CMU-CS-85-149

A MECHANIZABLE THEORY FOR EXISTENCE PROOFS OF INCLUSIVE PREDICATES [1]

Ketan Mulmuley
Computer Science Department,
Carnegie-Mellon University,
Schenley Park,
Pittsburgh, PA-15213,
U.S.A.

[1]The research reported in this document was supported in part by funds from the Computer Science Department of Carnegie-Mellon University, and by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539. The views and conclusions contained in it are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

1. Introduction

The Scott-Strachey approach to language semantics is well known. In this approach, a language is given semantics by mapping each language construct to its meaning in an appropriately constructed mathematical domain – this map is called denotational semantics of the language. The map is denotational in the sense that a denotation of a complex term depends only on the denotations of its constituents and not their structure. Of course, if the language is to be of any use at all, it must have an operational semantics. Generally this operational semantics is specified in terms of a compiler or an interpreter.

The question which arises is: are the denotational and operational semantics equivalent? This is the problem of semantic equivalence. Milne and Reynolds gave a general technique for proving such semantic equivalences. Their technique involves constructing certain predicates, called inclusive predicates, which connect the domains used for denotational and operational semantics. The most difficult part of their technique is showing the existence of such inclusive predicates. Unfortunately these existence proofs are known to be quite complicated, hence one is reluctant to carry out these proofs. Moreover, so far nobody has known any nontrivial example of a system of equations over inclusive predicates which does not have a solution. In the absence of such a counterexample it does not not seem justified to carry out the complicated existence proofs with all their details. In this paper we shall construct such a counter example through diagonalization. This means one must carry out the existence proofs with care. For quite a long time many people had expressed a need for much simpler methods for proving these existences. For example, several people suggested that there might be a language which can define most of the frequently arising predicates. Unfortunately the above mentioned counterexample shows that there are some fundamental difficulties in doing this. However, in this paper we shall give a theory for proving inclusive predicate existence which has a distinct advantage of being mechanizable. This means a computer can now carry out a large chunk of details. In fact, a system, which we shall call IPL (Inclusive Predicate Logic), was implemented on top of LCF which can almost automatically prove the existence of most of the inclusive predicates which arise in practice.

2. The Problem Of Inclusive Predicate Existence

Let us see where inclusive predicates enter into picture in Milne's or Reynolds' technique.

To put the problem in somewhat abstract setting, let the language be L, and let $\mathcal{E}_1$ and $\mathcal{E}_2$ be two different semantics of the language which map L into D_1 and D_2 respectively. Our problem is to show that $\mathcal{E}_1$ and $\mathcal{E}_2$ are equivalent. Towards this end, one might start by constructiong a predicate P which relates equivalent values from two domains D_1 and D_2, i.e., $(d_1, d_2) \in P$ iff d_1 and d_2 are in some sense equivalent. Then we can try to show that, for all $l \in L$, $(\mathcal{E}_1(l), \mathcal{E}_2(l)) \in P$. But, what is the exact nature of this predicate and does such a predicate always exist? We shall see later that this predicate existence problem is indeed a difficult problem. Sometimes one might simply be unlucky, and the predicate relating equivalent values from different domains might not exist. However, it might be the case that a weaker predicate Q exists such that $(d_1, d_2) \in Q$ iff d_1 is weaker

than d_2 in some sense. One then proves the equivalence of the two semantics by proving the following two assertions independently.

1. $\mathcal{E}_1(l)$ is weaker than $\mathcal{E}_2(l)$, for all $l \in L$.
2. $\mathcal{E}_2(l)$ is weaker than $\mathcal{E}_1(l)$, for all $l \in L$.

The key problem is then the construction of an appropriate predicate and proving its existence.

3. A Simple Language

To make the ideas concrete, we shall consider a simple programming language and its semantic equivalence problem. This example has been taken from [Stoy1]. The language to be described is basically a typed lambda calculus with atoms. The set of expressions in this language is defined as:

e	$:=$	I	identifiers
	$:=$	b	basic constants
	$:=$	$(\lambda I.e_1)e_2$	lambda abstraction applied to an argument.

We have not allowed lambda abstractions as first class citizens to simplify the discussion. The scope rule of this language is intended to be dynamic, i.e, free variables of a procedure get bound at the invocation time.

We shall first give an operational semantics to the language. Let Ide be the syntactic domain of identifiers, B be the domain of basic values, E be the syntactic domain of expressions and F be the domain of syntactic environments: $F = Ide \rightarrow E$. Then an operational semantics $O : E \rightarrow F \rightarrow B$ is defined as follows. For any $f \in F$,

$$\begin{aligned} O(I)f &= O(fI)I \\ O(b)f &= b \\ O((\lambda I.e_1)e_2) &= O(e_1)(f[e_2/I]), \end{aligned}$$

where $f[e_2/I]$ is a syntactic environment which is exactly like f except that $f[e_2/I]I = e_2$.

Note that in the third clause, when a function is invoked, its argument is not evaluated but, rather, the unevaluated expression is carried around. This is a usual trick used to implement dynamic scope.

Next we give a denotational semantics to the language. Let D be the domain of values and C be the domain of enviroments. As the scoping is dynamic, the value of an expression will be a function which will take the invocation environment as an argument and produce a basic value as the result, i.e, $D = C \rightarrow B$. And, of course, $C = Ide \rightarrow D$. More formally, D and C are the least domains satisfying the domain equations:

$$\begin{aligned} D &= C \rightarrow B \\ C &= Ide \rightarrow D. \end{aligned}$$

A denotational semantics $\mathcal{E} : E \rightarrow C \rightarrow B$ is given by the following clauses. For any $c \in C$,

$$\begin{aligned} \mathcal{E}(b)c &= b \\ \mathcal{E}(I)c &= (cI)c \\ \mathcal{E}((\lambda I.e_1)e_2) &= \mathcal{E}(e_1)(c[\mathcal{E}(e_2)/I]), \end{aligned}$$

where $c[\mathcal{E}(e_2)/I]$ is an environment in C which is exactly like c except that $c[\mathcal{E}(e_2)/I]I = \mathcal{E}(e_2)$.

4. Semantic Equivalence

Now that we have the denotational semantics E and the operational semantics O for our language, we must show that both these semantics are in some sense equivalent, i.e, they are in effect saying the same thing. Let us define a semantification function $\bar{}$ which takes a syntactic environment as an argument and produces as result its semantic image. Formally $\bar{} : F \rightarrow C$ is defined as follows. For any $f \in F$,

$$\bar{f} = \lambda(I : Ide).\mathcal{E}(fI).$$

Now showing that $\mathcal{E}$ and $\mathcal{O}$ are equivalent amounts to showing:

$$\mathcal{E}(e)\bar{f} = \mathcal{O}(e)f \text{ for all } e \in E \text{ and } f \in F.$$

It is easy to prove (see [Stoy1]) by an induction on the number of steps taken by the interpreter specified by $\mathcal{O}$ that:

$$\mathcal{O}(e)f \sqsubseteq \mathcal{E}(e)\bar{f} \text{ for all } e \in E \text{ and } f \in F.$$

Hence let us see how to prove

$$\mathcal{E}(e)\bar{f} \sqsubseteq \mathcal{O}(e)f \text{ for all } e \in E \text{ and } f \in F.$$

Towards this end; let us define two predicates $\Theta \subseteq D \times E$ and $\Pi \subseteq C \times F$ such that:

$$\begin{aligned} \Theta &= \{(d, e) \mid \forall (c, f) \in \Pi.dc \sqsubseteq \mathcal{O}(e)f\}, \\ \Pi &= \{(c, f) \mid \forall I \in Ide.(cI, fI) \in \Theta\}. \end{aligned} \tag{1}$$

Intituively $(d, e) \in \Theta$ if d is weaker than e, and $(c, f) \in \Pi$ if c is weaker than f. Immediately the question arises whether such predicates actually exist. Assume for the moment that they do. Then it is easy to prove by structural induction that (see [Stoy1]), for any $e \in E$ and $f \in F$, $(\mathcal{E}(e), e) \in \Theta$ and $(\bar{f}, f) \in \Pi$. This implies, by the definition of Θ, that $\mathcal{E}(e)\bar{f} \sqsubseteq \mathcal{O}(e)f$, which is what we wanted to prove.

Thus we proved the semantic equivalence of $\mathcal{E}$ and $\mathcal{O}$ but for one gap: we have to show that there exist Θ and Π such that (1) is satisfied. *Unfortunately that is the most difficult part.* Thus the major component of the proof still remains. In [Stoy1] this existence is shown using Milne's technique. The proof is undeniably complicated. We shall later how this existence can proved automatically by the implemented IPL system. For the moment we shall content ourselves with the knowledge that such predicates do exist.

However, one might ask if the issue of existence of inclusive predicates is really so sensitive as to demand this much of care and trouble. In fact, though Milne and Reynolds gave general methods for proving such existences, not many such proofs are found in the

literature and one senses reluctance to go through such proofs. There are two reasons for this reluctance.

1. There has not been given before any nontrivial equation over predicates which does *not* have a solution. In the absence of counter examples it does not seem justified to go through such tedious existence proofs.
2. Both Reynolds' and Milne's techniques have certain disadvantages. Though Reynolds' technique is very systematic, it can be used only when two domains have similar shapes or when one domain can be 'transformed' into the other domain. This technique then can not be used to prove the existence of a solution to (1). Milne's technique, on the other hand, is a general scheme, but the particular instantiation of this scheme in a specific situation can seem ad hoc making the proofs complicated. This justifies the reluctance even more! (In a recent private communication, Milne has told the author that in many special cases the details of his method can actually be bypassed. However, one still feels need for a *uniform* way of doing things.)

In this paper we shall provide answers to both these arguments. As for the first argument we shall provide in the next section a counter example using diagonalization through self application, thus dissolving any doubt about the nontriviality of the existence problem. As an answer to the second argument, we shall give a theory to prove such existences which has a distinct advantage of being mechanizable. In fact a system called IPL (Inclusive Predicate Logic) was implemented on top of LCF which mechanizes and automates this theory.

5. Diagonalization And Self Application

It is tempting ask if the predicates Θ and Π exist even when $\sqsubseteq$ in the definition of Θ (see (1)) is replaced by $=$, because then it will be possible prove that $\mathcal{E}(e)\bar{f} = \mathcal{O}(e)f$, for all $e \in E$ and $f \in F$, without having to break the proof in two parts as we did. If this replacement is carried out, the 'new' Θ and Π will satisfy the equations:

$$\begin{aligned} \Theta &= \{(d,e) \mid \forall (c,f) \in \Pi . dc = \dot{\mathcal{O}} ef\}, \\ \Pi &= \{(c,f) \mid \forall I \in Ide . (cI, fI) \in \Theta\}. \end{aligned} \tag{2}$$

Whether these predicate exist or not is an open question. The question was raised in [Stoy1]. However, we shall answer this question partially below.

We shall show later that a solution to (1) exists for *any* $\mathcal{O}$. (See [Stoy1] too.) What we shall show here is that this is not true in the case of (2). That is to say, we shall show, using diagonalization, that there exists an $\mathcal{O}$ such that (2) has no solution.

To simplify the discussion, assume that there is only one identifier, i.e., Ide is a trivial one point domain $\perp$. In that case

$$\begin{aligned} C &= Ide \to D = \perp \to D \simeq D, \\ D &= C \to B \simeq D \to B, \\ F &= Ide \to E \simeq E, \end{aligned}$$

and hence Θ can be identified with Π. The definition of Θ can now be simplified to:

$$\Theta = \{(d,e) \mid \forall (d',e') \in \Theta.\ dd' = \mathcal{O}(e)e'\}.$$

Let $\mathcal{O} = \lambda e \lambda e'.b$, where b is an element of B such that $b \neq \perp$. (We are assuming that B is not trivial). Then

$$\Theta = \{(d,e) \mid \forall (d',e') \in \Theta.\ dd' = b\}.$$

As expressions do not play any role now, the existence of Θ is equivalent to the existence of $\pi \subseteq D$, where $D = D \to B$ and

$$\pi = \{d \mid \forall d' \in \pi.\ dd' = b\}.$$

We show that such a π does not exist. Let $f : D = \lambda x.xx$. Now consider two cases.

1. $f \in \pi$. Then, from the definition of π, it follows that $ff = b \neq \perp$. But it can be shown, as in [Park], that $ff = \perp$, which is a contradiction.
2. $f \notin \pi$. Let d be an arbitrary element of π. Then, as $d \in \pi$, it follows from the definiton of π that $dd = b$. Hence $fd = (\lambda x.xx)d = dd = b$. Hence, for all $d \in \pi$, $fd = b$, which means $f \in \pi$. But this is again a contradiction.

We conclude that such a π does not exist.

If we consider $\sqsupseteq$ instead of $=$ exactly the same reasoning goes through. Note that $ff = (\lambda x.xx)(\lambda x.xx)$ is our usual friend from combinator theory. And we suceeded in diagonalization with its help.

This example shows that the inclusive predicate existence problem is very sensitive to the syntax. What this means is that there can not be a rich enough *purely syntactic* language such that *any* predicate expressed in that language exists. Secondly note that Reynolds' method can not be used to prove the existence of Θ and Π satisfying (1) as, using his terminology, no relational functor corresponding to this equation can be constructed in his theory.

6. An Overview Of Domain Theory

Before we proceed to the theory of existence proofs, we recall in this sections the major concepts of Scott's domain theory. (See [Scott1]) By domain we shall mean an algebraic, consistently complete cpo. If C and D are domains then we shall denote by $C \to D$ the domain of continuous functions from C to D. $C \times D$ and $C + D$ will denote the product and disjoint sum of C and D. It is possible to regard $\to$, $\times$, and $+$ as the functors on the category of domains in which continuous functions are taken as morphisms.

There is a natural notion of embedding among domains. Given two domains C and D, we say that C can be embedded in D if there exists an injection-projection pair (i,j), where $i \in C \to D$ and $j \in D \to C$, such that $j \circ i = I_C$, and $i \circ j \sqsubseteq I_D$. We say $C \lhd^i_j D$ or simply $C \lhd D$ when the embedding pair is implictly understood. Pictorially we shall represent an embedding as follows:

$$C \underset{j}{\overset{i}{\rightleftarrows}} D$$

There is good way to internalize the notion of embeddings: through finitary projections. We shall introduce them soon.

We say $f \in D \to D$ is a retract of D if f is idempotent, i.e., $f \circ f = f$. Note that the fixpoint set of f is a cpo; we shall denote it by $|f|$. One can think of the retract f as a *notation* for its fixpoint cpo $|f|$. Hence, we shall use the notations f and $|f|$ interchangably, when no ambiguity arises. Thus, $x \in f$ actually means $x \in |f|$.

A projection is a special kind of retract. A retract f of D is called a projection if $f \sqsubseteq I$. It is finitary if, in addition, the set of its fixpoints is isomorphic to a domain.

Now if a domain C can be embedded in a domain D, i.e. $C \lhd^i_j D$, then C is isomorphic to the fixpoint set of the finitary projection $i \circ j$ of D. Conversely, every finitary projection f of a domain D comes from such an embedding, namely, $|f| \lhd^I_f D$. Thus *a domain C can be embedded in D iff it is isomorphic to the fixpoint set of some finitary projection of D.*

If D is a domain, let us denote by $\bar{D}$ the set of finitary projections of D. Scott shows in [Scott1] that $\bar{D}$ is a domain again. In fact, $\bar{D}$ comes from a finitary projection of $(D \to D)$, i.e., there exists a finitary projection f of $(D \to D)$ such that $\bar{D} = |f|$.

If f is a finitary projection of D then we shall say that $|f|$ is a subdomain of D. The embedding relation on subdomains of D gets translated to the relation $\sqsubseteq$ on $\bar{C}$: *if f and g are finitary projections of D then $f \sqsubseteq g$ iff $|f|$ is a subdomain of $|g|$, i.e., $|f| \subseteq |g|$ and f is a finitary projection of $|g|$.*

Now one sees that a finitary projection can be looked upon as just a *notation* for a subdomain and it captures in a nice way the notion of embedding. Even more force gets attached to this argument by the existence of a domain U (see [Scott1]) which is universal in the sense: *every domain D is isomorphic to a subdomain of U, i.e. the fixpoint set of some finitary projection of U.*

Let us give $\bar{U}$, the set of finitary projections of U, a special name V. Universality of U means that the category of domains D can be 'internalized' in U in the form of V. In fact, we shall call the elements of V domains. Thus, when refer to a 'domain' $D \in V$, what we are referring to is actually its fixpoint set $|D|$. This duality between finitary projections and domains is of central importance, and will be assumed implicitly throughout this paper.

After allowing so much of confusion between finitary projections and domains, one would think this is enough. Well, not yet. To complete this confusion, we have to internalize the functors $\to$, $\times$, $+$ on the category of domains as well. We shall soon see that corresponding these functors there are appropriate *continuous* functions $\to, \times, + \in V \times V \to V$. (We are using the same notation for the internalizations of the functors as well.)

First note that, by universality of U, the domains $U \to U$ is isomorphic to a subdomain of U, i.e., there exists an injection-projection pair $(i_{\to}, j_{\to})$ between $U \to U$ and U:

$$(U \to U) \underset{j_{\to}}{\overset{i_{\to}}{\rightleftarrows}} U$$

Now given $a, b \in V$, define $(a \to b) \in U \to U$ by:

$$a \to b = i_{\to} \circ (\lambda f.(b \circ f \circ a)) \circ j_{\to}.$$

Obviously $\rightarrow$ is continuous. Moreover, it can be shown that $(a \rightarrow b) \in V$, and $|a \rightarrow b| \simeq |a| \rightarrow |b|$. This means $\rightarrow \in V \times V \rightarrow V$. With the functors $\times$ and $+$ one can similarly associate continuous functions $\times, + \in V \rightarrow V$. Thus we have 'internalized' the category of domains in the form of V and the continuous functions $\rightarrow, \times, + \in V \rightarrow V$.

As $\rightarrow, \times, +$ are all continuous functions on V, it becomes possible now solve the domain equations like:

$$D = (D \rightarrow B) + (D \times D),$$

where $B \in V$ is some fixed domain. In fact, all such domain equations have the least solutions. For example, the least solution of the above domain equation is

$$fix(\lambda D \in V.(D \rightarrow B) + (D \times D)),$$

where fix denotes the continuous fixpoint operation. Mutual recursion poses no problem due to the presence of the product construction.

7. Outline Of The Theory

In Section 5 we have seen that the inclusive predicate existence problem is indeed nontrivial. What we want is a theory which could be mechanized on computer so that a large chunk of the existence proof could be automated. Though Milne's technique to prove the existence of an inclusive predicate is general enough, the specific instantiation of his scheme to the case at hand can be ad hoc making the proofs complicated, and, moreover, these proofs can not be mechanized. On the other hand, as we remarked earlier, though Reynold's technique in this regard is more systematic, it suffers from the disadvantage that it is not always applicable. The theory we give in this section has a distinct advantage of being mechanizable. This means, though the existence proof will, by no means, be easy, a *computer* can go through most of the details.

We shall informally and crudely sketch the issues and main ideas involved by considering a simple example. Suppose we are given a domain D which is the least domain satisfying $D = T(D)$, where T is a continuous domain constructor, i.e., $T \in V \rightarrow V$. And we want to know if there exists a predicate P on domain D such $P = w(P)$, where w is some predicate transformation.

Let a L be a retract of V such that

$$\{\perp, T(\perp), T^2(\perp), \cdots, D\} \subseteq |L|.$$

We shall see in Section 13 that, for all the domain constructors T which arise in practice, it is possible to construct such a retract a uniform way. However, it need *not* be the case that:

$$\{\perp, T(\perp), T^2(\perp), \cdots, D\} = |L|.$$

Roughly L encapsulates the process of inverse limit construction of the domain D. Once this is done we shall construct on L a predicate cpo $\mathcal{L}$:

$$\mathcal{L} = \{(C, Q) \mid \begin{array}{l} C \in L \text{ and } Q \text{ is a directed complete pred-} \\ \text{icate on } C \text{ satisfying some constraints } \}. \end{array}$$

(A predicate P is called directed complete, if the limit of every monotone chain in P belongs to P.) Thus each element of $\mathcal{L}$ is a pair consisting of a domain which belongs to L and a predicate on this domain satisfying some constraints. We shall see the precise nature of the above mentioned constraints later on.

Once such predicate cpos are constructed, it natural to construct functions on them, which we shall call predicators (named after the similar predicators of Milne). As each element of $\mathcal{L}$ is a domain-predicate pair, it is reasonable to assume that a predicator will also be a pair consisting of a transformation on domains and a transformation on predicates. So let (T, w) be a predicator from $\mathcal{L}$ to $\mathcal{L}$, where T and w are the domain and predicate transformations as in the previous section. Pictorially:

$$\mathcal{L} \xrightarrow{(T,w)} \mathcal{L}$$

If we can somehow show that (T, w) is continuous then we can take the least fixpoint of (T, w):

$$(C, Q) = (T, w)(C, Q);$$

where domain $C \in L$ and Q is a predicate on C. This means:

$$\begin{aligned} C &= T(C), \text{ and} \\ Q &= w(Q). \end{aligned}$$

But, as D is the least fixed point of T, this implies that $C = D$, hence Q is actually a predicate on D. Thus Q is the required solution of the equation $P = w(P)$.

The question is: how does one prove that (T, w) is continuous? Here is the crucial point: *There exist an algorithm to generate sufficient goals to guarantee continuity of such predicators. All of these goals can be proved within a computer– in fact, most of them automatically. Thus, by employing this reduction algorithm, computer can reduce the goal proving continuity of a predicator – and hence, that of inclusive predicate existence – to the goals which it can generally prove itself, and sometimes with some user assistance. This transfers the burden specific details of an existence proof to a computer.*

Let us see how we can go about proving continuity of (T, w). It will definitely depend upon the structure of w, but, what is more important, it will also depend upon the exact structure of $\mathcal{L}$. Remember that each element of $\mathcal{L}$ is of the form (C, Q), where $C \in L$ and Q is a predicate on C satisfying some constraints. The significance of these constraints can now be made clear: these constraints critically determine continuity of (T, w). These constraints are what we shall call upward and downward closure. For the convenience of future reference, we shall state a general definition:

An n-ary relation Q on a domain C is said to be upward (downward) closed in the ith argument if $(x_1, \ldots, x_i, \ldots, x_n) \in Q$ and $x_i \sqsubseteq y_i$ ($y_i \sqsubseteq x_i$) implies $(x_1, \ldots, y_i, \ldots, x_n) \in Q$.

In the unary case, as the one we are considering in this section, these constraints are not important, however, otherwise they are. For example, note that $\sqsubseteq$ is upward closed in the second argument and downward closed in the first, $\sqsupseteq$ is upward closed in the first argument and downward closed in the second, whereas $=$ is neither upward closed nor

downward closed in any of the arguments. Remember that in section 5 we proved that if $\sqsubseteq$ in (1) is replaced by $=$ or $\sqsupseteq$ the equation need not have a solution. The sensitivity in this example to syntax can be traced to the different upward and downward closure properties of $\sqsubseteq$, $=$ and $\sqsupseteq$. It then is not surprising that continuity of predicators should depend on these properties too.

Returning back to our unary case, we now have discussed the important factors which determine continuity of the predicator (T, w). In Section 11 we shall give a crucial reduction algorithm which will generate sufficient goals to guarantee continuity of (T, w). As we said before, it is crucial because the goals generated by it can be proved within the LCF formalism, and this is what makes the theory mechanizable.

Remember that we said continuity of (T, w) depends on the structure of $\mathcal{L}$. But the structure of $\mathcal{L}$ in turn depends on that of L. Hence, one expects that some of these generated goals would depend upon L. In section 13 we shall see how L can actually be specified as the least fixed point of some easily constructible higher order functional. This makes it possible to prove the validity of such goals using fixpoint induction in LCF.

In the succeeding sections we shall put this whole theory on a formal footing.

8. Predicate CPOs

Our discussion in the preceding section motivates us to define 'predicate cpo' as follows.

Let L be a retract on V^n, where V is the domain of finitary projections of the universal domain U, (thus an element of $|L|$ will be an n-tuple of finitary projections) and let $A, B \subseteq \{1, \ldots, n\}$ be sets of indices. Let P be a directed complete n-ary predicate on $\perp_L$, the bottom domain of L, which is upward closed in the indices of A and downward closed in the indices of B. Then a predicate cpo, $\mathcal{L} = (L, A, B, P)$, on L is a set of elements (C, Q) satisfying .

1. C is a domain such that $C \in L$ (i.e. C is in $|L|$, the fixpoint set of L).
2. Q is a directed complete predicate on C, i.e., $Q \subseteq |C|$.
3. Q is upward closed in all of the indices of A and downward closed in all the indices of B.
4. $\perp_L(Q) = P$ and $P \subseteq Q$.

Significance of the fourth condition will be made clear soon. We shall say that L is the underlying retract of $\mathcal{L}$. Often when A and B are empty sets, we shall write $\mathcal{L} = (L, P)$ instead of $\mathcal{L} = (L, \{\,\}, \{\,\}, P)$.

We convert $\mathcal{L}$ into a partial order as follows. We say $(C, Q) \sqsubseteq (D, R)$, where $(C, Q), (D, R) \in \mathcal{L}$, iff

1. $C \sqsubseteq D$,
2. $C(R) = Q$ and $Q \subseteq R$.

Remember from our discussion in Section 6 that the first condition is equivalent to saying: the domain $|C|$ is a subdomain of the domain $|D|$. The second condition says that Q can be embedded into R. This second condition is equivalent to the following one:

2'. $C(R) \subseteq Q$ and $Q \subseteq R$,

because, if $Q \subseteq R$ then

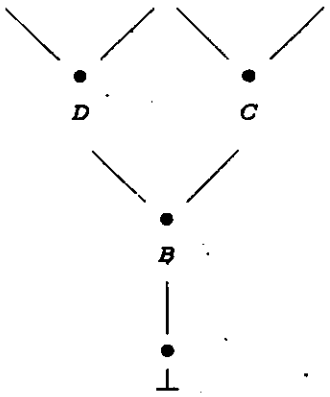
$$\begin{aligned} Q &= C(Q) \quad \text{as } Q \subseteq |C|, \\ &\subseteq C(R). \end{aligned}$$

Hence, if $Q \subseteq R$ then $C(R) \subseteq Q$ implies $C(R) = Q$, and the converse is trivial. We shall use this alternate, simpler formulation of the second condition on many occasions.

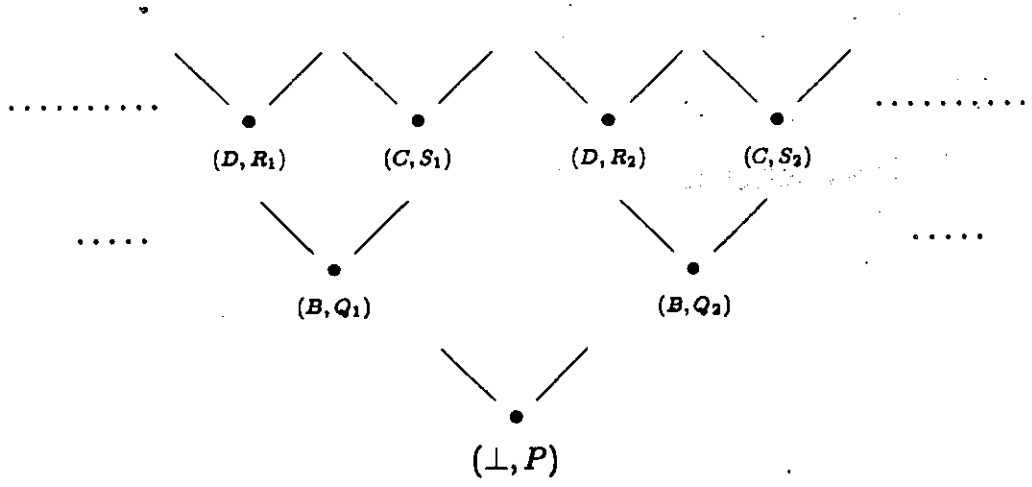
In the light of this partial ordering, the fourth condition in the definition of $\mathcal{L}$ can be seen to be equivalent to:

$$(\bot_L, P) \sqsubseteq (C, Q) \text{ for every } (C, Q) \in \mathcal{L}.$$

As an example, suppose L looks as shown below:



Then $\mathcal{L} = (L, A, B, P)$ can be visualized as follows:



Note that if (C, Q) and $(C, R) \in \mathcal{L}$ then

$$(C, Q) \sqsubseteq (C, R) \text{ implies } Q = R.$$

This is because

$$\begin{aligned} R &= C(R) && \text{as } R \subseteq |C|, \\ &= Q && \text{as } (C,Q) \sqsubseteq (C,R). \end{aligned}$$

Theorem 8.1: $\mathcal{L} = (L, A, B, P)$ is a cpo. Further, $(\perp_L, P)$ is the least element of this cpo.

Proof: Let

$$X = \{(C_j, Q_j) \mid j \in J\},$$

be a directed subset of $\mathcal{L}$, where J is a directed index set. Then $Y = \{C_j \mid j \in J\}$ is an indexed directed set of $|L|$. Hence, Y has a least upper bound $C \in L$. Let us define a predicate Q on C as:

$$Q = \{x \in C \mid \text{ for all } j \in J.\ C_j(x) \in Q_j\}.$$

We claim that (C, Q) is the least upper bound of X in $\mathcal{L}$.

First we have to show that $(C, Q) \in \mathcal{L}$. We shall show this later. For the moment assume that $(C, Q) \in \mathcal{L}$.

Let us show that (C, Q) is an upper bound of X, i.e., for all $j \in J$, $(C_j, Q_j) \sqsubseteq (C, Q)$. This entails showing three things.

1. $C_j \sqsubseteq C$. This is obvious.
2. $C_j(Q) \subseteq Q_j$. This follows immediately from the definition of Q.
3. $Q_j \subseteq Q$. Let $x \in Q_j$ be an arbitrary element. We have to show that $x \in Q$, i.e., $C_i(x) \in Q_i$, for all $i \in J$. As J is directed, there exists $k \in J$ such that $i, j \sqsubseteq k$. Then $(C_j, Q_j) \sqsubseteq (C_k, Q_k)$, and hence $Q_j \subseteq Q_k$. On the other hand $(C_i, Q_i) \sqsubseteq (C_k, Q_k)$ implies $C_i(Q_k) \subseteq Q_i$, which, as $Q_j \subseteq Q_k$, implies $C_i(Q_j) \subseteq Q_i$. As $x \in Q_j$, this means $C_i(x) \in Q_i$. Thus we have shown that, for all $i \in J$, $C_i(x) \in Q_i$ which means $x \in Q$.

Next we have to show that (C, Q) is the least upper bound of X. Let (D, R) be any other upper bound of X. We have to show that $(C, Q) \sqsubseteq (D, R)$. Again we have to show three things.

1. $C \sqsubseteq D$. It is clear that D is an upper bound of Y. However, as C is the least upper bound of Y, it follows that $C \sqsubseteq D$.
2. $C(R) \subseteq Q$. Let $x \in R$. Then

   $$C(x) = lub\{C_j(x) \mid j \in J\}.$$

   But, for each $j \in J$, as $(C_j, Q_j) \sqsubseteq (D, R)$, we conclude that $C_j(x) \in Q_j$. As $C_j \sqsubseteq C$, we have $C_j(C(x)) = C_j(x)$. Hence $C_j(C(x)) \in Q_j$ for all $j \in J$. From the definition of Q it follows that $C(x) \in Q$.
3. $Q \subseteq R$. Let $x \in Q$. As C is the *lub* of $\{C_j \mid j \in J\}$, we know that

   $$x = lub\{C_j(x) \mid j \in J\}.$$

From the definition of Q, for each $j \in J$, $C_j(x) \in Q_j$. However, as $(C_j, Q_j) \sqsubseteq (D, R)$, we know that $Q_j \subseteq R$ and hence $C_j \in R$. By directed completeness of R it now follows that

$$x = lub\{C_j(x) \mid j \in J\} \in R.$$

Now we let us show that $(C, Q) \in \mathcal{L}$ indeed. For this we have to verify four conditions.

1. $C \in |L|$. But we have already seen this.
2. Q is directed complete. Let $Z = \{z_i \mid i \in I\}$, where I is a directed index set, be a directed subset of Q. Let z be the least upper bound of Z. We want to show that $z \in Q$. For each $j \in J$, by continuity of C_j it follows that:

   $$C_j(z) = lub\{C_j(z_i) \mid i \in I\}.$$

   Now, as each $z_i \in Q$, $C_j(z_i) \in Q_j$ for all j. By directed completeness of Q_j it follows that $C_j(z) \in Q_j$. Thus, for all j, $C_j(z) \in Q_j$. Hence, by the definition of Q it follows that $z \in Q$.

   This proves that Q is directed complete.
3. Q is upward closed in the indices of A. Let $i \in A$. Suppose

   $$x = (x_1, \ldots, x_i, \ldots, x_n) \in Q$$

   and $x_i \sqsubseteq \bar{x}_i$. We want to show that $\bar{x} = (x_1, \ldots, \bar{x}_i, \ldots, x_n) \in Q$. For each j, let

   $$\begin{array}{l}(x_1^j, \ldots, x_i^j, \ldots, x_n^j) = C_j(x) \quad \text{and} \\ (\bar{x}_1^j, \ldots, \bar{x}_i^j, \ldots, \bar{x}_n^j) = C_j(\bar{x}).\end{array}$$

   Note that x_l^j and $\bar{x}_l^j$ need *not* be equal for $l \neq i$. As $C_j(x) \in Q_j$ and Q_j is upward closed in the ith index, we conclude that

   $$(x_1^j, \ldots, \bar{x}_i^j, \ldots, x_n^j) \in Q_j.$$

   As we have already shown that $Q_j \subseteq Q$,

   $$(x_1^j, \ldots, \bar{x}_i^j, \ldots, x_n^j) \in Q.$$

   Now by directed completeness of Q,

   $$\bar{x} = (x_1, \ldots, \bar{x}_i, \ldots, x_n) = lub\{(x_1^j, \ldots, \bar{x}_i^j, \ldots, x_n^j) \mid j \in J\} \in \overline{Q}.$$

   Hence Q is indeed upward closed in the ith index for every $i \in B$.

   The case of downward closedness is similar.
4. $(\bot_L, P) \sqsubseteq (C, Q)$. But we have already shown that $(C_j, Q_j) \sqsubseteq (C, Q)$, for every $j \in J$. However, as $(C_j, Q_j) \in \mathcal{L}$, we know that $(\bot_L, P) \sqsubseteq (C_j, Q_j)$. Hence by transitivity the result follows.

Thus $(C, Q) \in \mathcal{L}$.

Finally, it is trivial to see that $(\bot_L, P)$ is the least element of $\mathcal{L}$. ∎

9. Predicators

Once we have constructed predicate domains the next step is to construct functions on them, which we shall call predicators. Because every element of a predicate cpo is a pair consisting of a domain and a predicate on it, it is natural to assume that a predicator too will be a pair consisting of a transformation on domains and a transformation on predicates.

Let $\mathcal{L}_i$, $(1 \leq i \leq k)$, be predicate domains on retracts L_i of V^{n_i}, and let $\mathcal{M}$ be a predicate domain on a retract M of V^n. We want to construct a predicator from $\mathcal{L}_1 \times \cdots \times \mathcal{L}_k$ to $\mathcal{M}$. This predicator will be a pair consisting of a domain transformation and a predicate transformation. Let $(C_i, R_i) \in \mathcal{L}_i$, for $i = 1$ to k, be k arguments. Then the first component of the predicator will be a continuous domain transformation, $T \in V^{n_1} \times \cdots \times V^{n_k} \to V^n$, which will map $C_i \in |L_i|$ to a domain $D = T(C_1, \ldots, C_n) \in |M|$. The second map will be a predicate transformation which will map the predicates R_i on C_i to some predicate S on D such that $(D, S) \in \mathcal{M}$. If we want our theory to be mechanizable, there must be a way representing this predicate transformation in a computer. We shall soon give a first order language which can be used to specify a predicate transformation.

The basic idea is as follows. We shall construct a formula w with n free variables $v_1, \ldots, v_n$. Each of these variables should be taken as ranging over the universal domain U. Let (X_i, Y_i), $1 \leq i \leq k$, stand for a variable domain-predicate pair from $\mathcal{L}_i$, where the variable Y_i stands for a predicate on the domain $X_i \in L_i$. As the formula w is going to represent a predicate transformation, it will naturally depend on these k arguments $(X_i, Y_i) \in \mathcal{L}_i$. The predicate S on the domain $D = T(C_1, \ldots, C_k) \in V^n$ can then be defined as the set of all $(v_1, \ldots, v_n) \in |D|$ which 'satisfy' w when X_i is 'interpreted' as C_i and Y_i is 'interpreted' as R_i, for all $(1 \leq i \leq k)$.

We shall put this in formal terms now.

9.1. A Language For Predicate Transformation

Let us fix for this section $(X_i, Y_i) \in \mathcal{L}_i$, $(1 \leq i \leq k)$, to be k domain-predicate variable pairs which will stand for the k arguments of a predicator; here X_i denotes an element in L_i and Y_i is to be interpreted as a predicate on X_i. We shall let (X, Y) stand for the tuple $((X_1, Y_1), \ldots, (X_k, Y_k))$ and X stand for the tuple $(X_1, \ldots, X_k)$.

The set of domain terms over X, written $\Gamma = \Gamma(X)$, is defined to be the set of all lambda calculus terms $t \in V$ which are built from the domain variables $X_1, \ldots, X_k$, and possibly some constant symbols.

Examples:

1. $(X_1 \to Int) \in \Gamma$. Here $\to : V \times V \to V$ is a constant symbol which stands for the domain exponentiation function. $Int : V$ is a constant symbol standing for the domain of integers.
2. $((X_1 \times X_2) + Bool) \to X_1 \in \Gamma$. Here $\times, + : V \times V \to V$ and $Bool : V$ are constant symbols to be interpreted in the standard way.

Intuitively a domain term in Γ will represent some domain in V when its free domain variables $X_1, \ldots, X_k$ and constant symbols are appropriately interpreted.

$\Theta = \Theta(X, Y)$ is the set of domain-predicate pairs as below:

$$\begin{aligned}\Theta = \; & \{ (X_i, Y_i) \mid 1 \leq i \leq k \} \cup \\ & \{ (D, \sqsubseteq), (D, =), (D, \sqsupseteq) \mid D \in \Gamma \} \cup \\ & \{ (E, Q) \mid Q \text{ is a predicate constant and } E \text{ is a} \\ & \text{domain constant } \}.\end{aligned}$$

Intuitively if $(D, P) \in \Theta$ then P will be a predicate on the domain D after an appropriate interpretation.

Let $\Delta = \Delta(X)$ be the set of all terms t such that t is a tuple over the universal domain (i.e. $t \in U^l$ for some l) and is built from constant symbols, $X_1, \ldots, X_k$, and some free variables which we shall assume to range over U. (Note that the occurrence of X_is in t is possible because of the duality between domains and finitary projections. Remember that $X_i \in V^{n_i}$. Hence, as $V \subseteq U \to U$, X_i can also be regarded as a function on the universal domain.)

Examples:

1. $(proj_1^{n_1}(X_1)x) \in \Delta$. Here $proj_1^{n_1}$ is a constant symbol which stands for the projection function which extracts the first component of an n_1-tuple. Hence $proj_1^{n_1}(X_1) \in V$. The term contains a free variable x which (by the convention established in the definition of Δ) is implicitly assumed to range over U, i.e. $x \in U$. Hence $(proj_1^{n_1}(X_1)x) \in U$.
2. $(j_{\to}x)y \in \Delta$, where the constant symbol $j_{\to} : U \to (U \to U)$ stands for the projection of U onto its function space, and where the free variables x, y range over U.
3. Let $n_1 = 2$. Then $(X_1(x, y)) \in \Delta$. Hence $X_1 \in V^2$. As the free variables $x, y \in U$, it follows that $(X_1(x, y)) \in U^2$.

Finally, $\Phi = \Phi(X, Y)$, the set of well formed formulae (wffs), is recursively defined as follows:

1. "$t \in P$ in D" $\in \Phi$,
   where $t \in \Delta$ and $(D, P) \in \Theta$.
2. "$\forall y \in D.\ g$", "$\exists y \in D.\ g$" $\in \Phi$,
   where y is a variable ranging over U, $D \in \Gamma$ and $g \in \Phi$.
3. "$f \wedge g$", "$f \vee g$", "$f \Rightarrow g$" $\in \Phi$,
   where $f, g \in \Phi$.

The notions of a free and a bound variable occurrences are as usual. As syntactic sugar, we define "$\forall x \in Y_i.\ g$" as a shortform for "$\forall x \in X_i.\ (x \in Y_i) \Rightarrow g$" and "$\exists x \in Y_i.\ g$" as a shortform for "$\exists x \in X_i.\ (x \in Y_i) \wedge g$". Similarly "$t \in Y_i$" will be a shortform for "$t \in Y_i$ in X_i". Apart from these we shall also allow the use of infix notation and obvious shortforms such as "$\forall (x_1, \ldots, x_{n_1}) \in X_1 \ldots$." and so on.

Examples:

1. Let $k = 1$ and $n_1 = 2$, i.e., we have only one domain-predicate variable pair $(X, Y) = (X_1, Y_1)$, where $X \in V^2$ and the variable Y stands for a predicate on X.

Then $w \in \Phi(X, Y)$, where

$$w = \text{“}\forall x \in proj_1^2(X).\ (X(x,z), \bot) \in \sqsupseteq \ \ \wedge (x,x) \in X\text{”}.$$

The wff w has one free variable ranging over U, namely z; one constant predicate symbol $\sqsubseteq$ which stands for the obvious predicate over U^2, and one more constant symbol $proj_1^2 : V^2 \to V$. It has one bound variable $x \in U$. For the sake of simplicity, using infix notation, we shall write w, simply as

$$\text{“}\forall x \in proj_1^2(X).\ X(x,z) \sqsupseteq \bot \ \wedge \ (x,x) \in X\text{”}.$$

2. Let $k = 2$, $n_1 = 2$ and $n_2 = 1$. We now have two domain-predicate variable pairs (X_1, Y_1) and (X_2, Y_2), where $X_1 \in V^2$ and $X_2 \in V$. Then $w \in \Phi(X, Y)$, where

$$w = \text{“}\exists (y,z) \in Y_1.\ X_2((j \to y)z) \sqsubseteq x \ \wedge \\ \forall u \in (proj_1^2(X_1)).\ (w,u) \in Y_1\text{”}.$$

9.2. Interpretation Of The Language

We define in this section how the language given in the last section can be interpreted. We shall denote this interpretation by $\Im$. We shall use the same symbol $\Im$ to denote the interpretation of the various syntactic classes of the language.

Interpretation of domain terms in $\Gamma(X)$*:*

Assume that

1. $\Im$ assigns to each X_i a domain $X_i^{\Im} \in V^{n_i}$.
2. $\Im$ assigns to each constant C that occurs in Γ an element $C^{\Im}$ of the appropriate type.

Then this uniquely determines $\Im$ on Γ, i.e., each domain term $D \in \Gamma$ can be uniquely assigned a domain $D^{\Im} \in V$, assuming that lambda calculus terms are interpreted in the standard way.

Interpretation of predicate-variable pairs in $\Theta(X, Y)$*:*

If $(D, P) \in \Theta$ then $\Im$ assigns to the predicate symbol P a predicate on the domain $D^{\Im}$ as follows.

1. $(D, P) = (X_i, Y_i)$,$(\ 1 \le i \le k)$: Then $\Im$ assigns to Y_i some predicate $Y_i^{\Im}$ on $X_i^{\Im}$.
2. P is $\sqsubseteq$: Then $\sqsubseteq$ is assigned the standard partial order predicate $\sqsubseteq$ on the domain $D^{\Im}$. The cases when P is $=$ or $\sqsupseteq$ are similar.
3. P is a constant predicate symbol: Then P is assigned some given predicate $P^{\Im}$ on the constant domain $D^{\Im}$.

Interpretation of the simple terms in $\Delta = \Delta(X)$*:*

Assume that $\Im$ assigns to every constant symbol c occurring in Δ terms an element $c^{\Im}$ of an appropriate type. (If c also occurs in Γ terms then, of course, both the interpretations must coincide.) Let s be an assignment function from a set of variables ranging over U into U. As the interpretation of the X_i, $(1 \le i \le k)$, have already been specified, it follows

that every $t : U^l$ of Δ can be uniquely assigned an interpretation $t^{\Im}[s] \in U^l$ relative to s, assuming that lambda calculus terms are interpreted in the standard way.

Interpretations of the wffs in $\Phi = \Phi(X, Y)$:

Now we can inductively define what it means for $\Im$ to satisfy a $w \in \Phi$ with s, $\Im \models w[s]$.

1. $\Im \models (t \in P \text{ in } D)[s]$ iff $D^{\Im}(t^{\Im}[s]) \in P^{\Im}$.
   (Note that the domain $D^{\Im}$ is playing the role of a finitary projection here.)
2. $\Im \models (\forall y \in D.g)[s]$ iff $\forall a \in D^{\Im}$, $\Im \models g[s(a/y)]$,
   where $s(a/y)$ is an assignment function exactly like s except that $s(a/y)(y) = a$.
3. $\Im \models (\exists y \in D.g)[s]$ iff $\exists a \in D^{\Im}$, $\Im \models g[s(a/y)]$.
4. $\Im \models (f \Rightarrow g)[s]$ iff $\Im \models f[s]$ implies $\Im \models g[s]$.
5. $\Im \models (f \vee g)[s]$ iff $\Im \models f[s]$ or $\Im \models g[s]$.
6. $\Im \models (f \wedge g)[s]$ iff $\Im \models f[s]$ and $\Im \models g[s]$.

For future convenience we shall establish some notation. Let w be a wff in Φ and t be a term in Δ whose free variables ranging over U are contained in the list $v_1, \ldots, v_n$. Let a be a tuple in U^n and let s_a be an assignment function such that $s_a(v_j) = a_j$, where a_j is the jth component of the tuple a. Then we shall say

$$\begin{array}{lll} \Im \models w[a] & \text{iff} & \Im \models w[s_a] \text{ and} \\ \Im \models t[a] & \text{iff} & \Im \models t[s_a]. \end{array}$$

As the interpretation intended for constant symbols and constant predicate symbols should be clear from the context, often we shall not mention it. In fact, if $\Im$ is an interpretation which assigns D_i to X_i , P_i to Y_i and the interpretation intended for constant symbols and constant predicate symbols is standard then we shall simply write $(D, P) \models w[s]$, where (D, P) denotes the tuple $((D_1, P_1), \ldots, (D_k, P_k))$, instead of $\Im \models w[s]$.

10. Predicator Specification

Now we are in a position to specify a predicator precisely.

Let $\mathcal{L}_i$, $(1 \leq i \leq k)$, be a predicate domain on a retract L_i of V^{n_i}, and let $\mathcal{M}$ be a predicate domain on a retract M of V^n. Let $T \in V^{n_1} \times \cdots \times V^{n_k} \to V^n$ be a domain transformation such that if $C_i \in L_i$, $(1 \leq i \leq k)$, then $T(C_1, \ldots, C_k) \in M$. Let $\Phi = \Phi(X, Y)$ be the set of wffs, where (X, Y) stands for the tuple $((X_1, Y_1), \ldots, (X_k, Y_k))$ and $(X_i, Y_i) \in \mathcal{L}_i$. Let $w \in \Phi$ be a wff whose free variables over the universal domain U are contained in the list $v_1, \ldots, v_n$. Then a predicator (T, w) is a map from $\mathcal{L}_1 \times \cdots \times \mathcal{L}_k$ to $\mathcal{M}$ defined as follows: for all $(C_i, Q_i) \in \mathcal{L}_i$, letting (C, Q) denote the tuple $((C_1, Q_1), \ldots, (C_k, Q_k))$, and C denote the tuple $(C_1, \cdots, C_k)$,

$$(T, w)(C, Q) = (T(C), \{a \in D \mid (C, Q) \models w[a]\}).$$

Obviously for the predicator (T, w) to be well defined it must be the case that $(D, R) = (T, w)(C, Q) \in M$, for all $(C_i, Q_i) \in \mathcal{L}_i$, $(1 \leq i \leq k)$.

We shall present several simple examples of predicators below. These examples are derived from the relational functors of Reynolds. (See [Reynolds].)

10..1. $(\rightarrow, arrow)$ predicator

Let $\rightarrow \in V \times V \rightarrow V$ be the domain exponentiation function. One naturally associates with this function a predicator as follows. Let L, M, N be retracts of V such that, for all $C \in L$ and $D \in M$, $(C \rightarrow D) \in N$. Let us define predicate cpos:

$$\begin{aligned} \mathcal{L} &= (L, \{\perp\}), \\ \mathcal{M} &= (M, \{\perp\}), \\ \mathcal{N} &= (N, \{\perp\}). \end{aligned}$$

Let $arrow \in \Phi((C,P),(D,Q))$, where $(C,P) \in \mathcal{L}$ and $(D,Q) \in \mathcal{M}$, be a wff defined as follows:

$$arrow(x) = \text{“}\forall u \in P.\ x \cdot u \in Q\text{”},$$

where the application functon $\cdot : U \times U \rightarrow U$ is defined as:

$$x \cdot u = (j_{\rightarrow} x)u \text{ for all } x, u \in U.$$

Then $(\rightarrow, arrow)$ is a predicator from $\mathcal{L} \times \mathcal{M}$ to $\mathcal{N}$.

One can similarly construct predicators $(+, plus)$,$(\times, product)$ and $(\odot, lift)$ corresponding to the functions $+, \times \in V \times V \rightarrow V$ and $\odot \in V \rightarrow V$.

10..2. $(Cont, cont)$ predicator

For some fixed domain $O \in V$, let $Cont \in V \rightarrow V$ be a continuation function defined by: for all $C \in V$,

$$Cont(C) = (C \rightarrow O) \rightarrow O.$$

(This construction is used in giving a semantics to programming languages through continuations. See [Reynolds].) Let L and M be retracts on V such that whenever $C \in L$, $(C \rightarrow O) \rightarrow O \in M$. Let $\mathcal{L}$ and $\mathcal{M}$ be predicate cpos defined by:

$$\begin{aligned} \mathcal{L} &= (L, \{\perp\}), \text{ and} \\ \mathcal{M} &= (M, \{\perp\}). \end{aligned}$$

Let a wff $cont \in \Phi(C,P)$, where $(C,P) \in \mathcal{L}$, be defined as follows:

$$cont(x) = \text{“}\exists u \in P.\ x = i_{\rightarrow}(\lambda(v : C \rightarrow O).v \cdot u) \text{ in } ((C \rightarrow O) \rightarrow O)\text{”},$$

where $\lambda(v : C \rightarrow O).\ v \cdot u$ stands for $\lambda(v : U).\ ((C \rightarrow O)v) \cdot (u)$, and the application function $\cdot$ is as before. Then $(Cont, cont)$ is a predicator from $\mathcal{L}$ to $\mathcal{L}$.

11. Reduction Algorithm

Once we have constructed a predicator we ask if it is continuous. In this section we shall give a reduction algorithm which generates sufficient goals to guarantee this continuity.

The following theorem reduces the question of continuity to that of monotonicity.

Theorem 11.1: If a predicator (T, w) from $\mathcal{L}_1 \times \cdots \times \mathcal{L}_k$ to $\mathcal{M}$ is monotonic, it is continuous.

Proof: For simplicity, we shall consider the case of one argument only, the general case being similar. Hence, let (T, w) be a monotonic predicator from $\mathcal{L}$ to $\mathcal{M}$. Remember that by the definition of predicator T is a continuous domain transformation. Let $X = \{(C_i, P_i) \mid i \in I\}$ be a directed set in $\mathcal{L}$, where I is a directed index set. As (T, w) is monotonic the set $Y = \{(T, w)(C_i, P_i) \mid i \in I\}$ is directed in $\mathcal{M}$. We want to show that

$$(T, w)(lub(X)) = lub(Y).$$

Let $(C, P) = lub(X)$. Then $C = lub\{C_i \mid i \in I\}$. Hence $(T, w)(lub(X)) = (T(C), R)$, for some predicate R on $T(C)$. Similarly, as each $(T, w)(C_i, P_i) = (T(C_i), Q_i)$ for some predicate Q_i on $T(C_i)$, it follows that:

$$\begin{array}{rcll} lub(Y) & = & lub\{(T, w)(C_i, P_i) \mid i \in I\}, & \\ & = & lub\{(T(C_i), Q_i) \mid i \in I\}, & \\ & = & (lub\{T(C_i) \mid i \in I\}, Q) & \text{for some } Q \text{ on } lub\{T(C_i) \mid i \in I\}, \\ & = & (T(C), Q) & \text{by continuity of } T. \end{array}$$

As each $(C_i, P_i) \sqsubseteq lub(X)$ it is clear that

$$lub(Y) \sqsubseteq (T, w)(lub(X)),$$

which means

$$(T(C), Q) = lub(Y) \sqsubseteq (T, w)(lub(X)) = (T(C), R).$$

From this it follows that:

$$\begin{array}{rcll} R & = & T(C)(R) & \text{as } R \subseteq |T(C)|, \\ & = & Q & \text{as } (T(C), Q) \sqsubseteq (T(C), R). \end{array}$$

Hence,

$$lub(Y) = (T(C), Q) = (T(C), R) = (T, w)(lub(X)).$$

∎

Once the issue of continuity of predicators is reduced to that of monotonicity, it becomes desirable to see if there is a general way of proving monotonicity of predicators. Below we shall give a crucial algorithm which generates *sufficient* goals to guarantee monotonicity of a predicator. The algorithm is crucial because the goals generated by the algorithm can be proved within the LCF formalism, thereby making mechanization of the theory possible.

Let $\mathcal{L}_i$, $(1 \leq i \leq k)$, be a predicate domain on a retract L_i of V^{n_i} and let $\mathcal{M}$ be a predicate domain on a retract M of V^n. Let (T, w) be a predicator from $\mathcal{L}_1 \times \cdots \times \mathcal{L}_k$ to $\mathcal{M}$ specified as follows.

$$T \in V^{n_1} \times \cdots \times V^{n_k} \rightarrow V^n,$$
$$w \in \Phi(X, Y),$$

where (X, Y) stands for a tuple $((X_1, Y_1), \ldots, (X_k, Y_k))$ and each $(X_i, Y_i) \in \mathcal{L}_i$.

We want to prove that (T, w) is monotonic. Let $(C, P), (\bar{C}, \bar{P}) \in \mathcal{L}_1 \times \cdots \times \mathcal{L}_k$, where both (C, P) and $(\bar{C}, \bar{P})$ are k-tuples, be such that $(C, P) \sqsubseteq (\bar{C}, \bar{P})$. We want to show that

$$(T, w)(C, P) \sqsubseteq (T, w)(\bar{C}, \bar{P}),$$

i.e.,

$$(T(C), \{a \in T(C) \mid (C, P) \models w[a]\}) \sqsubseteq (T(\bar{C}), \{\bar{a} \in T(\bar{C}) \mid (\bar{C}, \bar{P}) \models w[\bar{a}]\}).$$

This amounts to showing three things:

1. $T(C) \sqsubseteq T(\bar{C})$.
2. For all $a \in T(C)$, $(C, P) \models w[a]$ implies $(\bar{C}, \bar{P}) \models w[a]$.
3. For all $\bar{a} \in T(\bar{C})$, $(\bar{C}, \bar{P}) \models w[\bar{a}]$ implies $(C, P) \models w[T(C)(\bar{a})]$.

The first condition trivially follows from monotonicity of T. To prove the second condition we generate a goal

$$\text{"}(C, P) \models w[a] \text{ implies } (\bar{C}, \bar{P}) \models w[a]\text{"}$$

and reduce it to simpler goals using the following algorithm Reduce1. To economize on notation in the following discussion, we shall assume that in the above goal a is implicitly universally quantified over $T(C)$. (Of course, it is possible to say '$\forall a \in T(C)$' explicitly in the goal.) To prove the third condition we generate a goal

$$\text{"}(\bar{C}, \bar{P}) \models w[\bar{a}] \text{ implies } (C, P) \models w[T(C)(\bar{a})]\text{"}$$

and reduce it to simpler goals using the following algorithm Reduce2. Again we shall assume that in the above goal $\bar{a}$ is implicitly universally quantified over $T(\bar{C})$.

Reduce1 and Reduce2 are mutually recursive.

Reduce1 reduces a goal of the form

$$(C, P) \models w[z] \text{ implies } (\bar{C}, \bar{P}) \models w[y], \text{ with } z \sqsubseteq y \text{ as induction hypothesis.}$$

Reduce2 reduces a goal of the form

$$(\bar{C}, \bar{P}) \models w[v] \text{ implies } (C, P) \models w[u], \text{ with } u \sqsubseteq v \text{ as induction hypothesis.}$$

One can easily see that initially the induction hypotheses of both Reduce1 and Reduce2 are satisfied. The induction hypothesis of Reduce2 is satisfied because $\bar{a} \in T(\bar{C})$ and $T(C) \sqsubseteq T(\bar{C})$ implies $T(C)\bar{a} \sqsubseteq T(\bar{C})\bar{a} = \bar{a}$. The check in the case of Reduce1 is trivial.

We shall describe Reduce1 and Reduce2 now. Let $\Im$ be the interpretation which assigns the domain vector C to X and the predicate vector P to Y. And similarly Let $\bar{\Im}$ be the interpretation which assigns the domain vector $\bar{C}$ to X and the predicate vector $\bar{P}$ to Y. Interpretation of constant symbols is supposed to be standard.

11.1. Reduce1

The input goal of Reduce1 is

$$(C, P) \models w[z] \text{ implies } (\bar{C}, \bar{P}) \models w[y],$$

i.e.,

$$\mathfrak{S} \models w[z] \text{ implies } \bar{\mathfrak{S}} \models w[y],$$

where $z \sqsubseteq y$. We shall achieve reduction through structural induction. We have to consider several cases.

1. w is "$t \in P$ in D". Now the goal can be written as:

$$D^{\mathfrak{S}}(t^{\mathfrak{S}}[z]) \in P^{\mathfrak{S}} \text{ implies } D^{\bar{\mathfrak{S}}}(t^{\bar{\mathfrak{S}}}[y]) \in P^{\bar{\mathfrak{S}}}.$$

Assume $D^{\mathfrak{S}}(t^{\mathfrak{S}}[z]) \in P^{\mathfrak{S}}$. Then, as $(D^{\mathfrak{S}}, P^{\mathfrak{S}}) \sqsubseteq (D^{\bar{\mathfrak{S}}}, P^{\bar{\mathfrak{S}}})$, we conclude that $D^{\mathfrak{S}}(t^{\mathfrak{S}}[z]) \in P^{\bar{\mathfrak{S}}}$. Now, as $C \sqsubseteq \bar{C}$ and $z \sqsubseteq y$, we know

$$D^{\mathfrak{S}}(t^{\mathfrak{S}}[z]) \sqsubseteq D^{\bar{\mathfrak{S}}}(t^{\bar{\mathfrak{S}}}[y]).$$

Suppose we know, in addition, that $P^{\bar{\mathfrak{S}}}$ is upward closed in the indices of some set G. Let

$$(r_1, \ldots, r_l) = D^{\mathfrak{S}}(t^{\mathfrak{S}}[z]), \text{ and}$$
$$(\bar{r}_1, \ldots, \bar{r}_l) = D^{\bar{\mathfrak{S}}}(t^{\bar{\mathfrak{S}}}[y]).$$

Then, as $(r_1, \ldots, r_l) \in P^{\bar{\mathfrak{S}}}$, it follows from upward closedness that to prove

$$(\bar{r}_1, \ldots, \bar{r}_l) = D^{\bar{\mathfrak{S}}}(t^{\bar{\mathfrak{S}}}[y]) \in P^{\bar{\mathfrak{S}}}$$

it suffices to show that

$$r_j = \bar{r}_j \text{ for all } j \notin G.$$

Hence, for each $j \notin G$ we generate a goal:

$$(D^{\mathfrak{S}}(t^{\mathfrak{S}}[z])_j) = (D^{\bar{\mathfrak{S}}}(t^{\bar{\mathfrak{S}}}[y])_j).$$

It is illuminating to consider the following special cases.

(a) The predicate symbol P is $\sqsubseteq$. In this case w can be written more simply as "$t_1 \sqsubseteq t_2$ in D". As $\sqsubseteq$ is upward closed in the second argument, by what we have seen above, it suffices to produce the following single goal.

$$D^{\mathfrak{S}} t_1^{\mathfrak{S}}[z] = D^{\bar{\mathfrak{S}}} t_1^{\bar{\mathfrak{S}}}[y].$$

(b) The predicate symbol P is $\sqsupseteq$. In this case w can be written more simply as "$t_1 \sqsupseteq t_2$ in D". As $\sqsupseteq$ is upward closed in the first argument, it suffices to produce the following single goal.

$$D^{\mathfrak{S}} t_2^{\mathfrak{S}}[z] = D^{\bar{\mathfrak{S}}} t_2^{\bar{\mathfrak{S}}}[y].$$

(c) The predicate symbol P is $=$. In this case w can be written more simply as "$t_1 = t_2$ in D". As $=$ is upward closed in neither of the arguments, we have to produce two subgoals in this case:

$$D^{\Im} t_1^{\Im}[z] = D^{\bar{\Im}} t_1^{\bar{\Im}}[y],$$
$$D^{\Im} t_2^{\Im}[z] = D^{\bar{\Im}} t_2^{\bar{\Im}}[y].$$

2. w is "$\forall u \in D.g$". The goal can now be written as:

$$(\forall a \in D^{\Im}.\Im \models g[a, z]) \text{ implies } (\forall \bar{a} \in D^{\bar{\Im}}.\bar{\Im} \models g[\bar{a}, y]).$$

But for all $\bar{a} \in D^{\bar{\Im}}$ we know $D^{\Im}(\bar{a}) \in D^{\Im}$. Hence it suffices to generate the goal:

$$\Im \models g[D^{\Im}(\bar{a}), z] \text{ implies } \bar{\Im} \models g[\bar{a}, y],$$

where $\bar{a}$ is assumed to be universally quantified over $D^{\bar{\Im}}$. (An alert reader will notice that a weakening has resulted during this reduction: although this goal is sufficient it is not necessary. This will be the case for the following cases too.) This goal can now be reduced further by recursively calling Reduce1. Note that $\bar{a} \in D^{\bar{\Im}}$ and $D^{\Im} \sqsubseteq D^{\bar{\Im}}$ implies $D^{\Im}(\bar{a}) \sqsubseteq \bar{a}$. Hence, as $z \sqsubseteq y$, we conclude $(D^{\Im}(\bar{a}), z) \sqsubseteq (\bar{a}, y)$, which means the induction hypothesis of Reduce1 is indeed satisfied.

3. w is "$\exists u \in D.g$". The goal can now be written as:

$$(\exists a \in D^{\Im}.\Im \models g[a, z]) \text{ implies } (\exists \bar{a} \in D^{\bar{\Im}}.\bar{\Im} \models g[\bar{a}, y]).$$

But, as $D^{\Im} \sqsubseteq D^{\bar{\Im}}$, we know that $a \in D^{\Im}$ implies $a \in D^{\bar{\Im}}$. Hence it suffices to generate the goal:

$$\Im \models g[a, z] \text{ implies } \bar{\Im} \models g[a, y],$$

where a is assumed to be quantified over $D^{\Im}$. This goal can now be reduced further by recursively calling Reduce1. Note that the induction hypothesis of Reduce1 is trivially satisfied.

4. w is "$h \Rightarrow g$". In this case the goal can be written as:

$$(\Im \models h[z] \text{ implies } \Im \models g[z]) \text{ implies}$$
$$(\bar{\Im} \models h[y] \text{ implies } \bar{\Im} \models g[y]).$$

But to prove this it obviously suffices to produce the following subgoals:

$$\bar{\Im} \models h[y] \text{ implies } \Im \models h[z], \text{ and}$$
$$\Im \models g[z] \text{ implies } \bar{\Im} \models g[y].$$

The first goal can be simplified further by Reduce2 and the second one by Reduce1. As $z \sqsubseteq y$, the induction hypothesis of both Reduce1 and Reduce2 are quite trivially satisfied.

5. w is "$h \wedge g$". In this case the goal can be written as:

$$(\Im \models h[z] \text{ and } \Im \models g[z]) \text{ implies}$$
$$(\bar{\Im} \models h[y] \text{ and } \bar{\Im} \models g[y]).$$

But to prove this it obviously suffices to produce the following subgoals:

$$\Im \models h[z] \text{ implies } \bar{\Im} \models h[y], \text{ and}$$
$$\Im \models g[z] \text{ implies } \bar{\Im} \models g[y].$$

Both of these goals can be simplified further by Reduce1.

6. w is "$h \vee g$". In this case the goal can be written as:

$$(\Im \models h[z] \text{ or } \Im \models g[z]) \text{ implies}$$
$$(\bar{\Im} \models h[y] \text{ or } \bar{\Im} \models g[y]).$$

But to prove this it obviously suffices to produce the following subgoals:

$$\Im \models h[z] \text{ implies } \bar{\Im} \models h[y], \text{ and}$$
$$\Im \models g[z] \text{ implies } \bar{\Im} \models g[y].$$

Both of these goals can be simplified further by Reduce1.

11.2. Reduce2

The input goal of Reduce1 is

$$(\bar{C}, \bar{P}) \models w[v] \text{ implies } (C, P) \models w[u],$$

i.e.,

$$\bar{\Im} \models w[v] \text{ implies } \Im \models w[u],$$

where $u \sqsubseteq v$. Again we shall achieve reduction through structural induction. We have to consider several cases.

1. w is "$t \in P$ in D". Now the goal can be written as:

$$D^{\bar{\Im}}(t^{\bar{\Im}}[v]) \in P^{\bar{\Im}} \text{ implies } D^{\Im}(t^{\Im}[u]) \in P^{\Im}.$$

Assume $D^{\bar{\Im}}(t^{\bar{\Im}}[v]) \in P^{\bar{\Im}}$. Then, as $(D^{\Im}, P^{\Im}) \sqsubseteq (D^{\bar{\Im}}, P^{\bar{\Im}})$, we conclude that $D^{\Im}(t^{\bar{\Im}}[v]) = D^{\Im} \circ D^{\bar{\Im}}(t^{\bar{\Im}}[v]) \in P^{\Im}$. Now, as $C \sqsubseteq \bar{C}$ and $u \sqsubseteq v$, we know

$$D^{\Im}(t^{\Im}[u]) \sqsubseteq D^{\Im}(t^{\bar{\Im}}[v]).$$

Suppose we know, in addition, that $P^{\Im}$ is downward closed in the indices of some set H. Let

$$(r_1, \ldots, r_l) = D^{\Im}(t^{\Im}[u]), \text{ and}$$
$$(\bar{r}_1, \ldots, \bar{r}_l) = D^{\Im}(t^{\bar{\Im}}[v]).$$

Then, as $(\bar{r}_1, \ldots, \bar{r}_l) \in P^{\bar{\mathfrak{D}}}$, it follows from downward closedness that to prove

$$(r_1, \ldots, r_l) = D^{\mathfrak{D}}(t^{\mathfrak{D}}[u]) \in P^{\mathfrak{D}}$$

it suffices to show that

$$r_j = \bar{r}_j \text{ for all } j \notin H.$$

Hence, for each $j \notin H$ we generate a goal:

$$(D^{\mathfrak{D}}(t^{\mathfrak{D}}[u])_j = (D^{\bar{\mathfrak{D}}}(t^{\bar{\mathfrak{D}}}[v])_j.$$

Once again it is illuminating to consider the following special cases.

(a) The predicate symbol P is $\sqsubseteq$. In this case w can be written as "$t_1 \sqsubseteq t_2$ in D". As $\sqsubseteq$ is downward closed in the first argument, it suffices to produce the following single goal.

$$D^{\mathfrak{D}} t_2^{\mathfrak{D}}[u] = D^{\bar{\mathfrak{D}}} t_2^{\bar{\mathfrak{D}}}[v].$$

(b) The predicate symbol P is $\sqsupseteq$. In this case w can be written as "$t_1 \sqsupseteq t_2$ in D". As $\sqsupseteq$ is downward closed in the second argument, it suffices to produce the following single goal.

$$D^{\mathfrak{D}} t_1^{\mathfrak{D}}[u] = D^{\bar{\mathfrak{D}}} t_1^{\bar{\mathfrak{D}}}[v].$$

(c) The predicate symbol P is $=$. In this case w can be written more simply as "$t_1 = t_2$ in D". As $=$ is downward closed in neither of the arguments, we have to produce two subgoals in this case:

$$D^{\mathfrak{D}} t_1^{\mathfrak{D}}[u] = D^{\bar{\mathfrak{D}}} t_1^{\bar{\mathfrak{D}}}[v],$$
$$D^{\mathfrak{D}} t_2^{\mathfrak{D}}[u] = D^{\bar{\mathfrak{D}}} t_2^{\bar{\mathfrak{D}}}[v].$$

2. w is "$\forall z \in D.g$". The goal can now be written as:

$$(\forall \bar{a} \in D^{\bar{\mathfrak{D}}}.\bar{\mathfrak{D}} \models g[\bar{a}, v]) \text{ implies } (\forall a \in D^{\mathfrak{D}}.\mathfrak{D} \models g[a, u]).$$

But $a \in D^{\mathfrak{D}}$ implies $a \in D^{\bar{\mathfrak{D}}}$. Hence, it suffices to generate the goal:

$$\bar{\mathfrak{D}} \models g[a, v] \text{ implies } \mathfrak{D} \models g[a, u],$$

where a is assumed to be quantified over $D^{\mathfrak{D}}$. This goal can now be reduced further by recursively calling Reduce2. Note that the induction hypothesis is trivially satisfied.

3. w is "$\exists z \in D.g$". The goal can now be written as:

$$(\exists \bar{a} \in D^{\bar{\mathfrak{D}}}.\bar{\mathfrak{D}} \models g[\bar{a}, v]) \text{ implies } (\exists a \in D^{\mathfrak{D}}.\mathfrak{D} \models g[a, u]).$$

But, as $\bar{a} \in D^{\bar{\mathfrak{D}}}$ implies $D^{\mathfrak{D}}(\bar{a}) \in D^{\mathfrak{D}}$, it suffices to generate the goal:

$$\bar{\mathfrak{D}} \models g[\bar{a}, v] \text{ implies } \mathfrak{D} \models g[D^{\mathfrak{D}}(\bar{a}), u],$$

where $\bar{a}$ is assumed to be quantified over $D^{\bar{\mathfrak{I}}}$. This goal can now be reduced further by recursively calling Reduce2. As $\bar{D}^{\bar{\mathfrak{I}}}$ implies $D^{\mathfrak{I}}(\bar{a}) \sqsubseteq \bar{a}$, one sees that the induction hypothesis of Reduce2 is indeed satisfied.

4. w is "$h \Rightarrow g$". In this case the goal can be written as:

$$(\bar{\mathfrak{I}} \models h[v] \text{ implies } \bar{\mathfrak{I}} \models g[v]) \text{ implies}$$
$$(\mathfrak{I} \models h[u] \text{ implies } \mathfrak{I} \models g[u]).$$

But to prove this it obviously suffices to produce the following subgoals:

$$\mathfrak{I} \models h[u] \text{ implies } \bar{\mathfrak{I}} \models h[v], \text{ and}$$
$$\bar{\mathfrak{I}} \models g[v] \text{ implies } \mathfrak{I} \models g[u].$$

The first goal can be simplified further by Reduce1 and the second one by Reduce2. As $u \sqsubseteq v$, the induction hypotheses of both Reduce1 and Reduce2 are quite trivially satisfied.

5. w is "$h \wedge g$". In this case the goal can be written as:

$$(\bar{\mathfrak{I}} \models h[v] \text{ and } \bar{\mathfrak{I}} \models g[v]) \text{ implies}$$
$$(\mathfrak{I} \models h[u] \text{ and } \mathfrak{I} \models g[u]).$$

But to prove this it obviously suffices to produce the following subgoals:

$$\bar{\mathfrak{I}} \models h[v] \text{ implies } \mathfrak{I} \models h[u], \text{ and}$$
$$\bar{\mathfrak{I}} \models g[v] \text{ implies } \mathfrak{I} \models g[u].$$

Both of these goals can be simplified further by Reduce2.

6. w is "$h \vee g$". In this case the goal can be written as:

$$(\bar{\mathfrak{I}} \models h[v] \text{ or } \bar{\mathfrak{I}} \models g[v]) \text{ implies}$$
$$(\mathfrak{I} \models h[u] \text{ or } \mathfrak{I} \models g[u]).$$

But to prove this it obviously suffices to produce the following subgoals:

$$\bar{\mathfrak{I}} \models h[v] \text{ implies } \mathfrak{I} \models h[u], \text{ and}$$
$$\bar{\mathfrak{I}} \models g[v] \text{ implies } \mathfrak{I} \models g[u].$$

Both of these goals can be simplified further by Reduce2.

12. Examples

Let us prove continuity of some predicators using the reduction algorithm of the last section.

12.1. $(\rightarrow, arrow)$*predicator*

Let $(\rightarrow, arrow)$ be the predicator from $\mathcal{L} \times \mathcal{M}$ to $\mathcal{N}$ of 10..1, where $\mathcal{L}$, $\mathcal{M}$, $\mathcal{N}$ are predicate cpos on retracts L, M, and N of V respectively. For this predicator, the reduction algorithm generates the following goals. Here, unlike in the discussion of the reduction algorithm where we did not carry with a goal its assumption list, we shall explicitly mention within a goal all the assumptions. The goals are:

$\forall C, C' \in L$ and $D, D' \in M$,
$C \sqsubseteq C'$ and $D \sqsubseteq D'$ implies

1. $\forall x \in (C \rightarrow D)$ and $u' \in C'$, $x \cdot u' = x \cdot (C(u'))$,
2. $\forall x' \in (C' \rightarrow D')$ and $u \in C$. $C(x' \cdot u) = ((C \rightarrow D)x') \cdot u$,

where the application function $\cdot : U \times U \rightarrow U$ as usual stands for $\lambda y, z.(j_{\rightarrow} y)z$. It is possible to prove validity of both of these goals from the definition of $\rightarrow: V \times V \rightarrow V$ and the properties of the relevant functions on the universal domain.

This proves continuity of $(\rightarrow, arrow)$. Now we can rightfully say

$$(\rightarrow, arrow) \in (\mathcal{L} \times \mathcal{M}) \rightarrow \mathcal{N}.$$

12.2. $(Cont, cont)$ predicator

Let $(Cont, cont)$ be the predicator from $\mathcal{L}$ to $\mathcal{L}$ of 10..2, where $\mathcal{L}$ is predicate cpo on a retract L of V. For this predicator, the reduction algorithm generates the following goals.
$\forall C, C' \in L$.
$C \sqsubseteq C'$ implies

1. $\forall x \in (C \rightarrow O) \rightarrow O$ and $u \in C$.
   (a) $i_{\rightarrow}(\lambda v : (C \rightarrow O).v \cdot u) = i_{\rightarrow}(\lambda v : (C' \rightarrow O).v \cdot u)$,
   (b) $x = x$,
2. $\forall x' \in (C' \rightarrow O) \rightarrow O$ and $u' \in C'$.
   (a) $i_{\rightarrow}(\lambda v : (C \rightarrow O).v \cdot (Cu')) = ((C \rightarrow O) \rightarrow O)(i_{\rightarrow}(\lambda v : (C' \rightarrow O).v \cdot u'))$,
   (b) $((C \rightarrow O) \rightarrow O)(x') = ((C \rightarrow O) \rightarrow O)(x')$,

where the application function $\cdot : U \times U \rightarrow U$ as usual stands for $\lambda y, z.(j_{\rightarrow} y)z$. Again it is possible to prove validity of these goals using the definition of $\rightarrow: V \times V \rightarrow V$ and the properties of relevant functions on the universal domain.

This proves continuity of $(Cont, cont)$. Hence, $(Cont, cont) \in \mathcal{L} \rightarrow \mathcal{L}$.

12.3. Continuation Of An Earlier Example

The predicators considered so far do not illustrate the use of upward closure and downward closure properties. This we shall remedy in the next example.

Let us complete the semantic equivalence proof of Section 4 by showing that (1) has a solution. For later convenience we shall change the notation a bit. We shall now denote

the domain of values by $\bar{D}$ and the domain of environments by $\bar{C}$. Then we have the domain equations:

$$\bar{D} = \bar{C} \to B,$$
$$\bar{C} = Ide \to \bar{D}.$$

The domain of syntactic expessions will now be denoted by $\bar{E}$, and the domain of syntactic environments by $\bar{F}$; $\bar{F} = Ide \to \bar{E}$.

We want to show that there exist $\bar{\Theta} \subseteq \bar{D} \times \bar{E}$ and $\bar{\Pi} \subseteq \bar{C} \times \bar{F}$ such that

$$\begin{array}{rcl} \bar{\Theta} & = & \{(d,e) \mid (c,f) \in \bar{\Pi}.\ dc \sqsubseteq \mathcal{O}(e)f\}, \\ \bar{\Pi} & = & \{(c,f) \mid \forall I \in Ide.\ (cI, fI) \in \bar{\Theta}\}. \end{array} \tag{3}$$

We shall proceed to prove this existence.

As the fixpoint set of the identity retract of V^2 is V^2 again, we shall denote, with some abuse of notation, the identity retract by V^2 again. Let $\mathcal{L}$ be the following predicate domain on V^2:

$$\mathcal{L} = (V^2, \{2\}, \{1\}, \{\bot\}).$$

Note that now we are only considering the predicates which are upward closed in the second argument and downward closed in the first. The significance of this should be clear soon.

Now from equation 3 we construct, by an almost verbatim translation, the predicators (T,w) and (S,g) from $\mathcal{L}$ to $\mathcal{L}$ as follows. The domain transformations $T, S \in V^2 \to V^2$ are given by:

$$T(C,F) = (C \to B, \bar{E}),$$
$$S(D,E) = (Ide \to D, Ide \to E).$$

The wff $w \in \Phi(X,\Pi)$, where $(X,\Pi) \in \mathcal{L}$ (it might help to read (C,F) in place of X), is a predicate transformation corresponding to the first equation of 3, and is defined as:

$$w(d,e) = \forall (c,f) \in \Pi.\ d \cdot c \sqsubseteq \mathcal{O}(e)f \text{ in } B,$$

where $\cdot$ is the usual application function and $\mathcal{O} : U \to U \to U$ is strictly speaking the unique extension of the operational semantics given in Section 3 to the universal domain.

The wff $g \in \Phi(Z,\Theta)$, where $(Z,\Theta) \in \mathcal{L}$ (again it might help to read (D,E) in place of Z), is a predicate transformation corresponding to the second equation of 3, and is given by:

$$g(c,f) = \forall I \in Ide.\ (c \cdot I, f \cdot I) \in \Theta.$$

Of course, we have to show first that (T,w) and (S,g) are well defined. To show that (T,w) is well defined we have to show that for all $(X,\Pi) \in \mathcal{L}$, $(T,w)(X,\Pi) \in \mathcal{L}$. This amounts to showing that the predicate transformation w preserves directed completeness, upward closedness in the second argument and downward closedness in the first. However, all of these checks are trivial.

Now, assume for the moment that (T,w) and (S,g) are continuous, i.e., assume

$$(T,w), (S,g) \in \mathcal{L} \to \mathcal{L}.$$

Then there exist the least $((D^*, E^*), \Theta^*)$ and $((C^*, F^*), \Pi^*) \in \mathcal{L}$ such that

$$((D^*, E^*), \Theta^*) = (T, w)((C^*, F^*), \Pi^*),$$
$$((C^*, F^*), \Pi^*) = (S, g)((D^*, E^*), \Theta^*).$$

It is clear that (D^*, E^*) and (C^*, F^*) are the least elements in V^2 such that

$$(D^*, E^*) = T(C^*, F^*) = (C^* \to B, \bar{E}), \text{ and}$$
$$(C^*, F^*) = S(D^*, E^*) = (Ide \to D^*, Ide \to E^*).$$

But we know that $(\bar{D}, \bar{E})$ and $(\bar{C}, \bar{F})$ also constitute the least solution of the above equations, hence

$$\bar{D} = D^*,\ \bar{C} = C^*,\ \bar{E} = E^*,\ \bar{F} = F^*.$$

Thus $\Theta^* \subseteq \bar{D} \times \bar{E}$ and $\Pi^* \subseteq \bar{C} \times \bar{F}$. Now from the definitions of (T, w) and (S, g) it follows that Θ^* and Π^* indeed satisfy 3.

It remains to prove our major assumption, namely that (T, w) and (S, g) are continuous. The reduction algorithm produces the following goals to guarantee continuity of (T, w). (We have transformed the goals into equivalent but more readable ones, carried out minor simplifications and have omitted trivial tautologies.)

$\forall (C, F)$ and $(C', F') \in V^2$.
$(C, F) \sqsubseteq (C', F')$ implies

1. $\forall (d, e) \in (C \to B) \times \bar{E}$ and $(c', f') \in C' \times F'$.
   $B(d \cdot c') = B(d \cdot (Cc'))$,
2. $\forall (d', e') \in (C' \to B) \times \bar{E}$ and $(c, f) \in C \times F$.
   $B(\mathcal{O}(e')f) = B(\mathcal{O}(\bar{E}e')f)$.

$\forall (D, E)$ and $(D', E') \in V^2$.
$(D, E) \sqsubseteq (D', E')$ implies

3. $\forall (c, f) \in (Ide \to D) \times (Ide \to E)$ and $I' \in Ide$.
   $D'(c \cdot I') = D(c \cdot (Ide\, I'))$,
4. $\forall (c', f') \in (Ide \to D') \times (Ide \to E')$ and $I \in Ide$.
   $E(f' \cdot I) = E((Ide \to E)f' \cdot I)$.

All of these goals can be shown to be valid for *any* $\mathcal{O}$. This proves continuity of (T, w) and (S, g), thus finishing the existence proof.

It is tempting to see what happens if we replace $\sqsubseteq$ in 3 by $=$. Then, as $=$ is neither upward closed nor downward closed in any of the arguments, we have to let

$$\mathcal{L} = (V^2, \{\perp\}) = (V^2, \{\ \}, \{\ \}, \{\perp\}).$$

(T, w) and (S, g) are exactly as before except that in the definition of w one has to replace $\sqsubseteq$ by $=$. Now to guarantee continuity of (T, w) and (S, g), the reduction algorithm produces the following extra goals in addition to the four goals we saw above.

$\forall (C, F)$ and $(C', F') \in V^2$.
$(C, F) \sqsubseteq (C', F')$ implies

5. $\forall(d,e) \in (C \to B) \times \bar{E}$ and $(c',f') \in (C',F')$.
   $B(\mathcal{O}(e)f') = B(\mathcal{O}(e)(Ff'))$,
6. $\forall(d',e') \in (C' \to B) \times \bar{E}$ and $(c,f) \in (C,F)$.
   $B(d' \cdot c) = B((C \to B)d' \cdot c)$.

$\forall(D,E)$ and $(D',E') \in V^2$.
$(D,E) \sqsubseteq (D',E')$ implies

7. $\forall(c,f) \in (Ide \to D) \times (Ide \to E)$ and $I' \in Ide$.
   $E'(f \cdot I') = E(f \cdot (Ide(I')))$,
8. $\forall(c',f') \in (Ide \to D') \times (Ide \to E')$ and $I \in Ide$.
   $D(c' \cdot I) = D(((Ide \to D)c') \cdot I)$.

The goals 6,7,8 can again be proved to be valid. However, the goal 5 is no longer valid for all $\mathcal{O}$. Has our reduction algorithm produced an unnecessary goal? No! We have already shown in 5, through diagonalization, that 3 need not have solution for all $\mathcal{O}$s. Hence, though the goals produced by our reduction algorithm are sufficient but by no means necessary, they seem to be necessary in some weak sense.

If we replace $\sqsubseteq$ in 3 by $\sqsupseteq$ the reduction algorithm generates the goals 5. to 8. and hence the same reasoning goes through and we conclude that, in this case too, there is no solution in general.

13. Projectable Retracts And Constructible Functions

The predicators constructed so far do not exhaust the whole possible spectrum still, because validity of the goals generated for all these predicators could be proved without using properties of the underlying retracts of the predicate cpos. It is too optimistic to hope that this will always be the case. In an example treated Section 15, we shall see that validity of the goals generated by the reduction algorithm depends very critically on how the underlying retract is chosen. In this section we shall prepare ourselves for that eventuality.

We shall provide in this section *continuous* constructs which can be used in building underlying retracts of predicate cpos. Continuity implies that we shall be able to build these retracts as the least fixed points, and – what is more important – we shall be able to use fixpoint induction to prove the goals generated by the reduction algorithm, if the need arises. This advantage can not be underestimated because this is what makes proving in LCF all the goals generated by the reduction algorithm feasible, as LCF can reason only about cpos and continuous functions. With this in mind, we now proceed with the theory of these constructors.

Let L and M be retracts of V. Let $\to(|L|,|M|)$ be the image of $|L|$ and $|M|$ under the function $\to: V \times V \to V$, i.e.

$$\to(|L|,|M|) = \{C \to D \mid C \in |L| \text{ and } D \in |L|\}.$$

(Note that this set is *different* from $|L| \to |M|$, which is the set of continuous functions from $|L|$ to $|M|$.) We ask if there exists a retract $N_\to$ of V such that $|N_\to| = \to(|L|,|M|)$.

Of course, the same question could be asked as regards to $+(|L|,|M|)$, and $\times(|L|,|M|)$. The following theorem asserts this in positive.

Theorem 13.1: There exist

$$\rightarrow^{-1}, +^{-1}, \times^{-1} \in V \rightarrow V \times V$$

such that if

$$\begin{aligned} N_{\rightarrow} &= \rightarrow \circ < L, M > \circ \rightarrow^{-1}, \\ N_{+} &= + \circ < L, M > \circ +^{-1}, \\ N_{\times} &= \times \circ < L, M > \circ \times^{-1}, \end{aligned}$$

then

$$\begin{aligned} |N_{\rightarrow}| &= \rightarrow (|L|,|M|), \\ |N_{+}| &= +(|L|,|M|), \\ |N_{\times}| &= \times(|L|,|M|). \end{aligned}$$

Proof: Omitted. (see [Mulmuley3]) ∎

We shall call $\rightarrow^{-1}, +^{-1}$, and $\times^{-1}$ right inverses of $\rightarrow, +$, and $\times$ respectively.

More generally, given $T \in V^{n_1} \times \cdots \times V^{n_k} \rightarrow V^n$, we say $T^{-1} \in V^n \rightarrow V^{n_1} \times \cdots \times V^{n_k}$ is a right inverse of T if for any retracts L_i, $(1 \leq i \leq k)$, of V_{n_i},

1. $N_T = T \circ < L_1, \ldots, L_k > \circ T^{-1}$ is a retract of $V^{n_1} \times \cdots \times V^{n_k}$ and
2. $|N_T| = T(|L_1|, \ldots, |L_k|)$.

In this case we can define

$$\hat{T} = \lambda(L_1, \ldots, L_k).\ T \circ < L_1, \ldots, L_k > \circ T^{-1},$$

and then $\hat{T}$ can be thought of as a continuous constructor on underlying retracts of predicate cpos.

It is obvious that *Identity* and constant functions in $V \rightarrow V$ have right inverses too:

$$\begin{aligned} &Identity^{-1} = Identity, \\ &(\lambda x.b)^{-1} = (\lambda x.\bot). \end{aligned}$$

What is missing so far in the discussion is the projection function $proj_i^n \in V^n \rightarrow V$, $(1 \leq i \leq n)$:

$$proj_i^n = \lambda(x_1, \ldots, x_n).x_i).$$

The first guess for its right inverse would be:

$$(proj_i^n)^{-1} = \lambda x_i.(\bot, \ldots, x_i, \ldots, \bot).$$

Unfortunately, as it stands, $proj_i^n$ need not have right inverse. In fact, it can be shown that there exists a retract L of V^2 such that $proj_1^2(|L|)$ is not even a cpo, and hence it can not be a fixpoint set of any retract. This means we need to carry the construction of the underlying retracts of predicate cpos in a restricted subdomain if we want to avail of the constructor corresponding to the projection operation.

A retract L of V^n is called projectable if for every $1 \leq i \leq n$:

1. $N_i = proj_i^n \circ L \circ (proj_i^n)^{-1}$ is a retract of V, and
2. $|N| = proj_i^n(|L|)$.

The set of projectable retracts of V^n is a cpo, which we shall call $Pr(V^n)$. Trivially $Pr(V)$ is a cpo of all retracts of V. It is obvious that $(proj_i^n)^{-1}$ is a right inverse of $proj_i^n$ in the space of projectable retracts $Pr(V^n)$, because that is how we defined the notion of projectable retracts. Hence, we shall carry out the construction of underlying retracts of predicate cpos in the space of projectable retracts.

To summarize: now we have basic constructors $\hat{\rightarrow}, \hat{\times}, \hat{+}, \hat{proj}_i^n, \hat{I}, \hat{C}$ on projectable retracts, where I is the identity function on V and C is a constant function on V. What we need now is a way of constructing complex constructors.

Call $T \in V^{n-1} \times \cdots \times V^{n_k} \rightarrow V^n$ constructible if it can be constructed from $\rightarrow, \times, +, \odot, I, C, \circ, <\ ,\ >$, where I is the identity function on V, C is a constant function on V, $\circ$ is function composition and $\langle\ ,\ \rangle$ is the function pairing operation: $\langle f, g\rangle x = \langle f(x), g(x)\rangle$.

First note that if T is constructible and retracts L_i of V^{n_i} are projectable then so is $T(L_1, \ldots, L_k)$.

Theorem 13.2: In the space of projectable retracts each constructible function T has a right inverse T^{-1}.

Proof: Omitted. (see [Mulmuley3].) ∎

From the above theorem it follows that corresponding to each constructible function T we have a continuous constructor $\hat{T}$ on projectable retracts.

Let $(\ )_\perp$ be a combinator which makes a function strict, i.e.,

$$\begin{aligned}(f)_\perp(x) &= \perp \quad \text{if } x = \perp,\\ &= f(x) \quad \text{otherwise.}\end{aligned}$$

Let $[\times]$ be the usual function product combinator. Then both $(\ )_\perp$ and $[\times]$ are combinators on projectable retracts as well:

$$\begin{aligned}&(\ )_\perp \in Pr(V^n) \rightarrow Pr(V^n),\\ &[\times] \in Pr(V^m) \times Pr(V^n) \rightarrow Pr(V^{m+n}).\end{aligned}$$

Now we have a rich theory of projectable retracts and constructible functions.

13.1. Example

Let $B \in V$ be some constant domain. Let $B' = \lambda x.B$ be a retract of V. Obviously B' has a single fixpoint, namely, B. Let a retract L^* on V be defined as:

$$L^* = fix(\ \lambda(L : Pr(V)).\ ((L \hat{\rightarrow} B') \hat{\rightarrow} L \hat{+} B')_\perp).$$

Then if D^* is the least solution of the domain equation:

$$D = (D \rightarrow B) \rightarrow D + B,$$

the fixpoint set of L^* will look as shown in Figure 13.1 In particular, note that the inverse limit construction of D^* can be carried out entirely within $|L^*|$.

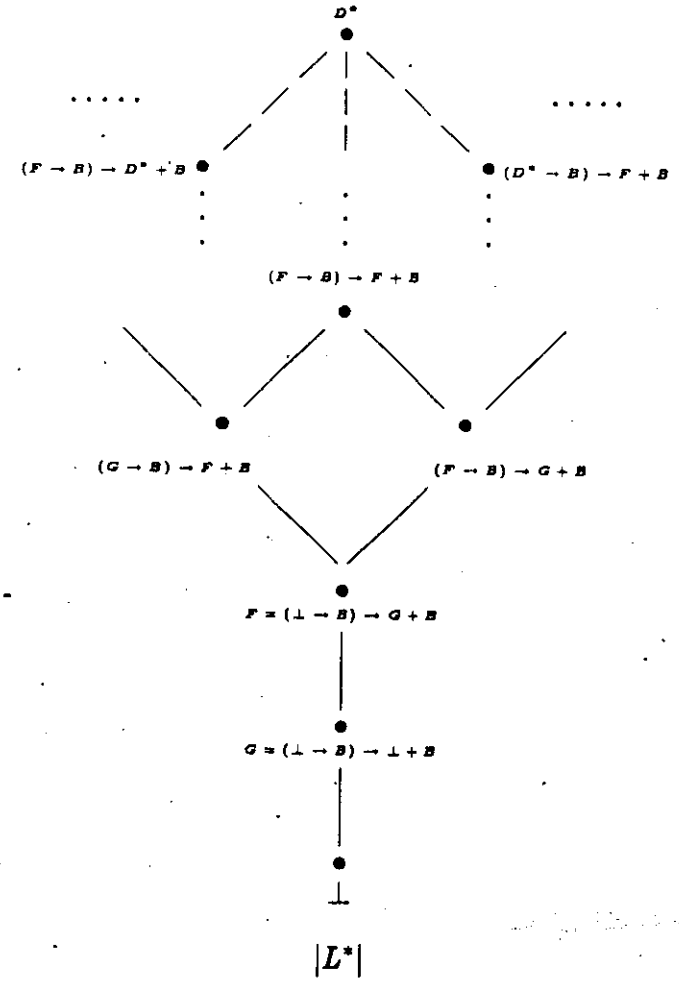


Figure 1: The fixpoint set of L^*

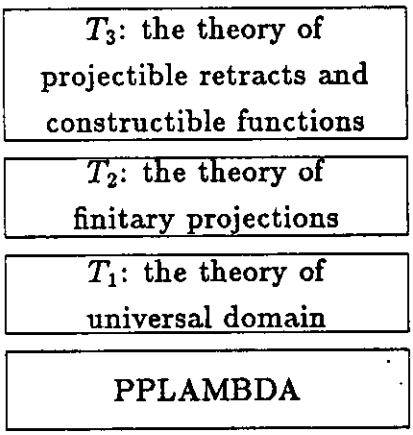


Figure 2: The Hierarchy Within IPLAMBDA

14. IPL Design

In the preceding sections we gave a mechanizable theory to prove inclusive predicate existence. Now we shall briefly describe how this theory was actually implemented on computer. For details see [Mulmuley3]. This system IPL (stands for Inclusive Predicate Logic) was implemented on top of LCF. It not only mechanizes the theory but it automates a large chunk too. For example, continuity of all the predicators considered so far can be proved by this system *automatically*. In particular, the existence of a solution to (1) is proved automatically. To understand the significance of this one only needs to look at the existence proof in [Stoy1], where the existence is proved using Milne's technique. It will illustrate the power of our system. IPL consists of three parts:

1. Goal Generator,
2. IPLAMBDA,
3. Automatic Theorem Provers,

To prove the existence of an inclusive predicate, one proceeds as follows. First one constructs predicators, by almost a verbatim translation, from the given equations over inclusive predicates. Next the specification of these predicators is fed into Goal Generator, which is an ML implementation of the reduction algorithm given in Section 11. The goals generated by Goal Generator are to be proved in IPLAMBDA, which is an ~~L~~CF theory. However, to ease this task of proving goals, Automatic Theorem Provers are provided which prove a large chunk of generated goals.

The theory IPLAMBDA is organized in a hierarchical fashion as shown in Figure 2. Theories $T1$ and $T2$ basically constitute the axiomatization of domain theory, i.e., the theory of the universal domain U and the domain of finitary of projections of the universal domain V. T_3 is an axiomatization of the theory of constructible functions and projectable retracts discussed in Section 13.

The set of automatic theorem provers provided by the IPL system is quite powerful. These theorem provers have a knowledge of domain theory and the theory of projectable

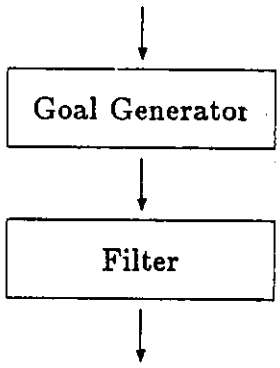


Figure 3: The Cascade

retracts and constructible functions. In particular, there is a theorem prover called Filter which has an extensive knowledge of the theory of the universal domain U (theory T_1) and the theory of the domain of finitary projections V (theory T_2). Because Filter is powerful, it is a good idea to always pipe the output of the Goal Generator into Filter. Thus, instead of Goal Generator, what one actually uses in practice is the cascade of Goal generator and Filter as shown in Figure 3: What comes out of this cascade is a "filtered" goal list. The amount of filtering which goes on can be judged from the fact that, *in case of all the predicators considerd so far, what comes out of this cascade is an empty list.* (what could be more pleasing?)

The overall design of the IPL system is summarized in Figure 4.

15. Example

This example is taken from [Gordon1]. The predicates, whose existence will be shown here, were used by Gordon to show correctness of a small Lisp implementation.

As Lisp has dynamic scope rule, the domains used for denotational semantics are the same as the ones in Section 3. So let $\bar{D}$ be the domain values, and let $\bar{C}$ be the domain of environments. $\bar{D}$ and $\bar{C}$ are the least domains such that:

$$\begin{aligned} \bar{D} &= \bar{C} \to B, \\ \bar{C} &= Ide \to \bar{D}, \end{aligned}$$

where Ide is the domain of identifiers and B is the domain of basic values.

We want to show the existence of the recursively defined predicates,

$$\begin{aligned} \bar{\Theta} &\subseteq \bar{D} \times \{Z \mid Z \subseteq Ide\}, \quad \text{and} \\ =^Z &\subseteq \bar{C} \times \bar{C} \qquad\qquad\quad \text{for each subset } Z \text{ of } Ide, \end{aligned}$$

which we shall describe soon. Note that $=^Z$ is actually a family of predicates, one predicate for every subset Z of Ide.

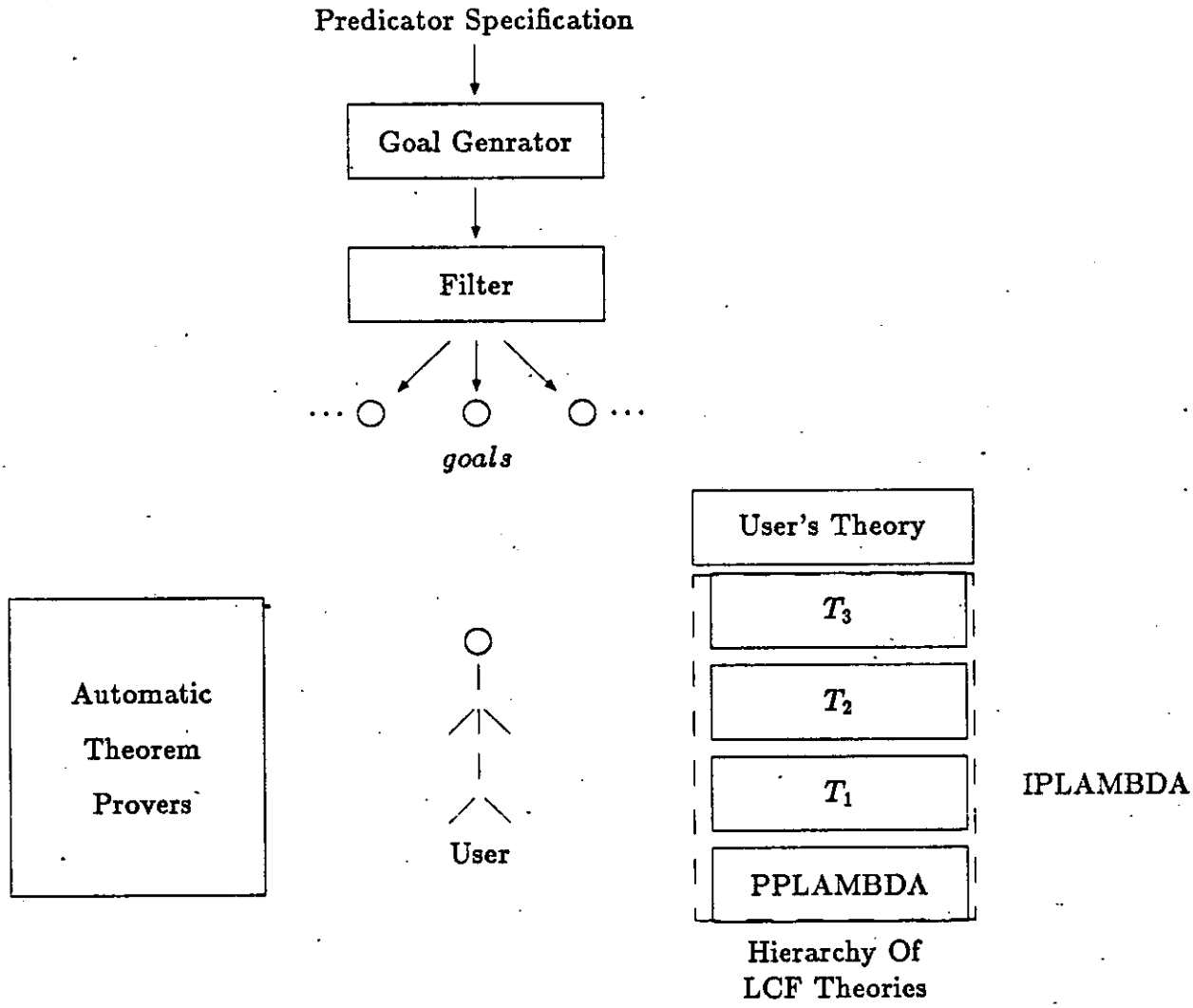


Figure 4: The Design Of IPL

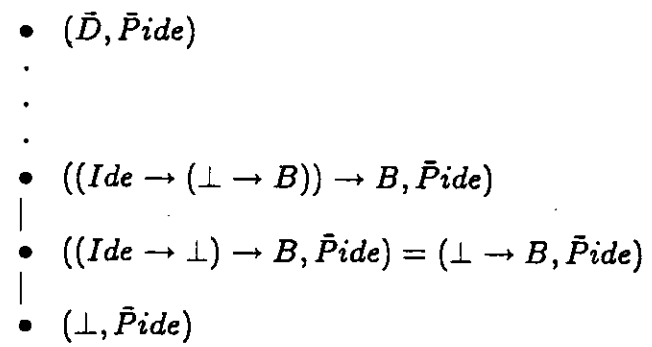


Figure 5: The Fixpoint Set Of L

Informally $(v, Z) \in \bar{\Theta}$, where $v \in \bar{D}$ and Z is a subset of Ide, will mean that "free variables" of v are contained in the set Z. And $(\rho, \rho') \in =^Z$, where ρ and $\rho' \in \bar{C}$, will mean that ρ and ρ' strongly agree on all the identifiers $z \in Z$. Formally,

$$\begin{aligned} \bar{\Theta} &= \{(v,z) \mid \forall Y \subseteq Ide,\ \rho \text{ and } \rho' \in \bar{C}.\ Z \subseteq Y \Rightarrow \\ &\qquad \rho =^Y \rho' \Rightarrow v(\rho) = v(\rho')\}, \text{ and} \\ =^Z &= \{(\rho,\rho') \mid \forall z \in Z.\ \rho(z) = \rho'(z) \text{ and } (\rho(z), Z) \in \bar{\Theta}\}. \end{aligned}$$

We shall show the existence of $\bar{\Theta}$, then that of $=^Z$ follows. By rearrangement we get:

$$\begin{aligned} \bar{\Theta} = \{(v,Z) \mid\ & \forall Y \subseteq Ide, \rho \text{ and } \rho' \in \bar{C}.\ Z \subseteq Y \Rightarrow \\ & (\forall y \in Y.\ \rho(y) = \rho'(y) \text{ and } (\rho(y), Y) \in \bar{\Theta}) \Rightarrow \\ & v(\rho) = v(\rho')\}. \end{aligned} \tag{4}$$

First we shall construct an underlying retract on which a predicate cpo will be constructed.

Let $\bar{P}ide$ be the flat domain of subsets of Ide, i.e., we just adjoin $\bot$ to the powerset of Ide. We shall ambiguously let Ide denote the retract $(\lambda(D : V).Ide)$ of V whose only fixpoint is $Ide \in V$. Similarly $\bar{P}ide$ and B will ambiguously denote the retracts of V whose only fixpoints are $\bar{P}ide$ and B respectively. Note that $\bar{D}$, the domain of values, is the least domain such that

$$\bar{D} = (Ide \to \bar{D}) \to B.$$

With this in mind, using the theory of projectable retracts and constructible functions given in Section 13, we define $L \in Pr(V^2)$, a projectable retract of V^2, as follows:

$$\begin{aligned} L = fix(\lambda L' \ &\in Pr(V^2). \\ &(\langle (Ide \hat{\to} (\hat{proj}_1^2 L')) \hat{\to} B, \bar{P}ide \rangle)_\bot). \end{aligned}$$

The fixpoint set of L, $|L| \subseteq V^2$, looks simply as shown in Figure 5. One sees that $|L|$ simply encapsulates the inverse limit construction of $\bar{D}$. Now let

$$\mathcal{L} = (L, Q),$$

where Q is a predicate on the least element of $|L|$, $\perp_L = (\perp, \bar{P}ide) \in V^2$, and is defined as follows:

$$Q = \{(\perp, Y) \mid Y \in \bar{P}ide\}.$$

Next we construct a predicator (T, w) from $\mathcal{L}$ to $\mathcal{L}$. Let $T \in V^2 \to V^2$ be defined as:

$$T(D, Pide) = ((Ide \to D) \to B, \bar{P}ide).$$

Let the wff $w \in ((D, Pide), \Theta)$, where $((D, Pide), \Theta) \in \mathcal{L}$, be defined as follows.

$$\begin{aligned} w(v, Z) = \ & \text{“}\forall Y \in \bar{P}ide. \forall \rho, \rho' \in (Ide \to D).\ Z \subseteq Y \Rightarrow \\ & \forall y \in Ide. (y \bar{\in} Y) \Rightarrow \\ & (\rho \cdot y = \rho' \cdot y' \wedge (\rho \cdot y, Y) \in \Theta) \Rightarrow \quad v \cdot \rho = v \cdot \rho' \text{”}. \end{aligned}$$

Note that w has two constant predicate symbols $\subseteq$ and $\bar{\in}$ which are to be given the standard interpretations. We have used $\bar{\in}$ instead of $\in$ because $\in$ already occurs in w as a part of the predicate transformation language. Also $\cdot : U \times U \to U$ as usual stands for $\lambda x, y.(j_{\to} x)y$.

When the specification of (T, w) was fed into the cascade of Goal Generator and Filter, the following single goal was generated: (we are not using the LCF syntax for reading convenience.)

$$\begin{aligned} & \forall (D, Pide), (D', Pide') \in L.\ (D, Pide) \subseteq (D', Pide') \text{ implies} \\ & \quad \forall Y \in \bar{P}ide. Pide(Y) = Pide'(Y). \end{aligned} \tag{5}$$

Acually Goal Generator produced sixteen goals. This does not include the trivial tautologies which were taken care of by Goal Generator itself. Out of these sixteen goals, all were proved by Filter except the above goal. Once again we see how strong the filtering action is.

Proving the goal (5) is not at all difficult. By an easy fixpoint induction on the definition of L one shows that:

$$\forall (D, Pide) \in L.\ Pide = \bar{P}ide.$$

From this (5) follows immediately. Once continuity of (T, w) is proved, what remains is purely routine. Let $((D^*, Pide^*), \Theta^*)$ be the least fixed point of (T, w). Then it follows that $D^* = \bar{D}$ and $Pide^* = \bar{P}ide$. Hence Θ^* is actually a predicate on $\bar{D} \times \bar{P}ide$, and one sees from the definition of (T, w) that it satisfies (4).

If one is pedantic, he will notice that we did not prove one little point. Is (T, w) really a predicator from $\mathcal{L}$ to $\mathcal{L}$, i.e., if $((D, Pide), \Theta) \in \mathcal{L}$, is it always the case that

$$(T, w)((D, Pide), \Theta) \in \mathcal{L}?$$

For this one has to prove that:

$$\forall (D, Pide) \in L.\ T(D, Pide) \in L.$$

But, using the definition of L, automatic theorem provers which have an extensive knowledge of the theory T_3 can prove this almost automatically. One also has to check that w transforms a directed complete predicate into a directed complete predicate. This is trivial. Finally one has to prove that, for all $((D, Pide), \Theta) \in \mathcal{L}$,

$$(T, w)((D, Pide), \Theta) \sqsupseteq (\perp_L, Q).$$

But $\perp_L$ and Q were deliberately chosen such that this will be the case trivially.

Now the existence proof is complete.

16. Conclusion

It is hoped the theory given in this paper meets the long present demand for a mechanizable theory for carrying out the existence proofs of inclusive predicates. The system IPL shows the plausibility of a mechanization and automation of this theory. However, more works needs to be done to make this implementation practical.

17. Acknowledgements

This work was done under the guidence of Prof. Dana Scott. I thank him for the valuable suggestions he gave me. Also thanks to Steve Brookes for his numerous helpful comments and to Roberto Minio for his help in TEX.

REFERENCES

[Gordon1] M.J.C. Gordon, "Towards a Semantic Theory of Dynamic Binding", Memo AIM-265, Computer Science Department, Stanford University.

[Gordon2] M.J.C. Gordon, R.Milner, C.Wadsworth, "Edinburgh LCF", Lecture Notes in Computer Science, 78, Springer-Verlag, 1979.

[Milne] R. Milne, C. Strachey, "A Theory of Programming Language Semantics", Chapman and Hall, London, and John Wiley, New York (1976).

[Mulmuley1] K. Mulmuley, "The Mechanization of Existence Proofs of Recursive Predicates", Proceedings of the Seventh International Conference on Automated Deduction, Napa, California, Springer-Verlag (May, 1984).

[Mulmuley2] K. Mulmuley, " Fully Abstract Submodels Of Typed Lambda Calculi", To appear in the special issue of JCSS on 25th FOCS (1984).

[Mulmuley3] K. Mulmuley, "Full Abstraction And Semantic Equivalence", Ph.D thesis, computer science department, Carnegie-Mellon University, Pittsburgh, Aug. 1985.

[Park] D.M.R.Park, "The Y-combinator in Scott's Lambda Calculus Models", Theory of Computation Report n. 13, University of warwick, U.K., (1976).

[Reynolds] J. Reynolds, "On The Relation Between Direct and Continuation Semantics", pp.141-156 of Proceedings of the Second Colloquim on Automata, Languages and Programming, Saabrücken, Springer-Verlag, Berlin (1974).

[Scott1] D. Scott, "Lectures on a Mathematical Theory of Computation", Technical Monograph PRG-19 (May 1981), Oxford University Computing Laboratory, Programming Research Group.

[Stoy1] J.E.Stoy, "Denotational Semantics", MIT Press, Cambridge, Mass.(1977).

[Stoy2] J.E.Stoy, "The Congruence of Two Programming Language Definitions", TCS 13 (1981), 151-174.