Integrating Reflection into SLD-resolution
(Extended Abstract)

Jonas Barklund \quad Stefania Costantini
Pierangelo Dell’Acqua* \quad Gaetano A. Lanzarone
Uppsala University \quad Università degli Studi di Milano
Uppsala University
Computing Science Department

Box 311, S-751 05 Uppsala, Sweden
Phone: +46–18–18 25 00
Fax: +46–18–51 19 25

Abstract

In this paper we present an extension of SLD-resolution that allows us to model reflection rules in metalevel architectures. We employ an abstract language, introduce the concept of a name theory for such a language, and present an inference system that is parameterized with a name theory. The proposed mechanism is completely general in such a way that for any metalevel architecture where the procedural semantics is independent of the specific device for unification and naming mechanism, a similar generalization is also possible.

*P. Dell’Acqua has been supported by both Università degli Studi di Milano and Uppsala Univ.
1 Introduction

The need for expressing and using metalevel knowledge has been widely recognized in the AI literature. Aiello et al. [1], for example, argue that metaknowledge and metalevel reasoning are suitable for devising proof strategies in automated deduction systems, for controlling the inference in problem solving, and for increasing the expressive power of knowledge representation languages.

The need to formally represent knowledge and metaknowledge has led to the study of metalevel architectures, where these two kinds of knowledge are explicitly represented.

In order to carry out metalevel reasoning there must be a stated relationship between expressions at different levels, i.e., a naming relation and a reflection principle. The basic features of a metalevel architecture can be summarized in:

1. the naming mechanism, providing names at the metalevel for the expressions of the object-level theory;
2. the metalevel formalization of the object-level properties;
3. the reflection rules, that establish the connection between object-level and metalevel in the inference process.

As the metalanguage is used to formalize properties of the object-level theory, predicates at the metalevel must take expressions of the object-level theory as arguments. This is achieved by means of a naming mechanism that provides names for object-level expressions. The metalevel predicates, expressing properties of the object-level, are related to the object-level itself by reflection rules.

The choice of which reflection rules to employ in a given metalevel architecture depends on the particular type of connection we want to establish between the metalevel and the object-level. For example, reflection can be explicit (i.e., it must be invoked explicitly in the program) or implicit, it can be applied to atoms, to conjunctions of atoms, or even to the whole state of the computation.

The choice of naming depends on the kind of information we need at the metalevel. The more fine the granularity of the naming is, the more information is encoded. Typically, two kinds of names have been employed: atomic names (i.e., names that describe expressions as a whole) and structural descriptive names (i.e., names that reflect the structure of the expression they name). However, names that encode more information are also possible.

In this paper we show how it is possible to characterize the naming mechanism of a metalevel architecture by means of a first order theory and how to integrate reflection rules into SLD-resolution. To this purpose, the paper is organized as follows. The next two sections introduce an abstract metalanguage and a theory that allow us to formally characterize the naming mechanism. Sect. 4 shows how to integrate reflection rules into SLD-resolution. The conclusion briefly mentions the semantic properties of the proposed approach and ends with some concluding remarks.
2 An Abstract Metalanguage

In this section we introduce the basic syntactic features of a metalanguage. We do not refer to any specific approach, and therefore define an abstract language that generalizes the naming device of a number of proposed metalevel architectures. We call the abstract language $L'$. To do this, we first assume the basic features of the language $L_0$ of definite programs, as defined by Lloyd [6], then we extend it in order to include expressions that name expressions of $L'$.

**Alphabet.** The difference in the alphabet of $L$, as compared with the alphabet of $L'$, lies in a distinction between object variables and metavariables, and in the presence of various other metasymbols. If $\alpha$ is an expression in $L'$, then $\alpha^1$ denotes a metasymbol in $L$ that is *intended* as a name of $\alpha$.

1 By $\alpha^2$ we mean some symbol in $L$ that is intended as a name of $\alpha^1$, etc. In general, we write $\alpha^k$, where $k \geq 0$, to mean $\alpha$ if $k = 0$ and some symbol in $L$ that is a name of $\alpha^{k-1}$ if $k > 0$. Given a program $P$ we can identify the alphabet of $P$ as the set of symbols occurring in $P$ extended with the symbols $\alpha^k$ for every $k$, $k \geq 0$, such that for some $l$, $l \geq 0$ and $l \neq k$, a symbol $\alpha^l$ occurs in $P$.

**Terms.** The definition of a term in $L$ extends the definition of a term in $L'$ in order to contain *name terms* that allow the representation of expressions of $L'$ in $L$. Name terms contain metavariables and metasymbols.

**The language of a program.** Given an $L$ program $P$, the *language of $P$* is the subset of $L$ that can be generated from the alphabet of $P$.

---

**Example 1.** The alphabet of Reflective Prolog [5] contains various metasymbols that are intended as names for constants, function symbols and predicate symbols. Since Reflective Prolog employs structural descriptive names, name terms of the language also contain compound name terms. We write them here in the form $[\alpha_0, \alpha_1, \ldots, \alpha_n]$, where $\alpha_0$ is the name of a function symbol or a predicate symbol, and every $\alpha_1, \ldots, \alpha_n$ is a name term. The name of the term $f(a)$, for example, is the compound name term $[f^1, a^1]$, and the name of the atom $p(f(a), b)$ is the compound name term $[p^1, [f^1, a^1], b^2]$. In general a “$k$-th” name of the term $f(a)$ is the compound name term $[f^k, a^k]$.

**Example 2.** 'Log [2] provides two meta-representations for every expression of the language: atomic names and structural descriptive names. The alphabet of 'Log therefore contains a metasymbol for every expression of the language. If $f(a)$ is a term of the language, for example, then the alphabet contains a metasymbol (represented in the concrete syntax as '$f(a)$') that is a name of $f(a)$. Since 'Log employs structural descriptive names, name terms also contain compound name terms.
The definitions of substitutions and variable assignments, as well as the unification algorithm, have to be properly modified to take into account name terms and metavariables. (Note that only name terms may be substituted for metavariables.)

3 Naming as Inference from a Name Theory

A name theory $T$ for the language $L$ is a theory whose alphabet is that of $L$ plus the symbols ‘false’, ‘=’, ‘↑’ and ‘↓’. The expressions of $T$ are defined as follows. Every name term and term of $L$ is a name term and term of $T$, respectively. If $α$ is any term, then $↑α$ is an expression of $T$ (name term). If $β$ is a name term, then $↓β$ is an expression of $T$ (a term). If $α$ and $β$ are terms of $T$, then $α = β$ is an equation. Intuitively, $α = ↑β$ means that $α$ is the name of $β$, and $α = ↓β$ means that $α$ is what $β$ names. If $N$ is a finite set of equations, then $N$ is an expression of $T$.

**Definition 3.** Let $α$ be a metavariable. If $α$ is equated with a variable $β$ by $α = ↑β$ or $β = ↓α$, then $α$ **immediately depends** on $β$. If $α$ is equated with a compound term $β$, then $α$ immediately depends on each variable of $β$. The **depends** relation is the transitive closure of the immediately depends relation.

**Definition 4.** A set of equations $\{α_1 = β_1, \ldots, α_k = β_k\}$ is **safe** if $α_1, \ldots, α_k$ are distinct variables not depending on themselves.

Every name theory contains the axioms that characterize the equality interpretation of ‘=’ and the freeness axioms [3]. In order to consider names of expressions of $L$, a name theory must also contain the axioms that characterize name terms and the symbols ‘↑’ and ‘↓’.

**Example 5.** We can characterize the intended role of every metasymbol $α^{i+1}$ in $L$ with the following axiom (due to lack of space, we only show the axioms concerning the ‘↑’ operator)

$↑α^i = α^{i+1}$.

If ‘↑’ is functional, then

$↑α = ↑β \iff α = β$.

Compound name terms and structural descriptive names can be characterized by

$[α_0, \ldots, α_k] = [β_0, \ldots, β_k] \iff α_0 = β_0, \ldots, α_k = β_k$

$α = [↑β_0, \ldots, ↑β_k] \iff α = ↑[β_0, \ldots, β_k]$ for every metasymbol $α_0$ and $β_0$ that is a name for a function or predicate symbol.

Each step of the naming mechanism can be seen as a particular use of one of the axioms of $T$. (We shall use the notation $N \rightarrow N'$ to indicate the input $N$ and the output $N'$ of the naming mechanism.)
An equation rewrite algorithm. Martelli and Montanari [7] recast the problem of finding a most general unifier of a set of terms as the problem of finding a solution of a set of equations. Clark [3] showed that every step of the unification algorithm is a particular use of one of the freeness axioms or one of the equality axioms. Since any name theory contains the axioms proposed by Clark plus the axioms that characterize name terms and the operators ‘↑’ and ‘↓’, it is possible to extend the algorithm proposed by Martelli and Montanari in order to consider the new axioms introduced in the name theory. We require that the extended equation rewrite algorithm has the following properties.

1. It reduces a set of equations \( N \) to an equivalent set of equations \( N' \), i.e.,
   
   \[ \forall T \in \mathcal{T} \left( \mathcal{T} \models N \iff \mathcal{T} \models N' \right) \]

2. It terminates with a set of equations that is safe (a success termination) or it terminates with a set of equations augmented with the atom \textit{false} (a failure termination).

Definition 6. An algorithm is \textit{canonical} if it is confluent and finitely terminating.

If the equation rewrite algorithm is canonical, then each expression \( N \) admits a unique normal form \( N' \) such that the algorithm reduces \( N \) to \( N' \).

4 Reflection Integrated into SLD-Resolution

In this section we show how it is possible to integrate reflection rules into SLD-resolution. We take as a basis the implicit reflection of Reflective Prolog, which specifies what are probably the weakest conditions for level-shifting: each subgoal can be resolved at any level where there are applicable clauses. In the language design, this has been achieved by introducing forms of reflection into SLD-resolution to shift from the object level to the metalevel and vice versa. The extended SLD-resolution has been called RSLD-resolution [5].

RSLD-resolution. In the following we assume that names of expressions of \( L \) are structural descriptive and the alphabet of the language contains the distinguished predicate symbol \textit{solve}.

Let \( G \) be a definite goal \( \leftarrow A_1, \ldots, A_k \), let \( A_m \) be an atom, called the selected atom, in \( G \) and let \( C \) be a definite clause. Then the goal \( \leftarrow A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k \) is derived from \( G \) and \( C \) using \text{mgu} \( \theta \) iff one of the following conditions holds:

(We use the notation \( \alpha \xrightarrow{\gamma} \beta \) and \( \alpha' \xrightarrow{d} \beta' \) to indicate that the naming mechanism of Reflective Prolog returns \( \beta \) as name of \( \alpha \) and \( \beta' \) as what \( \alpha' \) names, respectively.)

i. \( C \) is \( A \leftarrow B_{q}, \ldots, B_q \)
   \[ \theta \text{ is a mgu of } A_m \text{ and } A \]

ii. \( C \) is \( \text{solve}(\alpha) \leftarrow B_1, \ldots, B_q \)
   \[ A_m \neq \text{solve}(\delta) \]
\[ A_m \xrightarrow{\theta} \alpha' \]
\( \theta \) is a mgu of \( \alpha' \) and \( \alpha \)

iii. \( A_m \) is \( \text{solve}(\alpha) \)

- \( C \) is \( A \leftarrow B_1, \ldots, B_q \)
- \( \alpha \xrightarrow{d} A' \)
- \( \theta \) is a mgu of \( A' \) and \( A \)

To explain the definition, consider the following possibilities for the selected atom \( A_m \).

1. The atom \( A_m \) can be proved in two ways. First, using the clauses defining the corresponding predicate (case (i)); for instance, if \( A_m \) is \( p(a, X) \), the clauses defining the predicate \( p \), whose head unify with \( p(a, X) \) with mgu \( \theta \). Second, the clauses defining the predicate \( \text{solve} \) can be used (case (ii), meta-to-object reflection) if the name of \( A_m \) and \( \alpha \) unify with mgu \( \theta \).

2. \( A_m \) is \( \text{solve}(\alpha) \). Again, there are two possibilities. First, using the clauses defining the predicate \( \text{solve} \) itself, similarly to any other goal (case (i)). Second, using the clauses defining the predicate corresponding to the atom partially denoted by the argument \( \alpha \) of \( \text{solve} \) (case (iii), object-to-meta reflection).

**SLD-Resolution.** In order to characterize the reflection rules of Reflective Prolog, we extend the theory of names \( T \) associated to \( L \) with the following axioms (reflection axioms):

\[
p(\alpha_1, \ldots, \alpha_k) \leftarrow \text{solve}([p^1, \beta_1, \ldots, \beta_k]), \alpha_1 = \downarrow \beta_1, \ldots, \alpha_k = \downarrow \beta_k
\]

\[
\text{solve}([\alpha_0, \alpha_1, \ldots, \alpha_k]) \leftarrow p(\beta_1, \ldots, \beta_k), \alpha_0 = \uparrow p, \alpha_1 = \uparrow \beta_1, \ldots, \alpha_k = \uparrow \beta_k
\]

for every predicate symbol \( p \) in the alphabet of an \( L \) program \( P \). They characterize meta-to-object reflection (case (ii)) and object-to-meta reflection (case (iii)), respectively.

In order to have uniformity with the language of the name theory, we express substitutions as sets of equations and we take as a basis of the presentation one of the abstract schemes presented by Clark [4]. A computation is invoked by a goal that is a multiset of atoms \( G \) meaning a conjunction of its atoms \( A_1, \ldots, A_k \). A *state* of the computation is a pair \( \langle G, Q \rangle \), where \( G \) is a multiset of atoms and \( Q \) is a safe set of equations that may contain false. The initial state is \( \langle \{\}, \{\} \rangle \). A *success* termination state is any state in the form \( \langle\{\}, Q\rangle \) where false is not in \( Q \). A *failure* termination state is any state in the form \( \langle G, Q \cup \{\text{false}\} \rangle \). The computation is a non-deterministic evaluation branching from the initial state. The computation is controlled by a *computation rule CR*. This is a rule that, for every non-terminal state \( \langle G, Q \rangle \), selects exactly one atom \( A_m \), say \( p(t_1', \ldots, t_n') \), from the multiset of atoms \( G \). The number of next states of the computation is the number of clauses in \( P \) plus the number of reflection axioms in \( T \) about the predicate symbol \( p \) of the selected atom. Let

\[
p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_s, E_1, \ldots, E_t
\]
be a variant of a clause for $p$ in $P \cup T$ such that its set of variables is disjoint from the set of variables of the state $\langle G, Q \rangle$. If the clause is in $P$, then $t = 0$, i.e., the clause does not contain any equation.

A next state is $\langle G - \{A_m\} \cup \{B_1, \ldots, B_s\}, Q' \rangle$, where $Q'$ is the result of applying the extended equation rewrite algorithm to

$$Q \cup \{t_1 = t'_1, \ldots, t_n = t'_n, E_1, \ldots, E_k\}$$

The search tree for a query $G$ is the finitely branching tree rooted at $\langle G, \{\} \rangle$. The strategy for constructing the search tree is the search strategy. The strategy is fair if it does not indefinitely postpone the construction of some branch of the search tree.

Answers to the goal $G$ are given by success terminating branches. The answer computed by a success branch terminating in $\langle \{\}, Q \rangle$ is the subset of $Q$ containing the equations whose variables are in $G$. If $G$ contains no variables, the answer is true.

5 Conclusion

In this paper we have shown how it is possible to axiomatize the main components of a metalevel architecture, namely the naming mechanism and the reflection rules. We have also presented the procedural semantics for the proposed approach. Such an axiomatization has mainly two advantages. First, it allows us to define the extended resolution of Reflective Prolog strictly in terms of SLD-resolution, i.e., $P \vdash_{RSLD} A$ iff $P \cup T \vdash_{SLD} A$. Second, a name theory being a first order theory, the declarative semantics of a metalanguage $L$ can be completely defined in terms of Herbrand interpretations.

The procedural semantics of the proposed inference system can be shown to be correct. It is complete provided that the extended equation rewrite algorithm is canonical. Because $P \vdash_{RSLD} A$ iff $P \cup T \vdash_{SLD} A$, it holds that if $P \vdash_{RSLD} A$ then $P \cup T \models A$. If the equation rewrite algorithm is canonical, then $P \cup T \models A$ implies that $P \vdash_{RSLD} A$.

References


*TOPLAS*, 4:258–282 (1982).