Logic Programming and Substitutions

Gudmund Frandsen
Department of Computer Science
Aarhus University
Ny Munkegade
DK-8000 Aarhus C
Denmark

1. Introduction

In this paper we construct a domain of substitutions that may form the basis of a semantics for logic programming, since a computation involving a logic program and a query naturally results in a substitution. Our domain differs from usual domains in that ω-continuity is defined with respect to both increasing and decreasing chains. This difference is a consequence of our principle of computational equivalence: The results of two computations should be given the same denotation precisely when they bear the same finitely computable information content.

In the following we discuss the principle of computational equivalence alternated with a stepwise construction of the substitution domain. Finally we present a denotational semantics based on the finished domain.

This paper is abstracted from a research report [2], where further details can be found. Concerning prerequisites in the areas of logic programming and Scott-domains the reader may consult [3] and [4] respectively.

2. Information content

We mentioned above that a computation involving a logic program and a query naturally results in some sort of substitution. In a semantic domain such substitutions should have the same denotation if and only if they are equivalent in an appropriate sense. Consider the following example:

Example 1.

Program: Q(f(Z),g(Z)) ← p(Z).
P(a).

Query: Q(X,Y)?

Result of computation in the form of some possible syntactic answers:

1): [X=f(Z), Y=g(Z), Z=a].
2): [X=f(a), Y=g(a), Z=a].
3): [X=f(a), Y=g(a)].

Underscore denotes an anonymous variable. All variables that do not occur in the query are made anonymous in the answer. By giving names to anonymous variables, we obtain the possibility of relating different undefined values. However, the information content of a substitution is independent of a specific choice of names; answers 1) and 2) are equivalent. In the case of ex. 1 we may completely remove anonymous variables and obtain the answer 3). The information contents of the three substitution answers are identical. Consequently the three substitutions should have the same denotation in a domain of substitutions.

We are going to build a domain based on Scott’s information system framework [4]. Apparently concepts such as information content and anonymous variable are important, when dealing with computational equivalence. We will demonstrate, how these concepts can form the basis of a definition of substitution dataobject.

3. Substitution dataobject

Initially we provide some basic definitions. The syntactic basis consists of one countable set of identifiers and another of variables. We do not bother whether an identifier denotes a function or a predicate, neither do we deal with arities. On the basis of identifiers and variables we form the recursively defined set of terms and the set of substitutions.
Basic definitions:

- \( I \)  
  a countable set of identifiers

- \( V \)  
  a countable set of variables

- \( T = IT*V \)  
  the set of terms

- \( S = V \rightarrow T \)  
  the set of substitutions

- \( T_0 = IT_0* \)  
  the set of groundterms

- \( S_0 = V \rightarrow T_0 \)  
  the set of groundsubstitutions

- \( t[s] = "t, where any variable occurrence V is replaced by s(v)." \)

- \( h: S \rightarrow 2^{S_0} \)  
  the groundexpansion of substitutions: \( h(s) = (s' \in S_0 \mid \forall v. \ s(v)[s'] = s'(v)) \)

- \( ;: 2^{S_0} \times 2^{S_0} \rightarrow 2^{S_0} \)  
  the restriction operator, which "erases" any information on variables that are not members of the specified set: \( s' |_{V'} = \{ s' \in S_0 \mid \exists s' \in S'. \ \forall v \in V'. \ s(v) = s'(v) \} \)

Groundterms and groundsubstitutions are introduced to measure information content, and the restriction operator is defined to formalize the intuitive notion of anonymous variable. For a treatment of the intuition behind groundexpansions the reader may consult [2]. Here we present a simple example:

Example 2.

Let the substitution \( s \) be defined as follows:

\[
 s(v) = \begin{cases} 
 f(z), & v = X \\
 g(z), & v = Y \\
 a, & v = z \\
 v, & \text{otherwise} 
\end{cases}
\]

The groundexpansion of \( s \) is:

\[
 h(s) = \{s \in S_0 | s(X) = f(s(z)), s(Y) = g(s(z)), s(z) = a\}
\]

The restriction of this expansion to \( (X, Y) \) is easily constructed:

\[
 h(s)|_{(X, Y)} = \{s \in S_0 | s(X) = f(a), s(Y) = g(a)\}
\]

which should be the common groundexpansion of the three equivalent substitution dataobjects occuring in ex. 1.

The set of dataobjects is now formed from the finite, cyclefree substitutions with distinct named (non anonymous) variables:

Substitution dataobjects:

\[
 \mathcal{D}_s = \{ (V', s) | (V' \times S | i) V' \text{ and } \text{def}(s) \text{ are finite } ii) h(s) \neq \emptyset \text{ (s is cyclefree)} \} 
\]

where \( \text{def}(s) = (v | s(v) \neq v) \)

\[
 ;: 2^{X' \times S} \rightarrow 2^{X' \times S} \text{ the restriction operator on dataobjects:} \\
 (V', s)|_{V'} = (V' \setminus V, s) 
\]

\[
 h: 2^{S_0} \rightarrow 2^{S_0} \text{ the groundexpansion of dataobjects:} \\
 h((V', s)) = h(s)|_{V'} 
\]
In a dataobject \((V', s) \in D_s\), the substitution \(s\) may contain information on both named variables \((V')\) and anonymous variables \((\text{atom}(V'))\). The set of named variables may be reduced by use of the restriction operator. The ground expansion \(h\) measures the information content of a dataobject, i.e. a small ground expansion corresponds to a big information content. In this way a dataobject with empty expansion is inconsistent. However, such dataobjects are excluded from \(D_s\) by restriction ii) in the definition. At the opposite end of the information scale reside the uninformative dataobjects, among which one is special: \(\Delta_s = (V's)\), where \(V' = \emptyset\) and \(\text{def}(s) = \emptyset\). It is easily seen that \(h(\Delta_s) = S_0\). Many dataobjects are equally uninformative, since a dataobject \((V', s)\) with trivial substitution \(\text{def}(s) = \emptyset\) or a dataobject \((V', s)\) without named variables \((V' = \emptyset)\) both have expansion \(S_0\).

It may now be easily verified that the three dataobjects of ex.1 have equal information contents by use of the formal definition (cfr. ex.2).

Before continuing the construction of a Scott-information-system we consider nondeterministic computations.

4. Computational equivalence

Consider the following example:

**Example 3.**

**Program:**

\[
p(f(X)) = p(X),
p(a).
\]

**Query:**

\[
p(Y)\]

*result of nondeterministic computation in the form of possible syntactic answers:*

1) \([Y=?a]\)
2) \([Y=f(a)]\)
3) \([Y=f^2(a)]\)

Unlike ex.1, the possible answers above are not equivalent. Here each answer corresponds to a specific sequence of nondeterministic choices made during the computation. We may syntactically describe the result of such a nondeterministic computation as an infinite disjunction of substitution dataobjects. In the case of ex.3:

\[s = \{[Y=a], [Y=f(a)], [Y=f^2(a)], \ldots\}.\]

What information content should we associate with such a disjunction? It seems natural to take the ground expansion of a disjunction to be the union of the individual ground expansions \([2]\), i.e. the information content of a disjunction is less than the information content of any conjunct. In the above case we get \(h(s) = \bigcup_{i=0}^{1} h([Y=f^i(a)])\). With a partial order based on information content, \(s\) may be perceived as the greatest lower bound of all its finite approximations (i.e. the finite subdisjunctions of \(s\)). What denotation is "correct" for an (in)finite disjunction of dataobjects? Scott's powerdomain of indeterminacy [4] has a natural denotation for each of the finite approximations. However, the greatest lower bound of these denotations is the uninformative bottom element \(\nabla_s\) which is not a good approximation for \(s\), since \(h(\nabla_s) = S_0 = h(s)\). As a consequence we must build a new domain capable of representing such infinite disjunctions in a satisfactory way.

There is yet another reason for demanding a new type of domain. If in a case \(s\) had an information content equal to the information content of the bottom element (i.e. none), we should not use \(\nabla_s\) as a denotation for \(s\) anyway, because every finite approximation to \(s\) contains some information. We may formulate:

**Principle of computational equivalence:**

Two substitutions should have the same denotation iff

1) They bear equal information content and
2) the truth of 1) is established in a finite computation.

We need a domain, which is continuous for decreasing chains such that the glb of a sequence of informative approximations is finitely informative (and possibly infinitely uninformative).
Such a domain is easily constructed if we reverse the information order, so the above demand is on increasing chains. Yet if we want to treat negation by finite failure in addition, we will need \( \omega \)-continuity of decreasing and increasing chains. Hence we do not solve the problem by reversing the information order.

For a more thorough discussion of negation by finite failure, we refer to [2]. Here should just be mentioned that Apt and van Emden [1] characterize the semantics of negation by finite failure by means of a concept equivalent to the information content of a term, although they do not discuss the necessity of a finite computability restriction. Consequently they have to deal with a discontinuous operator.

We are now going to realize the principle of computational equivalence by building a domain, where \( \omega \)-continuity is defined with respect to both decreasing and increasing chains.

5. The cd-domain of substitutions

When building Scott domains from information systems, a domain element consists of the conjunction of all those data-objects that contain an amount of information less than or equal to the information content of the element itself. We will now build a domain (the cd-domain), where each element consists of the conjunction of all those disjunctions of data-objects that contain no more information that the information content of the element itself.

\[
\begin{align*}
\text{Basic definitions:} & \\
\mathcal{P}_s & = 2^{\mathcal{D}_s} \\
\mathcal{I}_s & = 2^{S_0} & \text{The set of all conjunctions of disjunctions of substitution data-objects (cds's).} \\
\mathcal{H}_s & = \mathcal{P}_s \cap \mathcal{I}_s & \text{The ground-expansion generalized to} \\
\mathcal{E}_s & = \{ c \in \mathcal{P}_s | c \text{ is finite, \forall d(c, d \text{ is finite})}. \text{The set of finite cds's.} \\
\end{align*}
\]

We represent conjunctions of disjunctions by sets of sets (cds's). The ground-expansion is designed in order to measure the information content of a cds according to this representation. If \( h(c_1) \subseteq h(c_2) \) then \( c_1 \) contains more information than \( c_2 \). The set of finite cds's, \( \mathcal{E}_s \), is all we need to represent the result of a finite computation. The information system of substitutions is now formed by defining consistency and entailment for such finite cds's.

\[
\begin{align*}
\text{Information system of substitutions: } & (\mathcal{P}_s, \mathcal{I}_s, \mathcal{H}_s, \mathcal{E}_s) \\
\mathcal{P}_s & \text{ The set of data-objects with a distinct uninformative element has already been defined.} \\
\mathcal{I}_s & \text{ The set of finite cds's, which are consistent, i.e. Scott's Con} \\
\mathcal{H}_s & \text{ modified to conjunctions of disjunctions.} \\
\mathcal{E}_s & \text{ The entailment relation, i.e. Scott's } \vdash \text{ modified to conjunctions of disjunctions.} \\
\end{align*}
\]

If a cds is consistent then there exists a ground-substitution that is included in at least one alternative (disjunct) of every conjunct of this cds. Unlike Scott we define entailment from an inconsistent cds. A finite cds \( c \) entails every data-object, which contains no more information than \( c \) contains. In particular, an inconsistent cds entails everything. The cd-domain of substitutions is defined as the set of deductively closed cds's ordered by information content, i.e. set-inclusion:
6. A denotational semantics

We start by defining an abstract syntax, where literals are the basic syntactic unit.

\[
\text{Abstract syntax:} \quad 
\begin{align*}
L & \quad \text{The set of literals is identical to the set of terms defined previously} \\
N=L^* & \quad \text{Negative clauses} \\
C=\text{IN} & \quad \text{Positive (definite) clauses} \\
P=C^* & \quad \text{Programs}
\end{align*}
\]

As mentioned priorly we do not distinguish predicate and function identifiers \((L=T)\). Our motivation is merely to obtain technical simplicity. So the result should not be perceived as a proposal to extend logic programming beyond first order logic.

When specifying semantics, we shall see that the list ordering \((\ast\text{-notation})\) 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 with respect to clauses \((P=C^*)\).

Apart from the domain of substitutions we need a domain of program meanings, i.e. functions from questions (literals) to answers (substitutions).
The semantics contains no detailed specification of the most general unifier (mgu). There exist recursive definitions of mgu [2], from which we may obtain a fixed point characterization on request.

The above renaming of could be handled by an explicit introduction of environments if wanted.

Logic programming has a procedural interpretation, in which a positive clause may be regarded as a procedure declaration. This view leads us to perceive program meanings (M) as continuations. The meaning of a positive clause (C), becomes a continuation transformation. The above semantics is equivalent to the model theoretic semantics:

Theorem:

Let a program p and a question 1 be given and form the completion* p' of p. Then the following holds:

i) The denotation of a program is correct:
\[
(s \in S_0 \models 1[s]) = h(p[1]).
\]

ii) The same is true for negation by finite failure:
\[
(s \in S_0 \models 1[\neg s]) = h(p[1]).
\]

(*) : The completion of a logic program is liberally formed by replacing "if" by "if and only if", e.g. the completion of the program in ex.1 is:
\[
\forall X, Y. (q(X, Y) \equiv \exists Z. X = f(Z) \land Y = g(Z) \land p(2)), \forall X. (p(X) \equiv X = a) \text{ and some axioms about equality.}
\]

Proof: see [2].

Apart from the correctness stated above, it should be noted that the present semantics are more detailed about the notion of substitution than well-known semantics [1,3]. Nevertheless our semantics are independent of specific resolution strategies. Besides we have managed to characterize negation by finite failure in terms of the least fixed point of a continous operator.
7. References:


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