

Partial Evaluation for Hierarchies of Logic Theories

M. Bugliesi

E. Lamma P. Mello

ENIDATA S.p.A
Viale Aldo Moro 38,
40121 Bologna

DEIS University of Bologna
Viale Risorgimento 2,
40136 Bologna

Abstract

We discuss the impact of Partial Evaluation within the framework of structured logic programming. We define a general scheme for Partial Evaluation to be applied to a wide class of structuring policies for logic programming, capable of supporting both block- and inheritance-based systems. We show how the properties of soundness and completeness of Partial Evaluation in Logic Programming are preserved in this extended scheme.

Keywords: Partial Evaluation, Structured Logic Programming, Blocks, Modules, Inheritance-based Schemes.

1 Introduction

Structuring mechanisms have become nowadays a crucial topic in Logic Programming. Several authors have addressed this problem in the literature, on the basis of different approaches: modules and blocks [?, ?, ?, ?, ?], inheritance [?, ?, ?] and viewpoints, [?, ?].

Some efforts have recently been devoted, [?, ?], to the design of a unifying framework in which a wide range of current proposals can be integrated in a uniform and coherent way. This approach, inspired by [?], is based on the idea that a program can be structured as a collection of elementary components, each component being a separate logic theory called *unit*. Units can be (possibly dynamically) connected into linear hierarchies called *contexts* and contexts, in turn, provide the (possibly dynamic) data base of clauses used for the evaluation of the queries. Depending on the different policies adopted for unit combination, different structuring mechanisms can be identified. Blocks, modules, and inheritance-based systems [?] are easily implemented by means of statically configured hierarchies of units, whereas dynamic configurations provide a natural support for some well-known Artificial Intelligence techniques such viewpoints and hypothetical reasoning.

Some interesting results have also been achieved from the implementation point of view. In [?], the authors address the relevance of a compilative approach based on the definition of an extended Warren Abstract Machine, [?], where new instructions and data structures have been introduced to deal with units and contexts.

In this paper we discuss the application of Partial Evaluation as a further enhancement to the compilative approach proposed in [?]. We focus on statically configured systems since they are the best-suited for such a compilative approach. Static configurations allow in fact predicate calls to be directly bound to the corresponding predicate definitions at compile-time thus eliminating most of the run-time overhead due to context handling.

We introduce a definition of Partial Evaluation which extends that given in [?] to capture the notions of unit and unit composition, and which produces a specialization of the source program with respect to a given goal to be evaluated in a given context.

An interesting feature of our technique is that it preserves the original structure of the program so that no collation of several units into a more compact configuration occurs during the transformation. This ensures non-replication of code for the new program, high generality for it, and finally, full compatibility with the compilation technique addressed in [?].

Furthermore it allows to isolate some closedness conditions for the soundness and the completeness of the transformation, which involve only syntactic checks on the partially evaluated program and the goal. For block-based systems these conditions actually correspond to those introduced in [?] for definite logic programs; for inheritance-based systems, instead, some further checks on the static structure of the program are required.

The paper is structured as follows. In section 2 we first sketch the basic notions of unit and context and then we show how they can be used to implement block- and inheritance-based systems. In section 3, the theory of partial evaluation for these systems is presented together with some simple examples. In section 4, we address the soundness and completeness issues and finally, in section 5, we show some relevant applications.

2 Hierarchies of Logic Programming Theories

Our characterization of structured logic programs originates from the Contextual Logic Programming paradigm introduced by [?]. The key idea is that a program can be conceived as a collection of independent modules called *units*. A unit is simply identified by the set of clauses it defines and by a unique, atomic name used to denote it. Units can be (possibly dynamically) connected into contexts and contexts, in turn, provide the set of definitions for the evaluation of the queries. As a matter of fact, contexts are represented as ordered lists of units of the form $[u_N, \dots, u_i, \dots, u_1]$ and denote the union of the sets of clauses of the composing units.

Example 2.1 Let P be the program composed by the following units:

$$\begin{array}{lll} \mathbf{unit}(u_1) : & \mathbf{unit}(u_2) : & \mathbf{unit}(u_3) : \\ a(X) : -b(X). & b(1). & c(X) : -a(X). \end{array}$$

The proof of the goal $c(X)$ in the context $[u_3, u_2, u_1]$ corresponds, in logic programming, to a proof with respect to the set of clauses:

$$\begin{array}{l} c(X) : -a(X). \\ b(1). \\ a(X) : -b(X). \end{array}$$

Different policies for composing units into contexts can be adopted and, accordingly, different classes of structuring mechanisms can be identified as discussed in [?] and [?].

2.1 Static and Dynamic Systems

A first relevant issue concerns the distinction between static and dynamic composition.

Statically configured systems are systems in which each unit has a *fixed* associated context. This means that, if $[u_N, \dots, u_1]$ is the context associated with a unit u , then, whenever u is asked for the proof of a goal g , the evaluation of g actually takes place in the context $[u, u_N, \dots, u_1]$. The association of a unit with a context is obtained by establishing explicit inheritance links between units. Therefore, when a unit gets defined, its **parent** unit must be also specified and its associated context is obtained by recursively computing the ordered list of its ancestors. Formally, if $U(P)$ and $C(P)$ denote respectively the set of units of a program P and the set of contexts formable over $U(P)$, the context associated with a unit u is computed by the function $hierarchy : U(P) \mapsto C(P)$ defined as follows:

$$hierarchy(u) = \begin{cases} u.hierarchy(u') & \text{if } u' \text{ is the parent unit of } u. \\ \lfloor \rfloor & \text{if } u \text{ is the top unit of the hierarchy} \end{cases}$$

A *top*, empty unit, common to all the hierarchies, is assumed. A switch operator “:” allows us the switching from one hierarchy to the others. The invocation of a goal of the form $u : g$ in the current hierarchy causes a switch to the hierarchy associated with the unit u .

Example 2.2 Let P be the following structured program:

$$\begin{array}{lll} \mathbf{unit}(u_1, \mathbf{parent}(top)) : & \mathbf{unit}(u_2, \mathbf{parent}(u_1)) : & \mathbf{unit}(u_3, \mathbf{parent}(u_2)) : \\ a(X) : -b(X). & b(1). & c(X) : -a(X). \end{array}$$

The call $u_3 : c(X)$ enforces $[u_3, u_2, u_1]$ as the new current context for $c(X)$. The unit specified as parameter of the **parent** structure represents the inheritance link for the unit itself.

Dynamically configured systems, on the other side, provide a more flexible framework in which a context $[u_N, \dots, u_1]$ can be dynamically extended to include new units at each instant of the computation. Each unit can have therefore a dynamic associated context. This is actually the idea behind the proposal of [?].

2.2 Conservative/Evolving Systems

The evaluation of a goal in a context involves two issues: first we have to identify, in that context, the appropriate set of clauses which provide the definition for the goal, and then we solve the goal with respect to that definition.

An important remark here concerns the distinction between conservative and evolving policies. Let's suppose that $C = [u_N, \dots, u_i, \dots, u_1]$ is the current context.

An *evolving system* is a system in which, for each predicate call g occurring in the unit u_i , the corresponding predicate definition is given by the clauses contained in the whole context C . Examples of evolving systems are Miller's one [?], Multi-Prolog [?] and N-Prolog [?].

A *conservative system* is a system in which, for each predicate call g occurring in the unit u_i , the corresponding predicate definition is given by the clauses contained in the the sub-context $[u_i, \dots, u_1]$. The definitions *visible* from a unit u , are therefore those found in u and its ancestors in the current context. Examples of conservative systems are blocks and modules [?], Meta-Prolog [?], Contextual Logic Programming [?].

Example 2.3 Let us consider the program of example 2.2 and the top goal: $u_3 : c(X)$. We use the notation $Ctx \vdash G$, introduced in [?] (see appendix), to explicitly represent the context associated to each predicate call.

In an evolving system we get the derivation steps:

$$\begin{array}{ll} [] & \vdash u_3 : c(X) \\ [u_3, u_2, u_1] & \vdash c(X) \\ [u_3, u_2, u_1] & \vdash a(X) \\ [u_3, u_2, u_1] & \vdash b(X) \\ \mathit{success} & : \{X \leftarrow 1\} \end{array}$$

Notice that the goal $b(X)$ is proved in the whole context $[u_3, u_2, u_1]$ even though it is called in the sub-context $[u_1]$.

By converse, in a conservative system, the context for the call $b(X)$ is $[u_1]$ and, since $[u_1]$ contains no definition for $b(X)$, we get a failure as shown below:

$$\begin{array}{ll} [] & \vdash u_3 : c(X) \\ [u_3, u_2, u_1] & \vdash c(X) \\ [u_2, u_1] & \vdash a(X) \\ [u_1] & \vdash b(X) \\ \mathit{failure} & \end{array}$$

2.3 Examples of Statically Configured Systems

Most of the proposals for structuring logic programs found in the literature, can be included in the above classification. An interesting discussion about this topic can be found in [?].

In this paper we focus on statically configured systems. The main reason for this choice is that, due to their static configuration, these systems are better-suited for a compilative approach such as the one based on Partial Evaluation. The details about this point will be clarified in the next sections. What is worth mentioning here is that, even though static systems are not powerful enough to cover the whole range of structuring mechanisms, nonetheless they capture several interesting proposals in this field such as modules, blocks, inheritance-based systems (see [?]).

2.3.1 Blocks and Modules

In block-based systems, static scope rules determine predicate visibility on the basis of the nesting of blocks in the program. To prove an atomic goal occurring in a clause of a given block, only those clauses defined in that block or in enclosing blocks can be used. In terms of our classification, such a behaviour corresponds to a conservative policy where each block is encapsulated into a separate unit having the enclosing block as its *parent*.

Example 2.4 Let us consider the following program presented in [?]:

$$P = \{r \rightarrow q, (((q \rightarrow p) \wedge r) \Rightarrow p) \rightarrow s\}$$

and structured into two nested blocks: b_1 , the enclosing one, which contains the definitions for q and s , and b_2 , the inner one, which contains the definitions for p and r . Block b_2 is nested into the second clause of block b_1 and, therefore, its clauses are not visible by the predicates defined in b_1 . P can be mapped to the following conservative system:

$$\begin{array}{ll} \mathit{unit}(b_1, \mathit{parent}(top)) : & \mathit{unit}(b_2, \mathit{parent}(b_1)) : \\ q : -r. & p : -q. \\ s : -b_2 : p. & r. \end{array}$$

Since the system is conservative, the goal $b_1 : s$ cannot be proved. In fact, the predicate r , which is locally defined in b_2 , is not visible from the external unit b_1 . Notice that, on the other side, in the case of an evolving system the goal s would succeed.

Such scope rules become more strict in the case of closed modules (see [?, ?]): to prove a goal in a module M , only those predicate definitions which are local to M have to be taken into account. Modular systems are actually a special case of conservative systems, where each unit defines the *top* unit as its parent and therefore it only uses its local definitions.

2.3.2 Inheritance-based systems

Inheritance- and object-based systems can be classified as evolving systems. We can interpret contexts as the explicit representation of a branch in an inheritance tree (for the sake of simplicity we will not consider multiple inheritance). The first unit in the context is the tip node, while the last one is the top of the hierarchy. Inheritance-based systems are intrinsically evolving since, as stated in [?], “a self-reference in a type or class is bound to the object on whose behalf an operation (demonstration) is being executed, rather than on the textual module (unit) in which the self-reference occurs.”

Example 2.5 Let us consider the class template language described in [?]. When we say that a bird is a special case of animal we are stating that whatever is true for animals is also true for

birds: the bird theory inherits from the animal theory. In the class template language we express this kind of relationship between classes by means of class rules:

$$\begin{aligned} \textit{bird} &\Leftarrow \textit{animal} \\ \textit{tweety} &\Leftarrow \textit{bird} \\ \textit{horse} &\Leftarrow \textit{animal} \\ \textit{person} &\Leftarrow \textit{animal} \end{aligned}$$

where:

$$\begin{aligned} \textit{animal} : & \left[\begin{array}{l} \textit{mode}(\textit{walk}). \\ \textit{mode}(\textit{run}) : -\textit{self} : \textit{no_of_legs}(2). \\ \textit{mode}(\textit{gallop}) : -\textit{self} : \textit{no_of_legs}(4). \end{array} \right. \\ \\ \textit{bird} : & \left[\begin{array}{l} \textit{mode}(\textit{fly}). \\ \textit{no_of_legs}(2) \\ \textit{covering}(\textit{feather}). \end{array} \right. \\ \\ \textit{horse} : & \left[\textit{no_of_legs}(4). \right. \\ \\ \textit{human} : & \left[\textit{no_of_legs}(2). \right. \\ \\ \textit{tweety} : & \left[\textit{no_of_wings}(2). \right. \end{aligned}$$

The call *self:g* causes the proof of *g* to be performed in the tip class of the current hierarchy, no matter what the current class is. The use of *self* allows therefore to model the expected behaviour of inheritance.

This behaviour is intrinsically evolving; in our framework this program can be translated into the following:

$$\begin{aligned} \textit{unit}(\textit{animal}, \textit{parent}(\textit{top})) : & \left[\begin{array}{l} \textit{mode}(\textit{walk}). \\ \textit{mode}(\textit{run}) : -\textit{no_of_legs}(2). \\ \textit{mode}(\textit{gallop}) : -\textit{no_of_legs}(4). \end{array} \right. \\ \\ \textit{unit}(\textit{bird}, \textit{parent}(\textit{animal})) : & \left[\begin{array}{l} \textit{mode}(\textit{fly}). \\ \textit{no_of_legs}(2) \\ \textit{covering}(\textit{feather}). \end{array} \right. \\ \\ \textit{unit}(\textit{horse}, \textit{parent}(\textit{animal})) : & \left[\textit{no_of_legs}(4). \right. \\ \\ \textit{unit}(\textit{human}, \textit{parent}(\textit{animal})) : & \left[\textit{no_of_legs}(2). \right. \\ \\ \textit{unit}(\textit{tweety}, \textit{parent}(\textit{bird})) : & \left[\textit{no_of_wings}(2). \right. \end{aligned}$$

Now the taxonomy is embedded in the parent declaration, and the self label behaviour is automatically expressed by the evolving policy.

3 Extending Partial Evaluation

Partial Evaluation (PE) has been devoted great attention in the last few years since Komorowski [?] introduced it in logic programming in 1981. PE can be conceived as a source-to-source transformation technique which, given a program *P* and a goal *G*, produces a new program *P'*, which is

more efficient than P and has the same set of answer substitutions for the goal G or its instances. The basic technique for obtaining P' from P is to construct a partial search tree for P and suitably chosen atoms as goals, and then extract the definitions — the *resultants* — associated with the branches of the tree. P' is then obtained from P by replacing the set of clauses in P , whose head contain one of the predicate symbols appearing in G , with the set of the produced resultants.

As pointed out in [?], the main foundational questions about PE concern soundness and completeness. *Soundness* of the partially evaluated program P' wrt the original program P and the goal G means that each correct (computed) answer for G and P' is a correct (computed) answer for G and P . *Completeness* is the converse of this.

In standard logic programming, soundness follows from the soundness of SLD-resolution. A further closedness condition must be satisfied by the resulting program in order to ensure completeness [?].

The application of these issues to the structured logic systems presented in section 2 involves several major extensions to the basic principles of Partial Evaluation.

In fact, we have to consider that the evaluation of the goal occurs now in a context and, furthermore, we have to take into account the modular configuration of the program.

Given a program P , a goal G and a context C , we should therefore define a new program P' which specializes P with respect to the definition of G found in the context C . Furthermore, in order to ensure the highest level of flexibility for the resulting program, we should also impose that the transformation preserve the original structure of P .

This rises further problems with respect to the case of logic programming. Given the partial tree resulting from unfolding, and the corresponding set of resultants, we have first to define an *assignment* function to determine which unit each resultant should be assigned to. Then, the derived program is obtained from the original one by replacing (possibly) all the units in the initial context with their specialized versions. These, in turn, are obtained by erasing the old definitions for the initial goal and by introducing the new ones, according to the assignment function.

In the following, we will denote by $PE(P, G, C)$ the Partial Evaluation of P with respect to a goal G and a context C . A formal definition of $PE(P, G, C)$ can be given, in two steps, in terms of the following notions of *program representation* and of *Partial Evaluation of a goal in a context*:

Definition 3.1 (Program Representation) Let $U(P)$ be the set of units of a program P . We denote by $|u|$ the set of clauses of a unit u . Then the representation of P is given by:

$$\mathcal{R}(P) = \{\langle d, u \rangle : u \in U(P), d \in |u|\}$$

Definition 3.2 (Partial Evaluation of a goal in a context) Let P be a program, C a context, G an atomic goal, and R_T the set of resultants of a partial search-tree T for P and $\langle G, C \rangle$. Let also $\mathcal{F}_{ass} : R_T \mapsto U(P)$ be a given assignment function, which assigns the resultants in R_T to the units of P . The Partial Evaluation of G in C is given by:

$$PE(G, C) = \{\langle d, u \rangle : d \in R_T \text{ and } u = \mathcal{F}_{ass}(d)\}$$

Definition 3.3 (Partial Evaluation of P wrt C , G) Let P be a program, C a context, G an atomic goal. Then:

$$PE(P, G, C) = P' \text{ such that : } \mathcal{R}(P') = (\mathcal{R}(P) \setminus \mathcal{D}(C, G)) \cup PE(G, C)$$

where $\mathcal{D}(C, G)$ is the representation of the set of definitions for the goal G contained in the context C .

The definition of the assignment function is a crucial point of our construction. In fact, different choices for the assignment function determine different configurations for the transformed program

and, correspondingly, different properties for the transformation. Therefore, the choice of the assignment function will be determined by the properties we expect from Partial Evaluation.

In logic programming, Partial Evaluation provides a specialization of the program with respect to a goal G ; accordingly we cannot expect to be able to have correct answers for calls *more general* (less instantiated) than G .

In the same way, in structured logic programming, since we specialize the program with respect to a goal in a context C , we won't be able to have correct answers for that goal in contexts more *general* than C . In this case *more general* means *logically subsuming*. As a matter of fact, we say that a context C' is logically subsumed by a context C ($C' \sqsubseteq C$) if all the theorems derivable in C are also derivable in C' . Notice that, being the composition of units in contexts a monotonic operation, it is easy to show that $C' \sqsubseteq C$ if C' is an initial sublist (a *sub-context*) of C .

Therefore, what we can expect from the transformation is that it preserves the equivalence between P and P' with respect to instances of G (as in [?]) and *sub-contexts* of C , i.e. with respect to all those computations in which (instances of) G are evaluated in sub-contexts of C .

3.1 Configuring the Transformed Program

Even in sub-contexts of the initial one, choosing an assignment function that ensures the safeness of the transformation is not straightforward. In fact, replacing each clause with the corresponding set of resultants in the same unit, which might appear the obvious choice, leads to unsoundness in some cases.

Example 3.1 Let P be the evolving program:

$$\begin{array}{ll} \mathbf{unit}(m_1, \mathbf{parent}(top)) : & \mathbf{unit}(m_2, \mathbf{parent}(m_1)) : \\ p : -q. & q. \end{array}$$

By partially evaluating P wrt the goal p in the context $[m_2, m_1]$ if this assignment function is assumed, we get the following program

$$\begin{array}{ll} \mathbf{unit}(m_1, \mathbf{parent}(top)) : & \mathbf{unit}(m_2, \mathbf{parent}(m_1)) : \\ p. & q. \end{array}$$

The transformation is unsound: in fact, in the context $[m_1]$, which is a sub-context of the initial one, the goal p fails in P and succeeds in P' .

Here the problem is that, while in P the body of the clause $p : -q$ needs the definition of q in m_2 to succeed, the corresponding resultant in P' succeeds in sub-context $[m_1]$ without even calling q . In other words, while in P the *deduction context* for $p : -q$ is $[m_2, m_1]$, in P' the *deduction context* for the corresponding resultant p is $[m_1]$.

On the contrary, in order that the transformation be sound, we should ensure that the deduction context for the clause and the corresponding resultant be kept unchanged.

More in general, given a goal in a context C and the corresponding partial tree, for each resultant R_i in the tree, we have first to identify the *deduction context* $\mathcal{Dc}(R_i)$, which is the *minimal sub-context of C , needed to derive R_i* . Then, in order to guarantee the consistency of the deduction contexts, R_i is to be assigned to the *head* unit of $\mathcal{Dc}(R_i)$.

A formal definition of an assignment function satisfying these requirements needs introducing some preliminary notation and definitions. We first consider a restricted class of structured programs in which no dynamic context switch occurs during the evaluation of a query. The problem of dynamic context switches during PE will be considered in section 4.1. Let:

- $\langle G, C \rangle$ denote a *c-atom*, where G is an atomic formula and C a context,
- a *c-goal* be a conjunction of c-atoms

- $\langle (g_1, \dots, g_n), C \rangle$ be a shorthand for $\langle g_1, C \rangle, \dots, \langle g_n, C \rangle$.

We give an extended notion of derivability, well-suited for computing the deduction contexts associated to the resultants.

Definition 3.4 (Matching Context Set.) Let G be an atomic goal. Then the matching context set for $\langle G, C \rangle$ is given by:

$$\mathcal{Mcs}(G, C) = \{C' : C' \sqsubseteq C \wedge |head(C')| \text{ contains a clause for } G\}$$

Each element of $\mathcal{Mcs}(G, C)$ is therefore a sub-context of C which contains a clause for G in its head unit.

Definition 3.5 (Derivability) Let P be a program, CG be the c-goal

$$\langle g_1, c_1 \rangle, \dots, \langle g_i, c_i \rangle, \dots, \langle g_n, c_n \rangle$$

and let $\langle g_i, c_i \rangle$ the selected c-atom. We say that the new c-goal CG' is derived from CG and P via the substitution σ , the clause Cl and the context L_C , if the following conditions hold:

$$\begin{aligned} Cl &= h : -b_1, \dots, b_m \text{ is the selected clause in a unit } u \text{ of } c_i \\ \sigma &= mgu(h, g_i) \\ CG' &= [\langle g_1, c_1 \rangle, \dots, \langle g_{i-1}, c_{i-1} \rangle, \langle (b_1, \dots, b_m), C' \rangle, \langle g_{i+1}, c_{i+1} \rangle, \dots, \langle g_n, c_n \rangle] \sigma \\ L_C &\in \mathcal{Mcs}(G, c_i) \text{ is the matching context corresponding to } Cl (u = head(L_C)) \end{aligned}$$

The new context C' is computed according to the inference rules (see appendix). Namely:

$$C' = \begin{cases} L_C & \text{if the system is conservative} \\ c_i & \text{if the system is evolving} \end{cases}$$

We will henceforth denote a derivation step from a c-goal CG to a c-goal CG' , by $CG \vdash_{L_C} CG'$ where, for the sake of simplicity, we omit mentioning the substitution.

A *C-SLD Derivation* will be, accordingly, a (finite or infinite) sequence of derivation steps $CG_0 \vdash_{c_1} \dots \vdash_{c_j} CG_j \dots$

Finally we generalize the notion of resultant and we introduce a constructive definition of *deduction context*.

Definition 3.6 (c-resultant) Let G_0 be a goal and C_0 a context. Given a finite C-SLD derivation:

$$\langle G_0, C_0 \rangle \vdash_{c_1} \dots \vdash_{c_j} \langle g_1, c_k \rangle, \dots, \langle g_i, c_i \rangle, \dots, \langle g_n, c_n \rangle$$

the c-resultant corresponding to the resultant $G_0 : -g_1, \dots, g_n$, is given by:

$$G_0 : -\langle g_1, c_1 \rangle, \dots, \langle g_i, c_i \rangle, \dots, \langle g_n, c_n \rangle.$$

Proposition 1 (Deduction context) Given a finite C-SLD derivation:

$$\langle G_0, C_0 \rangle \vdash_{c_1} \dots \vdash_{c_j} \langle G_j, C_j \rangle$$

the deduction context $\mathcal{Dc}(R)$ for the c-resultant $R = G_0 : -\langle G_j, C_j \rangle$ is the maximum element of the set $\{c_j, \dots, c_1\}$ which is also a sub-context of C_0 .

Proof. By construction from the definition of C-SLD derivation.

We are now ready to introduce the *assignment function*:

Definition 3.7 (Assignment function: \mathcal{F}_{ass}) Let T be a finite C-SLD tree. Let CR be a c-resultant in T and $\mathcal{D}_c(CR) = [u_k, \dots, u_1]$. Let finally R be the corresponding resultant. Then

$$\mathcal{F}_{ass}(R) = u_k.$$

Example 3.2 In example 3.1, if this assignment function is assumed, we get the following (finite) C-SLD tree. Each branch is marked by the label contexts used in the derivation.

$$\begin{array}{c} \langle p, [m_2, m_1] \rangle \\ | \\ [m_1] \\ | \\ \langle q, [m_2, m_1] \rangle \\ | \\ [m_2, m_1] \\ | \\ \square \end{array}$$

$R = p.$ is the only resultant; its deduction context is $\mathcal{D}_c(R) = [m_2, m_1]$. Therefore the new clause produced by $PE(P, p, [m_2, m_1])$ is assigned to m_2 and the previous definition for p is dropped from m_1 . Accordingly the following *sound* program P' is produced,

$$\begin{array}{l} \mathbf{unit}(m_1, \mathbf{parent}(top)) : \quad \mathbf{unit}(m_2, \mathbf{parent}(m_2)) : \\ \quad p. \\ \quad q. \end{array}$$

3.2 Determining the Context for the Residuals

The choice of a well-defined assignment function only partially answers the questions about the soundness and completeness of the transformation. Once the new configuration of the program has been established, there is still the problem of the residuals occurring in the bodies of the resultants created by unfolding. Namely, for each resultant of the form $h : -b_1, \dots, b_n$, it must be ensured that each of the b_i s be evaluated in *equivalent* contexts before and after the transformation. As a matter of fact, two contexts C and C' are equivalent if they have the same success set.

Example 3.3 In a conservative system, let P be the following program:

$$\begin{array}{l} \mathbf{unit}(u_1, \mathbf{parent}(top)) : \quad \mathbf{unit}(u_2, \mathbf{parent}(u_1)) : \quad \mathbf{unit}(u_3, \mathbf{parent}(u_2)) : \\ q : -r. \quad \quad \quad r. \quad \quad \quad p : -q. \end{array}$$

Let us consider a $PE(P, p, [u_3, u_2, u_1])$. A finite C-SLD tree for $\langle p, [u_3, u_2, u_1] \rangle$ is:

$$\begin{array}{c} \langle p, [u_3, u_2, u_1] \rangle \\ | \\ [u_3, u_2, u_1] \\ | \\ \langle q, [u_3, u_2, u_1] \rangle \\ | \\ [u_1] \\ | \\ \langle r, [u_1] \rangle \end{array}$$

Then the resulting program P' is given by:

$$\begin{array}{l} \mathbf{unit}(u_1, \mathbf{parent}(top)) : \quad \mathbf{unit}(u_2, \mathbf{parent}(u_1)) : \quad \mathbf{unit}(u_3, \mathbf{parent}(u_2)) : \\ q : -r. \quad \quad \quad r. \quad \quad \quad p : -r. \end{array}$$

Again, the transformation is unsound: r is evaluated in $[u_3, u_2, u_1]$ in P' and in $[u_1]$ in P and these two contexts are obviously not equivalent.

Finding the correct context to be associated with each atom in the body of the resultants is straightforward in our scheme since it can be determined by a direct inspection of the partial search trees. The switch operator can be explicitly used to enforce the right context.

Definition 3.8 (Residuals) Let $CR = G : -\langle g_i, C_i \rangle$ be a c-resultant for the c-goal $\langle G_0, C_0 \rangle$. Then, the corresponding resultant R is determined depending of the the relationship between $\mathcal{D}c(CR)$ and C_i . Namely,

(i) if $C_i = \mathcal{D}c(CR)$, then $R = G : -g_i$.

(ii) if $C_i \sqsubset \mathcal{D}c(CR)$, i.e. $\mathcal{D}c(CR) = [u_N, \dots, u_j, \dots, u_1]$ and $C_i = [u_j, \dots, u_1]$, then:

$$R = G : -u_j : g_i$$

In both cases R is assigned to the head unit of $\mathcal{D}c(CR)$. The general case of a resultant whose body contains more than one atom can be simply obtained by the previous cases, by simply applying (i) and (ii) to each atom in the body.

Example 3.4 In example 3.3 condition (ii) occurred. Accordingly the transformation produces the following program:

$$\begin{array}{lll} \mathit{unit}(u_1, \mathit{parent}(top)) : & \mathit{unit}(u_2, \mathit{parent}(u_1)) : & \mathit{unit}(u_3, \mathit{parent}(u_2)) : \\ q : -r. & r. & p : -u_1 : r. \end{array}$$

Notice that now, in both the source and the transformed program, r is evaluated in $[u_1]$.

4 Soundness and Completeness

As already noticed, what we expect from the transformation is that it preserves the equivalence between the source and the resulting programs with respect to all those computations in which (instances of) G are evaluated in sub-contexts of C . This is in fact what the following lemma establishes.

Lemma 4.1 Given a program P , a goal $G = g(T)$ and a context C , let P' be a Partial Evaluation of P wrt G and C .

- P' is sound wrt P if each occurrence of g in P' is evaluated in a context $C' \sqsubseteq C$.
- P' is complete if each occurrence of g in P' is an instance of $g(T)$ and is evaluated in a context $C' \sqsubseteq C$.

Proof.

If the occurrences of g are always evaluated in contexts which share no units with C , the proposition is trivially true since these contexts are not affected by the transformation. Otherwise, notice that the Partial Evaluation of a program P with respect to a goal G can be seen as the iterative process:

$$P \equiv P^0 \rightsquigarrow P^1 \rightsquigarrow \dots \rightsquigarrow P^n \rightsquigarrow \dots$$

where each P^i corresponds to a partial search tree T^i for P and G and where P^{i+1} is obtained from P^i by an unfolding step on one of the resultant of T^i .

The proof that, for any given n , P^n is sound and complete wrt $P \equiv P^0$ can be then obtained by showing that each single unfolding step defines semantic-preserving transformation.

Correspondingly, in the structuring framework we have devised, if C is the initial context, the unfolding process maps into a sequence of transformations, over C , of the form:

$$C \equiv C^0 \rightsquigarrow C^1 \rightsquigarrow \dots \rightsquigarrow C^n \rightsquigarrow \dots$$

Then, if we denote by $C|_i$ the sub-context of C of length i , the soundness and completeness of Partial Evaluation in sub-contexts of C can be proved by showing that, given a query Q and a substitution σ , the following relation holds:

$$\forall n, i \quad C^n|_i \vdash_\sigma g \iff C^{n+1}|_i \vdash_\sigma g$$

We use here the notation $ctx \vdash_\sigma$ as a shorthand for the extended notation $ctx \vdash_\sigma^{L_C}$ where the label context is also mentioned. Let

- g be an instance of G
- T^n be the partial tree obtained by the application of n steps of unfolding;
- C^n be the associated context;
- $R = G\theta : -b$ a resultant in T^n ;
- b the node selected for the next unfolding step;
- T^{n+1} be the partial tree obtained by unfolding b ;
- S be the set of resultants derived from R

We assume, for the sake of simplicity, that the set S be the singleton $\{R_j = G\theta\sigma_j : -B_j\}$

Now, let's consider the structure of the contexts $C^n|_i$ and $C^{n+1}|_i$.

First notice that, from the definition of \mathcal{F}_{ass} , it follows that $R \in head(\mathcal{Dc}(R))$ and $R_j \in head(\mathcal{Dc}(R_j))$. Furthermore, by the construction of the deduction context (proposition 1), it follows that $\mathcal{Dc}(R) \sqsubseteq \mathcal{Dc}(R_j)$.

Therefore, if we denote by $\|\mathcal{Dc}(R)\|$ the length of $\mathcal{Dc}(R)$, since no change occurs in sub-contexts of $\mathcal{Dc}(R)$, it immediately follows that

$$\forall i < \|\mathcal{Dc}(R)\| \quad C^n|_i \equiv C^{n+1}|_i \tag{1}$$

For indexes i such that $i \geq \|\mathcal{Dc}(R)\|$ it is worth distinguishing the cases of evolving and conservative systems explicitly.

[Evolving Systems].

$$\text{Let } i \text{ be such that } \|\mathcal{Dc}(R)\| \leq i < \|\mathcal{Dc}(R_j)\| \tag{2}$$

(Soundness). First notice that $R \notin C^{n+1}|_i$ since R has been erased from $head(\mathcal{Dc}(R))$. Furthermore, by definition of \mathcal{F}_{ass} and by $i < \|\mathcal{Dc}(R_j)\|$, it follows that $R_j \notin C^{n+1}|_i$. Therefore, we get

$$C^{n+1}|_i = C^n|_i \setminus_{head(\mathcal{Dc}(R))} R$$

where $C \setminus_u R$ denotes the context obtained from C by erasing R from the unit u . Finally

$$C^{n+1}|_i \vdash_\sigma g \iff C^n|_i \setminus_{head(\mathcal{Dc}(R))} R \vdash_\sigma g \Rightarrow C^n|_i \vdash_\sigma g$$

(Completeness). Let's assume that $C^n|_i \vdash_\sigma g$. By contradiction, if the deduction of G in $C^n|_i$ used the clause $R = G\theta : -b$, then it should also use the definition for b . Such definition,

by construction, is contained in $head(\mathcal{D}c(R_j))$, but this contradicts the assumption $i < \|\mathcal{D}c(R_j)\|$. Therefore we get

$$C^n|_i \vdash_\sigma g \Rightarrow C^n|_i \setminus_{head(\mathcal{D}c(R))} R \vdash_\sigma g \iff C^{n+1}|_i \vdash_\sigma g$$

$$\text{Let finally } i \text{ be such that } i \geq \|\mathcal{D}c(R_j)\| \quad (3)$$

In this case we get $C^{n+1}|_i = C^n|_i \setminus_{head(\mathcal{D}c(R))} R \oplus_{head(\mathcal{D}c(R_j))} R_j$ where $C \oplus_u R$ denotes the context obtained from C by adding R to the unit u .

If $C^{n+1}|_i \vdash_\sigma g$, then there exists a c-derivation of the form

$$\langle C^{n+1}|_i, g \rangle \vdash_\phi \langle C^{n+1}|_i, B_j \rangle$$

At the same time, since R_j is contained from R by unfolding, and g is an instance of the initial goal, a c-derivation of the form

$$\langle C^n|_i, g \rangle \vdash_\theta \langle C^n|_i, b \rangle \vdash_\sigma \langle C^n|_i, B_j \rangle$$

exists in $C^n|_i$ with $\phi = \theta\sigma$.

If B_j is a recursive call for g , we can repeat the above construction. Otherwise, since the only differences between $C^n|_i$ and $C^{n+1}|_i$ concern the definitions for g , it immediately follows that

$$C^n|_i \vdash_\delta B_j \iff C^{n+1}|_i \vdash_\delta B_j$$

[Conservative Systems].

In conservative systems, by construction, it always holds that $\mathcal{D}c(R) = \mathcal{D}c(R_j)$. Then, let's consider the case

$$i \geq \|\mathcal{D}c(R_j)\| \quad (4)$$

As in the case of evolving systems, if $C^{n+1}|_i \vdash_\sigma g$, then there exists a c-derivation of the form:

$$\langle C^{n+1}|_i, g \rangle \vdash_\phi \langle \mathcal{D}c(R_j), B_j \rangle$$

At the same time, since R_j derives from R by unfolding, and g is an instance of the initial goal, a c-derivation of the form

$$\langle C^n|_i, g \rangle \vdash_\theta \langle \mathcal{D}c(R), b \rangle \vdash_\sigma \langle L_C(b), B_j \rangle$$

exists in $C^n|_i$, with $\phi = \theta\sigma$ and $L_C(b)$ is the label context for b .

If $L_C(b) \sqsubset \mathcal{D}c(R)$ the two c-derivations are not equivalent. But in this case, as stated in definition 3.8, if $u = head(L_C(b))$, R_j is actually defined as $R_j = G\theta\sigma_j : -u_j : B_j$. Accordingly, also in $C^{n+1}|_i$ we actually get the c-derivation

$$\langle C^{n+1}|_i, g \rangle \vdash_\phi \langle \mathcal{D}c(R_j), u : B_j \rangle \vdash_\sigma \langle L_C(b) : B_j \rangle$$

Again the two derivations are equivalent. This concludes the proof.

On the basis of this result, we can prove some further relevant properties of our transformation scheme. By simply extending the results of the lemma, Partial Evaluation turn out to be always sound for statically configured system.

Proposition 2 (Soundness) Let P be a program, and P' a Partial Evaluation of P wrt a goal and a context C . Then, for any given goal g , each computed answer for g and P' is a computed answer for g and P .

Proof (sketched)

From lemma 4.1, the proposition trivially holds for any goal G in sub-contexts of C . It also holds in all those contexts which share no units with C . Let's then consider the case of a context $\mathcal{C} \not\sqsubseteq C$. Since the program is statically configured, \mathcal{C} can only be of the form $u_n \dots u_1.C|_i$ where $C|_i$ is an arbitrary sub-context of C .

Let's then consider the simpler case $\mathcal{C} = u.C|_i$. We prove that each single unfolding step is a *sound* transformation on \mathcal{C} .

If we denote by \mathcal{C}^n the result of applying n unfolding steps on \mathcal{C} ($\mathcal{C}^n = u.C^n|_i$), what we have to prove is that:

$$\mathcal{C}^{n+1} \vdash_{\theta} g \Rightarrow \mathcal{C}^n \vdash_{\theta} g$$

If $\mathcal{C}^{n+1} \vdash_{\theta} g$, then either the whole derivation takes place in $[u]$, or it also involves $\mathcal{C}^{n+1}|_i$. In the first case the proof is straightforward. Otherwise, let $R_j \equiv G\theta\sigma_j : -B_j$ be the resultant obtained at the $n+1$ step by unfolding the previous resultant $R \equiv G\theta : -b$. As shown in lemma 4.1, if R_j is eventually selected in the derivation of G in \mathcal{C}^{n+1} , then an equivalent derivation involving R is obtained in \mathcal{C}^n .

Under the closedness condition stated in [?], for conservative systems, the proof of proposition 2 can be performed backwards thus proving the converse implication and therefore, the completeness of the transformation.

Proposition 3 (Completeness for Conservative Systems) Let P be a program, and P' a Partial Evaluation of P wrt a goal $G = g(T)$ and a context C . Then, given a goal G' , each computed answer for G' and P is a computed answer for G' and P' provided that $P \cup G'$ is $\{g(T)\}$ -closed.

Proof (sketched)

Again, from lemma 4.1, the proposition trivially holds for any goal G and sub-contexts of C . It also holds for all those contexts which share no units with C .

Let's then consider the case of a context $\mathcal{C} \not\sqsubseteq C$. Again let $\mathcal{C} = u.C|_i$ where $C|_i$ is an arbitrary sub-context of C . We prove that each single unfolding step is a *complete* transformation on \mathcal{C} .

If we denote by \mathcal{C}^n the result of applying n unfolding steps on \mathcal{C} ($\mathcal{C}^n = u.C^n|_i$), what we have to prove is that:

$$\mathcal{C}^n \vdash_{\theta} g \Rightarrow \mathcal{C}^{n+1} \vdash_{\theta} g$$

Let $R_j \equiv G\theta\sigma_j : -B_j$ be the resultant obtained at the $n+1$ step by unfolding the previous resultant $R \equiv G\theta : -b$. Therefore a clause $b' : -B_j$ must exist in $\mathcal{D}c(R)$, with $\sigma_j = mgu(b, b')$. Then the evaluation of b in \mathcal{C}^n actually takes place in the sub-context $\mathcal{D}c(R)$, no matter which units have been stacked on top of $\mathcal{C}^n|_i$ and accordingly, the only selected clause for b will be $b' : -B_j$ which is just the clause used to derive R_j . Therefore the proof immediately follows from lemma 4.1.

The result of proposition 3 cannot be extended to evolving systems, as shown below.

Example 4.1 Let $P \equiv P^0$ be the following evolving program:

$$\begin{array}{lll} \mathbf{unit}(u_1, \mathbf{parent}(top)) : & \mathbf{unit}(u_2, \mathbf{parent}(u_1)) : & \mathbf{unit}(u_3, \mathbf{parent}(u_2)) : \\ p(X) : -q(X). & r(1). & q(2). \\ q(X) : -r(X). & & \end{array}$$

Let $C = [u_2, u_1]$ be the initial context and $G = p(X)$ the initial goal. Then by a single unfolding step we obtain P^1 :

$$\begin{array}{lll} \mathbf{unit}(u_1, \mathbf{parent}(top)) : & \mathbf{unit}(u_2, \mathbf{parent}(u_1)) : & \mathbf{unit}(u_3, \mathbf{parent}(u_2)) : \\ p(X) : -r(X). & r(1). & q(2). \\ q(X) : -r(X). & & \end{array}$$

Now consider the context $\mathcal{C} = [u_3, u_2, u_1]$; the transformation is not complete in \mathcal{C} . In fact $\mathcal{C}^0 \vdash p(2)$ but $\mathcal{C}^1 \not\vdash p(2)$

For evolving systems, it is therefore necessary to impose a stronger *closedness* condition for completeness. This condition is, in fact, a natural extension of that given for Horn Clause Logic in [?] to consider the notion of context.

Definition 4.1 ($\langle G, C \rangle$ -closedness) Let P be a program, $\langle G, C \rangle$ a c-atom, with $G = g(T)$.

- P is $\langle G, C \rangle$ -closed if each occurrence of g in P' is an instance of $g(T)$ and is evaluated in a context $C' \sqsubseteq C$.

Now we are ready to state the following:

Proposition 4 (Completeness in Evolving Systems) Let P be a program, and P' a Partial Evaluation of P wrt a goal G and a context C . Then, P' is complete with respect to P if P' is $\langle G, C \rangle$ -closed.

4.1 Dealing with “:” During Partial Evaluation

Extending the Partial Evaluation scheme we have devised so far to include dynamic context switch is indeed quite straightforward. The only remark concerns the computation of the deduction context for the residuals. In fact, when dynamic context switch is considered, a unique maximum element for the label contexts along each derivation path might not exist. Furthermore, if the construction defined in proposition 1 is assumed, some resultant may get assigned to units which do not belong to the initial context.

Example 4.2 Let us consider the following program :

$$\begin{array}{ll} \mathbf{unit}(u_1, \mathbf{parent}(top)) : & \mathbf{unit}(u_2, \mathbf{parent}(u_1)) : \\ p : -u_2 : q. & q. \end{array}$$

A Partial Evaluation of p in $[u_1]$ produces the following C-SLD tree:

$$\begin{array}{c} \langle p, [u_1] \rangle \\ | \\ [u_1] \\ | \\ \langle u_2 : q, [u_1] \rangle \\ | \\ \langle q, [u_2, u_1] \rangle \\ | \\ [u_2, u_1] \\ | \\ \square \end{array}$$

Accordingly, the resultant p is assigned to u_2 . This, in turn, yields uncompleteness since the derivation $[u_1] \vdash p$ succeeds in P , while it fails in P' .

Therefore, the computation of the deduction context needs restating as follows:

Proposition 5 (Deduction context revised) Given a C-SLD derivation:

$$\langle G_0, C_0 \rangle \vdash_{c_1} \dots \vdash_{c_j} \langle G_j, C_j \rangle$$

the deduction context $\mathcal{Dc}(R)$ for the c-resultant $R = G_0 : -\langle G_j, C_j \rangle$ is the maximum element of the set $\{c_k, \dots, c_1 \mid k \leq j\}$ where $\{c_k, \dots, c_1\}$ is the set of label contexts which occur before the first context switch in the derivation.

Finally, the construction of the residuals in the body of the resultants (definition 3.8), immediately extends to the case $C_i \not\subseteq \mathcal{Dc}(CR)$. Formally:

Definition 4.2 (Residuals) Let $CR = G : -\langle g_i, C_i \rangle$ be a c-resultant for the c-goal $\langle G_0, C_0 \rangle$. Then, the corresponding resultant R is determined depending of the the relationship between $\mathcal{Dc}(CR)$ and C_i . Namely,

(i) if $C_i = \mathcal{Dc}(CR)$, then $R = G : -g_i$.

(ii) if $C_i \not\subseteq \mathcal{Dc}(CR)$ and $C_i = [u_j, \dots, u_1]$, then:

$$R = G : -u_j : g_i$$

5 Applications

For the class of systems we have considered, the transformation technique described in the previous sections provides an effective framework for some relevant applications.

First notice that the closedness condition simply corresponds to a static check on the inheritance chains established in the partially evaluated program. It is worth mentioning that this wouldn't be possible for dynamically configured systems due to the possibility of dynamically associating units with contexts.

A further important remark concerns evolving systems: in this case the completeness of Partial Evaluation is automatically guaranteed if the choice of the initial context corresponds to a whole chain in the inheritance tree. If this is the case, the resulting program is complete-closed by definition. This provides an effective use of Partial Evaluation in inheritance-based systems. Given the whole inheritance tree for a program P a complete transformation is obtained by specializing P with respect to a goal G and to the inheritance chain corresponding to all the branches of this tree.

If $\{C_1, \dots, C_n\}$ are the contexts corresponding to these branches, we get the set of Partial Evaluations $\{PE(G, C_1), \dots, PE(G, C_n)\}$. The new program is then obtained by deleting all the original definitions for G , and adding the new ones. This yields to the extended notion of Partial Evaluation with respect to a goal and a set of contexts. Namely,

$$PE(P, G, \{C_1, \dots, C_n\}) = P' \text{ such that : } \mathcal{R}(P') = (\mathcal{R}(P) \setminus \mathcal{D}(C, G)) \cup (\cup_{i \in [1, \dots, n]} PE(G, C_i))$$

Example 5.1 Let us consider the inheritance scheme given in example 2.5. Suppose we are interested in specializing the program with respect to the goal $\text{mode}(X)$. We have here three branches inheritance chains:

$$\text{top} \rightarrow \text{animal} \rightarrow \text{bird} \rightarrow \text{tweety}$$

$$\text{top} \rightarrow \text{animal} \rightarrow \text{horse}$$

$$\text{top} \rightarrow \text{animal} \rightarrow \text{human}$$

A Partial Evaluation with respect to the goal $\text{mode}(X)$ and the single branch

$$[\text{tweety}, \text{bird}, \text{animal}, \text{top}]$$

produces the transformed program:

$$\begin{aligned}
\mathbf{unit}(animal, \mathbf{parent}(top)) &: & [mode(walk). \\
\mathbf{unit}(bird, \mathbf{parent}(animal)) &: & [mode(fly). \\
& & mode(run). \\
& & no_of_Legs(2). \\
& & covering(feather). \\
\mathbf{unit}(horse, \mathbf{parent}(animal)) &: & [no_of_Legs(4). \\
\mathbf{unit}(human, \mathbf{parent}(animal)) &: & [no_of_Legs(2). \\
\mathbf{unit}(tweety, \mathbf{parent}(bird)) &: & [no_of_wings(2).
\end{aligned}$$

Notice that the goal $human : mode(run)$ fails now while it succeeded in the original program since $[animal, human]$ is not a subcontext of $[tweety, bird, animal, top]$. By converse, if we take into account all the branches, the union of all the resulting Partial Evaluations yields the following complete program:

$$\begin{aligned}
\mathbf{unit}(animal, \mathbf{parent}(top)) &: & [mode(walk). \\
\mathbf{unit}(bird, \mathbf{parent}(animal)) &: & [mode(fly). \\
& & mode(run). \\
& & no_of_Legs(2).covering(feather). \\
\mathbf{unit}(horse, \mathbf{parent}(animal)) &: & [mode(gallop). \\
& & no_of_Legs(4). \\
\mathbf{unit}(human, \mathbf{parent}(animal)) &: & [mode(run). \\
\mathbf{unit}(tweety, \mathbf{parent}(bird)) &: & [no_of_wings(2).
\end{aligned}$$

6 Conclusions

In this work we presented a Partial Evaluation scheme that can be suitably applied to different structured logic programming systems. The framework we have devised extends the one presented in [?] for Horn Clause Logic, to capture the notions of units, contexts and contextual evaluation of goals. For the class of systems we have considered the transformation turns out to be always sound; different *closedness* conditions must be satisfied by the partially evaluated program to preserve the completeness.

In particular, for the class of conservative statically configured systems, which represent the traditional, block-based, structured system, these conditions just correspond to those introduced in [?] for Horn Clause Logic.

For evolving systems, if Partial Evaluation is performed with respect to all the leaves of the inheritance tree, these conditions can be still conveniently applied.

Acknowledgements

This work has been partially supported by the “Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo” of CNR under grant n.890004269. We also would like to thank ENIDATA that partially

supported this work.

References

- [Bac88] H. Bacha. Meta-prolog design and implementation. In *Proc. 5th Int. Conf. and Symp. on Logic Programming*, 1988. Seattle, MIT press.
- [BLM89] A. Brogi, E. Lamma, and P. Mello. Structuring logic programs: A unifying framework and its declarative and operational semantics. Technical report, University of Bologna and University of Pisa, 1989.
- [Che87] W. Chen. A theory of modules based on second order logic. In *IEEE Symposium on Logic Programming*, 1987. San Francisco.
- [CLM88] M. Cavaliere, E. Lamma, and P. Mello. An extended prolog machine for dynamic context handling. In *Proc. ECAI88*, 1988. Munich, Pitman Publishing.
- [FH86] K. Fukunaga and S. Hirose. An experience with a prolog-based object-oriented language. In *OOPSLA-86*, 1986. Portland, Oregon.
- [Gal86] H. Gallaire. Merging objects and logic programming: Relational semantics. In *AAAI '86*, 1986.
- [GMR88] L. Giordano, A. Martelli, and G. F. Rossi. Local definitions with static scope rules in logic languages. In *Proceedings of the FGCS Int. Conf.*, 1988. Tokyo.
- [GR84] D.M. Gabbay and N. Reyle. N-prolog: An extension of prolog with hypothetical implications. In *Journal of Logic Programming*, n. 4, 1984. pages 319-355.
- [GS89] H. Gaifman and E. Shapiro. Fully abstract compositional semantics for logic programs. In *Proc. POPL'89*, 1989.
- [KG86] H. Kaufman and A. Grumbach. Multilog: Multiple worlds in logic programming. In *Proc. ECAI86*, 1986. North-Holland.
- [Kom81] H.J Komorowski. A specification of an abstract Prolog machine and its application to Partial Evaluation. In *Linkoping Studies in Science and Technology Dissertation*, 1981.
- [LMN89] E. Lamma, P. Mello, and A. Natali. The design of an abstract machine for efficient implementation of contexts in logic programming. In *Proc. 6th Int. Conf. on. Logic Programming*, Lisbon, The Mit Press 1989.
- [LS87] J. W. Lloyd and J. C. Shepherdson. Partial Evaluation in Logic Programming. Technical Report CS-87-09, Dept. of Comp. Sc. Univ. of Bristol, Dept. of Math., Univ. Walk Bristol, December 1987.
- [McC88a] F. G. McCabe. Logic and objects: Language, application and implementation. In *Ph.D. Thesis*, 1988. University of London.
- [McC88b] L.T. McCarthy. Clausal intuitionistic logic. 1. fixed-point semantics. In *Journal of Logic Programming*, no.5, 1988. pages 1-31.
- [Mel90] P. Mello. Inheritance as combination of Horn Clause Theory. In *Inheritance Hierarchies in Knowledge Representation*, 1990. (to appear) Wiley eds, chapter 14.
- [Mil86] D. Miller. Theory of modules for logic programming. In *Proc. 1986 Symp. on Logic Programming*, 1986. Salt Lake City (USA).

- [MNR89] P. Mello, A. Natali, and C. Ruggieri. Logic programming in a software engineering perspective. In *Proc. North American Conf. on L.P.*, Cleveland U.S. 1989. MIT press.
- [MP89] L. Monteiro and A. Porto. Contextual logic programming. In *Proc. 6th Int. Conf. on L.P.*, Lisbon 1989.
- [MPr85] MPr. Mprolog language reference manual. In *LOGICWARE Inc.*, 1985. Toronto (C).
- [War83] D. H. D. Warren. An Abstract Prolog Instruction Set. In *SRI Technical Note 309*, 1983. SRI International.
- [WZ88] P. Wegner and S. B. Zdonik. Inheritance as an incremental modification mechanism or what like is and isnt like. In *in Proc. ECOOP88, and in LNCS, Vol. 322*, 1988. Springer-Verlag.

Appendix: The Operational Semantics

Let:

- A, A' be atomic goal formulae;
- g be an atomic goal or an extension goal;
- G a conjunction of atomic goals;
- $\epsilon, \theta, \sigma, \delta$ be answer substitutions; ϵ be the empty answer substitution;
- $(\theta\sigma)$ be the composition of the answer substitutions θ and σ ;
- $G\theta$ be the application of the substitution θ to the formula G ;
- $mgu(A, A')$ be the most general unifier of the atomic formulas A and A' ;
- the atomic clauses have the conventional body "true", which always holds;
- $U(P)$ be the set $\{u \mid u \text{ is a unit name}\}$;
- $|u| = \{c \mid c \text{ is a clause in } u\}$;
- $C(P) = \{ctx \mid ctx \text{ is a list of unit names}\}$
- $ctx \vdash_{\theta} g_i$ a top down derivation of g_i from the context ctx with substitution θ

The following set of rules is inspired by those reported in [?]
TRUE:

$$\frac{}{[u_N, \dots, u_1] \vdash_{\epsilon} true}$$

CONJUNCTION:

$$\frac{[u_N, \dots, u_1] \vdash_{\theta} g; [u_N, \dots, u_1] \vdash_{\sigma} G\theta}{[u_N, \dots, u_1] \vdash_{\theta\sigma} (g, G)}$$

ATOMIC GOAL (**Evolving Systems**):

$$\frac{A' : -G \in |u_i|, \theta = mgu(A, A'); [u_N, \dots, u_1] \vdash_{\sigma} G\theta}{[u_N, \dots, u_1] \vdash_{\theta\sigma} A}$$

ATOMIC GOAL (Conservative Systems):

$$\frac{A' : -G \in |u_i|, \theta = mgu(A, A'); [u_i, \dots, u_1] \vdash_{\sigma} G\theta}{[u_N, \dots, u_1] \vdash_{\theta\sigma} A}$$

CONTEXT SWITCH

$$\frac{u \in U(P); ctx = hierarchy(u); [u|ctx] \vdash_{\theta} G}{[u_N, \dots, u_1] \vdash_{\theta} u : G}$$