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 \( R \) between symbols in the program’s signature, while keeping the syntax for program clauses as in classical LP. Another recent proposal is the QLP\((D)\) 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 \( D \) whose elements qualify logical assertions by measuring their closeness to various users’ expectations. In this paper we propose a more expressive scheme QAQLP\((R, D)\) which subsumes both SLP and QLP\((D)\) as particular cases. We also show that SQLP\((R, D)\) programs can be transformed into semantically equivalent QLP\((D)\) programs. As a consequence, existing QLP\((D)\) 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; E.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 \( A ← d − \overline{B} \), 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 \( T \) in place of the lattice of the real numbers in the interval \([0, 1] \) under their natural ordering. \( T \) includes two isomorphic copies of \([0, 1] \) whose elements are incomparable under \( T \)’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 \( R \) (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 \( R \)-based SLD-resolution procedure relying on \( R \)-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 \( R \)-based SLD resolution w.r.t. a given logic program \( P \) and the answers computed by classical SLD resolution w.r.t. the two transformed programs \( H_\lambda(P) \) (built by adding to \( P \) new clauses \( R \)-similar to those in \( P \) up to the degree \( \lambda \in (0, 1] \)) and \( \overline{P}_\lambda \) (built by replacing all the function and predicate symbols in \( P \) by new symbols that represent equivalence classes modulo \( R \)-similarity up to \( \lambda \)). 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\((D)\), a programming scheme for Qualified LP over a parametrically given Qualification Domain \( D \) 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\((D)\) use \( \mathcal{D} \)-attenuated clauses of the form \( A ← d − \overline{B} \) where \( A \) is an atom, \( \overline{B} \) a finite conjunction of atoms and \( d \in D \setminus \{\bot\} \) is the attenuation value.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

\( \text{PPDP'08, July 15–17, 2008, Valencia, Spain.} \)

Copyright \( \text{c 2008 ACM 978-1-60558-117-0/08/07. . .$5.00} \)
attached to the clause’s implication, used to propagate to the head the qualification value \( d \circ b \), where \( b \) is the infimum in \( D \) 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 \( D \). 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\((D)\) scheme, and potentially useful instances of the scheme have been implemented on top of the Constraint Functional Logic Programming (CFLP) system TOY\(^3\) [4].

The results presented in this paper can be summarized as follows: we consider generalized similarity relations over a set \( S \) as mappings \( R : S \times S \rightarrow D \) taking values in the carrier set \( D \) of an arbitrarily given qualification domain \( D \), and we extend QLP\((D)\) to a more expressive scheme SLP\((R, D)\) with two parameters for programming modulo \( R \)-similarity with \( D \)-attenuated Horn clauses. We present a declarative semantics for SLP\((R, D)\) and a program transformation mapping each SLP\((R, D)\) program \( P \) into a QLP\((D)\) program \( S_R(P) \) whose least Herbrand model corresponds to that of \( P \). Roughly, \( S_R(P) \) is built adding to \( P \) new clauses obtained from the original clauses in \( P \) by computing various new heads \( \sim \) similar to a linearized version of the original head, adding also \( R \)-similarity conditions \( X_i \sim X_j \) to the body and suitable clauses for the new predicate \( \sim \) to emulate \( R \)-based unification. Thanks to the \( S_R(P) \) transformation, the sound and complete procedure for solving goals in SLP\((R, D)\) has been extended to the SLD resolution and its implementation in the TOY\(^3\) system [14] can be used to implement SLP\((R, D)\) 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 agregation of atoms in rule bodies. In comparison to the QLP\((D)\) 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\((D)\) 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\((D)\), 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 \( P \) can be transformed into a MALP program \( P_{E,R} \) which can be executed using only syntactical unification and emulates the successful computations of \( P \) using the SLD resolution with \( R \)-based unification introduced in [16]. \( P_{E,R} \) 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]. \( P_{E,R} \) also introduces clauses for a binary predicate \( \sim \) which emulates \( R \)-based unification, as in our transformation \( S_R(P) \). Nevertheless, \( S_R(P) \) is defined for a more general class of programs and uses the \( R \)-similarity predicate \( \sim \) only if the source program \( P \) has some clause whose head is non-linear. More detailed comparisons between the program transformations \( S_R(P), H_x(P), P, \) and \( P_{E,R} \) will be given in Subsection 4.2.

The rest of the paper is structured as follows: In Section 2 we recall the qualification domains \( D \) first introduced in [14] and we define similarity relations \( R \) over an arbitrary qualification domain.

In Section 3 we recall the scheme QLP\((D)\) and we introduce its extension SLP\((R, D)\) with its declarative semantics, given by a logical calculus which characterizes the least Herbrand model \( M_P \) of each SLP\((R, D)\) program \( P \). In Section 4 we define the transformation \( S_R(P) \) of any given SLP\((R, D)\) program \( P \) into a QLP\((D)\) program \( S_R(P) \) such that \( M_P(S_R(P)) = M_P \), we give some comparisons to previously known program transformations, and we illustrate the application of \( S_R(P) \) 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 \( D = (D, \sqcup, \sqcap, \top, \circ, \sim) \) verifying the following requirements:

1. \((D, \sqcup, \sqcap, \top)\) 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 \( d \equiv e \) 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 \( d \equiv e \) as abbreviation for \( d \sqsubseteq e \land e \sqsubseteq d \).

2. \( \circ : D \times D \rightarrow D \), 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) \( \forall d \in D : d \circ \bot = d \).
   (d) \( \forall d, e \in D : d \circ (e \sqcap f) = (d \circ e) \sqcap (d \circ f) \).
   (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, \( D \) 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 \( \sqcap S \)) 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 \( d \circ \top = \sqcap \{d \circ e \mid e \in S \} \).

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. \( B = \{(0, 1), \leq, 0, 1, \wedge\} \), 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. \( U = (U, \leq, 0, 1, \times) \), 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 \( \sqcap S \) 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 U is due to the interpretation of \( c \), as min and \( \circ \) as \( \times \).

3. \( W = (P, \geq, \infty, 0, +) \), 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
∞ + d = d + ∞ = ∞ for any d ∈ P). In this domain, the
top element ⊤ is 0 and the greatest lower bound ⋃ S of a finite
S ⊆ P is the maximum value max(S), which is 0 if S = ⌀.
Attaching an element d ∈ P ∪ {∞} 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 W
is due to the interpretation of ⋃ as max and ◦ ⋆.
4. Given 2 qualification domains D1 = ⟨D1, ⊑, ⊔, ⊤, ⊥, σ⟩ (i ∈
{1, 2}), their cartesian product D1 × D2 is D = def ⟨D, ⊑,
⊔, ⊤, ⊥, σ⟩, where D = def D1 × D2, the partial ordering ⊑ is
defined as (d1, d2) ⊑ (e1, e2) ⋆ def d1 ⊑ e1 and
d2 ⊑ e2. ⊔ = def (⊔1, ⊔2), ⊤ = def (⊔1, ⊔2), and the
attenuation operator ◦ is defined as (d1, d2) ◦ (e1, e2) = def
(d1 ⊓ e1, d2 ⊓ e2). The product of two given qualification
domains is always another qualification domain, as proved
in [14]. Intuitively, each value (d1, d2) belonging to D1 × D2
imposes the qualification d1 and also the qualification d2.
For instance, values (c, d) belonging to U × 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 U′ resp. W′ defined analogously
to U resp. W, except that ◦ behaves as min in U′ and as max
in W′. Note that almost all the axioms for qualification domains
enumerated in Definition 1 hold in U′ and W′, except that axiom
2.(d) holds only in the relaxed form ∀d,e ∈ D′ : d ⋆ e ⊆ e. Therefore, we will refer to U′ and 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 R : S × S → [0, 1] that satisfy
three axioms analogous to those required for classical equivalence
relations. Each value R(x, y) computed by a similarity relation 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 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 D with carrier set D and
a set S be given.

1. A D-valued similarity relation over S is any mapping R : S × S → D
such that the following three axioms hold for all x, y, z ∈ S:
(a) Reflexivity: R(x, x) = ⊤.
(b) Symmetry: R(x, y) = R(y, x).
(c) Transitivity: R(x, z) ⊑ R(x, y) ⊑ R(y, z).

2. The mapping R : S × S → D is defined as R(x, x) = ⊤ for all
x ∈ S and R(x, y) = ⊥ for all x, y ∈ D; x ≠ y is trivially a
D-valued similarity relation called the identity.

3. A D-valued similarity relation R over S is called admissible
iff S = ∀ar ∪ CS ∪ PS (where the three mutually disjoint
sets ∀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) R restricted to ∀ar behaves as the identity, i.e. R(X, X) = ⊤
for all X ∈ ∀ar and R(X, Y) = ⊥ for all X, Y ∈ ∀ar,
X ≠ Y.
(b) R(x, y) ≠ ⊥ holds only if some of the following three cases
holds x ⊓ y: either x, y ∈ ∀ar are both the same
variable; or else x, y ∈ CS are constructor symbols with
the same arity; or else x, y ∈ PS are predicate symbols
with the same arity.

The similarity degrees computed by a D-valued similarity relation
must be interpreted w.r.t. the intended role of D-elements as
qualification values. For example, let R be an admissible similarity
relation, and let c, d ∈ CS be two nullary constructor symbols
(i.e., constants). If R is U-valued, then R(c, d) can be interpreted
as a certainty degree for the assertion that c and d are similar.
On the other hand, if R is W-valued, then 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 ⋃ and
ⓡ in U and W, respectively.

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

Definition 3. (R acting over terms).

1. For X ∈ ∀ar and for any term t different from X:
R(X, t) = ⊤ and R(t, X) = ⊥.

2. For c, c′ ∈ CS with different arities n, m:
R(c(t1, ..., tn), c′(t1′, ..., tn′)) = ⊥.

3. For c, c′ ∈ CS with the same arity n:
R(c(t1, ..., tn), c′(t1′, ..., tn′)) = R(c, c′) ⋆ R(t1, t1′) ⋆ ... ⋆ R(tn, tn′).

3. Similarity-based Qualified Logic Programming

In this section we extend our previous scheme QLP(D) to a
more expressive scheme called Similarity-based Qualified Logic Programming
over (R, D) abbreviated as SQLP(R, D) – which
supports both qualification over D in the sense of [14] and 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(D) already presented in [14], while
the extensions needed to conform the new SQLP(R, D) scheme
are presented in subsection 3.2.

3.1 Qualified Logic Programming

QLP(D) was proposed in our previous work [14] as a generic
scheme for qualified logic programming over a given qualification
domain 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
∀ar (disjoint from Σ) and Atoms are of the form p(t1, ..., tn)
(shortened as p(tn) or simply p(t)) where p is a n-ary predicate
symbol and t is a term. We write AtΣ, called the open Herbrand base,
for the set of all atoms. A QLP(D) program P is a finite set
of D-qualified definite Horn clauses of the form A ← d − d− B
where A is an atom, B a finite conjunction of atoms and d ∈ D \ {⊥} 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, D-qualified atoms
(A ^ d where A is an atom and d ∈ D \ {⊥}) are introduced to
represent the statement that the atom A holds for at least the quali-
fication value d. For use in goals to be solved, open D-annotated
atoms (A ^ W where A is an atom and W a qualification
value intended to take values over D) are also introduced, and a
countably infinite set ∀ar of qualification variables (disjoint from ∀ar
and Σ) is postulated. The annotated Herbrand base over D is
defined as the set AtΣ(D) of all D-qualified atoms. A D-entailment
relation over AtΣ(D), defined as A ^ d ⊨ A’ ^ d’ iff there is some
substitution θ such that A’ = Aθ and d’ ⊑ d, is used to for-
mally define an open Herbrand interpretation over $D$ — from now on just an interpretation — as any subset $I \subseteq A_{\Sigma}(D)$ which is closed under $D$-entailment. We write $\text{Int}_{\Sigma}(D)$ for the family of all interpretations. The notion of model is such that given any clause $C \subseteq A \rightarrow d - B_1, \ldots, B_n$ in the QLP($D$) program $P$, an interpretation $I$ is said to be a model of $C$ iff any substitution $\theta$ and any qualification values $d_1, \ldots, d_k \in D \setminus \{\bot\}$ such that $B, \theta \models d_i \in I$ for all $1 \leq i \leq k$, one has $A \models d \in \prod\{d_1, \ldots, d_k\} \in I$. The interpretation $I$ is also said to be a model of the QLP($D$) program $P$ (written as $I \models P$) iff it happens to be a model of every clause in $P$.

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

The logic QHL($D$) is defined as a deductive system consisting just of one inference rule: QMP($D$), called Qualified Modus Ponens over $D$. Such rule allows us to give the following inference step given that there were some $(A \rightarrow d - B_1, \ldots, B_n) \in P$, some substitution $\theta$ 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 $d' \models d \in \prod\{d_1, \ldots, d_k\}$:

$$A' \models d'$$

Roughly, each QMP($D$) inference step using an instance of a program clause $A \rightarrow d - B$ has the effect of propagating to the head the qualification value $d$ over $b$, where $b$ is the inductive in $D$ 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 $U$ and $W$ for qualifying logical assertions. We use the notations $P \models_{\text{QHL}(D)} A \models d$ (resp. $P \models_{\text{QHL}(D)} A \models d$) to indicate that $A \models d$ can be inferred from the clauses in program $P$ in finitely many steps (resp. n steps). The least Herbrand model of $P$ happens to be $M_P = \{A \models d \mid P \models_{\text{QHL}(D)} A \models d\}$, as proved in [14].

### 3.2 Similarity-based Qualified Logic Programming

The scheme $\text{SQLP}(\mathcal{R}, D)$ presented in this subsection has two parameters $\mathcal{R}$ and $D$, where $D$ can be any qualification domain and $\mathcal{R}$ can be any admissible $D$-value similarity relation, in the sense of Definition 2. The new scheme subsumes the approach in [14] by behaving as QLP($D$) 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 $D$ is chosen as $U$.

Syntactically, SQLP($\mathcal{R}, D$) presents almost no changes w.r.t. QLP($D$), 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 $\Sigma$ providing again constructor and predicate symbols. Terms and Atoms are built the same way they were in QLP($D$), and $A_{\Sigma}$ 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($\mathcal{R}, D$) program $P$ is a finite set of $D$-qualified definite Horn clauses with the same syntax as in QLP($D$), along with a $D$-valued admissible similarity relation $\mathcal{R}$ in the sense of Definition 2, item 2. Figure 1 shows a simple SQLP($\mathcal{R}, U$) program built from the similarity relation $\mathcal{R}$ given in the same figure and the qualification domain $U$ 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.

$D$-qualified atoms $(A \models d$ with $A$ an atom and $d \in D \setminus \{\bot\})$ and open $D$-annotated atoms $(A \models W$ with $A$ and $W$ a qualification variable intended to take values in $D \setminus \{\bot\}$) will still be used here. Similarly, the annotated open Herbrand base over $D$ is again defined as the set $A_{\Sigma}(D)$ of all $D$-qualified atoms. At this point, and before extending the notions of $D$-entailment relation and interpretation to the SQLP($\mathcal{R}, D$) 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 $A \in A_{\Sigma}$ are built from a linearized version of $A$ which has the form $\text{lin}(A) = (A_i, S_i)$ and is constructed as follows: $A_i$ is a linear atom built from $A$ by replacing each $n$ additional occurrences of a variable $X$ by fresh variables $X_i$ ($1 \leq i \leq n$); and $S_i$ is a set of similarity conditions $X \sim X_i$ (with $1 \leq i \leq n$) asserting the similarity of all variables in $A_i$ 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: $A_i$ is just the same as $A$ and $S_i$ is empty.

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

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

**Definition 4.** (R-instance of an atom). Assume an atom $A \in A_{\Sigma}$ and its linearized version $\text{lin}(A) = (A_i, S_i)$. Then, an atom $A_i'$ is
said to be an $\mathcal{R}$-instance of $A$ with similarity degree $\delta$, noted as 
$(A', \delta) \in [A]_\mathcal{R}$, if there are some atom $A'$ and some substitution $\theta$ such that $A' = A^\theta$ and $\delta = \mathcal{R}(A, A') \cap \bigcap \{(R(X_\theta, X'_\theta) \mid (X_\theta \sim X'_\theta) \in S) \neq \perp \}$.

Next, the $(\mathcal{R}, D)$-entailment relation over $\mathcal{A}_{\mathcal{LS}}(D)$ is defined as follows: $A \not\vdash d \not\models (\mathcal{R}, D)$ $A' \not\vdash d'$ iff there is some similarity degree $\delta$ such that $(A', \delta) \in [A]_\mathcal{R}$ and $d' \subseteq d \delta$. Finally, an open Herbrand interpretation – just interpretation from now on – over $(\mathcal{R}, D)$ is defined as any subset $I \subseteq \mathcal{A}_{\mathcal{LS}}(D)$ which is closed under $(\mathcal{R}, D)$-entailment. That is, an interpretation $I$ including a given $D$-qualified atom $A \not\vdash d$ is required to include all the ‘similar instances’ $A' \not\vdash d'$ such that $A \not\vdash d \models (\mathcal{R}, D)$ $A' \not\vdash d'$, because we intend to formalize a semantics in which all such similar instances are valid whenever $A \not\vdash d$ is valid. This complements the intuition given for the $D$-entailment relation in $\mathcal{QLP}(D)$ 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 $D$) qualification. Note that $(\mathcal{R}, D)$-entailment is a refinement of $D$-entailment, since: $A \not\vdash d \models (\mathcal{R}, D)$ $A' \not\vdash d'$ implies there is some substitution $\theta$ such that $A' = A^\theta$ and $d' \subseteq d \theta \Rightarrow (A', \mathcal{T}) \in [A]_\mathcal{R}$ and $d' \subseteq d \theta \Rightarrow A \not\vdash d \models (\mathcal{R}, D)$ $A' \not\vdash d'$.

Given any domain $\mathcal{D}$, we will use the notation $\mathcal{U} = \mathcal{I} \subseteq \mathcal{A}_{\mathcal{LS}}(\mathcal{D})$ for the family of all interpretations over $(\mathcal{D}, \mathcal{R})$, 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 $\mathcal{I}_{\mathcal{LS}}(\mathcal{R}, \mathcal{D})$ of all interpretations over $(\mathcal{D}, \mathcal{R})$ is a complete lattice under the inclusion ordering $\subseteq$, whose extreme points are $\mathcal{I}_{\mathcal{LS}}(\mathcal{R})$ as maximum and $\emptyset$ as minimum. Moreover, given any family of interpretations $\mathcal{I} \subseteq \mathcal{I}_{\mathcal{LS}}(\mathcal{R}, \mathcal{D})$, its lub and glib are $\bigvee \mathcal{I} = \bigcup \{I \in \mathcal{I} \subseteq \mathcal{I}_{\mathcal{LS}}(\mathcal{R}, \mathcal{D}) \mid I \subseteq I \}$ and $\bigwedge \mathcal{I} = \bigcap \{I \in \mathcal{I} \subseteq \mathcal{I}_{\mathcal{LS}}(\mathcal{R}, \mathcal{D}) \mid I \subseteq I \}$, 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 lin$(A) = (A_r, S_r)$. Then, a clause $C'$ is said to be an $\mathcal{R}$-instance of $C$ with similarity degree $\delta$, noted as $(C', \delta) \in [C]_\mathcal{R}$, if there are some atom $A'$ and some substitution $\theta$ such that $\delta = \mathcal{R}(A, A') \cap \bigcap \{(R(X_\theta, X'_\theta) \mid (X_\theta \sim X'_\theta) \in S) \neq \perp \}$ and $C' \equiv A'^\theta \leftarrow d = B_1, \theta, \ldots, B_k, \theta$.

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 $C \leftarrow d = B_1, \ldots, B_k$ in the program $\mathcal{P}$, and $\mathcal{I} \subseteq \mathcal{I}_{\mathcal{LS}}(\mathcal{R}, \mathcal{D})$ any interpretation over $(\mathcal{D}, \mathcal{R})$. 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 $d_1, \ldots, d_k \in D \setminus \{\perp\}$ such that $B'_i, d_i \in I$ for all $1 \leq i \leq k$, one has $H' \not\vdash d' \in I$ where $d' = d \circ \prod(e, d_1, \ldots, d_k)$. And we say that $\mathcal{I}$ is a model of the $\mathcal{QLP}(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$ (also written $\mathcal{I} \models \mathcal{P}$) iff $\mathcal{I}$ is a model of each clause in $\mathcal{P}$.
4. Reducing Similarities to Qualifications

4.1 A Program Transformation

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

**Definition 6.** Let \(P\) be a SSQLP(\(R, D\)) program. We define the transformed program SR(\(P\)) as:

\[ SR(P) = PS \cup P_\sim \cup P_{pay} \]

where the auxiliary sets of clauses \(P_S, P_\sim, P_{pay}\) are defined as:

- For each clause \((H' \leftarrow d \rightarrow \overline{B}) \in P\) and for each \(H'\) such that \(R(H', H') \neq \perp\)
  \[ (H' \leftarrow d - payR(H', H'), B, \overline{B}) \in P_S \]
- \(P_\sim = \{X \sim X \leftarrow \perp\} \cup \{(c(\overline{X}_n) \sim c'(\overline{Y}_n) \leftarrow \perp \sim payR(c', c'), X_1 \sim Y_1, \ldots, X_n \sim Y_n) | c, c' \in CS\text{ of arity } n, R(c, c') \neq \perp\} \]
- \(P_{pay} = \{\text{for each atom pay}_w \text{ occurring in } \overline{P}_\sim \cup P_S\} \)

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

\[
\begin{align*}
p(X, X_1) & \leftarrow 1.0 - psy_{1,0}, X \sim X_1, \\
X & \sim X \leftarrow 1.0, \\
c & \sim d \leftarrow 1.0 - psy_{0,0}, \\
psy_{1,0} & \leftarrow 1.0, \\
psy_{0,0} & \leftarrow 0.8
\end{align*}
\]

thus enabling the corresponding derivation \(SR(P) \vdash_{\text{QLI}(U)} p(1.0) \equiv 0.8\) in QLI(\(U\)).

In general, \(P\) and \(SR(P)\) are semantically equivalent in the sense that \(P \vdash_{\text{SQLI}(R,D)} A \vdash d \iff SR(P) \vdash_{\text{QLI}(D)} A \vdash d\) holds for any \(D\)-qualified atom \(A \vdash d\), as stated in Theorem 1 below. The next technical lemma will be useful for the proof of this theorem.

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

1. \(SR(P) \vdash_{\text{QLI}(D)} (t \sim s) \vdash d \implies d \subseteq R(t, s)\)
2. \(R(t, s) = d \iff SR(P) \vdash_{\text{QLI}(D)} (t \sim s) \vdash d\)

**Proof.** We prove the two items separately.

1. Let \(T\) be a QLI(\(D\)) proof tree witnessing
   \(SR(P) \vdash_{\text{QLI}(D)} (t \sim s) \vdash d\)
   We prove by induction on number of nodes of \(T\) that \(d \subseteq R(t, s)\). 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 \(X \sim X \leftarrow \perp\) is the only possibility. In this case \(t, s\) must be the same term and by the reflexivity of \(R\) (Def. 2), \(R(t, s) = \perp\), which means \(d \subseteq R(t, s)\) 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 \((c(\overline{X}_n) \sim c'(\overline{Y}_n) \leftarrow \sim payR(c', c'), X_1 \sim Y_1, \ldots, X_n \sim Y_n) \in P_\sim\), and must be of the form:
   \[
pay_w \vdash_{\text{QLP}(D)} (t_1 \sim s_1) \vdash e_1 \ldots (t_n \sim s_n) \vdash e_n\]
   where \(w = R(c, c'), v \in D, v \subseteq w, t = (\overline{t}_n), s = c'(\overline{s}_n)\), and \(e_i, \ldots, e_n \text{ s.t. } d \subseteq T \circ \text{proj} \{v, e_1, \ldots, e_k\}, i.e., d \subseteq \text{proj}\{v, e_1, \ldots, e_k\}\). By induction hypothesis \(e_i \subseteq R(t_i, s_i)\) for \(i = 1 \ldots n\). Then \(d \subseteq \text{proj}\{v, e_1, \ldots, e_n\}\) implies \(d \subseteq \text{proj}\{w, R(t_1, s_1), \ldots, R(t_n, s_n)\}\) and hence \(d \subseteq R(t, s)\) (Def. 3).

2. If \(R(t, s) = d, d \neq \perp\), we prove that \(SR(P) \vdash_{\text{QLI}(D)} (t \sim s) \vdash d\) 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 \(P_\sim\) of the form \((c \sim c' \leftarrow \perp - payR(c'))\). Using this clause and the identity substitution we can write the root inference step of a proof for \(SR(P) \vdash_{\text{QLI}(D)} (c \sim c') \vdash d\) as follows:
   \[
payR(c', c') \vdash_{\text{QLP}(D)} c \sim c' \vdash d\]

The condition required by the inference rule QMP(\(D\)) is in this particular case \(d \subseteq \text{proj}\{d\}\), and \(\perp \subseteq \text{proj}\{d\}\). Proving the only premise \(payR(c') \vdash_{\text{QLP}(D)} d\) is direct from its definition. If \(t = Y\), with \(Y\) a variable, then \(s = Y\) and \(d = \perp\) (otherwise \(R(t, s) = \perp\)). Then \(SR(P) \vdash_{\text{QLI}(D)} (Y \sim Y) \vdash \perp\) can be proved by using the clause \((X \sim X \leftarrow \perp) \in P_\sim\) with substitution \(\theta = \{X \mapsto Y\}\).

In the inductive step, \(t, s\) must be of the form \((\overline{t}_n)\), with \(n \geq 1\), and then \(s\) must be of the form \(c'(\overline{s}_n)\) (otherwise \(R(t, s) = \perp\)). From \(d = R(t, s) \neq \perp\) (hypotheses of the lemma) and Definition 3 we have that \(R(t, s) \neq \perp\). Then, by Definition 6, there is a clause in \(P_\sim\) of the form:

\[
\begin{align*}
&\begin{align*}
&c(\overline{X}_n) \sim c'(\overline{Y}_n) \leftarrow \sim payR(c', c'), X_1 \sim Y_1, \ldots, X_n \sim Y_n, \\
&\begin{array}{l}
&\text{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\} \text{ we can write the root inference step in QLI(\(D\)) as:} \\
&\text{payR(c', c') \vdash_{\text{QLP}(D)} R(t, s) \vdash d}\n&\end{array}
\end{align*}
\end{align*}
\]

The inference can be applied because the condition

\[
d \subseteq \text{proj}\{R(c, c'), R(t_1, s_1), \ldots, R(t_n, s_n)\}
\]

reduces to

\[
d \subseteq \text{proj}\{R(c, c'), R(t_1, s_1), \ldots, R(t_n, s_n)\}
\]

which holds by Definition 3, item 3. Moreover, the premises \(t_i \sim s_i \subseteq R(t_i, s_i), i = 1 \ldots n\) hold in QLI(\(D\)) due to the inductive hypotheses, and proving

\[
payR(c', c') \vdash_{\text{QLP}(D)} R(t, s)
\]

is straightforward from its definition. □

Now we can prove the equivalence between semantic inferences in QLI(\(D\)) w.r.t. \(P\) and semantic inferences in SSQLI(\(R, D\)) w.r.t. \(SR(P)\).

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

\[
P \vdash_{\text{SQLI}(R,D)} A \vdash d \iff SR(P) \vdash_{\text{QLI}(D)} A \vdash d
\]
Proof. Let $T$ be a $SQH L(D)$ proof tree for some annotated atom $A \sharp d$ in $P$’s signature witnessing $P \vdash_{SQH L(D)} A \sharp d$. We prove that $S_R(P) \vdash_{QHL(D)} A \sharp d$ by induction on the number of nodes of $T$.

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

$$B_1^* \sharp d_1 \cdots B_k^* \sharp d_k$$

with $(A \equiv e - B_1^*, \ldots, B_k^*, e, \delta) \in [C]_R$ for some clause $C \equiv (H \equiv e - B_1, \ldots, B_k, e) \in P$ (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 $\theta$ and atom $H'$ such that $\delta = R(H_1, H') \cap \{\{X, \theta : X \equiv \theta \mid (X, \sim X) \in S_1\} \neq \bot$, with $\text{lin}(H) = (H_1, S_1)$. This means in particular that $w \equiv R(H_1, H') \neq \bot$, which by Definition 6 implies that there is a clause $C'$ in $S_R(P)$ of the form $C' \equiv (H' \equiv e - p_{ay}, S_1, B_1, \ldots, B_k)$. Then the root inference step of the deduction proving $P \vdash_{QHL(D)} A \sharp d$ will use the inference rule QMP(D) with $C'$ and substitution $\theta$ (such that $H' \equiv A$) as follows:

$$\text{pay}_{\theta} \equiv (w, (u_i, \sim v_i, \theta) e_i)_{1 \leq i \leq m} B_1^* \sharp d_1 \cdots B_k^* \sharp d_k$$

where $S_1 = \{u_1 \sim v_1, \ldots, u_m \sim v_m\}$, and $e_i = R(u_1, v_i, \theta)$ for $i = 1 \ldots m$. Next we check that the premises can be proved from $S_R(P)$ in $QHL(D)$:

- $\text{pay}_{\theta} \equiv \text{pay}_{\theta}$, since $\text{pay}_{\theta}$ is a nullary predicate for every $w$. Therefore $S_R(P) \vdash_{QHL(D)} \text{pay}_{\theta} \equiv w$ is immediate from the definition of $\text{pay}_{\theta}$ in Definition 6.

- For each $1 \leq i \leq m$, we observe that $R(u_i, v_i, \theta) \neq \bot$ because $\delta$ has not been computed above as the infimum of a set including $R(u_i, v_i, \theta)$ among its members. Then $S_R(P) \vdash_{QHL(D)} (u_i \sim v_i, \theta) e_i$ holds by Lemma 1, item 2.

- For each $1 \leq i \leq k$, (1) shows that $P \vdash_{SQH L(D)} B_i^* \sharp d_i$ with a proof tree having less nodes. Therefore, $S_R(P) \vdash_{QHL(D)} B_i^* \sharp d_i$, by induction hypothesis.

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

- As defined above, $\delta = R(H, H') \cap \{\{R(X, \theta) : X \equiv \theta \mid (X, \sim X) \in S_1\} \neq \bot$, with $w \equiv \bigcap \{e_1, \ldots, e_m\}$.

- By the SQMP(D, R) inference (1) we know that $d \subseteq e \equiv \bigcap \{d_1, d_2, \ldots, d_k\}$.

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

Since $A$ is in $P$’s signature, the clause employed at the inference step at the root of $T$ must be in the set $P_3$ 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 $C \equiv (H \equiv e - B_1, \ldots, B_k, e) \in P$ and some atom $H'$ such that $A = H' \equiv$ and $R(H, H') \neq \bot$, where $\text{lin}(H) = (H_1, S_1)$.

Then we can use $C$ and $\theta$ to prove $P \vdash_{QHL(D)} A \sharp d$ by a SQMP(D, R) inference like (1) using the $R$-instance $C' \equiv A \equiv e - B_1, \ldots, B_k$ of $C$. The premises can be proved in $SQH L(D)$ by induction hypotheses, since all of them are also premises in (2). Finally, we must check that the conditions required by (1) hold:

$$C' \equiv (\delta', \delta) \in [C]_R \text{ for some } \delta \in D, \delta \neq \bot \land (e \equiv \bigcap \{d_1, d_2, \ldots, d_k\}) \text{, and this is true for } e = \{w, e_1, \ldots, e_m\} \text{, with } e_i = R(u_i, v_i, \theta) \text{ for } i = 1 \ldots m.$$
2. According to [16], \( P_0.9 \) is computed from \( P \) 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 \( \{r\}, \{p\}, \{q\}, \{s\}, \{c,d\} \). This can be represented by the symbols \( r, p, q, s \) an \( e \), respectively. Then, \( P_0.9 \) replaces the two clauses \( C_p \) and \( C_q \) by \( p(e(U)) \leftrightarrow q(e(V)) \), respectively, leaving the other two clauses unchanged. Solving \( G \) w.r.t. \( P_0.9 \) by means of classical \( SLD \) resolution produces the answer substitution \( \sigma' = \{ X \mapsto e(U), Y \mapsto e(U) \} \), which corresponds to \( \sigma \) 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 \( P \) can be trivially converted into a semantically equivalent a \( SQLP(\mathcal{R}, \mathcal{U}) \) program, just by replacing each occurrence of the implication sign \( \rightarrow \) in \( P \)'s clauses by \( \rightarrow 1.0 \)-. Then \( S_{\mathcal{R}}(P) \) can be built as a \( QLP(\mathcal{U}) \) program by the method explained in Subsection 4.1. It includes three clauses corresponding to \( C_p, C_q \) and \( C_r \) of \( P \) plus the following three new clauses:

\[
\begin{align*}
C_p' & : \quad p(d(U)) \rightarrow 1.0 - pay_{0.9} \\
C_q' & : \quad q(c(V)) \rightarrow 1.0 - pay_{0.9} \\
C_r' & : \quad s(Z_1, Z_2) \rightarrow 1.0 - Z_1 \sim Z_2
\end{align*}
\]

where \( C_p \) resp. \( C_q \) come from replacing the linear heads of \( C_p \) resp. \( C_q \) by similar heads, and \( C_r \) comes from linearizing the head of \( C_r \), which allows no replacements by similarity. \( S_{\mathcal{R}}(P) \) includes also the proper clauses for \( P \) and \( P_{pay} \), in particular the following three ones:

\[
\begin{align*}
I & : \quad X \sim X \sim 1.0 - \\
S & : \quad c(X_1) \sim d(Y_1) \rightarrow 1.0 - pay_{0.9}, X_1 \sim Y_1 \\
P & : \quad pay_{0.9} \rightarrow 0.9 -
\end{align*}
\]

Solving goal \( G \) w.r.t. \( S_{\mathcal{R}}(P) \) by means of the \( \mathcal{U} \)-qualified \( SLD \) resolution procedure described in [14] can compute the answer substitution \( \sigma \) 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 \( C_p ', I, S \) 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 \( QLP(D) \) scheme (see [14] and Subsection 3.1 above) shows that \( MALP \) programs over the multi-adjoint lattice \( G \) 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 \( P_{E, R} \) as presented with he syntax of a \( QLP(\mathcal{U}) \) program. The original program \( P \) can also be written as a \( QLP(\mathcal{U}) \) program just by replacing each the implication sign \( \rightarrow \) occurring in \( P \) by \( \rightarrow 1.0 \). As explained in [11], \( P_{E, R} \) is built by extending \( P \) with clauses for a new binary predicate \( \sim \) intended to emulate the behaviour of \( \mathcal{R} \)-based unification between terms. In our example, \( P_{E, R} \) will include (among others) the following clause for \( \sim \):

\[
\begin{align*}
S' : \quad c(X_1) \sim d(Y_1) \rightarrow 0.9 - X_1 \sim Y_1
\end{align*}
\]

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

Moreover, \( P_{E, R} \) is defined to include a clause of the following form for each pair of \( n \)-ary predicate symbols \( pd \) and \( pd' \) such that \( R(pd, pd') \neq 0 \):

\[
\begin{align*}
C_{pd, pd'} : \quad pd(Y_1, \ldots , Y_n) \leftarrow R(pd, pd') - pd'(X_1, \ldots , X_n), X_1 \sim Y_1, \ldots , X_n \sim Y_n
\end{align*}
\]

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 \( R(pd, pd') = 1.0 \). Solving goal \( G \) w.r.t. \( S_{\mathcal{R}}(P) \) by means of the procedural semantics described in Section 4 of [11] can compute the answer substitution \( \sigma \) to the similarity degree 0.9. More generally, Theorem 24 in [11] claims that for any choice of \( \mathcal{P} \), \( P_{E, R} \) can emulate any successful computation performed by \( P \) using \( \mathcal{R} \)-based \( SLD \) resolution.

In conclusion, the main difference between \( S_{\mathcal{R}}(P) \) and \( P_{E, R} \) 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. \( P_{E, R} \) always relies on the clauses of the form \( C_{pd, pd'} \) and the clauses for \( \sim \), while \( S_{\mathcal{R}}(P) \) 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 \( H_\lambda(P) \) and \( P_\lambda \), our transformation \( S_{\mathcal{R}}(P) \) does not depend on a fixed similarity degree \( \lambda \) 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 \( S_{\mathcal{R}}(P) \) for solving goals w.r.t. the original program \( P \), we consider the case where \( P \) is the \( SQLP(\mathcal{R}, \mathcal{U}) \) program displayed in Figure 1. The transformed program \( S_{\mathcal{R}}(P) \) 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 \( R(A, A) = 1 \) 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 \( pay_{0.9} \). 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 \( R(\text{lynx}, \text{cat}) = 0.8 \), \( R(\text{boar}, \text{pig}) = 0.7 \), respectively.

- Analogously, for instance the clause at line 10 (with head \( \text{farm}(\text{lynx}) \)) is obtained by head-similarity with the clause of line 6 in the \( SQLP(\mathcal{R}, \mathcal{U}) \) program (head \( \text{domestic}(\text{cat}) \),...
and the subindex at pay is obtained from

\[ \mathcal{R}(\text{domestic(cat)}, \text{farm(lynx)}) = \mathcal{R}(\text{domestic.farm}) \cap \mathcal{R}(\text{cat.lynx}) = 0.3 \geq 0.8 = 0.3 \]

- 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 \( \text{pay} \) correspond to the fragment \( \mathcal{P}_{\text{pay}} \) in Definition 6.

In the rest of this subsection, we will show an execution for the goal \( \text{pet(A)#W} \mid W \geq 0.50 \) over the program \( \mathcal{S}_\mathcal{R}(\mathcal{P}) \) (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 \( QLP(U) \) and \( QLP(W) \). Although this prototype hasn’t been released as an integrated part of \( TOY \), you can download it 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 \( \mathcal{S}_\mathcal{R}(\mathcal{P}) \), because it was developed mainly for [14]. Therefore, the transformed program shown in Figure 2 has been computed manually.

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

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

this will have the effect of loading the Real Domain Constraints library and the \( QLP(U) \) 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 \( TOY \) 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 -> 0.5699999999999999 }

sol.1, more solutions (y/n/d/a) [y]?
{ A -> lynx,
  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 \( QLP(D) \) scheme introduced in [14], which supports logic programming with attenuated program clauses over a parametrically given domain \( D \) 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.

---

Figure 2. Example of transformed program. (Note: no clauses for \( \sim \) are needed because the original program was left-linear).
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 \( D \), we have extended \( QLP(D) \) to a more expressive scheme \( SQLP(R, D) \) with two parameters, for programming modulo \( R \)-similarity with \( D \)-attenuated Horn clauses. We have presented a declarative semantics for \( SQLP(R, D) \) programs and a semantics-preserving program transformation which embeds \( SQLP(R, D) \) into \( QLP(D) \). As a consequence, the sound and complete procedure for solving goals in \( QLP(D) \) by \( D \)-qualified \( SLD \) resolution and its implementation in the \( TOY \) system [14] can be used to implement \( SQLP(R, D) \) 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 \( QLP(D) \) and \( SQLP(R, D) \) 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, \( D \)-qualified \( SLD \) resolution does not rely on costly computations of redundant clauses and has been efficiently implemented.

As future work, we plan to investigate an extension of the \( R \)-based \( SLD \) resolution procedure proposed in [16] to be used within the \( SQLP(R, D) \) 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


