PB - 186

G. Frandsen: Logic Programming ...

TRYK: DAIMI/RECAU

ISSN 0105-8517

Logic Programming, Substitutions and Finite Computability

Gudmund Frandsen

DAIMI PB - 186
January 1985

Computer Science Department
AARHUS UNIVERSITY
Ny Munkegade - DK 8000 Aarhus C - DENMARK
Telephone: 06 - 12 83 55

Abstract

Apt and van Emden have studied the semantics of logic programming by means of fixed point methods. From a model theoretic point of view, their formalisation is very nice. Least and greatest fixed points correspond to least and greatest Herbrand-models respectively.

Viewed operationally, there is an ugly asymmetry. The least fixed point expresses finite computability, but the greatest fixed point denotes negation by trans-finite failure, i.e. the underlying operator is not ω-continuous for decreasing chains in general.

We use the notion of finite computability inherent in Scott domains to build a domainlike construct (the cd-domain) that offers ω-continuity for increasing and decreasing chains equally. On this basis negation by finite failure is expressed in terms of a fixed point.

The fixed point semantics of Apt and van Emden is very abstract concerning the concept of substitution, although it is fundamental for any implementation. Hence it becomes quite tedious to prove the correctness of a concrete resolution algorithm. The fixed point semantics of this paper offers an intermediate step in this respect. Any commitments to specific resolution strategies are avoided, and the semantics may be the basis of sequential and parallel implementations equally. Simultaneously the set of substitution dataobjects is structured by a Scott information theoretic partial order, namely the cd-domain.

1. Introduction

In 1965 J. A. Robinson proved the completeness of predicate logic equipped with one inference rule only [3]. It transpired that automatic theorem proving based on this resolution inference rule could be performed efficiently, when restricting axioms to Horn Clause format. This restricted class was interpreted as a programming language by Kowalski [2].

The semantics of logic programming has been studied model-theoretically and by means of fixed point methods initially by van Emden and Kowalski [4]. Later, Apt and van Emden extended this treatment to comprise negation [1].

They discuss three bases for augmenting logic programming with negation: (1) the closed world assumption, (2) the if and only if assumption and (3) the finite failure assumption. They are able to characterize negation based on (1) or (2) in terms of least and greatest fixed points respectively. We give a similar fixed point characterization of the third sort of negation by adopting an operational point of view.

The concept of substitution is fundamental to any implementation, and so it should be emphasized in an operational semantics. Actually, we define a set of finite substitution data-objects that represent the result of finite computations, and use Scott's information system framework [5] (properly modified) to build a domain-like construct, the cd-domain.

The cd-domain consists of infinite conjunctions of possibly infinite disjunctions of dataobjects. In comparison, the power-domain of indeterminacy [5] consists of infinite conjunctions

of finite disjunctions only. The addition of infinite disjunctions assures ω-continuity of decreasing chains as well as ω-continuity of increasing chains in the cd-domain. We can thus model the result of a non-halting computation that makes non-deterministic choices incessantly. The cd-domain is fully-abstract in the sense that two substitutions are identified exactly, when they do bear the same information contents with respect to finite computations.

The cd-domain is used to define a semantics for logic programming. This new semantics is proved to be equivalent to the standard semantics. It is shown that the new semantics induces a denotational semantics.

In what follows we start by recapitulating the fixed point semantics of Apt and van Emden [1], and proceed by reexpressing this semantics in terms of ground substitutions. Sets of ground substitutions are later used as a measure of the information contents of a substitution dataobject. The ground substitution semantics provides thus an intermediate step that facilitates the proof of correctness of the cd-domain semantics. Finally the denotational semantics is given.

2. Logic programming and the basic fixed point semantics

This section is based on the semantic analysis performed by van Emden, Kowalski [4] and Apt, van Emden [1]. The description of negation rely partly on a recent monograph by Lloyd [6].

The exposition starts by presenting an Example to establish intuition, and proceeds by giving a formal definition of the syntax and semantics of logic programming. This example formalism alternation will also structure the subsequent sections.

A logic program consists of a set of positive Horn Clauses, and it is a canonical representation of a certain limited number of first order predicate calculus formulae. Ex. 1 demonstrates a very simple program. It is composed of (open) rules and propositions.

Example 1

Program p:

airborne(X) ⇐ bird(X)
bird(X) ⇐ eagle(X) } rules
eagle(willy) proposition

Question q:

airborne(Y)?

Answer:

airborne(willy)

Groundterms T

T_0 consists of the elements of the free structure generated by the identifiers occurring in p, i.e. no element of T_0 refers to any variables.

Groundexpansion h: Question → 2^{T_0}

$h(q) = h(\text{airborne}(Y)) = \{\text{airborne}(Y) \mid Y \in T_0\}$

Inference operator I_p: $2^{T_0} \to 2^{T_0}$

$I_p(\emptyset) = \{\text{eagle(willy)}\}$

$I_p^2(\emptyset) = \{\text{eagle(willy), bird(willy)}\}$

$I_p^3(\emptyset) = \{\text{eagle(willy), bird(willy), airborne(willy)}\}$

$I_p^n(\emptyset) = I_p^3(\emptyset)$ for $n \geq 3$. $I_p^3(\emptyset)$ is the least fixed point of I_p: $\text{lfp}(I_p) = I_p^3(\emptyset)$.

The meaning of the question q in terms of the program p is:

$Q_1 \llbracket q \rrbracket p = \text{lfp}(I_p) \cap h(q) = \{\text{airborne(willy)}\}$.

Consider the last part of Example 1, where an inference operator I_p is defined. The basic fixed point semantics relies on this operator. $I_p(\emptyset)$ yields exactly the instantiations of all propositions in p, while $I_p(h)$ generally yields the conclusions of all rules that have their premises entirely in h. Propositions may be regarded as rules without premises. $lfp(I_p)$ is precisely the set of propositions that can be concluded from rules and propositions in p.

Consider also the question q = airborne(Y)? It is represented by the infinite ground expansion h(q) = {airborne(willy), airborne(eagle),..., airborne(eagle(willy)),...}. The meaning of q is the specific "subset" that may be concluded from p: $h(q) \cap lfp(I_p)$.

Example 1 is limited in what it reflects. It does not expose rules with multiple premises or rules with mutually intersecting conclusions. Hopefully it has established the intuitive basis for a formal semantics that includes all these aspects.

Let us begin by defining an abstract syntax precisely.

Abstract syntax

	I a countable set of identifiers
	V a countable set of variables
T = IT*|V	terms
L = T	literals (= questions)
N = L*	negative clauses
C = LN	positive clauses
P = C*	programs

Usually one distinguishes predicate and function identifiers and deals with arities. By removing these distinctions, we obtain technical simplicity. This is our motivation and the result should not be perceived as a proposal to extend logic programming beyond first order logic.

It should be noted that we have countably many variables and identifiers to our disposal, but we refer to only a finite number in a single program.

When specifying semantics, we shall see that the list-ordering (*-notation) of negative clauses (N = L*) is irrelevant. A negative clause may be regarded as a finite set of literals. The same is true for programs (P = C*).

Let us now define the ground expansion of terms:

Groundterms

$T_0 = IT_0^*$	The set of groundterms
$T_1 = 2^{T_0}$	The lattice of sets-of-groundterms ordered by setinclusion
$S_0 = V \to T_0$	The set of ground substitutions
$\cdot[\circ]: T \times S_0 \to T_0$	The substitution operation t[s] = "t, where any variable occurrence, v, is replaced by s(v)".
$h: T \to T_1$	The ground expansion of terms $h(t) = \{t[s] \mid s \in S_0\}$

Usually the set of groundterms is called the Herbrand base and is defined as the free structure generated by the identifiers in a specific program. For technical simplicity we choose to ignore the specific program and arities. We let T_0 denote "all variable free terms".

Let us turn to a definition of the inference operator

Inference operator: $I_p: T_1 \to T_1$

$$I_p(u) = \{hd(c)[s] \mid s \in S_0,\ c \in p,\ \forall l \in tl(c).l[s] \in u\},$$

where hd(c) and tl(c) denote "l" and "$l_1,\ldots,l_n$" respectively, if c is "$l \Leftarrow l_1,\ldots,l_n$".

It can be easily observed that T_1 is a complete lattice and I_p is a monotonic function. These facts imply the existence of least and greatest fixed points [1] that fulfil

---

Theorem 1

$$\emptyset \subseteq I_p^{\omega}(\emptyset) = lfp(I_p) \subseteq gfp(I_p) \subseteq I_p^{\omega}(T_0) \subseteq T_0,$$

and any of the above inclusions is proper for some program.

The above notation I_p^{ω} is defined as follows $I_p^{\omega}(\emptyset) = \bigcup_{n<\omega} I_p^n(\emptyset)$ and $I_p^{\omega}(T_0) = \bigcap_{n<\omega} I_p^n(T_0)$. We shall later use the same notation for arbitrary limit ordinals without explicit definition.

The inclusion hierarchy of Theorem 1 is mentioned and interpreted by Apt and van Emden [1]. Later Lloyd has compiled further results on the subject [6]. We present an example to illustrate the interpretation.

Example 2

Consider the program

$$p:\ r \Leftarrow q(Y)$$

$$q(f(X)) \Leftarrow q(X)$$

$$s \Leftarrow s.$$

We may compute

$$I_p(\emptyset) = \emptyset,$$

and thus

$$lfp(I_p) = I_p^n(\emptyset) = \emptyset \text{ for any ordinal } n.$$

$$I_p^n(T_0) = \{s,r\} \cup \{q(f^i(t)) \mid i \geq n,\ t \in T_0\}, \text{ for any ordinal } n < \omega.$$

$$I_p^\omega(T_0) = \bigcap_{n<\omega} I_p^n(T_0) = \{s,r\}$$

$$I_p^{\omega+1}(T_0) = \{s\} = I_p^{\omega+2}(T_0)$$

and thus

$$gfp(I_p) = I_p^n(T_0) = \{s\} \text{ for any ordinal } n \geq \omega + 1.$$

The result $lfp(I_p) = \emptyset$ means that nothing can <u>positively</u> be concluded from p. However, we may draw some <u>negative</u> conclusions. Precisely which ones is left to our discretion. By using the <u>Closed World assumption</u>, we take anything to be false unless it is explicitly said to be true. The negative conclusions are thus $\mathcal{C}\, lfp(I_p) = T_0$. A more cautious attitude is reflected in the

<u>If and only if assumption</u>: From p we form the completion p' that consists of: $r \Leftrightarrow \exists Y.q(Y)$, $\forall Z.\ (q(Z) \Leftrightarrow \exists X.Z = f(X) \wedge q(X))$, $s \Leftrightarrow s$ and some axioms about equality. The negative conclusions of p are defined to be the set of formulae, which is false in every Herbrand-model of p'. This yields $\mathcal{C} gfp(I_p) = T_0 \setminus \{s\}$, which may be computationally approximated by the

<u>Finite failure assumption</u>: In this case the following negative conclusions can be drawn: $\mathcal{C} I_p^\omega(T_0) = T_0 \setminus \{s,r\}$. These conclusions coincide with the set of formulae, which is false in every model (not necessarily Herbrand-) of the above p'.

We emphasize computability and define the meaning of a question with respect to a program in terms of what may be concluded in a finite number of inferences:

Meaning of a question:

Q_1: L → P → T_1 — The positive meaning of a question $Q_1 [\![l]\!]_p = h(l) \cap lfp(I_p)$.

N_1: L → P → T_1 — The negative meaning of a question $N_1 [\![l]\!]_p = h(l) \setminus I_p^{\omega}(T_0)$.

These definitions have been related to the model theoretic semantics by van Emden, Kowalski [4] (Q_1) and by Apt, van Emden [1] and Jaffar, Lassez and Lloyd [7] (N_1).

It is our goal to characterize each of Q_1 and N_1 in terms of fixed points by using the concept of finite compatability inherent in domain theory. Furthermore our domain (the cd-domain) will reflect the importance of substitutions for any actual computation. As an intermediate step, the meaning of a question is reexpressed in terms of ground substitutions.

3. Semantics and ground substitutions

In the earlier semantics, the meaning of a question was defined as a certain subset of the corresponding ground expansion (dependent on a specific program). This subset might be defined in terms of the associated ground substitutions. In support of intuition another example is presented here:

Example 3

program p:

$\text{family}(Y_1,Y_2) \Leftarrow \text{married}(Y_1,Y_2)$

$\text{family}(Z_1,Z_2) \Leftarrow \text{cousins}(Z_1,Z_2)$

$\text{married}(\text{pia},W)$

$\text{cousins}(\text{grethe},\text{hans})$

question q:

$\text{family}(\text{grethe},X)?$

answer: $[X = \text{hans}]$

Ground-substitution lattice:

The elements of this lattice are sets of ground-substitutions ordered by setinclusion. $[X = \text{hans}]$ is represented by the element $\{s \in S_0 \mid s(X) = \text{hans}\}$ (This representation will be elaborated in terms of domain theory during subsequent chapters). An answer involving no restrictions is represented by S_0, and $\emptyset$ denotes "no positive answer". Remember S_0 and $\emptyset$ are top and bottom elements respectively.

Meaning of a question:

We are going to define the meaning of a question as the least fixed point of a certain operator, but in the meantime the following recursive characterization will suffice:

$$\mathcal{Q}_2[\![l]\!]p = \bigcup_{c \in p} \left(\text{"unify}_2(l,\text{hd}(c))\text{"} \cap \bigcap_{l' \in tl(c)} \mathcal{Q}_2[\![l']\!]p \right) \Big|_{\text{var}(l)} .$$

We may compute the following:

$$\mathcal{Q}_2[\![\text{married}(Y_1,Y_2)]\!]p = \{s \in S_0 \mid s(Y_1) = \text{pia}, s(Y_2) = s(W)\} \Big|_{\{Y_1,Y_2\}}$$
$$= \{s \in S_0 \mid s(Y_1) = \text{pia}\}$$

$$\mathcal{Q}_2[\![\text{cousins}(Z_1,Z_2)]\!]p = \{s \in S_0 \mid s(Z_1) = \text{grethe},\ s(Z_2) = \text{hans}\}$$

$$\mathcal{Q}_2[\![\text{family,(grethe,X)}]\!]p = (\{s \in S_0 \mid s(Y_1) = \text{grethe},\ s(Y_2) = s(X)\} \cap \{s \in S_0 \mid s(Y_1) = \text{pia}\}) \Big|_{\{X\}}$$
$$\cup (\{s \in S_0 \mid s(Z_1) = \text{grethe},\ s(Z_2) = s(X)\} \cap \{s \in S_0 \mid s(Z_1) = \text{grethe}, s(Z_2) = \text{hans}\}) \Big|_{\{X\}}$$
$$= \emptyset \cup \{s \in S_0 \mid s(Z_1) = \text{grethe}, s(Z_2) = s(X) = \text{hans}\} \Big|_{\{X\}}$$
$$= \{s \in S_0 \mid s(X) = \text{hans}\}.$$

By the restriction "$\Big|_{\text{var}(l)}$" we ignore the bindings of any variable but X, on delivering the final result.

We proceed by extending the formalism of section 2, so as to encapsulate the concepts of Example 3.

Ground-substitutions

$S_0 = V \to T_0$	The set of ground-substitutions as defined previously.
$S_1 = 2^{S_0}$	The ground-substitution lattice.
$\cdot\big|_{\cdot} : S_1 \times 2^V \to S_1$	The restriction operator, which "erases" any information on variables that are not members of the specified set: $S\big|_V = \{s \in S_0 \mid \exists s' \in S.\ \forall v \in V.\ s(v) = s'(v)\}$
$\text{unify}_2 : T \times T \to S_1$	The most general unifier of two terms, expressed as a set of groundsubstitutions. $\text{unify}_2(t_1, t_2) = \{s \in S_0 \mid t_1[s] = t_2[s]\}$.
$A : L \to S_1$	The lattice of ground-substitution answers canonically ordered, i.e. $a_1 \sqsubseteq a_2 \overset{\text{def}}{\Leftrightarrow} \forall l.\ a_1(l) \subseteq a_2(l)$.
$k : T_1 \to A$	k is the canonical transformation of a set of groundterms into a ground-substitution answer: $k(t)(l) = \{s \mid l[s] \in t\}$.

It should be noted that the bottom and top elements of the two lattices are characterized by $\forall l\ .\ \bot_A(l) = \bot_{S_1} = \emptyset$ and $\forall l\ .\ \top_A(l) = \top_{S_1} = S_0$ respectively.

Let us now define the inference operator

Inference operator: $I_{2p}: A \to A$

$$I_{2p}(a)(l) = \bigcup_{c \in p} (\text{unify}_2(l, hd(c')) \cap \bigcap_{l' \in tl(c')} a(l'))\Big|_{var(l)},$$

where var(l) denotes the set of variables occurring in the literal l, and c' denotes c with variables renamed properly, so $var(l) \cap var(c') = \emptyset$. The restriction to var(l) means ignorance of specific choices of renaming, which therefore lose their importance.

It can be easily observed that A is a complete lattice and I_{2p} is a monotonic function. Hence I_{2p} has least and greatest fixed points. The following result makes it clear to us that Theorem 1 may be directly restated in terms of ground-substitutions:

Theorem 2:

i) k is an injection

ii) $k(I_p^n(\emptyset)) = I_{2p}^n(\bot_A)$, for any ordinal n

iii) $k(I_p^n(T_0)) = I_{2p}^n(\top_A)$, for any ordinal n

iv) $\bot_A \sqsubseteq I_{2p}^\omega(\bot_A) = lfp(I_{2p}) \sqsubseteq gfp(I_{2p}) \sqsubseteq I_{2p}^\omega(\top_A) \sqsubseteq \top_A$, and any of the above "inclusions" is proper for some program.

Proof:

i) Assume $t_1, t_2 \in T_1$ and $t_1 \neq t_2$. Furthermore $t \in t_1 \setminus t_2$ wlog. Then $k(t_1)(t) = S_0$ and $k(t_2)(t) = \emptyset$.

ii) $k(I_p^n(\emptyset)) = I_{2p}^n(\perp_A)$ is proved by induction.

Basis: $k(\emptyset) = \perp_A$

Successor step: $k(I_p^{n+1}(\emptyset))(l)$

$= \{s \mid l[s] \in I_p^{n+1}(\emptyset)\}$

$= \{s \mid \exists s' \in S_0 \ \exists c \in p.\ l[s] = hd(c)[s'], \ \forall l' \in tl(c).\ l'[s'] \in I_p^n(\emptyset)\}$

$= \bigcup_{c \in p} \{s \mid l[s] = hd(c')[s], \ \forall l' \in tl(c').\ l'[s] \in I_p^n(\emptyset)\}\Big|_{var(l)}$

(c' is c renamed)

$= \bigcup_{c \in p} \left(unify_2(l, hd(c')) \cap \bigcap_{l' \in tl(c')} k(I_p^n(\emptyset))(l')\right)\Big|_{var(l)}$

$= I_{2p}(k(I_p^n(\emptyset)))(l)$

$\overset{ind}{=} I_{2p}^{n+1}(\perp_A)(l)$

Limit step: $k(I_p^n(\emptyset))(l)$

$= \{s \mid l[s] \in I_p^n(\emptyset)\}$

$= \{s \mid l[s] \in \bigcup_{\alpha<n} I_p^\alpha(\emptyset)\}$

$= \bigcup_{\alpha<n} \{s \mid l[s] \in I_p^\alpha(\emptyset)\}$

$= \bigcup_{\alpha<n} k(I_p^\alpha(\emptyset))(l)$

$\overset{ind}{=} \bigcup_{\alpha<n} I_{2p}^\alpha(\perp_A)(l)$

$= I_{2p}^n(\perp_A)(l)$.

iii) May be proved similarly.

iv) Apply i), ii), iii) and Theorem 1.

Q.E.D.

We are now ready to specify the meaning of a question:

Meaning of a question

$\mathcal{Q}_2$: $L \to P \to S_1$ — The positive meaning of a question.
$\mathcal{Q}_2[\![l]\!]p = lfp(I_{2p})(l)$

N_2: $L \to P \to S_1$ — The negative meaning of a question.
$N_2[\![l]\!]p = \int I^{\omega}_{2p}(\tau_A)(l)$

The following corollary to Theorem 2 creates the anticipated connection between the two semantics:

Corollary 1

i) $k(\mathcal{Q}_1[\![l]\!]p)(l) = \mathcal{Q}_2[\![l]\!]p$

ii) $k(N_1[\![l]\!]p)(l) = N_2[\![l]\!]p$

We are now going to leave the ground-substitution lattice and turn our attention to the cd-domain of substitutions, which may be regarded as a more pragmatic representation of the somewhat abstract ground-substitutions.

4. The cd-domain of substitutions

Our aim is to express the semantics of logic programming in terms of ordinary substitutions rather than large sets of ground-substitutions in order to stress aspects of computability. For this purpose the cd-domain of substitutions (a domainlike construct) is built by using Scott's information system framework. The cd-domain will be fully-abstract in the sense that two substitutions will be represented by the same domain element if and only if they bear the same information contents and their equivalence can be confirmed within a finite computation.

To begin with an example will show an intended "operational" interpretation, which may later guide the domain construction.

Example 4

An algorithm for computing the answer to a question, need only work with one sort of substitution dataobject, from which finite conjunctions are constructed.

program p:

$goal(X) \Leftarrow cond_1(X,Y,Z),\ cond_2(X,Y,Z).$
$cond_1(Z,Y,Z).$
$cond_1(f(Y),Y,Z).$
$cond_2(f(g(Z)),Y,Z).$

question:

q_1: $goal(X)$?
q_2: $cond_1(X,Y,Z)$?
q_3: $cond_2(X,Y,Z)$?

The answer to q_2 may be represented by

$$c_2 = \{\{[X \to Z],[X \to f(Y)]\}\},$$

that has an intended interpretation: "apply either the substitution $[X \to Z]$ or the substitution $[X \to f(Y)]$ to the question $cond_1(X,Y,Z)$, so as to obtain an answer." Similarly the reply

to q_3 is represented by

$$c_3 = \{\{[X \rightarrow f(g(Z))]\}\}.$$

In order to obtain an answer to q_1, we must form the conjunction of c_2 and c_3 and forget about variable names local to q_2 or q_3. This result in the cds (conjunction of disjunctions of substitutions):

$$c_1 = \left\{ \begin{array}{l} \{[X \rightarrow Z], [X \rightarrow f(Y)]\} \\ \{[X \rightarrow f(g(Z))]\} \end{array} \right\} \Big|_{\{X\}}$$

By means of unification c_1 may be reduced to

$$c_1^{(1)} = \{\{[X \rightarrow f(Y), Y \rightarrow g(Z)]\}\}\Big|_{\{X\}}$$

The first alternative in c_2 is inconsistent with c_3, i.e. we do not allow cyclic substitutions of the form $[Z \rightarrow f(g(Z))]$. This leaves only one disjunkt for $c_1^{(1)}$. This disjunct can be expressed in more than one way, and

$$c_1^{(2)} = \{\{[X \rightarrow f(g(Z)),\ Y \rightarrow g(Z)]]\}\}\Big|_{\{X\}}$$

might do equally well.

Finally, we apply the restriction $\Big|_{\{X\}}$ inside $c_1^{(1)}$:

$$c_1^{(3)} = \{\{[X \rightarrow f(\underline{Y}),\ \underline{Y} \rightarrow g(\underline{Z})]\}\}.$$

Underscore denotes an anonymous variable. We have restricted the set of named (non-anonymous) variables to those referred to by q_1, i.e. the set $\{X\}$. The choice of names for anonymous variables is arbitrary and

$$c_1^{(4)} = \{\{[X \rightarrow f(\underline{Z}),\ \underline{Z} \rightarrow g(\underline{Y})]\}\}$$

should be perceived as equivalent to c_1. We may even remove an anonymous variable without changing the information content:

$$c_1^{(5)} = \{\{[X \to f(g(\underline{Z}))]\}\}.$$

The distinction between anonymous and named variables is important. The cds's

$$c_4 = \{\{[X \to f(Y),\ Y \to g(Z)]\}\} \quad \text{and}$$

$$c_4' = \{\{[X \to f(Z),\ Z \to g(Y)]\}\}$$

with no anonymous variables are mutually inconsistent, whereas $c_4\big|_{\{X\}}$ (reduces to $c_1^{(3)}$) and $c_4'\big|_{\{X\}}$ (reduces to $c_1^{(4)}$) are equivalent.

The existence of different equivalent representations calls for a canonical representation, which is obtained by constructing the cd-domain of substitutions.

The domain element s_1, represented by c_1, is itself a cds, namely the deductive closure of c_1, which is the conjunction of all these disjunctions, which make no more commitments than those made by c_1 (or $c_1^{(1)},\ldots,c_1^{(5)}$). We adapt the notation $s_1 = \overline{c_1}$.

$\overline{c_1}$ = {d| d is a set (disjunction) of substitution dataobjects, such that d contains no more information than c_1}.

$c_1,\ldots c_1^{(5)}$ are all finite sets of finite sets, which is all that may possibly result from a finite computation. However, s_1 is an infinite set of possibly infinite sets. This may seem undesirable. Yet, we cannot confine ourselves to finite sets if we wish to represent the result of a non-halting computation.

Consider the programs

P_2: r⇐g(Y).
g(f(x))⇐g(x).

P_3: g(a).
g(f(x))⇐g(x).

and the question

q: g(Y)?

It seems obvious that the P_2-answer to q should be the infinite chain $s_2 = \bigsqcup_i s_{2,i}$, where

$$s_{2,i} = \{\{[Y \to f(X_1), X_1 \to f(X_2), \ldots, X_{i-1} \to \overline{f(X_i)}]\}\}|_{\{Y\}} = \overline{\{\{[Y \to f^i(\underline{X})]\}\}}$$

If we were to represent s_2 by a finite cds, we might choose $\{\emptyset\}$, where $\emptyset$ is the empty disjunction that poses no alternatives, i.e. it denotes inconsistency. This is unsatisfactory because all of the $s_{2,i}$, denoting finite subresults, are consistent. In comparison, the groundsubstitution representation of the previous section has such "discontinuity" that

$$gfp(I_{2p_2}) \neq I^{\omega}_{2p_2}(\top_a)$$

$$(gfp(I_{2p2}) = I^{\omega+1}_{2p2}(\top_a) = \emptyset \neq \{r\} = I^{\omega}_{2p_2}(\top_a)).$$

is the result. We want

$$\{\{[Y \to f^1(X)]\}, \{[Y \to f^2(X)]\}, \ldots, \{[Y \to f^n(X)]\}, \ldots\}$$

and not $\{\emptyset\}$ to be a proper representative of s_2. By introducing infinite sets of conjunctions, we obtain more structure than the groundsubstitutionlattice could offer.

With respect to p_3, similar considerations can be made. It seems obvious that the answer to q with respect to p_3 should be $s_3 = \bigsqcap_i s_{3,i}$, where

$$s_{3,i} = \{\{[Y \to a], [Y \to f(X_{i1}), X_{i1} \to a], .., [Y \to f(X_{i1}), .., x_{ii} \to a]\}\}^{\{Y\}}$$
$$= \{\{[Y \to a], [Y \to f(a)], .., [Y \to f^i(a)]\}\}.$$

Each computation step adds another alternative, and the result becomes less specific with the passage of time: We lose information. The limit result s_3 might be represented by the finite cds $\{\{[]\}\}$, i.e. the empty substitution, which contains no information at all. Such a representation is unsatisfactory, because every finite subresult ($s_{3,i}$) contains some information. We want

$$s_3 = \{\{[Y \to a], [Y \to f(a)], ..., [Y \to f^n(a)], ...\}\}$$

The ground substitution lattice has sufficient structure to model the "continuous" representation of s_3. This is reflected in $lfp(I_{2p}) = I_{2p}^{\omega}(\bot_A)$ for all programs p.

Our aim is to formalise the above observations in a domain-construction: Scott's power domain of indeterminacy [5] is almost what we want. This power domain models infinite sets (conjunctions) of disjunctions. Unfortunately, only finite, nonempty disjunctions are allowed. We have just seen that infinite disjunctions are necessary to model the nonhalting computation connected to program p_3 (ex.4). Such chains of <u>decreasing</u> information contents seem important in the description of nondeterminism. Yet only chains of <u>increasing</u> information contents generally have nice continuity-properties in Scott-domains. Apart from this the empty disjunction is a natural representative of "no alternatives" or inconsistency as the result of unification-failure.

We choose to construct a new sort of domain, the cd-domain, which assures continuity of decreasing and increasing chains equally, and it contains an inconsistent topelement as well as the uninformative bottomelement.

The cd-domain will be based on Scott's information system framework [5], which is briefly given beneath.

An information system is a structure $(\mathcal{D},\Delta,\mathrm{Con},\vdash)$, where $\mathcal{D}$ is the set of dataobjects (or propositions) and $\Delta \in \mathcal{D}$ is the uninformative (or least informative) dataobject. Dataobjects provide finite information about possible elements of the domain to be constructed. Conjunctions of dataobjects are represented by subsets of $\mathcal{D}$. Con are those finite subsets of $\mathcal{D}$ which are consistent. $\vdash$ is the entailment relation between members of Con and members of $\mathcal{D}$.

Any information system must fulfil six simple axioms (cf. [5]).

The domain induced by the above informationsystem consists of the finitely consistent, deductively closed subsets of $\mathcal{D}$, i.e. the domain fulfils:

i) Any element x is finitely consistent:
$\forall u \subseteq x.$ u is finite $\Rightarrow u \in \mathrm{Con}$.

ii) Any element x is closed under entailment:
$\forall u \subseteq x. \quad \forall X \in \mathcal{D}. \quad (u \text{ is finite}, u \vdash X) \Rightarrow X \in x.$

An element may be perceived as the conjunction of its constituent dataobjects, and the domain is ordered by set inclusion.

In the following we shall use the information system framework, properly modified. Firstly, we do not require domain elements to be finitely consistent, but allow an inconsistent topelement (to represent "no alternatives" or inconsistency as the result of unification-failure). In relation to this we define entailment from inconsistent sets as well.

Secondly, domain elements are sets of sets of dataobjects rather than merely sets of dataobjects, and domainelements may be interpreted as conjunctions of disjunctions of dataobjects instead of conjunctions alone. For this reason, consistency and entailment will be defined in terms of such sets of sets of dataobjects.

Due to these changes, we no longer take Scott's original six axioms into consideration.

We are now ready to define the cd-information system of substitutions $(\mathcal{D}_s, \Delta_s, Con_s, \vdash_s)$:

Substitution dataobjects:

$S = V \to T$	The set of substitutions
$h: S \to S_1$	The groundexpansion of substitutions: $h(s) = \{s' \in S_0 \mid \forall v.s(v)[s'] = s'(v)\}$
$\mathcal{D}_s$	The set of substitution-dataobjects $\mathcal{D}_s = \{(v,s) \in 2^V \times S \mid$ i) v and def(s) are finite ii) $h(s) \neq \emptyset$ iii) $v \subseteq def(s) \cup ref(s)\}$ where $def(s) = \{v \mid s(v) \neq v\}$ and $ref(s) = \{v \mid \exists v' \in def(s).\ s(v')$ refer to $v\}$.
$\Delta_s = (\emptyset, s)$	The uninformative dataobject, where $def(s) = \emptyset$.
$h: \mathcal{D}_s \to S_1$	The groundexpansion generalised to $\mathcal{D}_s$: $h((v,s)) = h(s)\big\vert_v$
$\cdot\big\vert_\cdot : \mathcal{D}_s \times 2^V \to \mathcal{D}_s$	The restriction operation on dataobjects $(v,s)\big\vert_{V'} = (v \cap V', s)$

A dataobject $(v,s) \in \mathcal{D}_S$ consists of a substitution s with an appointed set v of named (non anonymous) variables. The requirement of def(s) being finite reflects the fact that in a finite computation only substitutions with a finite number of non-trivial values need be considered. The second condition $h(s) \neq \emptyset$ demands that s be a consistent substitution, i.e. s does not contain cyclic variable references (directly or indirectly). The last condition ($v \subseteq def(s) \cup ref(s)$) states that some information must be supplied on any named variable.

The groundexpansion h is defined so as to erase any information on anonymous variables. We may perceive h as denoting the information contents of a dataobject: Δ_S poses no restrictions and $h(\Delta_S) = S_0$. It should be noted that any dataobject with no named variables is equally uninformative: $h((\emptyset,s)) = S_0$ for any s. At the opposite end of the information scale reside the inconsistent dataobjects, which have empty expansion. They are, however, excluded from $\mathcal{D}_S$ by restriction ii) in the definition.

The restriction operator delimits the set of named variables and fulfils the commutative law: $h(s|_v) = h(s)|_v$. Note that $\Delta_S|_v = \Delta_S$ for any set of variables v.

The notation for dataobjects can be illustrated by an example:

Example 5

We use the syntax of Example 4, when writing down dataobjects:

$$[X \to f(\underline{Y}), \underline{Y} \to g(Z)]$$

denotes the element $(v,s) \in \mathcal{D}_S$, for which $v = \{X,Z\}$ and

$$s(v') = \begin{cases} f(Y) & , \quad v' = X \\ g(Z) & , \quad v' = Y \\ v' & , \quad \text{otherwise} \end{cases}$$

In particular [ ] denotes Δ_S.

For the above dataobject we compute the following expansion:

$$h([X \to f(\underline{Y}), \underline{Y} \to g(Z)]) = h([X \to f(Y), Y \to g(Z)])|_{\{X,Z\}} =$$
$$\{s \mid s(X) = f(s(Y)),\ s(Y) = g(s(Z))\}|_{\{X,Z\}} = \{s \mid s(X) = f(g(s(Z)))\}.$$

Let us now define consistency and entailment in terms of conjunction of disjunctions of substitution dataobjects:

Conjunctions of disjunctions of substitution dataobjects (cds's)

$\mathcal{P}_S = 2^{2^{\mathcal{D}_S}}$ The set of all cds's.

$h: \mathcal{P}_S \to S_1$ The groundexpansion generalised to $\mathcal{P}_S$: $h(c) = \bigcap_{d \in c} \bigcup_{s \in d} h(s)$.

$\mathcal{F}_S = \{c \in \mathcal{P}_S \mid c \text{ is finite}, \forall d \in c.\ d \text{ is finite}\}$ The set of finite cds's.

$Con_S = \{c \in \mathcal{F}_S \mid h(c) \neq \emptyset\}$ The set of finite cds's which are consistent, i.e. Scott's Con, modified to conjunctions of disjunctions.

$\vdash_S = \{(c,s) \in \mathcal{F}_S \times \mathcal{D}_S \mid h(c) \subseteq h(s)\}$. The entailment relation, i.e. Scotts $\vdash$, modified to conjunctions of disjunctions.

Conjunctions of disjunctions of substitution dataobjects (cds's) are represented by sets of sets of dataobjects. The groundexpansion of a cds reflects the fact that the (global) information contents of a (possibly infinite) cds is a conjunction of disjunctions of single dataobject-information contents. When $h(c_1) \subseteq h(c_2)$, the information content of c_1 is more specific than the information content of c_2.

The set of finite cds's, $\mathcal{F}_S$, is all that we need to represent the result of a finite computation. Consistency and entailment are defined exclusively in terms of such finite cds's. For a cds to

be consistent, there must exist a groundsubstitution included in at least one alternative (disjunct) of every conjunct of c. Note that an inconsistent cds entails everything.

Let us now illustrate the above definitions with an example:

Example 6

Consider the dataobjects

$$s_1 = [X \to f(Y)],\ s_2 = [X \to f(\underline{Y})],\ s_3 = [X \to g(\underline{Y})]$$

and the cds's

$$c_1 = \{\{s_1\}\},\ c_2 = \{\{s_2\}\},\ c_{23} = \{\{s_2, s_3\}\}.$$

c_1, c_2, c_{23} are sequenced according to decreasing information contents: c_1 is a singleton with a single disjunct, which refers to no anonymous variables. In c_2 the variable Y has been discarded and replaced by an anonymous one ($s_2 = s_1\big|_{\{X\}}$). Thus all information about Y is lost. In c_{23} another disjunct has been added, and c_{23} becomes less specific than c_2. These observations are reflected formally:

$$h(c_1) = h(s_1) = \{s \mid s(X) = f(s(Y))\}$$

$$h(c_2) = h(s_2) = h(s_1\big|_{\{X\}}) = h(s_1)\big|_{\{X\}} = \{s \mid s(X) = f(t), \text{ for some } t\}.$$

$$h(c_{23}) = h(s_2) \cup h(s_3) = \{s \mid \exists t_1.s(X) = f(t_1) \vee \exists t_2.s(X) = g(t_2)\}.$$

This means $h(c_1) \subsetneq h(c_2) \subsetneq h(c_{23})$ which proves decreasing information contents of the sequence c_1, c_2, c_{23}.

The dataobjects s_2 and s_3 are incompatible and the cds $c_{2,3} = h(s_2) \cap h(s_3) = \emptyset$. Furthermore, $c_{2,3} \vdash d$ for any dataobject d.

We are now about to introduce the cd-domain

The cd-domain of substitutions

$\vdash_s = \{(c,d) \in \mathcal{FF}_s \times 2^{\mathcal{D}_s} \mid h(c) \subseteq \bigcup_{s\in d} h(s)\}$ Entailment is generalised to disjunctions of dataobjects.

$\bar{\cdot}: \mathcal{PP}_s \to \mathcal{PP}_s$ The deductive closure of a cds:

$\bar{u} = \{d \mid \exists\{d_1,\ldots,d_n\} \subseteq u \, \forall f_1,..,f_n.(\forall i.\, f_i \subseteq d_i, f_i \text{ is finite}) \Rightarrow \{f_1,..,f_n\} \vdash_s d\}$

$= \{d \mid \exists\{d_1,\ldots,d_n\} \subseteq u.\ h(\{d_1,\ldots,d_n\}) \subseteq \bigcup_{s\in d} h(s)\}$

$|S| = \{s \in \mathcal{PP}_s \mid s = \bar{s}\}$ The cd-domain of substitutions, ordered by setinclusion.

$\bot_s = \bar{\emptyset} = \overline{\{\{\Delta_s\}\}} = \{d \in 2^{\mathcal{D}_s} \mid \exists s \in S.\ (\emptyset,s) \in d\}$ The bottom element of $|S|$.

$\top_s = \overline{\{\emptyset\}} = 2^{\mathcal{D}_s}$ The topelement of $|S|$.

$\cdot|_\cdot: |S| \times 2^V \to |S|$ The restriction operator generalised to $|S|$:

$s|_V = \overline{\{d|_V \mid d \in s\}}$, where the restriction of a disjunction is $\{s_1,\ldots,s_n,..\}|_V = \{s_1|_V,\ldots,s_n|_V,\ldots\}$.

$\text{unify}_3: T \times T \to |S|$ The most general unifier of two terms expressed as an element of the cd-domain.

$$\text{unify}_3(t_1,t_2) = \begin{cases} \overline{\{\{[v\to t]\}\}} & , (t_1 = v \in V, t_2 = t \text{ or } t_2 = v \in V, t_1 = t) \text{ and } h([v\to t]) \neq \emptyset. \\ \bigsqcup_{j=1,..,n} \text{unify}_3(t_{1j},t_{2j}) & , t_1 = it_{11}\ldots t_{1n}, t_2 = it_{21},\ldots,t_{2n} \\ \top_s & , \text{otherwise.} \end{cases}$$

The closure of a cds u consists of those disjunctions, which are entailed by all finite subdisjunctions of some finite subconjunction of u. Note that the abbreviation after the last equals-sign of the closure definition is permissable. However, the final step "$\bar{u} = \{d \mid h(u) \subseteq h(d)\}$" is incorrect! To understand this, consider $u = \bigcup_i \{d_i\}$, where $d_i = \{[X \to f^i(Z)]\}$. Here $h(\{d_{i_1},\ldots,d_{i_n}\}) \neq \emptyset$ for all $\{d_{i_1},\ldots,d_{i_n}\} \subseteq u$, but $h(u) = \emptyset$. Observe that the closure-operation preserves the groundexpansion.

The cd-domain of substitutions is the set of deductively closed cds's ordered by information contents, i.e. setinclusion. The finite elements of $|S|$ are those which may be represented by a finite cds: $\{\bar{c} \mid c \in \mathcal{F}_S\}$.

$\perp_S$ is the only uninformative element of $|S|$. Every conjunct of $\perp_S$ offers at least one alternative (disjunct), which is uninformative, i.e. has no named variables. $h(\perp_S) = \top_{S_1} = S_0$ and $\perp_S|_v = \perp_S$ for any set of variables v.

$\top_S$ is the only finitely inconsistent element of $|S|$. To understand this, it should be remembered that a finite inconsistent cds entails everything. $h(\top_S) = \perp_{S_1} = \emptyset$ and $\top_S|_v = \top_S$ for any set of variables v.

Let us now illustrate some properties of the above definitions with an example.

Example 7 (Full abstractness)

The opening example indicated that the cd-domain should have more structure than the groundsubstitution-lattice. We are illustrating here that this has been achieved: In the cd-domain the notion of finiteness (computability) is incorporated in the structure. Compare the inconsistent element $\top_S$ with the element $s = \overline{\{d_1, d_2, \ldots, \ldots\}}$, where $d_i = \{[X \to f^i(Z)]\}$. Both elements are expanded into the empty set: $h(\top_S) = h(s) = \emptyset$. This result means that taking a global, infinite perspective s is inconsistent, but seen from a computational point of view, we may look at finite portions only and $s_n = \overline{\{d_1, .., d_n\}}$ certainly is consistent: $h(s_n) \neq \emptyset$.

If we remove the finiteness restriction in the definitions of entailment and closure, we obtain a (non-domain) structure isomorphic (by h) to the groundsubstitution lattice.

The property of finiteness has further implications. In general restriction and expansion do not commute: For the above element s, $h(s)\big|_{\emptyset} = \emptyset\big|_{\emptyset} = \emptyset$, but $h(s\big|_{\emptyset}) = h((\bigsqcup_i s_i)\big|_{\emptyset}) = h(\bigsqcup_i (s_i\big|_{\emptyset})) = \bigcap_i h(s_i\big|_{\emptyset})$ $= \bigcap_i (h(s_i)\big|_{\emptyset}) = \bigcap_i S_0 = S_0$. We have used the results of the next theorem to perform the computation.

For finite elements, however, there is a law of commutation, which is stated in the theorem to follow, together with some other results.

For future use, we state some essential properties of the cd-domain |S| in a theorem:

Theorem 3

i) Least upper bounds and greatest lower bounds in |S| may be expressed in terms of setoperators. For $s_1, s_2 \in |S|$: $s_1 \sqcup s_2 = \overline{s_1 \cup s_2}$, $s_1 \sqcap s_2 = s_1 \cap s_2$. If $s_1 \sqsubseteq s_2 \sqsubseteq \ldots$ is an increasing chain in |S| then $\bigsqcup_i s_i = \bigcup_i s_i$. If $s_1 \sqsupseteq s_2 \sqsupseteq \ldots$ is a decreasing chain in |S| then $\bigsqcap_i s_i = \bigcap_i s_i$.

ii) A finite element $s \in |S|$ may be represented by a single disjunction, i.e. we can find $s_1, \ldots, s_n \in \mathcal{D}_s$ such that $s = \overline{\{\{s_1, \ldots, s_n\}\}}$.

iii) When $s_1, s_2 \in |S|$ are finite, so are $s_1 \sqcup s_2$ and $s_1 \sqcap s_2$. Furthermore $h(s_1 \sqcup s_2) = h(s_1) \cap h(s_2)$ and $h(s_1 \sqcap s_2) = h(s_1) \cup h(s_2)$.

iv) When t_1, t_2 are any terms, then $\text{unify}_3(t_1, t_2)$ is finite and $h(\text{unify}_3(t_1, t_2)) = \text{unify}_2(t_1, t_2)$.

v) When $s \in |S|$ is finite and V is any set of variables, then $s\big|_V$ is finite and $h(s\big|_V) = h(s)\big|_V$.

vi) If $\{s_n\} \subseteq |S|$ is an increasing chain of finite elements, $s \in |S|$ is finite and V is any set of variables, then:

$\{s_n|_V\}$ is an increasing chain

$(\bigsqcup_n s_n)|_V = \bigsqcup_n (s_n|_V)$

$s \sqcup (\bigsqcup_n s_n) = \bigsqcup_n (s \sqcup s_n)$

$s \sqcap (\bigsqcup_n s_n) = \bigsqcup_n (s \sqcap s_n)$

$h(\bigsqcup_n s_n) = \bigcap_n h(s_n)$

vii) If $\{S_n\} \subseteq |S|$ is a decreasing chain of finite elements, $s \in |S|$ is a finite element and V is any set of variables, then:

$\{s_n|_V\}$ is a decreasing chain

$(\bigsqcap_n s_n)|_V = \bigsqcap_n (s_n|_V)$

$s \sqcup (\bigsqcap_n s_n) = \bigsqcap_n (s \sqcup s_n)$

$s \sqcap (\bigsqcap_n s_n) = \bigsqcap_n (s \sqcap s_n)$

$h(\bigsqcap_n s_n) = \bigcup_n h(s_n)$.

Proof

i) Observe that $s_1 \cap s_2 = \overline{s_1 \cap s_2}$, $\bigcup_n s_n = \overline{\bigcup_n s_n}$ and $\bigcap_n s_n = \overline{\bigcap_n s_n}$.

ii) The argument is tedious, and has been transferred to form an appendix to this paper.

iii) Assume $s_i = \{\overline{d_i}\}$ according to ii). If so $s_1 \sqcup s_2 = \{\overline{d_1, d_2}\}$ and $s_1 \sqcap s_2 = \{\overline{d_1 \cup d_2}\}$ are both finite. Furthermore $h(s_1 \sqcup s_2) = h(d_1) \cap h(d_2) = h(s_1) \cap h(s_2)$ and $h(s_1 \sqcap s_2) = h(d_1 \cup d_2) = h(d_1) \cup h(d_2) = h(s_1) \cup h(s_2)$.

iv) The finiteness of $unify_3(t_1,t_2)$ follow from iii). The equality $h(unify_3(t_1,t_2)) = unify_2(t_1,t_2)$ is proved by induction on the structure of t_1,t_2.

v) Assume $s = \{\overline{d}\}$ according to ii). We may then compute $s\big|_V = \overline{\{d'\big|_V | h(d) \subseteq h(d')\}} = \{\overline{d'\big|_V} | h(d)\big|_V \subseteq h(d')\big|_V\} = \{\overline{d'\big|_V} | h(d\big|_V) \subseteq h(d'\big|_V)\} = \{\overline{d\big|_V}\}$. Hence $s\big|_V$ is finite and $h(s\big|_V) = h(d\big|_V) = h(d)\big|_V = h(s)\big|_V$.

vi) Assume $s_n = \{\overline{d_n}\}$, $s = \{\overline{d}\}$ according to ii). It is easy to verify that $\{s_n\big|_V\}$ is an increasing chain. We compute

$$(\bigsqcup_n s_n)\big|_V = \{\overline{d_1,\ldots}\}\big|_V = \{\overline{d\big|_V | \exists i_1,\ldots,i_n . h(\{d_{i_1},\ldots,d_{i_n}\}) \subseteq h(d)}\} =$$

$$\{\overline{d\big|_V | \exists n . h(\{d_n\}) \subseteq h(d)}\} = \bigcup_n (\overline{\{d_n\}}\big|_V) = \bigsqcup_n (s_n\big|_V).$$

$$s \sqcup (\bigsqcup_n s_n) = \{\overline{d,d_1,d_2,\ldots}\} = \{\overline{d,d_1}\} \sqcup \{\overline{d,d_2}\} \sqcup \ldots = \bigsqcup_n (s \sqcup s_n).$$

$$s \sqcap (\bigsqcup_n s_n) = \{\overline{d}\} \sqcap \{\overline{d_1,d_2,\ldots}\} = \{\overline{d \cup d_1, d \cup d_2,\ldots}\} = \bigsqcup_n (s \sqcap s_n).$$

$$h(\bigsqcup_n (s_n)) = h(\{\overline{d_1,d_2,\ldots}\}) = \bigcap_n h(d_n) = \bigcap_n h(s_n).$$

vii) Assume $s_n = \{\overline{d_n}\}$, $s \in \{\overline{d}\}$ according to ii). It can be easily verified that $\{s_n\big|_V\}$ is a decreasing chain.

$$(\bigsqcap_n s_n)\big|_V = \{\overline{\bigcup_n d_n}\}\big|_V = \{\overline{(\bigcup_n d_n)\big|_V}\} = \{\overline{\bigcup_n (d_n\big|_V)}\} = \bigsqcap_n (s_n\big|_V).$$

$$s \sqcup (\bigsqcap_n s_n) = \{\overline{d}\} \sqcup \{d_1 \cup d_2 \cup \ldots\} = \{\overline{d,d_1}\} \sqcap \{\overline{d,d_2}\} \sqcap \ldots = \bigsqcap_n (s \sqcup s_n)$$

$$s \sqcap (\bigsqcap_n s_n) = \{\overline{d \cup d_1 \cup d_2 \cup \ldots}\} = \{\overline{d \cup d_1}\} \sqcap \{\overline{d \cup d_2}\} \sqcap \ldots = \bigsqcap_n (s \sqcap s_n)$$

$$h(\bigsqcap_n s_n) = h(\bigcup_n d_n) = \bigcup_n h(d_n) = \bigcup_n h(s_n).$$

Q.E.D.

Theorem 3 ii) and the abbreviation following the definition of closure for cds's both suggest that the present exposition exhibits redundant structure. Future work might seek to find a simplification or explain the failure to do so.

In the meantime we put the present cd-domain to use in another semantics for logicprogramming.

5. Semantics and the cd-domain of substitutions

We are going to define a semantics in terms of the cd-domain of substitutions. In order to establish intuition we will translate ex. 3, treating of groundsubstitutions, into domain terminology.

Example 8

We use program, question and answer, unchanged from ex. 3.

Cd-Domain of substitutions

The answer [X=hans] is represented by the domain element $\{\{\overline{[X=hans]}\}\}$.

Meaning of a question

In a sense the domain is a reverse of the ground substitution-lattice (Th. 3 iii). It appears natural, that we define the meaning of a question as a greatest fixed point. Yet, we take a recursive characterization only in this example:

$$Q_3[\![l]\!]p = \bigsqcap_{c\in p} (unify_3(l, hd(c')) \sqcup \bigsqcup_{l'\in tl(c)} Q_3[\![l']\!]p) \Big|_{var(l)}.$$

We may compute the following:

$$Q_3[\![\text{married}\ (Y_1, Y_2)]\!]p = \{\{[Y_1 \to \text{pia},\ Y_2 \to W]\}\}\big|_{\{Y_1, Y_2\}} = \overline{\{\{[Y_1 \to \text{pia}]\}\}}.$$

$$Q_3[\![\text{cousins}\ (Z_1, Z_2)]\!]p = \{\{\overline{[Z_1 \to \text{grethe}, Z_2 \to \text{hans}]}\}\}\big|_{\{Z_1, Z_2\}} = \overline{\{\{[Z_1 \to \text{grethe}, Z_2 \to \text{hans}]\}\}}$$

$$Q_3[\![\text{family}\ (\text{grethe}, X)]\!]p = (\overline{\{\{Y_1 \to \text{grethe}, Y_2 \to X]\}\}} \sqcup Q_3[\![\text{married}(Y_1, Y_2)]\!]p)\big|_{\{X\}}$$

$$\sqcap(\overline{\{\{[Z_1 \to \text{grethe},\ Z_2 \to X]\}\}} \sqcup Q_3[\![\text{cousins}(Z_1, Z_2)]\!]p)\big|_{\{X\}} =$$

$$\top_S \sqcap(\{\{[Z_1 \to \text{grethe}, Z_2 \to X,\ X \to \text{hans}]\}\}\big|_{\{X\}} = \overline{\{\{[X \to \text{hans}]\}\}}.$$

We wish to define a formal counterpart of ex. 8 and to accomplish this aim we translate the previous groundsubstitution-results into our new domain-framework.

Substitution answers

$B = L \to |S|$ The lattice of substitution domain-answers, canonically ordered, i.e. $b_1 \sqsubseteq b_2 \Leftrightarrow \forall l.\ b_1(l) \sqsubseteq b_2(l)$.

$h: B \to A$ The expansion operator extended to comprise B: $h(b)(l) = h(b(l))$.

The top- and bottom-elements of the lattice B are characterized by $\forall l.\ \bot_B(l) = \bot_S$ and $\forall l.\ \top_B(l) = \top_S$.

The meaning of a question will be defined in terms of an inference operator:

Inference operator: $I_{3p}: B \to B$

$$I_{3p}(b)(l) = \bigsqcap_{c \in p} (\text{unify}_3(l, hd(c')) \sqcup \bigsqcup_{l' \in tl(c')} b(l'))\Big|_{var(l)}$$

where c' denotes c with variables renamed properly so:
$var(l) \cap var(c') = \emptyset$.

As for I_{2p} so for I_{3p}: The restriction "$|_{var(l)}$" means ignorance of specific choices of renaming, which therefore lose their importance.

By generalizing th. 3_i), we may easily observe that B is a complete lattice. Furthermore it is evident that I_{3p} is a monotonic function, and so has least and greatest fixed points. The following result is importantly different from theorem 2:

Theorem 4

i) $h(I_{3p}^n(\bot_B)) = I_{2p}^n(\top_A)$, for any ordinal $n \leq \omega$.

ii) $h(I_{3p}^n(\top_B)) = I_{2p}^n(\bot_A)$, for any ordinal $n \leq \omega$.

iii) $lfp(I_{3p}) = I_{3p}^{\omega}(\bot_A)$.

iv) $gfp(I_{3p}) = I_{3p}^{\omega}(\top_A)$.

v) $\bot_B \sqsubseteq I_{3p}^{\omega}(\bot_B) = lfp(I_{3p}) \sqsubseteq gfp(I_{3p}) = I_{3p}^{\omega}(\top_B) \sqsubseteq \top_B$, and any of the above "inclusions" is proper for some program.

Proof:

i) This is demonstrated by induction

Basis: $h(\bot_B) = \top_A$.

Successor step for finite ordinals: $h(I_{3p}^{n+1}(\bot_B))(l)$

$= h(I_{3p}^{n+1}(\bot_B)(l))$

$= h(\prod_{c\in p}(unify_3(l,hd(c')) \bigsqcup \bigsqcup_{l'\in tl(c')} I_{3p}^{n}(\bot_B)(l'))\Big|_{var(l)})$

Th.3.

$= \bigcup_{c\in p}(unify_2(l,hd(c')) \wedge \bigcap_{l'\in tl(c')} h(I_{3p}^{n}(\bot_B))(l'))\Big|_{var(l)}$

Ind.

$= \bigcup_{c\in p}(unify_2(l,hd(c')) \wedge \bigcap_{l'\in tl(c')} I_{2p}^{n}(\top_A)(l'))\Big|_{var(l)}$

$= I_{2p}^{n+1}(\top_A)(l)$.

Limitstep: $h(I_{3p}^{\omega}(\bot_B))(l)$

$= h(I_{3p}^{\omega}(\bot_B)(l))$

$= h(\bigsqcup_{n<\omega} I_{3p}^{n}(\bot_B)(l))$

Th.3.

$= \bigcap_{n<\omega} h(I_{3p}^{n}(\bot_B))(l)$

Ind.

$= \bigcap_{n<\omega} I_{2p}^{n}(\top_A)(l)$

$= I_{2p}^{\omega}(\top_A)(l)$

ii) This is similar to i).

iii) $I_{3p}(\bigsqcup_n I_{3p}{}^n(\perp_B))(l)$

$= \bigsqcap_{c\in p} (\text{unify}_3(l,\text{hd}(c')) \sqcup \bigsqcup_{l'\in tl(c')} \bigsqcup_n I_{3p}{}^n(\perp_B)(l')) \Big|_{\text{var}(l)}$

Th.3.

$= \bigsqcup_n (\bigsqcap_{c\in p} (\text{unify}_3(l,\text{hd}(c')) \sqcup \bigsqcup_{l'\in tl(c')} I_{3p}{}^n(\perp_B)(l')) \Big|_{\text{var}(l)})$

$= \bigsqcup_n I_{3p}(I_{3p}{}^n(\perp_B))(l)$

$\bigsqcup_n I_{3p}{}^n(\perp_B)(l)$

iv) This is similar to iii).

v) The result follows from i)-iv) and theorem 2.

Q.E.D.

We are now ready to specify the meaning of a question:

Meaning of a question:

Q_3: L→ P→ |S|	The positive meaning of a question: $Q_3[\![l]\!]p = gfp(I_{3p})(l)$.
N_3: L→ P→ |S|	The negative meaning of a question: $N_3[\![l]\!]p = lfp(I_{3p})(l)$.

We may immediately obtain:

Corollary 2:

i) $h(Q_3[\![l]\!]p) = Q_2[\![l]\!]p$.

ii) $h(N_3[\![l]\!]p = \int N_2[\![l]\!]p$.

We have now fulfilled what was previously promised and conclude with an example:

Example 9:

Consider the program P (p_2 of ex. 4):

P: $r \Leftarrow g(Y)$
$g(f(X)) \Leftarrow g(X)$.

We may compute:

$$I_{3p}(\perp_A)(l) = \begin{cases} \underline{l=r}: & \perp_s|_{\emptyset} = \perp_s \\ \underline{l=g(Z)} & \overline{\{\{[Z \to f(X)]\}\}}|_{\{Z\}} = \overline{\{\{[Z \to f(\underline{X})]\}\}} \end{cases}$$

$$I_{3p}^{n}(\perp_A)(l) = \begin{cases} \underline{l=r}: & \perp_s \\ \underline{l=g(Z)}: & \overline{\{d_n\}}, \text{ where } d_n = \{[Z \to f^n(\underline{X}_n)]\} \end{cases}$$

$$I_{3p}^{\omega}(\perp_A)(l) = \begin{cases} \underline{l=r}: & \perp_s \\ \underline{l=g(Z)}: & \overline{\{d_1, d_2, \ldots.\}}, \text{ where } d_n = \{[Z \to f^n(\underline{X}_n)]\} \end{cases}$$

$$I_{3p}^{\omega+1}(\perp_A)(l) = I_{3p}^{\omega}(\perp_A)(l).$$

Observe that $I_{3p}^{\omega}(\perp_A)(g(Z)) \neq \top_s$ and $lfp(I_{3p}) = I_{3p}^{\omega}(\perp_A)$ in agreement with th. 4.

6. A denotational semantics

The semantics of logic programming was connected to the cd-domain of substitutions in the previous section. This result hides a denotational semantics, which will be unfold now.

We define semantic functions for each of the major syntactic components of a logic program. Every program has two different meanings (P_+,$P_\div$) corresponding to the positive and negative conclusions respectively, which may be drawn from the program in question.

Abstract syntax

	I a countable set of identifiers
	V a countable set of variables
$T = IT^*\|_V$	terms
$L = T$	literals
$N = L^*$	negative clauses
$C = LN$	positive clauses
$P = C^*$	programs
$Q = L$	questions

Semantic domains

$\|S\|$	substitutions
$A = Q \to \|S\|$	substitution-answers

Semantic functions

$$\mathcal{L} : \mathbf{L} \to \mathbf{A} \to |\mathbf{S}|$$

$$\mathcal{N} : \mathbf{N} \to \mathbf{A} \to |\mathbf{S}|$$

$$\mathcal{C} : \mathbf{C} \to \mathbf{A} \to \mathbf{A}$$

$$\mathcal{P}_+ : \mathbf{P} \to \mathbf{A}$$

$$\mathcal{P}_\div : \mathbf{P} \to \mathbf{A}$$

$$\mathcal{L}[\![l]\!]a = a(l)$$

$$\mathcal{N}[\![n]\!]a = \bigsqcup_{l \in n} \mathcal{L}[\![l]\!]a$$

$$\mathcal{C}[\![l \cdot n]\!]a = \lambda q.(mgu(l',q) \sqcup \mathcal{N}[\![n']\!]a)\big|_{var(q)} \qquad (*)$$

$$\mathcal{P}_+[\![p]\!] = gfp(\lambda a. \bigsqcap_{c \in p} \mathcal{C}[\![c]\!]a)$$

$$\mathcal{P}_\div[\![p]\!] = lfp(\lambda a. \bigsqcap_{c \in p} \mathcal{C}[\![c]\!]a)$$

(*): (l',n') denotes (l,n) with fresh variables, not contained in q. The specific choice of new variable names is unimportant, because the variables of (l',n') are made anonymous by the restriction "$|_{var(q)}$".

The semantics contain no detailed specification of the most general unifier (mgu). In section 4 we gave a recursive definition of mgu, namely $unify_3$. From this definition we may obtain a fixed point characterization on request.

Logic programming has a procedural interpretation, in which a positive clause may be regarded as a procedure declaration. This view leads us to perceive substitution answers (A) as continuations. The meaning of a positive clause ($\mathcal{C}$), becomes a continuation-transformation.

Unfortunately the above semantics is not entirely denotational. A denotational semantics should only refer original syntactic structures inside "$[\![\cdot]\!]$". Yet the occurrence of the renamed structure n' inside the brackets of $N[\![\cdot]\!]$, breaks this principle. I have not found a satisfactory modification that is able to repair the defect.

The correctness of the denotational semantics follows from the observation that $P_+[\![p]\!] = gfp(I_{3p})$ and $P_\div[\![p]\!] = lfp(I_{3p})$. We state this equivalence in a theorem:

Theorem 5

i) $Q_3[\![1]\!]_p = P_+[\![p]\!](1)$

ii) $N_3[\![1]\!]_p = P_\div[\![p]\!](1)$

Theorem 5 guarantees that the denotational semantics is fully-abstract in the sense of section 4.

7. Conclusion

For logic programming we have constructed a semantics which emphasizes finite computability. By providing a detailed description of substitutions as dataobjects, our semantics is nearer to an actual implementation than earlier fixed point semantics. Simultaneously, any commitment to sequentiality is avoided. Hence this semantics constitutes a basis for new parallel as well as sequential implementations. Such implementations remain to be constructed. In the process of building a domain of substitution it seemed impossible to gear domain theory to our specific needs without modifications. The result became the cd-domain. It remains to be investigated whether the cd-domain can be reduced to a simpler structure, as suggested at the end of section 4.

Acknowledgement:

I am grateful to Mogens Nielsen for his valuable suggestions. In particular I have received detailed comments from him on earlier drafts of this paper.

My work has also benefited from discussions with Erik Meineche Schmidt, Peter Mosses, Brian Mayoh and Glynn Winskel.

I wish to thank Julie Franks for correcting my English, although I am responsible for any remaining errors.

This paper was typed by Angelika Paysen and Winnie Sørensen. They have done an excellent job in deciphering my manuscript.

Appendix

As promised previously, we are going to prove Th. 3 ii), which is part iv) of the following lemma.

Lemma

i) If two terms $t_1, t_2 \in T$ equipped with a finite substitution $s \in S$ (def(s) is finite) are mutually consistent ($h(s) \cap \text{unify}_2(t_1, t_2) \neq \emptyset$), then we may construct their most general unifier: the substitution $\text{unify}(t_1, t_2, s) \in S$, where $\text{def}(\text{unify}(t_1, t_2, s))$ is finite and $h(\text{unify}(t_1, t_2, s)) = h(s) \cap \text{unify}_2(t_1, t_2)$.

ii) If two substitutions $s_1, s_2 \in S$ are finite and mutually consistent ($\text{def}(s_1), \text{def}(s_2)$ are both finite and $h(s_1) \cap h(s_2) \neq \emptyset$), then we may construct their conjunction: the substitution $\text{conj}(s_1, s_2) \in S$, where $\text{def}(\text{conj}(s_1, s_2))$ is finite and $h(\text{conj}(s_1, s_2)) = h(s_1) \cap h(s_2)$.

iii) If two dataobjects $s_1, s_2 \in \mathcal{D}_s$ are mutually consistent ($h(s_1) \cap h(s_2) \neq \emptyset$) then we may construct their conjunction: the dataobject $\text{conj}_d(s_1, s_2) \in \mathcal{D}_s$, where $h(\text{conj}_d(s_1, s_2)) = h(s_1) \cap h(s_2)$.

iv) A finite element $s \in |S|$ may be represented by a single disjunction, i.e. we can find $s_1, \ldots, s_n \in \mathcal{D}_s$ ($n \geq 0$), such that $s = \{\{s_1, \ldots, s_n\}\}$.

Proof:

i) The unify-algorithm is defined recursively as follows:

$$\text{unify}(v,t,s) = \begin{cases} \text{unify}(s(v),t,s) & , \ v \in \text{def}(s) \\ s[v \rightarrow t] & , \ v \notin \text{def}(s) \end{cases}$$

$$\text{unify}(i\bar{t}_1, t_2, s) = \begin{cases} \text{unify}(t_2, i\bar{t}_1, s) & , \ t_2 \in V \\ \text{unify}_*(\bar{t}_1, \bar{t}_2, s) & , \ t_2 = i\bar{t}_2 \end{cases}$$

$$\text{unify}_*(t_1 \cdot \bar{t}_1, t_2 \cdot \bar{t}_2, s) = \text{unify}_*(\bar{t}_1, \bar{t}_2, \text{unify}(t_1, t_2, s))$$

$$\text{unify}_*(\text{nil}, \text{nil}, s) = s.$$

The partial correctness of unify is easily established. The total correctness follows from the preconditions of i).

ii) The conj-algorithm is defined recursively as follows:

$$conj(s_1,s_2) = \begin{cases} s_1 & , def(s_2)=\emptyset \\ unify(v,s_2(v),conj(s_1,s_2[^{v}/_{v}])) & , def(s_2)\neq\emptyset \\ \text{and } v \text{ is minimal in some order of } def(s_2). \end{cases}$$

The partial correctness of conj is proved by induction, based on the following assumption: (and using i))

$$p(n): \#def(s_2)=n \Rightarrow h(conj(s_1,s_2))=h(s_1)\cap h(s_2),\ def(conj(s_1,s_2)) \text{ is finite.}$$

The total correctness is trivial.

iii) The $conj_d$-construction is straightforward:

$$conj_d(s_1,s_2)=(v_1\cup v_2, conj(s_1',s_2')), \text{ where } s_1=(v_1,s_1') \text{ and } s_2=(v_2,s_2').$$

Furthermore the anonymous variables of s_1 and s_2 have been renamed, so as to avoid unwanted common variables, i.e. s_1,s_2 fulfil:

a) $var(s_1')\cap var(s_2')\subseteq v_1\cup v_2$

b) $var(s_1')\cap v_2\subseteq v_1$, $var(s_2')\cap v_1\subseteq v_2$

where $var(s)=ref(s)\cup def(s)$.

We observe

$$h(conj(s_1,s_2))=h(conj(s_1',s_2'))|_{v_1\cup v_2} \overset{ii)}{=} (h(s_1')\cap h(s_2'))|_{v_1\cup v_2}$$

$$\overset{a)}{=} h(s_1')|_{v_1\cup v_2} \cap h(s_2')|_{v_1\cup v_2} \overset{b)}{=} h(s_1')|_{v_1} \cap h(s_2')|_{v_2} = h(s_1)\cap h(s_2)$$

iv) This is proved by induction based on the following assumption:

$P(n)$: $\Downarrow$ $s=\{\overline{d_1,\ldots,d_n}\}$ for finite disjunctions $d_i \subseteq \mathcal{D}_s$

There exist a single finite disjunction $d \subseteq \mathcal{D}_s$, such that $s=\{\bar{d}\}$.

Basis:

$s=\bar{\emptyset}$ means that $s=\perp_s$ and we put $d=\{\Delta_s\}$.

Inductionstep:

Assume $s=\{\overline{d_1\ldots,d_n,d_{n+1}}\}$ for finite disjunctions $d_i \subseteq \mathcal{D}_s$.

then $s=\{\overline{d_1 \ldots,d_n}\} \sqcup \{\overline{d_{n+1}}\} \overset{ind}{=} \{\bar{\tilde{d}}\} \sqcup \{\overline{d_{n+1}}\} = \{\overline{\tilde{d},d_{n+1}}\}$, where the finite disjunction $\tilde{d}$ exist due to the induction assumption.

Now let:

$$d=\{conj_d(s_1,s_2) \mid s_1 \in \tilde{d}, s_2 \in d_{n+1},\ h(s_1) \cap h(s_2) \neq \emptyset\}.$$

We compute:

$$\{\bar{d}\}=\{d' \mid h(d) \subseteq h(d')\} \overset{iii)}{=} \{d' \mid h(\{\tilde{d},d_{n+1}\}) \subseteq h(d')\} = \{\overline{\tilde{d},d_{n+1}}\}, \text{ i.e. } s=\{\bar{d}\}.$$

Q.E.D.

References

[1] Apt, van Emden: "Contributions to the theory of logic programming". JACM, Vol. 29, 1982, pp. 841 - 862.

[2] Kowalski: "Predicate logic as a programming language". Proceedings of the IFIP 1974 congress, pp. 569 - 574.

[3] Robinson: "A machine oriented logic based on the resolution principle". JACM, Vol. 12, 1965, pp. 23 - 41.

[4] Van Emden, Kowalski: "The semantics of predicate logic as a programming language". JACM, Vol. 23, 1976, pp. 733 - 742.

[5] Scott: "Domains for denotational semantics". A corrected and expanded version of a paper prepared for ICALP '82, Aarhus, Denmark, July 1982.

[6] Lloyd: "Foundations of Logic Programming". Springer Verlag, 1984.

[7] Jaffar, Lassez, Lloyd: "Completeness of the negation as failure rule". Proceedings of the IJCAI 1983 conference, pp. 500 - 506.