Baltzer Journals

A Formal Definition and a Sound Implementation of Analogical Reasoning in Logic Programming

Stefania Costantini[1], and Gaetano Aurelio Lanzarone[1]

[1] *Universita' degli Studi di Milano, Dipartimento di Scienze dell'Informazione Via Comelico 39/41, I-20135, Milano*

E-mail: `costanti@imiucca.csi.unimi.it, lanzarone@hermes.mc.dsi.unimi.it`

A formal model of analogy is introduced in the logic programming setting, and an analogical reasoning program (called DIANA, i.e. Declarative Inference by ANAlogy) is developed in accordance with precise procedural and declarative semantics. Given the source and target domains of analogy as two logic programs P_s and P_t, together with a specification S of the analogical correspondence between predicate symbols, atoms involving these symbols are analogically derived from $P = P_s \cup P_t$ given S, which are not derivable from P_s or P_t or $P_s \cup P_t$ alone. In the paper, the requirements of the analogical process are first stated. The declarative semantics of analogy is then given, by defining the least analogical model of P as an extension of the classical semantics of Horn clauses. A procedural semantics is also described, in terms of extension of SLD–resolution. Both semantics rely on implicit analogical axioms defining the kind of analogical reasoning envisaged. The implementation of DIANA has been done in Reflective Prolog, a metalogic programming language previously developed by the authors. It is shown that analogical axioms can be viewed as an instance of reflection axioms used in Reflective Prolog. By exploiting this feature, the implementation of DIANA is argued to be sound w.r.t. the defined semantics. Examples of analogical reasoning in DIANA are also described. By comparison with the AI literature on analogy, it is claimed that this is the first approach which gives a declarative semantics to analogical reasoning, thanks to the possibility of carrying over in this field the basic logic programming concepts.

1 – Introduction

Analogy is based on the assumption that if two situations are similar in some respect, then they may be similar in other respects as well [18]. Thus, an analogy is a mapping of knowledge from a known "source" domain into a novel "target" domain. Analogy can be applied to problem-solving, planning, proving, etc., on the basis of the following kind of inference: knowing that from premises A conclusion B follows, and that A' correspond to A, analogically conclude B'. As first stated by Peirce [14], research on analogy can be divided into two directions. The first one (that we call **replacement-based analogy**) defines analogy as a replacement of the source object with the target object [19], [10], [11], [8]. The other one (that we call **determination-based analogy**) defines analogy as a generalization from one instance [6], [2], [9].

Approaches to analogical reasoning in AI have been concerned mainly with the cognitive and the algorithmic aspects of this kind of reasoning. Most of the implemented analogical reasoners are domain-specific and knowledge-representation sensitive (see [12], [7] and [15] for overviews, comparisons and a collection of papers, respectively).

In this paper, on the contrary, we are interested in the formal characterization of what is analogically concluded from a given representation of the domains and their correspondence. This formal characterization is aimed at soundly and efficiently implementing analogical reasoning on a computer. Logic Programming seems to be a suitable context for such a task, since first formalizations and implementations have already been given for both determination-based analogy [2], [9] and replacement-based analogy [8]. In the approach of [8], given two logic programs which take the role of source and target domains, an analogy is considered as a partial identity between their least models. The partial identity is based on a pairing of ground terms where the paired terms are treated just as a single term, in order to put the two domains together. Predicates of the two programs are assumed to be the same, which determines a one-to-one relation between facts of their least models. A notion of rule transformation is defined which, given a rule in the source program whose premises are in its least model, simulates the analogous rule in the target program. A semantics is outlined (but not fully developed) for characterizing the set of analogical consequences thus defined. A procedural characterization is also given by means of a Prolog metainterpreter.

This paper is concerned with replacement-based analogy, and is an extension and refinement of the approach introduced in [8]. We extend the notion of partial identity to a more general notion of correspondence between the two domains. We also relax the constraint of provability of the premises in the source, and introduce explicit analogy between predicates with different names. All this, as we show by examples, significantly enlarges the applicability of this kind of analogy, though the treatment at the semantic level is of course less immediate. This is in our opinion the first main point of interest of this paper. We also give a complete declarative and procedural semantics for replacement-based analogy in logic programming, as a simple extension of the classical semantics of the Horn clause language [13]. In particular, we introduce a declarative analogical principle in the definition of the least Herbrand model, in its fixpoint characterization, and in the definition of SLD–resolution. This is, to the best of our knowledge, the first complete semantic definition of this kind of reasoning, and the second main point of interest of this paper.

The new semantics is applicable for introducing analogy as a basic reasoning principle not only in the plain Horn clause language, but possibly in all its variations and extensions which are mainly based on variations and extensions of the classical semantics [13]. One of these extensions is the metalogic programming language Reflective Prolog (for short RP), previously developed by the authors [1], [3]. It is in fact immediate to obtain Reflective Prolog with analogy, which further enlarges the range of applicability. It provides, for free, declarative representation and use of metaknowledge and a declarative treatment of negation in analogy. The latter issue has never been considered before with respect to replacement-based analogy (though it is considered in [9] for determination-based analogy).

Reflective Prolog has been also a suitable language for the implementation of this new approach. On the basis of the above-mentioned semantic characterization, we present our model of an analogical reasoner, called DIANA (Declarative Inference by ANAlogy), and discuss its implementation. Analogy being in essence a metatheoretical concept, and the technicalities required being related to treating (atomic) formulas as first-class objects, RP has proved most appropriate to the task. We argue that the stated analogical principle can be viewed as an instance of the kind of reflective principles adopted by RP to allow the user to express metalevel rules to extend object-level theories, and on that basis we maintain that DIANA is a sound implementation of the given semantics. Thus, we propose the explicit treatment of analogy at the metalevel, and the use of techniques of meta-programming in logic to this end as a third point of interest.

The paper is organized as follows. In Section 2 our approach to replacement-based analogy is informally introduced and discussed, also by examples. In Section 3 declarative and procedural semantics are defined. In Section 4, the metalogic programming language Reflective Prolog is briefly introduced, and the possibility and advantages of performing analogical reasoning in RP are discussed. Section 5 describes the implemention in Reflective Prolog of the approach, with attention paid to the related issues of soundness of the resulting analogical reasoning program with respect to the given semantics. In Section 6 we show the advantages of treating analogy at the metalevel, also in terms of possible further extensions to the approach, which naturally arise in the context of DIANA. In Section 7 we conclude with comparisons to related work and open problems.

2 – Informal Description of the Approach

Replacement-based analogy is based on the following principle, due to Winston [19] and reported in [8].

> *Assume that the premises $\beta_1, \ldots, \beta_n$ logically imply α in the source domain. Assume also that the analogous premises $\beta'_1, \ldots, \beta'_n$ hold in the target domain. Then we conclude atom α' in the target which is analogous to α.*

In [8] the source and target domains are chosen to be two Horn clause programs $P1$ and $P2$, with least models $M(P1)$ and $M(P2)$ respectively. Thus, they restate the principle as follows:

> *Assume that the premises $\beta_1, \ldots, \beta_n$ in $M(P1)$ logically imply α in $P1$. Assume also that there exist $\beta'_1, \ldots, \beta'_n$ in $M(P2)$ such that β_i is isomorphic to β'_i for all i. Then we conclude atom α' in $P2$ which is isomorphic to α.*

Note that $\{\beta_1, \ldots, \beta_n\} \subseteq M(P1)$ logically imply α in the Horn program $P1$ if, according to the rules of $P1$, $\beta_1, \ldots, \beta_n$ allow α to be derived.

Isomorphism between β_i and β'_i (which in [8] is called "partial identity") is stated to hold if the predicate symbols are the same, and there exists a one-to-one pairing of the terms they contain. The requirement of partial identity allows to treat the two domains, semantically, as a single one. Notice however that the requirement of $\beta_1, \ldots, \beta_n$ to be in $M(P1)$, i.e. of $\beta_1, \ldots, \beta_n$ to be provable in $P1$, is stronger than Winston's original principle.

Procedurally, analogy is performed by transforming rules in the source domain into analogous rules in the target domain. The partial identity between terms is computed and verified during the inference process. Below is an example of how inference is performed.

Example 1
Consider the following logic programs.
$P1$ *(source domain)*
$f(b,c).$
$m(a,b).$
$p(X,Y):-m(X,Y).$ $(***)$
$g(X,Z):-p(X,Y),f(Y,Z).$ $(*)$

$P2$ *(target domain)*
$p(X1,Y1):-m(X1,Y1).$ $(**)$
$m(a',b').$
$f(b',c').$

The goal $?-g(a',c')$ *is clearly not provable in* $P2$*. It is however provable by analogy. In particular, the interpreter identifies the candidate analogous rule (*), since the predicate symbol is the same and the arguments are unifiable. First, it tries to prove:*
$:-p(X,Y),f(Y,Z).$
in $P1$*, in order to find values for* X*,* Y *and* Z*. The goal succeeds with substitution* $\{X/a,Y/b,Z/c\}$ *since the fact* $m(a,b)$ *implies* $p(a,b)$*. With partial identity* $\{(a,a'),(c,c')\}$*, the interpreter can generate the goal:*
$:-p(a',Y1),f(Y1,c').$
in $P2$*, which succeeds with substitution* $\{Y1/b'\}$*. This extends the partial identity between terms of the two domains to* $\{(a,a'),(b,b'),(c,c')\}$*. Therefore, since* $m(a,b)$ *and* $f(b,c)$ *are in the minimal model of* $P1$*, and they logically imply* $g(a,c)$*, and also* $m(a',b')$ *and* $f(b',c')$ *are in the minimal model of* $P2$*, it is possible to prove by analogy the original goal* $g(a',c')$*. Notice that nested analogy is allowed: if rule (**) were dropped from* $P2$*, the subgoal* $p(a',Y1)$ *could be proved in turn by analogy, using rule (***) in P1.*

In the rest of this section, we illustrate the extensions/generalization that we introduce with respect to the approach of [8].

Extension: *from identity to correspondence*. Instead of requiring a partial identity, i.e. a one-to-one invertible relation between terms of the two domains, we introduce a notion of one-to-many correspondence. Although this requires some extra technicalities at the semantic level, applicability of this kind of analogy is significantly enlarged, as shown by the following example.

Example 2
Consider the following programs P_s *(source domain) and* P_t *(target domain).*

P_s
$kills(X,Y):-hates(X,Y),has\_weapon(X).$ $(\#\#)$
$hates(john,george).$
$has\_weapon(john).$ $(\#)$

P_t
$hates(anne,joe).$
$hates(X,X):-depressed(X).$ $(\#\#\#)$
$has\_weapon(anne).$
$has\_weapon(bill).$
$depressed(bill).$

In the approach of [8], it is possible, as outlined above, to prove by analogy the goal $?-kills(anne, joe)$*, with partial identity* $\{(john, anne), (george, joe)\}$*. We extend the approach in order to be able to prove by analogy also the goal* $?-kills(bill, bill)$*, by allowing the correspondence* $\{(john, bill), (george, bill)\}$*. In a correspondence, an element of the source domain can correspond to only one element of the target domain; on the contrary, two elements of the source domain, like the constants john and george in the example, may correspond to the same element of the target domain. Correspondences will be written like the following:* $\{john//bill, george//bill\}$*.*

Extension: *provability in the source not required*. Winston's original principle does not require the premises $\beta_1, \ldots, \beta_n$ to be provable in the source, but only to logically imply the conclusion. We generalize the approach of [8] by going back to this original formulation. This further enlarges the applicability of the approach. In fact, referring to Example 2, assume that fact (#) is removed from P_s. In the approach of [8], it is no more possible to prove $kills(anne, joe)$ by analogy. Instead, we want to still be able to prove both:
$kills(anne, joe)$ and $kills(bill, bill)$. We do this by the following instantiation of Winston's principle, exemplified for the case of $kills(bill, bill)$.

> *Since* $hates(john, george)$ *and* $has\_weapon(john)$ *logically imply* $kills(john, george)$ *in the source program* P_s*, and since* $hates(bill, bill)$ *and* $has\_weapon(bill)$ *are provable in the target program* P_t*, with correspondence* $\{john//bill, george//bill\}$ *we can conclude by analogy* $kills(bill, bill)$ *in* P_s*.*

It is necessary to make two important observations about the application of Winston's principle.

First, since it is based on logical implication, analogy must be performed always on rules, and not on facts. Analogy on facts has no conceptual basis in this context, and would be highly arbitrary: for instance, from the fact $hates(john, george)$ in the source, it might be concluded that everybody hates everybody else in the target. Consider for instance the attempt to prove in the target program P_t of Example 2 the goal $?-kills(anne, anne)$, which is clearly an unwanted conclusion. Analogy by means of rule (##) leads to the goal $?-hates(anne, anne), has\_weapon(anne)$. The first subgoal is not provable in P_t, and it is not provable by analogy either if analogies on facts are forbidden. Otherwise, with correspondence $\{john//anne, george//anne\}$ the goal would unsoundly succeed.

Second, the pairing of ground terms of the two domains makes the basic difference between analogy, and the simple restatement of rules of the source program in the target program. This is clarified by the following example.

Example 3
In the programs of Example 2, let us assume to copy rule (##) in P_t*. This would clearly allow to prove* $kills(anne, joe)$ *and* $kills(bill, bill)$ *in* P_t*. Then, it would seem that replacement-based analogy reduces to a simple form of program transformation. To see why this is not true, assume instead that rule (##) is formulated in* P_s *in a less general form, for instance:*
$kills(john, Y) :- hates(john, Y), has\_weapon(john).$

In this case, copying this rule in P_t would be of no help for deriving the given goals. Instead, it is easy to see that by replacement-based analogy the goals can be still proved, exactly as shown before. This is due to the concept of correspondence, that for this kind of application is more general than unification.

Generalization: *explicit analogy between predicates*. In the previous examples we have followed [8] in implicitly stating analogy between predicates with the same name in the two domains. Some extra expressivity may be achieved by explicitly stating analogy between predicate symbols.

Example 4
With reference to Example 2, let us assume to add the following facts to P_t:
$hates(marc, bruce)$.
$violent(marc)$.
$beats(john, carl)$.

Let us also assume to explicitly state analogy between predicate symbols of P_s and P_t, by means of a set S of couples of the form $< p_s, p_t >$ where p_s is a predicate symbol appearing in P_s, and p_t a predicate symbol appearing in P_t. In the example, let
$S = \{< kills, kills >, < kills, beats >, < hates, hates >,$
$< has\_weapon, has\_weapon >, < has\_weapon, violent >\}$.
Then, we can prove by analogy the goal $?-beats(marc, bruce)$. This is done by the following steps.

Since the couple $< kills, beats >$ is in S, i.e. the two predicate symbols are stated to be analogous, we may use as a candidate analogous rule clause (##) of P_s. Then, with correspondence $\{X//marc, Y//bruce\}$, and by applying the analogies $< hates, hates >$,$< has\_weapon, violent >$ we can generate the goal $?-hates(marc, bruce), violent(marc)$ in P_t, which obviously succeeds.

In all the examples in the following sections, predicates with the same name in the source and target programs are assumed for simplicity to be in analogy by default, thus S will explicitly contain only the analogies between predicates with different names. The former example shows that analogy between predicates is not necessarily one-to-one.

In Section 3 we provide a complete declarative and procedural semantics for our approach.

3 – Semantics

In the rest of the paper, in particular in definitions and theorems, we make the following assumptions. Let P_s and P_t be Horn clause definite programs, which play the role of source and target domains of the analogy respectively. Let U_{P_s} and U_{P_t} be the Herbrand universes of P_s and P_t, and B_{P_s}, B_{P_t} their Herbrand bases, respectively. Let $pred(P_s)$ and $pred(P_t)$ be the sets of predicate symbols appearing in the alphabet of P_s and P_t respectively.

We assume the alphabet of P_s to contain a number of distinct constant symbols at least equal to the number of distinct variable symbols, possibly by adding some new constants to the alphabet itself. To understand this requirement, assume for instance that the program P_s in Example 2 had no facts: the added constants would serve to build correspondences, on which to perform analogy.

Below we formalize the concept of predicate analogies and the concept of correspondence, informally introduced and discussed in the previous Section.

Definition 5
A set of predicate analogies S is a finite set, of cardinality $\#_\sigma \geq 0$, of couples $< p_{s_i}, p_{t_i} >$, $p_{s_i} \in pred(P_s)$, $p_{t_i} \in pred(P_t)$.

Definition 6
A correspondence σ is a finite set, of cardinality $\#_\sigma \geq 0$, of couples of terms, written in the form $t_i//t'_i$, where: t_i and t'_i are terms of the language of P_s and P_t respectively; and for every $j, k \leq \#_\sigma$, $t_j \neq t_k$ and t_j is not a subterm of t_k.

That is, in a correspondence an element of the source domain can be bound to only one element of the target domain. The converse, as already discussed, does not hold. Basically, a correspondence is similar to a substitution, except that the first element of each couple is any term, rather than a variable. Given two sets of terms $f_1, \ldots, f_n$ and $f'_1, \ldots, f'_n$, we say that there exists a correspondence between them if the set of couples $f_i//f'_i$, $1 \leq i \leq n$, is in fact a correspondence. The correspondence can be applied to a term or atom A in the obvious way, by substituting every occurrence of f_i with f'_i. The result of the application of a correspondence σ to a term/atom A is indicated by $A\sigma$. Correspondences can be composed with substitutions in the obvious way, giving a new correspondence as result. In particular, if θ is a substitution, the correspondence $\sigma' = \sigma\theta$ is obtained by replacing every variable in σ by its assignment in θ, if any.

Defining a semantics for the proposed form of analogical reasoning means characterizing the set of consequences of the program P_t. In fact, this set will consist not only of its Least Herbrand Model $M(P_t)$, but rather of $M(P_t)$ augmented with the set of analogical consequences that can be drawn based on P_s, given S. In the following definitions, for the sake of simplicity and as it is customary in the literature, without loss of generality we will implicitly refer to the instantiated versions of P_s and P_t, i.e. the (possibly infinite) programs obtained by instantiating every variable in each program in all possible ways over ground terms of its Herbrand universe.

Declarative semantics of replacement-based analogy will be defined by applying the concepts introduced in [1], [3], [5], where it is shown that many extensions to the Horn clause language can be modeled with very limited modifications to the standard semantics. This is obtained by implicitly augmenting a program with a denumerable set of new axioms, characterized by an axiom schema $\mathcal{AX}$ called a *Reflection Principle*. In the present context, we take an axiom schema to be of the form:

$$\alpha \leftarrow \beta$$

where α and β stand for Horn clauses.

The following general definitions can be given.

Definition 7
Let P be a Horn clause program, and $I \subseteq B_P$ an interpretation for P. Let $\mathcal{AX}$ be an axiom schema, and $\mathcal{AX}'$ the (possibly denumerable) set of instances of $\mathcal{AX}$ over B_P. I is an $\mathcal{AX}$–Reflective Model for P iff it is a model for $P' = P \cup \mathcal{AX}'$.

Definition 8
Let P be a Horn clause program, and $I \subseteq B_P$ an interpretation for P. Let $\mathcal{AX}$ be an axiom schema. $A \in B_P$ is an $\mathcal{AX}$-Reflective Logical Consequence of P iff for every interpretation $I \subseteq B_P$, I is an $\mathcal{AX}$–Reflective Model for P implies that I is a model for A.

Axioms in $\mathcal{AX}'$ are called *Reflection Axioms*. Reflective Models are clearly models in the usual sense. Therefore, for definite programs there exists the Least $\mathcal{AX}$–Reflective Herbrand Model. It is in general not minimal as a Herbrand model of the original program, but it is minimal with respect to the set of consequences which can be drawn from the program and the reflection axioms.

Given this general framework, in order to define the set of consequences of the program P_t under replacement-based analogical reasoning based on P_s, it suffices to suitably define a schema $\mathcal{AX}$, that we call *Analogical Reflection Principle.*

Definition 9 (Analogical Reflection Principle)
Let A_s, A_t be atoms and B_s, B_t be conjunctions of atoms. Let S be a set of predicate analogies. The Analogical Reflection principle has the form:

$$((A_t \leftarrow B_t) \leftarrow (A_s \leftarrow B_s))$$

where:
$A_s \leftarrow B_s$ *is a non-unit clause in* P_s*;*
$A_s = p_s(t_1, \ldots, t_k)$, $A_t = p_t(t'_1, \ldots, t'_k)$*;*
$t_1, \ldots, t_k$ *are terms in the language of* P_s*, and*
$t'_1, \ldots, t'_k$ *are terms in the language of* P_t $(k \geq 0)$*;*
$< p_s, p_t > \in S$*;*
there exists a correspondence σ *between* $t_1, \ldots, t_k$ *and* $t'_1, \ldots, t'_k$*, and* $B_t = B_s\sigma$*.*

That is, the analogical reflection principle defined above maps a rule of P_s over an analogous rule of P_t, given the analogy between the predicate symbols of the conclusions, and a correspondence between their arguments. The analogical reflection principle defines, according to Definition 7, a set of Analogical Reflective Models for P_t. Since P_t is a definite program, there exists the Least Analogical Reflective Model LAM_{P_t}, which entails: the consequences of P_t; the analogical consequences drawn by means of the analogical reflection principle; and the further consequences obtained from both.

It is now easy to define an extended function T_{PA} which gives LAM_{P_t} a constructive characterization, as well as an extended resolution principle. They will both be composed of the normal case, plus a second case to model the analogical reflection principle.

Definition 10
Let $I \subseteq B_{P_t}$. The mapping $T_{PA} : 2^{B_{P_t}} \rightarrow 2^{B_{P_t}}$ is defined as follows:

$$\begin{array}{rcl} T_{PA}(I) & = & \{A_t \in B_{P_t} : A_t \leftarrow B_1, \ldots, B_n \textit{ is a clause in } P_t, \{B_1, \ldots, B_n\} \subseteq I\} \\ & & \bigcup \\ & & \{A_t \in B_{P_t} : A_s \leftarrow B_1, \ldots, B_n \textit{ is a clause in } P_s, n \geq 0, \\ & & A_s = p_s(t_1, \ldots, t_k), A_t = p_t(t'_1, \ldots, t'_k), < p_s, p_t > \in S, \\ & & \{B_1\sigma, \ldots, B_n\sigma\} \subseteq I \\ & & \textit{ where } \sigma \textit{ is a correspondence between } t_1, \ldots, t_k \textit{ and } t'_1, \ldots, t'_k\} \end{array}$$

The mapping T_{PA} is clearly monotonic, and we have proved the following.

Theorem 11
The mapping T_{PA} is continuous.

Theorem 12
Let $I \subseteq B_{P_t}$. I is an Analogical Reflective Model for P_t iff $T_{PA}(I) \subseteq I$.

Proof
According to Definitions 7 and 9, I is an Analogical Reflective Model for P_t iff

(i) for every clause $A_t \leftarrow B_1, \ldots, B_n$ in P_t, $\{B_1, \ldots, B_n\} \subseteq I$ implies $A_t \in I$, since I is a model of each axiom of P_t, or

(ii) for every clause $A_s \leftarrow B_1, \ldots, B_n$ in P_s, $n \geq 0$, $A_s = p_s(t_1, \ldots, t_k)$, for every $A_t = p_t(t'_1, \ldots, t'_k)$ such that $< p_s, p_t > \in S$, and there exists a correspondence σ between $t_1, \ldots, t_k$ and $t'_1, \ldots, t'_k$, $\{B_1\sigma, \ldots, B_n\sigma\} \subseteq I$ implies $A_t \in I$, since I is a model of the reflection axioms defined by the Analogical Reflection Principle

iff $T_{PA}(I) \subseteq I$. □

Therefore we can apply the result by Van Emden and Kowalski [13] (Theorem 6.5), thus providing a fixpoint characterization of the Least Analogical Reflective Model LAM_{P_t}.

Theorem 13
$LAM_{P_t} = T_{PA} \uparrow \omega = \bigcup_{n<\omega} T_{PA} \uparrow n$ *(where $T_{PA} \uparrow 0 = \emptyset$)*

Example 14
Let P_s and P_t be the programs in Example 2. Below we show how LAM_{P_t} is computed by T_{PA}. For the sake of clarity, for every analogical conclusion (indicated in boldface) we indicate the ground instance of the clause in P_s and the correspondence by which it is obtained.

$T_{PA} \uparrow 0 = \emptyset$
$T_{PA} \uparrow 1 = \{hates(anne, joe), has\_weapon(anne), has\_weapon(bill), depressed(bill)\}$
$T_{PA} \uparrow 2 = \{hates(bill, bill),$ **kills(anne,joe)**$\}$
kills(john,george) ← hates(john,george),has_weapon(john).

σ = {john//anne, george//joe}

$T_{PA} \uparrow 3 =$ {**kills(bill,bill)**}

kills(john,george) ← hates(john,george),has_weapon(john).

σ = {john//bill, george//bill}

LAM_{P_t} is obtained immediately as the union of the former sets.

Definition 15 (Analogical SLD-Resolution)
Let S be a set of predicate analogies.
Let G be a definite goal $\leftarrow A_1, \ldots, A_m, \ldots, A_k$ and C a clause.
Let A_m be the selected atom in G.
G' is derived from G and C using mgu θ or correspondence σ iff one of the following conditions holds:

(i) *C is a clause $A_t \leftarrow B_1, \ldots, B_n$ of P_t,*
θ is the mgu of A_m and A_t,
G' is $\leftarrow (A_1, \ldots, B_1, \ldots, B_n, \ldots, A_k)\theta$

(ii) *C is a non-unit clause $A_s \leftarrow B_1, \ldots, B_n$ of P_s,*
$A_s = p_s(t_1, \ldots, t_k)$, $A_m = p_t(t'_1, \ldots, t'_k)$, $< p_s, p_t > \in S$,
σ is a correspondence between $t_1, \ldots, t_k$ and $t'_1, \ldots, t'_k$,
G' is $\leftarrow A_1, \ldots, (B_1, \ldots, B_n)\sigma, \ldots, A_k$

We shorten "Analogical SLD" as "ASLD". The definitions of ASLD derivation and refutation, success set, answer, computed answer are exactly as usual. Below is the obvious extension of the definition of correct answer.

Definition 16
Let G be a definite goal $\leftarrow A_1, \ldots, A_k$ and θ an answer for $P_t \cup \{G\}$. We say that θ is a correct answer for $P_t \cup \{G\}$ if $\forall((A_1 \wedge \ldots \wedge A_k)\theta)$ is an Analogical Reflective logical consequence of P_t.

Theorem 17 (Soundness of ASLD-Resolution)
Every computed answer for $P_t \cup \{G\}$ is a correct answer for $P_t \cup \{G\}$.

Theorem 18 (Completeness of ASLD-Resolution)
For every correct answer θ for $P_t \cup \{G\}$, there exists a computed answer λ for $P_t \cup \{G\}$ and a substitution γ such that $\theta = \lambda\gamma$.

We do not report here the proofs of the above results because they are quite long and (similarly to the proof of Theorem 2) they are straightforward extensions of the proofs in [13].

Example 19
Let us consider again the programs in Example 2. Below are the steps for proving a goal by analogy by ASLD-Resolution. Clauses of P_s (case (ii)) are in roman.

?-kills(A,B).

C is kills(X,Y):-hates(X,Y),has_weapon(X).
σ_1 *is* $\{X//A, Y//B\}$
?-hates(A,B),has_weapon(A).
C is hates(anne,joe).
θ_1 *is* $\{A/anne,B/joe\}$
?-has_weapon(anne).
C is has_weapon(anne).
θ_2 *is* $\{\}$
A=anne, B=joe
θ *is* $\theta_1\theta_2$*, i.e.* $\{A/anne,B/joe\}$

On backtracking, a similar inference process will produce A = bill, B = bill. Notice that a correspondence between ground terms of the source and target domains on which the analogy can be based according to Wiston's principle may be obtained in the following way: doing the union of the correspondences generated in all the analogical steps (in the example, just σ_1*); then composing the resulting correspondence with the computed answer* θ *(in the example,* $\sigma_1\theta$ *is* $\{X//anne, Y//joe\}$*); and, finally, instantiating the result to any tuple of distinct terms of the source domain (in the example, one of the possibilities is* $\{john//anne,george//joe\}$*).*

The semantics we have just presented is applicable for introducing analogy as a basic reasoning principle not only in the plain Horn clause language, but possibly in all its variations and extensions which are mainly based on variations and extensions of the classical semantics [13]. One of these extensions is the metalogic programming language Reflective Prolog (for short RP), that we shortly recall in the next Section (for a full technical presentation, the reader may refer to [1], [3], [4]).

4 – Reflective Prolog

RP augments pure Horn clauses with the following three basic features. (1) Expressions of the Horn clause language are given names by a full naming relation, thus becoming first-class objects usable in metalevel clauses. (2) Special metalevel clauses (called metaevaluation clauses) can be used in the program to define the distinguished predicates *solve* and *solve_not*, that allow to declaratively extend/restrict the meaning of the other predicates, respectively. In fact, *solve* and *solve_not* take as arguments (names of) atoms (with other predicates as head predicate), and thus they express sentences about relations. 3) A form of logical reflection is embedded in RP, which makes these extensions/restrictions effective, both semantically and procedurally.

The semantics of an RP definite program P is given in terms of Reflective Model of P, obtained as follows (where by $\uparrow A$ we mean the name of the term/atom A).

We first define an Extended Herbrand Interpretation I_E of P as an Herbrand Interpretation of P which satisfies the condition that $solve(\uparrow A)$ belongs to I_E whenever A belongs to I_E. I_E is a subset of the Extended Herbrand Base B_{PE}, which contains the usual terms plus new terms, representing names.

Then, to characterize metalevel negation (rules for *solve_not* specify when the predicates given as argument do not hold) the program P is taken as standing for a program

P* obtained by substituting each clause
$\bar{A} : -B_1, \ldots, B_n$ $\quad (\bar{A} = A$ or $\bar{A} = solve(\uparrow A)$, $A \neq solve\_not(\ldots))$
of P with a clause
$\bar{A} : -B_1, \ldots, B_n, \neg solve\_not(\uparrow A)$
where '$\neg$' is the classical negation of first-order predicate logic.

Last, the following reflection principle $\mathcal{REFL}$ is introduced:

$$\{A \leftarrow solve(\uparrow A) : A \in B_{PE}, A \neq solve\_not(\ldots), A \neq solve(\ldots)\}$$

By means of $\mathcal{REFL}$, solve rules play the role of additional clauses for the other predicates. For instance, given the clauses:
$solve(\uparrow P(X,X)) : -reflexive(\uparrow P), in\_domain(\uparrow P, \uparrow X).$
$in\_domain(\uparrow P, \uparrow X) : -solve(\uparrow P(X,Y)).$
$in\_domain(\uparrow P, \uparrow X) : -solve(\uparrow P(Y,X)).$

the result of $\mathcal{REFL}$ is to add the couples $p(t,t)$, for every binary predicate p which has been declared to be reflexive (by means of a fact in the program) and for every term t in its domain. As before, here $\uparrow A$ denotes the name of the term/atom A, though this is not the concrete notation used in RP.

In a complementary way, metalevel negation allows irreflexivity to be expressed by means of the clause:
$solve\_not(\uparrow P(X,X)) : -irreflexive(\uparrow P).$

This definition rules out the couples $p(t,t)$, for every binary predicate p which has been declared to be irreflexive (by means of a fact in the program) and for every term t.

The program P', consisting of P* plus the reflection axioms generated by $\mathcal{REFL}$, can be considered to be the *extended version* of P, i.e. the program which P – due to the reflection capabilities of RP – stands for. For positive programs (programs without *solve_not* clauses) there exists a Least Reflective Herbrand Model RM_P of a program P. For programs with metalevel negation, among the minimal models of P' we choose the one that never entails A if it entails $solve\_not(\uparrow A)$. Negation as failure can explicitly be expressed by means of metalevel negation.

Procedural semantics for positive RP programs is based on RSLD-Resolution (Reflective SLD-Resolution, formally defined in [3]). Computationally, reflection consists in the ability of dynamically changing the level of the computation. With resolution, this means being able, on certain conditions, to use metalevel clauses to resolve object-level goals, and vice versa. In particular, RSLD-Resolution is able to use clauses with conclusion $solve(\uparrow A)$ to resolve a goal A (*upward reflection*), and vice versa clauses with conclusion A to resolve $solve(\uparrow A)$ (*downward reflection*). Thus, RSLD-Resolution is a form of SLD resolution which can apply both the reflection axioms (implicitly present in the program) and the basic property of Extended Herbrand Interpretations (which entail $solve(\uparrow A)$ in consequence of A). NRSLD-Resolution, for programs with metalevel negation (formally defined in [4]), is based on RSLD-Resolution. In particular, if a goal G succeeds by RSLD-resolution with computed answer θ, then $solve\_not(\uparrow G\theta)$ is attempted: if it succeeds, then G is forced to failure; otherwise, if *solve_not* fails the success of G is confirmed.

As summarized above, declarative and procedural semantics of RP programs are defined as additive extensions to the declarative and procedural semantics of positive Horn clause programs [13]. RP's semantics is also independent of the naming relation. Therefore, in order to introduce the approach to replacement-based analogy presented in this paper in RP, it is sufficient to merge the semantics for analogy presented in Section 3 with the semantics of RP, since the two kinds of extensions do not interfere. In particular, when introducing analogy in RP, both metalevel negation and negation as failure are not influenced at all.

It is worth making some observations about the usability of metalevel negation for ensuring the coherence of analogical reasoning.

Example 20
Let us reconsider the programs of Example 4, listed again below for the sake of clarity.

P_s
$kills(X,Y) :- hates(X,Y), has\_weapon(X).$ (##)
$hates(john, george).$
$has\_weapon(john).$ (#)

P_t
$hates(anne, joe).$
$hates(X,X) :- depressed(X).$
$has\_weapon(anne).$
$has\_weapon(bill).$
$depressed(bill).$
$hates(marc, bruce).$
$violent(marc).$
$beats(john, carl).$

$S = < kills, beats >, < has\_weapon, violent >$. We have seen that we can prove by analogy $? - beats(marc, bruce)$. Assume however to add to P_t the fact $violent(bill)$. An unwanted consequence is that, again by analogy, it is possible to derive the (quite funny) consequence $beats(bill, bill)$. Such unwanted inferences can be prevented by stating negative properties of predicates of P_t by means of metalevel negation. In this case, the definition of irreflexivity given before may help, together with the declaration:
$irreflexive(\uparrow beats)$.

Therefore, by the definition of NRSLD-Resolution summarized above, the conclusion $beats(bill, bill)$ is discarded.

In general, metalevel negation allows to state domain-dependent constraints on the coherence of analogical reasoning by stating negative properties of single predicates, or of their interrelations.

5 – Implementation in Reflective Prolog (DIANA)

Though the analogical reflection principle is, semantically, an axiom schema, from the implementation point of view it can be seen as a metatheoretical statement, based on the treatment of predicate symbols and atoms as first-class objects. Thus Reflective Prolog, which provides these features in a clean and easily usable form, has proved a good language for the implementation of the approach presented in the previous Sections. The resulting analogical reasoning program has been called DIANA, for Declarative Inference by ANAlogy [1].

By examining the definition of ASLD-Resolution, it comes natural to implement a search stategy where the analogical derivation of a goal (case (ii)) is attempted on failure of the standard SLD derivation (case (i)). It is also useful to notice that the definition implicitly allows nested analogical processes, thus permitting complete interleaving of deductive and analogical derivations.

We have used the mechanism of RP reflection, to realize the switch between SLD resolution and analogy-based resolution. Below we show how analogical reflection axioms can be represented by means of RP's reflection axioms.

RP's Reflection axioms:

$$A \leftarrow solve(\uparrow A)$$

and, in the Least Reflective Herbrand Model

$$M \models solve(\uparrow A) \leftarrow M \models A$$

Analogical Reflection axioms:

$$(A_t : -B_{t_1}, \ldots, B_{t_n}) \leftarrow (A_s : -B_{s_1}, \ldots, B_{s_n})$$

where (as shown in detail in Definition 9) the atoms involved are analogous via a suitable correspondence σ.

The analogical reflection axioms can be rewritten in the form:

$$A_t \leftarrow (A_s \leftarrow B_{s_1}, \ldots, B_{s_n}) \wedge B_{t_1}, \ldots, B_{t_n}$$

A computational metatheoretical schema for this formulation of the analogical axioms can be:

$A_t \leftarrow$
  $find\_correspondence(A_t, A_s, Sigma),$
  $(A_s \leftarrow B_{s_1}, \ldots, B_{s_n}),$
  $apply\_correspondence(B_{t_1}, \ldots, B_{t_n}, B_{s_1}, \ldots, B_{s_n}, Sigma),$
  $B_{t_1}, \ldots, B_{t_n}.$

(we assume a mechanism taking care of a variable number n of conjuncts, and a suitable recursive definition of *apply_correspondence*).

Let *theory_clause* be the declarative equivalent of the Prolog predefined predicate *clause* [3]. The above schema can be expressed as a Reflective Prolog program as follows.

$demo(\uparrow A \leftarrow \uparrow B) : -$
  $theory\_clause(\uparrow A \leftarrow \uparrow B), solve(\uparrow B).$
$solve(\uparrow A_t) : -$
  $find\_correspondence(\uparrow A_t, \uparrow A_s, Sigma),$
  $demo(\uparrow A \leftarrow \uparrow B_{s_1}, \ldots, \uparrow B_{s_n}),$
  $apply\_correspondence(\uparrow B_{t_1}, \ldots, \uparrow B_{t_n}, \uparrow B_{s_1}, \ldots, \uparrow B_{s_n}, Sigma),$
  $solve((\uparrow B_{t_1}, \ldots, \uparrow B_{t_n})).$

[1] The implementation has been developed by Lorenza Sbarbaro, during her Degree Thesis in Computer Science, supervised by the authors [16].

(the RP interpreter actually takes care of a variable number of conjuncts). This rule in RP is automatically applied whenever A_t fails at the object level, i.e. analogical reasoning is automatically performed, on failure of goals by SLD-Resolution.

The fact, shown above, that the analogical axioms can be expressed by RP's reflection axioms, allows us to sketch now, though mostly informally, a proof of soundness of the implementation w.r.t. the given semantics. RP's Extended Resolution has been proved correct and complete w.r.t. the Least Reflective Herbrand model [3]. DIANA's success set corresponds to the success set of a suitable RP program, namely, a program P augmented with the analogical reflection axioms expressed by means of the RP's reflection axioms, as described above. Let us assume that the given one is a sound representation (the main point is that the procedures $find\_correspondence$ and $apply\_correspondence$ correctly implement the definitions). Since DIANA's resolution procedure is implemented by means of RP's resolution procedure, on this assumption we can conclude that the Least Analogical Reflective Herbrand model is contained in the Least Reflective Herbrand model, and thus the correctness and completeness results formally proved for the latter carry over to the former.

Clearly, ASLD-Resolution can be implemented directly. Should this be the case, however, proving soundness of an implementation would be problematic.

6 – Extensions

It may be interesting to observe that the RP implementation of replacement-based analogical inference provides some additional features w.r.t. those previously described.

General-Purpose Constraints. RP's metalevel negation allows one to introduce not only domain-dependent constraints on the analogical inference, but also (in the context of DIANA) possible domain independent ones. Constraints are automatically controlled during the inference process, preventing any analogical consequence from being inferred if not consistent with the available information. A combination of positive and negative metarules may also help define constraints on performing analogical steps, depending on the interaction between knowledge represented in the source and target domains. For instance, the general definition of correspondence may need to be limited, according to the compatibility of the domains of analogous predicates.

Example 21

Let us add the following rule to the program P_s of Example 2.

$kills(water\_pollution, fish) : -exceeds\_limit(water\_pollution).$

Clearly, this rule would be of no help for analogically concluding who kills who in the target domain. It only adds inefficiency. The use of this rule in analogy could be prevented by adding to both programs a specification of the type of terms, like e.g.:

$person(george).$

$animal(fish).$

and then, at a metalevel which is common to both programs, prevent correspondences to be stated between terms of incompatible types.

Generalized Correspondence. DIANA is flexible enough to allow a less strict cor-

respondence between source and target domains. This means that analogy may also be applied between domains where knowledge is represented differently, for instance domains where the predicates which are in analogy have different arities. Again, RP is flexible enough for an easy implementation of such a general notion of correspondence.
Reversibility of Source and Target Domains. The implementation of DIANA in RP contains a metalevel rule enabling to reverse the couples of the set S (analogically correspondent predicate symbols). The advantage is that a priori knowledge of source and target roles of the two domains is not required: they will be determined automatically and correctly by DIANA on demand. This is because term correspondences are dynamically determined, while predicate symbol correspondences, given in input, are used independently of the position of the elements in the couples, due to the stated symmetry. In case a circularity occurs in exchanging the domains, termination is controlled by means of RP's built-in loop checks [3].

Example 22
In program P_s of Example 2, let us add the following facts:
$has\_weapon(bob)$.
$depressed(bob)$.

The goal $?-kills(bob,bob)$ in P_s may succeed if P_s takes the role of target domain and P_t that of source domain, thus allowing the analogical use of rule (###).

7 – Related Work and Concluding Remarks

As briefly mentioned in the introduction, most of the work on analogical reasoning reported in the AI literature is concerned with defining plausible heuristics for identifying analogies, and working out methods and algorithms to implement special-purpose systems to experiment with.

The motivation and main aim of the present work being different, as previously discussed, we think that a direct comparison is more appropriate with respect to those approaches proposed in the logic programming context. We are aware of three of them, namely [8], [9] and [17].

[9] is concerned with determination-based reasoning, which is different in essence from the replacement-based approach discussed here, as mentioned in the introduction. The authors of the present paper have examined the connection between determination-based analogy and metalogic programming in [2], showing how determination rules are naturally expressed at the metalevel and usefully used by the reflection mechanism of the metalogic language. The task of formulating a formal analogical principle and declarative semantics, however, was not even attempted in that work.

[17] addresses the problem of the analogical definition of a new Prolog predicate according to a known one, and describes a tool, MARs, performing this task. An operational semantics is given, using forward chaining with inverse resolution. Declarative semantics is not considered, however.

[8] has been discussed in the previous Sections, since it has been a starting point for the present work, which can be regarded as an extension and generalization. It is also the piece of work nearest to the issue of model-theoretic semantics, but still no complete

formal definition is given.

Thus, the present paper is, to the best of our knowledge, the first to provide a complete semantics to replacement-based analogical reasoning.

As anticipated in the introduction, in this paper we have not been concerned with the general problem of how to constrain the analogical reasoning process in order to get only reasonable, sound analogies. Heuristics apt to produce meaningful, useful analogical conclusions, and to discard absurd, irrelevant ones, are certainly needed. We only mention in this regard that the declarative approach, on which our work is based, naturally leads to the explicit and logical representation of such heuristics, whatever they could be, as opposed to the implicit and algorithmic hard-wiring of them in some systems. The advantage is that in principle the explicitly represented heuristics can themselves be formally reasoned about.

We believe that the metalogic framework we have put forward could be powerful and flexible enough to accommodate several such heuristics. For instance, Example 20 of Section 4, though very simple, gives an idea of the use of metalevel negation to block meaningless analogical derivations. However, further work is certainly necessary in order to find out how this might be done in general.

Another important open problem concerns to what extent conclusions obtained by analogy should be used in further proofs. The problem of course is common to any form of plausible reasoning. A possible strategy is that of making these conclusions defeasible in case of inconsistency with deductive or observational sentences, or also according to some predefined metric on the acceptability of conclusions (e.g. certainty factors).

References

[1] S. Costantini and G. A. Lanzarone, A Metalogic Programming Language, in *Logic Programming, Proceedings of the Sixth International Conference*, G. Levi and M. Martelli eds., The MIT Press, Cambridge (Mass.), 1989, 218–233.

[2] S. Costantini and G. A. Lanzarone, Analogical Reasoning in Reflective Prolog, in *Expert Systems in Law*, A. Martino ed., Elsevier Science Publ., Amsterdam, 1992, 133–145.

[3] S. Costantini and G. A. Lanzarone, A Metalogic Programming Approach: Language, Semantics and Applications, Journal of Experimental and Theoretical Artificial Intelligence, 6 (1994), 239–287.

[4] S Costantini and G. A. Lanzarone, Metalevel Negation and Non Monotonic Reasoning, Journal on Methods of Logic in Computer Science 1 (1994), 111–140.

[5] S. Costantini, P. Dell'Acqua, and G.A. Lanzarone, Extending Horn Clause Theories by Reflection Principles, in *Lecture Notes in Artificial Intelligence* n. 838, C. MacNish, D. Pearce and L. M. Pereira eds., Springer-Verlag, Berlin, 1994, 400–413.

[6] T. R. Davies and S. J. Russel, A Logical Approach to Reasoning by Analogy. in *Proceedings of IJCAI 87*, J. Mc Dermott ed., Morgan Kaufmann Publ., Los Altos, 1987, 264–270.

[7] R. P. Hall, Computational Approaches to Analogical Reasoning: A Comparative Analysis. Artificial Intelligence, 39 (1989), 39–120.

[8] M. Haraguchi, and S. Arikawa, Reasoning by Analogy as a Partial Identity between Models. in *Lecture Notes in Computer Science* n. 265, K. P. Jantke ed., Springer-Verlag, Berlin, 1987, 61–87.

[9] N. Iwayama, K. Satoh and J. Arima, A Formalization of Generalization-Based Analogy in General Logic Programs, in *ECAI92, Proceedings of the 10th European Conference on Artificial Intelligence*, B. Neumann ed., John Wiley and Sons Ltd, London, 1992, 129–133.

[10] S. Kedar-Cabelli, Toward a Computational Model of Purpose-Directed Analogy, in [14].

[11] Y. Kodratoff, Using Abductive Recovery of Failed Proofs for Problem Solving by Analogy, in *Proceedings of the 7th International Conference of Machine Learning*, 1990, 295–303.

[12] D. Leishman, An Annotated Bibliography of Works on Analogy, International Journal of Intelligent Systems, 5 (1990), 43–81.

[13] Lloyd J.W. *Foundations of Logic Programming*, Second Extended Edition, Springer-Verlag, Berlin, 1987.

[14] C. S. Peirce, Elements of Logic, in *Collected Papers of Charles Sanders Peirce*, Volume 2, C. Hartshorne and P. Weiss eds., Harvard University Press, Harvard, 1932.

[15] A. Prieditis, *Analogica*, Pitman, London, 1988.

[16] L. Sbarbaro, *Ragionamento per Analogia in Programmazione Logica*, Degree Thesis in Computer Science (in Italian), University of Milano, 1991.

[17] B. Tausend and S. Bell, Analogical Reasoning For Logic Programming , in *Lecture Notes in Artificial Intelligence* n. 482, Y. Kodratoff ed., Springer-Verlag, Berlin,1991, 391–397.

[18] P. Winston, Learning and Reasoning by Analogy, Communication of the Association for Computing Machinery, 23 (1980), 689–703.

[19] P. Winston, Learning New Principles from Precedents and Exercises, Artificial Intelligence, 19 (1982), 321–350.