Stanford Artificial Intelligence Laboratory
Memo AIM-265

August 1975

Computer. Science Department
Report No. STAN-CS-75-507

# TOWARDS A SEMANTIC THEORY of DYNAMIC BINDING

by

Michael Gordon

Research sponsored by

Advanced Research Projects Agency
ARPA Order No. 2494

COMPUTER SCIENCE DEPARTMENT
Stanford University

# TOWARDS A SEMANTIC THEORY OF DYNAMIC BINDING

by

Michael Gordon
Department of Computer Science,
James Clerk Maxwell Building,
The King's Buildings,
Mayfield Road,
Edinburgh EH9 3JZ.

## Abstract

The results in this paper contribute to the formulation of a semantic theory of dynamic binding (fluid variables). The axioms and theorems are language independent in that they don't talk about programs - i.e. syntactic objects - but just about elements in certain domains., Firstly the equivalence (in the circumstances where it's true) of "tying a knot" through the environment (elaborated in the paper) and taking a least fixed point is shown. This is central in proving the correctness of LISP "eval" type interpreters, Secondly the relation which must hold between two environments if a program is to have the same meaning in both is established. It is shown how the theory can be applied to LISP to yield previously known facts,

# ACKNOWLEDGEMENTS

Thanks to John Allen, Rod Burstall, Friedrich von Henke, Robert Milne, Gordon Plotkin, Bob Tennent and Chris Wadsworth for helpful discussions and correspondence. John Allen, Dana Scott and Akinori Yonezawa suggested improvements and pointed out errors in preliminary drafts of this report,

This research was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense under contract DAHC 15-73-C-0435, ARPA order no. 2494.

The views and conclusions in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the US Government.

# CONTENTS

SECTION **PAGE**

# 1. Introduction

The art of semantics is now sufficiently developed that most computer languages can be given concise, elegant and intuitive formal descriptions. The theory of these descriptions is well enough understood that useful facts - such as the correctness of implementation8 - are fairly straightforward to prove. Unfortunately proofs tend to be very long and the results obtained rather lacking in generality. For example the proof of correctness of an implementation for one language has to be. redone for a similar implementation of another., Of course once -the proof idea is known no real creative acts are needed in applying it and thus a certain amount of generality is. obtained. However this generality isn't of a type that's easy to use (except, pehaps, by people with considerable knowledge of the underlying theory). A more direct way of being general is to isolate explicitly the assumptions used and then to prove the results from these, Then to apply such a result 'one just needs to check the language satisfies the appropriate "axioms" - and this will normally be much less demanding than redoing a whole proof by analogy with an existing one.

In this note I've formulated abstract versions of two results about languages which use dynamic binding of free variables. Initially these were proved for LISP (they were needed in proving the correctness of an implementation). The abstract versions described below can be instantiated to yield the LISP ones. At hough the two results proven are completely language-independent (in that they don't talk about programs - i.e. synt actic object8 - but just about elements in certain domains) they aren't as general as one might hope. Some situations in which dynamic binding is used and which intuitively should fall under their compass don't. This is a defect of the present work - I don't think it's a necessary difficulty.

## 2. Informal Discussion of Results

When reasoning about programs it's often useful to be able to exhibit the denotation of a recursive procedure as the least fixed point of some functional. Doing this enables, for example, computation-induction to be used. The first result to be discussed helps with this as it concerns the equivalence (in certain circumstances) of "tying a knot" through the environment (elaborated below) with taking a least fixed point. Besides being of interest in its own right, this result is at the heart of the correctness of LISP **eval** type interpreters. Hopefully the abstract version wiill assist in proving the correctness of similar interpreters for other languages.

The way recursive definitions are handled by many LISP implementations is to bind the body of the function to its own name on the **alist.** This creates a circularity or "knot" in which places inside the function body (namely recursive calls) point back to the beginning of the function, Now the standard analysis of recursion is via the Y-operator (i.e. in terms of least fixed points) and consequently in proving the correctness of "knotting" interpreters with respect to standard semantics it's necessary to ascertain the conditions under which "knotting" and fixedpointing are equivalent. Contrary to what one might expect they aren't always the same. This is shown below,

The second result concerns what relation needs to hold between two environments **a,a'** (alists in the case of LISP) for a form e to evaluate to the same values in both a and **a'.** Call this condition **"a=*a'".**

A first guess might be that the two environments must agree on the free variables of e (as is the case for terms in predicate-calculus or the h-calculus). This won't do however for although a and a' might agree on e's free variables the things they bind to these might depend on other variables not free in e and on which a and a' differ (e.g. if e=x, a and a' both bind x to y but a binds y to 1 whilst a' binds it to 2). What is clearly needed is that a and a' agree on e's free variables and on the variables free in the things bound to these variables ... ... etc.

To formulate this for LISP one just needs a recursive definition like:

$a =^{e} a' \iff \forall x.\ [\ x \text{ free in } e \Rightarrow a(x)=a'(x) \text{ and } a =^{a(x)} a'\ ]$

Now given a syntax for e's its easy to formalise "x free in e" - the difficulty arises if one wants a syntax independent definition. What's needed is an abstract notion of free-ness applicable to elements of the type denoted by e (and 'hopefully denoted also by programs from languages other than LISP). I describe such a notion below.

# 3. Formalization

## 3.1. Knots and Fixed-points

Before proceding with abstract formulations of the above it's necessary to describe the environments needed to handle dynamic binding. Let $D$ be an arbitrary domain of expression values and let $Env=Id\rightarrow V_D$ be the associated domain of environments, Elements of $V_D$ are - in the case of dynamic binding - denotations of objects which may contain free variables and so might still depend on the environment. Hence $V_D=Env\rightarrow D$ and thus $Env$ must satisfy $Env=Id\rightarrow(Env\rightarrow D)$.

It turns out to be necessary (see lemma 8 below) to require in addition that if $\rho \in Env$ then $\rho$ is strict i.e. $\rho(\perp)=\perp$ thus if $/D_1 \to D_2/$ is the domain, of strict continuous functions from $D_1$ to $D_2$ then $Env$ must have type satisfying: $Env = Id \to / Env \to D /$.

From this one can immediately formulate what it means for "knotting" and fixedpointing to be the same viz. we require for $v \in V_D$ and $\rho \in Env$:

$$v(\rho[v/x]) = Y(F_x(v))\rho \quad \text{where } F_x(v) = \lambda v'.\lambda \rho'.v(\rho'[v'/x])$$

(knot: $v(\rho[v/x])$; fixedpoint: $Y(F_x(v))\rho$)

here $\rho[v/x]$ is $\rho$ updated to bind $v$ to $x$. Unfortunately this equality isn't true in general.

For example if:

$$v = \lambda \rho'.\rho'(y)\rho' \qquad \text{(where } y \in Id\text{)}$$
$$\rho = \perp[(\lambda \rho'.d)/x][(\lambda \rho'.\rho'(x)\rho)/y] \qquad \text{(where } \perp \neq d \in D\text{)}$$

Then it turns out that $v(\rho[v/x]) = d \neq \perp = Y(F_x(v))\rho$.

For we have:

$$\begin{aligned}
v(\rho[v/x]) &= (\rho[v/x])(y)(\rho[v/x]) && \text{(by definition of v)}\\
&= \rho(y)(\rho[v/x])\\
&= (\lambda \rho'.\rho'(x)\rho)(\rho[v/x]) && \text{(by definition of } p\text{)}\\
&= (\rho[v/x])(x)\rho\\
&= v(\rho)\\
&= \rho(y)\rho && \text{(by definition of v)}\\
&= (\lambda \rho'.\rho'(x)\rho)\rho && \text{(by definition of } p\text{)}\\
&= \rho(x)\rho\\
&= (\lambda \rho'.d)\rho && \text{(by definition of } p\text{)}\\
&= d
\end{aligned}$$

And-as $Y(F_x(v))\rho = \bigsqcup_n F_x(v)^n(\perp)\rho$ and

$F_x(v)^n(\perp)\rho=\perp$ implies

$$\begin{aligned}
F_x(v)^{n+1}(\perp)\rho &= F_x(v)(F_x(v)^n(\perp))\rho \\
&= v(\rho[F_x(v)^n(\perp)/x]) \\
&= (\rho[F_x(v)^n(\perp)/x])(y)(\rho[F_x(v)^n(\perp)/x]) \quad \text{(by definition of } v\text{)} \\
&= (\lambda\rho'.\rho'(x)\rho)(\rho[F_x(v)^n(\perp)/x]) \quad \text{(by definition of } p\text{)} \\
&= F_x(v)^n(\perp)\rho=\perp
\end{aligned}$$

It follows by induction on n that: Vn. $F_x(v)^n(\perp)\rho=\perp$ and so $Y(F_x(v))\rho=\perp$.

In [1] and [2] it is shown that for $v$'s and $\rho$'s which are the denotations of LISP functions and alists respectively the equation $v(\rho[v/x])=Y(F_x(v))\rho$ does in fact hold. The proof used was very specific to LISP (being essentially an induction on the size of computations on a certain abstract interpreter). Now hopefully the result should hold for dynamic binding in general rather than just for LISP. Thus the problem arises of isolating and stating those properties of dynamic binding which, when possesed by $v$ and $\rho$, entail $v(\rho[v/x])=Y(F_x(v))\rho$. To do this we need to introduce recursively defined (but not necessarily monotonic) relations of the type first studied by Milne [5] and Reynolds [7]. Using these we can then provide a (partial) abstract characterisation of dynamic binding by defining a notion of "regular" for which:

$v, \rho$ regular $\Rightarrow v(\rho[v/x])=Y(F_x(v))\rho$

From now on $x,x',x'',\ldots,y,y',y''$ etc. will range over $Id$. $X,Y$ will range over subsets of Id. $\rho,\rho',\rho''$ will range over $Env$. $v,v',v''$ will range over $V_D=Env\to D$ and $d,d',d''$ will range over $D$.

Using techniques developed by Robert Milne of Oxford [5] one can show that there exist predicates of types:

$\lhd \subseteq Env \times Enu$
$\lhd^x \subseteq V_D \times V_D$ (one for each $x \in Id$)
$\leftarrow \subseteq Env \times Enu$
$\leftarrow \subseteq V_D \times V_D$

which are directed-complete (i.e. if they hold of each member of 'a directed set then they hold of the union) and satisfy:

$\rho \lhd \rho' \iff \forall x.\ \rho(x) \lhd^x \rho'(x)$
$v \lhd^x v' \iff \forall \rho,\rho'.\ [\ p\ o\ p\ '\ \Rightarrow v(\rho[v/x]) \sqsubseteq v'(\rho'[v'/x])\ ]$
$\rho \leftarrow \rho' \iff \forall x.\ \rho(x) \leftarrow \rho'(x)$
$v \leftarrow v' \iff \forall \rho,\rho'.\ [\ \rho \lhd \rho' \Rightarrow v(p) \sqsubseteq v'(\rho')\ ]$

One can then show that:

$v \leftarrow v' \Rightarrow v \lhd^x Y(F_x(v'))$
$v' \leftarrow v \Rightarrow Y(F_x(v')) \lhd^x v$

And as it also turns out that $\rho \leftarrow \rho' \Rightarrow \rho \lhd \rho'$ we have:

$v \leftarrow v, \rho \leftarrow \rho \Rightarrow v(\rho[v/x]) = v(\rho[Y(F_x(v))/x]) = Y(F_x(v))$

Thus a definition of "regular" which works is given by;

## Definition 1

$v : Env \to D$ and $\rho : Env$ are regular $\iff v \leftarrow v$ and $\rho \leftarrow \rho$

To apply this to LISP one just shows that the denotations of forms and alists are regular, this is done in section 5.

In the next section proofs of the above assertions will be given *relative* to the existence of the predicates. This existence (which can't be shown with the Y-operator, as the necessary functionals aren't continuous) will be proved in section 6.

## 3.2. Eauivalent Environments

The formulation of the result about free variables also requires the use of **Milne** style recursive predicates viz.:

$\Phi \subseteq V_D \times \{X | X \subseteq Id\}$
$=^X \subseteq Enu \times Enu$ (one for each $X \subseteq Id$)

Where intuitively $\Phi(v,X)$ means the free variables of v are included in X and $\rho =^X \rho'$ means $\rho$ and $\rho'$ "strongly" agree for all $x \in X$. Formally we require that:

$\Phi(v,X) <=> \forall Y,\rho,\rho'.\ [\ X \subseteq Y => [\ \rho =^Y \rho' => v(\rho)=v(\rho')\ ]\ ]$
$\rho =^x \rho' <=> \forall x \in X.\ \rho(x)=\rho'(x)$ and $\Phi(\rho(x),X)$

In section 5 below I'll show that if e is a LISP form which denotes $\mathfrak{C}[\![e]\!]$ and if $vs(e)=\{x | x$ is free in e$\}$ then $\Phi(\mathfrak{C}[\![e]\!],vs(e))$. From this it follows (via the definition of $\rho =^{vs(e)} \rho'$) that:

$\forall \rho,\rho'.\ [\ \rho =^{vs(e)} \rho' => \mathfrak{C}[\![e]\!](\rho)=\mathfrak{C}[\![e]\!](\rho')$ 1

In particular if e has no free variables then $vs(e)=\{\}$ and (since it's clear that for any $\rho$ and $\rho'$: $\rho =^{\{\}} \rho'$) we have $\mathfrak{C}[\![e]\!](\rho)=\mathfrak{C}[\![e]\!](\rho')$.

Somewhat less trivially: if $\forall x \in vs(e).\ \rho(x)=\rho'(x)$ and also $\rho(x)$ is a constant function (i.e. is an environment independent quantity) then again $\rho =^{vs(e)} \rho'$ and so $\mathfrak{C}[\![e]\!](\rho)=\mathfrak{C}[\![e]\!](\rho')$. This last example corresponds to the case for static binding - i.e. when objects have all their free variables bound by the time they themselves are bound. The existence of $\Phi$ and $=^X$ will be discussed in section 6.

## 4. Proofs

Readers from now on are assumed to be familiar with notations commonly employed in the literature on Mathematical Semantics.

A "domain" is a partially ordered set in which each directed subset has a least upper bound. This notion of domain is used (rat her than complete lattices) for minor and nonessential technical reasons (see [1] for a discussion).

The domain intended by $Env=Id\to\!\!/\,Env\to D/$ is the minimal solution of the equation i.e. if id,d are retracts of a universal domain (eg Scott's $D_\infty$) which represent $Id$ and $D$ respectively (in the sence that $Id\equiv\{x|x=id(x)\}$ and $D\equiv\{x|x=d(x)\}$) then $Y(\lambda e.id\to\!\!/(e\to d))$ represents $Enu$. (here $a\to b=\lambda u.\lambda x.b(u(a(x)))$ and $a\to\!\!/b=\lambda u.\lambda x.b(str(u)(a(x)))$ where $str(u)=\lambda x.(x=\bot\to\bot,u(x))$). From this minimality it follows that there are mappings $\lambda\rho.\rho_n:Env\to Env$ such that:

(P1) $\bot=\rho_0\sqsubseteq\rho_1\sqsubseteq\ldots\ldots\sqsubseteq\rho_n\sqsubseteq\ldots\ldots\sqsubseteq\rho$
(P2) $\rho=\bigsqcup\rho_n$
(P3) $(\rho_n)_m=\rho_{\min\{n,m\}}$
(P4) $\rho_{n+1}(x)\rho'=\rho(x)\rho'_n$

In fact if $Enu$ is represented as above then $\rho_n=(\lambda e.id\to\!\!/(e\to d))^n(\bot)(\rho)$. For $v\in/Env\to D/$ $v_n$ is defined by $v_n(\rho)=v(\rho_n)$. (P4) can thus be written as: $\rho_{n+1}(x)=\rho(x)_n$ and it is easy to show (see [1] for details) that: $\rho[v/x]_{n+1}=\rho_{n+1}[v_n/x]$.

I shall prove $[\,v\prec v'\Rightarrow v\lhd^x Y(F_x(v'))\,]$ by showing (by induction on n) that $[\,v\prec v'\Rightarrow v_n\lhd^x Y(F_x(v'))\,]$ and then take a limit. Similarly $[\,v\prec v'\Rightarrow Y(F_x(v))\lhd^x v'\,]$ will be proved by showing that for all n: $[\,v\prec v'\Rightarrow F_x(v)^n(\bot)(v)\lhd^x v'\,]$.

The following rat her ad-hoc looking definition enables the clean statement of some of the lemmas below:

## Definition 2

$F:V_D \to V_D$ is "invarient at $x$" $\iff \forall \rho, v.\ F(v)(\rho[F(v)/x]) = v(\rho[F(v)/x])$

The useful applications of this definition are given in the next lemma.

## Lemma 1

For all x $(\lambda v.v)$ and $(\lambda v.Y(F_x(v)))$ are both-invarient at $x$.

## Proof'

Trivial for $(\lambda v.v)$, for $(\lambda v.Y(F_x(v)))$ use the fixed-point property of Y.

QED.

## Lemma 2

If F is invarient at x and $v \triangleleft v'$ then Vn. $v_n \triangleleft^x F(v')$.

## Proof

**n=0:** Must show $v_0 \triangleleft^x F(v')$

i.e. $\rho \triangleleft \rho' \Rightarrow v_0(\rho[v_0/x]) \sqsubseteq F(v')(\rho[F(v')/x])$

i.e. $\rho \triangleleft \rho' \Rightarrow v(\bot) \sqsubseteq v'(\rho[F(v')/x])$

OK as $v \triangleleft v'$ and $\bot \triangleleft \rho[F(v')/x]$

**n>0:** Assume true for n-l. Let $\rho \triangleleft \rho'$. Must show $v_n(\rho[v_n/x]) \sqsubseteq F(v')(\rho[F(v')/x])$

i.e. $v(\rho_n[v_{n-1}/x]) \sqsubseteq v'(\rho[F(v')/x])$

need $\rho_n[v_{n-1}/x] \triangleleft \rho[F(v')/x]$

need $v_{n-1} \triangleleft^x F(v')$ - OK by induction.

QED.

Lemma 3

If F is invariant at x and $v \lhd v'$ then $v \lhd^x F(v')$

Proof

Trivial from lemma 2 as $v = \bigsqcup_n v_n$ and $\lhd^x$ is directed-complete.

QED.

Lemma 4

$\forall x. [\, v \lhd v' \Rightarrow v \lhd^x v' \,]$
$\forall x. [\, v \lhd v' \Rightarrow v \lhd^x Y(F_x(v')) \,]$

Proof

Trivial consequence of lemmas 1 and 3 ,

**QED.**

Lemma S

If F is invarient at x and $v \lhd v'$ then $\forall n. F_x(v)^n(\bot) \lhd^x F(v')$.

Proof

n=0: Trivial

n>0: Assume true for n-1. Need $\rho \lhd \rho' \Rightarrow F_x(v)^n(\bot)(\rho[F_x(v)^n(\bot)/x]) \sqsubseteq F(v')(\rho[F(v')/x])$
i.e. $\rho \lhd \rho' \Rightarrow v(\rho[F_x(v)^{n-1}(\bot)/x]) \sqsubseteq v'(\rho[F(v')/x])$
OK if $F_x(v)^{n-1}(\bot) \lhd^x F(v')$ - true by induction

**QED.**

Lemma 6

If F is invarient at x and $v \lhd v'$ then $Y(F_x(v)) \lhd^x F(v')$.

**Proof**

Trivial from *lemma* 5 *as* $Y(F_x(v)) = \bigsqcup_n F_x(v)^n(\bot)$ and $\lhd^x$ is directed-complete,

QED.

Lemma 7

$\forall x.\ [\ v \triangleleft v' \Rightarrow Y(F_x(v)) \triangleleft^x v'\ ]$
$\forall x.\ [\ v \triangleleft v' \Rightarrow Y(F_x(v)) \triangleleft^x Y(F_x(v'))\ ]$

Proof

Trivial application of lemma 1 and lemma 6.

**QED.**

*Theorem* 1

If v and $\rho$ are regular then $v(\rho[v/x]) = Y(F_x(v))\rho$

Proof

By lemma 5 and lemma 7 we have:

$$Y(F_x(v)) \triangleleft^x v$$
$$v \triangleleft^x Y(F_x(v))$$

hence from the definition of $\triangleleft^x$

$$Y(F_x(v))(\rho[Y(F_x(v))/x]) \sqsubseteq v(\rho[v/x])$$
$$v(\rho[v/x]) \sqsubseteq Y(F_x(v))(\rho[Y(F_x(v))/x])$$

hence

$$Y(F_x(v))(\rho[Y(F_x(v))/x]) = v(\rho[v/x])$$

Finally, using the fixed-point property of $Y$ on the left hand side of this, we get:

$$Y(F_x(v))\rho = v(\rho[v/x])$$

QED.

## 5. Application to LISP

In this section **D** will be specialized to a domain appropriate for pure LISP and then the abstract results described above will be shown to hold of the denotations of LISP programs.

The semantics of LISP used here will only be described in barest outline, For furthur details, motivation and justification see [1] and [2].

## 5.1. Syntax

The syntax of LISP (as described in the manual [4] and in the notation of [9]) is given by the equations:

$$e ::= A \mid x \mid fn[e_1;\ldots;e_n] \mid [e_{11}\rightarrow e_{12};\ldots;e_{n1}\rightarrow e_{n2}]$$
$$fn ::= F \mid f \mid \lambda[[x_1;\ldots;x_n];e] \mid label[f;fn]$$
$$F ::= car \mid cdr \mid cons \mid atom \mid eq$$

where the ranges of the variables e,A,x,fn,F,f are as follows:

| | | | |
|---|---|---|---|
| A | ranges over | <S-expression> | (as in page 9 of [4]) |
| x,f,z | range over | <identifier> | (as in page 9 of [4]) |
| e | ranges over | <form> | (as defined above) |
| fn | ranges over | <function> | (as defined above) |
| F | ranges over | <standard function> | (as defined above) |

I use meta-variables x,f,z to range over <identifier>: x is used in contexts where the identifier is a form, f where it's a function and z where it could be either.

## 5.2. Some Notation

In the semantics below:

flat(S)=S U {ı} ordered by $\forall s \in S.\ \bot \sqsubseteq s$.

$$\underline{\lambda} s_1,\ldots,s_n.E(s_1,\ldots,s_n) = \lambda s_1,\ldots,s_n.(s_1=\bot \text{ or } s_2=\bot \text{ or } \ldots \text{ or } s_n=\bot \rightarrow \bot, E(s_1,\ldots,s_n))$$

car,cdr,cons,atom,eq are the appropriate functions on $S$=flat(<S-expression>).

Whenever an expression v of type S, $[Env\rightarrow S]$ or $[Env\rightarrow Funval]$ occurs in a context requiring something of type $[Env\rightarrow D]$ then v means (i.e should be "coerced" into) $(\lambda\rho.v \text{ in } D)$, $(\lambda\rho.v(\rho) \text{ in } D)$ and $(\lambda\rho.v(\rho) \text{ in } D)$ respectively.

## 5.3. Semantics

### 5.3.1. Denotation Domains

$$D = S + Funval$$
$$S = \text{flat}(\langle \text{S-expression} \rangle)$$
$$Funval = [S^* \to S]$$

### 5.3.2. Environment Domain

$$Env = Id \to [Env \to D]$$

### 5.3.3. Semantic Functions

$$\mathfrak{E}: Form \to [Env \to S]$$
$$\mathfrak{F}: Function \to [Env \to Funval]$$

### 5.3.4. Semantic Equations

(S1) $$\mathfrak{E}[\![A]\!]\rho = A$$

(S2) $$\mathfrak{E}[\![x]\!]\rho = \rho(x)\rho|S$$

(S3) $$\mathfrak{E}[\![fn[e_1;\ldots;e_n]]\!]\rho = \mathfrak{F}[\![fn]\!]\rho(\mathfrak{E}[\![e_1]\!]\rho,\ldots,\mathfrak{E}[\![e_n]\!]\rho)$$

(S4) $$\mathfrak{E}[\![[e_{11}\to e_{12};\ldots;e_{n1}\to e_{n2}]]\!]\rho = (\mathfrak{E}[\![e_{11}]\!]\rho \to \mathfrak{E}[\![e_{12}]\!]\rho,\ldots,\mathfrak{E}[\![e_{n1}]\!]\rho \to \mathfrak{E}[\![e_{n2}]\!]\rho)$$

(S5)
$$\mathfrak{F}[\![car]\!]\rho = \underline{car}$$
$$\mathfrak{F}[\![cdr]\!]\rho = \underline{cdr}$$
$$\mathfrak{F}[\![cons]\!]\rho = \underline{cons}$$
$$\mathfrak{F}[\![atom]\!]\rho = \underline{atom}$$
$$\mathfrak{F}[\![eq]\!]\rho = \underline{eq}$$

(S6) $$\mathfrak{F}[\![f]\!]\rho = \rho(f)\rho|Funval$$

(S7) $$\mathfrak{F}[\![\lambda[[x_1;\ldots x_n];e]]\!]\rho = \underline{\lambda} s_1,\ldots,s_n : S.\mathfrak{E}[\![e]\!]\rho[s_1/x_1]\ldots[s_n/x_n]$$

(S8) $$\mathfrak{F}[\![label[f;fn]]\!]\rho = Y(F_f(\mathfrak{F}[\![fn]\!]))\rho$$

Theorem 2 below shows that the denotations of LISP, forms and functions are regular and so Theorem 1 can be applied to them.

**Theorem 2**

$\mathfrak{E}[\![e]\!] \lhd \mathfrak{E}[\![e]\!]$ and $\mathfrak{F}[\![fn]\!] \lhd \mathfrak{F}[\![fn]\!]$

Proof

A straightforward induction works, The details are as follows:
Assume $\rho \lhd \rho'$. I must show $\mathfrak{E}[\![e]\!]\rho \sqsubseteq \mathfrak{E}[\![e]\!]\rho'$ and $\mathfrak{F}[\![fn]\!]\rho \sqsubseteq \mathfrak{F}[\![fn]\!]\rho'$.

(1): $\mathfrak{E}[\![A]\!]\rho = A \sqsubseteq A = \mathfrak{E}[\![e]\!]\rho'$

(2): $\mathfrak{E}[\![x]\!]\rho = \rho(x)\rho | S$
$\mathfrak{E}[\![x]\!]\rho' = \rho'(x)\rho' | S$
*Now* $\rho \lhd \rho' \Rightarrow \rho(x) \lhd^x \rho'(x) \Rightarrow \rho(x)(\rho[\rho(x)/x]) \sqsubseteq \rho'(x)(\rho'[\rho'(x)/x])$
$\Rightarrow \rho(x)(\rho) \sqsubseteq \rho'(x)\rho'$ by lemma 8 below

(3): $\mathfrak{E}[\![fn[e_1;\ldots;e_n]]\!]\rho = \mathfrak{F}[\![fn]\!]\rho(\mathfrak{E}[\![e_1]\!]\rho,\ldots,\mathfrak{E}[\![e_n]\!]\rho)$
$\sqsubseteq \mathfrak{F}[\![fn]\!]\rho'(\mathfrak{E}[\![e_1]\!]\rho',\ldots,\mathfrak{E}[\![e_n]\!]\rho')$
$= \mathfrak{E}[\![fn[e_1;\ldots;e_n]]\!]\rho'$

(4): $\mathfrak{E}[\![[e_{11}\rightarrow e_{12};\ldots;e_{n1}\rightarrow e_{n2}]]\!]\rho = (\mathfrak{E}[\![e_{11}]\!]\rho \rightarrow \mathfrak{E}[\![e_{12}]\!]\rho,\ldots,\mathfrak{E}[\![e_{n1}]\!]\rho \rightarrow \mathfrak{E}[\![e_{n2}]\!]\rho)$
$\sqsubseteq (\mathfrak{E}[\![e_{11}]\!]\rho' \rightarrow \mathfrak{E}[\![e_{12}]\!]\rho',\ldots,\mathfrak{E}[\![e_{n1}]\!]\rho' \rightarrow \mathfrak{E}[\![e_{n2}]\!]\rho')$
$= \mathfrak{E}[\![[e_{11}\rightarrow e_{12};\ldots;e_n \rightarrow e_{n2}]]\!]\rho'$

(5): $\mathfrak{F}[\![F]\!]\rho = F \sqsubseteq F = \mathfrak{F}[\![F]\!]\rho'$

(6): $\mathfrak{F}[\![f]\!]\rho = \rho(f)\rho | Fun$
$\mathfrak{F}[\![f]\!]\rho' = \rho'(f)\rho' | Fun$
and $\rho(f)\rho \sqsubseteq \rho'(f)\rho'$ as in (2) above,

(7): $\mathfrak{F}[\![\lambda[[x_1;\ldots;x_n];e]]\!]\rho = \underline{\lambda} s_1,\ldots,s_n.\mathfrak{E}[\![e]\!]\rho[s_1/x_1]\ldots[s_n/x_n]$
$\mathfrak{F}[\![\lambda[[x_1;\ldots;x_n];e]]\!]\rho' = \underline{\lambda} s_1,\ldots,s_n.\mathfrak{E}[\![e]\!]\rho'[s_1/x_1]\ldots[s_n/x_n]$
so it suffices to show $\rho[s_1/x_1]\ldots[s_n/x_n] \lhd \rho'[s_1/x_1]\ldots[s_n/x_n]$
and for this it suffices to show $\lambda\rho.(s_i \text{ in} D) \lhd^{x_i} \lambda\rho.(s_i \text{ in} D)$
i.e. $\rho \lhd \rho' \Rightarrow (\lambda\rho.s_i)(\rho[(\lambda\rho.s_i)/x_i]) \sqsubseteq (\lambda\rho.s_i)(\rho'[(\lambda\rho.s_i)/x_i])$
i.e. $\rho \lhd \rho' \Rightarrow s_i \sqsubseteq s_i$ - which is true,

(8): $\mathfrak{F}[\![\text{label}[f;fn]]\!]\rho = Y(F_f(\mathfrak{F}[\![fn]\!]))\rho$
$\mathfrak{F}[\![\text{label}[f;fn]]\!]\rho' = Y(F_f(\mathfrak{F}[\![fn]\!]))\rho'$
hence result by lemma 7,

QED.

Lemma 8

$\forall \rho,x.\ \rho = \rho[\rho(x)/x]$

Proof

Follows trivially from definition of "$\rho[\rho(x)/x]$" and strictness of $\rho$.

QED.

Theorem 3 below shows that if vs(e) is the set of free variables in e then in the abstract sense discussed above the free variables of $\mathfrak{E}[\![e]\!]$ "are included in" vs(e).

. The following lemma is needed for the proof. The definitions of $\Phi$ and $=^X$ are on page 7.

Lemma 9

(1) $\forall v,X,Y.\ [\ \Phi(v,X),\ X \subseteq Y \Rightarrow \Phi(v,Y)\ ]$

(2) $\forall d.\ \Phi((\lambda\rho.d),\{\})$

(3) $\forall v,x,X.\ [\ \Phi(v,X) \Rightarrow \Phi(Y(F_x(v)),X\backslash\{x\})$ I

Proof

(1): Trivial,

(2): Trivial.

(3): I show $\Phi(v,X) \Rightarrow \Phi(F_x(v)^n(\perp),X\cup\{x\})$ by induction on n. Assume $\Phi(v,X)$.

n=O: $\Phi(\perp,X\backslash\{x\})$ is clearly true.

n>0: Assume true for n-1.

$\Phi(F_x(v)^n(\perp),X\backslash\{x\}) \Leftrightarrow \rho=^{X\backslash\{x\}}\rho' \Rightarrow F_x(v)^n(\perp)\rho=F_x(v)^n(\perp)\rho'$

$\Leftrightarrow \rho=^{X\backslash\{x\}}\rho' \Rightarrow v(\rho[F(v)^{n-1}(\perp)/x])=v(\rho'[F_x(v)^{n-1}(\perp)/x]).$

$\Leftarrow \rho=^{X\backslash\{x\}}\rho' \Rightarrow \rho[F_x(v)^{n-1}(\perp)/x]=^X\rho'[F_x(v)^{n-1}(\perp)/x]$

which is true by induction and (1) above.

**QED.**

## Theorem 3

$\forall e \in$ <form>. $\Phi(\mathfrak{E}[\![e]\!],vs(e))$

$\forall e \in$ <funct ion>. $\Phi(\mathfrak{F}[\![fn]\!],vs(fn))$

## Proof

A straight forward structural induct ion works. Let $vs(e) \subseteq X$.

**e=x:**

Must show $\rho=^X\rho' \Rightarrow \rho(x)\rho=\rho'(x)\rho'$. Now $vs(e)=\{x\}\subseteq X$ so if $\rho=^X\rho'$:
$\rho(x)=\rho'(x)$ and $\Phi(\rho(x),X)$ hence $\rho(x)\rho=\rho(x)\rho'=\rho'(x)\rho'$.

**e=A:**

Must show $\rho=^X\rho' \Rightarrow \mathfrak{E}[\![A]\!]\rho=\mathfrak{E}[\![A]\!]\rho'$ - which is clearly true.

e=f n[ $e_1;\ldots;e_n$]:

we have by induction that $\Phi(\mathfrak{F}[\![fn]\!],vs(fn))$ and $\Phi(\mathfrak{E}[\![e_i]\!],vs(e))$.
Hence by lemma 9 $\Phi(\mathfrak{F}[\![fn]\!],X)$ and $\Phi(\mathfrak{E}[\![e_i]\!],X)$ as $vs(fn),vs(e_i)\subseteq vs(e)\subseteq X$.
So if $\rho=^X\rho'$ then $\mathfrak{F}[\![fn]\!]\rho=\mathfrak{F}[\![fn]\!]\rho'$ and $\mathfrak{E}[\![e_i]\!]\rho=\mathfrak{E}[\![e_i]\!]\rho'$
and hence $\mathfrak{E}[\![e]\!]\rho=\mathfrak{E}[\![e]\!]\rho'$.

**$e=[e_{11}\rightarrow e_{12};\ldots;e_{n1}\rightarrow e_{n2}]$:**

Argument as above.

Now let $vs(fn)\subseteq X$.

fn=f:

Simi tar to **"e=x"** case above.

fn=F:

Similar to **"e=A"** case above,

$fn=\lambda[[x_1;...;x_n];e]$:
$\mathfrak{F}[\![\lambda[[x_1;...x_n];e]]\!]\rho = \underline{\lambda} s_1,...,s_n;S.\mathfrak{E}[\![e]\!]\rho[s_1/x_1]...[s_n/x_n]$
$vs(fn)=vs(e)\backslash\{x_1,...,x_n\}$ so $vs(e)\subseteq X\cup\{x_1, \ ,x_n\}$.
Now by lemma 9(1,2) if $Y=X\cup\{x_1,...,x_n\}$ then
$\rho=^X\rho' \Rightarrow \rho[s_1/x_1]...[s_n/x_n]\lhd^Y\rho'[s_1/x_1]...[s_n/x_n]$
so as $\Phi(\mathfrak{E}[\![e]\!],vs(e))$: $\mathfrak{E}[\![e]\!]\rho[s_1/x_1]...[s_n/x_n]=\mathfrak{E}[\![e]\!]\rho'[s_1/x_1]...[s_n/x_n]$.

$fn=label[f;fn_1]$:
We have by induction $\Phi(\mathfrak{F}[\![fn_1]\!],vs(fn_1))$ where $vs(fn_1)\backslash\{f\}=vs(fn)\subseteq X$.
So by lemma 9(3) and induction $\Phi(\mathfrak{F}[\![fn]\!],vs(fn_1)\backslash\{f\})$
hence $\Phi(\mathfrak{F}[\![fn]\!],X)$.

QED.

As an application one can show that adding new definitions to an environment doesn't change the values of the old ones as long as previously used variables aren't overwritten. This is an important lemma needed in proving the correctness of **eval**. Here it's a trivial consequence of Theorem 3 but originally (see [1]) it needed a long ad-hoc proof which confused general arguments with LISP specific ones. To see how it follows consider an environment $\rho$ which defines a set of functions all of whose free variables are included in $X\subseteq Id$. Suppose x is a new function not included in X. We wish to show that if e is a form (or function) then as long as $vs(e)\subseteq X$ (i.e. e only uses the old functions) we have for any v: $\mathfrak{E}[\![e]\!]\rho=\mathfrak{E}[\![e]\!]\rho[v/x]$. But this is now trivial for $\Phi(\mathfrak{E}[\![e]\!],X)$ and $\rho=^X\rho[v/x]$. Saying this formally yields the following theorem (in which "$\rho[v/x]$" above is replaced by "$\rho'$").

Theorem 4

suppose $\rho,\rho' \in Env$, e∈<form> are such that for some $X \subseteq Id$ we have:

(1) $\forall x \in X.\exists fn_x \in$ <function% $\rho(x)=\rho'(x)=\mathfrak{F}[\![fn_x]\!]$ and $vs(fn_x) \subseteq X$.
(2) $vs(e) \subseteq X$

then $\mathfrak{G}[\![e]\!]\rho=\mathfrak{G}[\![e]\!]\rho'$.

Proof

By theorem 3 $\Phi(\mathfrak{G}[\![e]\!],X)$ and $\rho=^X\rho'$. The result follows from the definition of $\Phi$.

QED.

## 6. Existence of Predicates

In all the above the existence of the predicates $\lhd,\lhd^x,<,\Phi,=^X$ has been assumed, However this existence cannot be deduced immedeately from the recursive definitions 8s the predicates being defined arn't necessarily monotonic . The existence proofs to be described are directly based on techniques developed by Robert Milne [5]. Similar methods have recently been independently discovered by Reynolds [7]. For the current purposes it's only necessary to know that the required predicates exist, however Milne's work shows one can expect them to be unique also. I havn't checked this for the predicates used here.

We define by induction on n predicates:

$\lhd_n \subseteq Env \times Env$
$\lhd^x_n \subseteq V_D \times V_D$

and then set:

$\rho \lhd \rho' \iff \forall n.\ \rho \lhd_n \rho'$

$v \lhd^x v' \iff \forall n.\ v \lhd^x_n v'$

it follows (details below) that $\lhd, \lhd^x$ satisfy the desired equations and are directed-complete.

Definition 3

$\rho \lhd_n \rho' \iff \forall x.\ \rho(x) \lhd^x_n \rho'(x)$

$v \lhd^x_0 v' \iff v(\bot[v_0/x]) \sqsubseteq v'(\bot[v'/x])$

$v \lhd^x_{n+1} V' \iff \forall \rho,\rho'.\ [\ \rho \lhd_n \rho' \Rightarrow v_{n+1}(\rho[v/x]) \sqsubseteq v'(\rho'[v'/x])\ ]$

The following two lemmas are needed to prove Theorem 5 below,

Lemma **10**

$(1_n)$ $\forall \rho,\rho'.\ [\ \rho \lhd_{n+1} \rho' \Rightarrow \rho \lhd_n \rho'\ ]$

$(2_n)$ $\forall \rho,\rho'.\ [\ \rho \lhd_n \rho' \Rightarrow \rho_{n+1} \lhd_{n+1} \rho'\ ]$

$(3_n)$ $\forall v,v'.\ [\ v \lhd^x_{n+1} v' \Rightarrow v \lhd^x_n v'\ ]$

$(4_n)$ $\forall v,v'.\ [\ v \lhd^x_n v' \Rightarrow v_n \lhd^x_{n+1} V'\ ]$

Proof

I show that $(3_0), (4_0), (3_n) \Rightarrow (1_n), (4_n) \Rightarrow (2_n), (2_{n-1}) \Rightarrow (3_n), (1_{n-1}) \Rightarrow (4_n)$

(3,): Must show $v \lhd^x_1 v' \Rightarrow v \lhd^x_0 v'$. Clearly $\bot \lhd_0 \bot$ and we have:

$$v \lhd^x_1 V',\ \bot \lhd_0 \bot \Rightarrow v_1(\bot[v/x]) \sqsubseteq v'(\bot[v'/x])$$
$$\Rightarrow v(\bot[v_0/x]) \sqsubseteq v'(\bot[v'/x])$$
$$\iff v \lhd^x_0 v'$$

$(4_0)$: Must show $v \lhd^x_0 v' \Rightarrow v_0 \lhd^x_1 v'$.

Assume $v \lhd^x_0 v'$ and $\rho \lhd_0 \rho'$.

Must show $v_{01}(\rho[v_0/x]) \sqsubseteq v'(\rho'[v'/x])$

i.e. $v(\bot) \sqsubseteq v'(\rho'[v'/x])$

but $v(\bot) \sqsubseteq v(\bot[v_0/x]) \sqsubseteq v'(\bot[v'/x]) \sqsubseteq v'(\rho'[v'/x])$.

$(3_n) \Rightarrow (1_n)$: Assume $(3_n)$. To show $(1_n)$ let $\rho \lhd_{n+1} \rho'$.

Must show $\rho \lhd_n \rho'$ i.e. $\forall x.\rho(x) \lhd^x_n \rho'(x)$.

But if $\rho \lhd_{n+1} \rho'$ then $\forall x.\rho(x) \lhd^x_{n+1} \rho'(x)$ so $\forall x.\rho(x) \lhd^x_n \rho'(x)$ by $(3_n)$.

$(4_n) \Rightarrow (2_n)$: $\rho \lhd_n \rho' \iff \forall x.\rho(x) \lhd^x_n \rho'(x)$

$\Rightarrow \forall x.\rho(x)_n \lhd^x_{n+1} \rho'(x)$ by $(4_n)$

$\Rightarrow \forall x.\rho_{n+1}(x) \lhd^x_{n+1} \rho'(x)$

$\Rightarrow \forall x.\rho_{n+1} \lhd_{n+1} \rho'$

$(2_{n-1}) \Rightarrow (3_n)$: Assume $(2_{n-1})$. To show $(3_n)$ let $v \lhd^x_{n+1} v'$ and $\rho \lhd_{n-1} \rho'$.

Then $\rho_n \lhd_n \rho'$ from $(2_{n-1})$.

So $v_{n+1}(\rho_n[v/x]) \sqsubseteq v'(\rho'[v'/x])$

i.e. $v(\rho_n[v_n/x]) \sqsubseteq v'(\rho'[v'/x])$

hence $v_n(\rho[v/x]) = v(\rho_n[v_{n-1}/x])$

$\sqsubseteq v(\rho_n[v_n/x])$

$\sqsubseteq v'(\rho'[v'/x])$.

$(1_{n-1}) \Rightarrow (4_n)$: Assume $(1_{n-1})$. To show $(4_n)$ let $v \lhd^x_n v'$ and $\rho \lhd_n \rho'$.

Then $\rho \lhd_{n-1} \rho'$ so $v_n(\rho[v/x]) \sqsubseteq v'(\rho[v'/x])$

hence $(v_n)_{n+1}(\rho[v_n/x]) = v(\rho_n[v_{n-1}/x])$

$= v_n(\rho[v/x])$

$\sqsubseteq v'(\rho[v'/x])$

QED.

## Lemma 11

If $\{v_\alpha\}$ is directed then $[[\forall \alpha.\, v_\alpha \lhd^x_n v'] \Rightarrow [(\bigsqcup_\alpha v_\alpha) \lhd^x_n v']]$.

Proof

Cases on n:

n=0: $v_\alpha \lhd^x_0 v' \iff v_\alpha(\bot[v_{\alpha 0}/x]) \sqsubseteq v'(\rho\,[v'/x])$
so $\bigsqcup_\alpha v_\alpha(\bot[\bigsqcup_\alpha v_{\alpha 0}/x]) \sqsubseteq v'(\rho[v'/x])$.

n>0: Let $\rho \lhd_{n-1} \rho'$ then $\forall \alpha.\forall x.\ v_{\alpha n}(\rho[v_\alpha/x]) \sqsubseteq v'(\rho'[v'/x])$
so $(\bigsqcup_\alpha v_\alpha)_n(\rho[\bigsqcup_\alpha v_\alpha/x]) \sqsubseteq v'(\rho'[v'/x])$
hence $\bigsqcup_\alpha v_\alpha \lhd^x_n v'$.

**QED.**

## Theorem 5

$\lhd$ and $\lhd^x$ are directed-complete and satisfy:

$$\rho \lhd \rho' \iff \forall x.\ \rho(x) \lhd^x \rho'(x)$$
$$v \lhd^x v' \iff \forall \rho,\rho'.\ [\ \rho \lhd \rho' \Rightarrow v(\rho\,[v/x]) \sqsubseteq v'(\rho\,'[v'/x])\ ]$$

Proof

To show $\lhd^x$ directed-complete we have:

$$\begin{aligned}\forall \alpha.\, v_\alpha \lhd^x v' &\iff \forall \alpha.\forall n.\ v_\alpha \lhd^x_n v' \\ &\iff \forall n.\forall \alpha.\ v_\alpha \lhd^x_n v' \\ &\Rightarrow \forall n.\ \bigsqcup_\alpha v_\alpha \lhd^x_n v' \quad \text{by, lemma 11} \\ &\iff \bigsqcup_\alpha v_\alpha \lhd^x v'\end{aligned}$$

Showing $[[\forall \alpha.\, v \lhd^x v'_\alpha] \Rightarrow [v \lhd^x (\bigsqcup_\alpha v'_\alpha)]]$ is trivial.

The **directed-completeness of** $\lhd$ follows directly from its definition and the directed-completeness of $\lhd^x$ for all **x**.

To prove the rest of the theorem we have:

$\rho \lhd \rho' \iff \forall n.\forall x.\ \rho(x) \lhd^x_n \rho'(x)$
$\iff \forall x.\forall n.\ \rho(x) \lhd^x_n \rho'(x)$
$\iff \forall x.\ \rho(x) \lhd^x \rho'(x)$

To show $v \lhd^x v' \Rightarrow \forall \rho,\rho'.\ [\rho \lhd \rho' \Rightarrow v(\rho[v/x]) \sqsubseteq v'(\rho'[v'/x])]$ assume $v \lhd^x v'$ and $\rho \lhd \rho'$.
Then $\forall n.\ v \lhd^x_{n+1} v', \rho \lhd \rho'$
so $v_{n+1}(\rho[v/x]) \sqsubseteq v'(\rho'[v'/x])$
hence unioning over n: $v(\rho[v/x]) \sqsubseteq v'(\rho'[v'/x])$.

To show $\forall \rho,\rho'.\ [\rho \lhd \rho' \Rightarrow v(\rho[v/x]) \sqsubseteq v'(\rho'[v'/x])] \Rightarrow v \lhd^x v'$ 'assume
$\rho \lhd \rho' \Rightarrow v(\rho[v/x]) \sqsubseteq v'(\rho'[v'/x])$. I show $\forall n.\ v \lhd^x_n v'$ by induction on n.

n=0: $\bot \lhd \bot$ so $v(\bot[v/x]) \sqsubseteq v'(\bot[v'/x])$ so $v(\bot[v_0/x]) \sqsubseteq v(\bot[v/x]) \sqsubseteq v'(\bot[v'/x])$
so $v \lhd^x_0 v$.

n>0: By lemma 10: $\rho \lhd \rho' \Rightarrow \rho \lhd_{n-1} \rho' \Rightarrow \rho_n \lhd_n \rho' \Rightarrow \forall m.\ \rho_n \lhd_m \rho'$.
so $\rho_n \lhd \rho'$.
Hence $v(\rho_n[v/x]) \sqsubseteq v'(\rho'[v'/x])$ so $v_n(\rho[v_n/x]) \sqsubseteq v(\rho_n[v/x]) \sqsubseteq v'(\rho'[v'/x])$.
Thus $v \lhd^x_n v'$.

So $\forall n.\ v \lhd^x_n v'$ and hence $v \lhd^x v'$.

**QED.**

The construction of $\Phi$ and $=^X$ is very similar to the construction above. As before we start by defining "finite" approximations to the relations viz.

Definition 4

$\Phi_n(v,X) \iff \forall Y,\rho,\rho'.\ [\ X \subseteq Y \Rightarrow [\rho =^Y_n \rho' \Rightarrow v_n(\rho) = v_n(\rho')]\ ]$

$\rho =^X_0 \rho' = \text{true}$
$\rho =^X_{n+1} \rho' \iff \forall x \in X.\ \rho(x) = \rho'(x)$ and $\Phi_n(\rho(x),X)$

We then prove a lemma similar to lemma 10 viz.

Lemma 12

(1,) $\forall v,X.\ [\ \Phi_{n+1}(v,X) \Rightarrow \Phi_n(v,X)\ ]$

(2$_n$) $\forall v,X.\ [\ \Phi_n(v,X) \Rightarrow \Phi_{n+1}(v_n,X)\ ]$

(3$_n$) $\forall \rho,\rho',X.\ [\ \rho =^X_{n+1} \rho' \Rightarrow \rho =^X_n \rho'\ ]$

(4$_n$) $\forall \rho,\rho',X.\ [\ \rho =^X_n \rho' \Rightarrow \rho_n =^X_{n+1} \rho'\ ]$

Proof

Same as lemma 10 (mutatis mutandis).

QED.

From this it follows that if we define $\Phi$ and $=^X$ by:

$\Phi(v,X) \Leftrightarrow \forall n.\ \Phi_n(v,X)$
$\rho =^X \rho' \Leftrightarrow \forall n.\ \rho =^X_n \rho'$

then $\Phi$ and $=^X$ have the desired properties.

## 7. Concluding Remarks

We have presented above a partial axiomatization of dynamic binding. What has been shown is that if $v \in [Env \to D]$ satisfies $v \leftarrow v$ (i.e. is regular) and $\Phi(v,X)$ for some $X \subseteq Id$ then useful theorems follow. What is left open is just how many other axioms will eventually be required. To answer this we need first to know which theorems we want and to answer this we must attack "real" problems such as the correctness of compilers and interpreters. Doing this should reveal the general theorems about dynamic binding that must follow from any adequate theory.

The theorems proved here are not yet general enough. For example if we consider the obvious extension of the semantics to handle funargs (see [ 1]) then the proofs that $\mathfrak{C}[\![e]\!]$ and $\mathfrak{F}[\![fn]\!]$ are regular fail. in fact by replacing the occurences of "$\sqsubseteq$" in the definitions of $\lhd,\lhd^x$ and $\leftarrow$ by another predicate (which needs to be defined recursively) it's easy to cover this case. Unfortunately I don't at present see a uniform way of defining $\lhd,\lhd^x$ and $\leftarrow$ to cover ail useful $D$.

Having to separately prove the existence of all predicates is a big nuisance, One step toward a general justification of -recursive predicate definitions has been provided by Milne and Reynolds. Both give uniform accounts of how to construct recursive predicates from their defining equations. In fact the constructions given above are (more or less) instances of Milnes techniques. It would help a lot if syntactic criteria on definitions could be developed to decide if the things purported to be defined actually exist. Milne [private communication] has made progress toward this by anaiysing the structure of some of the expressions which occur in definitions and showing that these legitimate instances of his general construction.

It's clear that many of the above proofs can't be done in existing fomalisms (eg LCF) - the required predicates can't be defined in them. One way to fix this would be to develop extensions, another would -be to develop a translater from proofs using predicat es to proofs which don't. The latter probably won't be adequate because theorems. may require the use of predicates in their statement at the general level (even if ail their useful instances don't).

# 8. References

[1] Gordon, M.J.C. (1973) Models of pure LISP. Experimental Programming Report s:No.31. Department of Machine Int elligence, School of Artificial Intelligence, University of Edinburgh.

[2] Gordon, M.J.C. (1975) Operational Reasoning and Denotational Semantios. Presented at the International Symposium on Proving and Improving Programs, Arc-et-Senans, France (proceedings available from IRIA). Revised as Memo AIM 264, Computer Science Department, Stanford University.

[3] Gordon, M.J.C. (1975) Towards a Semantic Theory of Dynamic Binding. Memo AIM 265 , Computer Science Department, Stanford University,

[4] McCarthy, J. et.al. (1969) LISP 1.5 Programmer's Manual, MIT Press.

[5] Milne, R. (1974) The formal semantics of oomputer languages and their implementations. Oxford University Computing Laboratory, Programming Research Group, Technical Monograph PRG-13 (available on microfiche).

[6] Reynolds, J.C. (1972) Notes on a Lattice-Theoretic Approach to the Theory of Computation. Systems and information Science, Syracuse University.

[7] Reynolds, J.C. (1974) On the Relation between Direct and Continuation Semantios. Second colloquium on Automata, Languages, and Programming. Saarbrucken.

[8] Scott, D. (1974) Data Types as Lattices. To appear as Springer Lecture Not es.

[9] Scott, D. and Strachey, C. (1 972) Towards a Mathematical Semantics for Computer Languages. Proc. Symposium on Computers and Automata, Microwave Research. Institute Symposia Series, Vol.21, Polyt echnic Instit ut e of Brooklyn.