

Electronic Notes in Theoretical Computer Science 248 (2009) 67–82

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Qualified Logic Programming with Bivalued Predicates

Mario Rodr´ıguez-Artalejo[1](#_bookmark0) Carlos A. Romero-D´ıaz[2](#_bookmark0)

*Departamento de Sistemas Informa´ticos y Computacio´n Universidad Complutense de Madrid*

*Madrid, Spain*

**Abstract**

Research on the field of uncertainty in logic programming has evolved during the last 25 years. In a recent paper [[13](#_bookmark24)] we have revised a classical approach by van Emden’s to *Quantitative Logic Programming* [[19](#_bookmark33)], generalizing it to a generic scheme QLP(*D*) for so-called *Qualified Logic Programming* over a parametrically given domain *D*, whose elements play the role of generalized truth values and can be used to qualify logical assertions. In this paper we present an extension of QLP(*D*) yielding a more expressive scheme BQLP(*D*),

which supports a simple kind of negation based on bivalued predicates and allows *threshold constraints* in clause bodies in order to impose lower bounds to the qualifications computed by program clauses. The new scheme has a rigorous declarative semantics and a sound and strongly complete goal resolution procedure which can be efficiently implemented using constraint logic programming technology.

*Keywords:* Bivalued Predicates, Qualification Domains, Qualified Logic Programming, Threshold Constraints.

# Introduction

The historical evolution of research on uncertainty in logic programming has been described in a recent recollection by V. S. Subrahmanian [[18](#_bookmark32)] and briefly summa- rized in the introductory section of [[13](#_bookmark24)]. Early approaches include the quantitative treatment of uncertainty in the spirit of fuzzy logic, as in van Emden’s classical paper [[19](#_bookmark33)], and two subsequent papers by Subrahmanian [[16](#_bookmark30),[17](#_bookmark31)]. The main contri- bution of [[19](#_bookmark33)] was a rigorous declarative semantics for a LP language with program clauses of the form *A ← f − B*, where the head *A* is an atom, the body *B* is a conjunction of atoms, and the so-called *attenuation* factor *f ∈* (0*,* 1] attached to the clause’s implication is used to propagate to the head the certainty factor *f × b*,

٨ Research partially supported by projects MERIT-FORMS (TIN2005-09027-C03-03) and PROMESAS- CAM(S-0505/TIC/0407)

1 Email: [mario@sip.ucm.es](mailto:mario@sip.ucm.es)

2 Email: [cromdia@fdi.ucm.es](mailto:cromdia@fdi.ucm.es)

1571-0661 © 2009 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2009.07.060

where *b* is the minimum of the certainty factors previously computed for the various atoms occurring in the body. The papers [[16](#_bookmark30),[17](#_bookmark31)] proposed to use a special lattice *S* in place of the lattice of the real numbers in the interval [0*,* 1] under their natural ordering. *S* includes two isomorphic copies of [0*,* 1] whose elements are incompara- ble under *S*’s ordering and can be separately used to represent degrees of *truth* and *falsity*, respectively, thus enabling a simple treatment of negation. Other main con- tributions of [[16](#_bookmark30),[17](#_bookmark31)] were the introduction of annotated program clauses and goals (later generalized to a much more expressive framework in [[6](#_bookmark20)]) and the introduction of goal solving procedures more convenient and powerful than those given in [[19](#_bookmark33)].

In a recent paper [[13](#_bookmark24)] we have revised the approach to *Quantitative Logic Pro- gramming* (QLP for short) in [[19](#_bookmark33)], generalizing it to a generic scheme QLP(*D*) for so-called *Qualiﬁed Logic Programming* over a parametrically given domain *D*, which must be a lattice satisfying certain natural axioms. The class of qualification do- mains includes the lattice [0*,* 1] used in [[19](#_bookmark33)] as well as other lattices whose elements can be used to qualify logical assertions in other ways. In this paper we present an extension of QLP(*D*) to a more expressive scheme BQLP(*D*) which provides two main novelties w.r.t. [[13](#_bookmark24)]. Firstly, so-called *threshold constraints* are used to im- pose lower bounds to the qualifications computed for the individual atoms in clause bodies, with the aim of preventing inferences from insufficiently qualified premises. Secondly, a simple kind of negation is supported by the use of *marked atoms A* tt, and *A* ff, where tt and ff stand for the two classical truth values. Marked atoms can be viewed as logical assertions associated to bivalued predicates, and their de- gree of validity can be qualified by elements of a parametrically given qualification domain *D*. In particular, if the given *D* is the lattice [0*,* 1], qualifying marked atoms with numeric degrees *d ∈* (0*,* 1] is similar to annotating atoms with values of the lattice *S*, as proposed in [[16](#_bookmark30),[17](#_bookmark31)]. Nevertheless, the present approach differs from [[16](#_bookmark30),[17](#_bookmark31)] in two major aspects: a) our framework can operate with any parametrically

given qualification domain *D* instead of the fixed lattice *S*; and b) we use attenuated clauses with threshold constraints, whose expressivity is quite different from that of the annotated clauses used in [[16](#_bookmark30),[17](#_bookmark31)]. More comparisons to related work can be found in Section [6](#_bookmark14) and in the concluding section of [[13](#_bookmark24)].

As it was the case for QLP(*D*), the new scheme BQLP(*D*) has a rigorous declar- ative semantics and a sound and strongly complete goal resolution procedure which can be efficiently implemented using constraint logic programming technology. The rest of this paper is structured as follows: Section [2](#_bookmark1) presents the axioms for qualifica- tion domains *D* and its basic properties, including closure under cartesian product. The axioms have been revised w.r.t. [[13](#_bookmark24)] with the aim of enabling the technical treatment of threshold constraints in subsequent sections. Section [3](#_bookmark3) presents the syntax and declarative semantics of the BQLP(*D*) scheme. Section [4](#_bookmark8) presents a goal solving procedure for BQLP(*D*) along with its soundness and strong completeness properties. Section [5](#_bookmark13) sketches a general implementation technique for BQLP(*D*) which can be used to implement useful instances of the scheme on top of any system that supports sufficient CLP technology. Finally, Section [6](#_bookmark14) summarizes conclusions and plans for future work.

# Qualification Domains and their Properties

By definition, a *Qualiﬁcation Domain* is any structure *Ð* = *⟨D, ±, ⊥, T, ⊗, ⟩* such that:

1. *⟨D, ±, ⊥, T⟩* is a lattice with extreme points *⊥* and *T* w.r.t. the partial ordering

*±*. For given elements *d, e ∈ D*, we write *d H e* for the *greatest lower bound* (*glb*) of *d* and *e* and *d H e* for the *least upper bound* (*lub*) of *d* and *e*. We also write *d* и *e* as abbreviation for *d ± e Λ d /*= *e*.

1. *⊗* : *D × D —→ D*, called *attenuation operation*, verifies the following axioms:
   1. *⊗* is associative, commutative and monotonic w.r.t. *±* .
   2. *6d ∈ D* : *d ⊗T* = *d* .
   3. *6d ∈ D* : *d ⊗⊥* = *⊥* .
   4. *6d, e ∈ D \ {⊥, T}* : *d ⊗ e* и *e* .
   5. *6d, e*1*, e*2 *∈ D* : *d ⊗* (*e*1 *H e*2) = *d ⊗ e*1 *H d ⊗ e*2 .
2. : *D×D* −−· *D* is a partial operation defined as the inverse of *⊗*, that satisfies the following axioms: Given *d ∈ D \ {⊥}* and *e, e' ∈ D*,
   1. (Inverse) *e d* is defined and *e d* = *e' ⇐⇒ e' ⊗ d* = *e*.
   2. (Polarity) *e ± e'*, *d ± d'*, *e' d'* defined =*⇒ e d ± e' d'* also defined.

In the rest of the paper, *Ð* will always denote an arbitrary qualification domain. The axioms stated above are like those in [[13](#_bookmark24)] except that nooperation was required there. For any finite *S* = *{e*1*, e*2*,..., e*n*} ⊆ D*, the *glb* of *S* (noted as

. *S*) exists and can be computed as *e*1 *H e*2 *H · · · H e*n (which reduces to *T* in

the case *n* = 0). As an easy consequence of the axioms, one gets the identity

*d ⊗* . *S* = .*{d ⊗ e | e ∈ S}*. Three interesting instances of qualification domain are

shown below.

**The Domain of Classical Boolean Values.** *У* = (*{*0*,* 1*}, ≤,* 0*,* 1*, Λ,* ), where 0 and 1 stand for the two classical truth values *false* and *true*, *≤* is the usual numerical ordering over *{*0*,* 1*}*, *Λ* stands for the classical conjunction operation over *{*0*,* 1*}*, and

is defined by the two equations 01 = 0 and 11 = 1. The instance BQLP(*У*) of our BQLP(*Ð*) scheme will behave as classical logic programming extended with bivalued predicates.

**The Domain of Uncertainty Values.** *C* = (U*, ≤,* 0*,* 1*, ×, /*), where U = [0*,* 1] =

*{d ∈* R *|* 0 *≤ d ≤* 1*}*, *≤* is the usual numerical ordering, *×* is the multiplication operation, and */* is the division operation (with *e/*0 undefined). In this domain,

the top element *T* is 1 and the greatest lower bound . *S* of a finite *S ⊆* U is the

minimum value min(S), which is 1 if *S* = *∅*. Therefore, the instance BQLP(*C* ) of our BQLP(*Ð*) scheme will behave as *van Emden*’s QLP extended with bivalued predicates.

**The Domain of Weight Values.** *У* = (P*, ≥, ∞,* 0*,* +*, —*), where P = [0*, ∞*] =

*{d ∈* R *∪ {∞} | d ≥* 0*}*, *≥* is the reverse of the usual numerical ordering (with

*∞ ≥ d* for any *d ∈* P), + is the addition operation (with *∞* + *d* = *d* + *∞* = *∞*

for any *d ∈* P), and *—* is the substraction operation (with *e —∞* undefined and

*∞— d* = *∞* for *d /*= *∞*). In this domain, the top element *T* is 0 and the greatest

lower bound . *S* of a finite *S ⊆* P is the maximum value max(S), which is 0 if *S* = *∅*. When working in the instance BQLP(*У*) of our BQLP(*Ð*) scheme, one propagates to a clause head the qualification value *α* + *b*, where *α* is the clause’s *attenuation factor* and *b* is the maximum of the qualification values known for the body atoms. Therefore, qualification values in the instance BQLP(*У*) of our BQLP(*Ð*) scheme behave as a weighted measure of depths of proof trees.

It is easily checked that the axioms of qualification domains are satisfied by *У*, *C* and *У*. In fact, the axioms have been chosen as a natural generalization of some basic properties satisfied by the ordering *≤* and the operations *×* and */* in *C* . The following result states some intuitive properties ofas the inverse of *⊗*. The proof is an easy exercise.

**Proposition 2.1** *The following properties hold in any qualiﬁcation domain Ð, for any d, d*1*, d*2 *∈ D \ {⊥} and any e ∈ D:*

1. (*d ⊗ e*) *d* = (*e ⊗ d*) *d* = *e .*
2. *d ⊗* (*e d*) = (*e d*) *⊗ d* = *e .*
3. (*e d*1) *d*2 = *e* (*d*1 *⊗ d*2) = *e* (*d*2 *⊗ d*1) *.*

Given two qualification domains *Ð*i = *⟨D*i*, ±*i*, ⊥*i*, T*i*, ⊗*i*,* i*⟩* (*i ∈ {*1*,* 2*}*), their *cartesian product Ð*1*×Ð*2 is defined as *Ð* =def *⟨D, ±, ⊥, T, ⊗, ⟩*, where *D* =def *D*1*× D*2, the partial ordering *±* is defined as (*d*1*, d*2) *±* (*e*1*, e*2) *⇐⇒*def *d*1 *±*1 *e*1 and *d*2 *±*2 *e*2, *⊥* =def (*⊥*1*, ⊥*2), *T* =def (*T*1*, T*2), the attenuation operator *⊗* is defined as (*d*1*, d*2) *⊗* (*e*1*, e*2) =def (*d*1 *⊗*1 *e*1*, d*2 *⊗*2 *e*2) and its inverseis defined as (*d*1*, d*2)(*e*1*, e*2) =def (*d*11 *e*1*, d*22 *e*2). Intuitively, each value (*d*1*, d*2) belonging to a product domain *Ð*1 *× Ð*2 imposes the qualification *d*1 *and also* the qualification *d*2. The class of the qualification domains is closed under cartesian products, as stated in the following result. The proof is a simple extension of that found in [[12](#_bookmark25)], adding the arguments needed for the axioms of the operation.

**Proposition 2.2** *The cartesian product Ð* = *Ð*1 *× Ð*2 *of two given qualiﬁcation domains is always another qualiﬁcation domain.*

# Syntax and Semantics of the BQLP(*Ð*) Scheme

* 1. *Signature and Programs*

We assume a signature Σ providing free function symbols (a.k.a. constructors) and predicate symbols. *Terms* are built from constructors and variables from a countably infinite set *Var*, disjoint from Σ. The set SubstΣ of all substitutions of terms for variables in *Var* is defined in the usual way. *Atoms* are of the form *p*(*t*1*,..., t*n), shortened as *p*(*t*n) or simply *p*(*t*), where *p* is a *n*-ary predicate symbol and *t*i are terms. We write AtΣ, called the *open Herbrand base*, for the set of all atoms. We call *marked atom* to *A v* where *A ∈* AtΣ and *v ∈ {*tt*,* ff*}*, and *Ð- annotated atom* to *A* (*v, w*) where *A ∈* AtΣ, *v ∈ {*tt*,* ff*}* and *w ∈* (*D \ {⊥}*) *{*?*}* (note that tt*,* ff *∈/* Σ and ? *∈/ D*). A marked atom *A v* where *A* is *p*(*t*) is intended as a logical assertion related to a *bivalued predicate p*, and a *Ð*-annotated atom

*A* (*v, w*) is intended as the requirement that *A v* holds with at least a qualification *e ∈ D \ {⊥}* such that *e ±*? *w*, where *e ±*? *w ⇐⇒*def *w* = ? or else *w ∈ D \ {⊥}* and *e ± w*. Note that when *w* = ? no effective constraint is imposed over *e*. Requirements of the form *e ±*? *w* will be called *threshold constraints* in the rest of the paper.

A BQLP(*Ð*)-program *P* is a finite set of *program rules* of the form *A v ← α— B*1(*v*1*, w*1)*,..., B*k(*v*k*, w*k) where *A v* is a marked atom, *B*i(*v*i*, w*i) with 1 *≤ i ≤ k* are *Ð*-annotated atoms and *α ∈ D \ {⊥}*. Such a rule is also called an *attenuated deﬁnite Horn clause* with *attenuation value α* attached to its impli- cation and *threshold constraints* attached to its atoms bodies as indicated by their annotations. The behavior of a program clause as inference rule will be formalized in Subsection [3.2](#_bookmark6) below. Roughly, the idea is to propagate an annotation (*v, d*) to the head atom whenever annotations (*v*i*, d*i) for all 1 *≤ i ≤ k*, satisfying their

corresponding threshold constraints *d*i *±*? *w*i, are known to hold for the body atoms and *d ± α ⊗* .*{d*1*,..., d*k*}*.

**Example 3.1** The simple programs over the domains *C* , *C×У* and *У* shown below are not intended as realistic applications but just as illustrations. In each case, the program can be understood as a *knowledge base* given by the facts for the predicates animal, plant, human and eats, along with *knowledge inference rules* corresponding to the clauses with non-empty body. Due to the attenuation values attached to clause implications, qualification degrees can decrease when moving from a clause’s body to its head. Note the differences between them when the qualification domain varies.

* + 1. The BQLP(*C* )-program *PU* contains the clauses you can see in Figure [1](#_bookmark5).
    2. The BQLP(*C ×У*)-program *PU×W* is similar to *PU* , except that the attenuation value (c,1) *∈ C ×У* replaces the attenuation value c *∈ C* at the implication sign of every clause in *PU* and the *C ×У*-annotation (v,(c,1)) replaces the *C* -annotation (v,c) (where c *∈ C* ) at every body atom in *PU* (note that the remaining *C* -annotations (v,?) are also valid *C ×У*-annotations). Therefore, each clause is now intended to convey the additional information that the depth of a proof tree for the head is 1 plus the maximum depth of proof trees for the atoms in the body.
    3. The only possible attenuation value in the domain *У* is 1, which conveys no sig- nificant information. Therefore, the BQLP(*У*) program *PB* obtained form *PU* by placing 1 as attenuation value at all the clauses is essentially a classical logic program, where the marks tt and ff can be thought as additional predicate arguments. Due to the left recursion in the clauses for human and eats, some goals for *PB* have an infinite search space where SLD resolution with a left- most selection strategy fails to compute some expected answers. For instance, the answer *{*X *'→* mother(eve)*,* Y *'→* apple*}* would not be computed for the goal eats(X,Y)#tt. However, when solving goals for the qualified programs *PU* and *PU×W* using the resolution method presented in Section [4](#_bookmark8), threshold constraints can be used for pruning the search space, so that even the leftmost

1: cruel(X)#tt <-0.90- human(X)#(tt,?), eats(X,Y)#(tt,?), animal(Y)#(tt,?)

2: cruel(X)#tt <-0.40- human(X)#(tt,?), eats(X,Y)#(tt,?), plant(Y)#(tt,?)

3: cruel(X)#ff <-0.90- human(X)#(tt,?), eats(X,Y)#(ff,?), animal(Y)#(tt,?)

4: cruel(X)#ff <-0.40- human(X)#(tt,?), eats(X,Y)#(ff,?), plant(Y)#(tt,?)

5: animal(bird)#tt <-1.0-

6: animal(cat)#tt <-1.0-

7: plant(oak)#tt <-1.0-

8: plant(apple)#tt <-1.0-

9: human(adam)#tt <-1.0-

10: human(eve)#tt <-1.0-

11: human(father(X))#tt <-0.90- human(X)#(tt,0.50)

12: human(mother(X))#tt <-0.90- human(X)#(tt,0.50)

13: eats(adam,X)#tt <-0.80-

14: eats(eve,X)#tt <-0.30- animal(X)#(tt,?)

15: eats(eve,X)#tt <-0.60- plant(X)#(tt,?)

16: eats(father(X),Y)#tt <-0.80- eats(X,Y)#(tt,0.40)

17: eats(mother(X),Y)#tt <-0.70- eats(X,Y)#(tt,0.40)

18: eats(adam,X)#ff <-0.20-

19: eats(eve,X)#ff <-0.70- animal(X)#(tt,?)

20: eats(eve,X)#ff <-0.40- plant(X)#(tt,?)

21: eats(father(X),Y)#ff <-0.80- eats(X,Y)#(ff,0.40)

22: eats(mother(X),Y)#ff <-0.70- eats(X,Y)#(ff,0.40)

Fig. 1. BQLP(*U* ) program *PU*

selection strategy leads to successful computations.

* 1. *Declarative Semantics*

In order to formalize program semantics we define *qualiﬁcation values* as pairs (*v, d*) where *v ∈ {*tt*,* ff*}* and *d ∈ D \ {⊥}*, and *Ð-qualiﬁed atoms* as *Ð*-annotated atoms *A* (*v, d*) such that (*v, d*) is a qualification value (i.e., *d /*= ?). The *Ð-qualiﬁed Her- brand base* is defined as the set AtΣ(*Ð*) of all *Ð*-qualified atoms. The *Ð-entailment* relation over AtΣ(*Ð*) is defined as follows: *A* (*v, d*) *D A'* (*v', d'*) iff there is some substitution *θ* such that *A'* = *Aθ*, *v'* = *v* and *d' ± d*. Finally, we define an *open Herbrand interpretation* over *Ð* as any subset *I ⊆* AtΣ(*Ð*) which is closed under *Ð*-entailment. That is, a Herbrand interpretation *I* including a given *Ð*- qualified atom *A* (*v, d*) must also include all the “instances” *A'* (*v', d'*) such that *A* (*v, d*) *D A'* (*v', d'*). From now on we will write IntΣ(*Ð*) for the family of all Herbrand interpretations over *Ð*. The following proposition is easy to prove from the definition of an Herbrand interpretation and the definitions of the union and intersection of a family of sets.

**Proposition 3.2** *The family* IntΣ(*Ð*) *of all Herbrand interpretation over Ð is a complete lattice under the inclusion ordering ⊆, whose extreme points are* IntΣ(*Ð*) *as maximum and ∅ as minimum. Moreover, given any family of interpretations*

*I ⊆* IntΣ(*Ð*)*, its* lub *and* glb *are* . *I* = *{I ∈* IntΣ(*Ð*) *|I ∈ I} and* . *I* = *{I ∈*

IntΣ(*Ð*) *|I ∈ I}, respectively.*

Let *C* be any clause *A v ← α— B*1(*v*1*, w*1)*,..., B*k(*v*k*, w*k) in the program *P* and *I ∈* IntΣ(*Ð*) any interpretation over *Ð*. We say that *I* is a *model* of *C* (and write *I |*= *C*) iff for any substitution *θ*, *I |*= *Cθ*. Assuming *Cθ ≡ A' v ← α— B'* (*v*1*, w*1)*,..., B'* (*v*k*, w*k), then *I |*= *Cθ* iff for every *d*1*,..., d*k *∈ D \ {⊥}*

1 k

such that *d*i *±*? *w*i and *B'* (*v*i*, d*i) *∈I* for all 1 *≤ i ≤ k*, one has *A'* (*v, d*) *∈I* for

the value = . i . We say that is a model of (and write = )

*d α ⊗*

*{d*i*,..., d*k*}*

*I P I | P*

iff *I |*= *C* holds for every clause *C ∈ P*.

As in any logic language, we need some technique to infer formulas (in our case, *Ð*-qualified atoms) from a given BQLP(*Ð*)-program *P*. We consider two alternative ways of formalizing an inference step which goes from the body of a clause to its head: an *interpretation transformer* T*P* and a qualified variant of Horn Logic, noted as QHL(*Ð*) and called *Qualiﬁed Horn Logic over Ð*. The interpretation transformer T*P* : IntΣ(*Ð*) *→* IntΣ(*Ð*) is defined as follows:

T*P* (*I*) =def *{Aθ* (*v, d*) *|* (*A v ←α— B*1(*v*1*, w*1)*,..., B*k(*v*k*, w*k)) *∈ P, θ ∈* SubstΣ*, d*i *∈ D \ {⊥}* verifying *d*i *±*? *w*i (1 *≤ i ≤ k*)*,*

*B*i*θ* (*v*i*, d*i) *∈I* and *d ± α ⊗* .*{d*1*,..., d*k*}}*

The logic QHL(*Ð*) is defined as a deductive system consisting just of one in- ference rule QMP(*Ð*), called *Qualiﬁed Modus Ponens* over *Ð*. If there are some (*A v ← α — B*1(*v*1*, w*1)*,..., B*k(*v*k*, w*k)) *∈ P*, some *θ ∈* SubstΣ such that

*A'* = *Aθ* and *B'* = *B*i*θ* for all 1 *≤ i ≤ k*, and some *d*1*,..., d*k *∈ D \ {⊥}* such

i

that *d*i *±*? *w*i for all 1 *≤ i ≤ k*, then the following inference step is allowed for any

*d ± α ⊗* .*{d*1*,..., d*k*}*:

*B'* (*v*1*, d*1) *··· B'* (*v*k*, d*k)

1 k

*A'* (*v, d*)

QMP(*Ð*)

We will use the notations *P ▶*QHL(*D*) *A* (*v, d*) (resp. *P ▶*n

QHL(*D*)

*A* (*v, d*)) to

indicate that *A* (*v, d*) can be inferred from the clauses in program *P* in finitely many steps (resp. *n* steps). Note that QHL(*Ð*) proofs can be naturally represented as upwards growing *proof trees* with *Ð*-qualified atoms at their nodes, each node

corresponding to one inference step having the children nodes as premises.

The following proposition collects the main results concerning the declarative semantics of the BQLP(*Ð*) scheme. A full proof can be developed in analogy to the classical papers [[20](#_bookmark34),[1](#_bookmark15)], except that our Herbrand interpretations are open, as first suggested by Clark in [[4](#_bookmark18)]. Our use of the QHL(*Ð*) calculus is obviously related to the classical T*P* operator, although it has no direct counterpart in the historical papers we are aware of.

**Proposition 3.3** *The following assertions hold for any BQLP*(*Ð*) *program P:*

1. *I |*= *P ⇐⇒* T*P* (*I*) *⊆I .*
2. T*P is monotonous and continuous.*
3. *The least ﬁxpoint μ*(T*P* ) *is the least Herbrand model of P, noted as MP .*
4. *MP* = n*∈*N T*P↑*n (*∅*) = *{A* (*v, d*) *|P ▶*QHL(*D*) *A* (*v, d*)*} .*

**Proof (Sketch)** Item (1) is easy to prove from the definition of T*P* . In item (2), monotonicity (*I ⊆ J* =*⇒* T*P* (*I*) *⊆* T*P* (*J* )) follows easily from the definition of T*P* and continuity (T*P* ( n*∈*N *I*n) = n*∈*N T*P* (*I*n) for any chain *{I*n *| n ∈* N*}⊆* IntΣ(*Ð*) with *I*n *⊆ I*n+1 for all *n ∈* N) follows from monotonicity and properties of chains and sets of interpretations. Item (3) follows from (1), (2), Proposition [3.2](#_bookmark7) and some known properties about lattices. Finally, item (4) follows from proving the two

implications *P ▶*n *A* (*v, d*) =*⇒ ∃m* (*A* (*v, d*) *∈* T*P ↑*m (*∅*)) and *A* (*v, d*) *∈*

QHL(*D*)

T*P ↑*n (*∅*) =*⇒ ∃m* (*P ▶*m *A* (*v, d*)) by induction on *n*.

QHL(*D*)

The following example presents QHL(*Ð*) proofs related to the programs shown in Example [3.1](#_bookmark4) above.

## Example 3.4

1. The proof tree displayed below shows that the *C* -annotated atom at its root can be deduced from *PU* in QHL(*C* ). Therefore, the atom belongs to *MPU* .

animal(cat)#(tt,1.0)

human(eve)#(tt,1.0) human(mother(eve))#(tt,0.90)

eats(eve,cat)#(ff,0.70) eats(mother(eve),cat)#(ff,0.49) animal(cat)#(tt,1.0)

cruel(mother(eve))#(ff,0.25)

It is easy to find out which clause was used in each inference step. Note that the atom at the root could have been proved for a greater certainty value of up to 0.441. However, since 0.25 *≤* 0.441, the displayed inference is also correct (albeit less informative). Note also that inferring eats(mother(eve), cat)#(ff,0.49) by means of an instance of the last program rule for eats, eats(eve,cat)#(ff,d) must be proved with some certainty d *≥* 0.40, as required by the threshold constraint in the clause. Actually, the inference is allowed because eats(eve,cat)#(ff,0.70) can be proved.

1. A proof tree quite similar to the previous one, but with different annotations, can be easily built to show that cruel(mother(eve))#(ff,(0.25,4)) can be deduced from *PU×W* in QHL(*C ×У*). Therefore, this annotated atom belongs to *MPU×W* , and it carries information concerning both the certainty degree

0.25 *∈C* and the proof tree depth 4 *∈ У*.

# Goal Solving by Resolution in BQLP(*Ð*)

* 1. *Goals and Solutions*

In classical logic programming a goal is presented as a conjunction of atoms. In our setting, goals include *threshold constraints* intended to impose lower bounds to the qualifications of individual atoms. In the sequel we assume a countably infinite set *Уar*, disjoint from Σ and *Var*, of qualification variables *W* intended to take values over *D\{⊥}*. We consider *open Ð-annotated atoms A* (*v, W* ) with *v ∈ {*tt*,* ff*}* and *W ∈ Уar* and *threshold constraints W ±*? *β* with *W ∈ Уar* and *β ∈* (*D\{⊥}*) *{*?*}*.

Goal resolution will be formalized in the next subsection. It proceeds from an initial goal through intermediate goals until reaching a final solved goal. Initial goals look like: *A*1(*v*1*, W*1)*,..., A*n(*v*n*, W*n) *W*1 *±*? *β*1*,..., W*n *±*? *β*n, where *W*i *∈ Уar* and *β*i *∈* (*D\{⊥}*) *{*?*}*. Intermediate goals have a more general form, consisting of a composition of three items: a conjunction of open *Ð*-annotated atoms *A* waiting to be solved, a substitution *σ ∈* SubstΣ computed in previous steps, and a set of qualification constraints Δ. We consider two kinds of qualification constraints:

▢

* + 1. *W ±*? *β*, where *W ∈ Уar* is qualification variable and *β ∈* (*D \ {⊥}*) *{*?*}*. This is called a *threshold constraint* for *W* .
    2. *W* = *α ⊗* .*{W*1*,..., W*k*}*, where *W, W*1*,..., W*k *∈ Уar* are qualification vari-

ables and *α ∈ D \ {⊥}*. This is called a *deﬁning constraint* for *W* .

In order to understand why these two kinds of constraints are needed, let us anticipate the expected behavior of a resolution step. Given an initial goal including an open *Ð*-annotated atom *A* (*v, W* ) and a threshold constraint *W ±*? *β* for *W* , a resolution step with a program clause whose head unifies with *A* and whose attenuation value is *α ∈ D \ {⊥}* will be enabled only if *α ±*? *β* (thereby pruning useless parts of the computation search space) and it will lead to a new goal including

a defining constraint *W* = *α ⊗* .*{W*1*,..., W*k*}* for *W* and a threshold constraint

*W*i *±*? *β*i for each 1 *≤ i ≤ k*, where the new qualification variables *W*i correspond to the atoms in the clause’s body, and the *β*i are computed as a function of the previous lower bound *β*, the clause’s attenuation factor *α* and the lower bounds *w*i present in the threshold constraints of the clause’s body.

Let us now present some notations needed for a formal definition of goals. Given a conjunction of open annotated atoms *A* and a set of qualification constraints Δ, we define the following sets of variables:

* var(*A*) =def *{*var(*A*) *| A* (*v, W* ) *∈ A}* ,
* war(*A*) =def *{W | A* (*v, W* ) *∈ A}* ,
* war(Δ) as the set of all the qualification variables occurring in Δ, and
* dom(Δ) as the set of all *W ∈ Уar* such that *W* occurs as the left hand side of some qualification constraint in Δ.

We say that Δ is *satisﬁable* iff there is some *ω ∈* SubstΣ(*Ð*) –the set of all the substitutions of qualification values in *D \ {⊥}* for variables in *Уar*– such that *ω* is a *solution* of Δ –written *ω ∈ Sol*(Δ)–, meaning that *ω* satisfies every qualification constraint in Δ. We also say that Δ is *admissible* iff it satisfies the following three conditions:

* 1. Δ is satisfiable,
  2. for every *W ∈* war(Δ) there exists one and only one constraint for *W* in Δ (this implies dom(Δ) = war(Δ)), and
  3. the relation *>*Δ, defined by *W >*Δ *W*i iff there is some defining constraint

*W* = *α ⊗* .*{W*1*,..., W*i*,..., W*k*}* in Δ, satisfies that *>∗* is irreflexive.

Δ

Finally, we say that Δ is *solved* iff Δ is admissible and only contains defining constraints. Now we are in a position to define *goals* and their *solutions*.

**Definition 4.1 (Goals)** Given a conjunction of open *Ð*-annotated atoms *A*, a substitution *σ ∈* SubstΣ, and a set of qualification constraints Δ, we say that *G ≡ A* ▢ *σ* ▢ Δ is a *goal* iff

1. *σ ∈* SubstΣ is idempotent and such that dom(*σ*) *∩* var(*A*) = *∅*,
2. Δ is admisible, and
3. for every qualification variable in war(*A*) there is one and only one threshold constraint for *W* in Δ. And there are no more threshold constraints in Δ.

Furthermore, if *σ* = *ϵ* (the identity substitution) and Δ contains only threshold constraints, then *G* is called *initial* ; and if *A* is empty and Δ is solved, then *G* is called *solved*. For any goal *G*, we also define:

* var(*G*) =def var(*A*) *∪* dom(*σ*), and
* war(*G*) =def war(*A*) *∪* dom(Δ).

**Definition 4.2 (Goal Solutions)** A pair of substitutions (*θ, ρ*) such that *θ ∈* SubstΣ and *ρ ∈* SubstΣ(*Ð*) is called a *solution* of a goal *G ≡ A σ* Δ w.r.t. a BQLP(*Ð*)-program *P* iff:

▢ ▢

* 1. *θ* = *σθ*,
  2. *ρ ∈ Sol*(Δ), and
  3. *P ▶*QHL(*D*) *Aθ* (*v, Wρ*) for all *A* (*v, W* ) *∈ A* .

In addition, a solution (*σ, μ*) for a goal *G* is said to be more general than (or to subsume) another solution (*θ, ρ*) for the same goal *G* iff *σ* “ *θ* [var(*G*)] and *μ ± ρ* [war(*G*)], where *σ* “ *θ* [var(*G*)] means that there is some substitution *η* such that the composition *ση* behaves the same as *θ* over any variable in the set var(*G*) and *μ ± ρ* [war(*G*)] means that *μ*(*W* ) *± ρ*(*W* ) holds for any *W ∈* war(*G*).

Any solved goal *G' ≡ σ* Δ has the *associated solution* (*σ, μ*) where *μ* = *ω*Δ is the qualification substitution given by Δ such that *ω*Δ(*W* ) is the qualification value determined by the defining constraints in Δ for all *W ∈* dom(Δ) and *ω*Δ(*W* ) = *⊥*

▢

for any *W ∈ Уar \* dom(Δ). Note that for any *W ∈* dom(Δ) there exists one unique defining constraint *W* = *α ⊗* .*{W*1*,..., W*k*}* for *W* in Δ and then *ω*Δ(*W* ) can be recursively computed as *α ⊗* .*{ω*Δ(*W*1)*,..., ω*Δ(*W*k)*}*. The solutions associated to

solved goals are called *computed answers*. The next example illustrates solutions

for goals related to the programs in Example [3.1](#_bookmark4).

## Example 4.3

1. A possible goal for program *PU* in Example [3.1](#_bookmark4) is eats(father(X),Y)#(ff, W1), human(father(X))#(tt,W2) | W1>=0.4, W2>=0.6; and a valid solution for it is *{*X *'→* eve, Y *'→* bird*}* | *{*W1 *'→* 0.5, W2 *'→* 0.8*}*.
2. A goal for *PU×W* in Example [3.1](#_bookmark4) may be eats(X,Y)#(tt,W) | W *±* (0.5,4); and a valid solution is *{*X *'→* mother(adam), Y *'→* bird*}* | *{*W *'→* (0.6,3)*}*.

Note that the threshold constraint *W ±* (0*.*5*,* 4) in *С×У* imposes a qualification value *W* = (*C, S*) such that *C ≥* 0*.*5 and *S ≤* 4.

* 1. *QSLD*(*Ð*) *Resolution*

As goal solving procedure we propose *Qualiﬁed SLD Resolution*, abbreviated as QSLD(*Ð*), which extends classical *SLD* resolution with qualification constraints

over *Ð*. We write *G*0 HC1,σ1

*G*1 HC2,σ2

*···* HC*n*,σ*n*

*G*n, abbreviated as *G*0 H*∗ G*n

with *σ* = *σ*1*σ*2 *··· σ*n, to indicate a computation in *n* resolution steps starting at goal *G*0. One single resolution step is formally defined as follows:

σ

**Definition 4.4 (Resolution step)** A *resolution step* has the form *G* = *L, A* (*v,*

*W* )*, R* ▢ *σ* ▢ *W ±*? *β,* Δ HC ,σ *G*1 = (*L, B*1(*v*1*, W*1)*,..., B*k(*v*k*, W*k)*, R*)*σ*1 ▢ *σσ*1 ▢

1

1

Δ1 where *A* (*v, W* ) is called the selected atom; *C*1 *≡* (*H v ←α— B*1(*v*1*, w*1)*,..., B*k(*v*k*, w*k)) *∈*var *У* is chosen as a variant of a clause in *У* with fresh variables and

such that *α ±*? *β*; *σ*1 is a m.g.u. between A and H; *W*1*,..., W*k *∈ Уar* are fresh qual- ification variables; and Δ1 *≡ W*1 *±*? *β*1*,..., W*k *±*? *β*k*,W* = *α⊗* .*{W*1*,..., W*k*},* Δ,

where *β*i = *newThreshold*(*β, α, w*i) is defined as follows for all 1 *≤ i ≤ k*:

⎧⎪ *β α H w*i if *β /*=? and *w*i */*=?

*newThreshold*(*β, α, w*

⎪⎨ *β α* if *β /*=? and *w*i =?

i) =

*w*i if *β* =? and *w*i */*=?

⎪

⎪⎩ ? if *β* =? and *w*i =?

Note that “*W ±*? *β,* Δ” represents a set of qualification constraints including the threshold constraint *W ±*? *β* plus those in Δ with no particular ordering assumed. From the threshold constraint for *W* in *G* and the new constraints in Δ1 of *G*1 (particularly the new defining constraint for *W* ) easily follows that *α ±*? *β* must hold, therefore such condition can be required, without loss of completeness, to actually enable the resolution step. Moreover, the values *β*i are computed by means of the auxiliary operation *newThreshold* so that the new threshold constraints *W*i *±*? *β*i in conjunction with the defining constraint for *W* in *G*1 imply the threshold constraint *W ±*? *β* in *G* and the threshold constraints *W*i *±*? *w*i encoded in the body of *C*1. For instance, in the case that *β /*= ? and *w*i */*= ?, one must have *α ⊗ W*i *± β* due to *W ± β* and the defining constraint for *W* in *G*1. But *α⊗W*i *± β* is equivalent to *W*i *± β α* (see Proposition [2.1](#_bookmark2)), and *W*i *± β α* in conjunction with *W*i *± w*i yields *W*i *± β α H w*i. The other three cases can be argued similarly.

It is easy to check that *G*1 is again a legal goal whenever *G* is a goal and *G* HC1,σ1 *G*1. Moreover, in the instance BQLP(*У*) all the qualification values and constraints become trivial, so that QSLD(*У*) boils down to classical SLD resolution. The next two theorems relate QSLD(*Ð*) resolution to the declarative semantics of BQLP(*Ð*)-programs. The Soundness Theorem [4.5](#_bookmark10) guarantees that every computed

answer is correct in the sense that it is a solution of the given goal. The Strong Completeness Theorem [4.6](#_bookmark11) ensures that, for any solution of a given goal and any

fixed selection strategy, QSLD(*Ð*) resolution is able to compute an equal, if not better, solution. The proofs are analogous to those given for the QLP(*Ð*) scheme in [[12](#_bookmark25),[13](#_bookmark24)], using inductive techniques similar to those presented in [[15](#_bookmark29)] for classical *SLD* resolution. Example [4.7](#_bookmark12) below illustrates the Completeness Theorem.

**Theorem 4.5 (Soundness)** *Assume G*0 H*∗ G and G* = *σ* Δ *solved. Let* (*σ, μ*) *be the solution associated to G. Then* (*σ, μ*) *–called the computed answer– is a solution of G*0*.*

▢

**Theorem 4.6 (Strong Completeness)** *Assume a given solution* (*θ, ρ*) *for G*0 *and any ﬁxed strategy for choosing the selected atom at each resolution step. Then there is some computed answer* (*σ, μ*) *for G*0 *which subsumes* (*θ, ρ*)*.*

## Example 4.7

1. The following QSLD(*С* ) computation solves the goal for program *УU* presented in Example [4.3](#_bookmark9).

eats(father(X),Y)#(ff,W1), human(father(X))#(tt,W2) |

W1 >= 0.4, W2 >= 0.6 H*eats.*9*,{X'→*eve*}*

eats(eve,Y)#(ff,W3), human(father(eve))#(tt,W2) |

*{* X *'→* eve *}* |

W1 = 0.8 \* min *{*W3*}*, W2 >= 0.6, W3 >= 0.4/0.8 H*eats.*7*,{Y '→*bird*}*

animal(bird)#(tt,W4), human(father(eve))#(tt,W2) |

*{* X *'→* eve, Y *'→* bird *}* |

W1 = 0.8 \* min *{*W3*}*, W2 >= 0.6,

W3 = 0.7 \* min *{*W4*}*, W4 >= (0.4/0.8)/0.7 H*animal.*1*,c*

human(father(eve))#(tt,W2) |

*{* X *'→* eve, Y *'→* bird *}* |

W1 = 0.8 \* min *{* W3 *}*, W2 >= 0.6,

W3 = 0.7 \* min *{*W4*}*, W4 = 1.0 H*human.*3*,c*

human(eve)#(tt,W5) |

*{* X *'→* eve, Y *'→* bird *}* |

W1 = 0.8 \* min *{*W3*}*, W2 = 0.9 \* min *{*W5*}*,

W3 = 0.7 \* min *{*W4*}*, W4 = 1.0, W5 >= 0.6/0.7 H*human.*2*,c*

| *{* X *'→* eve, Y *'→* bird *}* |

W1 = 0.8 \* min *{*W3*}*, W2 = 0.9 \* min *{*W5*}*, W3 = 0.7 \* min *{*W4*}*, W4 = 1.0, W5 = 1.0

Note that the computed answer *{*X *'→* eve, Y *'→* bird*}* | *{*W1 *'→* 0.56, W2 *'→* 0.9*}* subsumes the solution for the same goal given in Example [4.3](#_bookmark9), because it has a higher certainty for both atoms.

1. Similarly, QSLD(*С ×У*) resolution can solve the goal eats(X,Y)#(tt,W) | W *±* (0.5,4) for *УU×W* , obtaining a computed answer *{*X *'→* mother(adam)*}*

| *{*W *'→* (0.56,2)*}* which subsumes the solution for the same goal given in Example [4.3](#_bookmark9).

# Towards an Implementation

The implementation technique proposed in [[13](#_bookmark24)] for the QLP(*Ð*) scheme can be easily adapted to BQLP(*Ð*). Assuming a qualification domain *Ð* and a constraint

domain *CD* such that the qualification constraints used in QSLD(*Ð*) resolution can be expressed as *CD* constraints, a translation of a given BQLP(*Ð*)-program *У* with goal *G* into a CLP(*CD*)-program *У*t with goal *G*t can be specified in such a way that solving *G* with QSLD(*Ð*) resolution using *У* corresponds to solving *G*t with constrained SLD resolution using *У*t and a solver for *CD*. The translation can be used to develop an implementation of QSLD(*Ð*) resolution for the BQLP(*Ð*) language on top of any CLP or CFLP system that supports *CD* constraints.

The translation of a BQLP(*Ð*) program works by adding three extra arguments to all predicates and translating each clause independently. Given the BQLP(*Ð*) clause

*C ≡ p*(*t*) *v ←α— q*1(*s*1)(*v*1*, w*1)*,..., q*k(*s*k)(*v*k*, w*k) *,*

its head is translated as *p*(*t, v, W, B*), where the new variables *W* and *B* correspond, respectively, to *W* and *β* in the threshold constraint *W ±*? *β* related to an open annotated atom *A* (*v, W* ) which could be selected for a QSLD(*Ð*) resolution step using the clause *C*. The clause’s body is translated with the aim of emulating such a resolution step, and the translated clause becomes:

*C*t *≡ p*(*t, v, W, B*) *← α ±*? *B,*

*B*1 = *newThreshold*(*B, α, w*1)*, q*1(*s*1*, v*1*, W*1*, B*1)*,*

.

*B*k = *newThreshold*(*B, α, w*k)*, q*k(*s*k*, v*k*, W*k*, B*k)*, W* = *α ⊗* .*{W*1*,..., W*k*} .*

The idea for translating goals is similar. Given the initial QLP(*Ð*) goal

*G ≡ q*1(*t*1)(*v*1*, W*1)*,..., q*m(*t*m)(*v*m*, W*m) ▢ *W*1 *±*? *β*1*,..., W*m *±*? *β*m

where *β*1*,..., β*m *∈* (*D \ {⊥}*) *{*?*}*, the translated goal becomes

*G*t *≡ q*1(*t*1*, v*1*, W*1*, β*1)*,..., q*m(*t*m*, v*m*, W*m*, β*m) *.*

For three particular choices for *Ð*, namely *С* , *У* and *С ×У*, we have imple- mented the instance QLP(*Ð*) on top of the CFLP system *T OY* [[2](#_bookmark16)], which supports constraint solving over the *real constraint domain R*. The current implementation is expected to be distributed within the *T OY* system itself (as well as any further development), but until its next release, a special distribution of *T OY* with QLP(*Ð*) embedded is available at <http://gpd.sip.ucm.es/cromdia/qlpd>. There you will also find specific instructions for its installation and some examples for different instances to try it out. These three prototypes could be easily extended to support the corresponding BQLP(*Ð*) instances.

# Conclusions and Future Work

In [[13](#_bookmark24)] we had proposed a generic scheme QLP(*Ð*) for *Qualiﬁed Logic Programming* over a parametrically given qualification domain *Ð*, which generalized and improved a classical approach by van Emden [[19](#_bookmark33)] to *Quantitative Logic Programming*. In this paper, we have presented an extension of QLP(*Ð*) to a more expressive scheme BQLP(*Ð*) supporting *threshold constraints* in clause bodies and a simple kind of negation based on bivalued predicates. The new scheme BQLP(*Ð*) has a rigorous declarative semantics and a sound and strongly complete goal resolution proce- dure which can be implemented using constraint logic programming technology. As implementation technique, we have proposed a translation of BQLP(*Ð*) programs and goals into CLP(*CD*), choosing a constraint domain *CD* able to compute with qualification constraints over *Ð*. In our opinion, this implementation technique is efficient because it can support some interesting instances of our scheme (namely, BQLP(*С* ), BQLP(*У*) and BQLP(*С ×У*)) just by solving simple arithmetic con- straints and avoiding the costly computation of so-called *reductant clauses* needed in other approaches to logic programming with uncertainty, as e.g. the *GAP frame- work* [[6](#_bookmark20)] or the *multi-adjoint approach* in [[8](#_bookmark22),[9](#_bookmark23)].

As it was already the case for QLP(*Ð*), the BQLP(*Ð*) scheme improves the semantic results given in [[19](#_bookmark33)]. With respect to the alternative to [[19](#_bookmark33)] proposed in [[16](#_bookmark30),[17](#_bookmark31)], our approach is more general in that it can operate with any parametrically given qualification domain, and our attenuated clauses with threshold constraints in the body have a quite different expressivity in comparison to the simple annotated clauses used in [[16](#_bookmark30),[17](#_bookmark31)]. The theory of generalized annotated logic programs (GAP for short) presented more recently in [[6](#_bookmark20)] allows to express attenuated clauses, but the comparisons between GAP and QLP(*Ð*) given in the concluding section of [[13](#_bookmark24)] apply *mutatis mutandis* to BQLP(*Ð*), showing that our scheme has some points of advantage w.r.t. GAP.

An even more recent line of related work is *logic programming with similarity- based uniﬁcation* [[14](#_bookmark28),[7](#_bookmark21)], which can be applied to *flexible data retrieval* problems. In this approach, programs just consist of definite Horn clauses as in classical logic programming, but SLD resolution is modified to work with a generalized unification algorithm, so that a given *similarity relation* (roughly, the fuzzy analogon of an equivalence relation) permits to unify not identical but similar symbols. Unifiers become substitutions paired with a number *d ∈* (0*,* 1] which measures the degree of similarity between the (not necessarily identical) unified terms or atoms. In recent joint work with Rafael Caballero [[3](#_bookmark17)] we have presented an extension SQLP(*R, Ð*) of the QLP(*Ð*) scheme, which supports similarity-based reasoning using any similarity relation *R* over any qualification domain *Ð*. A main result given in [[3](#_bookmark17)] is a semantics preserving translation of SQLP(*R, Ð*) programs into QLP(*Ð*) programs, showing that implementations of QLP(*Ð*) instances can be used to support similarity-based reasoning. The approach in [[3](#_bookmark17)] could be easily extended to accommodate bivalued predicates in the sense of the current paper.

Some more or less close relations to our work can be also found in existing re- search on fuzzy logic programming. For instance, the approach to Fuzzy Prolog

presented in [[5](#_bookmark19)] is similar to our approach in using CLP with real arithmetic con- straints as an implementation tool, but rather different in other respects, since it uses elements of the Borel algebra over the interval [0*,* 1] as a sophisticated kind of fuzzy truth values. Some further comparisons between our approach and other approaches to computing with uncertainty and similarity in *LP* can be found in [[3](#_bookmark17)]. We plan future work along several lines. Firstly, we would like to improve our current implementation of QLP(*Ð*) instances, possibly incorporating bivalued pred- icates and unification modulo a given similarity relation. We also plan to perform benchmarks in order to check the implementation’s performance, in particular the execution overload introduced by adding qualifications to ordinary logic programs. Next, we plan to extend our current schemes for similarity-based qualified LP to a more expressive scheme which supports multiparadigm declarative programming with lazy functions, predicates and constraints. Some work on functional logic pro- gramming with similarity-based unification is already available [[10](#_bookmark26),[11](#_bookmark27)]. Finally, we would like to test the usefulness of our approach, focusing on applications to solving

flexible information retrieval problems.

# Acknowledgement

The authors are thankful to their colleagues Paco Lo´pez and Rafa Caballero for their valuable help concerning bibliography and implementation techniques as well as to the anonymous reviewers whose constructive comments aided us to improve this work.

# References

1. 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.
2. P. Arenas, A. J. Fern´andez, A. Gil, F. J. L´opez-Fraguas, M. Rodr´ıguez-Artalejo, and F. S´aenz-P´erez. *T OY*, a multiparadigm declarative language. version 2.3.1, 2007. R. Caballero and J. S´anchez (Eds.), Available at [http://toy.sourceforge.net.](http://toy.sourceforge.net/)
3. R. Caballero, M. Rodr´ıguez-Artalejo, and C. A. Romero-D´ıaz. Similarity-based reasoning in qualified logic programming. In *PPDP ’08: Proceedings of the 10th international ACM SIGPLAN conference* *on Principles and Practice of Declarative Programming*, pages 185–194, New York, NY, USA, 2008. ACM.
4. K. L. Clark. Predicate logic as a computational formalism (res. report doc 79/59). Technical report, Imperial College, Dept. of Computing, London, 1979.
5. S. Guadarrama, S. Mun˜oz, and C. Vaucheret. Fuzzy prolog: A new approach using soft constraint propagation. *Fuzzy Sets and Systems*, 144(1):127–150, 2004.
6. 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.
7. 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.
8. J. Medina, M. Ojeda-Aciego, and P. Vojt´aˇs. 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.
9. J. Medina, M. Ojeda-Aciego, and P. Vojt´aˇs. 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.
10. 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.

1. G. Moreno and V. Pascual. Formal properties of needed narrowing with similarity relations. *Electronic* *Notes in Theoretical Computer Science*, 188:21–35, 2007.
2. M. Rodr´ıguez-Artalejo and C. A. Romero-D´ıaz. A generic scheme for qualified logic programming (Technical Report SIC-1-08). Technical report, Universidad Complutense, Departamento de Sistemas Inform´aticos y Computaci´on, Madrid, Spain, 2008.
3. M. Rodr´ıguez-Artalejo and C. A. Romero-D´ıaz. 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.

1. M. I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275(1-2):389–426, 2002.
2. R. F. St¨ark. A direct proof for the completeness of SLD-resolution. In E. B¨orger, H. K. Bu¨ning, and

M. M. Richter, editors, *Proceedings of the 3rd Workshop on Computer Science Logic (CSL’89)*, volume 440 of *LNCS*, pages 382–383. Springer Verlag, 1990.

1. 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.
2. 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.
3. V. S. Subrahmanian. Uncertainty in logic programming: Some recollections. *Association for Logic Programming Newsletter*, 20(2), 2007.
4. M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986.
5. 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.