedge x < y \;\rightarrow\; [C_{\nabla XC}]|G|^{y \rightarrowtail x+1} \;,$$

to getting derivations for both

$$\vdash' \ G \wedge x < y \wedge \neg x + 1 < y \ \rightarrow \ [bugga]|G|^{y \rightarrowtail x+1} \qquad \text{(i)}$$

and

$$\{G \wedge x < y \ \rightarrow \ [\nabla XC]|G|^{y \rightarrowtail x+1}\} \vdash' \ G \wedge x < y \wedge x+1 < y \ \rightarrow \ [x \leftarrow\!\div x+1][\nabla XC]|G|^{y \rightarrowtail x+1} \quad \text{(ii)}$$

As for (i), because $(x < y \wedge \neg x + 1 < y \rightarrow y \approx x + 1)$ is **ttN**, and by the innocuity of *bugga*, this comes down to deriving

$$G \wedge y \approx x + 1 \ \rightarrow \ |G|^{y \rightarrowtail x+1} \ ,$$

which is immediate from (ORACLE).

As for (ii), by $(\mathsf{AX})_{[\leftarrow\!\div]}$ we can immediately $\vdash'$-derive (recalling that G has no occurrences of x)

$$G \wedge x + 1 < y \ \rightarrow \ [x \leftarrow\!\div x + 1](G \ \wedge \ x < y) \ .$$

Having noted that $G \ \wedge \ x < y \ \wedge \ x + 1 < y \ \rightarrow \ G \wedge x + 1 < y$ is true in **N**, we are left only to $\vdash'$-derive

$$[x \leftarrow\!\div x + 1](G \ \wedge \ x < y) \ \rightarrow \ [x \leftarrow\!\div x + 1][\nabla XC]|G|^{y \rightarrowtail x+1} \ ,$$

which is just the premiss with $(\mathsf{UNAR})_{[\ ]}$ applied to it. Of course there are a few (HYSY)'s happening above. So that finishes the job.

Ex. 3. As for the second, total correctness, verification, first copy everything down we said for the partial case, up to and including the displays (i) and (ii). Now make some alterations in what you just wrote: First change all the [ ]'s to $<\ \ >$'s. Since it is $(\mathsf{RCS})_{<\ >}$ which will be used, next change (i) and (ii) and also the display immediately before them by saying instead that we must discover a $1^{\underline{\text{st}}}$order $F =$ '$F(w, x, y)$' such that (recall that w is neither x nor y)

(a) $G \ \wedge \ x < y \ \rightarrow \ \exists w F$ is **ttN**;

(b) $\neg|F|^{w \rightarrowtail 0}$ is **ttN**; and

(c) $\{F \rightarrow\!< \nabla XC > |G|^{y \rightarrowtail x+1}\} \vdash^{w} |F|^{w \rightarrowtail w+1} \rightarrow\!< C_{\nabla XC} > |G|^{y \rightarrowtail x+1}$.

The point is that (b), plus (c) with an application of $(\mathsf{AX})_{\nabla}$, will verify the two antecedents in an instance of $(\mathsf{RCS})_{<\ >}$ with Ω empty which immediately

derives the wfs to the left of $\vdash^w$ in (c). And so (a) gives us, after applying (PRE) and (HYSY), what we really want, namely that

$$G \wedge x < y \ \rightarrow \ < \nabla XC > |G|^{y \rightarrow x+1}$$

is $\vdash$-derivable.

So we take $F \ := \ G \wedge x < y \wedge w \approx y$. This makes (a) and (b) quite trivial to check.

As for (c), once again $C_{\nabla XC}$ is an *ite*-formula, so exactly the same arguments as the two examples we've already done in this subsection, with $\{[\ ]\ ,\ \vdash'\}$ changed to $\{<\ \ >\ ,\ \vdash^w\}$, show that it suffices to check

$$\vdash^w \ G \wedge x < y \wedge w+1 \approx y \wedge \neg x+1 < y \ \rightarrow \ < bugga > |G|^{y \rightarrow x+1} \qquad (1)'$$

and

$$\{G \wedge x < y \wedge w \approx y \ \rightarrow \ < \nabla XC > |G|^{y \rightarrow x+1}\} \ \vdash^w$$

$$G \wedge x < y \wedge w+1 \approx y \wedge x+1 < y \ \rightarrow \ < x \dot{-} x+1 >< \nabla XC > |G|^{y \rightarrow x+1} \quad (2)'$$

Because the formula $(x < y \wedge w + 1 \approx y \wedge \neg x + 1 < y \ \rightarrow \ y \approx x + 1)$ is true in $\mathbf{N}$, and by the innocuousness of *bugga*, producing a derivation for $(1)'$ [again as in (1)] comes down to deriving, and so, to observing the obvious truth in $\mathbf{N}$ of $(G \wedge y \approx x + 1 \ \rightarrow \ |G|^{y \rightarrow x+1})$.

A $\vdash^w$-derivation for $(2)'$ is more involved and interesting. By $(\mathsf{AX})_{<\dot{-}>}$ we can almost immediately derive

$$G \wedge x < y \wedge w+1 \approx y \wedge x+1 < y \ \rightarrow \ < x \dot{-} x+1 > (G \wedge x < y+1 \wedge w+1 \approx y \wedge x < y) \,,$$

and then

$$G \wedge x < y \wedge w+1 \approx y \wedge x+1 < y \ \rightarrow \ < x \dot{-} x+1 > (G \wedge x < y \wedge w+1 \approx y)$$

basically because $(x < y + 1 \wedge w + 1 \approx y \wedge x < y \ \rightarrow \ x < y \wedge w + 1 \approx y)$ is true in $\mathbf{N}$. We are left only to $\vdash^w$-derive

$$< x \dot{-} x+1 > (G \wedge x < y \wedge w+1 \approx y) \ \rightarrow \ < x \dot{-} x+1 >< \nabla XC > |G|^{y \rightarrow x+1}$$

from the premiss.

First use $(\mathsf{UNAR})_{<\ >}$ to delete both $< x \dot{-} x + 1 >$'s.

Now it seems to be necessary to split up "$x < y$" , that is, to use the fact that $(x < y \ \longleftrightarrow x + 1 < y \vee x + 1 \approx y)$ is true in $\mathbf{N}$. With a bit of good

faith, the oracle and propositional arguments, we become convinced that it suffices to separately $\vdash^w$-derive both

$$G \wedge x+1 < y \wedge w+1 \approx y \ \rightarrow \ < \nabla XC > |G|^{y \to x+1}$$

and

$$G \wedge x+1 \approx y \wedge w+1 \approx y \ \rightarrow \ < \nabla XC > |G|^{y \to x+1} \ .$$

We need the premiss only for the former of these.

For the latter, send arrows from

$G \wedge x+1 \approx y \wedge w+1 \approx y$ to
$G \wedge x+1 \approx y$ to
$\neg x+1 < y \wedge G \wedge x+1 \approx y$ to
$< \nabla XC > (\neg x+1 < y \wedge G \wedge x+1 \approx y)$ to
$< \nabla XC > |G|^{y \to x+1}$,

asserting we can $\vdash^w$-derive all four of the resulting '$\rightarrow$-wfs's. The first two are just oracular of course. The next combines **Ex. 6** ahead with $(\mathsf{AX})_{<\rightarrow>}$. The last combines (ORAC) and $(\mathsf{UNAR})_{<\ >}$.

For the former, there's a bit of substitution shenanigans that need to happen, so (abbreviating $|G|^{y \to x+1}$ to just H), we'll write down a $\vdash^w$-derivation. In the derivation, the variable u is chosen distinct from any variable which has occurred already anywhere in human descriptions of the (Platonic) universe. The second line depends on the fact that y is not occurring in $< \nabla XC > H$. The fifth line depends on the fact that u also is not occurring in $< \nabla XC > H$. The third line depends on the fact that neither of them are in ∇XC.

$G \wedge x < y \wedge w \approx y \ \rightarrow \ < \nabla XC > H$ (premiss)

$G \wedge x < u \wedge w \approx u \ \rightarrow \ < \nabla XC > H$ $(\mathsf{SUB})_1^{y \to u}$

$G \wedge x < u \wedge w \approx u \wedge u+1 \approx y \ \rightarrow \ < \nabla XC > (H \wedge u+1 \approx y)$ $(\mathsf{AX})_{<\rightarrow>}$ (MP)

$G \wedge x < u \wedge w \approx u \wedge u+1 \approx y \ \rightarrow \ < \nabla XC > H$ (ORACLE) $(\mathsf{UNAR})_{<\ >}$ (HYSY)

$\exists u(G \wedge x < u \wedge w \approx u \wedge u+1 \approx y) \ \rightarrow \ < \nabla XC > H$ (PRE)

$(G \wedge x+1 < y \wedge w+1 \approx y) \ \rightarrow \ \exists u(G \wedge x < u \wedge w \approx u \wedge u+1 \approx y)$ (ORACLE)

$G \wedge x+1 < y \wedge w+1 \approx y \ \rightarrow \ < \nabla XC > H$ (HYSY)

These 'u-shenanigans' will occur in a few other total correctness examples below, and will be seen in parts of the proofs of **23.1** and **23.3** in Section 4. The parallel between the partial and total cases is quite noticeable, though the latter is naturally a bit harder—you have to discover an F that will work and then make it work. Also, as noted earlier, in the end game, the string ∇XC is treated as an 'abstract symbol/black box' both times.

The most standard elementary program correctness example for either iteration or recursion is a command which keeps deceasing the value of a variable by 1 until it gets it down to 0, and then it quits. For us, this command is ∇XC for $C := ite(0 < y)(\text{`}y \dot{\leftarrow} y - 1\text{'} \ ; \ callX)(bugga)$. So we will derive both

$$(\mathbf{4}) \quad [\nabla XC]y \approx 0 \quad \text{and} \quad (\mathbf{9}) \quad < \nabla XC > y \approx 0 \ .$$

Ex. 4. Now, in the case of partial correctness, the '$y \dot{\leftarrow} y - 1$' is a red herring. Clearly (and easily verified) if $C := ite(0 < y)(D \ ; \ callX)(bugga)$ for *any* D, then $[\nabla XC]y \approx 0$ is $\mathbf{ttN}$ (whereas $< \nabla XC > y \approx 0$ isn't in general, as taking $D = bugga$ easily shows!) To derive that partial correctness fact within our system is fun and easy, as follows.

Firstly, we'll derive $1 \approx 1 \rightarrow [\nabla XC]y \approx 0$, and employ (MP), feeling rather assured that the oracle will tell us how to derive $1 \approx 1$! To do the derivation, we shall of course employ $(\mathsf{RCS})_{[\ ]\rightarrow}$ once again, reducing the job to showing

$$\{1 \approx 1 \rightarrow [\nabla XC]y \approx 0\} \ \vdash' \ 1 \approx 1 \rightarrow [C_{\nabla XC}]y \approx 0 \ .$$

And, as we've been observing in earlier examples, we need

$$\vdash' \ 1 \approx 1 \wedge \neg 0 < y \ \rightarrow \ [bugga]y \approx 0 \qquad (1)$$

and

$$\{1 \approx 1 \rightarrow [\nabla XC]y \approx 0\} \ \vdash' \ 1 \approx 1 \wedge 0 < y \ \rightarrow \ [D][\nabla XC]y \approx 0 \quad (2)$$

As for (1), because $(1 \approx 1 \wedge \neg 0 < y \ \rightarrow \ y \approx 0)$ is $\mathbf{ttN}$, and by the innocuousity of $bugga$, this comes down to deriving $y \approx 0 \ \rightarrow \ y \approx 0$, which is left as an exercise!

As for (2), apply (HYSY) a couple of times to the wfs's

$$1 \approx 1 \wedge 0 < y \ \rightarrow \ 1 \approx 1 \ \ ; \ \ 1 \approx 1 \ \rightarrow \ [D]1 \approx 1 \ \ ; \ \ [D]1 \approx 1 \ \rightarrow \ [D][\nabla XC]y \approx 0 \, .$$

These three are derived, respectively, using (ORACLE) , $(\mathsf{AX})_{disj}$, and $(\mathsf{UNAR})_{[\ ]}$ applied to the premiss.

To do total correctness for the particular $D =$ '$y \dot{\leftarrow} y - 1$' also gives us the opportunity to articulate a few easy generalities.

Ex. 5. $\vdash'$ $(\neg H \rightarrow < ite(H)(E)(bugga) > \neg H)$ *for any quantifier-free* H *and any command* E*. The same holds also for* $\vdash^w$ *for any variable* w*.*

Here it's a straight *ite*-command, and the usual two checks amount to deriving $(\neg H \wedge \neg H \ \rightarrow \ < bugga > \neg H)$ and $(\neg H \wedge H \ \rightarrow$ some wfs). The first is trivial from the usual misspelt noun re *bugga*, and the second is even more trivial from $(\mathsf{P})_3$ and (TAUT).

Ex. 6. $\vdash^w$ $(\neg H \ \rightarrow \ < \nabla XC > \neg H)$*, if* $C = ite(H)(D)(bugga)$ *for some quantifier-free* H *and some command* D*.*

By $(\mathsf{AX})_{\nabla}$, we can change $< \nabla XC >$ to $< C_{\nabla XC} >$. But now **5** applies, since $C_{\nabla XC} = ite(H)(E)(bugga)$ for $E = D_{\nabla XC}$.

Ex. 7. *If* $C = ite(0 < y)(\text{`}y \dot{\leftarrow} y - 1\text{'}; callX)(bugga)$*, then*

$$\vdash^w \ (y \approx 0 \ \rightarrow \ < \nabla XC > y \approx 0) \ .$$

Use **6** and a small argument which notices that $(y \approx 0 \ \leftrightarrow \ \neg 0 < y)$ is tt **N**.

Ex. 8. *Given any wfs* $\mathcal{B}$*, if there is a wfs* $\mathcal{A}$ *such that both* $\mathcal{A} \rightarrow \mathcal{B}$ *and* $\neg\mathcal{A} \rightarrow \mathcal{B}$ *can be derived, then so can* $\mathcal{B}$ *itself.*

This is just a propositional argument, pretty much immediate from the fact that $(\mathcal{A} \rightarrow \mathcal{B}) \wedge (\neg\mathcal{A} \rightarrow \mathcal{B}) \ \rightarrow \ \mathcal{B}$ is a tautology.

Ex. 9. $\vdash (< \nabla XC > y \approx 0)$*, if* $C = ite(0 < y)(\text{`}y \dot{\leftarrow} y - 1\text{'}; callX)(bugga)$*.*

This is the standard result in the total correctness case which we have displayed before **4** was done above. It follows from **7** and **8**, once we give a derivation witnessing $\vdash \ (\neg y \approx 0 \ \rightarrow \ < \nabla XC > y \approx 0)$, for which we proceed now to argue.

Somewhere between here and the penultimate paragraph of this proof of **9**, there is a single, rather serious, error. We'll fix it easily in the final paragraph, but the reader may be interested to try to catch us up, where the error happens.

Since it is $(\mathsf{RCS})_{<\,>}$ which will be used, we shall discover a $1^{\underline{\text{st}}}$order formula $F = `F(w,y)'$ such that

(a) $\neg y \approx 0 \ \rightarrow \ \exists w F$ is $\mathsf{tt}\mathbf{N}$;

(b) $\neg |F|^{w \mapsto 0}$ is $\mathsf{tt}\mathbf{N}$; and

(c) $\{F \rightarrow < \nabla XC > y \approx 0\} \vdash^w |F|^{w \mapsto w+1} \rightarrow < C_{\nabla XC} > y \approx 0$.

The point is that (b), plus (c) with an application of $(\mathsf{AX})_{\nabla}$, will verify the two antecedents in an instance of $(\mathsf{RCS})_{<\,>}$ with Ω empty which immediately derives the wfs to the left of $\vdash^w$ in (c). And so (a) gives us, after applying (PRE) and (HYSY), what we really want, namely that

$$\neg y \approx 0 \ \rightarrow \ \ < \nabla XC > y \approx 0$$

is $\vdash$-derivable. (Notice how we are just 'dejavuing' from earlier stuff!)

So we take $F \ := \ 0 < w \wedge w \approx y$. This makes (a) and (b) quite trivial to check.

As for (c), once again $C_{\nabla XC}$ is an *ite*-formula, so it suffices to check

$$\vdash^w \ 0 < w+1 \wedge w+1 \approx y \wedge \neg 0 < y \ \rightarrow \ \ < bugga > y \approx 0 \qquad (1)$$

and

$$\{0 < w \wedge w \approx y \ \rightarrow \ \ < \nabla XC > y \approx 0\} \ \vdash^w$$

$$0 < w+1 \wedge w+1 \approx y \wedge 0 < y \ \rightarrow \ \ < `y \dot{-} y - 1' >< \nabla XC > y \approx 0 \quad (2)$$

As usual, (1) is almost immediate from (ORACLE).

As for (2), by **(3)** we can derive

$$0 < w+1 \wedge w+1 \approx y \wedge 0 < y \ \rightarrow \ \ < `y \dot{-} y - 1' > (0 < w+1 \wedge w+1 \approx y+1) \ .$$

Having noted that $0 < w+1 \wedge w+1 \approx y+1 \ \rightarrow \ (0 < w \wedge w \approx y) \vee y \approx 0$ is true in $\mathbf{N}$, we are left only to $\vdash^w$-derive

$$< `y \dot{-} y - 1' > ((0 < w \wedge w \approx y) \vee y \approx 0) \ \rightarrow \ \ < `y \dot{-} y - 1' >< \nabla XC > y \approx 0 \, ,$$

Now we can $\vdash^w$-derive both

$$y \approx 0 \ \rightarrow \ \ < \nabla XC > y \approx 0$$

by **(7)**, and

$$0 < w \wedge w \approx y \ \rightarrow \ < \nabla XC > y \approx 0 \ ,$$

which is just the premiss, so propositionally, we can $\vdash^w$-derive

$$(0 < w \wedge w \approx y) \vee y \approx 0 \ \rightarrow \ < \nabla XC > y \approx 0 \ ,$$

Having applied $(\mathsf{UNAR})_{< >}$ to the latter, we get the required result. (Of course there are a few (HYSY)'s happening above.)

The error in this 'proof' of **9** is at the point where we say "... by **(3)**, we can derive ...". Let E be the $1^{\underline{\text{st}}}$order formula immediately following. Then "$\vdash^w E$" is what is needed, whereas all we have from **(3)** is "$\vdash E$". To fix the proof (and put some emphasis on this important matter) we take not the empty Ω case, but rather the case of $(\mathsf{RCS})_{< >}$ where $\Omega = \{E\}$. Then at the point mentioned just above, we use E as a premiss. But earlier in the proof, we must have verified "$\vdash E$", since $\vdash \mathcal{C}$ for all $\mathcal{C} \in \Omega$ is an extra antecedent in the statement of the rule $(\mathsf{RCS})_{< >}$. And of course, **(3)** has done that job for us, so **9** now does have an acceptable proof. We have placed this discussion here at the end (with the earlier mysterious warning to cover our ass!) in order to emphasize the following: *When dealing with* $< \nabla XD >$ *or* $[\nabla XD]$ *where* D *itself has a subcommand of the form* ∇XC*, it is usually necessary to invoke an instance or instances of the recursion rule with* **non-empty** Ω*.*

The next two examples give the partial and total correctness of a somewhat 'notorious' command. They hopefully show that there is no excessive subtlety when we use dynamic logic and, in particular, the present proof system. So perhaps the "subtle problems" referred to in the following quote really do have more to do with what I referred to earlier as a "straightjacket". In **[Nip]** p. 3, we find :

"Consider the following parameterless procedure which calls itself recursively:

$$\mathsf{proc} \ = \ \mathsf{if\ i = 0\ then\ skip\ else\ i := i - 1; CALL; i := i + 1}$$

(*sic—lack of punctuation, but the "else" clause does include the remainder of the line. And note that Nipkow's and other CSers' ":=" is our "*$\div$*"*) A classic example of the subtle problems associated with reasoning about procedures is the proof that i is invariant: $\{i = N\}CALL\{i = N\}$. This is done by induction: we assume $\{i = N\}CALL\{i = N\}$ and have to prove $\{i = N\}body\{i = N\}$, where *body* is the body of the procedure. The case $i = 0$ is trivial. Otherwise we have to show $\{i = N\}i := i - 1; CALL; i := i + 1\{i = N\}$,

which can be reduced to $\{i = N-1\}CALL\{i = N-1\}$. But how can we deduce $\{i = N-1\}CALL\{i = N-1\}$ from $\{i = N\}CALL\{i = N\}$? Clearly we have to instantiate N in the induction hypothesis—after all N is arbitrary as it does not occur in the program. The problems with procedures are largely due to unsound or incomplete adaptation rules."

In our language, we are dealing with ∇XC, where

$$C \;:=\; ite(y \approx 0)(bugga)(\text{`}y \leftarrow\!\div y - 1\text{'} \;;\; callX \;;\; y \leftarrow\!\div y + 1) \;.$$

We wish to obtain derivations for (with distinct variables y and z)

$$(\mathbf{10}) \quad y \approx z \to [\nabla XC]y \approx z \quad \text{and} \quad (\mathbf{11}) \quad y \approx z \to\!< \nabla XC > y \approx z \;.$$

The above quote refers only to the partial correctness one, **(10)**.

Firstly, a few comments, and a minor change to C so that it conforms with what I'd scribbled before I first saw that quote. This will give us a few more easy generalities to ponder. The new C will be

$$C \;:=\; ite(0 < y)(\text{`}y \leftarrow\!\div y - 1\text{'} \;;\; callX \;;\; y \leftarrow\!\div y + 1)(bugga) \;.$$

Intuitively, this change clearly makes no difference. Technically, we should define $C \;\mathsf{eq}\; C' \iff m(C) = m(C')$, prove that C and C' which are equivalent in this sense satisfy $\mathcal{A} \to\!< C > \mathcal{B}\; \mathsf{tt}@s \Leftrightarrow \mathcal{A} \to\!< C' > \mathcal{B}\; \mathsf{tt}@s$, and so, in particular, by completeness, either both or neither of those wfs's is $\vdash$-derivable. And then an elementary check shows that all three of the following are equivalent, given that $H \leftrightarrow H'$ is $\mathsf{tt}\mathbf{N}$:

$$ite(H')(C)(D) \;\;\mathsf{eq}\;\; ite(H)(C)(D) \;\;\mathsf{eq}\;\; ite(\neg H)(D)(C) \;.$$

Further remarks say something you have probably been thinking: why keep bothering with partial correctness when you're intending to prove total correctness anyway? So we'll do **11** first, that derivation being reasonably easy to produce. Note however that the correctness of the technical statement

$$\vdash \mathcal{A} \to\!< C > \mathcal{B} \implies \vdash \mathcal{A} \to [C]\mathcal{B}$$

really depends on the command language being deterministic. It follows indirectly from completeness, since

$$\vdash ((\mathcal{A} \to\!< C > \mathcal{B}) \;\to\; (\mathcal{A} \to [C]\mathcal{B}))$$

most hold because of the truth in **N** of that wfs. We may come back to this topic after completing these two examples or at least before the sun is a red giant.

Ex. 11. Producing a derivation to witness $y \approx z \rightarrow < \nabla XC > y \approx z$ (for the latter choice of C above) is done as was **9**, mostly by verifying the antecedents in a suitable instance of $(\mathsf{RCS})_{< >}$. Take $F := y < w \wedge y \approx z$. Then we have that (a) and (b) below clearly hold, and (c) will be verified presently.
(a) $y \approx z \ \rightarrow \ \exists w F$ is $\mathbf{ttN}$;
(b) $\neg |F|^{w \rightarrow 0}$ is $\mathbf{ttN}$; and
(c) $\Omega \cup \{F \rightarrow < \nabla XC > y \approx z\} \vdash^w |F|^{w \rightarrow w+1} \rightarrow < C_{\nabla XC} > y \approx z$, where Ω is the singleton set

$$\{\ y < w+1 \wedge y \approx z \wedge 0 < y \ \rightarrow \ < \text{`}y \dot{-} y - 1\text{'} > (y+1 < w+1 \wedge y+1 \approx z)\ \}\ ;$$

and we do have the derivability of that last formula almost directly from **(3)**, so that antecedent is dealt with.
The point is that (b), plus (c) with an application of $(\mathsf{AX})_{\nabla}$, will verify the other two antecedents in an instance of $(\mathsf{RCS})_{< >}$ which directly derives the wfs immediately to the left of $\vdash^w$ in (c). And so (a) gives us, after applying (PRE) and (HYSY), what we really want, namely that

$$y \approx z \ \rightarrow \ < \nabla XC > y \approx z$$

is $\vdash$-derivable. (Again we are just 'dejavuing' from earlier stuff!)

As for (c), once again $C_{\nabla XC}$ is an *ite*-formula, so it suffices to check both

$$\vdash^w \ y < w+1 \wedge y \approx z \wedge \neg 0 < y \ \rightarrow \ < bugga > y \approx 0 \qquad (1)$$

(which is again obvious), and

$$\Omega \cup \{y < w \wedge y \approx z \ \rightarrow \ < \nabla XC > y \approx z\} \ \vdash^w$$

$$y < w+1 \wedge y \approx z \wedge 0 < y \ \rightarrow \ < \text{`}y \dot{-} y - 1\text{'} >< \nabla XC >< y \dot{-} y+1 > y \approx z \quad (2)$$

For this, firstly we can $\vdash^w$-derive

$$y < w+1 \wedge y \approx z \wedge 0 < y \ \rightarrow \ < \text{`}y \dot{-} y - 1\text{'} > (y+1 < w+1 \wedge y+1 \approx z)\ ,$$

since it is just the extra premiss (in Ω). Furthermore,

$$\{y < w \wedge y \approx z \ \rightarrow \ < \nabla XC > y \approx z\} \ \vdash^w$$

$$y + 1 < w + 1 \wedge y + 1 \approx z \ \rightarrow \ < \nabla XC > y + 1 \approx z$$

[$(*)$ Below we'll place $<$ '$y \dot{\leftarrow} y - 1$' $>$ in front of both sides of $\rightarrow$ in this last line.] This is derived by applying $(\mathsf{SUB})_1^{z \rightarrow u}$ to the main premiss, for some brand-new variable u; appending a "$\wedge u + 1 \approx z$" to both sides by $(\mathsf{AX})_{<\rightarrow>}$; then replacing on the far right the formula $y \approx u \wedge u + 1 \approx z$ by just $y + 1 \approx z$; then placing a "$\exists u$" in front of the left-half by (PRE); and finally writing down the final desired product, the wfs to the right of $\vdash^w$ in the display immediately above. The truth in $\mathbf{N}$ of both

$$y \approx u \wedge u+1 \approx z \rightarrow y+1 \approx z \text{ and } y+1 < w+1 \wedge y+1 \approx z \rightarrow \exists u(y < w \wedge y \approx u \wedge u+1 \approx z)$$

is used for the 3rd last and the last steps, respectively.
Finally we have, from $(\mathsf{AX})_{<\Leftrightarrow>}$,

$$\vdash^w \ y + 1 \approx z \ \rightarrow \ < y \dot{\leftarrow} y + 1 > y \approx z \ .$$

[$(*)$—We'll be placing $<$ '$y \dot{\leftarrow} y - 1$' $><$ $\nabla XC >$ in front of both sides of $\rightarrow$ in this.]

Applying $(\mathsf{UNAR})_{< >}$ [as in $(*)$] and then (HYSY) a few times to wfs's in three of the four displays immediately above (all but the penultimate one), we're done with (2) and so, with **(11)**, as required.

The tricky contortions with u a couple of lines up are similar to part of the work on **(3)**, and will be seen later in general in Section 4's proof of **23.3**. So maybe Nipkow is correct about subtlety! But here the subtlety's not in the choice of rules; it's in their application. So one can sympathize with the 'practical' program verifiers. In any case, those contortions are needed in slightly simpler form in the partial correctness case for this command just below, so we will write them out in detail to help clarify (or muddy) the matter. Whichever style between **10** and **11** is preferred, translate the other one into it by comparing to the one preferred.

Ex. 10. Producing a derivation to witness $y \approx z \rightarrow [\nabla XC]y \approx z$ for the latter choice of C as above, namely

$$C \ := \ ite(0 < y)(\text{'}y \dot{\leftarrow} y - 1\text{'} \ ; \ callX \ ; \ y \dot{\leftarrow} y + 1)(bugga) \ ,$$

proceeds by applying $\mathsf{(RCS)}_{[\ ]\rightarrow}$ with a singleton Ω whose only element is the wfs $(*)$ a few lines below. Then that formula is $\vdash$-derivable directly from **(2)**, checking the one antecedent of the rule. This leaves us to verify that

$$y \approx z \rightarrow [\nabla XC]y \approx z \ \vdash' \ y \approx z \rightarrow [C_{\nabla XC}]y \approx z \ .$$

Again because $C_{\nabla XC}$ is an *ite*-command, this reduces to checking

$$\vdash' \ y \approx z \wedge \neg 0 < y \ \rightarrow \ [bugga]y \approx z$$

(which is again obvious), and

$$\Omega \cup \{y \approx z \ \rightarrow \ [\nabla XC]y \approx z\} \ \vdash'$$

$$y \approx z \wedge 0 < y \ \rightarrow \ [\text{`}y \leftarrow\!\div y - 1\text{'}][\nabla XC][y \leftarrow\!\div y + 1]y \approx z \ .$$

For the latter, the formula in Ω above is

$$y \approx z \wedge 0 < y \ \rightarrow \ [\text{`}y \leftarrow\!\div y - 1\text{'}]y + 1 \approx z \qquad\qquad (*)$$

so we can $\vdash'$-derive it by fiat. An instance of $\mathsf{(AX)}_{[\leftarrow]}$ is

$$y + 1 \approx z \ \rightarrow \ [y \leftarrow\!\div y + 1]y \approx z \ .$$

It remains to 'fill in the middle'— this gets automatic after a while, doesn't it?—by spelling out those contortions with u to produce a derivation witnessing

$$\{y \approx z \ \rightarrow \ [\nabla XC]y \approx z\} \ \vdash' \ y + 1 \approx z \ \rightarrow \ [\nabla XC]y + 1 \approx z \ .$$

Removing, for clarity, three obvious lines where we give the multiple justifications below, here's the derivation.

$y \approx z \ \rightarrow \ [\nabla XC]y \approx z$ (premiss)

$y \approx u \ \rightarrow \ [\nabla XC]y \approx u$ $(\mathsf{SUB})^{z \to u}_{1}$

$y \approx u \wedge u+1 \approx z \ \rightarrow \ [\nabla XC](y \approx u \wedge u+1 \approx z)$ $(\mathsf{AX})_{[\rightarrow]}$ (MP)

$y \approx u \wedge u+1 \approx z \ \rightarrow \ [\nabla XC]y+1 \approx z$ (ORACLE) $(\mathsf{UNAR})_{[\ ]}$ (HYSY)

$\exists u(y \approx u \wedge u+1 \approx z) \ \rightarrow \ [\nabla XC]y+1 \approx z$ (PRE)

$y+1 \approx z \ \rightarrow \ \exists u(y \approx u \wedge u+1 \approx z)$ (ORACLE)

$y+1 \approx z \ \rightarrow \ [\nabla XC]y+1 \approx z$ (HYSY)

(This is perhaps what is meant by "instantiate N in the induction hypothesis".)

Another well-trodden example is a standard recursive program for computing the factorial function. With '*y as the function,* $x!$ *, of* x', that command in our language is $(y \leftarrow\!\div 1 \ ; \ \nabla XC)$, where

$$C \ := \ ite(0 < x)(\text{`}x \leftarrow\!\div x-1\text{'} \ ; \ callX \ ; \ x \leftarrow\!\div x+1 \ ; \ y \leftarrow\!\div y \times x)(bugga) \ .$$

Now usually at this point you'll see a Floyd-Hoare statement to be derived which amounts to, for partial correctness,

$$[(y \leftarrow\!\div 1 \ ; \ \nabla XC)]y \approx x! \ , \quad \text{and perhaps even} \quad \ < (y \leftarrow\!\div 1 \ ; \ \nabla XC) > y \approx x! \ ,$$

for total correctness. Beginners might well be baffled to see those occurrences of the symbol "!" pretending to live within the language of $1^{\underline{\text{st}}}$order number theory. But often in other write-ups, the syntax isn't all that well spelled out anyway. Here we shall rectify the situation by referring to a (possible) $1^{\underline{\text{st}}}$order formula $Fac \ = \ \text{`}Fac(x,y)\text{'}$ such that the following are both ttN :

(a) $x \approx 0 \wedge y \approx 1 \ \rightarrow \ Fac$; and

(b) $Fac \ \rightarrow \ ||Fac|^{y \to y \times x}|^{x \to x+1}$.

[Notice that the informal version of the formula in (b) is

$$y \approx x! \ \rightarrow \ y \times (x+1) \approx (x+1)! \ \ ,$$

i.e. just apply the successive substitutions in (b) blindly to $y \approx x!$ and you get $y \times (x+1) \approx (x+1)!$. So there ought to be a unique (up to truth equivalence—but in $\mathbf{N}$ or in every interpretation??) $1^{\underline{\text{st}}}$order formula Fac satisfying (a) and (b). And it is highly satisfactory to take any such formula

to be the formula which says "$y \approx x!$". The student of logic ought to be keen to study the existence and uniqueness of such formulas Fac. Here we'll pass over that question in silence.]

More formal versions of the earlier display are:

Ex. 12. *For any Fac satisfying* (a) *and* (b)*, we have*

$$\vdash \ [(y \leftarrow\!\div 1 \ ; \ \nabla XC)]Fac \ .$$

Ex. 13. *For any Fac satisfying* (a) *and* (b)*, we have*

$$\vdash \ \ < (y \leftarrow\!\div 1 \ ; \ \nabla XC) > Fac \ .$$

For **12**, a derivation of $1 \approx 1 \rightarrow [y \leftarrow\!\div 1][\nabla XC)]Fac$ (clearly sufficient) comes by first noticing that

$$1 \approx 1 \rightarrow [y \leftarrow\!\div 1]y \approx 1$$

is derivable immediately from the assignment axiom, and then showing

$$\Omega \cup \{y \approx 1 \rightarrow [\nabla XC]Fac\} \ \vdash' \ y \approx 1 \rightarrow [C_{\nabla XC}]Fac$$

in order to apply the recursion rule with a set Ω which is again a singleton containing only the wfs $(**)$ displayed several lines below. From **(2)** (or more easily), we can derive $(**)$, so one antecedent in the rule is okay. As for the other, namely the display just above, it reduces, in the usual fashion, to getting both

$$\vdash' \ y \approx 1 \wedge \neg 0 < x \ \rightarrow \ [bugga]Fac$$

(that's just (a) pumped into the oracle plus the innocuity of *bugga*), and

$$\Omega \cup \{y \approx 1 \rightarrow [\nabla XC]Fac\} \ \vdash'$$

$$y \approx 1 \wedge 0 < x \ \rightarrow \ [\text{`}x \leftarrow\!\div x-1\text{'}][\nabla XC][x \leftarrow\!\div x+1][y \leftarrow\!\div y \times x]Fac \ .$$

The latter comes quite mechanically via two applications of the assignment axiom and the extra premiss $(**)$, with the sprinkling of (UNAR)'s and (HYSY)'s which by now should be embedded into your brain-muscle memory. Those three wfs's are

$$y \approx 1 \wedge 0 < x \;\rightarrow\; [\text{`}x \leftarrow\!\div x - 1\text{'}]y \approx 1 \qquad (**)$$

$$||Fac|^{y \to y \times x}|^{x \to x+1} \;\rightarrow\; [x \leftarrow\!\div x + 1]|Fac|^{y \to y \times x}$$

and

$$|Fac|^{y \to y \times x} \;\rightarrow\; [y \leftarrow\!\div y \times x]Fac\ .$$

These are combined with (b) in the definition of Fac, and the main premiss,

$$y \approx 1 \;\rightarrow\; [\nabla XC]Fac\ ,$$

by means of (HYSY) and (UNAR) to easily finish the job. That is, the five $\rightarrow$ wfs's, with anywhere from zero to four [*command*] strings in front of them, are combined by means of (HYSY) to produce the wfs which we need to $\vdash^w$-derive.

The corresponding *total* correctness proof, **13**, will be left as an exercise.

Other sources are at pains to point out here that what they've just done isn't really adequate for what we wish to verify (and at question isn't the cavalier use of "!" as a symbol in logic—*not* meaning "unique"!). The point to be made is: How does one know that the value x itself didn't change during the computation? We want the value of y after the computation to be the factorial of the value of x *before* the computation started, *not* of the most recent value of x. The latter is a program variable, so formally seeing that it didn't change is more than just a quick application of $(\mathsf{AX})_{disj}$. Thus, just below are the partial and total correctness wfs's which really ought to be verified, where z is a third, different variable, called an *auxiliary variable* by those who need a name for it :

$$x \approx z \;\rightarrow [(y \leftarrow\!\div 1\ ;\ \nabla XC)](Fac \wedge x \approx z)$$

and

$$x \approx z \;\rightarrow\!< (y \leftarrow\!\div 1\ ;\ \nabla XC) > (Fac \wedge x \approx z)$$

It is fairly clear that verifying these will involve little more than combining what we just did above with essentially the previous lengthy example. So we won't pursue this any further.

3. Proofs, many deadly dull, re pre-completeness matters.

To begin, here are the elementary proofs of the displayed results claimed in Section 1. Refer back for the statements and definitions.

Prop.2. Just observe that, of the right-hand sides in the definition, the top two are manifestly commands, and so are the next three inductively, and the last one obviously.

Prop.3. Same as the previous proof, using the relevant inductive definition.

Prop.4. Inductively, the first statement is immediate from **3**.
For the second, we prove further down by induction on C that $[C_D]_E \;=\; C_{D_E}$ for all C, D and E, where the square brackets are just for clarity, not part of the syntax—maybe I should have used vertical bars already in the definition here as well!
Then, inductively on a we get $[C^{<0>}]_{C^{<b>}} \;=\; [callX]_{C^{<b>}} \;=\; C^{<b>}$ and

$$[C^{<a+1>}]_{C^{<b>}} \;=\; [C_{C^{<a>}}]_{C^{<b>}} \;=\; C_{[C^{<a>}]_{C^{<b>}}} \;=\; C_{C^{<a+b>}} \;=\; C^{<a+b+1>}\ ,$$

as required.
The earlier induction is straightforward:

$$[[x \dot{\leftarrow} t]_D]_E \;=\; [x \dot{\leftarrow} t]_E \;=\; x \dot{\leftarrow} t \;=\; [x \dot{\leftarrow} t]_{D_E}$$

$$[[callX]_D]_E \;=\; D_E \;=\; [callX]_{D_E}$$

$$[(C;C')_D]_E \;=\; (C_D;C'_D)_E \;=\; ([C_D]_E\ ;\ [C'_D]_E) \;=\; (C_{D_E};C'_{D_E}) \;=\; (C;C')_{D_E}$$

$$[ite(H)(C)(C')_D]_E \;=\; ite(H)(C_D)(C'_D)_E \;=\; ite(H)([C_D]_E)([C'_D]_E) \;=$$

$$ite(H)(C_{D_E})(C'_{D_E}) \;=\; ite(H)(C)(C')_{D_E}$$

$$[[\nabla XC]_D]_E \;=\; [\nabla XC]_E \;=\; \nabla XC \;=\; [\nabla XC]_{D_E}$$

Prop.5. That nab(∇XC)=nab(C)+1 is clear from the definition of nab, since ∇XC clearly cannot be a subcommand of C. As for $C^{<n>}$, let NAB(C) be the actual *set* of all D for which ∇XD occurs as a subcommand of C. Then $\mathrm{NAB}(C_E) = \mathrm{NAB}(C) \cup \mathrm{NAB}(E)$, from which it follows by a very quick induction on n that $\mathrm{NAB}(C^{<n>}) = \mathrm{NAB}(C)$, and we're done.

Prop.6. We shall prove that $m(D) \subset m(D_E)$ for all D and E, by induction on D. Then, using **4** and taking $D = C^{<n>}$ and $E = C$, we get the first statement. As for the induction, the initial steps are immediate.
When $D = (D'; D'')$ we have

$$m(D) \;=\; \{(s, s'') \mid \exists s' \;:\; (s, s') \in m(D') \text{ and } (s', s'') \in m(D'')\}$$

$$\subset\; \{(s, s'') \mid \exists s' \;:\; (s, s') \in m(D'_E) \text{ and } (s', s'') \in m(D''_E)\} \;=\; m(D'_E; D''_E) \;=\; m(D_E)\,.$$

When $D = ite(H)(D')(D'')$ we have

$$m(D) \;=\; \{(s, s') \mid [(s, s') \in m(D') \text{ and } H\ \mathsf{tt}@s] \text{ or } [(s, s') \in m(D'') \text{ and } H\ \mathsf{ff}@s]\}$$

$$\subset\; \{(s, s') \mid [(s, s') \in m(D'_E) \text{ and } H\ \mathsf{tt}@s] \text{ or } [(s, s') \in m(D''_E) \text{ and } H\ \mathsf{ff}@s]\}$$
$$=\; m(ite(H)(D'_E)(D''_E)) \;=\; m(D_E)\;.$$

When $D = \nabla X B$ we have $D_E = D$, so the result is trivial.

As for the second statement, this will be immediate by induction on n from the more general claim that $m(D)$ and $m(E)$ both being graphs of functions implies the same is true for $m(D_E)$. Once again, we proceed by induction on D, and the initial cases of atomic D are immediate. So are the first two inductive cases, from the easily checked facts that, if $m(D')$ and $m(D'')$ are both graphs of functions, then so are $m(D'; D'')$ and $m(ite(H)(D')(D''))$. Again, when $D = \nabla X B$ we have $D_E = D$, so the result is trivial.

Theorem 1. The 'function-graphiness' of $m(D)$ for atomic D is completely obvious, and the first two inductive cases are simply the penultimate sentence in the previous proof. When $D = \nabla X C$, we use the definition of $m(D)$ as a union; in this case, of graphs of functions and an increasing union, by **6**; and finally the elementary set-theoretic fact that an increasing union of graphs of functions is also one.

Prop.7. $\quad \neg < C > \neg \mathcal{A}\ \mathsf{tt}@s \iff\ < C > \neg \mathcal{A}\ \mathsf{ff}@s$

$$\iff \text{ there is no } s' \text{ with } (s, s') \in m(C) \text{ and } \neg \mathcal{A}\ \mathsf{tt}@s'$$
$$\iff \text{ there is no } s' \text{ with } (s, s') \in m(C) \text{ and } \mathcal{A}\ \mathsf{ff}@s'$$
$$\iff \text{ for all } s', \text{ if } (s, s') \in m(C), \text{ then } \mathcal{A}\ \mathsf{tt}@s'\;.$$

Prop.8.

$$\mathcal{A} \rightarrow < C > \mathcal{B}\ \mathsf{ff}@s \iff \mathcal{A}\ \mathsf{tt}@s \text{ and } < C > \mathcal{B}\ \mathsf{ff}@s$$

$$\iff \mathcal{A}\ \mathsf{tt}@s \text{ but there is no } s' \text{ with } (s, s') \in m(C) \text{ and } \mathcal{B}\ \mathsf{tt}@s' \ .$$

Thus the negation of "$\mathcal{A} \rightarrow < C > \mathcal{B}\ \mathsf{ttN}$" is equivalent to the existence of an s for which the 1st, and so the 3rd, statement just above hold. But the latter is clearly the negation of the condition in the proposition.

Prop.9. Using **7**,

$$\mathcal{A} \rightarrow [C]\mathcal{B}\ \mathsf{ff}@s \iff \mathcal{A}\ \mathsf{tt}@s \text{ and } [C]\mathcal{B}\ \mathsf{ff}@s$$

$$\iff \mathcal{A}\ \mathsf{tt}@s \text{ and there is an } s' \text{ with } (s, s') \in m(C) \text{ and } \mathcal{B}\ \mathsf{ff}@s' \ .$$

Thus the negation of "$\mathcal{A} \rightarrow [C]\mathcal{B}\ \mathsf{ttN}$" is equivalent to the existence of an s for which the 1st, and so the 3rd, statement just above hold. But the latter is clearly the negation of the condition in the proposition.

Prop.10. When C is $y \doteqdot t$, so that $y \neq x$, we have $s' = s_{y \mapsto t^s}$, so s and s' do agree on x.

When C is $callX$, there are no $(s, s') \in m(C)$, so there is nothing to prove.

In the first inductive case, when C is $(C'; C'')$ and $(s, s'') \in m(C)$, we have s' with $(s, s') \in m(C')$ and $(s', s'') \in m(C'')$. Inductively, s and s' agree on x, as do s' and s'', hence so do s and s'', as required.

When C is $ite(H)(C')(C'')$, if $H\ \mathsf{tt}@s$, then $(s, s') \in m(C')$, so s and s' agree on x by induction; and if $H\ \mathsf{ff}@s$, then $(s, s') \in m(C'')$, so again s and s' agree on x, as required.

When C is $\nabla X D$, we evidently need to just show, by induction on n, that if $(s, s') \in m(D^{<n>})$, then s and s' agree on x, where the inductive assumption allows us to know this for $n = 1$. For this, fix x and let $(*)_C$ be the property

$$(*)_C \qquad (s, s') \in m(C) \implies s(x) = s'(x) \ .$$

It clearly suffices to prove, by induction on D, that $[(*)_D$ and $(*)_E]$ implies $(*)_{D_E}$, and then take $E = D^{<n>}$. In the cases when D is an assignment command or $\nabla X B$, we have $D_E = D$, so there is nothing to prove. When $D = callX$ we have $D_E = E$, so again there is nothing to prove. The

inductive cases of 'sequencing' and 'if-then-elsing' are mechanical, exactly as in the earlier induction in this proof.

Theorem 11. As remarked after the theorem's statement, it suffices to show that applying $|\ |^{\nabla XG\to E}$ to the antecedents and the consequent of each of the rules defining $\vdash'$ will again produce one of those rules (of the same type, as it happens). To be convincing (keeping in mind the number of incorrect arguments in the published literature of this subject!), we really have to be more formal about the definition of $|\mathcal{A}|^{\nabla XG\to E}$, writing it down inductively on wfs's, as follows:

$$|F|^{\nabla XG\to E} := F \text{ for atomics } \ ; \ \ |\neg\mathcal{A}|^{\nabla XG\to E} := \neg|\mathcal{A}|^{\nabla XG\to E} \ ;$$

$$|\forall x\mathcal{A}|^{\nabla XG\to E} := \forall x|\mathcal{A}|^{\nabla XG\to E} \ ; \ \ |\mathcal{A}\wedge\mathcal{B}|^{\nabla XG\to E} := |\mathcal{A}|^{\nabla XG\to E}\wedge|\mathcal{B}|^{\nabla XG\to E} \ ;$$
$$|<D>\mathcal{A}|^{\nabla XG\to E} :=<|D|^{\nabla XG\to E}>|\mathcal{A}|^{\nabla XG\to E} \ .$$

Now we work through the rules in the order given, combining the above inductive definition with the much earlier one defining that substitution operation on commands.

The operation on wfs's fixes $1^{\underline{\text{st}}}$order formulas, so (ORACLE) is fine.

The operation on wfs's commutes with propositional connectives, so (MP) and (TAUT) are fine. (See the 'argument' for $(\mathsf{AX})_{<;>}$ just below, if more convincing is needed).

The operation on commands fixes assignment commands, and on wfs's fixes $1^{\underline{\text{st}}}$order formulas, so $(\mathsf{AX})_{<\leftrightarrow>}$ is fine. (Again, see the next paragraph.).

Applying the operation to the stated version of $(\mathsf{AX})_{<;>}$ and following those inductive definitions in an entirely mechanical manner produces

$$<(|C|^{\nabla XG\to E};|D|^{\nabla XG\to E})>|\mathcal{A}|^{\nabla XG\to E} \ \leftrightarrow \ <|C|^{\nabla XG\to E}><|D|^{\nabla XG\to E}>|\mathcal{A}|^{\nabla XG\to E} \ ,$$

which is just another instance of $(\mathsf{AX})_{<;>}$, as required.

The exact same type of argument applies to all of $(\mathsf{AX})_{<ite>}$, $(\mathsf{AX})_{[ite]}$, $(\mathsf{UNAR})_{\forall}$ and $(\mathsf{UNAR})_{<\ >}$. For the second of those, one also needs to first replace all [ ]'s by $\neg <\ \ > \neg$'s directly, or else do that to prove a formula similar to the definition above of the operation on $<D>\mathcal{A}$.

In fact, all the remaining six rules also go the same way. This all depends, as mentioned earlier, on the fact that no ∇ appears explicitly in any of these rules. But three of those last six have restrictions placed on variables

occurring, so we must check that those restrictions still apply to the versions of the rules after applying the operation, using our condition that E has no variables not already occurring in C.

As for $(\mathsf{AX})_{disj}$, the transformed version is

$$\emptyset \ / \ (|\mathcal{A}|^{\nabla X \subset \to E} \to [|D|^{\nabla X \subset \to E}]|\mathcal{A}|^{\nabla X \subset \to E})$$

(Recall that "$\to$" is an abbreviation using "$\neg$" and "$\wedge$", so the inductive definition at the start of the proof is easily applicable to get this.)
This clearly is another instance of the same axiom, but, more to the point, the variables in $|D|^{\nabla X \subset \to E}$ are a subset of the variables in D itself. And those in $|\mathcal{A}|^{\nabla X \subset \to E}$ are a subset of the variables in $\mathcal{A}$. And so these two subsets are disjoint from each other, since their containing sets are given to be disjoint.

As for (PRE), the transformed version is

$$(|\mathcal{A}|^{\nabla X \subset \to E} \to |\mathcal{B}|^{\nabla X \subset \to E}) \ / \ ((\exists z |\mathcal{A}|^{\nabla X \subset \to E}) \to |\mathcal{B}|^{\nabla X \subset \to E})$$

(Recall similarly that "$\exists$" is an abbreviation using "$\neg$" and "$\forall$".)
This clearly is another instance of the same axiom, but, to get to the point, the variables in $|\mathcal{B}|^{\nabla X \subset \to E}$ are a subset of the variables in $\mathcal{B}$. And so the variable z is not in the former, since it is given to be not in the latter.

Finally, as for $(\mathsf{SUB})_2$, let us temporarily write F as though it were a more general wfs $\mathcal{A}$. Then the transformed version is

$$|\mathcal{A}|^{\nabla X \subset \to E} \to |\mathcal{B}|^{\nabla X \subset \to E} \ / \ (||\mathcal{A}|^{\nabla X \subset \to E}|^{z \to x} \to |\mathcal{B}|^{\nabla X \subset \to E})$$

This clearly is another instance of the same axiom, but, as to the real issue, the variables in $|\mathcal{B}|^{\nabla X \subset \to E}$ are a subset of the variables in $\mathcal{B}$. And so z is not in the subset, since it is given to be not in $\mathcal{B}$. Note that the correctness of the last display depends on the two types of substitution commuting with each other; that is,

$$||\mathcal{A}|^{z \to x}|^{\nabla X \subset \to E} \ = \ ||\mathcal{A}|^{\nabla X \subset \to E}|^{z \to x}$$

That seems very obvious; and a rigorous proof by induction on wfs's ought to be exceedingly mechanical. But then the painstaking, but definitely unbrilliant, mathematician, when wrestling with the "intricate details" I threatened earlier, discovers that it may very well not be true until he recalls the proviso that z cannot be a program variable in any command within $\mathcal{A}$! So it

is rather fortunate that we need the rule only when $\mathcal{A}$ has that property, and then the 'commutation of substitutions rule' just above has indeed a mechanical inductive proof.

With the major exception of **Theorem 14**, and the following proof of **16**, the brief proofs of **12** to **17** are all given right after their statements.

As for **16**, prove the contrapositive: *if* $\mathcal{A} \to [\nabla XC]\mathcal{B}$ ff@s, *then* $[\nabla XC]\mathcal{B}$ ff@s *and* $\mathcal{A}$ tt@s. So we have some $(s, s') \in m(\nabla XC)$ with $\mathcal{B}$ ff@s'. From the definition of $m(\nabla XC)$, we can find a natural number n for which we have $(s, s') \in m(C^{<n>})$. But these conditions show directly from definitions that $\mathcal{A} \to [C^{<n>}]\mathcal{B}$ ff@s, as required.

Theorem 14. As remarked in Section 1, we must go through the rules for $\vdash'$ and check that they are all valid.

The first seven rules are very standard, the latter four of them essentially classical Floyd-Hoare rules, so those seven will be left for now. The point is simply that the proofs of their validity are independent of the particular command language used as a basis for the dynamic logic language.

The (UNAR)-rules are straight from Harel, but in the pedestrian style of this write-up, we'll write down the proofs of validity, since his rules are derived from a more general non-deterministic system oriented towards iteration, not recursion. But the rather trivial proofs are again independent of the command language. In these and most of the 'non-axiom' ones below, we proceed by assuming the consequent is false at some particular state s, and find some state $\bar{s}$ and one of the antecedents which is false there.

If $\forall x\ \mathcal{A} \to \forall x\ \mathcal{B}$ is ff@s, then $\forall x\ \mathcal{B}$ is ff@s, and $\forall x\ \mathcal{A}$ is tt@s. Thus $\mathcal{A}$ is true at all states agreeing with s except possibly at x, and $\mathcal{B}$ is false at at least one of them, which can be our $\bar{s}$ in this case of $(\mathsf{UNAR})_{\forall}$.

As for $(\mathsf{UNAR})_{< >}$, if $< C > \mathcal{A} \to < C > \mathcal{B}$ is ff@s, then $< C > \mathcal{B}$ is ff@s, and $< C > \mathcal{A}$ is tt@s. By the second of these, there is a $(s, s') \in m(C)$, and $\mathcal{A}$ is true at s'; but $\mathcal{B}$ is false there by the first. So s' can be our $\bar{s}$ in this case.

Now for $(\mathsf{AX})_{call}$, assume $\neg < callX > \mathcal{A}$ is ff@s, so $< callX > \mathcal{A}$ is tt@s, which, among other things, implies the existence of $(s, s') \in m(callX)$, contradicting that set's utter emptiness and pain and sorrow.

For $(\mathsf{AX})_{disj}$, assume $\mathcal{A} \to [D]\mathcal{A}$ is $\mathsf{ff}@s$, so $[D]\mathcal{A}$ is $\mathsf{ff}@s$, and $\mathcal{A}$ is $\mathsf{tt}@s$. By the first of these, there is a $(s, s') \in m(D)$, and $\mathcal{A}$ is $\mathsf{ff}@s'$. By **10**, the states s and s' agree on all variables *not* in D, and so on all variables *in* $\mathcal{A}$, by disjointness of the sets of variables occurring in those two strings. And so the truth values of $\mathcal{A}$ at the two states must agree, giving the required contradiction.

For (AND), assume $\mathcal{A} \wedge \mathcal{A}' \to [D](\mathcal{B} \wedge \mathcal{B}')$ is $\mathsf{ff}@s$, so $[D](\mathcal{B} \wedge \mathcal{B}')$ is $\mathsf{ff}@s$, and both $\mathcal{A}$ and $\mathcal{A}'$ are $\mathsf{tt}@s$. By the first of these, there is an $(s, s') \in m(D)$, and $\mathcal{B} \wedge \mathcal{B}'$ is false at s'. If $\mathcal{B}$ is false at s', we see that the antecedent $\mathcal{A} \to [D]\mathcal{B}$ is $\mathsf{ff}@s$. On the other hand, if $\mathcal{B}'$ is false at s', we see that the antecedent $\mathcal{A}' \to [D]\mathcal{B}'$ is $\mathsf{ff}@s$. So $\bar{s} = s$ works here.

The rule (PRE) is a straightforward generalization of a well-known rule in 'static' logic, and the proof here of its validity is 'same-old', but here it is. Assume $(\exists w\mathcal{A}) \to \mathcal{B}$ is $\mathsf{ff}@s$, so $\mathcal{B}$ is $\mathsf{ff}@s$, and $\exists w\mathcal{A}$ is $\mathsf{tt}@s$. By the latter, $\mathcal{A}$ is $\mathsf{tt}@\bar{s}$ for some $\bar{s}$ agreeing with s except possibly at w. But that variable does not occur in $\mathcal{B}$, so that wfs's truth values agree at the two states. Thus $\bar{s}$ is a state where $\mathcal{A} \to \mathcal{B}$ is false, as required.

The rule $(\mathsf{SUB})_1$ is also the expected generalization of a well-known rule in classical logic, and the proof here of its validity is again the same as it was in ancient Frege-land, but here it is. Assume $|\mathcal{A}|^{u \to t}$ is $\mathsf{ff}@s$. Then $\mathcal{A}$ is $\mathsf{ff}@\bar{s}$, as required, where $\bar{s} = s_{u \mapsto s(t)}$. (As we keep emphasizing, u will not be a program variable in any command within $\mathcal{A}$; that makes the usual connection between truth at states before and after substitution work perfectly well for wfs's $\mathcal{A}$, with the same proof as for $1^{\underline{\text{st}}}$order formulas.)

The rule $(\mathsf{SUB})_2$ is another generalization of a well-known rule, and the proof here of its validity is the same as it was in 1880: We'll again work as though F is a general wfs $\mathcal{A}$, but we are definitely assuming that the variable x does not occur as a program variable in any command within $\mathcal{A}$, as indicated in the discussion on substitution just before the earliest presentation of the proof rules. Assume $|\mathcal{A}|^{z \to x} \to \mathcal{B}$ is $\mathsf{ff}@s$. Then $\mathcal{B}$ is $\mathsf{ff}@s$ and $|\mathcal{A}|^{z \to x}$ is $\mathsf{tt}@s$. By the latter, $\mathcal{A}$ is $\mathsf{tt}@\bar{s}$ where $\bar{s} = s_{z \mapsto s(x)}$. Since $\bar{s}$ agrees with s except possibly at z, which doesn't occur in $\mathcal{B}$, the latter has the same truth values at those two states, namely ff. Thus the antecedent is false at $\bar{s}$, as required.

Validity of $(\mathsf{AX})_{<\rightarrow>}$. Assume for a contradiction that

$$(\mathcal{A} \rightarrow < C > \mathcal{B}) \quad \rightarrow \quad (\mathcal{A} \wedge \mathcal{D} \rightarrow < C > (\mathcal{B} \wedge \mathcal{D}))$$

is $\mathsf{ff}@s$, so $\mathcal{A} \wedge \mathcal{D} \rightarrow < C > (\mathcal{B} \wedge \mathcal{D})$ is $\mathsf{ff}@s$, and $\mathcal{A} \rightarrow < C > \mathcal{B}$ is $\mathsf{tt}@s$. By the former, $< C > (\mathcal{B} \wedge \mathcal{D})$ is $\mathsf{ff}@s$, and both $\mathcal{A}$ and $\mathcal{D}$ are $\mathsf{tt}@s$. By the latter, and the immediately previous, $< C > \mathcal{B}$ is $\mathsf{tt}@s$. Thus we have an $(s, s') \in m(C)$ such that $\mathcal{B}$ is $\mathsf{tt}@s'$. By **10**, the states s and s' agree on all variables not in C, and so, on all variables in $\mathcal{D}$. Thus the truth values of $\mathcal{D}$ on those two states are the same, namely tt. We have now directly contradicted the earlier statement that $< C > (\mathcal{B} \wedge \mathcal{D})$ is $\mathsf{ff}@s$, as required.

Derivation of some extra rules. In the next two sections, the following rules are used in derivations, and occasionally (since they will be known to be valid now, once they've been derived) to shorten semantic arguments. We shall leave the provision of $\vdash'$-justifications in these derivations as exercises.

$(\mathsf{AX})_{[call]}$ This amounts to $\emptyset \; / \; \neg < callX > \neg\mathcal{A}$, which is simply another instance of $(\mathsf{AX})_{<call>}$

$(\mathsf{UNAR})_{\exists}$ Here's a derivation:

$\mathcal{A} \rightarrow \mathcal{B}$

$(\mathcal{A} \rightarrow \mathcal{B}) \; \rightarrow \; (\neg\mathcal{B} \rightarrow \neg\mathcal{A})$

$\neg\mathcal{B} \rightarrow \neg\mathcal{A}$

$\forall x \neg\mathcal{B} \rightarrow \forall x \neg\mathcal{A}$

$(\forall x \neg\mathcal{B} \rightarrow \forall x \neg\mathcal{A}) \; \rightarrow (\neg\forall x \neg\mathcal{A} \rightarrow \neg\forall x \neg\mathcal{B})$

$\exists x\mathcal{A} \rightarrow \exists x\mathcal{B}$

$(\mathsf{UNAR})_{[\,]}$ Here's a derivation:

$\mathcal{A} \rightarrow \mathcal{B}$

$(\mathcal{A} \rightarrow \mathcal{B}) \; \rightarrow \; (\neg\mathcal{B} \rightarrow \neg\mathcal{A})$

$\neg\mathcal{B} \rightarrow \neg\mathcal{A}$

$< C > \neg\mathcal{B} \rightarrow < C > \neg\mathcal{A}$

$(< C > \neg\mathcal{B} \rightarrow < C > \neg\mathcal{A}) \; \rightarrow (\neg < C > \neg\mathcal{A} \rightarrow \neg < C > \neg\mathcal{B})$

$[C]\mathcal{A} \rightarrow [C]\mathcal{B}$

$\underline{(\mathsf{AX})_{[\leftrightarrow]}}$ As twice above : switch to the contrapositive and play with the definitions.

$\underline{(\mathsf{AX})_{[;]}}$ As thrice above : switch to the contrapositive and play with the definitions.

As mentioned in the 1st section, we won't give a proof of **Theorem 18**, since it may be simply regarded as an expansion of the definition of $\vdash$.

Proof of Theorem 19—soundness of the $\vdash$-system.

This comes in three parts. We assume given a 'derivation plus justifications' as described in the statement of **Theorem 18** (we are only dealing with derivations with no premisses, that is, the case when Γ is empty), and show that the kth line of the derivation is true in $\mathbf{N}$, assuming that all the earlier lines are true in $\mathbf{N}$. The three parts here correspond to the three types of justification, as in (1), (2) and (3) in the statement of **Theorem 18**.

As for (1), if $\mathcal{J}_k$ is a $\vdash''$-rule for producing the kth line as its consequent, using some of the earlier lines as antecedents, we may appeal to the soundness of the $\vdash'$-system (**Theorem 15**) except when the rule involved is the new one :

$\underline{\text{Validity of } (\mathsf{AX})_{<\nabla>}}$. The required truth in $\mathbf{N}$ of

$$< C_{\nabla XC} > \mathcal{A} \quad \longleftrightarrow \quad < \nabla XC > \mathcal{A}$$

will clearly follow once we prove $m(C_{\nabla XC}) = m(\nabla XC)$. More generally, we'll prove that

$$m(C_{\nabla XD}) = \bigcup_{k \geq 0} m(C_{D^{<k>}}) \ .$$

To prove $\subset$ here is a straightforward induction on C: The first three cases give equality just as easily—when C is an assignment command or of the form ∇XB, we have $C_E = C$ for all E, so the result is trivial; when $C = callX$, the result amounts to the definition of $m(\nabla XD)$—and the two remaining inductive cases are just mechanical from definitions.

The proof of $\supset$ in the display above relates to a couple of lemmas of independent interest:

Monotonicity Lemma. *If $m(D) \subset m(E)$, then $m(C_D) \subset m(C_E)$.*

Proof. Proceed by induction on C. As above, three cases are trivial, and the other two are entirely mechanical.

To finish the validity proof: by definition of $m(\nabla XD)$, for all k we have $m(D^{<k>}) \subset m(\nabla XD)$. By monotonicity, $m(C_{D^{<k>}}) \subset m(C_{\nabla XD})$. So that does it.

The following is, in the present language, a rather famous "fixpoint" result, easily deduced now, but (surprisingly) not needed for anything in this paper, it seems.

Park's Lemma. *If $m(C_E) \subset m(E)$, then $m(\nabla XC) \subset m(E)$.*

Proof. Proceed by induction on k to show $m(C^{<k>}) \subset m(E)$. For $k = 0$ it's trivial, and the passage from k to $k+1$ is immediate by taking $D = C^{<k>}$ in monotonicity to get $m(C^{<k>}) \subset m(E) \implies m(C^{<k+1>}) \subset m(E)$.

As for (2), that is, when $\mathcal{J}_k$ is a $\vdash'$-derivation witnessing

$$\Omega \cup \{\mathcal{A} \to [\nabla XC]\mathcal{B}\} \vdash' \mathcal{A} \to [C_{\nabla XC}]\mathcal{B} ,$$

with all wfs's in the finite set Ω occurring as lines above the kth, the truth of the kth line, namely $\mathcal{A} \to [\nabla XC]\mathcal{B}$, is immediate from **17**, since all those higher lines are assumed to be themselves ttN. What we've just done is observe that validity of the rule $(\mathsf{RCS})_{[\ ]\to}$ has already been proved in **17**.

As for (3), that is, when $\mathcal{J}_k$ is a $\vdash''$-derivation witnessing

$$\Omega \cup \{\ \mathcal{A} \to \mathcal{B}\} \quad \vdash^w \quad (|\mathcal{A}|^{w \mapsto w+1} \to \mathcal{B}) ,$$

and the variable w does not occur in the wfs $\mathcal{B}$, assume $\neg|\mathcal{A}|^{w \mapsto 0}$ ttN and that $\mathcal{C}$ ttN for all $\mathcal{C} \in \Omega$. We prove by induction on k, as required, that, for all k,

$$(\mathcal{A} \to \mathcal{B})\ \mathsf{tt@}s_{w \mapsto k} \quad \text{for all } s .$$

For $k = 0_{\mathbf{N}}$, we have $|\mathcal{A}|^{w \mapsto 0}$ $\mathsf{ff@}s$ for all s, and so $\mathcal{A}$ $\mathsf{ff@}s_{w \mapsto 0_{\mathbf{N}}}$ for all s, and so $(\mathcal{A} \to \mathcal{B})$ $\mathsf{tt@}s_{w \mapsto 0_{\mathbf{N}}}$ for all s.
For the inductive step, by the inductive assumption and by the fact above about Ω respectively, both $\mathcal{A} \to \mathcal{B}$ and all wfs's in Ω are $\mathsf{tt@}s_{w \mapsto k}$ for all s. Thus by the general lemma immediately below, the $\vdash^w$ assumption above shows that $|\mathcal{A}|^{w \mapsto w+1} \to \mathcal{B} = |\mathcal{A} \to \mathcal{B}|^{w \mapsto w+1}$ is also $\mathsf{tt@}s_{w \mapsto k}$ for all s. But

that last statement amounts to $\mathcal{A} \to \mathcal{B}$ being $\mathsf{tt}@(s_{w\mapsto k})_{w\mapsto s_{w\mapsto k}(w+1)}$ for all s. And the latter state is simply $s_{w\mapsto k+_{\mathbf{N}}1_{\mathbf{N}}}$, as required.

Lemma 24. (Mendelsonian $<$ $\vdash^w$-validity $<$ Endertonian) *Let $k \in \mathbf{N}$. If $\Gamma \vdash^w \mathcal{A}$, and all $\mathcal{B}$ in Γ are $\mathsf{tt}@s_{w\mapsto k}$ for all s, then $\mathcal{A}$ is $\mathsf{tt}@s_{w\mapsto k}$ for all s.*

Remark. Saying "$\mathsf{tt}@s_{w\mapsto k}$ for all s" is just saying "true at all states with that fixed value, k, for w".

Proof. Once again go through, rule-by-rule, the proof of **14** plus the extra rule done just above, i.e. the proof of the soundness of the $\vdash''$-system. But this time, each rule must be valid in the stronger sense given in the lemma, using the fact that, for five of the rules, we have those restrictions re w not being essentially involved. (The axiom-style rules are trivial cases here.) Inspecting those proofs, all can be construed to have the form of showing the truth of antecedent(s) at some state $\bar{s}$ implying the truth of the consequent at some related state s which is 'arbitrary'. With the extra restrictions referred to above, in each case the value on w of the two states stays the same, and that's exactly what we need.

4. Proofs re completeness.

We shall leave the proofs of **22.0** and **23.0** to Appendix/Section 5, which is not untraditional. As mentioned, all the ideas in the following proof are due to Harel [**H**], in his sketch proof for a quite general dynamic logic system, whose specialization to the deterministic case deals with iteration as opposed to recursion. But this part of the proof seems to be very general.

Proof of Theorem 21. Accuse an occurrence of $\forall x$ in $\mathcal{A}$ of being dull when its scope is a $1^{\underline{\text{st}}}$order formula. Define, for each $\mathcal{A}$,

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

Then $\mathcal{A}$ is a $1^{\underline{\text{st}}}$order formula if and only if $M_{\mathcal{A}} = 0$.

We shall prove by induction on $M_{\mathcal{A}}$ that

$$(*) \quad \text{if } \mathcal{A} \text{ is true in } \mathbf{N}, \text{ then } \quad \vdash \mathcal{A} \ .$$

By (ORACLE), the start of the induction is trivial.

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

First reduce it to proving $(*)$ for wfs's $\mathcal{C} \to \mathsf{sym}\mathcal{B}$, in the four cases of the symbol string

$$\mathsf{sym} \ = \ < C > \ \text{ or } \ \neg < C > \ \text{ or } \ \forall x \ \text{ or } \ \neg\forall x \ ,$$

where the latter two cases involve non-dull occurrences (that is, $\mathcal{B}$ is not a $1^{\underline{\text{st}}}$order formula). Further reduce it to proving $(*)$ for assertions of the form $F \longrightarrow < C > G$ and $F \longrightarrow \neg < C > G$ for $1^{\underline{\text{st}}}$order formulas F and G. But now we are down to the two particular cases of when $M_{\mathcal{A}} = 1$ which are the hypotheses here, i.e. Theorems **23** and **22** respectively.

Lemma. *For any wfs $\mathcal{C}$, there is a propositional derivation of a wfs of the form*

$$\mathcal{C} \ \longleftrightarrow \wedge_{\alpha} \mathcal{D}_{\alpha} \ ,$$

where the finite conjunction on the right-hand side has each $\mathcal{D}_{\alpha}$ as a finite disjunction of wfs's, each of which has the following form : either each is a $1^{\underline{\text{st}}}$order formula , or else at least one of them has the form $\mathsf{sym}\mathcal{B}$ for our four possible sym's, where $\mathcal{B}$ is not a $1^{\underline{\text{st}}}$order formula when sym is either $\forall x$ or $\neg\forall x$.

(All we are really talking about here is conjunctive and disjunctive form.)

The proof of this proceeds by structural induction on $\mathcal{C}$, and is simplified by simultaneously proving the dual fact, where the prefixes "con" and "dis" trade places. The initial case is trivial. In two inductive cases, namely $\mathcal{A} \mapsto \forall x\ \mathcal{A}$; and $\mathcal{A} \mapsto < C > \mathcal{A}$, one simply has a single conjunct (resp. disjunct), which itself is a single disjunct (resp. conjunct). The inductive step for negations is easy because of proving the duals simultaneously : up to cancellation of double negations, deMorganization converts each expression from the right-hand side of the lemma to its dual. The symmetry breaks for the final case of conjuncting two assertions. For the actual lemma statement, the result is numbingly obvious. For the dual statement, it is run-of-the-mill obvious by using distributivity of $\wedge$ over $\vee$.

Since a conjunction is derivable (resp. true in $\mathbf{N}$) if and only if each conjunct has the same property (using propositionality of our system for the derivability), and since every true $1^{\underline{\text{st}}}$order formula is a premiss, the lemma reduces the question to proving $(*)$ only for assertions of the form $\mathcal{E} \vee \mathsf{sym}\mathcal{B}$. (One may use $\mathcal{E} = \ 0 \approx 1$ for any conjunct consisting of a single disjunct $\mathsf{sym}\mathcal{B}$.) Taking $\mathcal{A}$ as $\neg\mathcal{E}$,

we need only prove $(*)$ **for wfs's of the form** $\mathcal{A} \rightarrow \mathsf{sym}\mathcal{B}$,

where, since $\mathcal{B}$ is not a $1^{\underline{\text{st}}}$order formula in two of four relevant cases, the inductive assumption applies to all assertions whose 'M-function' does not exceed that of one or the other of $\mathcal{A}$ or $\mathcal{B}$. All we are saying here is that

$$M_{\mathcal{A} \rightarrow \mathsf{sym}\mathcal{B}} \ = \ M_{\mathcal{A}} + M_{\mathcal{B}} + 1 \ .$$

Using **22.0**(b), choose A and B, each a $1^{\underline{\text{st}}}$order formula , such that both $A \longleftrightarrow \mathcal{A}$ and $B \longleftrightarrow \mathcal{B}$ are true in $\mathbf{N}$.

The proof will now be completed by considering each of the four possibilities for sym.

$\underline{\mathsf{sym} \text{ is } < C >}$: Use (HYSY) after establishing the derivability of the three $\rightarrow$'s below :

$$\mathcal{A} \ \rightarrow \ A \ \rightarrow \ \ < C > B \ \rightarrow \ \ < C > \mathcal{B} \ .$$

The wfs $\mathcal{A} \rightarrow A$ is true, therefore derivable by the induction on the number M.

The wfs $B \rightarrow \mathcal{B}$ is true, therefore derivable by induction, and then so is the wfs $< C > B \rightarrow < C > \mathcal{B}$ derivable, using $(\text{UNAR})_{<>}$.
Derivability of $A \rightarrow < C > B$ follows from its truth and our hypothesis here that **23** holds. Its truth is clear, since it 'factors into true wfs's' :

$$A \ \rightarrow \ \mathcal{A} \ \rightarrow \ \ < C > \mathcal{B} \ \rightarrow \ \ < C > B \ .$$

sym is $\forall x$: Just replace $< C >$ by $\forall x$ everywhere in the previous paragraph, and appeal to $(\text{UNAR})_{\forall}$ rather than $(\text{UNAR})_{<>}$, and get derivability of $A \rightarrow \forall x \ B$ from (ORACLE).

sym is $\neg < C >$: Just replace $< C >$ by $\neg < C >$ everywhere in the second previous paragraph, and make a few tiny wording changes :

Use $\mathcal{B} \rightarrow B$ true, so derivable by induction on the number M, to get $< C > \mathcal{B} \rightarrow < C > B$, and thence $\neg < C > B \rightarrow \neg < C > \mathcal{B}$ both derivable.

This case of course uses also our hypothesis here that **22** holds.

sym is $\neg\forall x$: Write out the previous paragraph properly, then replace each $\neg < C >$ by $\neg\forall x$, again appealing to the oracle for derivability of the wfs $A \rightarrow \neg\forall x \ B$.

That completes the proof of **21**.

Proof of 22.5. For the sequencing command, given F and G such that $F \to [(D_1; D_2)]G$ ttN, by **22.0**(b), choose a $1^{\underline{\text{st}}}$order J such that $J \longleftrightarrow [D_2]G$ ttN. Then $\Gamma_2 \vdash' J \to [D_2]G$ since Γ_2 der $[D_2]$, so

$$\Gamma_2 \ \vdash' \ [D_1]J \to [D_1][D_2]G \qquad\qquad (I)$$

by $(\mathsf{UNAR})_{[\ ]}$. Since

$$(F \to [D_1]J) \ \longleftrightarrow \ (F \to [D_1][D_2]G) \quad \mathsf{ttN} \ ,$$

and $(F \to [D_1][D_2]G)$ ttN, we get

$$\Gamma_1 \ \vdash' \ F \to [D_1]J \qquad\qquad (II)$$

since Γ_1 der $[D_1]$. Now $(I), (II)$ and (HYSY) give

$$\Gamma_1 \cup \Gamma_2 \ \vdash' \ F \to [D_1][D_2]G \ .$$

Then $(\mathsf{AX})_{[;]}$ plus (HYSY) allow one to change $[D_1][D_2]$ to $[(D_1; D_2)]$ in the last display, as required.

For 'if-then-else', given F and G with $F \to [ite(H)(D_1)(D_2)]G$ ttN, it follows from the validity of $(\mathsf{AX})_{[ite]}$ that

$$F \ \to \ (H \to [D_1]G) \wedge (\neg H \to [D_2]G) \quad \mathsf{ttN} \ .$$

And so both the following are true in $\mathbf{N}$:

$$F \wedge H \ \to \ [D_1]G \quad \text{and} \quad F \wedge \neg H \ \to \ [D_2]G \ .$$

But then the two Γ_i der $[D_i]$ guarantee that these two can be derived from $\Gamma_1 \cup \Gamma_2$. The rules $(\mathsf{P})_1$ and $(\mathsf{P})_2$ now handily establish

$$\Gamma_1 \cup \Gamma_2 \ \vdash' \ F \ \to \ (H \to [D_1]G) \wedge (\neg H \to [D_2]G) \ .$$

But $(\mathsf{AX})_{[ite]}$ and $(\leftrightarrow)$ show

$$\vdash' \ (H \to [D_1]G) \wedge (\neg H \to [D_1]G) \quad \longrightarrow \quad [ite(H)(D_1)(D_2)]G \ ,$$

so (HYSY) applied to the last two displays finishes the job.

Proof of 22.4. Assume $A(\nabla XC, \vec{x} \approx \vec{z}) \wedge |F|^{\vec{x}\to\vec{z}}$ $\mathsf{tt}@s'$. Then, for some s we have

$$(s, s') \in m(\nabla XC) \quad , \quad \vec{x} \approx \vec{z}\ \mathsf{tt}@s \quad , \quad F\ \mathsf{tt}@s'_{\vec{x}\mapsto s'(\vec{z})} \ .$$

Now $s'_{\vec{x}\mapsto s'(\vec{z})}(\vec{x}) = s'(\vec{z}) = s(\vec{z}) = s(\vec{x})$. The last equality comes from the truth of $\vec{x} \approx \vec{z}$ indicated in the previous display. The equality before that is because all the variables in ∇XC are from $\vec{x}$—none are from $\vec{z}$.

Thus $s'_{\vec{x}\mapsto s'(\vec{z})}$ agrees with s except possibly on $\vec{z}$. But no z_i is in F. And so F $\mathsf{tt}@s$. Combined with $(s, s') \in m(\nabla XC)$ and $F \to [\nabla XC]G$ $\mathsf{tt}@s$, we get G $\mathsf{tt}@s'$, as required.

Proof of that 22.4 implies 22.3. First we prove the special case when no $\vec{z}$-variables occur in F nor G. Here is a derivation, shortening $A(\nabla XC, \vec{x} \approx \vec{z})$ to just A :

$\vec{x} \approx \vec{z} \ \to\ [\nabla XC]A$	(premiss)
$\|F\|^{\vec{x}\to\vec{z}} \to [\nabla XC]\|F\|^{\vec{x}\to\vec{z}}$	$(\mathsf{AX})_{disj}$
$\vec{x} \approx \vec{z} \wedge \|F\|^{\vec{x}\to\vec{z}} \to [\nabla XC](A \wedge \|F\|^{\vec{x}\to\vec{z}})$	(AND)
$A \wedge \|F\|^{\vec{x}\to\vec{z}} \to G$	(ORAC) and by **22.4**
$[\nabla XC](A \wedge \|F\|^{\vec{x}\to\vec{z}}) \to [\nabla XC]G$	$(\mathsf{UNAR})_{[\ ]}$
$\vec{x} \approx \vec{z} \wedge \|F\|^{\vec{x}\to\vec{z}} \to [\nabla XC]G$	(HYSY)
$\vec{x} \approx \vec{x} \wedge F \to [\nabla XC]G$	$(\mathsf{SUB})_2^{\vec{z}\to\vec{x}}$
$F \to\ \vec{x} \approx \vec{x} \wedge F$	(ORAC)
$F \to [\nabla XC]G$	(HYSY)

Line 4 uses that no z_i is in G or F, and the latter also gives line 7, since then $||F|^{\vec{x}\to\vec{z}}|^{\vec{z}\to\vec{x}} = F$.

Now in the general case, let $\vec{u}$ be a matching list of pairwise distinct variables, disjoint from $\vec{x}$ and $\vec{z}$. Define F_1 to be $|F|^{\vec{z}\to\vec{u}}$ and let $G_1 := |G|^{\vec{z}\to\vec{u}}$, so F_1 and G_1 have no $\vec{z}$-variables. Since the validity of $(\mathsf{SUB})_1^{\vec{z}\to\vec{u}}$ entails that the truth of $F \to [\nabla XC]G$ in $\mathbf{N}$ also guarantees $\mathsf{tt}\mathbf{N}$ for the wfs $F_1 \to [\nabla XC]G_1$, the special case above guarantees that

$$\{\vec{x} \approx \vec{z} \ \to\ [\nabla XC]A(\nabla XC, \vec{x} \approx \vec{z})\} \ \vdash' \ (F_1 \to [\nabla XC]G_1) \ .$$

But now one application of $(\mathsf{SUB})_1^{\vec{u}\to\vec{z}}$ allows us the erase the subscript on F and G in this last display, as required.

Proof that (22.3 plus 22.5) imply 22.2. Proceed by induction on D.

When $D = callX$: By $(\mathsf{AX})_{[call]}$ and $(\mathsf{P})_4$, we have $\vdash'\ (F \to [callX]G)$ (for *any* F and G).

When $D = y \leftarrow\!\!\div t$: By the soundness of $(\mathsf{AX})_{\leftarrow\!\!\div}$, we see that

$$(F \to [y \leftarrow\!\!\div t]G)\ \mathsf{ttN} \quad \Longrightarrow \quad (F \to |G|^{y\to t})\ \mathsf{ttN}\ .$$

So here's a little derivation doing the job :

$F \to |G|^{y\to t}$ $\quad (\mathsf{ORAC})$

$|G|^{y\to t} \leftrightarrow\ [y \leftarrow\!\!\div t]G$ $\quad (\mathsf{AX})_{[\leftarrow\!\!\div]}$

$|G|^{y\to t} \to\ [y \leftarrow\!\!\div t]G$ $\quad (\leftrightarrow)$

$F \to [y \leftarrow\!\!\div t]G$ $\quad (\mathsf{HYSY})$

When $D = \nabla XC$: The set $\Gamma_{\nabla XC} = \Gamma_D$ from **22.2** contains the element $\vec{x} \approx \vec{z}\ \to\ [\nabla XC]A$, so **22.3** gives the result immediately.

When $D = (D_1; D_2)$ or $ite(H)(D_1)(D_2)$: Apply **22.5** with $\Gamma_i = \Gamma_{D_i}$, noting that $\Gamma_D = \Gamma_{D_1} \cup \Gamma_{D_2}$ for both types of D here. Use the inductive hypotheses for D_1 and D_2 as the hypotheses in **22.5**. The latter's conclusion is exactly the required result.

Proof that 22.2 implies 22.1. We prove the displayed claim with Ω as specified in the last two lines of **22.1**. In **22.2**, take $F = \vec{x} \approx \vec{z}$ and $G = A(\nabla XC, \vec{x} \approx \vec{z})$ and $D = C_{\nabla XC}$. To see that $(F \to [D]G)$ is true in $\mathbf{N}$, the validity of $(\mathsf{AX})_{[\nabla]}$ reduces it to showing that $(F \to [\nabla XC]G)\ \mathsf{ttN}$. The latter is immediate from the definition of A in **22.0** and the comment just after that theorem. So we can apply **22.2** and get

$$\Gamma_{C_{\nabla XC}}\ \vdash'\ (F \to [D]G)\ .$$

We are required to show that

$$\Omega \cup \{\vec{x} \approx \vec{z}\ \to\ [\nabla XC]A(\nabla XC, \vec{x} \approx \vec{z})\}\ \vdash'\ (F \to [D]G)\ .$$

So it remains only to show that

$$\Gamma_{C_{\nabla XC}} \ \subset \ \Omega \cup \{\vec{x} \approx \vec{z} \ \rightarrow \ [\nabla XC]A(\nabla XC, \vec{x} \approx \vec{z})\} \ .$$

Thus, we need that, if E is such that ∇XE is a subcommand of $C_{\nabla XC}$, then ∇XE is either ∇XC or is a subcommand of C. (Note that, by unique readability, ∇XE can equal $C_{\nabla XC}$ only when $E = C = callX$.)

To complete the proof properly, we really need to give the inductive definition of the word *subcommand*:
The set of proper *subcommands is empty for atomic commands, is the union of the sets of* all *subcommands of C and D when the command is obtained from them by sequencing or ite, and is the set of* all *subcommands of C when the command is ∇XC.*
Now all that remains is to combine this with the inductive definition of C_D to prove that, *for any commands B, C and D, if B is a subcommand of C_D, then it must be : C_D ; or a subcommand of C ; or a subcommand of D.* The inductive proof (on C) is entirely mechanical.

N.B. Just above, it is important to use the case $D = C_{\nabla XC}$ from **22.2**, not just the case $D = \nabla XC$. We cannot use the extra rule $(\mathsf{AX})_{[\nabla]}$ (which is needed for $\vdash''$) when doing a $\vdash'$-derivation, or the trick of 'substituting for ∇XC' could lead to an unsound system—(see the proof of **11**).

Now we do the analogous (but somewhat harder) proofs of the technical results leading to **Theorem 23**.

Proof of 23.5. Just change every [ ] to < > in the proof of **22.5.**

Proof of 23.4(i)(a) Assume $\exists\vec{z}(\exists wB(C) \wedge |G|^{\vec{x}\rightarrow\vec{z}})$ ff@s. Then we shall show F ff@s, as required, by using that $(F \rightarrow< \nabla XC > G)$ tt@s, and instead showing $< \nabla XC > G$ ff@s.
Now $\exists wB(C) \wedge |G|^{\vec{x}\rightarrow\vec{z}}$ ff@s_1 for all s_1 which agree with s except possibly on $\vec{z}$. Fix such an s_1. Then either $|G|^{\vec{x}\rightarrow\vec{z}}$ ff@s_1 or $\exists wB(C)$ ff@s_1. So, by the remark after the definition of $B(C)$, one of the following must hold:
(I) $|G|^{\vec{x}\rightarrow\vec{z}}$ ff@s_1 ; or
(II) for some s_1', $(s_1, s_1') \in m(\nabla XC)$ but $s_1'(\vec{x}) \neq s_1'(\vec{z})$; or
(III) ∇XC 'diverges' at input s_1 .
But (III) $\implies$ $< \nabla XC > G$ ff@s_1 $\implies$ $< \nabla XC > G$ ff@s , as required, the latter $\implies$ because neither C nor G has any variable from $\vec{z}$.

So we may assume:

(IV) ∇XC 'converges' at input s_1 for all such s_1.

This last statement with $s_1 = s$ gives us an s' with $(s, s') \in m(\nabla XC)$. Use it to define $s_2 := s_{\vec{z} \mapsto s'(\vec{x})}$. Then s_2 is one such s_1, so (IV) with $s_1 = s_2$ gives us an s_2' with $(s_2, s_2') \in m(\nabla XC)$. Now

$$s_2'(\vec{z}) \;=\; s_2(\vec{z}) \;=\; s'(\vec{x}) \;=\; s_2'(\vec{x}) \;,$$

so (II) fails when $s_1 = s_2$, forcing (I) to hold for s_2. But that fact [namely, $|G|^{\vec{x} \to \vec{z}}$ $\mathsf{ff}@s_2$] forces G $\mathsf{ff}@s'$, because $s_2(\vec{z}) \;=\; s'(\vec{x})$. Since ∇XC converges at input s to s', we now have $< \nabla XC > G$ $\mathsf{ff}@s$, completing the proof.

Proof of 23.4(i)(b). We have

$$B(w, C, callX, G)\ \mathsf{tt}@s \iff < C^{<s(w)>} G\ \mathsf{tt}@s$$

$$\iff \exists s' \text{ with } (s, s') \in m(C^{<s(w)>}) \text{ and } G\ \mathsf{tt}@s' \;.$$

Now let $\bar{s} := s_{\vec{z} \mapsto s'(\vec{x})}$. We'll show that $B(C)$ and $|G|^{\vec{x} \to \vec{z}}$ are both $\mathsf{tt}@\bar{s}$. Since $\bar{s}$ agrees with s except at $\vec{z}$, this shows that $\exists \vec{z}(B(C) \wedge |G|^{\vec{x} \to \vec{z}})\mathsf{tt}@s$, as required.

As for the latter,

$$|G|^{\vec{x} \to \vec{z}}\ \mathsf{tt}@\bar{s} \iff G\ \mathsf{tt}@\bar{s}_{\vec{x} \mapsto \bar{s}(\vec{z})} \;.$$

But $\bar{s}_{\vec{x} \mapsto \bar{s}(\vec{z})}$ and s' agree except at $\vec{z}$, no variables in G are from $\vec{z}$, and G $\mathsf{tt}@s'$, so that does it.

As for the former,

$$B(C)\ \mathsf{tt}@\bar{s} \iff < C^{<\bar{s}(w)>} > \vec{x} \approx \vec{z}\ \ \mathsf{tt}@\bar{s}$$

$$\iff \exists \bar{s}' \text{ with } (\bar{s}, \bar{s}') \in m(C^{<\bar{s}(w)>}) \text{ and } \vec{x} \approx \vec{z}\ \ \mathsf{tt}@\bar{s}' \;.$$

But $\bar{s}(w) = s(w)$, and the unique $\bar{s}'$ with $(\bar{s}, \bar{s}') \in m(C^{<s(w)>})$ is $s'_{\vec{z} \mapsto s'(\vec{x})}$, where clearly $\vec{x} \approx \vec{z}$ is tt, so we are done.

Proof of 23.4(i)(c). We have

$$B(*, *, \nabla XE, G)\ \mathsf{tt}@s \iff < \nabla XE > G\ \mathsf{tt}@s$$

$$\iff \exists s' \text{ with } (s, s') \in m(\nabla XE) \text{ and } G\ \mathsf{tt}@s' \;.$$

Now let $\bar{s} := s_{\vec{z}\mapsto s'(\vec{x})}$. We'll show that $B(*,*,\nabla XE,\vec{x}\approx\vec{z})$ and $|G|^{\vec{x}\to\vec{z}}$ are both $\mathsf{tt}@\bar{s}$. Since $\bar{s}$ agrees with s except at $\vec{z}$, this shows that

$$\exists\vec{z}(B(*,*,\nabla XE,\vec{x}\approx\vec{z})\wedge|G|^{\vec{x}\to\vec{z}})\ \ \mathsf{tt}@s\ ,$$

as required.

As for the latter,

$$|G|^{\vec{x}\to\vec{z}}\ \mathsf{tt}@\bar{s}\ \iff\ G\ \mathsf{tt}@\bar{s}_{\vec{x}\mapsto\bar{s}(\vec{z})}\ .$$

But $\bar{s}_{\vec{x}\mapsto\bar{s}(\vec{z})}$ and s' agree except at $\vec{z}$, no variables in G are from $\vec{z}$, and $G\ \mathsf{tt}@s'$, so that does it.

As for the former,

$$B(*,*,\nabla XE,\vec{x}\approx\vec{z})\ \ \mathsf{tt}@\bar{s}\ \iff\ <\nabla XE>\vec{x}\approx\vec{z}\ \ \mathsf{tt}@\bar{s}$$

$$\iff\ \exists\bar{s}'\text{ with }(\bar{s},\bar{s}')\in m(\nabla XE>)\text{ and }\vec{x}\approx\vec{z}\ \ \mathsf{tt}@\bar{s}'\ .$$

But $\bar{s}(w)=s(w)$, and the unique $\bar{s}'$ with $(\bar{s},\bar{s}')\in m(\nabla XE)$ is $s'_{\vec{z}\mapsto s'(\vec{x})}$, where clearly $\vec{z}\approx\vec{x}$ is tt, so we are done.

Proof of 23.4(ii). For a contradiction, assume $\neg|B|^{w\to 0}\ \mathsf{ff}@s$, so that $|B|^{w\to 0}\ \mathsf{tt}@s$. Thus $B\ \mathsf{tt}@s_{w\mapsto 0}$. Then, by the remark after the definition of B, there is an s' with $(s_{w\mapsto 0},s')\in m(C^{<s_{w\mapsto 0}(w)>})$. This contradicts the fact that $m(callX)$ [which equals $m(C^{<0>})$] is empty (part of m's definition).

Proof of 23.4(iii). Let $s_+ := s_{w\mapsto s(w)+_{\mathbf{N}}1_{\mathbf{N}}}$. Then the statements in following list are all logically equivalent :

$|B|^{w\to w+1}\ \mathsf{tt}@s\ \ ;\ \ B\ \mathsf{tt}@s_+\ \ ;\ \ <callX_{C^{<s_+(w)>}}>\vec{x}\approx\vec{z}\ \ \mathsf{tt}@s_+\ \ ;$

$<C^{<s_+(w)>}>\vec{x}\approx\vec{z}\ \ \mathsf{tt}@s_+\ \ ;\ \ <C^{<s_+(w)>}>\vec{x}\approx\vec{z}\ \ \mathsf{tt}@s\ \ ;$

$<C_{C^{<s(w)>}}>\vec{x}\approx\vec{z}\ \ \mathsf{tt}@s\ \ ;\ \ B(w,C,C,\vec{x}\approx\vec{z})\ \mathsf{tt}@s\ .$

The 2nd and 6th 'semicolons' appeal to the definition of B; the 4th to the fact that w doesn't occur in $C\cup\vec{x}\cup\vec{z}$; and the 5th to the definition of $C^{<n>}$.

Thus we have proved the stronger result (not needed later), where the symbol $\to$ in **23.4**(iii) is replaced with $\leftrightarrow$.

Proof that 23.4(i) **implies 23.3.** First we prove the special case when no $\vec{z}$-variables occur in F nor in G. Here is a slightly abbreviated $\vdash'$-derivation showing the required $\vdash^w$:

$(\exists w B(C)) \ \rightarrow \ < \nabla X C > \vec{x} \approx \vec{z}$ (premiss)

$\exists w B(C) \wedge |G|^{\vec{x} \to \vec{z}} \ \rightarrow \ < \nabla X C > (\vec{x} \approx \vec{z} \wedge |G|^{\vec{x} \to \vec{z}})$ $(\mathsf{AX})_{<\rightarrow>}$ and (MP)

$\vec{x} \approx \vec{z} \wedge |G|^{\vec{x} \to \vec{z}} \rightarrow G$ (ORAC)

$< \nabla X C > (\vec{x} \approx \vec{z} \wedge |G|^{\vec{x} \to \vec{z}}) \rightarrow < \nabla X C > G$ $(\mathsf{UNAR})_{< >}$

$\exists w B(C) \wedge |G|^{\vec{x} \to \vec{z}} \ \rightarrow \ < \nabla X C > G$ (HYSY)

$\exists \vec{z} (\exists w B(C) \wedge |G|^{\vec{x} \to \vec{z}}) \ \rightarrow \ < \nabla X C > G$ $(\mathsf{PRE})^{\vec{z}}$

$F \rightarrow \ \exists \vec{z} (\exists w B(C) \wedge |G|^{\vec{x} \to \vec{z}})$ (ORAC) and by **23.4**(i)(a)

$F \rightarrow < \nabla X C > G$ (HYSY)

Now in the general case, let $\vec{u}$ be a matching list of pairwise distinct variables, disjoint from $w, \vec{x}$ and $\vec{z}$. Define F_1 to be $|F|^{\vec{z} \to \vec{u}}$ and let G_1 be $|G|^{\vec{z} \to \vec{u}}$, so F_1 and G_1 have no $\vec{z}$-variables. Since the validity of $(\mathsf{SUB})_1^{\vec{z} \to \vec{u}}$ entails that the truth of $F \rightarrow < \nabla X C > G$ in $\mathbf{N}$ also guarantees $\mathsf{tt}\mathbf{N}$ for the wfs $F_1 \rightarrow < \nabla X C > G_1$, the special case above assures us that

$$\{(\exists w B) \ \rightarrow \ < \nabla X C > \vec{x} \approx \vec{z}\} \ \vdash^w \ (F_1 \rightarrow < \nabla X C > G_1) \ .$$

But now one application of $(\mathsf{SUB})_1^{\vec{u} \to \vec{z}}$ (and that doesn't violate the $\vdash^w$-restrictions!) allows us the erase the subscripts on F and G in the last display, as required.

Proof that (23.3 plus 23.5) imply 23.2. This proceeds by induction on commands D_1, which we shall re-name as just D, since there is no danger now of confusing it with the D in **23.1**.

When $D = callX$: By validity of $(\mathsf{AX})_{<call>}$, the wfs $< callX > G$ $\mathsf{ff}@s$ for all s. So the truth in $\mathbf{N}$ of $F \rightarrow < callX > G$ gives us that F $\mathsf{ff}@s$, and so $\neg F$ $\mathsf{tt}@s$, for all s. Now (ORAC) and $(\mathsf{P})_3$ give us a 2-line derivation witnessing $\vdash^w (F \rightarrow < callX > G)$, as required.

When $D = y \leftarrow\!\!\div t$: By the soundness of $(\mathsf{AX})_{<\leftrightarrow>}$, we see that

$$(F \rightarrow < y \leftarrow\!\!\div t > G) \ \mathsf{tt}\mathbf{N} \quad \Longrightarrow \quad (F \rightarrow |G|^{y \to t}) \ \mathsf{tt}\mathbf{N} \ .$$

So here's a little derivation doing the job :

$F \to |G|^{y \to t}$ (ORAC)

$|G|^{y \to t} \leftrightarrow \ < y \leftarrow\!\!\div t > G$ $(\mathsf{AX})_{<\leftrightarrow>}$

$|G|^{y \to t} \to \ < y \leftarrow\!\!\div t > G$ $(\leftrightarrow)$

$F \to < y \leftarrow\!\!\div t > G$ (HYSY)

When $D = \nabla XE$: The set $\Lambda_{\nabla XE} = \Lambda_D$ from **23.2** contains the element $\exists w\ B(E) \to < \nabla XE > \vec{x} \approx \vec{z}$, so **23.3** with $C = E$ gives the result immediately.

When $D = (D_1; D_2)$ or $ite(H)(D_1)(D_2)$: Apply **23.5** with $\Gamma_i = \Lambda_{D_i}$, noting that $\Lambda_D = \Lambda_{D_1} \cup \Lambda_{D_2}$ for both types of D here. Use the inductive hypotheses for D_1 and D_2 as the hypotheses in **23.5**. The latter's conclusion is exactly the required result.

Proof of 23.1. This also proceeds by induction on D.

When $D = y \leftarrow\!\!\div t$: For any D with no free $callX$-subcommands, and twice using that $D_E = D$ for such D and all E,

$$(I) \quad B(w, C, D, G)\ \mathsf{tt}@s \iff < D_{C^{<s(w)>}} > G\ \mathsf{tt}@s \iff < D > G\ \mathsf{tt}@s\ ;$$

and

$$(II) \quad D_{\nabla XC} = D\ .$$

For D here, since $< D > G \longleftrightarrow |G|^{y \to t}$ is ttN, we can take $B(w, C, D, G)$ to be $|G|^{y \to t}$ by (I), and, by (II), we only need to establish

$$\vdash^w \ (|G|^{y \to t} \to \ < y \leftarrow\!\!\div t > G)\ ,$$

which is immediate from $(\mathsf{AX})_{<\leftrightarrow>}$.

For more explicit details re the next two paragraphs, see the proof of **M23.1** in [**mutu**]

When $D = callX$: Modify the (two paragraphs plus derivation)-proof of **23.3** above as follows:
Start by saying "Let $F = B(w, C, callX, G)$".
Erase "$\exists w$" on lines 1, 2, 5, 6 and 7 of the derivation.
Change **23.4**(i)(a) to **23.4**(i)(b) in the justification of line 7.

Note that the final line is the appropriate one to prove this case of **23.1**, because $[callX]_{\nabla XC} = \nabla XC$.

When $D = \nabla XE$: It suffices to prove

$$B(*,*,\nabla XE,\vec{x}\approx\vec{z}) \rightarrow \ < \nabla XE > \vec{x}\approx\vec{z}) \ \vdash^w \ B(*,*,\nabla XE,G) \rightarrow \ < \nabla XE > G)\,,$$

since to the right of the $\vdash^w$ we have the same wfs as in this case of **23.1**, and on the left we have an element of Ω. (This is where D being a subcommand of C is used.)

Again modify the proof of **23.3**, this time as follows:
Start by saying "Let $F = B(*,*,\nabla XE,G)$".
Change each of the "$\exists w B(C)$"s (on lines 1, 2, 5, 6 and 7 of the derivation) to $B(*,*,\nabla XE,\vec{x}\approx\vec{z})$.
Change all remaining C's to E's.
Change **23.4**(i)(a) to **23.4**(i)(c) in the justification of line 7.

When $D = (D_1; D_2)$: Firstly, letting $B_* \ := \ B(w,C,D_1,B(w,C,D_2,G))$, we'll show that we may take

$$B(w,C,(D_1;D_2),G) \ = \ B_*\ ,$$

by showing

$$B(w,C,(D_1;D_2),G) \ \longleftrightarrow \ B_* \quad \mathsf{ttN}\ .$$

This is immediate from the following logical equivalences:

$$B(w,C,(D_1;D_2),G)\ \mathsf{tt@}s \iff \ < (D_1;D_2)_{C^{<s(w)>}} > G\ \mathsf{tt@}s \iff$$

$$< (D_{1_{C^{<s(w)>}}}; D_{2_{C^{<s(w)>}}}) > G\ \mathsf{tt@}s \iff \ < D_{1_{C^{<s(w)>}}} >< D_{2_{C^{<s(w)>}}} > G\ \mathsf{tt@}s$$
$$\iff \ < D_{1_{C^{<s(w)>}}} > B(w,C,D_2,G)\ \mathsf{tt@}s \iff B_*\ \mathsf{tt@}s\ .$$

It remains then to establish, with

$$\Delta \ := \ \Omega \ \cup \ \{B(C) \rightarrow < \nabla XC > \vec{x}\approx\vec{z}\}\ ,$$

that

$$\Delta \quad \vdash^w \quad (B_* \rightarrow < D_{\nabla XC} > G)\ .$$

By the inductive assumption re D_1, we have

$$\Delta \quad \vdash^w \quad B_* \rightarrow < D_{1_{\nabla XC}} > B(w,C,D_2,G) \qquad (I)$$

And the inductive assumption re D_2 gives

$$\Delta \;\; \vdash^w \;\; B(w, C, D_2, G) \rightarrow < D_{2_{\nabla XC}} > G \qquad (II)$$

Combining $(\mathsf{UNAR})_{< >}$ and $(\mathsf{AX})_{<;>}$ allows us to derive

$$< D_{1_{\nabla XC}} > B(w, C, D_2, G) \rightarrow < (D_{1_{\nabla XC}}; D_{2_{\nabla XC}}) > G$$

from the RHS of (II). Using (HYSY) to combine the latter with (I) then yields the required result, since $D_{\nabla XC} = (D_{1_{\nabla XC}}; D_{2_{\nabla XC}})$.

Note that the use of $(\mathsf{UNAR})_{< >}$ is fine for $\vdash^w$ here, because w does not occur in $D_{1_{\nabla XC}}$, all of whose variables are from $\vec{x}$.

When $D = ite(H)(D_0)(D_1)$: In this case, letting

$$B_* \;:=\; (H \rightarrow B_+) \wedge (\neg H \rightarrow B_-) \;,$$

where

$$B_+ \;:=\; B(w, C, D_0, G) \;\text{ and }\; B_- \;:=\; B(w, C, D_1, G) \;,$$

we'll first show that we may take

$$B(w, C, ite(H)(D_0)(D_1), G) \;=\; B_* \;,$$

by showing

$$B(w, C, ite(H)(D_0)(D_1), G) \;\longleftrightarrow\; B_* \qquad \mathsf{ttN} \;.$$

This is immediate from the following logical equivalences:

$$B(w, C, ite(H)(D_0)(D_1), G) \;\; \mathsf{tt@}s \iff < ite(H)(D_0)(D_1)_{C^{<s(w)>}} > G \;\; \mathsf{tt@}s$$

$$\iff < ite(H)(D_{0_{C^{<s(w)>}}})(D_{1_{C^{<s(w)>}}}) > G \;\; \mathsf{tt@}s \iff$$

$$(H \rightarrow < D_{0_{C^{<s(w)>}}} > G) \wedge (\neg H \rightarrow < D_{1_{C^{<s(w)>}}} > G) \;\; \mathsf{tt@}s \iff$$

$$(H \rightarrow B(w, C, D_0, G)) \wedge (\neg H \rightarrow B(w, C, D_1, G)) \;\; \mathsf{tt@}s \;.$$

It remains then to establish, with Δ as in the previous case,

$$\Delta \;\; \vdash^w \;\; (B_* \rightarrow < D_{\nabla XC} > G) \;.$$

The inductive hypotheses re D_i for $i = 0, 1$ give

$$\Delta \quad \vdash^w \quad B_{(-1)^i} \rightarrow < D_{i_{\nabla XC}} > G \ .$$

Simple propositional arguments plus $(\mathsf{AX})_{<ite>}$ give that $B_* \rightarrow < D_{\nabla XC} > G$ can be $\vdash^w$-derived from

$$\{(B_* \rightarrow (H \rightarrow < D_{0_{\nabla XC}} > G)) \wedge (B_* \rightarrow (\neg H \rightarrow < D_{1_{\nabla XC}} > G))\} \ ,$$

using that $D_{\nabla XC} = ite(H)(D_{0_{\nabla XC}})(D_{1_{\nabla XC}})$. So we just need to 'fill in the middle', showing both

$$\{B_+ \rightarrow < D_{0_{\nabla XC}} > G\} \quad \vdash^w \quad B_* \rightarrow (H \rightarrow < D_{0_{\nabla XC}} > G)$$

and

$$\{B_- \rightarrow < D_{1_{\nabla XC}} > G\} \quad \vdash^w \quad B_* \rightarrow (\neg H \rightarrow < D_{1_{\nabla XC}} > G) \ .$$

A derivation for the second is exactly parallel (by changing subscript 0 to 1, $+$ to $-$, and H to $\neg H$) to the following (purely propositional) derivation for the first, so that will do it.

$B_+ \rightarrow < D_{0_{\nabla XC}} > G$	(premiss)
$(H \rightarrow B_+) \wedge H \rightarrow B_+$	(TAUT)
$(H \rightarrow B_+) \wedge H \rightarrow < D_{0_{\nabla XC}} > G$	(HYSY)
$B_* \wedge H \rightarrow (H \rightarrow B_+) \wedge H$	(TAUT)
$B_* \wedge H \rightarrow < D_{0_{\nabla XC}} > G$	(HYSY)
$B_* \rightarrow (H \rightarrow < D_{0_{\nabla XC}} > G)$	$(\mathsf{P})_1$

5. Appendix on $1^{\underline{\text{st}}}$order Definability—Proofs of 22.0 and 23.0.

As mentioned earlier, we shall use some very basic original results from 80 years ago of Gödel, rather than re-inventing the wheel with coding of symbol strings etc.

Recall that a *semi-decidable* relation is a $\{0,1\}$- or ff/tt-valued function whose domain's subset which takes the value 1 is *recursively enumerable*. Equivalently, there must be an 'algorithm' with input values from the domain which produces the answer "yes" exactly when 1 is the function's value (and whatever things it happens to do when 0 is the value, one could adjust it to be an algorithm which always 'loops' there). Now we will take for granted that implementing a command from $\mathcal{R}ec$ is acceptable as an algorithm. (If someone wishes to challenge on that, one hopes they are correct, since producing a counterexample to Church's thesis would bring fame and fortune!) Actually, showing $\mathcal{R}ec$ computes only (indeed exactly) the (partial) recursive functions is routine—see **[HHH], Sect. V**

The needed result is this:

Theorem 5.0.0. (essentially Gödel's—see **[HHH], V-2.3**) *Assume that one has a semi-decidable relation $R : \mathbf{N}^\ell \to \{0,1\}$. Then, for all strings of variables $\vec{x} = (x_1, \cdots, x_\ell)$, there is a $1^{\underline{\text{st}}}$order formula J_R, all of whose variables are from $\vec{x}$, and such that*

$$R(a_1, \cdots, a_\ell) = 1 \iff |J_R|^{\vec{x}\to\vec{a}}\ \mathsf{ttN} \ .$$

Expressed alternatively: for all states s,

$$J_R\ \mathsf{tt}@s \iff R(s(\vec{x})) = 1 \ .$$

Proof of 22.0(a). Let $\vec{y} = (y_1, \cdots, y_k)$ be a list of distinct variables which includes all of those occurring in C and F, and let $\vec{z}$ be a matching but disjoint list of distinct variables.

Firstly, we construct $G = G_C$, a $1^{\underline{\text{st}}}$order formula all of whose variables are from $\vec{y} \cup \vec{z}$, such that $\quad G\ \mathsf{tt}@s'' \iff$

for some (and so all) s_1 and s_1', agreeing away from $\vec{y}$, and such that

$$s_1(\vec{y}) = s''(\vec{z}) \text{ and } s_1'(\vec{y}) = s''(\vec{y}), \text{ we have } (s_1, s_1') \in m(C) \ .$$

For this, in **5.0.0** take $\ell = 2k$ and define

$$R = R_C : \mathbf{N}^k \times \mathbf{N}^k \ \to \ \{0,1\} \qquad \text{by}$$

$$R(\vec{a},\vec{b}) = 1 \iff (s_1, s_1') \in m(C) \text{ for all } s_1 \text{ and } s_1'$$
$$\text{agreeing away from } \vec{y} \text{ for which } s_1(\vec{y}) = \vec{b} \text{ and } s_1'(\vec{y}) = \vec{a}\ .$$

It is easy to see that this relation is semi-decidable—given $(\vec{a},\vec{b})$, one runs the command C on input $\vec{b}$. If $R(\vec{a},\vec{b}) = 1$ is actually true, after finitely many steps in this computation, the machine spits out $\vec{a}$, and the 'semi-deciding' has been done. And so **5.0.0** guarantees us the required G.

Now let $A := \exists\vec{z}(G_C \wedge |F|^{\vec{y}\to\vec{z}})$. To show that this A is the "$A(C,F)$" which we need:

$$A\ \mathtt{tt}@s' \iff$$
$$\exists s'', \text{ agreeing with } s' \text{ except on } \vec{z}, \text{ satisfying } G_C\ \mathtt{tt}@s'' \text{ and } |F|^{\vec{y}\to\vec{z}}\ \mathtt{tt}@s''$$
$$\iff [\exists s'', \text{ agreeing with } s' \text{ except on } \vec{z}, \text{ satisfying both } |F|^{\vec{y}\to\vec{z}}\ \mathtt{tt}@s'' \text{ and}$$
$$\text{for all } s_1 \text{ and } s_1', \text{ agreeing away from } \vec{y}, \text{ and such that } s_1(\vec{y}) = s''(\vec{z}) \text{ and}$$
$$s_1'(\vec{y}) = s''(\vec{y}), \text{ we have } (s_1, s_1') \in m(C)]$$
$$\iff \exists s \text{ with } (s,s') \in m(C) \text{ and } F\ \mathtt{tt}@s\ .$$

(Pardon me for using "$\exists$" in the metalanguage, but there's only so much space!) This completes the proof, modulo checking the truth of the logical equivalences in the display.

The 1st and 2nd $\iff$'s are immediate from the semantics of $\{\exists, \wedge\}$ and of G_C, respectively.

As for the 3rd:

To prove $\Longrightarrow$, take $s := s''_{\vec{y}\mapsto s''(\vec{z}),\vec{z}\mapsto s'(\vec{z})}$. Now s and s' agree away from $\vec{y}$—this is clear on $\vec{z}$ by the 2nd subscript, and away from $\vec{y}$ and $\vec{z}$, they both agree with s''. Also $s(\vec{y}) = s''(\vec{z})$ by that 1st subscript, and $s'(\vec{y}) = s''(\vec{y})$ because s'' agrees with s' except possibly on $\vec{z}$. So (s,s') qualifies as an (s_1, s_1'), and we get $(s,s') \in m(C)$. Finally $|F|^{\vec{y}\to\vec{z}}\ \mathtt{tt}@s''$ implies that $F\ \mathtt{tt}@s$, because $s(\vec{y}) = s''(\vec{z})$.

To prove $\Longleftarrow$, take $s'' := s_{\vec{z}\mapsto s(\vec{y})\ ,\ \vec{y}\mapsto s'(\vec{y})}$. Now s' agrees with s except possibly on $\vec{y}$, because C only involves the variables in $\vec{y}$. So s'' agrees with s' except possibly on $\vec{z}$, as required. Also as required, because $F\ \mathtt{tt}@s$ we get

that $|F|^{\vec{y}\to\vec{z}}$ $\mathtt{tt}@s''$, since $s''(\vec{z}) = s(\vec{y})$. Finally, assume given (s_1, s_1') agreeing away from $\vec{y}$ such that $s_1(\vec{y}) = s''(\vec{z})$ [which equals $s(\vec{y})$], and such that $s_1'(\vec{y}) = s''(\vec{y})$ [which equals $s'(\vec{y})$]. And so (s_1, s_1') agrees on $\vec{y}$ with (s, s'). And so we see that $(s, s') \in m(C)$ implies that $(s_1, s_1') \in m(C)$, the final requirement.

Proof of 22.0(b).

All cases except the last inductive case (of the proof by structural induction on $\mathcal{H}$) are very easy. The last case, when $\mathcal{H}$ is $< C > \mathcal{B}$, can be done readily with the help of **Theorem 5.0.0.**, as follows. (We shall vary the phraseology from the previous part (a)'s proof just to relieve the tedium of this kind of proof!)

Let all the variables in C and the free ones in $\mathcal{B}$ be from among the distinct variables $z_1, \cdots, z_k$. Let $y_1, \cdots, y_k$ be distinct variables, disjoint from the z_i's. Let $\tilde{R}_C$ be (more-or-less the reverse of R_C from the previous part) the relation mapping $(\vec{a}, \vec{b})$ to 1 exactly when input $\vec{a}$ into program C produces output $\vec{b}$. This is semi-decidable, so, by Gödel, let J be a $1^{\underline{\text{st}}}$order formula with free variables from among the y_i and z_i, such that

$$\tilde{R}_C(\vec{a}, \vec{b}) \quad \Longleftrightarrow \quad |J|^{\vec{y}\to\vec{a}\ ,\ \vec{z}\to\vec{b}} \text{ is true in } \mathbf{N} \ .$$

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

Now define H to be $\exists z_1 \cdots \exists z_k \ (J \wedge B)$.
Then, using "$||C||(\vec{a}) = \vec{b}$" as an alternative way of saying "$(\vec{a}, \vec{b}) \in m(C)$",

$$H \text{ is true at } \vec{a} \quad \Longleftrightarrow \quad \text{for some } \vec{b}\ ,\ |J \wedge B|^{\vec{z}\to\vec{b}} \text{ true at } \vec{a} \quad \Longleftrightarrow$$

$$\text{for some } \vec{b}\ ,\ \text{both } |J|^{\vec{y}\to\vec{a}\ ,\ \vec{z}\to\vec{b}} \text{ and } |B|^{\vec{z}\to\vec{b}} \text{ are true in } \mathbf{N} \quad \Longleftrightarrow$$

$$\text{for some } \vec{b}\ ,\ \text{both } \tilde{R}_C(\vec{a}, \vec{b}) \text{ holds and } B \text{ is true at } \vec{b} \quad \Longleftrightarrow$$

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

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

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

Proof of 23.0. Let $\vec{y} = (y_1, \cdots, y_k)$ be a list of distinct variables which includes all of those occurring in any of C, D or G, and let $\vec{z}$ be a matching but disjoint list of distinct variables, and also make sure that w occurs in neither list.

Firstly, we construct $J = J_{C,D}$, a $1^{\underline{\text{st}}}$order formula all of whose variables are from $\{w\} \cup \vec{y} \cup \vec{z}$ such that $\quad J\ \mathsf{tt}@s'' \iff$

$$\text{for all } s_1 \text{ and } s_1', \text{ agreeing away from } \vec{y}, \text{ for which } s_1(\vec{y}) = s''(\vec{y})$$

$$\text{and } s_1'(\vec{y}) = s''(\vec{z}) \ , \text{ we have } (s_1, s_1') \in m(D_{C^{<s''(w)>}}) \ .$$

For this, in **5.0.0** take $\ell = 2k + 1$ and define

$$R = R_{C,D} : \mathbf{N} \times \mathbf{N}^k \times \mathbf{N}^k \ \to \ \{0, 1\} \qquad \text{by} \quad R(n, \vec{a}, \vec{b}) = 1 \iff$$

$$(s, s') \in m(D_{C^{<n>}}) \text{ for all } s \text{ and } s', \text{ agreeing away from } \vec{y},$$

$$\text{and for which } s(\vec{y}) = \vec{a} \text{ and } s'(\vec{y}) = \vec{b} \ .$$

It is easy to see that this relation is semi-decidable—given $(n, \vec{a}, \vec{b})$, one writes out the command $D_{C^{<n>}}$, then runs it on input $\vec{a}$. If $R(n, \vec{a}, \vec{b}) = 1$ is actually true, after finitely many steps in this computation, the machine spits out $\vec{b}$, and the 'semi-deciding' has been done. And so **5.0.0** guarantees us the required J.

Now let $B := \exists \vec{z}(J_{C,D} \wedge |G|^{\vec{y} \to \vec{z}})$. Then to show that this B is the "$B(w, C, D, G)$" which we need:

$$B\ \mathsf{tt}@s \iff$$

$$\exists s'' \text{ agreeing with } s \text{ except on } \vec{z} \text{ satisfying } J\ \mathsf{tt}@s'' \text{ and } |G|^{\vec{y} \to \vec{z}}\ \mathsf{tt}@s''$$

$$\iff [\exists s'' \text{ agreeing as above satisfying } |G|^{\vec{y} \to \vec{z}}\ \mathsf{tt}@s'' \text{ and for all } s_1 \text{ and } s_1',$$

$$\text{agreeing away from } \vec{y}, \text{ and for which } s_1(\vec{y}) = s''(\vec{y}) \text{ and } s_1'(\vec{y}) = s''(\vec{z}) \ ,$$

$$\text{we have } (s_1, s_1') \in m(D_{C^{<s''(w)>}})]$$

$$\iff [\exists s' \text{ with } (s, s') \in m(D_{C^{<s(w)>}}) \text{ and } G\ \mathsf{tt}@s']$$

$$\iff \ < D_{C^{<s(w)>}} > G\ \mathsf{tt}@s \ .$$

(Again, pardon the use of "$\exists$" in the metalanguage.) This completes the proof, modulo checking the truth of the logical equivalences in the display.

The 1st, 2nd, and 4th $\Longleftrightarrow$'s are immediate, respectively, from the semantics of $\{\exists, \wedge\}$, of $J_{C,D}$, and of $< E > \mathcal{A}$.

As for the 3rd:

To prove $\Longrightarrow$, take $s' := s''_{\vec{y} \mapsto s''(\vec{z}), \vec{z} \mapsto s(\vec{z})}$. Then $s'(\vec{y}) = s''(\vec{z})$ by the definition of s', and $s(\vec{y}) = s''(\vec{y})$ because s'' agrees with s except possibly on $\vec{z}$. Also s' and s agree at $\vec{z}$ by the definition of s', and agree away from $\vec{y}$ and $\vec{z}$ because they both agree with s'' there. So s' and s agree away from $\vec{y}$. Thus (s, s') qualifies as an (s_1, s'_1), and we get $(s, s') \in m(D_{C^{<s''(w)>}})$. But $s''(w) = s(w)$ again because s'' agrees with s except possibly on $\vec{z}$. Finally $|G|^{\vec{y} \to \vec{z}}$ $\mathtt{tt}@s''$ implies that G $\mathtt{tt}@s'$, because $s'(\vec{y}) = s''(\vec{z})$.

To prove $\Longleftarrow$, take $s'' := s'_{\vec{z} \mapsto s'(\vec{y}) \ , \ \vec{y} \mapsto s(\vec{y})}$. Now s' agrees with s except possibly on $\vec{y}$, because C and D only involve the variables in $\vec{y}$. So s'' agrees with s except possibly on $\vec{z}$, as required. Also as required, because G $\mathtt{tt}@s'$ we get that $|G|^{\vec{y} \to \vec{z}}$ $\mathtt{tt}@s''$, since $s'(\vec{y}) = s''(\vec{z})$. Finally, assume given (s_1, s'_1) agreeing away from $\vec{y}$ such that $s_1(\vec{y}) = s''(\vec{y})$ [which equals $s(\vec{y})$], and such that $s'_1(\vec{y}) = s''(\vec{z})$ [which equals $s'(\vec{y})$]. And so (s_1, s'_1) agrees on $\vec{y}$ with (s, s'). And so we see that $(s, s') \in m(D_{C^{<s(w)>}})$ implies that $(s_1, s'_1) \in m(D_{C^{<s(w)>}})$. But, using the definition of s'' and then the fact that C and D don't contain the variable w, we get $s''(w) = s'(w) = s(w)$. And so we have the final requirement, that $(s_1, s'_1) \in m(D_{C^{<s''(w)>}})$.

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.

[**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.

[**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.

[**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

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

[**Hoff**] Hoffman, P. *Systems for 'while' Total Correctness with Connectives and Quantifiers.* (to appear).

[**HHH**] Hoffman, P. *Computability for the Mathematical.* (this website).

[**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.

[**mutu**] Hoffman, P. *Imperative Mutual Simple Recursion Proof Systems.* (website).

[**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.

[**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.

[**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.

[**Sok**] Sokolowski, Stefan *Total Correctness for Procedures.* in: [J. Gruska-ed] *Sixth Mathematical Foundations of Computer Science (Tatranská Lomnica)*, pp. 475-483 of Lecture Notes in CS # 53, Springer, 1977.

[**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**] Wang, Arne *An Axiomatic Basis for Proving Total Correctness of GOTO-Programs.* Bit 16(1976), 88-102.