Similarity-based Reasoning in Qualified Logic Programming

Revised Edition

Rafael Caballero Mario Rodríguez-Artalejo Carlos A. Romero-Díaz

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
{rafa,mario}@sip.ucm.es, cromdia@fdi.ucm.es

Abstract

Similarity-based Logic Programming (briefly, SLP) has been proposed to enhance the LP paradigm with a kind of approximate reasoning which supports flexible information retrieval applications. This approach uses a fuzzy similarity relation $\mathcal{R}$ between symbols in the program's signature, while keeping the syntax for program clauses as in classical LP. Another recent proposal is the $QLP(\mathcal{D})$ scheme for *Qualified Logic Programming*, an extension of the LP paradigm which supports approximate reasoning and more. This approach uses annotated program clauses and a parametrically given domain $\mathcal{D}$ whose elements qualify logical assertions by measuring their closeness to various users' expectations. In this paper we propose a more expressive scheme $SQLP(\mathcal{R}, \mathcal{D})$ which subsumes both SLP and $QLP(\mathcal{D})$ as particular cases. We also show that $SQLP(\mathcal{R}, \mathcal{D})$ programs can be transformed into semantically equivalent $QLP(\mathcal{D})$ programs. As a consequence, existing $QLP(\mathcal{D})$ implementations can be used to give efficient support for similarity-based reasoning.

Categories and Subject Descriptors D.1.6 [*Programming Techniques*]: Logic Programming; D.3.2 [*Programming Languages*]: Language Classifications—Constraint and logic languages; F.3.2 [*Theory of Computation*]: Logics and Meanings of Programs—Algebraic approaches to semantics

General Terms Algorithms, Languages, Theory

Keywords Qualification Domains, Similarity Relations

1. Introduction

The historical evolution of the research on uncertainty in *Logic Programming* (LP) has been described in a recent recollection by V. S. Subrahmanian [19]. Early approaches include the quantitative treatment of uncertainty in the spirit of fuzzy logic, as in van Emden's classical paper [20] and two subsequent papers by Subrahmanian [17, 18]. The main contribution of [20] was a rigorous declarative semantics for a LP language with program clauses of the form $A \leftarrow d - \overline{B}$, where the head A is an atom, the body $\overline{B}$ is a conjunction of atoms, and the so-called *attenuation* factor $d \in (0, 1]$ attached to the clause's implication is used to propagate to the head the certainty factor $d \times b$, where b is the minimum of the certainty factors $d_i \in (0, 1]$ previously computed for the various atoms occurring in the body. The papers [17, 18] proposed to use a special lattice $\mathcal{T}$ in place of the lattice of the real numbers in the interval $[0, 1]$ under their natural ordering. $\mathcal{T}$ includes two isomorphic copies of $[0, 1]$ whose elements are incomparable under $\mathcal{T}$'s ordering and can be used separately to represent degrees of *truth* and *falsity*, respectively, thus enabling a simple treatment of negation. Other main contributions of [17, 18] were the introduction of annotated program clauses and goals (later generalized to a much more expressive framework in [7]), as well as goal solving procedures more convenient and powerful than those given in [20].

A more recent line of research is *Similarity-based Logic Programming* (briefly, SLP) as presented in [16] and previous related works such as [3, 6, 5, 15]. This approach also uses the lattice $[0, 1]$ to deal with uncertainty in the spirit of fuzzy logic. In contrast to approaches based on annotated clauses, programs in SLP are just sets of definite Horn clauses as in classical LP. However, a *similarity relation* $\mathcal{R}$ (roughly, the fuzzy analog of an equivalence relation) between predicate and function symbols is used to enable the unification terms that would be not unifiable in the classical sense, measured by some degree $\lambda \in (0, 1]$. There are different proposals for the operational semantics of SLP programs. One possibility is to apply classical SLD resolution w.r.t. a transformation of the original program [6, 15, 16]. Alternatively, a $\mathcal{R}$-based SLD-resolution procedure relying on $\mathcal{R}$-unification can be applied w.r.t. to the original program, as proposed in [16]. Propositions 7.1 and 7.2 in [16] state a correspondence between the answers computed by $\mathcal{R}$-based SLD resolution w.r.t. a given logic program $\mathcal{P}$ and the answers computed by classical SLD resolution w.r.t. the two transformed programs $H_\lambda(\mathcal{P})$ (built by adding to $\mathcal{P}$ new clauses $\mathcal{R}$-similar to those in $\mathcal{P}$ up to the degree $\lambda \in (0, 1]$) and $\mathcal{P}_\lambda$ (built by replacing all the function and predicate symbols in $\mathcal{P}$ by new symbols that represent equivalence classes modulo $\mathcal{R}$-similarity up to λ). The $SiLog$ system [8] has been developed to implement SLP and to support applications related to flexible information retrieval from the web.

The aim of the present paper is to show that similarity-based reasoning can be expressed in $QLP(\mathcal{D})$, a programming scheme for *Qualified* LP over a parametrically given *Qualification Domain* $\mathcal{D}$ recently presented in [14] as a generalization and improvement of the classical approach by van Emden [20] to $Quantitative$ LP. Qualification domains are lattices satisfying certain natural axioms. They include the lattice $[0, 1]$ used both in [20] and in [16], as well as other lattices whose elements can be used to qualify logical assertions by measuring their closeness to different kinds of users' expectations. Programs in $QLP(\mathcal{D})$ use $\mathcal{D}$-attenuated clauses of the form $A \leftarrow d - \overline{B}$ where A is an atom, $\overline{B}$ a finite conjunction of atoms and $d \in D \setminus \{\bot\}$ is the *attenuation value*

PPDP'08, July 15–17, 2008, Valencia, Spain.

attached to the clause's implication, used to propagate to the head the *qualification value* $d \circ b$, where b is the infimum in $\mathcal{D}$ of the qualification values $d_i \in D \setminus \{\bot\}$ previously computed for the various atoms occurring in the body, and $\circ$ is an *attenuation operator* coming with $\mathcal{D}$. As reported in [14, 13], the classical results in LP concerning the existence of least Herbrand models of programs and the soundness and completeness of the SLD resolution procedure (see e.g.[21, 2, 1]) have been extended to the $QLP(\mathcal{D})$ scheme, and potentially useful instances of the scheme have been implemented on top of the *Constraint Functional Logic Programming* ($CFLP$) system $\mathcal{TOY}$ [4].

The results presented in this paper can be summarized as follows: we consider generalized similarity relations over a set S as mappings $\mathcal{R} : S \times S \to D$ taking values in the carrier set D of an arbitrarily given qualification domain $\mathcal{D}$, and we extend $QLP(\mathcal{D})$ to a more expressive scheme $SQLP(\mathcal{R}, \mathcal{D})$ with two parameters for programming modulo $\mathcal{R}$-similarity with $\mathcal{D}$-attenuated Horn clauses. We present a declarative semantics for $SQLP(\mathcal{R}, \mathcal{D})$ and a program transformation mapping each $SQLP(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$ into a $QLP(\mathcal{D})$ program $S_{\mathcal{R}}(\mathcal{P})$ whose least Herbrand model corresponds to that of $\mathcal{P}$. Roughly, $S_{\mathcal{R}}(\mathcal{P})$ is built adding to $\mathcal{P}$ new clauses obtained from the original clauses in $\mathcal{P}$ by computing various new heads $\mathcal{R}$-similar to a linearized version of the original head, adding also $\mathcal{R}$-similarity conditions $X_i \sim X_j$ to the body and suitable clauses for the new predicate $\sim$ to emulate $\mathcal{R}$-based unification. Thanks to the $S_{\mathcal{R}}(\mathcal{P})$ transformation, the sound and complete procedure for solving goals in $QLP(\mathcal{D})$ by $\mathcal{D}$-qualified SLD resolution and its implementation in the $\mathcal{TOY}$ system [14] can be used to implement $SQLP(\mathcal{R}, \mathcal{D})$ computations, including as a particular case SLP computations in the sense of [16].

Another recent proposal for reducing the SLP approach in [16] to a fuzzy LP paradigm can be found in [11], a paper which relies on the multi-adjoint framework for Logic Programming (*MALP* for short) previously proposed in [9, 10]. *MALP* is a quite general framework supporting LP with *weighted program rules* over different multi-adjoint lattices, each of which provides a particular choice of operators for implication, conjunction and aggregation of atoms in rule bodies. In comparison to the $QLP(\mathcal{D})$ scheme, the multi-adjoint framework differs in motivation and scope. Multi-adjoint lattices and qualification domains are two different classes of algebraic structures. Concerning declarative and operational semantics, there are also some significant differences between $QLP(\mathcal{D})$ and *MALP*. In particular, *MALP* 's goal solving procedure relies on a costly computation of *reductant clauses*, a technique borrowed from [7] which can be avoided in $QLP(\mathcal{D})$, as discussed in the concluding section of [14].

In spite of these differences, the results in [11] concerning the emulation of similarity-based can be compared to those in the present paper. Theorem 24 in [11] shows that every classical logic program $\mathcal{P}$ can be transformed into a *MALP* program $\mathcal{P}_{E,\mathcal{R}}$ which can be executed using only syntactical unification and emulates the successful computations of $\mathcal{P}$ using the SLD resolution with $\mathcal{R}$-based unification introduced in [16]. $\mathcal{P}_{E,\mathcal{R}}$ works over a particular multi-adjoint lattice $\mathcal{G}$ with carrier set $[0, 1]$ and implication and conjunction operators chosen according to the so-called Gödel's semantics [22]. $\mathcal{P}_{E,\mathcal{R}}$ also introduces clauses for a binary predicate $\sim$ which emulates $\mathcal{R}$-based unification, as in our transformation $S_{\mathcal{R}}(\mathcal{P})$. Nevertheless, $S_{\mathcal{R}}(\mathcal{P})$ is defined for a more general class of programs and uses the $\mathcal{R}$-similarity predicate $\sim$ only if the source program $\mathcal{P}$ has some clause whose head is non-linear. More detailed comparisons between the program transformations $S_{\mathcal{R}}(\mathcal{P})$, $H_{\lambda}(\mathcal{P})$, $\mathcal{P}_{\lambda}$ and $\mathcal{P}_{E,\mathcal{R}}$ will be given in Subsection 4.2.

The rest of the paper is structured as follows: In Section 2 we recall the qualification domains $\mathcal{D}$ first introduced in [14] and we define similarity relations $\mathcal{R}$ over an arbitrary qualification domain. In Section 3 we recall the scheme $QLP(\mathcal{D})$ and we introduce its extension $SQLP(\mathcal{R}, \mathcal{D})$ with its declarative semantics, given by a logical calculus which characterizes the least Herbrand model $\mathcal{M}_{\mathcal{P}}$ of each $SQLP(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$. In Section 4 we define the transformation $S_{\mathcal{R}}(\mathcal{P})$ of any given $SQLP(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$ into a $QLP(\mathcal{D})$ program $S_{\mathcal{R}}(\mathcal{P})$ such that $\mathcal{M}_{S_{\mathcal{R}}(\mathcal{P})} = \mathcal{M}_{\mathcal{P}}$, we give some comparisons to previously known program transformations, and we illustrate the application of $S_{\mathcal{R}}(\mathcal{P})$ to similarity-based computation by means of a simple example. Finally, in Section 5 we summarize conclusions and comparisons to related work and we point to planned lines of future work.

2. Qualification Domains and Similarity Relations

2.1 Qualification Domains

Qualification Domains were introduced in [14] with the aim of using their elements to qualify logical assertions in different ways. In this subsection we recall their axiomatic definition and some significant examples.

Definition 1. A *Qualification Domain* is any structure $\mathcal{D} = \langle D, \sqsubseteq, \bot, \top, \circ\rangle$ verifying the following requirements:

1. $\langle D, \sqsubseteq, \bot, \top\rangle$ is a lattice with extreme points $\bot$ and $\top$ w.r.t. the partial ordering $\sqsubseteq$. For given elements $d, e \in D$, we write $d \sqcap e$ for the *greatest lower bound* (glb) of d and e and $d \sqcup e$ for the *least upper bound* (lub) of d and e. We also write $d \sqsubset e$ as abbreviation for $d \sqsubseteq e \wedge d \neq e$.
2. $\circ : D \times D \to D$, called *attenuation operation*, verifies the following axioms:
   (a) $\circ$ is associative, commutative and monotonic w.r.t. $\sqsubseteq$.
   (b) $\forall d \in D : d \circ \top = d$.
   (c) $\forall d \in D : d \circ \bot = \bot$.
   (d) $\forall d, e \in D \setminus \{\bot, \top\} : d \circ e \sqsubset e$.
   (e) $\forall d, e_1, e_2 \in D : d \circ (e_1 \sqcap e_2) = d \circ e_1 \sqcap d \circ e_2$. □

In the rest of the paper, $\mathcal{D}$ will generally denote an arbitrary qualification domain. For any finite $S = \{e_1, e_2, \ldots, e_n\} \subseteq D$, the glb of S (noted as $\bigsqcap S$) exists and can be computed as $e_1 \sqcap e_2 \sqcap \cdots \sqcap e_n$ (which reduces to $\top$ in the case $n = 0$). As an easy consequence of the axioms, one gets the identity $d \circ \bigsqcap S = \bigsqcap\{d \circ e \mid e \in S\}$. The $QLP(\mathcal{D})$ scheme presented in [14] supports LP over a parametrically given qualification domain $\mathcal{D}$.

Example 1. *Some examples of qualification domains are presented below. Their intended use for qualifying logical assertions will become more clear in Subsection 3.1.*

1. $\mathcal{B} = (\{0, 1\}, \leq, 0, 1, \wedge)$, *where* 0 *and* 1 *stand for the two classical truth values* false *and* true*,* $\leq$ *is the usual numerical ordering over* $\{0, 1\}$*, and* $\wedge$ *stands for the classical conjunction operation over* $\{0, 1\}$*. Attaching* 1 *to an atomic formula* A *is intended to qualify* A *as 'true' in the sense of classical* LP*.*
2. $\mathcal{U} = (U, \leq, 0, 1, \times)$*, where* $U = [0, 1] = \{d \in \mathbb{R} \mid 0 \leq d \leq 1\}$*,* $\leq$ *is the usual numerical ordering, and* $\times$ *is the multiplication operation. In this domain, the top element* $\top$ *is* 1 *and the greatest lower bound* $\bigsqcap S$ *of a finite* $S \subseteq U$ *is the minimum value min(S), which is* 1 *if* $S = \emptyset$*. Attaching an element* $c \in U \setminus \{0\}$ *to an atomic formula* A *is intended to qualify* A *as 'true with certainty degree* c*' in the spirit of fuzzy logic, as done in the classical paper [20] by van Emden. The computation of qualifications* c *as certainty degrees in* $\mathcal{U}$ *is due to the interpretation of* $\sqcap$ *as* min *and* $\circ$ *as* $\times$*.*
3. $\mathcal{W} = (P, \geq, \infty, 0, +)$*, where* $P = [0, \infty] = \{d \in \mathbb{R} \cup \{\infty\} \mid d \geq 0\}$*,* $\geq$ *is the reverse of the usual numerical ordering (with* $\infty \geq d$ *for any* $d \in P$*), and* $+$ *is the addition operation (with*

$\infty + d = d + \infty = \infty$ for any $d \in P$). In this domain, the top element $\top$ is 0 and the greatest lower bound $\bigsqcap S$ of a finite $S \subseteq P$ is the maximum value max(S), which is 0 if $S = \emptyset$. Attaching an element $d \in P \setminus \{\infty\}$ to an atomic formula A is intended to qualify A as 'true with weighted proof depth d'. The computation of qualifications d as weighted proof depths in $\mathcal{W}$ is due to the interpretation of $\sqcap$ as max and $\circ$ as $+$.

4. *Given 2 qualification domains $\mathcal{D}_i = \langle D_i, \sqsubseteq_i, \bot_i, \top_i, \circ_i \rangle$ ($i \in \{1, 2\}$), their* cartesian product $\mathcal{D}_1 \times \mathcal{D}_2$ *is $\mathcal{D} =_{\mathrm{def}} \langle D, \sqsubseteq, \bot, \top, \circ \rangle$, where $D =_{\mathrm{def}} D_1 \times D_2$, the partial ordering $\sqsubseteq$ is defined as $(d_1, d_2) \sqsubseteq (e_1, e_2) \Longleftrightarrow_{\mathrm{def}} d_1 \sqsubseteq_1 e_1$ and $d_2 \sqsubseteq_2 e_2$, $\bot =_{\mathrm{def}} (\bot_1, \bot_2)$, $\top =_{\mathrm{def}} (\top_1, \top_2)$, and the attenuation operator $\circ$ is defined as $(d_1, d_2) \circ (e_1, e_2) =_{\mathrm{def}} (d_1 \circ_1 e_1, d_2 \circ_2 e_2)$. The product of two given qualification domains is always another qualification domain, as proved in [14]. Intuitively, each value (d_1, d_2) belonging to $\mathcal{D}_1 \times \mathcal{D}_2$ imposes the qualification d_1* and also *the qualification d_2. For instance, values (c, d) belonging to $\mathcal{U} \times \mathcal{W}$ impose two qualifications, namely: a certainty degree greater or equal than c and a weighted proof depth less or equal than d.* □

For technical reasons that will become apparent in Section 4, we consider the two structures $\mathcal{U}'$ resp. $\mathcal{W}'$ defined analogously to $\mathcal{U}$ resp. $\mathcal{W}$, except that $\circ$ behaves as min in $\mathcal{U}'$ and as max in $\mathcal{W}'$. Note that almost all the axioms for qualification domains enumerated in Definition 1 hold in $\mathcal{U}'$ and $\mathcal{W}'$, except that axiom 2.(d) holds only in the relaxed form $\forall d, e \in D : d \circ e \sqsubseteq e$. Therefore, we will refer to $\mathcal{U}'$ and $\mathcal{W}'$ as *quasi* qualification domains.

2.2 Similarity relations

Similarity relations over a given set S have been defined in [16] and related literature as mappings $\mathcal{R} : S \times S \to [0, 1]$ that satisfy three axioms analogous to those required for classical equivalence relations. Each value $\mathcal{R}(x, y)$ computed by a similarity relation $\mathcal{R}$ is called the *similarity degree* between x and y. In this paper we use a natural extension of the definition given in [16], allowing elements of an arbitrary qualification domain $\mathcal{D}$ to serve as similarity degrees. As in [16], we are especially interested in similarity relations over sets S whose elements are variables and symbols of a given signature.

Definition 2. Let a qualification domain $\mathcal{D}$ with carrier set D and a set S be given.

1. A *$\mathcal{D}$-valued similarity relation* over S is any mapping $\mathcal{R} : S \times S \to D$ such that the three following axioms hold for all $x, y, z \in S$:
   (a) *Reflexivity:* $\mathcal{R}(x, x) = \top$.
   (b) *Symmetry:* $\mathcal{R}(x, y) = \mathcal{R}(y, x)$.
   (c) *Transitivity:* $\mathcal{R}(x, z) \sqsupseteq \mathcal{R}(x, y) \sqcap \mathcal{R}(y, z)$.
2. The mapping $\mathcal{R} : S \times S \to D$ defined as $\mathcal{R}(x, x) = \top$ for all $x \in D$ and $\mathcal{R}(x, y) = \bot$ for all $x, y \in D$, $x \neq y$ is trivially a $\mathcal{D}$-valued similarity relation called the *identity*.
3. A $\mathcal{D}$-valued similarity relation $\mathcal{R}$ over S is called *admissible* iff $S = \mathcal{V}ar \cup CS \cup PS$ (where the three mutually disjoint sets $\mathcal{V}ar$, CS and PS stand for a countably infinite collection of *variables*, a set of *constructor symbols* and a set of *predicate symbols*, respectively) and the two following requirements are satisfied:
   (a) $\mathcal{R}$ restricted to $\mathcal{V}ar$ behaves as the identity, i.e. $\mathcal{R}(X, X) = \top$ for all $X \in \mathcal{V}ar$ and $\mathcal{R}(X, Y) = \bot$ for all $X, Y \in \mathcal{V}ar$, $X \neq Y$.
   (b) $\mathcal{R}(x, y) \neq \bot$ holds only if some of the following three cases holds x, y: either $x, y \in \mathcal{V}ar$ are both the same variable; or else $x, y \in CS$ are constructor symbols with the same arity; or else $x, y \in PS$ are predicate symbols with the same arity. □

The similarity degrees computed by a $\mathcal{D}$-valued similarity relation must be interpreted w.r.t. the intended role of $\mathcal{D}$-elements as qualification values. For example, let $\mathcal{R}$ be an admissible similarity relation, and let $c, d \in CS$ be two nullary constructor symbols (i.e., constants). If $\mathcal{R}$ is $\mathcal{U}$-valued, then $\mathcal{R}(c, d)$ can be interpreted as a *certainty degree* for the assertion that c and d are similar. On the other hand, if $\mathcal{R}$ is $\mathcal{W}$-valued, then $\mathcal{R}(c, d)$ can be interpreted as a *cost* to be paid for c to play the role of d. These two views are coherent with the different interpretations of the operators $\sqcap$ and $\circ$ in $\mathcal{U}$ and $\mathcal{W}$, respectively.

In the rest of the paper we assume that any admissible similarity relation $\mathcal{R}$ can be extended to act over terms, atoms and clauses. The extension, also called $\mathcal{R}$, can be recursively defined as in [16]. The following definition specifies the extension of $\mathcal{R}$ acting over terms. The case of atoms and clauses is analogous.

Definition 3. ($\mathcal{R}$ acting over terms).

1. For $X \in \mathcal{V}ar$ and for any term t different from X: $\mathcal{R}(X, X) = \top$ and $\mathcal{R}(X, t) = \mathcal{R}(t, X) = \bot$.
2. For $c, c' \in CS$ with different arities n, m: $\mathcal{R}(c(t_1, \ldots, t_n), c'(t'_1, \ldots, t'_m)) = \bot$.
3. For $c, c' \in CS$ with the same arity n: $\mathcal{R}(c(t_1, \ldots, t_n), c'(t'_1, \ldots, t'_n)) = \mathcal{R}(c, c') \sqcap \mathcal{R}(t_1, t'_1) \sqcap \ldots \sqcap \mathcal{R}(t_n, t'_n)$.

3. Similarity-based Qualified Logic Programming

In this section we extend our previous scheme $QLP(\mathcal{D})$ to a more expressive scheme called *Similarity-based Qualified Logic Programming* over $(\mathcal{R}, \mathcal{D})$ –abbreviated as $SQLP(\mathcal{R}, \mathcal{D})$– which supports both qualification over $\mathcal{D}$ in the sense of [14] and $\mathcal{R}$-based similarity in the sense of [16] and related research. Subsection 3.1 presents a quick review of the main results concerning syntax and declarative semantics of $QLP(\mathcal{D})$ already presented in [14], while the extensions needed to conform the new $SQLP(\mathcal{R}, \mathcal{D})$ scheme are presented in subsection 3.2.

3.1 Qualified Logic Programming

$QLP(\mathcal{D})$ was proposed in our previous work [14] as a generic scheme for qualified logic programming over a given qualification domain $\mathcal{D}$. In that scheme, a *signature* Σ providing constructor and predicate symbols with given arities is assumed. *Terms* are built from constructors and *variables* from a countably infinite set $\mathcal{V}ar$ (disjoint from Σ) and *Atoms* are of the form $p(t_1, \ldots, t_n)$ (shortened as $p(\overline{t_n})$ or simply $p(\overline{t})$) where p is a n-ary predicate symbol and t_i are terms. We write At_Σ, called the *open Herbrand base*, for the set of all atoms. A $QLP(\mathcal{D})$ program $\mathcal{P}$ is a finite set of *$\mathcal{D}$-qualified definite Horn clauses* of the form $A \leftarrow d - \overline{B}$ where A is an atom, $\overline{B}$ a finite conjunction of atoms and $d \in D \setminus \{\bot\}$ is the *attenuation value* attached to the clause's implication.

As explained in [14], in our aim to work with qualifications we are not only interested in just proving an atom, but in proving it along with a qualification value. For this reason, *$\mathcal{D}$-qualified atoms* ($A \sharp d$ where A is an atom and $d \in D \setminus \{\bot\}$) are introduced to represent the statement that the atom A holds for *at least* the qualification value d. For use in goals to be solved, *open $\mathcal{D}$-annotated atoms* ($A \sharp W$ where A is an atom and W a *qualification variable* intended to take values over $\mathcal{D}$) are also introduced, and a countably infinite set $\mathcal{W}ar$ of qualification variables (disjoint from $\mathcal{V}ar$ and Σ) is postulated. The *annotated Herbrand base* over $\mathcal{D}$ is defined as the set $\mathrm{At}_\Sigma(\mathcal{D})$ of all $\mathcal{D}$-qualified atoms. A *$\mathcal{D}$-entailment relation* over $\mathrm{At}_\Sigma(\mathcal{D})$, defined as $A \sharp d \succcurlyeq_\mathcal{D} A' \sharp d'$ iff there is some substitution θ such that $A' = A\theta$ and $d' \sqsubseteq d$, is used to formally define an *open Herbrand interpretation* over $\mathcal{D}$ –from now on just an *interpretation*– as any subset $\mathcal{I} \subseteq \mathrm{At}_\Sigma(\mathcal{D})$ which is closed under $\mathcal{D}$-entailment. We write $\mathrm{Int}_\Sigma(\mathcal{D})$ for the family of all interpretations. The notion of model is such that given any clause $C \equiv A \leftarrow d\!- B_1, \ldots, B_k$ in the $QLP(\mathcal{D})$ program $\mathcal{P}$, an interpretation $\mathcal{I}$ is said to be a *model* of C iff for any substitution θ and any qualification values $d_1, \ldots, d_k \in D \setminus \{\bot\}$ such that $B_i\theta \sharp d_i \in \mathcal{I}$ for all $1 \leq i \leq k$, one has $A\theta \sharp (d \circ \bigsqcap\{d_1, \ldots, d_k\}) \in \mathcal{I}$. The interpretation $\mathcal{I}$ is also said to be a model of the $QLP(\mathcal{D})$ program $\mathcal{P}$ (written as $\mathcal{I} \models \mathcal{P}$) iff it happen to be a model of every clause in $\mathcal{P}$.

As technique to infer formulas (or in our case $\mathcal{D}$-qualified atoms) from a given $QLP(\mathcal{D})$ program $\mathcal{P}$, and following traditional ideas, we consider two alternative ways of formalizing an inference step which goes from the body of a clause to its head: both an interpretation transformer $\mathrm{T}_\mathcal{P} : \mathrm{Int}_\Sigma(\mathcal{D}) \to \mathrm{Int}_\Sigma(\mathcal{D})$, and a qualified variant of Horn Logic, noted as $QHL(\mathcal{D})$, called *Qualified Horn Logic* over $\mathcal{D}$. As both methods are equivalent and correctly characterize the least Herbrand model of a given program $\mathcal{P}$, we will only be recalling the logic $QHL(\mathcal{D})$, although we encourage the reader to see Section 3.2 in [14], where the fix-point semantics is explained.

The logic $QHL(\mathcal{D})$ is defined as a deductive system consisting just of one inference rule: QMP($\mathcal{D}$), called *Qualified Modus Ponens* over $\mathcal{D}$. Such rule allows us to give the following inference step given that there were some $(A \leftarrow d\!- B_1, \ldots, B_k) \in \mathcal{P}$, some substitution θ such that $A' = A\theta$ and $B'_i = B_i\theta$ for all $1 \leq i \leq k$ and some $d' \in D \setminus \{\bot\}$ such that $d' \sqsubseteq d \circ \bigsqcap\{d_1, \ldots, d_k\}$:

$$\frac{B'_1 \sharp d_1 \quad \cdots \quad B'_k \sharp d_k}{A' \sharp d'} \quad \mathrm{QMP}(\mathcal{D})$$

Roughly, each QMP($\mathcal{D}$) inference step using an instance of a program clause $A \leftarrow d\!- \overline{B}$ has the effect of propagating to the head the *qualification value* $d \circ b$, where b is the infimum in $\mathcal{D}$ of the qualification values $d_i \in D \setminus \{\bot\}$ previously computed for the various atoms occurring in the body. This helps to understand the claims made in Example 1 above about the intended use of elements of the domains $\mathcal{U}$ and $\mathcal{W}$ for qualifying logical assertions. We use the notations $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} A \sharp d$ (resp. $\mathcal{P} \vdash^n_{\mathrm{QHL}(\mathcal{D})} A \sharp d$) to indicate that $A \sharp d$ can be inferred from the clauses in program $\mathcal{P}$ in finitely many steps (resp. n steps). The *least Herbrand model of* $\mathcal{P}$ happens to be $\mathcal{M}_\mathcal{P} = \{A \sharp d \mid \mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} A \sharp d\}$, as proved in [14].

3.2 Similarity-based Qualified Logic Programming

The scheme $SQLP(\mathcal{R}, \mathcal{D})$ presented in this subsection has two parameters $\mathcal{R}$ and $\mathcal{D}$, where $\mathcal{D}$ can be any qualification domain and $\mathcal{R}$ can be any admissible $\mathcal{D}$-valued similarity relation, in the sense of Definition 2. The new scheme subsumes the approach in [14] by behaving as $QLP(\mathcal{D})$ in the case that $\mathcal{R}$ is chosen as the identity, and it also subsumes similarity-based LP by behaving as the approach in [16] and related papers in the case that $\mathcal{D}$ is chosen as $\mathcal{U}$.

Syntactically, $SQLP(\mathcal{R}, \mathcal{D})$ presents almost no changes w.r.t. $QLP(\mathcal{D})$, but the declarative semantics must be extended to account for the behavior of the parametrically given similarity relation $\mathcal{R}$. As in the previous subsection, we assume a signature Σ providing again constructor and predicate symbols. *Terms* and *Atoms* are built the same way they were in $QLP(\mathcal{D})$, and At_Σ will stand again for the set of all atoms, called the *open Herbrand base*. An atom A is called *linear* if there is no variable with multiple occurrences in A; otherwise A is called *non-linear*. A $SQLP(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$ is a finite set of $\mathcal{D}$*-qualified definite Horn clauses* with the same syntax as in $QLP(\mathcal{D})$, along with a $\mathcal{D}$-valued admissible similarity relation $\mathcal{R}$ in the sense of Definition 2, item 2. Figure 1 shows a simple $SQLP(\mathcal{R},\mathcal{U})$ program built from the similarity

```
wild(lynx) <-0.9-
wild(boar) <-0.9-
wild(snake) <-1.0-

farm(cow) <-1.0-
farm(pig) <-1.0-

domestic(cat) <-0.8-
domestic(snake) <-0.4-

intelligent(A) <-0.9- domestic(A)
intelligent(lynx) <-0.7-

pacific(A) <-0.9- domestic(A)
pacific(A) <-0.7- farm(A)

pet(A) <-1.0- pacific(A), intelligent(A)
```

```
R(farm,domestic) = 0.3
R(pig,boar) = 0.7
R(lynx,cat) = 0.8
```

Figure 1. $SQLP(\mathcal{R},\mathcal{U})$ program.

relation $\mathcal{R}$ given in the same figure and the qualification domain $\mathcal{U}$ for certainty values. This program will be used just for illustrative purposes in the rest of the paper. The reader is referred to Section 2 for other examples of qualification domains, and to the references [8, 11] for suggestions concerning practical applications of similarity-based LP.

$\mathcal{D}$*-qualified atoms* ($A \sharp d$ with A an atom and $d \in D \setminus \{\bot\}$) and *open* $\mathcal{D}$*-annotated atoms* ($A \sharp W$ with A and atom and $W \in War$ a qualification variable intended to take values in $D \setminus \{\bot\}$) will still be used here. Similarly, the *annotated open Herbrand base* over $\mathcal{D}$ is again defined as the set $\mathrm{At}_\Sigma(\mathcal{D})$ of all $\mathcal{D}$-qualified atoms. At this point, and before extending the notions of $\mathcal{D}$-entailment relation and interpretation to the $SQLP(\mathcal{R}, \mathcal{D})$ scheme, we need to define what an $\mathcal{R}$-instance of an atom is. Intuitively, when building $\mathcal{R}$-instances of an atom A, signature symbols occurring in A can be replaced by similar ones, and different occurrences of the same variable in A may be replaced by different terms, whose degree of similarity must be taken into account. Technically, $\mathcal{R}$-instances of an atom $A \in \mathrm{At}_\Sigma$ are built from a linearized version of A which has the form $\mathrm{lin}(A) = (A_\ell, \mathcal{S}_\ell)$ and is constructed as follows: A_ℓ is a linear atom built from A by replacing each n additional occurrences of a variable X by new fresh variables X_i ($1 \leq i \leq n$); and $\mathcal{S}_\ell$ is a set of *similarity conditions* $X \sim X_i$ (with $1 \leq i \leq n$) asserting the similarity of all variables in A_ℓ that correspond to the same variable X in A. As a concrete illustration, let us show the linearization of two atoms. Note what happens when the atom A is already linear as in the first case: A_ℓ is just the same as A and $\mathcal{S}_\ell$ is empty.

- $H_1 = p(c(X), Y)$
  $\mathrm{lin}(H_1) = (p(c(X), Y),\ \{\})$
- $H_2 = p(c(X), X, Y)$
  $\mathrm{lin}(H_2) = (p(c(X), X_1, Y),\ \{X \sim X_1\})$

Now we are set to formally define the $\mathcal{R}$-instances of an atom.

Definition 4. ($\mathcal{R}$-instance of an atom). Assume an atom $A \in \mathrm{At}_\Sigma$ and its linearized version $\mathrm{lin}(A) = (A_\ell, \mathcal{S}_\ell)$. Then, an atom A' is said to be an $\mathcal{R}$-instance of A with similarity degree δ, noted as $(A', \delta) \in [A]_{\mathcal{R}}$, iff there are some atom $A^{\mathcal{S}}$ and some substitution θ such that $A' = A^{\mathcal{S}}\theta$ and $\delta = \mathcal{R}(A_\ell, A^{\mathcal{S}}) \sqcap \bigsqcap\{\mathcal{R}(X_i\theta, X_j\theta) \mid (X_i \sim X_j) \in \mathcal{S}_\ell\} \neq \bot$.

Next, the $(\mathcal{R}, \mathcal{D})$-*entailment relation* over $\mathrm{At}_\Sigma(\mathcal{D})$ is defined as follows: $A \sharp d \succcurlyeq_{(\mathcal{R},\mathcal{D})} A' \sharp d'$ iff there is some similarity degree δ such that $(A', \delta) \in [A]_{\mathcal{R}}$ and $d' \sqsubseteq d \circ \delta$. Finally, an *open Herbrand interpretation* –just *interpretation* from now on– over $(\mathcal{R}, \mathcal{D})$ is defined as any subset $\mathcal{I} \in \mathrm{At}_\Sigma(\mathcal{D})$ which is closed under $(\mathcal{R}, \mathcal{D})$-entailment. That is, an interpretation $\mathcal{I}$ including a given $\mathcal{D}$-qualified atom $A \sharp d$ is required to include all the 'similar instances' $A' \sharp d'$ such that $A \sharp d \succcurlyeq_{(\mathcal{R},\mathcal{D})} A' \sharp d'$, because we intend to formalize a semantics in which all such similar instances are valid whenever $A \sharp d$ is valid. This complements the intuition given for the $\mathcal{D}$-entailment relation in $QLP(\mathcal{D})$ to include the similar instances (obtainable due to $\mathcal{R}$) of each atom, and not only those which are true because we can prove them for a better (i.e. higher in $\mathcal{D}$) qualification. Note that $(\mathcal{R}, \mathcal{D})$-entailment is a refinement of $\mathcal{D}$-entailment, since: $A \sharp d \succcurlyeq_{\mathcal{D}} A' \sharp d' \Longrightarrow$ there is some substitution θ such that $A' = A\theta$ and $d' \sqsubseteq d \Longrightarrow (A', \top) \in [A]_{\mathcal{R}}$ and $d' \sqsubseteq d \circ \top$ $\Longrightarrow A \sharp d \succcurlyeq_{(\mathcal{R},\mathcal{D})} A' \sharp d'$.

As an example of the closure of interpretations w.r.t. $(\mathcal{R}, \mathcal{D})$-entailment, consider the $\mathcal{U}$-qualified atom `domestic(cat)#0.8`. As a trivial consequence of Proposition 2 below, this atom belongs to the least Herbrand model of the program in Figure 1. On the other hand, we also know that `lynx` is similar to `cat` with a similarity degree of 0.8 w.r.t. the similarity relation $\mathcal{R}$ in Figure 1. Therefore, `domestic(lynx)` is a $\mathcal{R}$-instance of `domestic(cat)` to the degree 0.8. Then, by definition of $(\mathcal{R}, \mathcal{U})$-entailment, it turns out that `domestic(cat)#0.8` $\succcurlyeq_{(\mathcal{R},\mathcal{U})}$ `domestic(lynx)#0.64`, and the $\mathcal{U}$-qualified atom `domestic(lynx)#0.64` does also belong to the least model of the example program. Intuitively, $0.64 = 0.8 \times 0.8$ is the best $\mathcal{U}$-qualification which can be inferred from the $\mathcal{U}$-qualification 0.8 for `domestic(cat)` and the $\mathcal{R}$-similarity 0.8 between `domestic(cat)` and `domestic(lynx)`.

We will write $\mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D})$ for the family of all interpretations over $(\mathcal{R}, \mathcal{D})$, a family for which the following proposition can be easily proved from the definition of an interpretation and the definitions of the union and intersection of a family of sets.

Proposition 1. *The family* $\mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D})$ *of all interpretations over* $(\mathcal{R}, \mathcal{D})$ *is a complete lattice under the inclusion ordering* $\subseteq$*, whose extreme points are* $\mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D})$ *as maximum and* $\emptyset$ *as minimum. Moreover, given any family of interpretations* $I \subseteq \mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D})$*, its lub and glb are* $\bigsqcap I = \bigcup\{\mathcal{I} \in \mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D}) \mid \mathcal{I} \in I\}$ *and* $\bigsqcup I = \bigcap\{\mathcal{I} \in \mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D}) \mid \mathcal{I} \in I\}$*, respectively.*

Similarly as we did for the $\mathcal{R}$-instances of an atom, we will define what the $\mathcal{R}$-instances of a clause are. The following definition tells us so.

Definition 5. ($\mathcal{R}$-instance of a clause). Assume a clause $C \equiv A \leftarrow d - B_1, \ldots, B_k$ and the linearized version of its head atom $\mathrm{lin}(A) = (A_\ell, \mathcal{S}_\ell)$. Then, a clause C' is said to be an $\mathcal{R}$-instance of C with similarity degree δ, noted as $(C', \delta) \in [C]_{\mathcal{R}}$, iff there are some atom $A^{\mathcal{S}}$ and some substitution θ such that $\delta = \mathcal{R}(A_\ell, A^{\mathcal{S}}) \sqcap \bigsqcap\{\mathcal{R}(X_i\theta, X_j\theta) \mid (X_i \sim X_j) \in \mathcal{S}_\ell\} \neq \bot$ and $C' \equiv A^{\mathcal{S}}\theta \leftarrow d - B_1\theta, \ldots, B_k\theta$.

Note that as an immediate consequence from Definitions 4 and 5 it is true that given two clauses C and C' such that $(C', \delta) \in [C]_{\mathcal{R}}$, and assuming A to be head atom of C and A' to be the head atom of C', then we have that $(A', \delta) \in [A]_{\mathcal{R}}$.

Let C be any clause $A \leftarrow d - B_1, \ldots, B_k$ in the program $\mathcal{P}$, and $\mathcal{I} \in \mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D})$ any interpretation over $(\mathcal{R}, \mathcal{D})$. We say that $\mathcal{I}$ is a model of C iff for any clause $C' \equiv H' \leftarrow d - B'_1, \ldots, B'_k$ such that $(C', \delta) \in [C]_{\mathcal{R}}$ and any qualification values $d_1, \ldots, d_k \in D \setminus \{\bot\}$ such that $B'_i \sharp d_i \in \mathcal{I}$ for all $1 \le i \le k$, one has $H' \sharp d' \in \mathcal{I}$ where $d' = d \circ \bigsqcap\{e, d_1, \ldots, d_k\}$. And we say that $\mathcal{I}$ is a model of the $SQLP(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$ (also written $\mathcal{I} \models \mathcal{P}$) iff $\mathcal{I}$ is a model of each clause in $\mathcal{P}$.

We will provide now a way to perform an inference step from the body of a clause to its head. As in the case of $QLP(\mathcal{D})$, this can be formalized in two alternative ways, namely an interpretation transformer and a variant of Horn Logic. Both approaches lead to equivalent characterizations of least program models. Here we focus on the second approach, defining what we will call *Similarity-based Qualified Horn Logic* over $(\mathcal{R}, \mathcal{D})$ –abbreviated as $SQHL(\mathcal{R}, \mathcal{D})$–, another variant of Horn Logic and an extension of the previous $QHL(\mathcal{D})$. The logic $SQHL(\mathcal{R}, \mathcal{D})$ is also defined as a deductive system consisting just of one inference rule $SQMP(\mathcal{R}, \mathcal{D})$, called *Similarity-based Qualified Modus Ponens* over $(\mathcal{R}, \mathcal{D})$:

If $((A' \leftarrow d - B'_1, \ldots, B'_k), \delta) \in [C]_{\mathcal{R}}$ for some clause $C \in \mathcal{P}$ with attenuation value d, then the following inference step is allowed for any $d' \in D \setminus \{\bot\}$ such that $d' \sqsubseteq d \circ \bigsqcap\{\delta, d_1, \ldots, d_k\}$:

$$\frac{B'_1 \sharp d_1 \quad \cdots \quad B'_k \sharp d_k}{A' \sharp d'} \quad \text{SQMP}(\mathcal{R}, \mathcal{D}) \ .$$

We will use the notations $\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d$ (respectively $\mathcal{P} \vdash^n_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d$) to indicate that $A \sharp d$ can be inferred from the clauses in program $\mathcal{P}$ in finitely many steps (respectively n steps). Note that $SQHL(\mathcal{R}, \mathcal{D})$ proofs can be naturally represented as upwards growing *proof trees* with $\mathcal{D}$-qualified atoms at their nodes, each node corresponding to one inference step having the children nodes as premises.

The following proposition contains the main result concerning the declarative semantics of the $SQLP(\mathcal{R}, \mathcal{D})$ scheme. A full proof can be developed in analogy to the $QLP(\mathcal{D})$ case presented in [14, 13].

Proposition 2. *Given any* $SQLP(\mathcal{R}, \mathcal{D})$ *program* $\mathcal{P}$*. The* least Herbrand model *(*$\mathcal{M}_{\mathcal{P}}$*) of* $\mathcal{P}$ *is*

$$\{A \sharp d \mid \mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d\} \ .$$

The following example serves as an illustration of how the logic $SQHL(\mathcal{R}, \mathcal{D})$ works over $(\mathcal{R}, \mathcal{U})$ using the example program displayed in Figure 1.

Example 2. *The following proof tree proves that the atom* `pet(lynx)` *can be inferred for at least a qualification value of* 0.50 *in the* $SQLP(\mathcal{R},\mathcal{U})$ *program* $\mathcal{P}$ *of Figure 1. Let's see it:*

$$\dfrac{\dfrac{\dfrac{}{\texttt{domestic(lynx)\#0.64}}\,\textit{(4)}}{\texttt{pacific(lynx)\#0.57}}\,\textit{(2)} \qquad \dfrac{}{\texttt{intelligent(lynx)\#0.70}}\,\textit{(3)}}{\texttt{pet(lynx)\#0.50}}\,\textit{(1)}$$

where the clauses and qualification values used for each inference step are:

(1) `pet(lynx) <-1.0- pacific(lynx),intelligent(lynx)` *is an instance of clause* 12 *in* $\mathcal{P}$ *and* $0.50 \le 1.0 \times \min\{1.0,$ $0.57,\ 0.70\}$*. Note that the first* 1.0 *in the minimum is the one which comes from the similarity relation as for this step we are just using a plain instance of clause* 12 *in* $\mathcal{P}$*.*

(2) `pacific(lynx) <-0.9- domestic(lynx)` *is a plain instance of clause* 10 *in* $\mathcal{P}$ *and* $0.57 \le 0.9 \times \min\{1.0, 0.64\}$*.*

(3) `intelligent(lynx) <-0.7-` *is clause* 9 *in* $\mathcal{P}$ *and* $0.70 \le 0.70 \times \min\{1.0\}$*.*

(4) The clause `domestic(lynx) <-0.8-` *is an* $\mathcal{R}$*-instance of clause* 6 *with a similarity degree of* 0.8 *and we have* $0.64 \le 0.8 \times \min\{0.8\}$*.* $\square$

4. Reducing Similarities to Qualifications

4.1 A Program Transformation

In this section we prove that any $SQLP(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$ can be transformed into an equivalent $QLP(\mathcal{D})$ program which will be denoted by $S_\mathcal{R}(\mathcal{P})$. The program transformation is defined as follows:

Definition 6. Let $\mathcal{P}$ be a $SQLP(\mathcal{R}, \mathcal{D})$ program. We define the transformed program $S_\mathcal{R}(\mathcal{P})$ as:

$$S_\mathcal{R}(\mathcal{P}) = \mathcal{P}_S \cup \mathcal{P}_\sim \cup \mathcal{P}_{\mathrm{pay}}$$

where the auxiliary sets of clauses $\mathcal{P}_S, \mathcal{P}_\sim, \mathcal{P}_{\mathrm{pay}}$ are defined as:

- For each clause $(H \leftarrow d - \overline{B}) \in \mathcal{P}$ and for each H' such that $\mathcal{R}(H_\ell, H') \neq \bot$
$$(H' \leftarrow d - pay_{\mathcal{R}(H_\ell,H')}, S_\ell, \overline{B}) \in \mathcal{P}_S$$
where $(H_\ell, S_\ell) = lin(H)$.
- $\mathcal{P}_\sim = \{X \sim X \leftarrow \top - \} \cup \{(c(\overline{X}_n) \sim c'(\overline{Y}_n) \leftarrow \top - pay_{\mathcal{R}(c,c')}, X_1 \sim Y_1, \ldots, X_n \sim Y_n) \mid c, c' \in CS$ of arity n, $\mathcal{R}(c, c') \neq \bot\}$
- $\mathcal{P}_{\mathrm{pay}} = \{(pay_w \leftarrow w-) \mid$ for each atom pay_w occurring in $\mathcal{P}_\sim \cup \mathcal{P}_S\}$

Note that the linearization of clause heads in this transformation is motivated by the role of linearized atoms in the $SQHL(\mathcal{R}, \mathcal{D})$ logic defined in Subsection 3.2 to specify the declarative semantics of $SQLP(\mathcal{R}, \mathcal{D})$ programs. For instance, assume a $SQLP(\mathcal{R}, \mathcal{U})$ program $\mathcal{P}$ including the clause $p(X, X) \leftarrow 1.0-$ and two nullary constructors c, d such that $\mathcal{R}(c, d) = 0.8$. Then, $SQHL(\mathcal{R}, \mathcal{U})$ supports the derivation $\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{U})} p(c, d) \sharp 0.8$, and the transformed program $S_\mathcal{R}(\mathcal{P})$ will include the clauses

$$\begin{array}{ll} p(X, X_1) & \leftarrow 1.0 - pay_{1.0}, X \sim X_1, \\ X \sim X & \leftarrow 1.0-, \\ c \sim d & \leftarrow 1.0 - pay_{0.8}, \\ pay_{1.0} & \leftarrow 1.0-, \\ pay_{0.8} & \leftarrow 0.8- \end{array}$$

thus enabling the corresponding derivation $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{U})} p(c, d) \sharp 0.8$ in $QHL(\mathcal{U})$.

In general, $\mathcal{P}$ and $S_\mathcal{R}(\mathcal{P})$ are semantically equivalent in the sense that $\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d \iff S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} A \sharp d$ holds for any $\mathcal{D}$-qualified atom $A \sharp d$, as stated in Theorem 1 below. The next technical lemma will be useful for the proof of this theorem.

Lemma 1. *Let $\mathcal{P}$ be a $SQLP(\mathcal{R}, \mathcal{D})$ program and $S_\mathcal{R}(\mathcal{P})$ its transformed program according to Definition 6. Let t, s be two terms in $\mathcal{P}$'s signature and $d \in D \setminus \{\bot\}$. Then:*

1. $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} (t \sim s) \sharp d \Longrightarrow d \sqsubseteq \mathcal{R}(t, s)$
2. $\mathcal{R}(t, s) = d \Longrightarrow S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} (t \sim s) \sharp d$

Proof. We prove the two items separately.

1. Let T be a $QHL(\mathcal{D})$ proof tree witnessing
$$S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} (t \sim s) \sharp d$$
We prove by induction on number of nodes of T that $d \sqsubseteq \mathcal{R}(t, s)$. The basis case, with T consisting of just one node, must correspond to some inference without premises, i.e., a clause with empty body for $\sim$. Checking $P_\sim$ we observe that $X \sim X \leftarrow \top-$ is the only possibility. In this case t and s must be the same term and by the reflexivity of $\mathcal{R}$ (Def. 2), $\mathcal{R}(t, s) = \top$, which means $d \sqsubseteq \mathcal{R}(t, s)$ for every d. In the inductive step, we consider T with more than one node. Then the inference step at the root of T uses some clause $(c(\overline{X}_n) \sim c'(\overline{X}'_n) \leftarrow \top - pay_{\mathcal{R}(c,c')}, X_1 \sim X'_1, \ldots, X_n \sim X'_n) \in \mathcal{P}_\sim$, and must be of the form:
$$\frac{pay_w \sharp v \;\; (t_1 \sim s_1) \sharp e_1 \; \ldots \; (t_n \sim s_n) \sharp e_n}{c(\bar{t}_n) \sim c'(\bar{s}_n) \sharp d}$$
where $w = \mathcal{R}(c, c')$, $v \in \mathcal{D}$, $v \sqsubseteq w$, $t = c(\bar{t}_n)$, $s = c'(\bar{s}_n)$, and $e_1, \ldots, e_n$ s.t. $d \sqsubseteq \top \circ \bigsqcap\{v, e_1, \ldots, e_k\}$, i.e., $d \sqsubseteq \bigsqcap\{v, e_1, \ldots, e_k\}$. By induction hypothesis $e_i \sqsubseteq \mathcal{R}(t_i, s_i)$ for $i = 1 \ldots n$. Then $d \sqsubseteq \bigsqcap\{v, e_1, \ldots, e_n\}$ implies $d \sqsubseteq \bigsqcap\{w, \mathcal{R}(t_1, s_1), \ldots, \mathcal{R}(t_n, s_n)\}$ and hence $d \sqsubseteq \mathcal{R}(t, s)$ (Def. 3, item 3).
2. If $\mathcal{R}(t, s) = d$, $d \neq \bot$, we prove that $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} (t \sim s) \sharp d$ by induction on the syntactic structure of t. The basis corresponds to the case $t = c$ for some constant c, or $t = Y$ for some variable Y. If $t = c$ then $s = c'$ for some other constant c'. By Definition 6 there is a clause in $\mathcal{P}_\sim$ of the form $(c \sim c' \leftarrow \top - pay_d)$. Using this clause and the identity substitution we can write the root inference step of a proof for $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} (c \sim c') \sharp d$ as follows:
$$\frac{pay_d \sharp d}{c \sim c' \sharp d}$$
The condition required by the inference rule QMP($\mathcal{D}$) is in this particular case $d \sqsubseteq \top \circ \bigsqcap\{d\}$, and $\top \circ \bigsqcap\{d\} = d$. Proving the only premise $pay_d \sharp d$ in $QHL(\mathcal{D})$ is direct from its definition. If $t = Y$, with Y a variable, then $s = Y$ and $d = \top$ (otherwise $\mathcal{R}(t, s) = \bot$). Then $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} (Y \sim Y) \sharp \top$ can be proved by using the clause $(X \sim X \leftarrow \top-) \in \mathcal{P}_\sim$ with substitution $\theta = \{X \mapsto Y\}$.
In the inductive step, t must be of the form $c(\bar{t}_n)$, with $n \geq 1$, and then s must be of the form $c'(\bar{s}_n)$ (otherwise $\mathcal{R}(t, s) = \bot$). From $d = \mathcal{R}(t, s) \neq \bot$ (hypotheses of the lemma) and Definition 3 we have that $\mathcal{R}(c, c') \neq \bot$. Then, by Definition 6, there is a clause in $\mathcal{P}_\sim$ of the form:
$$c(\overline{X}_n) \sim c'(\overline{Y}_n) \leftarrow \top - pay_{\mathcal{R}(c,c')}, X_1 \sim Y_1, \ldots, X_n \sim Y_n$$
By using the substitution $\theta = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n, Y_1 \mapsto s_1, \ldots, Y_n \mapsto s_n\}$ we can write the root inference step in $QHL(\mathcal{D})$ as:
$$\frac{pay_{\mathcal{R}(c,c')} \sharp \mathcal{R}(c, c') \; (t_i \sim s_i \sharp \mathcal{R}(t_i, s_i))_{i=1\ldots n}}{c(\bar{t}_n) \sim c'(\bar{s}_n) \sharp d}$$
The inference can be applied because the condition
$$d \sqsubseteq \top \circ \bigsqcap\{\mathcal{R}(c, c'), \mathcal{R}(t_1, s_1), \ldots, \mathcal{R}(t_n, s_n)\}$$
reduces to
$$d \sqsubseteq \bigsqcap\{\mathcal{R}(c, c'), \mathcal{R}(t_1, s_1), \ldots, \mathcal{R}(t_n, s_n)\}$$
which holds by Definition 3, item 3. Moreover, the premises $t_i \sim s_i \sharp \mathcal{R}(t_i, s_i)$, $i = 1 \ldots n$, hold in $QHL(\mathcal{D})$ due to the inductive hypotheses, and proving
$$pay_{\mathcal{R}(c,c')} \sharp \mathcal{R}(c, c')$$
is straightforward from its definition. □

Now we can prove the equivalence between semantic inferences in $QHL(\mathcal{D})$ w.r.t. $\mathcal{P}$ and semantic inferences in $SQHL(\mathcal{R}, \mathcal{D})$ w.r.t. $S_\mathcal{R}(\mathcal{P})$.

Theorem 1. *Let $\mathcal{P}$ be a $SQLP(\mathcal{R}, \mathcal{D})$ program, A an atom in $\mathcal{P}$'s signature and $d \in D \setminus \{\bot\}$. Then:*

$$\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d \iff S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} A \sharp d \;.$$

Proof. Let T be a $SQHL(\mathcal{R},\mathcal{D})$ proof tree for some annotated atom $A \sharp d$ in $\mathcal{P}$'s signature witnessing $\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d$. We prove that $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} A \sharp d$ by induction on the number of nodes of T.

The inference step at the root of T must be of the form

$$\frac{B'_1 \sharp d_1 \quad \cdots \quad B'_k \sharp d_k}{A \sharp d} \quad (1)$$

with $((A \leftarrow e - B'_1, \ldots, B'_k), \delta) \in [C]_\mathcal{R}$ for some clause $C \equiv (H \leftarrow e - B_1, \ldots, B_k) \in \mathcal{P}$ (observe that the case $k = 0$ corresponds to the induction basis). By Definition 5, $A = H'\theta$, $B'_i = B_i\theta$ for some substitution θ and atom H' such that $\delta = \mathcal{R}(H_\ell, H') \sqcap \bigsqcap\{\mathcal{R}(X_i\theta, X_j\theta) \mid (X_i \sim X_j) \in \mathcal{S}_\ell\} \neq \bot$, with $\mathrm{lin}(H) = (H_\ell, \mathcal{S}_\ell)$. This means in particular that $w = \mathcal{R}(H_\ell, H') \neq \bot$, which by Definition 6 implies that there is a clause C' in $S_\mathcal{R}(\mathcal{P})$ of the form $C' \equiv (H' \leftarrow e - pay_w, S_\ell, B_1, \ldots B_k)$. Then the root inference step of the deduction proving $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} A \sharp d$ will use the inference rule QMP($\mathcal{D}$) with C' and substitution θ (such that $H'\theta = A$) as follows:

$$\frac{pay_w\theta \sharp w \quad ((u_i \sim v_i)\theta \sharp e_i)_{1 \le i \le m} \quad B'_1 \sharp d_1 \cdots B'_k \sharp d_k}{A \sharp d} \quad (2)$$

where $S_\ell = \{u_1 \sim v_1, \ldots, u_m \sim v_m\}$, and $e_i = \mathcal{R}(u_i\theta, v_i\theta)$ for $i = 1 \ldots m$.

Next we check that the premises can be proved from $S_\mathcal{R}(\mathcal{P})$ in $QHL(\mathcal{D})$:

- $pay_w\theta = pay_w$, since pay_w is a nullary predicate for every w. Therefore $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} pay_w \sharp w$ is immediate from the definition of pay_w in Definition 6.
- For each $1 \le i \le m$, we observe that $\mathcal{R}(u_i\theta, v_i\theta) \neq \bot$ because $\delta \neq \bot$ has been computed above as the infimum of a set including $\mathcal{R}(u_i\theta, v_i\theta)$ among its members. Then $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} (u_i \sim v_i)\theta$ holds by Lemma 1, item 2.
- For each $1 \le i \le k$, (1) shows that $\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} B'_i \sharp d_i$ with a proof tree having less nodes that T. Therefore, $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} B'_i \sharp d_i$ by induction hypothesis.

In order to perform the inference step (2), the QMP($\mathcal{D}$) inference rule also requires that $d \sqsubseteq e \circ \bigsqcap\{w, e_1 \ldots, e_m, d_1, \ldots, d_k\}$. This follows from the associativity of $\sqcap$ since:

- As defined above, $\delta = \mathcal{R}(H_\ell, H') \sqcap \bigsqcap\{\mathcal{R}(X_i\theta, X_j\theta) \mid (X_i \sim X_j) \in \mathcal{S}_\ell\}$, i.e. $\delta = w \sqcap \bigsqcap\{e_1 \ldots e_m\}$.
- By the SQMP($\mathcal{R},\mathcal{D}$) inference (1) we know that $d \sqsubseteq e \circ \bigsqcap\{\delta, d_1, \ldots, d_k\}$.

Let T be a $QHL(\mathcal{D})$ proof tree witnessing $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} A \sharp d$ for some atom A in $\mathcal{P}$'s signature. We prove by induction on the number of nodes of T that $\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d$.

Since A is in $\mathcal{P}$'s signature, the clause employed at the inference step at the root of T must be in the set $\mathcal{P}_S$ of Definition 6, and the inference step at the root of T have of the form of the inference (2) above. Hence this clause must have been constructed from a clause $C \equiv (H \leftarrow e - B_1, \ldots, B_k) \in \mathcal{P}$ and some atom H' such that $A = H'\theta$ and $\mathcal{R}(H_\ell, H') \neq \bot$, where $lin(H) = (H_\ell, S_\ell)$.

Then we can use C and θ to prove $\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d$ by a SQMP($\mathcal{R},\mathcal{D}$) inference like (1) using the $\mathcal{R}$-instance $C' \equiv A \leftarrow e - B'_1, \ldots, B'_k$ of C. The premises can be proved in $SQHL(\mathcal{R},\mathcal{D})$ by induction hypotheses, since all of them are also premises in (2). Finally, we must check that the conditions required by (1) hold: $(C', \delta) \in [C]_\mathcal{R}$ for some $\delta \in \mathcal{D}$, $\delta \neq \bot$ s.t. $d \sqsubseteq e \circ \bigsqcap\{\delta, d_1, \ldots, d_k\}$. This is true for $\delta = \bigsqcap\{w, e'_1, \ldots, e'_n\}$, with $e'_i = \mathcal{R}(u_i\theta, v_i\theta)$ for $i = 1 \ldots m$. Observe that in the premises of (2) we have $QHL(\mathcal{D})$ proofs of $u_i\theta \sim v_i\theta \sharp e_i$ for $i = 1 \ldots m$. Therefore $e_i \sqsubseteq e'_i$, by Lemma 1, item 1. Then

$$\begin{aligned} d \ &\sqsubseteq e \circ \textstyle\bigsqcap\{w, e_1 \ldots, e_m, d_1, \ldots, d_k\} && (\text{by}(2)) \\ &\sqsubseteq e \circ \textstyle\bigsqcap\{w, e'_1 \ldots, e'_m, d_1, \ldots, d_k\} && (e_i \sqsubseteq e'_i) \\ &= e \circ \textstyle\bigsqcap\{\delta, d_1, \ldots, d_k\} \end{aligned}$$

We must still prove that $\delta \neq \bot$. Observe that by the distributivity of $\circ$ w.r.t. $\sqcap$ (Def. 1, axiom 2.(e)):

$$e \circ \bigsqcap\{\delta, d_1, \ldots, d_k\} = (e \circ \delta) \sqcap (e \circ \bigsqcap\{d_1, \ldots, d_k\}) \ .$$

Therefore

$$d \sqsubseteq (e \circ \delta) \sqcap (e \circ \bigsqcap\{d_1, \ldots, d_k\})$$

and from $d \neq \bot$ we obtain $(e \circ \delta) \neq \bot$ which implies $\delta \neq \bot$ due to axiom 2.(c) in Definition 1. This completes the proof. □

4.2 Comparison to Related Approaches

Other program transformations have been proposed in the literature with the aim of supporting $\mathcal{R}$-based reasoning while avoiding explicit $\mathcal{R}$-based unification. Here we draw some comparisons between the program transformation $S_\mathcal{R}(\mathcal{P})$ presented in the previous subsection, the program transformations $H_\lambda(\mathcal{P})$ and $\mathcal{P}_\lambda$ proposed in [16], and the program transformation $\mathcal{P}_{E,\mathcal{R}}$ proposed in [11]. These three transformations are applied to a classical logic program $\mathcal{P}$ w.r.t. a fuzzy similarity relation $\mathcal{R}$ over symbols in the program's signature. Both $H_\lambda(\mathcal{P})$ and $\mathcal{P}_\lambda$ are classical logic programs to be executed by SLD resolution, and their construction depends on a fixed similarity degree $\lambda \in (0, 1]$. On the other hand, $\mathcal{P}_{E,\mathcal{R}}$ is a multi-adjoint logic program over a particular multi-adjoint lattice $\mathcal{G}$, providing the uncertain truth values in the interval $[0, 1]$ and two operators for conjunction and disjunction in the sense of Gödel's fuzzy logic (see [22] for technical details). As in the case of our own transformation $S_\mathcal{R}(\mathcal{P})$, the construction of $\mathcal{P}_{E,\mathcal{R}}$ does not depend on any fixed similarity degree. The transformation $S_\mathcal{R}(\mathcal{P})$ proposed in this paper is more general in that it can be applied to an arbitrary $SQLP(\mathcal{R},\mathcal{D})$ program $\mathcal{P}$, yielding a $QLP(\mathcal{D})$ program $S_\mathcal{R}(\mathcal{P})$ whose least Herbrand model is the same as that of $\mathcal{P}$.

We will restrict our comparisons to the case that $\mathcal{P}$ is chosen as a similarity-based logic program in the sense of [16]. As an illustrative example, consider the simple logic program $\mathcal{P}$ consisting of the following four clauses:

- $C_r: \ r(X, Y) \leftarrow p(X), \, q(Y), \, s(X, Y)$
- $C_p: \ p(c(U)) \leftarrow$
- $C_q: \ q(d(V)) \leftarrow$
- $C_s: \ s(Z, Z) \leftarrow$

Assume an admissible similarity relation defined by $\mathcal{R}(c, d) = 0.9$ and consider the goal $G : \leftarrow r(X, Y)$ for $\mathcal{P}$. Then, $\mathcal{R}$-based SLD-resolution as defined in [16] computes the answer substitution $\sigma = \{X \mapsto c(U), Y \mapsto d(U)\}$ with similarity degree 0.9. This computation succeeds because $\mathcal{R}$-based unification can compute the $m.g.u.$ $\{Z \mapsto c(U), V \mapsto U\}$ with similarity degree 0.9 to unify the two atoms $s(c(U), d(V))$ and $s(Z, Z)$. Let us now examine the behavior of the the transformed programs $H_{0.9}(\mathcal{P})$, $\mathcal{P}_{0.9}$, $S_\mathcal{R}(\mathcal{P})$ and $\mathcal{P}_{E,\mathcal{R}}$ and when working to emulate this computation without explicit use of a $\mathcal{R}$-based unification procedure.

1. $H_{0.9}(\mathcal{P})$ is defined in [16] as the set of all clauses C' such that $\mathcal{R}(C, C') \ge 0.9$ for some clause $C \in \mathcal{P}$. In this case $H_{0.9}(\mathcal{P})$ includes the four clauses of $\mathcal{P}$ and the two additional clauses $p(d(U)) \leftarrow$ and $q(c(V)) \leftarrow$, derived by similarity from C_p and C_q, respectively. Solving G w.r.t. $H_{0.9}(\mathcal{P})$ by means of

classical SLD resolution produces two possible answer substitutions, namely $\sigma_1 = \{X \mapsto c(U), Y \mapsto c(U)\}$ and $\sigma_2 = \{X \mapsto d(U), Y \mapsto d(U)\}$. They are both similar to σ to a degree greater or equal than 0.9, but none of them is σ itself, contrary to the claim in Proposition 7.1 (i) from [16]. Therefore, this Proposition seems to hold only in a somewhat weaker sense than the statement in [16]. This problem is due to the possible non-linearity of a clause's head, which is properly taken into account by our transformation $S_{\mathcal{R}}(\mathcal{P})$.

2. According to [16], $\mathcal{P}_{0.9}$ is computed from $\mathcal{P}$ by replacing all the constructor and predicate symbols by new symbols that represent the equivalence classes of the original ones modulo $\mathcal{R}$-similarity to a degree greater or equal than 0.9. In our example these classes are $\{r\}$, $\{p\}$, $\{q\}$, $\{s\}$ and $\{c, d\}$, that can be represented by the symbols r, p, q, s an e, respectively. Then, $\mathcal{P}_{0.9}$ replaces the two clauses C_p and C_q by $p(e(U)) \leftarrow$ and $q(e(V)) \leftarrow$, respectively, leaving the other two clauses unchanged. Solving G w.r.t. $\mathcal{P}_{0.9}$ by means of classical SLD resolution produces the answer substitution $\sigma' = \{X \mapsto e(U), Y \mapsto e(U)\}$, which corresponds to σ modulo the replacement of the symbols in the original program by their equivalence classes. This is consistent with the claims in Proposition 7.2 from [16].

3. Note that $\mathcal{P}$ can be trivially converted into a semantically equivalent a $SQLP(\mathcal{R},\mathcal{U})$ program, just by replacing each occurrence of the implication sign $\leftarrow$ in $\mathcal{P}$'s clauses by $\leftarrow 1.0 -$. Then $S_{\mathcal{R}}(\mathcal{P})$ can be built as a $QLP(\mathcal{U})$ program by the method explained in Subsection 4.1. It includes three clauses corresponding to C_r, C_p and C_q of $\mathcal{P}$ plus the following three new clauses:
   - $C'_p :\ p(d(U)) \leftarrow 1.0 -\ pay_{0.9}$
   - $C'_q :\ q(c(V)) \leftarrow 1.0 -\ pay_{0.9}$
   - $C'_s :\ s(Z_1, Z_2) \leftarrow 1.0 -\ Z_1 \sim Z_2$

   where C'_p resp. C'_q come from replacing the linear heads of C_p resp. C_q by similar heads, and C_s comes from linearizing the head of C_s, which allows no replacements by similarity. $S_{\mathcal{R}}(\mathcal{P})$ includes also the proper clauses for $\mathcal{P}_{\sim}$ and $\mathcal{P}_{\mathrm{pay}}$, in particular the following three ones:
   - $I :\ X \sim X \leftarrow 1.0 -$
   - $S :\ c(X_1) \sim d(Y_1) \leftarrow 1.0 -\ pay_{0.9}, X_1 \sim Y_1$
   - $P :\ pay_{0.9} \leftarrow 0.9 -$

   Solving goal G w.r.t. $S_{\mathcal{R}}(\mathcal{P})$ by means of the $\mathcal{U}$-qualified SLD resolution procedure described in [14] can compute the answer substitution σ with qualification degree 0.9. More precisely, the initial goal can be stated as $r(X, Y)\#W \ [\!] \ W \geq 0.9$, and the computed answer is $(\sigma, \{W \mapsto 0.9\})$. The computation emulates $\mathcal{R}$-based unification of $s(c(U), c(V))$ and $s(Z, Z)$ to the similarity degree 0.9 by solving $s(c(U), c(V))$ with the clauses C'_s, I, S and P.

4. The semantics of the *MALP* framework depending on the chosen multi-adjoint lattice is presented in [11]. A comparison with the semantics of the $QLP(\mathcal{D})$ scheme (see [14] and Subsection 3.1 above) shows that *MALP* programs over the multi-adjoint lattice $\mathcal{G}$ behave as $QLP(\mathcal{U}')$ programs, where $\mathcal{U}'$ is the quasi qualification domain analogous to $\mathcal{U}$ introduced at the end of Subsection 2.1 above. For this reason, we can think of the transformed program $\mathcal{P}_{E,\mathcal{R}}$ as presented with he syntax of a $QLP(\mathcal{U}')$ program. The original program $\mathcal{P}$ can also be written as a $QLP(\mathcal{U}')$ program just by replacing each the implication sign $\leftarrow$ occurring in $\mathcal{P}$ by $\leftarrow 1.0 -$. As explained in [11], $\mathcal{P}_{E,\mathcal{R}}$ is built by extending $\mathcal{P}$ with clauses for a new binary predicate $\sim$ intended to emulate the behaviour of $\mathcal{R}$-based unification between terms. In our example, $\mathcal{P}_{E,\mathcal{R}}$ will include (among others) the following clause for $\sim$:
   - $S' :\ c(X_1) \sim d(Y_1) \leftarrow 0.9 -\ X_1 \sim Y_1$

   In comparison to the clause S in $S_{\mathcal{R}}(\mathcal{P})$, clause S' needs no call to a $pay_{0.9}$ predicate at its body, because the similarity degree $0.9 = \mathcal{R}(c, d)$ can be attached directly to the clause's implication. This difference corresponds to the different interpretations of $\circ$, which behaves as $\times$ in $\mathcal{U}$ and as min in $\mathcal{U}'$.

   Moreover, $\mathcal{P}_{E,\mathcal{R}}$ is defined to include a clause of the following form for each pair of n-ary predicate symbols pd and pd' such that $\mathcal{R}(pd, pd') \neq 0$:
   - $C_{pd,pd'} :\ pd(Y_1, \ldots, Y_n) \leftarrow \mathcal{R}(pd, pd') -$ $pd'(X_1, \ldots, X_n), X_1 \sim Y_1, \ldots, X_n \sim Y_n$

   In our simple example, all the clauses of this form correspond to the trivial case where pd and pd' are the same predicate symbol and $\mathcal{R}(pd, pd') = 1.0$. Solving goal G w.r.t.$S_{\mathcal{R}}(\mathcal{P})$ by means of the procedural semantics described in Section 4 of [11] can compute the answer substitution σ to the similarity degree 0.9. More generally, Theorem 24 in [11] claims that for any choice of $\mathcal{P}$, $\mathcal{P}_{E,\mathcal{R}}$ can emulate any successful computation performed by $\mathcal{P}$ using $\mathcal{R}$-based SLD resolution.

In conclusion, the main difference between $S_{\mathcal{R}}(\mathcal{P})$ and $\mathcal{P}_{E,\mathcal{R}}$ pertains to the techniques used by both program transformations in order to emulate the effect of replacing the head of a clause in the original program by a similar one. $\mathcal{P}_{E,\mathcal{R}}$ always relies on the clauses of the form $C_{pd,pd'}$ *and* the clauses for $\sim$, while $S_{\mathcal{R}}(\mathcal{P})$ can avoid to use the clauses for $\sim$ as long as all the clauses involved in the computation have linear heads. In comparison to the two transformations $H_\lambda(\mathcal{P})$ and $\mathcal{P}_\lambda$, our transformation $S_{\mathcal{R}}(\mathcal{P})$ does not depend on a fixed similarity degree λ and does not replace the atoms in clause bodies by similar ones.

4.3 A Goal Solving Example

In order to illustrate the use of the transformed program $S_{\mathcal{R}}(\mathcal{P})$ for golving goals w.r.t. the original program $\mathcal{P}$, we consider the case where $\mathcal{P}$ is the $SQLP(\mathcal{R},\mathcal{U})$ program displayed in Figure 1. The transformed program $S_{\mathcal{R}}(\mathcal{P})$ obtained by applying Definition 6 is shown in Figure 2. The following observations are useful to understand how the transformation has worked in this simple case:

- The value $\top$ in the domain $\mathcal{U}$ corresponds to the real number 1 and hence by reflexivity $\mathcal{R}(A, A) = 1$ for any atom in the signature of the program. Therefore, and as a consequence of Definition 6, every clause in the original program gives rise to a clause in the transformed program with the same head and with the same body except for a new, first atom $pay_{1.0}$. For instance, clauses 1, 2 and 3 in Figure 2 correspond to the same clause numbers in Figure 1.
- Apart of the clauses corresponding directly to the original clauses, the program of Figure 2 contains new clauses obtained by similarity with some clause heads in the original program. For instance, lines 4 and 5 are obtained by similarity with clauses at lines 1 and 2 in the original program, respectively. The subindexes at literal pay correspond to $\mathcal{R}(\texttt{lynx}, \texttt{cat}) = 0.8$, $\mathcal{R}(\texttt{boar}, \texttt{pig}) = 0.7$, respectively.
- Analogously, for instance the clause at line 10 (with head `farm(lynx)`) is obtained by head-similarity with the clause of line 6 in the $SQLP(\mathcal{R},\mathcal{U})$ program (head `domestic(cat)`),

```
1  wild(lynx) <-0.9- $pay_{1.0}$
2  wild(boar) <-0.9- $pay_{1.0}$
3  wild(snake) <-1.0- $pay_{1.0}$
4  wild(cat) <-0.9- $pay_{0.8}$
5  wild(pig) <-0.9- $pay_{0.7}$

6  farm(cow) <-1.0- $pay_{1.0}$
7  farm(pig) <-1.0- $pay_{1.0}$
8  farm(boar) <-1.0- $pay_{0.7}$
9  farm(cat) <-0.8- $pay_{0.3}$
10 farm(lynx) <-0.8- $pay_{0.3}$
11 farm(snake) <-0.4- $pay_{0.3}$

12 domestic(cat) <-0.8- $pay_{1.0}$
13 domestic(snake) <-0.4- $pay_{1.0}$
14 domestic(lynx) <-0.8- $pay_{0.8}$
15 domestic(cow) <-1.0- $pay_{0.3}$
16 domestic(pig) <-1.0- $pay_{0.3}$
17 domestic(boar) <-1.0- $pay_{0.3}$

18 intelligent(A) <-0.9- $pay_{1.0}$,domestic(A)
19 intelligent(lynx) <-0.7- $pay_{1.0}$
20 intelligent(cat) <-0.7- $pay_{0.8}$

21 pacific(A) <-0.9- $pay_{1.0}$,domestic(A)
22 pacific(A) <-0.7- $pay_{1.0}$,farm(A)

23 pet(A) <-1.0- $pay_{1.0}$,pacific(A),intelligent(A)

24 $pay_{1.0}$ <-1.0-
25 $pay_{0.8}$ <-0.8-
26 $pay_{0.7}$ <-0.7-
27 $pay_{0.3}$ <-0.3-
```

Figure 2. Example of transformed program. (Note: no clauses for $\sim$ are needed because the original program was left-linear).

and the subindex at pay is obtained from

$$\begin{aligned} &\mathcal{R}(\mathtt{domestic(cat)}, \mathtt{farm(lynx)}) &=\\ &\mathcal{R}(\mathtt{domestic}, \mathtt{farm}) \sqcap \mathcal{R}(\mathtt{cat}, \mathtt{lynx}) &=\\ &0.3 \sqcap 0.8 &=\\ &0.3 \end{aligned}$$

- There is no clause for predicate $\sim$ since all the heads in the original program were already linear and therefore $\mathcal{P}_{\sim}$ can be left empty in practice.
- The clauses for pay correspond to the fragment $\mathcal{P}_{\mathrm{pay}}$ in Definition 6.

In the rest of this subsection, we will show an execution for the goal `pet(A)#W | W >= 0.50` over the program $S_{\mathcal{R}}(\mathcal{P})$ (see Figure 2) with the aim of obtaining all those animals that could be considered a `pet` for at least a qualification value of 0.50.

We are trying this execution in the prototype developed along with [14] for the instances $QLP(\mathcal{U})$ and $QLP(\mathcal{W})$. Although this prototype hasn't been released as an integrated part of $\mathcal{TOY}$, you can download[1] the prototype to try this execution. Please notice that the prototype does not automatically do the translation process from a given $SQLP(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$ to its transformed program $S_{\mathcal{R}}(\mathcal{P})$, because it was developed mainly for [14]. Therefore, the transformed program shown in Figure 2 has been computed manually.

We will start running $\mathcal{TOY}$ and loading the $QLP(\mathcal{U})$ instance with the command `/qlp(u)`:

```
Toy> /qlp(u)
```

this will have the effect of loading the *Real Domain Constraints library* and the $QLP(\mathcal{U})$ library into the system, the prompt `QLP(U)>` will appear. Now we have to compile our example program (assume we have it in a text file called `animals.qlp` in `C:/examples/`) with the command `/qlptotoy` (this command will behave differently based on the actual instance loaded).

```
QLP(U)> /qlptotoy(c:/examples/animals)
```

Note that we didn't write the extension of the file because it *must* be `.qlp`. This will create the file `animals.toy` in the same directory as our former file. And this one will be an actual $\mathcal{TOY}$ program. We run the program with `/run(c:/examples/animals)` (again without the extension –although this time we are assuming `.toy` as extension–) and we should get the following message:

```
PROCESS COMPLETE
```

And finally we are set to launch our goal with the command `/qlpgoal`. The solutions found for this program and goal are:

```
QLP(U)> /qlpgoal(pet(A)#W | W>=0.50)
      { A -> cat,
        W -> 0.5599999999999999 }

sol.1, more solutions (y/n/d/a) [y]?
      { A -> cat,
        W -> 0.7200000000000001 }

sol.2, more solutions (y/n/d/a) [y]?
      { A -> lynx,
        W -> 0.5760000000000002 }

sol.3, more solutions (y/n/d/a) [y]?
      { A -> lynx,
        W -> 0.5760000000000002 }

sol.4, more solutions (y/n/d/a) [y]?
      no
```

At this point and if you remember the inference we did in Example 2 for `pet(lynx)#0.50`, we have found a better solution (as you can see there are two solutions for `lynx`, and this is due to the two different ways of proving `intelligent(lynx)`: `intelligent(lynx)#0.7` using clause 19, and `intelligent (lynx)#0.576` using clauses 18 and 14.

5. Conclusions

Similarity-based LP has been proposed in [16] and related works to enhance the LP paradigm with a kind of approximate reasoning which supports flexible information retrieval applications, as argued in [8, 11]. This approach keeps the syntax for program clauses as in classical LP, and supports uncertain reasoning by using a fuzzy similarity relation $\mathcal{R}$ between symbols in the program's signature. We have shown that similarity-based LP as presented in [16] can be reduced to Qualified LP in the $QLP(\mathcal{D})$ scheme introduced in [14], which supports logic programming with attenuated program clauses over a parametrically given domain $\mathcal{D}$ whose

[1] Available at: `http://gpd.sip.ucm.es/cromdia/qlpd`. There you will also find specific instructions on how to install and run it as well as text files with the program examples tried in here.

elements qualify logical assertions by measuring their closeness to various users' expectations. Using generalized similarity relations taking values in the carrier set of an arbitrarily given qualification domain $\mathcal{D}$, we have extended $QLP(\mathcal{D})$ to a more expressive scheme $SQLP(\mathcal{R}, \mathcal{D})$ with two parameters, for programming modulo $\mathcal{R}$-similarity with $\mathcal{D}$-attenuated Horn clauses. We have presented a declarative semantics for $SQLP(\mathcal{R}, \mathcal{D})$ programs and a semantics-preserving program transformation which embeds $SQLP(\mathcal{R}, \mathcal{D})$ into $QLP(\mathcal{D})$. As a consequence, the sound and complete procedure for solving goals in $QLP(\mathcal{D})$ by $\mathcal{D}$-qualified SLD resolution and its implementation in the $\mathcal{TOY}$ system [14] can be used to implement $SQLP(\mathcal{R}, \mathcal{D})$ computations via the transformation.

Our framework is quite general due to the availability of different qualification domains, while the similarity relations proposed in [16] take fuzzy values in the interval $[0, 1]$. In comparison to the multi-adjoint framework proposed in [11], the $QLP(\mathcal{D})$ and $SQLP(\mathcal{R}, \mathcal{D})$ schemes have a different motivation and scope, due to the differences between multi-adjoint algebras and qualification domains as algebraic structures. In contrast to the goal solving procedure used in the multi-adjoint framework, $\mathcal{D}$-qualified SLD resolution does not rely on costly computations of reductant clauses and has been efficiently implemented.

As future work, we plan to investigate an extension of the $\mathcal{R}$-based SLD resolution procedure proposed in [16] to be used within the $SQLP(\mathcal{R}, \mathcal{D})$ scheme, and to develop an extension of this scheme which supports lazy functional programming and constraint programming facilities. The idea of similarity-based unification has been already applied in [12] to obtain an extension of *needed narrowing*, the main goal solving procedure of functional logic languages. As in the case of [16], the similarity relations considered in [12] take fuzzy values in the real interval $[0, 1]$.

Acknowledgments

The authors have been partially supported by the Spanish National Projects MERIT-FORMS (TIN2005-09027-C03-03) and PROMESAS–CAM (S-0505/TIC/0407).

References

[1] K.R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 493-574. Elsevier and The MIT Press, 1990.

[2] K.R. Apt and M.H. van Emden. Contributions to the theory of logic programming. *Journal of the Association for Computing Machinery (JACM)*, 29(3):841-862, 1982.

[3] F. Arcelli and F. Formato. Likelog: A logic programming language for flexible data retrieval. In *Proceedings of the 1999 ACM Symposium on Applied Computing (SAC'99)*, pages 260-267, New York, NY, USA, 1999. ACM Press.

[4] P. Arenas, A.J. Fernández, A. Gil, F.J. López-Fraguas, M. Rodríguez-Artalejo and F. Sáenz-Pérez. $\mathcal{TOY}$, a multiparadigm declarative language. Version 2.3.1, 2007. R. Caballero and J. Sánchez (Eds.), available at `http://toy.sourceforge.net`.

[5] F. Formato, G. Gerla and M.I. Sessa. Similarity-based unification. *Fundamenta Informaticae*, 41(4):393-414, 2000.

[6] G. Gerla and M.I. Sessa. Similarity in logic programming. In G. Chen, M. Ying and K. Cai, editors, *Fuzzy Logic and Soft Computing*, pages 19-31. Kluwer Academic Publishers, 1999.

[7] M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programs and their applications. *Journal of Logic Programming*, 12(3&4):335-367, 1992.

[8] V. Loia, S. Senatore and M.I. Sessa. Similarity-based SLD resolution and its role for web knowledge discovery. *Fuzzy Sets and Systems*, 144(1):151-171, 2004.

[9] J. Medina, M. Ojeda-Aciego and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. In T. Eiter, W. Faber and M. Truszczyinski, editors, *Logic Programming and Non-Monotonic Reasoning (LPNMR'01)*, volume 2173 of *LNAI*, pages 351-364. Springer-Verlag, 2001.

[10] J. Medina, M. Ojeda-Aciego and P. Vojtáš. A procedural semantics for multi-adjoint logic programming. In P. Brazdil and A. Jorge, editors, *Progress in Artificial Intelligence (EPIA'01)*, volume 2258 of *LNAI*, pages 290-297. Springer-Verlag, 2001.

[11] J. Medina, M. Ojeda-Aciego and P. Vojtáš. Similarity-based unification: A multi-adjoint approach. *Fuzzy Sets and Systems*, 146:43-62, 2004.

[12] G. Moreno and V. Pascual. Programming with fuzzy logic and mathematical functions. In A.P.I. Bloch and A. Tettamanzi, editors, *Proceedings of the 6th International Workshop on Fuzzy Logic and Applications (WILF'05)*, volume 3849 of *LNAI*, pages 89-98. Springer-Verlag, 2006.

[13] M. Rodríguez-Artalejo and C.A. Romero-Díaz. A generic scheme for qualified logic programming (Technical Report SIC-1-08). Technical Report, Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain, 2008.

[14] M. Rodríguez-Artalejo and C.A. Romero-Díaz. Quantitative logic programming revisited. In J. Garrigue and M. Hermenegildo, editors, *Functional and Logic Programming (FLOPS'08)*, volume 4989 of *LNCS*, pages 272-288. Springer-Verlag, 2008.

[15] M.I. Sessa. Translations and similarity-based logic programming. *Soft Computing*, 5(2), 2001.

[16] M.I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275(1&2):389-426, 2002.

[17] V.S. Subrahmanian. On the semantics of quantitative logic programs. In *Proceedings of the 4th IEEE Symposium on Logic Programming*, pages 173-182, San Francisco, 1987.

[18] V.S. Subrahmanian. Query processing in quantitative logic programming. In *Proceedings of the 9th International Conference on Automated Deduction*, volume 310 of *LNCS*, pages 81-100, London, UK, 1988. Springer-Verlag.

[19] V.S. Subrahmanian. Uncertainty in logic programming: Some recollections. *Association for Logic Programming Newsletter*, 20(2), 2007.

[20] M.H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37-53, 1986.

[21] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machinery (JACM)*, 23(4):733-742, 1976.

[22] P. Vojtáš. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124:361:370, 2001.