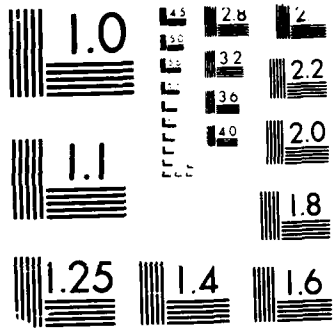
AD-A191 065 TOWARDS FULLY ABSTRACT SEMANTICS FOR LOCAL VARIABLES: 1/1
PRELIMINARY REPORT (U) MASSACHUSETTS INST OF TECH
CAMBRIDGE LAB FOR COMPUTER SCIENCE A R MEYER ET AL
UNCLASSIFIED NOV 87 MIT/LCS/TM-344 N00014-83-K-0125 F/G 12/5 NL



MICROCOPY RESOLUTION TEST CHART

LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-344

TOWARDS FULLY ABSTRACT SEMANTICS FOR LOCAL VARIABLES: PRELIMINARY REPORT

Albert R. Meyer

Kurt Sieber

November 1987

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS AD-A191 065
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TM-344	5 MONITORING ORGANIZATION REPORT NUMBER(S) N00014-83-K-0125
6a NAME OF PERFORMING ORGANIZATION MIT Laboratory for Computer Science 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION Office of Naval Research/Department of Navy
6c ADDRESS (City, State, and ZIP Code) 545 Technology Square Cambridge, MA 02139	7b ADDRESS (City, State, and ZIP Code) Information Systems Program Arlington, VA 22217
8a NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22217	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
Towards Fully Abstract Semantics for Local Variables: Preliminary Report

12 PERSONAL AUTHOR(S)
Meyer, Albert R. and Sieber, Kurt

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Technical	FROM _____ TO _____	1987 November	16

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Languages, Semantics, Logic, Correctness, ALGOL-like, block structure, stack discipline, cpo's, functors

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The Store Model of Halpern-Meyer-Trakhtenbrot is shown--after suitable repair--to be fully abstract model for a limited fragment of ALGOL in which procedures do not take procedure parameters. A simple counter-example involving a parameter of program type shows that the model is not fully abstract in general. Previous proof systems for reasoning about procedures are typically sound for the HMT store model, so it follows that theorems about the counter-example are independent of such proof systems. Based on a generalization of standard cpo based models to structures called locally complete partial orders (lcps's), improved models and stronger proof rules are developed to handle such examples.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Judy Little, Publications Coordinator	22b TELEPHONE (Include Area Code) (617) 253-5894 / 22c OFFICE SYMBOL

Towards Fully Abstract Semantics for Local Variables: Preliminary Report

Albert R. Meyer Kurt Sieber
MIT Laboratory for Computer Science

Abstract. The Store Model of Halpern-Meyer-Trakhtenbrot is shown—after suitable repair—to be a fully abstract model for a limited fragment of ALGOL in which procedures do not take procedure parameters. A simple counter-example involving a parameter of program type shows that the model is not fully abstract in general. Previous proof systems for reasoning about procedures are typically sound for the HMT store model, so it follows that theorems about the counter-example are independent of such proof systems. Based on a generalization of standard cpo based models to structures called *locally complete partial orders (lcpo's)*, improved models and stronger proof rules are developed to handle such examples.

CR Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*syntax, semantics*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*data type specifications, program equivalence*; F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs.

General Terms: Languages, Semantics, Logic, Correctness

Additional Key Words and Phrases: ALGOL-like, block structure, stack discipline, cpo's, functors

---

This Technical Memorandum will appear in the Proceedings of the 15^{th} Annual ACM Symposium on Principles of Programming Languages, January, 1988.

Both authors were supported in part by NSF Grant No. 8511190-DCR and by ONR grant No. N00014-83-K-0125. Sieber was also supported by a DAAD/NATO Fellowship; Sieber's present address is FN 10 Informatik, Univ. des Saarlandes, 66 Saarbrücken, West Germany.

1 Introduction

Some unexpected problems in the semantics and logic of block-structured local variables have been identified by Halpern, Meyer, and Trakhtenbrot [10.28]. The usual cpo based models for stores and programs do not satisfactorily model the stack discipline of blocks in ALGOL-like languages. The simplest example involves a trivial block which calls a parameterless procedure identifier P.

Example 1 *The block below is replaceable simply by the call P.*

```
begin
  new x;
  P;              % P is declared elsewhere
end
```

It is easy to argue informally that the block in Example 1 acts the same as P. Namely, since ALGOL-like languages mandate static scope for local variables, it follows that P has no access to the local variable x, so allocating x and then deallocating it if and when the call to P returns, can have no influence on the call to P.

A similar, slightly more interesting example illustates some of the features of the "pure" ALGOL-like dialect we consider here.

Example 2 *The block below always diverges.*

```
begin
  new x;
  x := 0;
  P;                    % P is declared elsewhere
  if contents(x) = 0 then diverge fi
end
```

To verify Example 2, we note that the definition of ALGOL-like languages in [10.28] implies that the call of P has side-effects on the store only, *viz.*, *no input/output effects*, and *no goto's* or other transfers of control. This is essentially the same language Reynolds has called the "essence" of ALGOL [22] without *goto*'s or jumps . In particular, the only way the call of P in the block can fail to return is by diverging. If the call does return, then since the contents of x equals zero immediately before the call, static scope again implies that the contents will still be zero when the call returns, so the conditional test will succeed causing divergence in any case.

Note that these arguments implicitly presuppose that P is a call to a declared procedure. That is, the arguments really show that if $C[\cdot]$ is any closed ALGOL-like program context such that $[\cdot]$ is a "hole" within the scope of a declaration of P, then C[Block 1] has exactly the same effect on the store as $C[P]$, and likewise C[Block 2] has exactly the same effect as $C[diverge]$. We say that the block in Example 1 and P are *observationally congruent* wrt ALGOL-like contexts; likewise the block in Example 2 is observationally congruent to $diverge$.

On the other hand, if P was a call of an independently compiled "library" program — even one originally written in ALGOL — which did not share the memory management mechanisms of the ALGOL compiler used on Blocks 1 and 2, then the call might detect changes on the stack of variables like x, and might even alter the contents of stack variables, making the behavior of the blocks unpredictable. Thus, we have *not* shown that the Block 1 is *semantically* equivalent to P, even when the values of P range only over ALGOL-like procedures.

Indeed, the congruences of Examples 1 and 2 are *not* semantical equivalences in the standard denotational semantics for ALGOL-like languages using "marked" stores [12,9]. In such semantics, Block 1 and P are only equivalent on stores in which the locations "accessible" to P are *correctly* marked as in use, but certainly not on incorrectly marked stores.

The problem which motivates this paper is to provide mathematical justification for the informal but convincing proofs of observational congruences like the two above. Following [10], we approach the problem by trying to construct semantical models of ALGOL-like languages in which semantical equivalence is a good guide to observational congruence. An ideal situation occurs when the mathematical semantics is *fully abstract*, *i.e.*, semantic equivalence *coincides* with observational congruence. However, experience in domain theory suggests that full abstraction is hard to achieve and may not even be appropriate [1,13]. Indeed, if the programming language in question suffers design weaknesses, it may be necessary to modify the language design to match a clean semantics; this is an important message of [20], for example. In this paper we describe several semantical models which are fully abstract for various ALGOL-like sublanguages, though not for the full range of ALGOL-like features.

In this preliminary paper, we omit a precise definition of full syntax and features of ALGOL-like languages, expecting that the examples below will be fairly clear without formal definition. It is helpful, as explained in [5,22,28,10,19], to regard the "true" syntax of ALGOL-like languages as the simply typed λ-calculus over the base types

$$Loc,\ Val,\ Locexp,\ Valexp,\ Prog$$

denoting memory *locations*, storable *values*, location thunks, value thunks, and *programs*. The calculus has fixed point, conditional, and other combinators suitable for interpreting the ALGOL-like phrase constructors such as assignments or command sequencing. Some theoretical ALGOL dialects restrict the set of procedure types so all calls are of type *Prog*, i.e., they return nothing, and exclude parameters of type *Locexp* and *Valexp*. These restrictions have little effect on our results.

2 The Usual Cpo Based Models

Denotational models must be *computationally adequate*, i.e., they must agree with the standard operational semantics (copy-rule in the case of ALGOL) giving the computational behavior of completely declared program blocks. In many adequate semantics, semantic equivalence implies observational congruence. The usual cpo based models are satisfactory in this respect.

For example, a typical cpo based "marked store" model takes the base type representing locations to be $Loc_\perp$, the flat cpo over a countably infinite set *Loc*, and the base type of storable values to be $Val_\perp$ for some set *Val*. Let $Stores = (Loc \xrightarrow{f} Val) \times \mathcal{P}_{fin}(Loc)$, where $A \xrightarrow{f} B$ denotes the set of all total set-theoretic functions from set A to set B, and $\mathcal{P}_{fin}(A)$ denotes the set of all finite subsets of A. The intention is that when $(c, m) \in Stores$, the set $m \subseteq Loc$ denotes the marked locations and $c(l) \in Val$ gives the contents of location l. Let the base type *Valexp* of value-thunks be interpreted as $Stores \xrightarrow{t} Val_\perp$, partially ordered pointwise; similarly for the base type *Locexp*. Let the base type *Prog* of programs be $Stores \xrightarrow{t} Stores$, where $\xrightarrow{p}$ denotes the set of all set-theoretic partial functions partially ordered under containment. Each of these base types is now a cpo, and we interpret higher order like functional types by taking the *continuous* functions. Then we may summarize the introductory discussion by:

Theorem 1 *The "marked stores" cpo based model for the* ALGOL*-like language* PROG *of [O'S] or the language Reynolds calls the "essence of* ALGOL*" [R2], without goto's or escapes, is computationally adequate but not fully abstract.*

In fact, *without local variables*, the simpler "continuous stores" model in which $Stores = Loc_\perp \to Val_\perp$, $Valexp = Stores \to Val_\perp$, similarly for *Locexp*, and $Proc = Stores \to Stores$, where $\to$ denotes total continuous functions, is fully abstract after one modification. Namely, for reasons which will be clear to readers familiar with [2], a "parallel or" combinator $por : (Val_\perp \times Val_\perp) \to Val_\perp$ must be added to the language, where

$$[\![por]\!](1, c) = 1, \quad [\![por]\!](c, 1) = 1,$$
$$[\![por]\!](0, 0) = 0, \quad [\![por]\!](\perp, \perp) = \perp.$$

for two distinguished elements $\perp \neq 0 \neq 1 \neq \perp$ and all $v \in Val_\perp$.

Theorem 2 *The continuous store model for the language* PROG, without *the* **new** *local variable declaration and* with *an additional* $\parallel\vee$*-combinator is computationally adequate and fully abstract.*

Note that Example 2 makes it clear that the marked store model is still not fully abstract even with the addition of $\parallel\vee$. Thus, Theorems 1 and 2 confirm that local variables are a source of difficulty in this approach. We remark that although Theorem 1 has nearly the status of a folk theorem in domain theory, we know of no published proof; our own proof follows the computability method applied to the functional language PCF in [20]. Our proof of Theorem 2 again applies the results of [20] about definability of "finite" elements together with some folk theorems connecting Clarke's restricted ALGOL-like language L3 [2,3,7] and higher-order recursive function schemes [4,8].

3 Halpern-Meyer-Trakhtenbrot Store Models

To handle Examples 1 and 2, Halpern-Meyer-Trakhtenbrot proposed a formal definition of the *support* of a function from *Stores* to *Stores*. Intuitively the support of a store transformation p is the set of locations which p can read or write. In the HMT store model [10], *Prog* is taken to be the set of p with *finite* support. To model local variables, the notion of support is extended to the type $Loc \to Prog$ of block bodies regarded as a function of their free location identifier. The semantical space used to interpret such block body functions is again restricted to be the elements in $Loc_\perp \xrightarrow{c} Prog$ with finite support. Since there are an infinite number of locations, this restriction guarantees that a location can be found which is not in the support of any given block body. Then local storage allocation for a block **begin new** x; *body* **end** is (uniquely) determined by the rule that x be bound to *any* location not in the support of the function denoted by $\lambda x.body$.

Thus the HMT model justifies the conclusion that Block 2 diverges: if P denotes some state transformation $p \in Prog$, then any location $l \notin support(p)$ can be bound to x. This proves divergence of the block, because p — by definition — cannot change the contents of locations outside its support.

The definition of support for block body functions requires another ingredient: locations which are "recognized" by the block body — even if they are neither read nor written — must be counted in the support of the block body. Thus:

Example 3 *The blocks*

$$\textbf{begin new } x\textbf{; new } y\textbf{: } x := 0\textbf{: } y := 0\textbf{: } Q(x, y) \textbf{ end}$$

and

$$\textbf{begin new } x\textbf{; new } y\textbf{: } x := 0\textbf{: } y := 0\textbf{: } Q(y, x) \textbf{ end}$$

are HMT equivalent.

The argument for equivalence of the blocks goes briefly as follows. Let $q \in (Loc_\perp \times Loc_\perp) \rightharpoonup Prog$ be the meaning of the procedure identifier Q. The definition of local variable allocation in the HMT model implies that x and y can be bound in the body of either block to distinct locations $l_x, l_y \notin support(q)$. By definition of support, q cannot recognize locations not in its support, treating them in a uniform way (*cf.* Appendix A), so the store transformations $q(l_x, l_y)$ and $q(l_y, l_x)$ agree on all stores s with $s(l_x) = s(l_y)$ whose restrictions to $support(q) \cup \{l_x, l_y\}$ are the same. Since $contents(l_x) = contents(l_y) = 0$ when the block bodies begin execution—and stack discipline specifies that the contents are restored to their original values on deallocation—it follows that both blocks define the same store transformation as $q(l_x, l_y)$ restricted to $support(q)$.

The HMT store model was claimed to be computationally adequate, but not necessarily fully abstract. Its successful handling of Examples 1–3 is a consequence of the following general result about the "first-order" ALGOL-like sublanguage without *goto*'s and jumps, in which procedure parameters are restricted to be of type *Val* and *Loc* (essentially the language considered in [6]).

Theorem 3 *The HMT store model is computationally adequate for all* ALGOL*-like language features other than* goto*'s and jumps. It is fully abstract wrt to the "first-order" sublanguage with an additional* $\|\vee$*-combinator.*

We remark here that we have been generous in our references to the HMT store model described in [10], since in fact the construction sketched there contains a serious technical error—noted independently by the second author and A. Stoughton. In Appendix A we repair this error, and moreover develop a methodology for constructing improved models based on the notion of locally complete partial orders (lcpo's). Thus, Theorem 3 refers to the corrected HMT store model.

We now consider some second-order examples.

Example 4 *The block below always diverges.*

```
begin
  new x; new y;
  procedure Twice; begin y := 2 * contents(y) end;
  x := 0; y := 0;
  Q(Twice);                          % Q is declared elsewhere
  if contents(x) = 0 then diverge fi
end
```

Two additional reasoning principles about support which hold in the HMT-model (*cf.* Appendix A), arise in handling this example. First, in reasoning about program text in the scope of a local variable declaration **new** x, we may assume that the value of x is any convenient location *not* in the support of (the values of) each of the free identifiers in the scope of the declaration. Second, we always have $support(Q(P)) \subseteq support(P) \cup support(Q)$. Now clearly, $support(Twice) = \{y\}$. Since x is free in the scope of the **new** y declaration, the first principle applied to y implies that x and y denote different locations, so $x \notin support(Twice)$. Since Q is free, $x \notin support(Q)$. By the second principle, we may now assume $x \notin support(Q(Twice))$. Hence, we may reason about the call $Q(Twice)$ in Example 4 exactly as we did for the call P in divergent block of Example 2.

Unfortunately the HMT model does not handle all examples with second-order procedures, as the following elegant counter-example pointed out to us by A. Stoughton makes clear:

Example 5 *The block below always diverges.*

```
begin
  new x;
  procedure Add_2 :                  % Add_2 is the ability to add 2 to x
      begin x := contents(x) + 2 end
  x := 0;
  Q(Add_2);                          % Q is declared elsewhere
  if contents(x) mod 2 = 0 then diverge fi
end
```

The block in Example 5 does not diverge identically in HMT because Q might denote an element $q \in Prog \xrightarrow{c} Prog$ such that $q(p)$ is a program which sets to one all locations writable by p. Such a q exists in the HMT model because it is continuous (in the HMT sense, *cf.* Appendix A) and has empty support. However, Block 5 *is* observationally equivalent to *diverge*: Q has no independent access to the local variable x, so the only ability the program $Q(Add\_2)$ has relative to x is the ability to increment its contents by two. Since

contents(x) is an even integer, namely zero, before execution of this program, it will still be even if and when the program terminates, so the conditional test will succeed and cause divergence. Thus we have

Lemma 1 *Block 5 is observationally congruent to diverge, but not equal to diverge in the HMT store model.*

Hence:

Theorem 4 *The HMT model is not fully abstract even for* PROG *programs whose procedure calls take parameters only of program type.*

This failure of full abstraction for the HMT store model is particularly interesting precisely because the model is a good one. In particular, the various rules and systems proposed in the literature for reasoning about procedures in ALGOL-like languages are all sound for the HMT model (insofar as they are sound at all, *cf.* [14]). It follows that the divergence of Block 5 (and perhaps Block 4 too) is *independent* of the theorems provable from other proof systems in the literature including [28,10,25,17,16,11,24]. Reynolds' specification logic [21,23] is shown in [26,27] to be intuitionistically sound using a functor category semantics; it is not yet clear how the semantics and logic of [27] handles these examples.

4 The Invariant-Preserving Model

In order to handle Example 5 we must know that every procedure Q of type $Proq \rightarrow Prog$ *preserves invariants* outside its support. This is expressed precisely by the following reasoning principle :

> Let Q be of type $Prog \rightarrow Prog$ and P of type $Proq$. Let r be a property of stores such that $support(r) \cap support(Q) = \emptyset$. If r is an invariant of P, then r is also an invariant of $Q(P)$.

This principle implies divergence of Block 5 because, letting r be defined by formula *contents*(x) mod 2 = 0, we see that $support(r) = \{x\}$ and r is an invariant of Add_2. Inside the block we may assume that $x \notin support(Q)$, and so the principle implies that r is also an invariant of $Q(Add\_2)$. Thus, the conditional test following the call $Q(Add\_2)$ will succeed leading to divergence.

The above reasoning principle is valid in the Invariant-Preserving model (*cf.* Appendix A). Actually *all* the previous examples are handled successfully by this model as a consequence of the following general result.

An ALGOL-like term is said to be *closed* iff its only free identifiers are of type *Loc*. A semantics is said to be *half-fully abstract* for a language iff semantic equality between two terms, *one of which is closed*, coincides with observational congruence.

Define the PASCAL-*like sublanguage* by the condition that procedure parameters are restricted to be of type *Val*, *Loc*, or $Val^n \times Loc^m \rightarrow Prog$ (essentially the language considered in [15,16]).

Theorem 5 *The Invariant-Preserving Model is computationally adequate for the full range of* ALGOL-*like language features. With an additional* $\forall$*-combinator, it is fully abstract for the "first-order" sublanguage and is half-fully abstract wrt to the* PASCAL-*like sublanguage.*

Since Example 5 involves observational congruence to a closed term which identically diverges, the Invariant-Preserving Model handles it as well as the following slightly more sophisticated variant. (Note that the test $z = x$ below indicates equality of locations, rather than their contents.)

Example 6 *The block*

```
begin
  new x;
  procedure AlmostAdd_2(z);
    begin if z = x then x := 1 else x := contents(x) + 2 fi end;
  x := 0;
  P(AlmostAdd_2);
  if contents(x) mod 2 = 0 then diverge fi
end
```

always diverges.

The following example illustrates failure of full abstraction in the Invariant-Preserving Model:

Example 7 *The block*

```
begin new x; procedure Add_1; begin x := contents(x) + 1 end; P(Add_1) end
```

is observationally congruent to the block

```
begin new x; procedure Add_2; begin x := contents(x) + 2 end; P(Add_2) end
```

The idea is that since P has no independent access to x, and since its actual parameters in Example 7 do not enable P to read *contents*(x), the procedure calls $P(Add\_1)$ and $P(Add\_2)$ differ only in their effect on x. Since x is deallocated on block exit, the two blocks are observationally equivalent. Nevertheless,

Lemma 2 *The* PASCAL*-like blocks in Example 7 are observationally congruent but not semantically equivalent in the Invariant-Preserving Model.*

Thus, still stronger proof principles than preservation of invariants are needed to formalize this last observational congruence argument. The reader may care to invent one.

5 Conclusion

We have seen a series of simple examples illustrating how to reason about block structured variables. Most of these principles have never been stated in the literature, let alone been proved sound. To establish soundness we constructed a series of models for ALGOL-like languages. The formal machinery for constructing the models based on lcpo's is sketched in Appendix A. It merits detailed discussion which we have had to forego here. The best of our models is still not fully abstract for PASCAL-like sublanguages, but we are working on a proof that our methods will extend to this case. We see no reason why our approach should not extend to the full range of ALGOL-like features, but it would be premature to conjecture that full abstraction can be achieved this way.

Oles and Reynolds [22,18,19] have also developed models of ALGOL-like languages using a categorical framework. They do not consider computational adequacy or full abstraction as explicit issues. Tennent has informed us in private communication that his version [27] of the Reynolds-Oles functor category semantics correctly handles Examples 1 and 2. The comparison between their approach and ours has yet to be worked out. Actually our approach can also be seen from a category theoretic viewpoint – an lcpo is a functor from a partially ordered index set to the category of cpo's, and the locally continuous functions are similar to, but not exactly, natural transformations between such functors – but thus far we have not found this viewpoint advantageous.

References

G. Berry, P. Curien, and J. Lévy. Full abstraction for sequential languages: the state of the art. In M. Nivat and J. C. Reynolds, editors, *Algebraic Methods in Semantics*, chapter 89, page 0, Cambridge Univ. Press, 1985.

[2] E. M. Clarke, Jr. Programming language constructs for which it is impossible to obtain good Hoare axiom systems. *J. ACM*, 26:129–147, 1979.

[3] E. M. Clarke, Jr., S. M. German, and J. Halpern. On effective axiomatizations of Hoare logics. *J. ACM*, 30:612–636, 1983.

[4] W. Damm. The IO- and OI-hierarchies. *Theoretical Computer Sci.*, 20:95–207, 1982.

[5] W. Damm and E. Fehr. A schematological approach to the analysis of the procedure concept in ALGOL-languages. In $5^{ième}$ *Coll. sur les Arbes en Algebre et en Programmation*, pages 130–134, Lille, 1980.

[6] J. DeBakker. *Mathematical Theory of Program Correctness*. Prentice-Hall, 1980.

[7] S. German, E. M. Clarke, and J. Halpern. Reasoning about procedures as parameters. In E. M. Clarke, Jr. and D. Kozen, editors, *Logic of Programs, 1983*, pages 206–220. Volume 164 of *Lect. Notes in Computer Sci.*, Springer-Verlag, 1984.

[8] A. Goerdt. Hoare logic for lambda-terms as basis of Hoare logic for imperative languages. In *Symp. Logic in Computer Sci.*, pages 293–299, IEEE, 1987.

[9] M. J. C. Gordon. *The Denotational Description of Programming Languages*. Springer-Verlag, 1979.

[10] J. Y. Halpern, A. R. Meyer, and B. A. Trakhtenbrot. The semantics of local storage, or what makes the free-list free? In 11^{th} *Symp. on Principles of Programming Languages*, pages 245–257, ACM, 1984.

[11] Z. Manna and R. Waldinger. Problematic features of programming languages: a situational-calculus approach. *Acta Informatica*, 16:371–426, 1981.

[12] R. Milne and C. Strachey. *A Theory of Programming Language Semantics*. Chapman and Hall, 1976.

[13] K. Mulmuley. *Full abstraction and semantic equivalence*. Ph.D. thesis, Carnegie-Mellon Univ., Pittsburgh, PA, 1985. Available as Technical Report CMU-CS-85-148.

[14] M. J. O'Donnell. A critique of the foundations of Hoare-style programming logic. *Commun. ACM*, 25:927–934, 1982.

[15] E. Olderog. A characterization of Hoare's logic for programs with Pascal-like procedures. In 15^{th} *Symp. on Theory of Computing*, pages 320–329, ACM, 1983.

[16] E. Olderog. Correctness of programs with PASCAL-like procedures without global variables. *Theoretical Computer Sci.*, 30:49–90, 1984.

[17] E. Olderog. Hoare's logic for program with procedures — what has been accomplished? In E. Clarke and D. Kozen, editors, *Logic of Programs, Proceedings 1983*, pages 383–395. Volume 164 of *Lect. Notes in Computer Sci.*, Springer-Verlag, 1984.

[18] F. J. Oles. *A category-theoretic approach to the semantics of programming languages*. Ph.D. thesis, Syracuse Univ., 1982.

[19] F. J. Oles. Type algebras, functor categories, and block structure. In M. Nivat and J. C. Reynolds, editors, *Algebraic Methods in Semantics*, chapter 15, pages 543–573. Cambridge Univ. Press, 1985.

[20] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Sci.*, 5:223–256, 1977.

[21] J. C. Reynolds. *The Craft of Programming*. Prentice-Hall Int'l., 1981.

[22] J. C. Reynolds. The essence of ALGOL. In J. DeBakker and van Vliet, editors, *Int'l. Symp. Algorithmic Languages*, pages 345–372, IFIP, North-Holland, 1981.

[23] J. C. Reynolds. Idealized ALGOL and its specification logic. In D. Néel, editor, *Tools and Notions for Program Construction*, pages 121–161, Cambridge Univ. Press, 1982.

[24] R. L. Schwartz. An axiomatic treatment of ALGOL 68 routines. In H. Maurer, editor, *6th Int'l. Coll. Automata, Languages and Programming*, pages 530–545. Volume 71 of *Lect. Notes in Computer Sci.*, Springer-Verlag, 1979.

[25] K. Sieber. A partial correctness logic for programs (in an ALGOL-like language). In R. Parikh, editor, *Logics of Programs*, pages 320–342. Volume 193 of *Lect. Notes in Computer Sci.*, Springer-Verlag, 1985.

[26] R. D. Tennent. Semantical analysis of specification logic (preliminary report). In R. Parikh, editor, *Logics of Programs*, pages 373–386. Volume 193 of *Lect. Notes in Computer Sci.*, Springer-Verlag, 1985.

[27] R. D. Tennent. Semantical analysis of specification logic. 1987. Submitted, 22 pp.

[28] B. A. Trakhtenbrot, J. Y. Halpern, and A. R. Meyer. From denotational to operational and axiomatic semantics for ALGOL-like languages: an overview. In E. Clarke and D. Kozen, editors, *Logic of Programs, Proceedings 1983*, pages 474–500. Volume 164 of *Lect. Notes in Computer Sci.*, Springer-Verlag, 1984.

A Appendix. Locally Complete CPO Models

For cpo's A, B, we write $A \lhd B$ to indicate that A is a strict sub-cpo of B, *i.e.*, A is a sub-cpo of B and $\perp_B \in A$.

Definition 1 *Let $(I, \leq)$ be a directed set. An I-lcpo is a partially ordered set $D = \bigcup_{i \in I} D_i$ with a least element $\perp_D$, where D restricted to D_i is a cpo, and $D_i \lhd D_j$ whenever $i \leq j$.*

It follows from this definition that $\perp_{D_i} = \perp_D$ for every $i \in I$.

Definition 2 *Let D and E be I-lcpo's. For $f : D \xrightarrow{t} E$, let f_i denote the restriction of f to D_i, and define*

$$(D \xrightarrow{lc} E)_i = \{f : D \xrightarrow{t} E \mid f_j \in D_j \xrightarrow{c} E_j \text{ for all } j \geq i\}.$$

Then $D \xrightarrow{lc} E = \bigcup_{i \in I} (D \xrightarrow{lc} E)_i$ is called the set of locally continuous *functions from D to E.*

Lemma 3 *$D \xrightarrow{lc} E$, partially ordered pointwise, is an I-lcpo.*

Lemma 4 *Every locally continuous function on an I-lcpo D has a least fixed point, which is characterized as usual.*

Definition 3 *Let D be an I-lcpo and $i \in I$. An n-ary relation R on D is called i-*admissible, *if $R(d, \ldots, d)$ holds for every $d \in D_i$, and the restriction of R to the cpo D_j^n is admissible for every $j \in I$.*

Definition 4 *A* tag set over I *is a set K such that every $k \in K$ has an associated number $n_k \geq 1$, and downward closed set, $down(k) \subseteq I$.*

Definition 5 *Let K be a tag set over I. A K-*relationally structured *I-lcpo (short: (I, K)-*rcpo*) is an I-lcpo D with, for each $k \in K$, an n_k-ary relation R_k on D, such that R_k is i-admissible for every $i \in down(k)$.*

For $R \subseteq D^n$, $S \subseteq E^n$ we let $R \to S$ denote the *lifted* relation on $D \xrightarrow{t} E$ defined by

$$(R \to S)(f_1, \ldots, f_n) \text{ iff } \forall d_1, \ldots, d_n.\ R(d_1, \ldots, d_n) \Rightarrow S(f_1(d_1), \ldots, f_n(d_n)).$$

Definition 6 *Let D, E be (I, K)-rcpo's. For every i define*

$$(D \xrightarrow{rp} E)_i = \{f \in (D \xrightarrow{lc} E)_i \mid (R_k^D \to R_k^E)(f, \ldots, f) \text{ whenever } i \in down(k)\}.$$

Then $D \xrightarrow{rp} E = \bigcup_{i \in I} (D \xrightarrow{rp} E)_i$ is called the set of relation preserving *functions from D to E.*

Lemma 5 *$D \xrightarrow{rp} E$, partially ordered pointwise and K-relationally structured by (the restrictions of) the lifted relations $R_k^{(D \xrightarrow{rp} E)} = (R_k^D \to R_k^E)$, is an (I,K)-rcpo.*

Theorem 6 *The category (I,K)-**RCPO**, whose objects are (I,K)-rcpo's and whose morphisms are relation preserving functions, is Cartesian closed. Hence, if every ground type is interpreted as an (I,K)-rcpo, and $D^{\tau_1 \to \tau_2}$ is defined inductively to be $D^{\tau_1} \xrightarrow{rp} D^{\tau_2}$, then $\{D^\tau\}$ is a model of the simply typed λ-calculus. In this model, the meaning of any pure λ-term of type τ is contained in D_i^τ for all i. Hence there is a least fixed point operator for each type τ, which is contained in $D_i^{(\tau \to \tau) \to \tau}$ for all i.*

Lemma 6 *Let $\{D^\tau\}$ be an rcpo-model as in Theorem 6. Then $R_k^\tau(d, \ldots, d)$ holds for all $d \in D_i^\tau$ whenever $i \in down(k)$, and if $i, j \leq h$, then $D_i^{\tau_1 \to \tau_2}(D_j^{\tau_1}) \subseteq D_h^{\tau_2}$ for all types τ_1, τ_2.*

A first application of Theorem 6 leads to a repaired version of the Halpern-Meyer-Trakhtenbrot model. Namely, let $Perm(Loc)$ be the set of strict permutations $\mu : Loc_\perp \xrightarrow{t} Loc_\perp$, and for every $\mu \in Perm(Loc)$, let $Fix(\mu)$ denote the set of fixed points of μ. Further let $Stores = Loc \xrightarrow{t} Val$, and for every $L \subseteq Loc$, let $=_L$ be the binary relation on $Stores_\perp$ defined by

$$s_1 =_L s_2 \quad \text{iff} \quad s_1 = \perp = s_2 \vee (s_1 \neq \perp \neq s_2 \wedge \forall l \in L.\ s_1(l) = s_2(l)).$$

Definition 7 *Let $I = \mathcal{P}_{fin}(Loc)$. Let $K = Perm(Loc)$, and for every $\mu \in K$, let $n_\mu = 2$ and $down(\mu) = \{L \mid L \subseteq Fix(\mu)\}$. Then the* HMT Store Model *is defined by ground type lcpo's*

$$\begin{aligned}
D_L^{Val} &= Val_\perp, \\
D_L^{Loc} &= L \cup \{\perp_{Loc}\}, \\
D_L^{Valexp} &= \{f : Stores \xrightarrow{t} Val_\perp \mid \forall s_1, s_2.\ s_1 =_L s_2 \Rightarrow f(s_1) = f(s_2)\}, \\
D_L^{Locexp} &= \{f : Stores \xrightarrow{t} Loc_\perp \mid (\forall s.\ f(s) \in L \cup \{\perp_{Loc}\}) \wedge \forall s_1, s_2.\ s_1 =_L s_2 \Rightarrow f(s_1) = f(s_2)\}, \\
D_L^{Prog} &= \{f : Stores \xrightarrow{t} Stores_\perp \mid (\forall s.\ f(s) \neq \perp \Rightarrow s =_{Loc-L} f(s)) \wedge \\
&\qquad \forall s_1, s_2.\ s_1 =_L s_2 \Rightarrow f(s_1) =_L f(s_2)\};
\end{aligned}$$

relationally structured by

$$\begin{aligned}
R_\mu^{Val}(v_1, v_2) \quad & \text{iff} \quad v_1 = v_2, \\
R_\mu^{Loc}(l_1, l_2) \quad & \text{iff} \quad \mu(l_1) = l_2, \\
R_\mu^{Valexp}(f_1, f_2) \quad & \text{iff} \quad \forall s.\ f_1(s \circ \mu) = f_2(s), \\
R_\mu^{Locexp}(f_1, f_2) \quad & \text{iff} \quad \forall s.\ f_1(s \circ \mu) = \mu(f_2(s)), \\
R_\mu^{Prog}(f_1, f_2) \quad & \text{iff} \quad \forall s.\ f_1(s \circ \mu) = f_2(s) \circ \mu.
\end{aligned}$$

By Theorem 6 this defines a model of the simply typed λ-calculus. It turns out that for every element $d \in D^\tau$ there is a *smallest* set L such that $d \in D^\tau_L$. This set L is called the *support* of d. Theorem 6 then implies that all pure λ-terms have meanings with empty support and Lemma 6 implies that $support(d_1(d_2)) \subseteq support(d_1) \cup support(d_2)$ always holds.

The definition of the model captures several aspects of *support* mentioned earlier. In particular for every element d of type τ in the model, $R^\tau_\mu(d, d)$ holds by Lemma 6 whenever $support(d) \subseteq Fix(\mu)$. This expresses the "uniformity" of d on locations outside its support, which was important in Example 3 and is crucial for defining the semantics of the New combinator.

Finally, the "intended" interpretations, namely, interpretations which guarantee computational adequacy, must be proved to exist in the model for all the ALGOL constants. An example is the combinator **Assign** of type $Loc \to Valexp \to Prog$, for interpreting assignment, whose intended meaning is the function

$$assign(l)(f)(s) = \begin{cases} s[f(s)/l] & \text{if } l \neq \bot, f(s) \neq \bot, \\ \bot & \text{otherwise.} \end{cases}$$

It follows from the definitions that $assign \in D_\emptyset^{Loc \to Valexp \to Prog}$; in particular, it has empty support—as do all the necessary combinators. So we can define $[\![\mathbf{Assign}]\!] = assign$ in the model.

The combinator which causes the main semantical problem is **New** of type $(Loc \to Prog) \to Prog$ used to explain the semantics of block structure by translation into λ-calculus:

$$\text{Translation}(\mathbf{begin\ new}\ x;\ Cmd\ \mathbf{end}) \ ::= \ (\mathbf{New}\ (\lambda x{:}Loc.\ \text{Translation}(Cmd))).$$

The definition of $[\![\mathbf{New}]\!]$ follows [10]; we omit the details here. This completes our summary of the HMT semantics.

Proof Sketch for full abstraction part of Theorem 3: Adapt the ideas of [20] to locally continuous models. Every element of a local cpo D^τ_L (where τ is first-order) is the lub of a directed set of finite elements in D^τ_L, and these finite elements are definable by closed ALGOL-like terms. Local continuity then implies that two semantically different phrases can be distinguished by choosing definable objects for their free procedures. This means that they can be distinguished by a program context. □

A further application of Theorem 6 leads to the Invariant-Preserving Model. For every $L \subseteq Loc$, let $Pred(L)$ be the set of predicates on stores which only depend on L, namely

$$Pred(L) = \{\pi : Stores \xrightarrow{t} \{true, false\} \mid \forall s_1, s_2 \in Stores.\ s_1 =_L s_2 \Rightarrow (\pi(s_1) \equiv \pi(s_2))\}.$$

Definition 8 *Let* $I = \mathcal{P}_{fin}(Loc)$ *and* $K = Perm(Loc) \cup \{(L, \Pi) \mid L \in I \text{ and } \Pi \subseteq Pred(L)\}$. *For* $\mu \in K$, *let* $n_\mu = 2$ *and* $down(\mu) = \{L \mid L \subseteq Fix(\mu)\}$ *as in the HMT model (Def. 7). For* $(L, \Pi) \in K$, *let* $n_{(L,\Pi)} = 1$ *and* $down(L, \Pi) = \{L' \mid L \cap L' = \emptyset\}$. *The* Invariant-Preserving Model *is then defined by the same ground type lcpo's as the HMT model, relationally structured by*

R_τ	*as in Def. 7,*	*for all ground types* τ.
$R_{(L,\Pi)}(l)$	*iff* $l \notin L$.	
$R_{(L,\Pi)}(f)$	*iff* $\forall s.\ (\pi(s) \wedge f(s) \neq \bot) \Rightarrow \pi(f(s))$,	i.e., *every* $\pi \in \Pi$ *is an* invariant *of* f.
$R_{(L,\Pi)}$	$\equiv$ *true,*	*for the other ground types* τ.

Again we get a model of the simply typed λ-calculus, in which all ALGOL constants can be given their intended interpretations and in which $support(d)$ can be defined as the smallest set L such that $d \in D^\tau_L$. It has the additional property that $R^\tau_{(L,\Pi)}(d)$ holds whenever $L \cap support(d) = \emptyset$. A particular instance of this property is:

> Let g be of type $Prog \rightarrow Prog$ and f of type $Prog$. If $\pi \in Pred(Loc - support(g))$ is an invariant of f, then π is also an invariant of $g(f)$.

This is the reasoning principle which we have applied to Example 5.

We get only half-full abstraction in Theorem 5 because, in contrast to the first-order sublanguage, an element $d \in D^\tau_L$ (where τ is a PASCAL procedure type) is not necessarily equal to an lub of definable elements in D^τ_L but is only *bounded above* by such an lub.

Cambridge, Massachusetts, USA
October 20, 1987

OFFICIAL DISTRIBUTION LIST

Director
Information Processing Techniques Office
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

2 Copies

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217
Attn: Dr. R. Grafton, Code 433

2 Copies

Director, Code 2627
Naval Research Laboratory
Washington, DC 20375

6 Copies

Defense Technical Information Center
Cameron Station
Alexandria, VA 22314

12 Copies

National Science Foundation
Office of Computing Activities
1800 G. Street, N.W.
Washington, DC 20550
Attn: Program Director

2 Copies

Dr. E.B. Royce, Code 38
Head, Research Department
Naval Weapons Center
China Lake, CA 93555

1 Copy

Dr. G. Hooper, USNR
NAVDAC-OOH
Department of the Navy
Washington, DC 20374

1 Copy

END
DATE
FILMED
5-88
DTIC