Lucid - A Formal System for Writing and Proving Programs

by

E.A. Ashcroft

W.W. Wadge*

Research Report CS-75-01

Department of Computer Science

University of Waterloo
Waterloo, Ontario, Canada

February 1975

* Computer Science Department
University of Warwick

0. Introduction

Lucid is both a language in which programs can be written, and a formal system for proving properties of such programs. These properties are also expressed in Lucid. This is possible because a Lucid program is itself simply a set of assertions, or axioms, from which other assertions may be derived by conventional mathematical reasoning.

In this paper we present the formal basis for Lucid, giving its semantics and justifying various axioms and rules of inference that are used in Lucid proofs. We assume that the reader is familiar with [1], in which an informal introduction to Lucid can be found, together with a discussion of implementation considerations. Consequently, this paper will be quite formal, with little in the way of motivating explanations and examples. It is intended for those people whose need for detailed information about Lucid has not been satisfied by [1].

Lucid is a formal system similar, in some respects, to first order logic. On the other hand, Lucid can be viewed as a tense logic, a branch of modal logic which formalises certain kinds of reasoning about time. (The suitability of modal logic for proofs about programs has already been recognised by Burstall [2].) In Lucid a term, such as $X > Y$, need not be simply true or false. It can be true at some 'times' and false at others (and even undefined at others). Semantically, the value of $X > Y$ depends on various time parameters, because the values of the variables X and Y themselves depend on time parameters. These time parameters are the numbers of iterations of the various loops in the program within which the variables X and Y are embedded. The interpretations necessary to give such meanings to variables, and corresponding meanings to functions and operations on variables, are essentially Kripke models (see [3]). (However, no knowledge of modal logic is required to understand this paper.)

Lucid also differs from first-order logic in that there is no distinction between terms and formulas, since truth values are treated as data objects. This is necessary to allow programs to compute truth values, and to be able to deal with "undefined" within the logic. This means also that the law of the excluded middle does not hold. Nevertheless, the rules of inference for Lucid are almost identical to those for first-order logic.

Sections 1 to 3 of the paper are devoted to setting up the interpretations mentioned above. Then in Section 4 we define the class of sets of terms that are Lucid programs. We show that every program has a unique minimal solution, or "meaning". In the rest of the paper we discuss a formal system for proving properties of programs, justifying the sort of reasoning used in the proofs given in [1]. In particular, in Section 7 we justify a 'nested proof' technique for proving things about programs with nested loops. In this connection, it is worth mentioning that the basic definition of programs does not allow the use of begin ... end to delimit loops, using instead the function latest (see [1]). The technique in Section 7 allows proofs of programs that use begin ... end, and the proofs need make no use of the function latest.

1. Formalism

1.1 Syntax

A Lucid alphabet Σ is a set containing the symbols "U", "$\exists$" and, for each natural number n, any number of n-ary operation symbols, including, for n=0 the nullary operation symbol T.

We also have at our disposal at set of variables, e.g. X,Y,Z.

The set of Σ-terms is defined as follows:

(a) every variable is a Σ-term;

(b) if G is an n-ary operation symbol in Σ and $A_1,\ldots,A_n$ are Σ-terms then $G(A_1,\ldots,A_n)$ is a Σ-term;

(c) if V is a variable and A is a Σ-term then $\exists V A$ is a Σ-term.

1.2 Semantics

If Σ is an alphabet then a Σ-structure S is a function which assigns to each symbol σ in Σ a "meaning" σ_S in such a way that U_S is a set, $\exists_S$ is a function from subsets of U_S to elements of U_S and, if G is an n-ary operation symbol, G_S is an n-ary operation on U_S.

An S-interpretation I extends S to assign to each variable V an element V_I of U_S.

If A is a Σ-term, S a Σ-structure and I an S-interpretation then we define an element $|A|_I$ of U_S (the "meaning" of A) as follows:

(a) for variable V, $|V|_I$ is V_I

(b) for Σ-terms $A_1,A_2,\ldots,A_n$ and n-ary operation symbol G of Σ,

$$|G(A_1,\ldots,A_n)|_I \text{ is } G_S(|A_1|_I,|A_2|_I,\ldots,|A_n|_I)$$

(c) for Σ-term A and variable V

$$|\exists\ V\ A|_{I} = \exists_S(\{|A|_{I(V/a)} : a \in U_S\})$$

where $I(V/a)$ denotes the S-interpretation differing from I only in that it assigns a to V.

We say: $\models_I A$ (I satisfies A) iff $|A|_I = T_S$; if Γ is a set of terms then $\models_I \Gamma$ iff $\models_I B$ for each B in Γ; $\Gamma \models_S A$ iff $\models_I \Gamma$ implies $\models_I A$ for all S-interpretations I.

2. Basic results

2.1 Substitution

An occurrence of a variable V in a Σ-term A is bound if and only if the occurrence is in a sub-term of A of the form $\exists$ V B, otherwise the occurrence is free. If A and Q are Σ-terms and V is a variable then A(V/Q) is the term formed by replacing all free occurrences of V in A by Q. In this situation V is said to be free for Q in A iff this substitution does not result in a free variable in Q becoming bound in A(V/Q), i.e. iff V does not occur free in A in a subterm of the form $\exists$ W B for some variable W occurring free in Q.

Lemma 1 For Σ-structure S, S-interpretation I, Σ-terms A and Q and variable V, if V is free for Q in A then

$$|A(V/Q)|_I = |A|_{I(V/|Q|_I)}$$

Proof The proof of the analogous result for first-order logic carries over directly. □

2.2 Power structures

For any Σ-structure S and any set X, S^X is the unique Σ-structure C such that

(a) U_C is the set of all functions from X to U_S. If $x \in X$ and $\alpha \in U_C$ we will write α_x instead of $\alpha(x)$.

(b) If G is an operation symbol in Σ and $\alpha, \beta, \ldots \in U_C$ and $x \in X$ then $(G_C(\alpha,\beta,\ldots))_x = G_S(\alpha_x, \beta_x, \ldots)$.

(c) If K is a subset of U_C and $x \in X$ then $(\exists_C(K))_x = \exists_S(\{\alpha_x : \alpha \in K\})$.

Thus S^X carries over the operations and quantifiers of S by making them work 'pointwise' on the elements of U_C. In particular, T_C is the constant function on X with value T_S.

The following lemma establishes that every Σ-term acts "pointwise" in S^X, even those terms containing quantifiers.

Lemma 2 For any Σ-structure S, S^X-interpretation I, Σ-term A and element $x \in X$,

$$(|A|_I)_x = |A|_{I_x}$$

where I_x is the unique S-interpretation which assigns each variable V the value $(V_I)_x$.

Proof Let C be the structure S^X. The proof proceeds by structural induction on A.

(a) If A is a variable the result is immediate.

(b) If A is $G(A_1,\ldots,A_n)$ for n-ary operation symbol G in Σ, and Σ-terms $A_1,\ldots,A_n$, then

$$\begin{aligned}(|A|_I)_x &= (|G(A_1,\ldots,A_n)|_I)_x \\ &= G_S((|B_1|_I)_x,\ldots,(|B_n|_I)_x) \\ &= G_S(|B_1|_{I_x},\ldots,|B_n|_{I_x}) \\ &= |G(B_1,\ldots B_n)|_{I_x}.\end{aligned}$$

(c) If A is $\exists$ V B for some variable V and Σ-term B then

$$(|\exists\, V\, B|_{I})_x = (\exists_C(\{|B|_{I(V/\alpha)} : \alpha \in U_C\}))_x$$

$$= \exists_S(\{(|B|_{I(V/\alpha)})_x : \alpha \in U_C\})$$

$$= \exists_S(\{|B|_{I_x(V/\alpha_x)} : \alpha \in U_C\})$$

(since if $I' = I(V/\alpha)$ then $I'_x = I_x(V/\alpha_x)$)

$$= \exists_S(\{|B|_{I_x(V/a)} : a \in U_S\})$$

(since as α ranges over U_C, α_x ranges over U_S)

$$= |\exists\, V\, B|_{I_x}. \qquad \square$$

It follows that S and S^X have the same theory:

<u>Corollary 2.1</u> For any Σ-structure S, any set X, any Σ-term A and set Γ of Σ-terms,

$$\Gamma \models_S A \text{ iff } \Gamma \models_C A$$

where $C = S^X$.

<u>Proof</u> Suppose first that $\Gamma \models_S A$. Let J be a C-interpretation such that $\models_J \Gamma$. Then for any B in Γ and any x in X, $T_S = (T_C)_x = (|B|_J)_x = |B|_{J_x}$ by Lemma 2. Thus $\models_{J_x} \Gamma$ and so $\models_{J_x} A$; hence $|A|_{J_x} = T_S$. Therefore $(|A|_J)_x = |A|_{J_x} = T_S = (T_C)_x$. Since x was arbitrary, $|A|_J = T_C$ and so $\models_J A$.

How suppose $\Gamma \models_C A$. Let I be an S-interpretation such that $\models_I \Gamma$. Define the C-interpretation J by setting $(V_J)_x = V_I$ for each x in X and each variable V, i.e. $J_x = I$ for each x. Then, for any B in Γ, $(|B|_J)_x = |B|_{J_x} = |B|_I = T_S = (T_C)_x$ for any x in X and so $\models_J \Gamma$. Therefore, $\models_J A$ and, choosing any x in X, $T_S = (|A|_J)_x = |A|_{J_x} = |A|_I$ and so $\models_I A$. $\square$

3. Models of Computation

We define Spec. to be the set of special Lucid function symbols {first, next, as soon as, hitherto, latest, followed by}.

3.1 Standard structures

An alphabet Σ is standard if in addition to T and $\exists$ it contains the nullary operation symbols $\bot$ and F, the unary operation symbol $\neg$, the binary operation symbols $\vee$ and $=$ and the ternary operation symbol if then else, but none of the special Lucid symbols in Spec.

A standard structure is a structure whose alphabet is standard and such that

(a) T_S, F_S and $\bot_S$ are true, false, and undefined respectively.

(b) $\neg_S$ yields true if its argument is false, false if its argument is true, undefined otherwise.

(c) $\vee_S$ yields true if at least one argument is true, false if both are false, undefined otherwise.

(d) $=_S$ yields true if its arguments are identical, false otherwise.

(e) if then else_S yields its second argument if its first is true, its third if its first is false, undefined otherwise.

(f) for any subset K of U_S, $\exists_S(K)$ is true if true $\in$ K, false if K = {false}, undefined otherwise.

(g) all operations of S, except $=_S$, are monotonic, for the ordering on U_S defined by $x \sqsubseteq y$ iff $x = y$ or $x =$ undefined.

Standard structures are our basic domains of data objects and correspond most closely to ordinary first-order structures.

3.2 Computation structures

3.2.1 Comp(S)

If S is a standard Σ-structure, then Comp(S) is the unique ($\Sigma \cup$ Spec)-structure C which extends S^{N^N} * to the larger alphabet as follows:

For $\alpha, \beta \in U_C$ and $\bar{t} = t_0 t_1 t_2 \ldots \in N^N$

i) $(\text{first}_C(\alpha))_{\bar{t}} = \alpha_{0 t_1 t_2 \ldots}$

ii) $(\text{next}_C(\alpha))_{\bar{t}} = \alpha_{t_0+1\ t_1 t_2 \ldots}$

iii) $(\alpha \text{ as soon as}_C \beta)_{\bar{t}} = \alpha_{s t_1 t_2 \ldots}$ if there is a necessarily unique s such that $\beta_{s t_1 t_2 \ldots}$ is true and $\beta_{r t_1 t_2 \ldots}$ is false for all $r < s$, undefined if no such s exists.

iv) $(\text{hitherto}_C(\alpha))_{\bar{t}}$ = true if $\alpha_{s t_1 t_2 \ldots}$ is true for all $s < t_0$,
false if $\alpha_{s t_1 t_2 \ldots}$ is false for some $s < t_0$,
undefined otherwise.

v) $(\text{latest}_C(\alpha))_{\bar{t}} = \alpha_{t_1 t_2 \ldots}$.

vi) $(\alpha \text{ followed by}_C\ \beta)_{0 t_1 t_2} = \alpha_{0 t_1 t_2 \ldots}$
$(\alpha \text{ followed by}_C\ \beta)_{t_0+1\ t_1 t_2 \ldots} = \beta_{t_0 t_1 t_2 \ldots}$

3.2.2 Loop(S)

If Σ and S are as above and Σ' is the alphabet of Comp(S), omitting latest, then Loop(S) is the unique Σ'-structure C' which extends S^N to Σ' in such a way that $\text{first}_{C'}$, $\text{next}_{C'}$, etc. are defined as for Comp(S), but with $t_1 t_2 \ldots$ omitted. For example

* N is the set of natural numbers and N^N is the set of functions from N to N i.e. the set of infinite sequences of natural numbers.

$$(\underset{\sim\sim\sim}{\text{first}}_{C'}(\alpha))_{t_0} = \alpha_0 \text{ and } (\underset{\sim\sim\sim}{\text{next}}_{C'}(\alpha))_{t_0} = \alpha_{t_0+1}.$$

3.2.3 Loop(S) and Comp(S) are used for modelling programs. Loop(S) is simpler, but is adequate only for simple programs without nested loops (i.e. without occurrences of latest). The usefulness of Loop(S) lies in the fact that Loop(S) and Comp(S) have the same theory for terms not involving latest:

Theorem 1 For any standard structure S and any term A and set Γ of terms all in the language of Comp(S),

if A and Γ contain no occurrence of latest then

$\Gamma \models_{\text{Comp}(S)} A$ iff $\Gamma \models_{\text{Loop}(S)} A$

Proof Let C' be the restriction of Comp(S) to the language of Loop(S) and let N^+ be the set of positive integers.

Then it is easily verified that C' is isomorphic to $\text{Loop}(S)^{N^{N^+}}$ and so the result follows from Corollary 2.1. □

3.2.4 Quiescence and constancy

Let $C = \text{Comp}(S)$ and $\alpha \in U_C$. If $\alpha_{\bar{t}}$ is independent of the first element of $\bar{t}$ (i.e. $\alpha_{t_0t_1t_2\ldots} = \alpha_{0t_1t_2\ldots}$ for all t_0) then we say α is quiescent. A term A is quiescent (in C) if $\models_C A =$ first A. Note that for terms A and B, first A, latest A and A as soon as B are all quiescent.

If $\alpha_{\bar{t}}$ is independent of $\bar{t}$, then α is said to be constant. Note that G_C is constant for any nullary operation symbol G.

In Loop(S) we can use the same definitions, but then constancy and quiescence are identical.

4. Programs

4.1 Syntax

A Σ-program P is a set of $(\Sigma \cup \text{Spec})$-terms such that

(a) each element of P is an equation of the form $\phi = \psi$, where ψ is a quantifier-free term having no occurrences of =, and ϕ is of the form X, first X, next X or latest X for some variable X.

(b) The variable input may not occur on the left hand side of any equation in P.

(c) Every other variable X occurring in P, when appearing on the left hand side of an equation, may only do so as part of a definition of X. X must be defined exactly once, in one of the following ways:

directly	i.e. $X = \psi_1$
indirectly	i.e. latest $X = \psi_2$
iteratively	i.e. first $X = \psi_3$
	next $X = \psi_4$.

In the above, the terms ψ_2 and ψ_3 must be syntactically quiescent in P, a property which is defined as follows:

(i) first ϕ, latest ϕ and ϕ as soon as ψ are syntactically quiescent in P.

(ii) if $\phi_1, \phi_2, \ldots, \phi_n$ are syntactically quiescent in P and G is an n-ary operation symbol in Σ, then $G(\phi_1, \phi_2, \ldots, \phi_n)$ is syntactically quiescent in P.

(iii) if $Y = \phi$ is in P and ϕ is syntactically quiescent in P, then Y is syntactically quiescent in P.

4.2 Semantics

The meanings of programs are specified by Comp(S)-interpretations, where S is the standard structure corresponding to the domain of data.

4.2.1 Solutions

For any Σ-program P and standard Σ-structure S, if C = Comp(S) and α is an element of U_C then a (S,α)-solution of P is a C-interpretation I such that $\text{input}_I = \alpha$ and $\models_I$ P.

4.2.2 Theorem 2

For any Σ-program P and standard Σ-structure S, if C = Comp(S) and $\alpha \in U_C$ then there is a (S,α)-solution I of P that is minimal, i.e. for any (S,α)-solution I' of P, for all $\bar{t} \in N^N$ and all variables V in Σ, $(V_I)_{\bar{t}} \sqsubseteq (V_{I'})_{\bar{t}}$.

Proof (sketch)

The first step is to transform P into a set of simple equations. This is done by replacing each pair of equations of the form first X = ϕ, next X = ϕ' by the single simple equation X = ϕ followed by ϕ', and replacing each equation of the form latest X = ϕ by the simple equation X = $\text{latest}^{-1}\phi$. The operation latest_C^{-1} is defined by $(\text{latest}_C^{-1}(\alpha))_{t_0 t_1} = \alpha_{0 t_0 t_1 \cdots}$

This transforms the program P into a 'program' P' of the form $\bar{X} = \tau(\bar{X})$, where $\bar{X}$ is the vector of all the variables in P other than input

We now note that the 'programs' P and P' have the same solutions. That every solution I of the original program P is a solution of the transformed program P' is clear, and the converse follows from the quiescence restrictions on P, as follows.

Suppose that $\models_I X = \phi$ followed by ϕ'. Then by the definition of followed by, $\models_I$ first $X =$ first ϕ and $\models_I$ next $X = \phi'$. But if $X = \phi$ followed by ϕ' in P' came from first $X = \phi$ and next $X = \phi'$ in P, then the syntactic quiescence of ϕ ensures that $\models_I \phi =$ first ϕ, so $\models_I$ first $X = \phi$. Similarly, $\models_I X =$ latest^{-1} ϕ implies $\models_I$ latest $X =$ first ϕ, and so by syntactic quiescence $\models_I$ latest $X = \phi$.

Now we note that the ordering on U_C given in the statement of the theorem makes U_C into a cpo (complete partial order), and it is easily verified that all the operations in C that are used in the 'term' τ are continuous.

Therefore, the transformed program P' has a unique minimal (S,α)-solution, and hence so does P. □

4.3 Syntactic Enrichment

To facilitate the writing of programs we introduce 'nesting' in programs, as a syntactic abbreviation. We say that the expression

```
begin
    $\phi_1$
    $\phi_2$
     :
    $\phi_n$
end
```

is shorthand for the set of terms

$$\begin{array}{l} \phi_1' \\ \phi_2' \\ \vdots \\ \phi_n' \end{array},$$

where ϕ_i' is obtained from ϕ_i by replacing each 'global' variable V by latest V. A global variable is one which occurs within the rest of the program enclosing the original begin ... end expression. The symbols begin and end are used to denote inner loops (not "blocks"), since, within inner loops, global variables become quiescent. Loops can be nested to any depth. Note that for a program using begin ... end to be legal, the result of removing the begin ... end's must be a program.

5. Axioms

5.1 The following abbreviations will be used in the rest of the paper:

$A \wedge B$ means $\neg(\neg A \vee \neg B)$

$A \rightarrow B$ means $\neg(A = T) \vee B$

$\forall V\ A$ means $\neg \exists V \neg A$

5.2 Parentheses will be (and have been) dropped from terms by using the following ranking of priorities for operators (from highest to lowest):

first, next, latest, hitherto, $\neg$, $\wedge$, $\vee$, if then else, as soon as, followed by, $=$, $\rightarrow$.

5.3 Theorem 3 The following are valid in Comp(S) for any standard Σ-structure S, and $(\Sigma \cup \text{Spec})$-terms X, Y and P

(a) $(X = Y) \vee \neg(X = Y)$

(b) $((A = T) = (\neg\neg A = T)) \wedge ((A = F) = (\neg A = T))$

(c) $(\text{first first } X = \text{first } X) \wedge (\text{next first } X = \text{first } X)$

(d) $(\text{first}(X \text{ followed by } Y) = \text{first } X) \wedge (\text{next}(X \text{ followed by } Y) = Y)$

(e) $(\text{first hitherto } P = T) \wedge (\text{next hitherto } P = P \wedge \text{hitherto } P)$

(f) $X \text{ as soon as } P = \text{if first } P \text{ then first } X \text{ else}(\text{next } X \text{ as soon as next } P)$

(g) $X \text{ as soon as } P = (X \text{ as soon as } P \wedge \text{hitherto} \neg P)$

(h) $\text{first}(X \text{ as soon as } P) = X \text{ as soon as } P$

(i) $P \wedge \text{hitherto} \neg P \rightarrow (X \text{ as soon as } P) = X$

(j) $T \text{ as soon as } P \rightarrow (\text{first } X \text{ as soon as } P) = \text{first } X$

(k) $(\text{if } T \text{ then } X \text{ else } Y = X) \wedge (\text{if } F \text{ then } X \text{ else } Y = Y)$

Proof These results (for variables X, Y and P) are easily verified in Loop(S) and carry over to Comp(S) by Theorem 1. The variables can then be replaced by $(\Sigma \cup \text{Spec})$-terms. □

If we define eventually P to be T as soon as P, we have the following corollary.

Corollary 3.1 With S, X and P as above, the following are valid in Comp(S):

(a) $\text{eventually } P \rightarrow \text{first } X \text{ as soon as } P = \text{first } X$

(b) $\text{eventually } P = \text{eventually } (P \wedge \text{hitherto} \neg P)$

(c) $\text{eventually } P = \text{if first } P \text{ then } T \text{ else eventually next } P.$

Proof These follow from the axioms of Theorem 3. □

5.4 The next theorem justifies 'pushing' first and next past quantifiers and non-Lucid operations.

Theorem 4 For any standard Σ-structure S and any Σ-term A in which $X_1, X_2, \ldots, X_k$ are the variables occurring freely:

(a) $\text{first } A = A(X_1/\text{first } X_1, X_2/\text{first } X_2, \ldots)$ is valid in Comp(S),

along with corresponding equations for next and latest.

(b) $\text{eventually } P \rightarrow A \text{ as soon as } P = A(X_1/X_1 \text{ as soon as } P,\ X_2/X_2 \text{ as soon as } P, \ldots$

is valid in Comp(S).

Proof Let I be a Comp(S)-interpretation and let $\bar{t}(= t_0 t_1 t_2 \ldots)$ be any element of N^N. Let $\bar{t}' = 0 t_1 t_2 \ldots$. Then, if $\bar{X}$ denotes $X_1, X_2, \ldots, X_k$,

$$(|\text{first } A|_I)_{\bar{t}} = (|A|_I)_{\bar{t}'}$$

$$= |A|_{I_{\bar{t}'}} \text{ (by Lemma 2) } = |A|_{I_{\bar{t}'}(\bar{X}/(\bar{X}_I)_{\bar{t}'})}$$

$$= |A|_{I_{\bar{t}'}(\bar{X}/(|\text{first } \bar{X}|_I)_{\bar{t}})} = |A|_{I_{\bar{t}}(\bar{X}/(|\text{first } \bar{X}|_I)_{\bar{t}})}$$

(since A has no other free variables)

$$= (|A|_{I(\bar{X}/|\text{first } \bar{X}|_I)})_{\bar{t}} \text{ (by Lemma 2) } = (|A(\bar{X}/\text{first } X)|_I)_{\bar{t}}$$

(by Lemma 1)

Similar reasoning verifies the other results. □

6. Rules of Inference

Lucid cannot be a *complete* formal system because the Lucid functions are powerful enough to characterise unsolvable problems that are not even partially decidable. All we can do is add to Lucid whatever axioms and rules of inference seem natural and useful. In this section we give rules of inference for the logical connectives, and useful rules for the special Lucid functions. The 'logical' rules of inference are those of a simple natural deduction system (see, for example [4]).

6.1 Natural Deduction Rules

6.1.1 Theorem 5 The following rules are valid for standard Σ-structure S, Σ-terms A,B,C,D,P,Q, finite sets Γ and Δ of Σ-terms and variable V, provided V does not occur freely in Γ or D, and is free for P and Q in A:

($\wedge$I)	$A, B \models_S A \wedge B$	($\wedge$E)	$A \wedge B \models_S A$ $A \wedge B \models_S B$
($\vee$I)	$A \models_S A \vee B$ $B \models_S A \vee B$	($\vee$E)	$A \rightarrow C,\ B \rightarrow C, A \vee B \models_S C$
(FI)	$A, \neg A \models_S F$	(FE)	$F \models_S B$
($\rightarrow$I)	if $\Delta, A \models_S B$ then $\Delta \models_S A \rightarrow B$	($\rightarrow$E)	$A \rightarrow B, A \models_S B$
($\forall$I)	if $\Gamma \models_S A$ then $\Gamma \models_S \forall V A$	($\forall$E)	$\forall V A \models_S A(V/Q)$
($\exists$I)	$A(V/Q) \models_S \exists V A$	($\exists$E)	if $\Gamma \models_S A \rightarrow D$ then $\Gamma, \exists V A \models_S D$
(=I)	$\models_S V = V$	(=E)	$A(V/P), P = Q \models_S A(V/Q)$.
(TI)	$A \models_S A = T$	(TE)	$A = T \models_S A$

Proof The validity of the rules can be established by straightforward calculation from the definitions. □

There are no rules for $\neg$ because we do not have the law of the excluded middle: $A \vee \neg A$ is not valid in general, because A may not be truth-valued. This means that some of the tautologies and derived rules of first-order logic are not valid in standard structures. For example $(A \rightarrow B) \rightarrow \neg A \vee B$ is not valid, and if we were to define $A \leftrightarrow B$ to mean $(A \rightarrow B) \wedge (B \rightarrow A)$, then we would not have substitutivity of $\leftrightarrow$ (note, for example, that $\bot \leftrightarrow F$).

6.1.2 Most of the rules of Theorem 5 hold also for Comp(S):

Theorem 6 All the rules of Theorem 5, except $(\rightarrow I)$, are valid for $C =$ Comp(S) in place of S.

Proof Apart from the quantifier rules, and $(\rightarrow I)$, all rules are of the form $\phi \models \psi$ and carry over directly because of the point wise definition of the connectives. We illustrate this for the $(\vee E)$ rule. Consider any C-interpretation I for which $\models_I A \rightarrow C$, $\models_I B \rightarrow C$ and $\models_I A \vee B$. Then, for all $\bar{t} \in N^N$, $(|A \rightarrow C|_I)_{\bar{t}}$, $(|B \rightarrow C|_I)_{\bar{t}}$ and $(|A \vee B|_I)_{\bar{t}}$ are all true. By definition of C, we then have $(|A|_I)_{\bar{t}} \rightarrow_S (|C|_I)_{\bar{t}}$, $(|A|_I)_{\bar{t}} \rightarrow_S (|C|_I)_{\bar{t}}$ and $(|A|_I)_{\bar{t}} \vee_S (|B|_I)_{\bar{t}}$ are all true. By the $(\vee E)$ rule for S (Theorem 1) we then have $(|C|_I)_{\bar{t}} =$ true. This holds for all $\bar{t} \in N^N$, so $\models_I C$.

We illustrate the proof for the quantifier rules by considering $(\forall E)$ and $(\exists E)$.

$(\forall E)$: Let I be any C-interpretation for which $|\forall V A|_I = T_I$. Then for all $\bar{t} \in N^N$

$$\underline{\text{true}} = (|\forall V A|_I)_{\bar{t}}$$

$$= \forall_S\{(|A|_{I(V/\alpha)})_{\bar{t}} : \alpha \in U_C\}.$$

Therefore for all $\bar{t} \in N^N$ and all $\alpha \in U_C$ we have $|A|_{I(V/\alpha)} =$ true. Now $|A|_{I(V/|Q|_I)} = |A(V/Q)|_I$, by Lemma 1, and so, for all $\bar{t} \in N^N$, $|A(V/Q)|_I =$ true, that is $\models_I A(V/Q)$.

$(\exists E)$: Assume $\Gamma \models_C A \rightarrow D$ and consider any C-interpretation I for which $\models_I B$, for all $B \in \Gamma$, and $\models_I \forall V A$. Consider any $\bar{t} \in N^N$. By the definition of $\exists_C$, there is some $\alpha \in U_C$ such that $(|A|_{I(V/\alpha)})_{\bar{t}} =$ true. Now $I(V/\alpha)$ is a C-interpretation and $\models_{I(V/\alpha)} \Gamma$, since V is not free in Γ. Thus $\models_{I(V/\alpha)} A \rightarrow D$, and so $(|D|_{I(V/\alpha)})_{\bar{t}} =$ true. Since V is not free in D, we then have $(|D|_I)_{\bar{t}}$. We chose $\bar{t}$ arbitrarily, so $\models_I D$. □

6.2 Lucid Rules

6.2.1 One of the most important rules is that a standard Σ-structure S and Comp(S) have the same theory, when restricted to Σ-terms, so any "elementary" property can be used directly in any proof about a program.

Theorem 7 For any standard Σ-structure S, any Σ-term A and any set Γ of Σ-terms $\Gamma \models_S A$ iff $\Gamma \models_C A$, where $C = \mathrm{Comp}(S)$.

Proof Since Γ and A are in the language of S and since Comp(S) is an extension of S^{N^N} the result follows immediately from Corollary 2.1. □

6.2.2 Other Lucid rules including induction and termination are given by the following theorem.

Theorem 8 For any standard Σ-structure S, if $C = \mathrm{Comp}(S)$ then

(a) $P \models_C$ first P and $P \models_C$ next P

(b) first P, $P \rightarrow$ next $P \models_C P$

(c) $Q =$ first Q, $P \rightarrow Q$, eventually $P \models_C Q$

(d) $P \rightarrow F \models_C X$ as soon as $P = \bot$

(e) integer Y, $Y >$ next $Y \models_C$ eventually $(Y \leq 0)$

where in (e) we assume S includes the integers.

Proof Immediate. □

6.3 Recovering the Deduction Theorem

We have seen that the ($\rightarrow$I) rule is not valid in Comp(S). However, we can recover this rule, at the expense of weakening the (=E) rule, by a form of reasoning which intuitively corresponds to confining oneself to a particular moment during the execution of a program.

6.3.1 Definition of $|\approx$

If S is a Σ-structure, C = Comp(S) and A is a term and Γ a set of terms on the alphabet of C, then we define $\Gamma \ |\approx_C$ A to mean that for any C-interpretation I, if $(|B|_I)_{\bar{t}}$ = true for every B in Γ, then $(|A|_I)_{\bar{t}}$ = true.

Thus $\Gamma \ |\approx_C$ A means that, at any time, if all the terms in Γ are true, then A is true. It is immediate that $\models$ A implies $|\approx$ A, and that $\Gamma \ |\approx_C$ A implies $\Gamma \models$ A but not vice versa, e.g. P $\models_C$ next P but not P $|\approx_C$ next P.

6.3.2 Theorem 9 For any standard Σ-structure S, if C = Comp(S)

(a) every rule of Theorem 6 except the (=E) rule remains valid if $\models$ is replaced by $|\approx$.

(b) for Γ,A,B as in Theorem 6 if Γ,A $|\approx_C$ B then $\Gamma \ |\approx_C$ A $\to$ B

(c) for A,P,Q as in Theorem 6, if A contains no Lucid functions then A(V/P), P = Q $|\approx_C$ A(V/Q).

Proof Let Γ,A,B and V be as in Theorem 6.

(a) We will illustrate the proof by considering the ($\exists$E) rule. Assume $\Gamma \ |\approx_C$ A $\to$ B and $\Gamma \ |\approx_C \exists$ V A. Let $\bar{t} \in N^N$ and let I be a C-interpretation such that $(|D|_I)_{\bar{t}}$ for every D in Γ. Then by the second assumption $(|\exists \text{ V A}|_I)_{\bar{t}}$ = true and so $(|A|_{I(V/\alpha)})_{\bar{t}}$ = true for some α in U_C by the definition of $\exists_C$. Since V does not occur in any D in Γ, $(|D|_{I(V/\alpha)})_{\bar{t}} = (|D|_I)_{\bar{t}}$ = true for any such D, and so by the first assumption $(|B|_I)_{\bar{t}}$ = true. Therefore $\Gamma \ |\approx_C$ B.

(b) Let $\bar{t} \in N^N$ and suppose that every C-interpretation which makes A and everything in Γ true at $\bar{t}$ also makes B true at $\bar{t}$. Suppose now that C-interpretation I makes everything in Γ true at $\bar{t}$. If I makes A true at $\bar{t}$ then it must make B true at $\bar{t}$ and so makes

$A \rightarrow B$ true at $\bar{t}$. On the other hand, if I makes A other than true at $\bar{t}$ then $A \rightarrow B$ will be true at $\bar{t}$ regardless of the value I assigns B at $\bar{t}$. In either case $A \rightarrow B$ is true at $\bar{t}$ and so $\Gamma \mid\approx_C A \rightarrow B$.

(c) Suppose that $(|A(V/P)|_I)_{\bar{t}} = \underline{\text{true}}$ and $(|P{=}Q|_I)_{\bar{t}} = \underline{\text{true}}$. Now $|A(V/P)|_I = |A|_{I(V/|P|_I)}$ by Lemma 1 and $(|A|_{I(V/|P|_I)})_{\bar{t}} = |A|_{I_{\bar{t}}(V/(|P|_I)_{\bar{t}})}$ since A contains no Lucid functions. But $(|P{=}Q|_I)_{\bar{t}}$ implies $(|P|_I)_{\bar{t}} = (|Q|_I)_{\bar{t}}$. Thus

$$\begin{aligned}|A|_{I_{\bar{t}}(V/(|P|_I)_{\bar{t}})} &= |A|_{I_{\bar{t}}(V/(|Q|)_I)_{\bar{t}}} \\ &= (|A|_{I(V/|Q|_I)})_{\bar{t}} \\ &= (|A(V/Q)|_I)_{\bar{t}}\ .\end{aligned}$$

Therefore $(|A(V/Q)|_I)_{\bar{t}} = \underline{\text{true}}$. □

We call the rule in Theorem 9(c) the (weak =E) rule. To illustrate that (=E) does not work for $|\approx$, note that next P, P = Q $|\approx$ next Q is not valid (informally, if P equals Q at some time when P will be true at the next step, it does not necessarily follow that Q will be true at the next step).

6.3.3 There is another way in which we can regain the deduction theorem. If we are reasoning about a simple loop, and the assumption A is constant, i.e. A = first A, then the assumption can be cancelled:

Theorem 10 For any Σ-structure S, if C = Comp(S) and A and B are terms and Γ a set of terms on the alphabet of C omitting latest, then

$$\Gamma,\ \text{first } A \models_C B \text{ implies } \Gamma \models_C \text{first } A \rightarrow B$$

Proof The theorem holds for Loop(S) in place of Comp(S), because if first A is ever true it is always true. The result carries over to Comp(S) by Theorem 1. □

7. Proofs within Loops

The structuring of programs that is made possible by the use of begin and end also allows "structured proofs". We will show that

(a) Within a begin .. end loop, all the rules of inference are valid and so is the assumption that $X = \mathrm{first}\ X$ for every global variable X. Anything that can be proved by introducing latest can also be proved without latest, in this fashion.

(b) Any assertion that does not use Lucid functions can be moved into and out of begin .. end loops.

Theorem 11 For any standard Σ-structure S, if $C = \mathrm{Comp}(S)$, then for any term A and set of terms Γ on the alphabet of C, and any finite set of variables $\bar{X}$,

(a) $\bar{X} = \mathrm{first}\ \bar{X},\ \Gamma \models_C A$ iff

$$\Gamma(\bar{X}/\mathrm{latest}\ \bar{X}) \models_C A(\bar{X}/\mathrm{latest}\ \bar{X}).$$

(b) If A is a Σ-term and $\bar{X}$ is the set of variables occurring freely in A, then

$\Gamma \models_C A$ iff $\Gamma \models_C A(\bar{X}/\mathrm{latest}\ X)$.

Proof (a) Assume $\bar{X} = \mathrm{first}\ \bar{X},\ \Gamma \models_C A$, and that, for C-interpretation I, $\models_I \Gamma(\bar{X}/\mathrm{latest}\ X)$. Let $\bar{\alpha}$ be $|\mathrm{latest}\ \bar{X}|_I$, and $I' = I(\bar{X}/\bar{\alpha})$. Then $\models_{I'} \bar{X} = \mathrm{first}\ \bar{X}$ and $\models_{I'} \Gamma$, therefore $\models_{I'} A$, and so $\models_I A(\bar{X}/\mathrm{latest}\ \bar{X})$.

Conversely, assume that $\Gamma(\bar{X}/\mathrm{latest}\ \bar{X}) \models_C A(\bar{X}/\mathrm{latest}\ \bar{X})$ and that for C-interpretation I, $\models_I \bar{X} = \mathrm{first}\ \bar{X}$ and $\models_I \Gamma$. Then $|\bar{X}|_I$ is $\mathrm{latest}_C\ \bar{\alpha}$ for some $\bar{\alpha}$ in U_C.* Let I' be $I(\bar{X}/\bar{\alpha})$. Then $\models_{I'} \Gamma(\bar{X}/\mathrm{latest}\ X)$, and so $\models_{I'} A(\bar{X}/\mathrm{latest}\ \bar{X})$. Hence $\models_I A$.

(b) Let I be a C-interpretation such that $\models_I \Gamma$. Then since $(|\mathrm{latest}\ A|_I)_{t_0 t_1 t_2 \ldots} = (|A|_I)_{t_1 t_2 \ldots}$, $(|\mathrm{latest}\ A|_I)_{\bar{t}} = \mathrm{true}$ for all $\bar{t}$ iff $(|A|_I)_{\bar{t}} = \mathrm{true}$ for all $\bar{t}$. Then since $\models_I \mathrm{latest}\ A = A(\bar{X}/\mathrm{latest}\ \bar{X})$ by Theorem 4(a), the result follows. □

* In fact $\bar{\alpha} = |\mathrm{latest}^{-1}\ \bar{X}|_I$ (see the proof of Theorem 2, Section 4.2.2).

8. References

[1] E.A. Ashcroft and W.W. Wadge, "Demystifying Program Proving", Technical Report CS-75-02. Computer Science Department, University of Waterloo.

[2] R. Burstall, "Program Proving as Hand Simulation with a Little Induction", Proceedings IFIP Congress 1974, Stockholm.

[3] G.E. Hughes and M.J. Cresswell, "An Introduction to Modal Logic", Methuen (1968).

[4] Z. Manna, "Introduction to Mathematical Theory of Computation", McGraw Hill, New York, 1974.