Logical and mathemtical reasoning about imperative programs.[1]

Preliminary Report

Daniel Leivant
Carnegie-Mellon University

1. A framework for reasoning about programs.

1.1. *Logical vs. mathematical reasoning.*

There have been two traditions of formalizing reasoning about programs and computations, one arithmetical in nature, the other logical. In this chapter we propose Second-Order Logic, with controlled forms of comprehension, as a framework in which both approaches can be suitably represented, analyzed and compared. We also maintain that it is by itself a highly suitable framework for reasoning about programs, computations and data types, and as a tool for deriving metamathematical results.

It is well known that reasoning about computation can be carried out within Peano Arithmetic [Kle52]. Although this was demonstrated originally for Turing Machine computations, the method adapts easily to reasoning about computations of imperative programs (see e.g. [Zuc80] and [BT81]). The cumbersome numeric coding, with which such reasoning frameworks are plagued, is necessary only when one confines oneself to the meager language of first-order arithmetic: when the language is enriched, to allow direct reference to programming constructs, one obtains a reasonable framework for reasoning about imperative programs. A formalism of this type might be dubbed "Arithmetic of Programs". In a sense, the "Non-Standard Dynamic Logic" of Nemeti & als. [Nem82, ANS82], is such an arithmetic.

Logics of programs, in the styles of Floyd, Hoare, Pratt and others, represent an altogether different approach to reasoning about programs. Each one of these styles is based on a *modal* language for reasoning about programs, with no extraneous reference to numbers or to time, and on a mostly *syntax driven* axiomatics. Both choice of language and choice of axiomatics reflect an attempt to capture the analytic, *a priori*, meaning of programs, thereby justifying these formalisms as "logical", as opposed to "mathematical".

Without venturing a general differentiation between Logic and Mathematics, it seems sound to agree that the infinite is not analytic, and that reasoning about the infinite is, therefore, a mathematical rather than a logical enterprise (cf. [Wan74]). (In Russell's Logicism, however, no such distinction is admitted [Rus20,38; HA38].) Recall, for example, that syntax-driven reasoning is impossible in First-Order Arithmetic: for every $k>0$ there is a Π^0_1 theorem of Peano Arithmetic with no proof where induction is used only over formulas with less than k quantifiers. It is then clear that some of the reasoning about programs, namely that part which deals with iterative constructs such as looping and recursion, is really not logical, and can not be syntax driven. Thus, reasoning about programs factorizes into reasoning about program compositionality, for whose logical character a tenable case can be made, and into a typically mathematical component of reasoning about iterative aspects of program execution.

In Mathematical Logic the term "logic" has been used for formalisms which do involve the infinite in implicit and even explicit ways, such as $L_{\omega_1\omega}$, and ω-logic. The kinship of these two examples to Dynamic Logic has been extensively explored, and to the extent that these formalisms are "logics", so are all logics of programs without further ado. However, the term "logic" is applied to formalisms of this kind only by extension: they were designed neither as feasible carriers of reasoning, nor as conveyers of epistemological distinctions, but merely as technical tools (to calibrate expressiveness, to characterize hierarchies, for proof-theoretic analysis, and so on). The traditional aims of Logic cannot, however, be overlooked for logics of programs. On the one hand, no logic of programs has yet gained wide

©1984 ACM 0-89791-147-4/85/001/0132 $00.75

1. An early announcement of this research bore the title ***The logical completeness of Hoare-style logics***

acceptance as a feasible carrier of reasoning about programs, while the need for such a carrier is dire. On the other hand, we feel that fundamental meta-logical concepts, such as completeness and soundness, have not yet been identified for logics of programs in a satisfactory manner, and that identifying the correct concepts depends on a precise delineation of the "logical" component of reasoning about programs.

In summary, we wish to identify a formalism F satisfying the following conditions. The first three criteria are needed to delineate the purely logical portion of logics of programs:

1. Traditional logics of programs have natural interpretions in F.
2. There is a natural fragment F_L of F which captures the "logical" portion of F.
3. The formulas (and terms) of F which are essential for reasoning about programs are present in F_L.

The following additional conditions are desirable for F to be useful as a formalism for reasoning about programs of independent interest.

4. Reasoning about programs can be carried out in F in a straightforward manner.
5. F is proof theoretically strong.
6. F is potentially suitable for expressing and reasoning about less elementary programming language concepts, such as inductively defined data types, recursive procedures, procedures as parameters, concurrency and polymorphism.

1.2. Second-Order Logic and Weak Second-Order Logic.

We now identify a solution to conditions (1—6).

Condition (2) implies, by Lindstrom's Theorem [Lin69,EFT84] (see also [Wan74, chapter 4]), that F_L must be a variant of First-Order Logic. At the same time, (3) implies that the input-output relations generated by while-programs, for instance, should be expressible in the language of F_L. The input-output relation generated by a programming language can be explicitly defined using natural numbers as an auxiliary notion (e.g. [Zuc80]). It can also defined, without any auxiliary objects, using quantification over relations, as in §2.1 below. This suggests the choice of Second-Order Logic as the formalism F. The "logical" portion F_L of F can then be taken as the restriction of F to first-order comprehension, in the sense specified momentarily.

First let us recall in a nutshell the essentials of Second-Order Logic. Given a first order language, say the language of Pure First-Order Predicate Logic, the corresponding *second-order language* is obtained by using quantifiers over predicate letters as additional constructs. The intended, *standard semantics*, is that quantifiers over k-ary predicate letters range over all k-ary relations over the domain of (ground) elements. Among these relations are the ones definable by formulas of the language, so the *Comprehension Schema*,

$$\mathrm{Comp}_\varphi:\ \exists R \forall \bar{x}\, (R(\bar{x}) \leftrightarrow \varphi),$$

is valid under the standard semantics, where φ is any formula of the language.

In this language equality and natural numbers are definable:

$$x = y \ \equiv\ \forall R (R(x) \leftrightarrow R(y))$$

and

$$N(x)\ \equiv\ \forall R (\forall z (R(z) \rightarrow R(S(z))) \rightarrow R(0) \rightarrow R(x)).$$

Similarly, the graph of any primitive recursive function is explicitly definable. Hence First-Order Arithmetic, in fact Second-Order Arithmetic (= Mathematical Analysis), is interpretable in Second-Order Logic. It follows that Second-Order Logic is in fact a mathematical system, and is not a genuine logic in the sense of §1.1. It also follows that there is no formal calculus for Second-Order Logic, since the set of valid second-order formulas is not definable in (second-order) arithmetic, let alone effectively enumerable.

Nonetheless, a natural calculus L_2 for Second-Order Logic is obtained by extending the quantifier axioms and rules of First-Order Predicate Logic to second-order quantifiers, and by adding the Comprehension Schema as an axiom scheme [HA38]. Equivalent calculi can be defined using Gentzen's natural deduction style [Pra65], or Gentzen's sequential calculus style (e.g. [Tak75]). In the natural deduction style Comprehension becomes implicit in the rules for second-order universal quantifier elimination and for second-order existential quantifier introduction:

$$\frac{\forall R \varphi}{\varphi[\psi / R]} \qquad\qquad \frac{\varphi[\psi / R]}{\exists R \varphi}$$

Here R is a k-ary predicate variable, and the substitution $\varphi[\psi / R]$ of (the k-ary relation canonically defined by) the formula ψ for R in φ is defined in a straightforward manner.

This formalism can, however , be interpreted as a many-sorted first-order theory [KK67]. The sorts are ι, for individuals, and $\tilde{k}$, for k-ary relations ($k \geq 0$). Using, for each $k \geq 0$, a predicate constant I_k of sort $(\tilde{k}, \iota, \ldots, \iota)$, an atomic formula $R(t_1, \ldots, t_k)$ of Second-Order Logic can be rendered by $I_k(R, t_1, \ldots, t_k)$, quantification over predicates becomes first-order, and Comprehension turns into a first-order axiom schema.

From the Completeness Theorem for First-Order Logic it then follows that the theory is complete for its models, which are simply models of many-sorted first-order logic in which all instances of the

Comprehension Schema are valid. In such a model each object r of sort $\tilde{k}$ represents the k-ary relation $R(\vec{x}) \equiv I_k(r,\vec{x})$ over the domain M of individuals, but these relations constitute any collection closed under second-order definability, not necessarily the entire power set $P(M^k)$. Thus, L_2 is complete for validity in models of the language of Second-Order Logic, with the collection of relations explicitly specified, subject to the condition that it be closed under second-order definability [Hen50, End70]. These models are referred to as *Henkin models*.

More generally, one may consider subsystems of L_2, where the Comprehension Schema is restricted to formulas in a given class C of formulas. The only requirement on Henkin models of such theories is then that they be closed under definability by formulas in C. Of particular interest is the subsystem L_{21} of L_2 of *first-order comprehension*, that is, where $Comp_\varphi$ is postulated only for first-order formulas φ. We dub L_{21} *Weak Second-Order Logic*[2].

Our feeling is that L_{21} is in fact a variant of First-Order Logic, and should qualify as a genuine logic (whereas L_2 is a first-order *mathematical theory*). Shelving for now the epistemological evidence for this claim, we mention three technical facts which support it. Firstly, L_{21} is *schematically conservative* over First-Order Logic, in the following sense: If a first-order theory T is axiomatized by a collection Γ of schemas $\gamma[\vec{\varphi}]$, where the $\vec{\varphi}$'s range over all (first-order) formulas in the language of T, then no new theorems in that language will arise if the logic is expanded to L_{21}, and where schemas are interpreted eith $\vec{\varphi}$ ranging over all second-order formulas in the language of T). For example, Peano Arithmetic is axiomatized using the schema of Induction; no new first-order theorems arise from reasoning in L_{21}, with induction over all second-order formulas in the language of arithmetic. This is to be contrasted with L_2: using the Induction Schema over all second-order formulas, with unrestricted Comprehension, permits the derivation of many first-order formulas which are not theorems of Peano Arithmetic (for example, a formula rendering the consistency of Peano Arithmetic).

Secondly, L_{21} is syntax oriented: if a formula φ is provable in L_{21}, then it has a proof in which all formulas are *internal first-order instances* of φ, where the set C of these formulas is defined inductively by: all subformulas of φ are in C; if $QR\psi$ is in C ($Q=\forall$ or $\exists$), and $\chi \in C$ is first-order, then $\psi[\chi/R] \in C$. Note that C may be characterized as the set of term-substitution instances of a certain finite set of formulas, generalizing only slightly the textual subformulas of φ.

Finally, the proof theory of L_{21} is essentially the same as that of First-Order Logic. Proofs of such meta-logical properties as normalization of natural deductions and cut elimination for sequential proofs have the same complexity as for first-order logic.

In summary, we are interested in Second-Order Logic with its standard semantics in L_2, and in L_{21}. We shall write $\models^S$, $\models^{L2}$ and $\models^{L21}$ for semantic entailment in these three formalisms, respectively.

1.3. The relevance of Second-Order Logic and weak comprehension to Computer Science.

The second-order definability of the natural numbers, mentioned above, is one example of the definability of all inductively defined sets. Among such sets are the inductively defined data-types, such as lists and trees. Induction for objects of the data type is an immediate consequence of the Comprehension Schema to the type's definition.

Similarly, fixpoint operations in denotational semantics of programming languages are akin to inductive definitions. The use of restricted comprehension is also relevant to denotational semantics: one considers the space of *continuous* functions, for example, rather than the space of *all functions* over given domains.

On a more specific and technical level, the definability of inductively defined data types can be combined with second-order reasoning about purely functional programs, yielding a highly economical and streamlined calculus for reasoning about functional computations over such types [Lei83]. From this there follow connections between second-order (natural-deduction) proofs on the one hand and the Lambda Calculus, polymorphic types, and complexity classes on the other hand. The relation between polymorphism and second-order logic is clearly fundamental [Gir72, Rey74, FLO83, MPS83, Lei84].

Finally, we note the increasing prominence of comprehension principles in current studies in the Foundation of Analysis. Various principles and classical theorems have been identified as equivalent (modulo weak proof theoretic means) to particular forms of comprehension [FSS83]. Restricted forms of comprehension are therefore a natural and powerful tool in calibrating the logical strength of theorems of Mathematical Analysis.

1.4. Classification of Second-Order formulas.

Formulas in the language of Second-Order Arithmetic can be brought to a prenex normal form, by the Tarski-Kuratowski-Kleene procedure [Rog67]. A prenex normal form formula has a block of alternating second-order quantifiers, followed by a first-order formula. The crucial step of the TKK procedure is Kleene's observation that second-order quantifiers permute (in an appropriate sense) with first-order quantifiers:

$$\forall x\, \exists R^k \varphi[R] \leftrightarrow \exists Q^{k+1} \forall x \varphi[Q_x],$$

where $Q_x(t_1, \cdots, t_k) \equiv Q(x, t_1, \cdots, t_k)$. The implication from left

2. This use of the term should be distinguished from other uses, such as for logic with second-order quantifiers ranging over finite sets.

to right is intuitively valid by the Axiom of Choice, but is not a theorem of L_2.

Though second-order formulas can not be easily classified in L_{21}, two classes of formulas stand out as particularly relevant to our discussions below. We say that a second order formula φ is Π_1^R if no second-order universal quantifier occurs negatively in φ, and no second-order existential quantifier occurs positively. The definition of Σ_1^R formulas is dual. When insisting on the prenex normal form of a Π_1^R formula we shall use the phrase "pure Π_1^R."

Proposition 1. Let φ be a Π_1^R formula.

$$\models^S \varphi \quad \text{iff} \quad \models^{L1} \varphi \quad \text{iff} \quad \models^{L12} \varphi.$$

Proof. The implications from right to left are trivial, since a standard second-order model is a Henkin model of L_1, and a Henkin model of L_1 is a model of L_{21}.

Suppose $\models^S \varphi$, and consider a Henkin model <M,C> of L_1, where M is the ground domain and C the collection of relations. φ is true in the standard model over M, and restricting positive occurrences of universal quantifiers or negative occurrences of existential quantifiers (to C) weakens a formula. So φ is true in <M,C>.

The second implication from left to right is proved similarly. ■

2. Direct reasoning about imperative programs.

2.1. Explicit definition of the operational semantics of programming constructs.

We show that the input-output relation determined by programming constructs is definable by Π_1^R formulas. The two key ideas are, firstly, that the set of initial traces of a computation is inductively definable; secondly, that the "principal" store used by a program is finite. The latter point may be controversial when it comes to recursive procedures, for example. Our point is that such computations proceed through an unbounded number of auxiliary stores, each finite, which have only a paranthetical effect on the computation. In concrete implementations, garbage collection is the closing paranthesis. In explicit definition, as below, the paranthesis are the scope of (first-order) universal quantifiers.

We assume, for simplicity and without loss of generality, that programs are over a single type of individual objects. As mentioned, we also assume that the input-output behavior of a program α involves only a finite number of store locations; and that, consequently, it constitute a $2k$-ary relation R_α over ground objects. It is customary to assume that the store locations affected by a program are the variables active (=assigned to) in the program, but our assumption is consistent with the existence of side effects. Also, programming constructs such as recursive procedures may affect an unbounded portion of the store, but for a terminating calculation all store locations activated by recursive calls would be treated as auxiliary store, and would be subjected to garbage collection without effect on subsequent calculation (at least not when the model of execution is idealized).

We describe the input-output behavior of a program without explicit reference to the store. To achieve this we refer to the store indirectly through a fixed enumeration (without repetition) of all program variables, $x_1, \cdots$. If a program α affects only variables among $\langle x_{i_1} \cdots x_{i_k} \rangle$, where $i_1 < \cdots < i_k$, then we describe the input-output behavior of α by a relation M_α over $2 * i_k$ (logical) variables, of which i_k variables denote the initial values of all program variables $x_1 \cdots x_{i_k}$, and the remaining i_k (logical) variables denote the terminal values. More generally, we shall need formulas M_α^n, describing the effect of α on the store $\langle x_1 \cdots x_n \rangle$, for any $n \geq i_k$.

We define below the input-output relation generated by certain programming constructs. We do include recursive procedures, but not such constructs as call-by-name, procedures as parameters, typing (i.e. abortion on type errors) or type as arguments (as in RUSSELL). These will be treated elsewhere. In the following definitions we assume throughout that the program variables (active) in the program α have all index $\leq n$, and that $\bar{u}$ and $\bar{v}$ are n-tuples of variables, all distinct.

Elementary constructs:

- If $\alpha \equiv x_i \leftarrow t$, then

$$M_\alpha^n[\bar{u},\bar{v}] \equiv$$
$$\forall R^n\, [R(u_1, \cdots, u_{i-1}, t[\bar{u}/\bar{x}], u_{i+1}, \cdots u_n) \rightarrow R(\bar{v})].$$

This is of course the same as $v_i = t[\bar{u}/\bar{x}]$, $v_j = u_j$ $(j \neq i)$.

- If $\alpha \equiv \beta;\gamma$, then

$$M_\alpha^n[\bar{u},\bar{v}] \equiv \forall R^n\, [(\forall \bar{z}(M_\beta^n(\bar{u},\bar{z}) \rightarrow R(\bar{z})) \rightarrow \exists \bar{w}(R(\bar{w}) \wedge M_\gamma^n(\bar{w},\bar{v}))].$$

- If $\alpha \equiv$ test φ, then

$$M_\alpha^n[\bar{u},\bar{v}] \equiv \forall R^n[R(\bar{u}) \wedge \varphi[\bar{u}/\bar{x}] \rightarrow R(\bar{v})].$$

- If $\alpha \equiv$ if φ then β else γ, then

$$M_\alpha^n[\bar{u},\bar{v}] \equiv$$
$$\varphi[\bar{u}/\bar{x}] \rightarrow M_\beta^n[\bar{u},\bar{v}] \,.\wedge.\, \neg\varphi[\bar{u}/\bar{x}] \rightarrow M_\gamma^n[\bar{u},\bar{v}].$$

- If $\alpha \equiv \beta \cup \gamma$ (non-deterministic choice), then

$$M_\alpha^n[\bar{u},\bar{v}] \equiv M_\beta^n[\bar{u},\bar{v}] \vee M_\gamma^n[\bar{u},\bar{v}].$$

- If $\alpha \equiv x_i \leftarrow ?$, (non-deterministic assignment) then

$$M^n_\alpha[\bar{u},\bar{v}] \equiv \exists z \forall R^n\, [R(u_1,\cdots,u_{i-1},z,u_{i+1},\cdots u_n) \rightarrow (R(\bar{v})].$$

Looping:

- If $\alpha \equiv$ while φ do β, then

$$M^n_\alpha[\bar{u},\bar{v}] \equiv \neg\varphi[\bar{v}/\bar{x}] \wedge \forall R^n\, [R(\bar{u}) \wedge Prog_{\beta\varphi}[R] \rightarrow R(\bar{v})],$$

where

$$Prog_{\beta\varphi}[R] \equiv \forall \bar{z},\bar{w}[R(\bar{z}) \wedge \varphi[\bar{z}/\bar{x}] \wedge M^n_\beta[\bar{z},\bar{w}] \rightarrow R(\bar{w})].$$

Procedures:

The point of our including procedure is to support our claim that the operational semantics of programs can be defined using a finite number of store locations. Although sketchy, we hope that it will convey the idea.

A collection of procedure definitions

$$Q_j(\bar{x}_j) \leftarrow \beta_j,\ j \in J.$$

form an environment, with the usual scope rules. The variables *affected* in the environment are the ones global in some of the procedure bodies. Suppose n is the last index of an affected variable. The *proper store* of a procedure Q_j is the set of procedure arguments (= formal parameters). W.l.o.g. the latter are consecutive variables, following the last affected variable of the environment. Let $r(j)$ be the number of actual parameters of Q_j. Let P_j $(j \in J)$ be predicate letters, with arity $n + r(j)$, respectively.

Suppose α is in the scope of the environment above. The procedure definitions may be (mutually) recursive, and α may use global procedure identifiers, i.e. ones for which no definition is given.

- Suppose $\alpha \equiv$ call $Q_{j0}(\bar{t})$ (call by value). The variables affected by α are the same as the ones affected in the environment, plus the proper store of β_{j0} (if $j0 \in J$). Below, $\bar{u}$ and $\bar{v}$ denote the initial and final values of the variables affected, respectively. $\bar{z}$ and $\bar{w}$ denote the initial and final values in the proper store (the initial values are irrelevant).

Define

$$M^n_\alpha[\bar{u}\ \bar{z}\ ,\bar{v}\ \bar{w}] \equiv (\forall P_j)_{j \in J}\, [\ \wedge_{j \in J}\{\forall \bar{p},\bar{q}(P_j(\bar{p}) \wedge M_{\beta_j}[\bar{p},\bar{q}] \rightarrow P_j(\bar{q})\} \rightarrow (P_{j0}(\bar{u}\ \bar{t}[\bar{u}/\bar{x}]\,) \rightarrow P_{j0}(\bar{v}\ \bar{w}))].$$

Note that when Q_{j0} is a global procedure identifier, the definition boils down to

$$P_{j0}(\bar{u}\ \bar{t}[\bar{u}/\bar{x}]\,) \rightarrow P_{j0}(\bar{v}\ \bar{w}),$$

where P_{j0} is a free predicate variable.

We write M_α for M^n_α where n is such that M^n_α is defined, i.e. if all variables affected by α have index $\leq n$. We may require that n be the smallest such number, but the choice is of no consequence: if $n<m$, then (the universal closure of)

$$M^n_\alpha[\bar{u},\bar{v}] \leftrightarrow M^m_\alpha[\bar{u}^\frown \bar{z},\bar{v}^\frown \bar{w}].$$

is clearly a theorem of L_{21}.

Given the definitions above, or similar definitions for other programming constructs, we define the *canonical logic* of the programming language at hand to be L_{21} augmented with the definitions. And we define the *canonical calculus* of the language to be L_2 similarly augmented.

2.2. *Partial correctness assertions and termination assertions.*

A *partial correctness assertion* is a formula of the form

$$\varphi[\bar{u}] \wedge M_\alpha[\bar{u},\bar{v}] \rightarrow \psi[\bar{v}].$$

A *total-correctness* (or *termination*) assertion is a formula of the form

$$\varphi[\bar{u}] \rightarrow \exists \bar{v}\ (\ M_\alpha[\bar{u},\bar{v}] \wedge \psi[\bar{v}]\,).$$

In both cases φ and ψ are first-order, with no occurrence in φ of a variable in $\bar{u}$, and no occurrence in ψ of a variable in $\bar{v}$.

Proposition 2. For every program α, the formula M_α is Π^R_1.

Proof. Induction on α. ■

Proposition 3. A total-correctness assertion is valid iff it is a theorem of L_{21}. Hence, the canonical logic of the programming language is complete for total correctness assertions.

Proof. By proposition 2 every total correctness assertion is a Π^R_1 formula. The statement then follows from proposition 1. ■

Note that proposition 3 does *not* imply that every termination assertion true over the natural numbers is provable in L_{21}. The termination assertion above, when interpreted over natural numbers, is rendered by

$$\wedge_{i \in I} N(u_i) \wedge \varphi[\bar{u}] \rightarrow \exists \bar{v}\ (\ M_\alpha[\bar{u},\bar{v}] \wedge \psi[\bar{v}]\,),$$

which is no longer a Π^R_1 formula.

2.3. *Meyer's Separation Principle.*

In [MH82] and [MM83] it has been observed that if programs α and β differ in their input-output behavior on some structure, then they have different partial-correctness theories, and different total-correctness theories: there are first-order formulas φ and ψ such that one of the partial-correctness assertions $\varphi\{\alpha\}\psi$ and $\varphi\{\beta\}\psi$ is valid, while the other is not; and similarly for total-correctness.

The proof for the partial-correctness case in [MH82, theorems 4.1,4.4] uses numeric (Godel-) coding and a first-order axiomatization of the ring of integers [MH82, appendix B]. The proofs for the total-correctness case [MH82, theorem 5.1; MM83 theorem 1] use a approximations of recursive programs by simpler programs.

We show how Meyer's Separation Principle for partial correctness falls out as a trivial consequence of the Π_1^R definability of the semantics of programs. This holds for any programming language whose semantics has such a definition.

Proposition 4 (Separation by partial-correctness). Suppose α and β are programs such that

$$M_\alpha \leftrightarrow M_\beta$$

is not valid. Then there is a first-order formula ψ such that one of the partial correctness assertions

$$\mathrm{true}\{\alpha\}\psi \quad \text{and} \quad \mathrm{true}\{\beta\}\psi$$

is valid, while the other not. Moreover, the valid one is a theorem of L_2.

Proof. Suppose $\mathbf{M}$ is a structure where

$$(1) \quad \mathbf{M} \models M_\alpha[\bar{a},\bar{b}] \quad \text{but} \quad \mathbf{M} \not\models M_\beta[\bar{a},\bar{b}],$$

where $\bar{a}$ and $\bar{b}$ are tuples of elements in the (ground) domain of $\mathbf{M}$. Let $\hat{M}_\alpha \equiv \forall \bar{R}\chi_\alpha$ be the pure Π_1^R form of M_α, with χ_α first-order; similarly for $\hat{M}_\beta \equiv \forall \bar{R}\chi_\beta$. By (1) there is a model $\mathbf{M}^+$ such that

$$(2) \quad \mathbf{M}^+ \models \neg\chi_\beta[\bar{a},\bar{b}].$$

Let

$$\psi[x] \equiv \exists\bar{u}\chi_\beta[\bar{u},\bar{x}].$$

Then

$$\models M_\beta[\bar{u},\bar{v}] \rightarrow \psi[\bar{v}/\bar{x}]$$

trivially. But

$$\not\models M_\alpha[\bar{u},\bar{v}] \rightarrow \psi[\bar{v}/\bar{x}],$$

since

$$\mathbf{M}^+[\bar{a}/\bar{u}][\bar{b}/\bar{v}] \models M_\alpha[\bar{u},\bar{v}],$$

by (1), but

$$\mathbf{M}^+[\bar{a}/\bar{u}][\bar{b}/\bar{v}] \models \neg\psi[\bar{v}],$$

by (2). ■

To apply the same method to Meyer's Separation Principle for total-correctness we need an existential definition of the semantics of programs. Such a definition is straightforward in $L_{\omega_1\omega}$ (using equality as a primitive, this time). Using $\tilde{M}_\alpha[\bar{u},\bar{v}]$ to denote the input-output relation generated by program α, we define, for example, for $\alpha \equiv$ while φ do β,

$$\tilde{M}_\alpha[\bar{u},\bar{v}] \equiv \neg\varphi[\bar{v}/\bar{x}] \wedge (\vee_{j\geq 0} Step_j[\bar{v}/\bar{z}]),$$

where

$$Step_0[\bar{z}] \equiv \bar{z}=\bar{u},$$
$$Step_{j+1}[\bar{z}] \equiv \exists\bar{y}\,(Step_j[\bar{y}] \wedge \varphi[\bar{y}/\bar{x}] \wedge \tilde{M}_\beta[\bar{y},\bar{z}]).$$

Meyer's Separation Principle for total correctness theories is then proved by an argument dual to the above.

Note that $L_{\omega_1\omega}$ is used here purely as a technical tool.

3. Modal reasoning about imperative programs.

3.1. *Modal languages and logically closed fragments.*

The intended semantics of modal logics is usually based on some notions of "state" and of "transition" between states, but with no explicit reference made to either notion in the modal language itself. Explicit reference to states is avoided by considering states only insofar as they are related by transitions. Such states, the "before" and "after" of a transition τ, can be implicitly separated in a formula by the modal operator-occurrence which denotes τ: the scope of that occurrence refers to the aftermath of the transition, the rest of the formula to its antecedent.

Pratt has applied these ideas to reasoning about programs, yielding Dynamic Logic [Pra76, Pra80, Har79]. The semantics of Pratt's box and diamond modal operators is given by

$$(1) \quad [\alpha]\varphi \equiv \forall\bar{v}\,(M_\alpha(\bar{x},\bar{v}) \rightarrow \varphi[\bar{v}/\bar{x}])$$

and

$$(2) \quad \langle\alpha\rangle\varphi \equiv \exists\bar{v}(M_\alpha(\bar{x},\bar{v}) \wedge \varphi[\bar{v}/\bar{x}])$$

These definitions are quite general, and may be applied to *any* programming language (whose operational semantics is definable in Second-Order Logic).

We can then define, for any programming language, its canonical modal language, built freely from first-order logic using the modalities $[\alpha]$ and $\langle\alpha\rangle$, where α ranges over the programs in the language. Each formula φ in this language can be translated into a formula φ^{exp} of Second-Order Logic, using (1) and (2). A *canonical logical formalism* and a *canonical mathematical formalism* can now be specified: The first consists of the modal formulas φ for which φ^{exp} is a theorem of L_{21}. The second corresponds similarly to L_2.

It may be appropriate to note that not every form of reasoning about programs is modal. By definition, whenever one reasons about states directly, rather than about transitions, the reasoning is *not* modal, and attempts to present it as such are unnatural at best. Much of the reasoning about concurrency, for example, falls under this comment.

In practice one may be interested in only a fragment of the modal language of a given programming language. Hoare's Logic, for example, is a formalism for deriving only modal formulas of the form $\psi \rightarrow [\alpha]\psi$. Similarly, Hoare-style calculi for total correctness assertions are formalisms for deriving formulas of the for $\psi \rightarrow \langle\alpha\rangle\psi$. Are there sound criteria for choosing particular sublanguages of Dynamic Logic, and attempt to design formalisms for them?

Our use of L_2 and L_{21} as master theories indicate a relative criterion of this kind. Suppose one is interested in proving modal formulas of a particular form, say $[\alpha]\varphi$. It may be necessary to use modal formulas of other forms in such proofs. (To see what formulas are needed, one looks at *normal* natural deduction proofs, or cut-free sequential proofs of L_{21}; see §3.2). For example, it becomes quickly clear that proving a formula of the form $[\alpha]\psi$ may require proving formulas of the form $\psi \rightarrow [\beta]\psi$. Some more reflection will show that, for while programs, no additional forms of modal formulas are necessary. The choice of partial correctness assertions as the only use of modality is therefore clearly natural.

We now generalize this example. Let L be the modal language of a programming language, F a fragment of L. The *logical closure* of F is the smallest fragment C of L, containing F, and such that for any formula $\varphi \in C$, if φ^{exp} is a theorem of L_{21}, then φ^{exp} has a proof in which all (translations of) modal formulas are in C. A fragment F of L is *logically closed* if it is equal to its own closure. A concept of *mathematically closed* fragments can be defined similarly, with L_2 in place of L_{21}, but it is of lesser interest (because the unrestricted use of Comprehension usually prohibits non-trivial mathematically closed fragments).

Using this terminology, partial correctness assertions constitute the closure of the fragment $\{ [\alpha]\varphi \mid \varphi$ first-order, α a while-program $\}$. This is no longer true for a programming language allowing global procedures (compare [MM83]). Similarly, logical reasoning about concurrency (i.e. reasoning in L_{21}) can *not* be confined to partial correctness assertions, as discussed in [Bro85]. Insisting on using only partial correctness assertions here may lead to rather contrived formalisms. Another interesting example is provided by the fragment $\{ \langle\alpha\rangle\varphi \mid \varphi$ first-order, α a while-program $\}$. Its closure is *not* the class of total correctness assertions. Consequently, the fragment consisting of total correctness assertions is not logically closed.

Given a logically closed fragment F, the *canonical logic for F* is the canonical logic (for the programming language at hand) restricted to F.

3.2. The logical completeness of Hoare logic.

We are now in a position to show that Hoare Logic for while programs is complete for logical reasoning about such programs. By *Hoare Logic* (for while programs) we understand here the modal calculus H over partial correctness assertions, with Assignment and Invariance as axioms (axioms 1 and 9 in [Apt81]), and with the Composition, Branching, Iteration and Consequence Rules (rules 2,3,4,5 in [Apt81]).

Proposition 5. H is (strongly) sound for $\models^{L21}$, and hence for $\models^{S}$. I.e., if Γ is a set of first-order formulas, and

$$\Gamma \vdash_H \varphi\{\alpha\}\psi$$

then

$$\Gamma \vdash_{L_{21}} \varphi[\bar{u}/\bar{x}] \wedge M_\alpha[\bar{u},\bar{v}] \rightarrow \psi[\bar{v}/\bar{x}].$$

Proof. Verify that the interpretation of each axiom of H is provable in L_{21}, and that the inference rules are valid in L_{21}. ■

The converse is somewhat more complicated.

Lemma 6. (Interpolation for Π_1^R formulas) Suppose $\varphi \rightarrow \psi$ is a Π_1^R formula. If it is a theorem of L_{21}, then there is a *first-order* formula χ, in which all first- (and second-) order variables are common to φ and ψ, such that $\varphi \rightarrow \chi$ and $\chi \rightarrow \psi$ are theorems of P_{21}.

Proof. Similar to any one of the standard proof-theoretic proofs of the interpolation theorem for First-Order Logic (see e.g. [Tak75]). ■

Lemma 7. Let $\alpha \equiv \beta;\gamma$, and let φ and ψ be first-order formulas. Suppose that

(1) $\varphi[\bar{u}] \wedge M_\alpha[\bar{u},\bar{v}] \rightarrow \psi[\bar{v}]$.

is a theorem of L_{21}. Then there is a *first*-order formula ι such that

$$\varphi[\bar{u}] \wedge M_\beta[\bar{u},\bar{z}] \rightarrow \iota[\bar{z}]$$

and

$$\iota[\bar{z}] \wedge M_\gamma[\bar{z},\bar{v}] \rightarrow \psi[\bar{v}]$$

are theorems of L_{21}.

Proof. Suppose (1), i.e.

$$\begin{aligned}&\varphi[\bar{u}] \wedge \\ &\quad \forall R^n \,[\,(\forall \bar{z}(M^n_\beta(\bar{u},\bar{z}) \rightarrow R(\bar{z})) \rightarrow \\ &\qquad \exists \bar{z}(R(\bar{z}) \wedge M^n_\gamma(\bar{z},\bar{v}))]. \\ &\qquad\qquad\qquad \rightarrow \psi[\bar{v}],\end{aligned}$$

by a normal proof of L_{21} [Pra71]. It is easy to see that there are first-order formulas χ_i, $i \in I$, such that the following is derived:

$$\varphi \wedge \wedge_{i \in I}(U_i \rightarrow E_i) \rightarrow \psi.$$

where

$$U_i \equiv \forall \bar{z}(\, M_\beta[\bar{u},\bar{z}] \rightarrow \chi_i[\bar{z}])$$

and

$$E_i \equiv \exists \bar{z}(\, \chi_i[\bar{z}] \wedge M_\gamma[\bar{z},\bar{v}]),$$

and where $\varphi \equiv \varphi[\bar{u}]$ and $\psi \equiv \psi[\bar{v}]$. Thus

$$\varphi \wedge \wedge_{i \in I}(\neg U_i \vee E_i) \rightarrow \psi,$$

and so

$$\varphi \wedge \vee_{J \subset I}\{(\wedge_{j \in J} \neg U_j) \wedge (\wedge_{j \in \bar{J}} E_j)\} \rightarrow \psi,$$

where $\bar{J} \equiv I - J$. It follows that

$$\wedge_{J \subset I}\{\, \varphi \wedge (\wedge_{j \in J} \neg U_j) \rightarrow ((\wedge_{j \in \bar{J}} E_j) \rightarrow \psi\,)\}.$$

Note that the premise of the main implication, in each conjunct, has no occurrence of variables in $\bar{v}$, and that the antecedent has no occurrence of variables in $\bar{u}$. By lemma 6, it follows that there are first-order formulas ξ_J, $J \subset I$, with no free occurrence of any variable in $\bar{u}$, $\bar{v}$, such that

$$\varphi \wedge \wedge_{j \in J} \neg U_j \rightarrow \xi_J$$

and

$$\xi_J \wedge \wedge_{j \in \bar{J}} E_j \rightarrow \psi.$$

Substituting the definitions of U_j and E_j, this implies, for each $J \subset I$,

$$\vee_{j \in J} \forall \bar{z}\,[\, \varphi \wedge M_\beta \rightarrow (\chi_j \vee \xi_J)\,]$$

and

$$\vee_{j \in \bar{J}} \forall \bar{z}\,[\, \chi_j \wedge \xi_J \wedge M_\gamma \rightarrow \psi\,],$$

where

$$M_\beta \equiv M_\beta[\bar{u},\bar{z}], \quad M_\gamma \equiv M_\gamma[\bar{z},\bar{v}].$$

This implies

$$\forall \bar{z}\,[\, \varphi \wedge M_\beta \rightarrow \vee_{j \in J}\, \chi_J \vee \xi_J)\,]$$

and

$$\forall \bar{z}\,[\, \wedge_{j \in J}\, \chi_j \wedge \xi_{\bar{J}} \wedge M_\gamma \rightarrow \psi\,].$$

Hence

$$\varphi \wedge M_\beta \rightarrow \wedge_{J \subset I}[\vee_{j \in J}\, \chi_j \vee \xi_J)\,]$$

and

$$[\,\vee_{J \subset I}(\, \wedge_{j \in J}\, \chi_j \wedge \xi_{\bar{J}}\,) \wedge M_\gamma\,] \rightarrow \psi\,.$$

The proof is then concluded by applying the following lemma. ■

Lemma 8. For any set I, the following schema is derivable in Propositional Logic.

$$\begin{aligned}&\wedge_{J \subset I}[\vee_{j \in J}\, p_j \vee q_J)\,] \rightarrow \\ &\qquad\qquad \vee_{J \subset I}[\, \wedge_{j \in J}\, p_j \wedge q_{\bar{J}}\,].\end{aligned}$$

Proof. Assume the premise true and the conclusion false. Then $\neg q_J$ for all $J \subset I$. Taking $J = I$, we get from the premise that p_{i_0} holds for some $i_0 \in I$. Let $J_1 \equiv I - \{i_0\}$. Then the premise implies similarly that p_{i_1} holds for some $i_1 \in I_1$, unless $I_1 = \emptyset$. Continuing, we see that p_i holds for all $i \in I$. But then $\wedge_{i \in J}\, p_i$ holds for $J = I$. Also, the assumption for $J = \emptyset$ implies q_I, since and empty disjunction is true. Thus the conclusion holds for the disjunct $J = I$, a contradiction. ■

Proposition 9. **(Completeness of Hoare's Logic for logical reasoning about programs)** H is complete for logical reasoning about programs: if a partial correctness assertion is provable in L_{21}, then it is provable in H.

Proof. By induction on the program in the partial correctness assertion. Lemma 7 is the induction step for the composition case. Other cases are treated similarly. ■

Corollary 10. Hoare's Logic is exactly the canonical logic for partial correctness assertions (for **while** programs).

Proof. Immediate from propositions 5 and 9. ■

Proposition 9 can be slightly generalized to the following, which we use in the sequel.

Proposition 11. **(Strong Completeness of Hoare's Logic)** If a partial correctness assertion is provable in L_{21} from a set of first-order formulas (or even Σ^R_1 formulas), then it is provable in H from the same set. ■

References.

[ANS82] A. Andreka, I. Nemeti and I Sain, *A complete logic for reasoning about programs via non-standard model theory*. **Theoretical Computer Science** 17 (1982) 193-212, 259-278.

[Apt81] Krzysztof Apt, *Ten years of Hoare's Logic: a survey — part I*; **ACM Transactions on Programming languages and Systems** 3 (1981) 431-483.

[Bro85] Stephen D. Brookes, *On the axiomatic treatment of concurrency*. **Proceedings of the NSF-CERS Seminar on Concurrency**, Spinger-Verlag, New York (1985).

[BT81] Jan A. Bergstra and John V. Tucker, *Hoare's logic and Peano's Arithmetic*, Mathematisch Centrum REport IW 160/81 (1981) i+27.

[deB80] Jaco de Bakker, **Mathematical Theory of Program Correctness**, Prentice-Hall, Englewood Cliffs (1980) xvii+505.

[EFT84] H.D. Ebbinghaus, J. Flum and W. Thomas, **Mathematical logis**; Springer-Verlag, New York, (1984) ix+216.

[End70] Herbert B. Enderton, **A Mathematical Introduction to logic**; Academic Press, New York (1970) xiii+295pp.

[FLO83] Steven Fortune, Daniel Leivant, and Michael O'Donnell, "The expressiveness of simple and second order type structures," *Journal of the ACM 30* (1983), pp 151-185.

[FSS83] Harvey M. Friedman, Stephen G Simpson and Rick L. Smith, *Countable algebras and set existence axioms*, **Annals of Pure and Applied Logic** 25 (1983) 141-181.

[Gir72] Jean-Yves Girard, *Interprètation fonctionelle et èlimination des coupures dans l'arithmètique d'ordre superieur*, Thèse de Doctorat d'Etat, 1972, Paris.

[HA38] David Hilbert and Wilhelm Ackermann, **Grundzuge der theoretischen Logik** (Second Edition), Springer, Berlin (1938) viii+133.
English translation: **Principles of Mathematical Logic**, Chelsea, New York (1950) xii+172.

[Har79] David Harel, **First-Order Dynamic Logic**, Lecture Notes in Computer Science 68, Springer-Verlag, 1979.

[Hen50] Leon Henkin, *Completeness in the Theory of Types*, **Journal of Symbolic Logic** 15 (1950), 81-91.

[KK67] Georg Kreisel and Jean-Louis Krivien, **Elements de Logique Mathematique**, Dunod, Paris (1967) viii+213

[Kle52] S.C. Kleene, **Introduction to Metamathematics**, Noordhoff, Groningen (1952) x+550.

[Lei83] Daniel Leivant, *Reasoning about functional programs and complexity classes associated with type disciplines*", **Twenty-fourth Annual Symposium on Foundations of Computer Science** (1983) 460-469.

[Lei84] Daniel Leivant, *Typing and comvergence in the Lambda Calculus*, manuscript (1984).

[Lin69] Per Lindstrom, *On extensions of elementary logic*, **Theoria** 35 (1969) 1-11.

[MH82] Albert Meyer and Joseph Halpern, *Axiomatic definition of programming languages: a theoretical assessment*, **Journal of the ACM** 29 (1982) 555-576.

[MH82] Albert Meyer and John C. Mitchell, *Termination assertions for recursive programs: complexity and axiomatic definability*, **Information and Control** 56 (1983) 112-138.

[MPS84] David B. MacQueen, Gordon Plotkin and Ravi Sethi, "An ideal model for recursive polymorphic types," *Conference Record of the Eleventh Annual ACM Symposium on Principles of Programming Languages* (1984) 165-174.

[Nem82] I. Nemeti, *Nonstandard dynamic logic*: in: D. Kozen (ed.), **Logics of Programs**, Springer-Verlag (LNCS #131), Berlin (1982) 311-348.

[OG76] Susan Owicki and David Gries, *An axiomatic proof thechnique for parallel programs I*, **Acta Informatica** 6 (1976) 319-340.

[Pra76] Vaughan Pratt, *Semantical considerations on Floyd-Hoare logics*, **Proceedings of the Seventeenth IEEE Symposium on Foundations of Computer Science** (1976) 109-121.

[Pra76] Vaughan Pratt, *Applications of modal logic to programming*, **Studia Logica** 39 (1980) 257-274.

[Pra71] Dag Prawitz, *Ideas and results in Proof Theory*, **Proceedings of the Second Scandinavian Logic Symposium** (J.E. Fenstad, editor) North-Holland, Amsterdam (1971) 235-307.

[Pra65] Dag Prawitz, **Natural Deduction**, Almqvist and Wiksell, Uppsala, 1965.

[Rey74] John C. Reynolds, *Towards a theory of type structures*, **Programming Symposium (Colloque sur la Programmation, Paris)**, Springer (LNCS # 19), Berlin (1974) 408-425.

[Rog67] Hartley Rogers Jr., **Theory of Recursive Functions and Effective Computability**, McGrow-Hill, New York (1967) ix+482.

[Rus20] Bertrand Russell, **Introduction to Mathematical Philosophy**, (Second Edition), London (1920).

[Rus37] Bertrand Russell, **Principles of Mathematics**, (Second Edition), London (1937).

[Tak75] Gaisi Takeuti, **Proof Theory**, North-Holland, Amsterdam (1975) vii+372.

[Wan74] Hao Wang, **From Mathematics to Philosophy**, Routlege & Kegan Paul, London (1974) xiv+428.

[Zuc80] Jeffery Zucker, *Expressibility of pre- and post-conditions*, in [deB80] 444-465.