Fixpoint & Proof-theoretic Semantics for CLP with Qualification and Proximity ∗

Technical Report SIC-1-10

MARIO RODRÍGUEZ-ARTALEJO and CARLOS A. ROMERO-DÍAZ

Departamento de Sistemas Informáticos y Computación, Universidad Complutense
Facultad de Informática, 28040 Madrid, Spain
(*e-mail:* mario@sip.ucm.es, cromdia@fdi.ucm.es)

Abstract

Uncertainty in Logic Programming has been investigated during the last decades, dealing with various extensions of the classical LP paradigm and different applications. Existing proposals rely on different approaches, such as clause annotations based on uncertain truth values, qualification values as a generalization of uncertain truth values, and unification based on proximity relations. On the other hand, the CLP scheme has established itself as a powerful extension of LP that supports efficient computation over specialized domains while keeping a clean declarative semantics. In this report we propose a new scheme SQ-CLP designed as an extension of CLP that supports qualification values and proximity relations. We show that several previous proposals can be viewed as particular cases of the new scheme, obtained by partial instantiation. We present a declarative semantics for SQCLP that is based on observables, providing fixpoint and proof-theoretical characterizations of least program models as well as an implementation-independent notion of goal solutions.

KEYWORDS: Constraint Logic Programming, Qualification Domains and Values, Proximity Relations.

1 Introduction

Many extensions of logic programming (shortly LP) to deal with uncertainty have been proposed in the last decades. A line of research not related to this report is based on probabilistic extensions of LP such as (Ng and Subrahmanian 1992). Other proposals in the field replace classical two-valued logic by some kind of many-valued logic whose truth values can be attached to computed answers and are usually interpreted as certainty degrees. The next paragraphs summarize some relevant approaches of this kind.

There are extensions of LP using annotations in program clauses to compute a certainty degree for the head atom from the certainty degrees previously computed

∗ This work has been partially supported by the Spanish projects STAMP (TIN2008-06622-C03-01), PROMETIDOS–CM (S2009TIC-1465) and GPD–UCM (UCM–BSCH–GR58/08-910502).

for the body atoms. This line of research includes the seminal proposal of Quantitative Logic Programming by (van Emden 1986) and inspired later works such as the Generalized Annotated logic Programs (shortly GAP) by (Kifer and Subrahmanian 1992) and the QLP scheme for Qualified LP (Rodríguez-Artalejo and Romero-Díaz 2008b). While (van Emden 1986) and other early approaches used real numbers of the interval $[0, 1]$ as certainty degrees, QLP and GAP take elements from a parametrically given lattice to be used in annotations and attached to computed answers. In the case of QLP, the lattice is called a *qualification domain* and its elements (called *qualification values*) are not always understood as certainty degrees. As argued in (Rodríguez-Artalejo and Romero-Díaz 2008b), GAP is a more general framework, but QLP's semantics have some advantages for its intended scope.

There are also extended LP languages based on fuzzy logic (Zadeh 1965; Hájek 1998), which can be classified into two major lines. The first line includes Fuzzy LP languages such as (Vojtáš 2001; Vaucheret et al. 2002; Guadarrama et al. 2004) and the Multi-Adjoint LP (shortly MALP) framework by (Medina et al. 2001a; Medina et al. 2001b). All these approaches extend classical LP by using clause annotations and a fuzzy interpretation of the connectives and aggregation operators occurring in program clauses and goals. There is a relationship between Fuzzy LP and GAP that has been investigated in (Krajči et al. 2004). Intended applications of Fuzzy LP languages include expert knowledge representation.

The second line includes Similarity-based LP (shortly SLP) in the sense of (Arcelli and Formato 1999; Sessa 2002; Loia et al. 2004) and related proposals, which keep the classical syntax of LP clauses but use a *similarity relation* over a set of symbols S to allow "flexible" unification of syntactically different symbols with a certain approximation degree. Similarity relations over a given set S have been defined in (Zadeh 1971; Sessa 2002) and related literature as fuzzy relations represented by mappings $\mathcal{S} : S \times S \to [0, 1]$ which satisfy reflexivity, symmetry and transitivity axioms analogous to those required for classical equivalence relations. A more general notion called *proximity relation* was introduced in (Dubois and Prade 1980) by omitting the transitivity axiom. As noted by (Shenoi and Melton 1999) and other authors, the transitivity property required for similarity relations may conflict with user's intentions in some cases. The Bousi∼Prolog language (Julián-Iranzo et al. 2009; Julián-Iranzo and Rubio-Manzano 2009b; Julián-Iranzo and Rubio-Manzano 2009a) has been designed with the aim of generalizing SLP to work with proximity relations. A different generalization of SLP is the SQLP scheme (Caballero et al. 2008), designed as an extension of the QLP scheme. In addition to clause annotations in QLP style, SQLP uses a given similarity relation $\mathcal{S} : S \times S \to D$ (where D is the carrier set of a parametrically given qualification domain) in order to support flexible unification. In the sequel we use the acronym SLP as including proximity-based LP languages also. Intended applications of SLP include flexible query answering. An analogy of proximity relations in a different context (namely partial constraint satisfaction) can be found in (Freuder and Wallace 1992), where several metrics are proposed to measure the proximity between the solution sets of two different constraint satisfaction problems.

Several of the above mentioned LP extensions (including GAP, QLP, the Fuzzy

LP language in (Guadarrama et al. 2004) and SQLP) have used constraint solving as an implementation technique. However, we only know two approaches which have been conceived as extensions of the classical CLP scheme (Jaffar and Lassez 1987). Firstly, (Riezler 1996; Riezler 1998) extended the formulation of CLP by (Höhfeld and Smolka 1988) with quantitative LP in the sense of (van Emden 1986); this work was motivated by problems from the field of natural language processing. Secondly, (Bistarelli et al. 2001) proposed a semiring-based approach to CLP, where constraints are solved in a soft way with levels of consistency represented by values of a semiring. This approach was motivated by constraint satisfaction problems and implemented with `clp(FD,S)` in (Georget and Codognet 1998) for a particular class of semirings which enable to use local consistency algorithms. The relationship between (Riezler 1996; Riezler 1998; Bistarelli et al. 2001) and the results of this report will be further discussed in Section 4.

Finally, there are a few preliminary attempts to combine some of the above mentioned approaches with the Functional Logic Programming (shortly FLP) paradigm found in languages such as Curry (Hanus) and $\mathcal{TOY}$ (Arenas et al. 2007). Similarity-based unification for FLP languages has been investigated by (Moreno and Pascual 2007), while (Caballero et al. 2009) have proposed a generic scheme QCFLP designed as a common extension of the two schemes CLP and QLP with first-order FLP features.

In this report we propose a new extension of CLP that supports qualification values and proximity relations. More precisely, we define a generic scheme SQCLP whose instances SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) are parameterized by a proximity relation $\mathcal{S}$, a qualification domain $\mathcal{D}$ and a constraint domain $\mathcal{C}$. We will show that several previous proposals can be viewed as particular cases of SQCLP, obtained by partial instantiation. Moreover, we will present a declarative semantics for SQCLP that is inspired in the observable CLP semantics by (Gabbrielli and Levi 1991; Gabbrielli et al. 1995) and provides fixpoint and proof-theoretical characterizations of least program models as well as an implementation-independent notion of goal solution that can be used to specify the expected behavior of goal solving systems.

The reader is assumed to be familiar with the semantic foundations of LP (Lloyd 1987; Apt 1990) and CLP (Jaffar and Lassez 1987; Jaffar et al. 1998). The rest of the report is structured as follows: Section 2 introduces constraint domains, qualification domains and proximity relations. Section 3 presents the SQCLP scheme and the main results on its declarative semantics. Finally, Section 4 concludes by giving an overview of related approaches (many of which can be viewed as particular cases of SQCLP) and pointing to some lines open for future work.

2 Constraints, Qualification & Proximity

2.1 Constraint Domains

The Constraint Logic Programming paradigm (CLP) was introduced in (Jaffar and Lassez 1987) with the aim of generalizing the Herbrand Universe which underlies classical Logic Programming (LP) to other domains tailored to specific applica-

tion areas. In this seminal paper, CLP was introduced as a generic scheme with instances CLP($\mathcal{C}$) parameterized by *constraint domains* $\mathcal{C}$, each of which supplies several items: a *constraint language* providing a class of domain specific formulae, called *constraints* and serving as logical conditions in CLP($\mathcal{C}$) programs and computations; a *constraint structure* serving as interpretation of the constraint language; a *constraint theory* serving as a basis for proof-theoretical deduction with constraints; and a *constraint solver* for checking constraint satisfiability. Certain assumptions were made to ensure the proper relationship between the constraint language, structure, theory and solver, so that the classical results on the operational and declarative semantics of LP (Lloyd 1987; Apt 1990) could be extended to all the CLP($\mathcal{C}$) languages. A revised and updated presentation of the main results from (Jaffar and Lassez 1987) can be found in (Jaffar et al. 1998), while a survey of CLP as a programming paradigm is given in (Jaffar and Maher 1994).

The notion of constraint domain is a key ingredient of the CLP scheme. In addition to the classical formulation in (Jaffar and Lassez 1987; Jaffar et al. 1998), other formalizations have been used for different purposes. Some significative examples are: the CLP scheme proposed in (Höhfeld and Smolka 1988), motivated by applications to computational linguistics and allowing more than one constraint structure to come along with a given constraint language; the proof-theoretical notion of constraint system given in (Saraswat 1992), intended for application to concurrent constraint languages; and the constraint systems proposed in (Lucio et al. 2008) as the basis of a functorial semantics for CLP with negation, where a single constraint structure is replaced by a class of elementary equivalent structures.

In this paper we will use a simple notion of constraint domain, motivated by three main considerations: firstly, to focus on declarative semantics, rather than proof-theoretic or operational issues; secondly, to provide a purely relational framework; and thirdly, to clarify the interplay between domain-specific programming resources such as basic values and primitive predicates, and general-purpose programming resources such as data constructors and defined predicates.

2.1.1 Preliminary notions

Before presenting constraint domains in a formal way, let us introduce some mainly syntactic notions that will be used all along the paper.

Definition 2.1 (*Signatures*)
We assume a *universal programming signature* $\Gamma = \langle DC, DP \rangle$ where $DC = \bigcup_{n\in\mathbb{N}} DC^n$ and $DP = \bigcup_{n\in\mathbb{N}} DP^n$ are infinite and mutually disjoint sets of free function symbols (called *data constructors* in the sequel) and *defined predicate* symbols, respectively, ranked by arities. We will use *domain specific signatures* $\Sigma = \langle DC, DP, PP \rangle$ extending Γ with a disjoint set $PP = \bigcup_{n\in\mathbb{N}} PP^n$ of *primitive predicate* symbols, also ranked by arities. The idea is that primitive predicates come along with constraint domains, while defined predicates are specified in user programs. Each PP^n maybe any countable set of n-ary predicate symbols. In practice, PP is expected to be a finite set. □

In the sequel, we assume that any signature Σ includes two nullary constructors $\mathsf{true}, \mathsf{false} \in DC^0$ to represent the boolean values, a binary constructor $\mathsf{pair} \in DC^2$ to represent ordered pairs, as well as constructors to represent lists and other common data structures. Given a signature Σ, a set B of *basic values* u and a countably infinite set $\mathcal{V}ar$ of variables X, *terms* and *atoms* are built as defined below, where $\overline{o}_n$ abbreviates the n-tuple of syntactic objects $o_1, \ldots, o_n$ and $\mathrm{var}(o)$ denotes the set of all variables occurring in the syntactic object o.

Definition 2.2 (*Terms and atoms*)

- *Constructor Terms* $t \in \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ have the syntax $t ::= X|u|c(\overline{t}_n)$, where $c \in DC^n$. They will be called just terms in the sequel. In concrete examples, we will use Prolog syntax for terms built with list constructors, and we will write (t_1, t_2) rather than $\mathsf{pair}(t_1, t_2)$ for terms representing ordered pairs.
- The set of all the variables occurring in t is noted as $\mathrm{var}(t)$. A term t is called *ground* iff $\mathrm{var}(t) = \emptyset$. $\mathrm{Term}(\Sigma, B)$ stands for the set of all ground terms.
- *Atoms* $A \in \mathrm{At}(\Sigma, B, \mathcal{V}ar)$ can be *defined atoms* $r(\overline{t}_n)$, where $r \in DP^n$ and $t_i \in \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ $(1 \leq i \leq n)$; *primitive atoms* $p(\overline{t}_n)$, where $p \in PP^n$ and $t_i \in \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ $(1 \leq i \leq n)$; and *equations* $t_1 == t_2$, where $t_1, t_2 \in \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ and '==' is the *equality symbol*, which does not belong to the signature Σ. Primitive atoms are noted as κ and the set of all primitive atoms is noted $\mathrm{PAt}(\Sigma, B, \mathcal{V}ar)$. Equations and primitive atoms are collectivelly called *$\mathcal{C}$-based atoms*.
- The set of all the variables occurring in A is noted as $\mathrm{var}(A)$. An atom A is called *ground* iff $\mathrm{var}(A) = \emptyset$. The set of all ground atoms (resp. ground primitive atoms) is noted as $\mathrm{GAt}(\Sigma, B)$ (resp. $\mathrm{GPAt}(\Sigma, B)$). □

Note that the equality symbol '==' used as part of the syntax of equational atoms is not the same as the symbol '=' generally used for mathematical equality. In particular, metalevel equations $o = o'$ can be used to assert the identity of two syntactical objects o and o'.

Following well-known ideas, the syntactical structure of terms and atoms can be represented by means of trees with nodes labeled by signature symbols, basic values and variables. In the sequel we will use the notation $\|t\|$ to denote the *syntactical size* of t measured as the number of nodes in the tree representation of t. The *positions* of nodes in this tree can be noted as finite sequences p of natural numbers. In particular, the empty sequence ε represents the root position. The next definition presents essential notions concerning positions in terms. Positions in atoms can be treated similarly.

Definition 2.3 (*Positions*)

1. The set $\mathrm{pos}(t)$ of positions of the term t is defined by recursion on the structure of t:
   - $\mathrm{pos}(X) = \{\varepsilon\}$ for each variable $X \in \mathcal{V}ar$.
   - $\mathrm{pos}(u) = \{\varepsilon\}$ for each basic value $u \in B$.
   - $\mathrm{pos}(c(\overline{t}_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{iq \mid q \in \mathrm{pos}(t_i)\}$ for each $c \in DC^n$.

2. Given $p \in \mathrm{pos}(t)$, the symbol $t \circ p$ of t at position p is defined recursively:
   - $X \circ \varepsilon = X$ for each variable $X \in \mathcal{V}ar$.
   - $u \circ \varepsilon = u$ for each basic value $u \in B$.
   - $c(t_1, \ldots, t_n) \circ \varepsilon = c$ if $c \in DC^n$.
   - $c(t_1, \ldots, t_n) \circ iq = t_i \circ q$ if $c \in DC^n$, $1 \leq i \leq n$ and $q \in \mathrm{vpos}(t_i)$.
3. Given $p \in \mathrm{pos}(t)$, the subterm $t|_p$ of t at position p is defined as follows:
   - $t|_\varepsilon = t$ for any t.
   - $c(t_1, \ldots, t_n)|_{iq} = t_i|_q$ if $c \in DC^n$, $1 \leq i \leq n$ and $q \in \mathrm{pos}(t_i)$.
4. $p \in \mathrm{pos}(t)$ is called a *variable position* of t iff $t|p$ is a variable, and a *rigid position* of t otherwise. We define $\mathrm{vpos}(t) = \{p \in \mathrm{pos}(t) \mid p \text{ is a variable position}\}$ and $\mathrm{rpos}(t) = \{p \in \mathrm{pos}(t) \mid p \text{ is a rigid position}\}$.
5. Given $p \in \mathrm{vpos}(t)$ and another term s, the result of replacing s for the subterm of t at position p is noted as $t[s]_p$. See e.g. (Baader and Nipkow 1998) for a recursive definition. □

As usual, *substitutions* are defined as mappings $\sigma : \mathcal{V}ar \to \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ assigning terms to variables. The set of all substitutions is noted as $\mathrm{Subst}(\Sigma, B, \mathcal{V}ar)$. Substitutions are extended to act over terms and other syntactic objects o in the natural way. By convention, the result of replacing each variable X occurring in o by $\sigma(X)$ is noted as $o\sigma$. Other common notions concerning substitutions are defined as follows:

Definition 2.4 (*Notions concerning Substitutions*)
- The *composition* $\sigma\sigma'$ of two substitutions is such that $o(\sigma\sigma')$ equals $(o\sigma)\sigma'$.
- For a given $\sigma \in \mathrm{Subst}(\Sigma, B, \mathcal{V}ar)$, the *domain* $\mathrm{dom}(\sigma)$ is defined as $\{X \in \mathcal{V}ar \mid X\sigma \neq X\}$, and the *variable range* $\mathrm{vran}(\sigma)$ is defined as $\bigcup_{X \in \mathrm{dom}(\sigma)} \mathrm{var}(X\sigma)$.
- A substitution σ is called *ground* iff $X\sigma$ is a ground term for all $X \in \mathrm{dom}(\sigma)$. The set of all ground substitutions is noted $\mathrm{GSubst}(\Sigma, B)$.
- A substitution σ is called *finite* iff $\mathrm{dom}(\sigma)$ is a finite set, say $\{X_1, \ldots, X_k\}$. In this case, σ can be represented as the *set of bindings* $\{X_1 \mapsto t_1, \ldots, X_k \mapsto t_k\}$, where $t_i = X_i\sigma$ for all $1 \leq i \leq k$.
- Assume two substitutions σ, σ', a set of variables $\mathcal{X}$ and a variable Y. The notation $\sigma =_{\mathcal{X}} \sigma'$ means that $X\sigma = X\sigma'$ holds for all variables $X \in \mathcal{X}$. We also write $\sigma =_{\setminus \mathcal{X}} \sigma'$ and $\sigma =_{\setminus Y} \sigma'$ to abbreviate $\sigma =_{\mathcal{V}ar \setminus \mathcal{X}} \sigma'$ and $\sigma =_{\mathcal{V}ar \setminus \{Y\}} \sigma'$, respectively. □

2.1.2 Constraint domains, constraints and their solutions

We are now prepared to present constraint domains as mathematical structures providing a set of basic values along with an terms and an interpretation of primitive predicates[1]. The formal definition is as follows:

[1] As we will see in Section 3, the interpretation of defined predicate symbols is program dependent.

Definition 2.5 (*Constraint Domains*)
A *Constraint Domain* of signature Σ is any relational structure of the form $\mathcal{C} = \langle C, \{p^{\mathcal{C}} \mid p \in PP\}\rangle$ such that:

1. The carrier set C is $\mathrm{Term}(\Sigma, B)$ for a certain set B of *basic values*. When convenient, we note B and C as $B_{\mathcal{C}}$ and $C_{\mathcal{C}}$, respectively.
2. $p^{\mathcal{C}} : C^n \to \{0,1\}$, written simply as $p^{\mathcal{C}} \in \{0,1\}$ in the case $n = 0$, is called the *interpretation* of p in $\mathcal{C}$. A ground primitive atom $p(\bar{t}_n)$ is *true* in $\mathcal{C}$ iff $p^{\mathcal{C}}(\bar{t}_n) = 1$; otherwise $p(\bar{t}_n)$ is *false* in $\mathcal{C}$. □

For the examples in this paper we will use a constraint domain $\mathcal{R}$ which allows to work with arithmetic constraints over the real numbers, as formalized in Definition 2.6 below.

Definition 2.6 (*The Real Constraint Domain* $\mathcal{R}$)
The constraint domain $\mathcal{R}$ is defined to include:

- The set of basic values $B_{\mathcal{R}} = \mathbb{R}$. Note that $C_{\mathcal{R}}$ includes ground terms built from real values and data constructors, in addition to real numbers.
- Primitive predicates for encoding the usual arithmetic operations over $\mathbb{R}$. For instance, the addition operation $+$ over $\mathbb{R}$ is encoded by a ternary primitive predicate op_+ such that, for any $t_1, t_2 \in C_{\mathcal{R}}$, $op_+(t_1, t_2, t)$ is true in $\mathcal{R}$ iff $t_1, t_2, t \in \mathbb{R}$ and $t_1 + t_2 = t$. In particular, $op_+(t_1, t_2, t)$ is false in $\mathcal{R}$ if either t_1 or t_2 includes data constructors. The primitive predicates encoding other arithmetic operations such as $\times$ and $-$ are defined analogously.
- Primitive predicates for encoding the usual inequality relations over $\mathbb{R}$. For instance, the ordering $\leq$ over $\mathbb{R}$ is encoded by a binary primitive predicate $cp_{\leq}$ such that, for any $t_1, t_2 \in C_{\mathcal{R}}$, $cp_{\leq}(t_1, t_2)$ is true in $\mathcal{R}$ iff $t_1, t_2, t \in \mathbb{R}$ and $t_1 \leq t_2$. In particular, $cp_{\leq}(t_1, t_2)$ is false in $\mathcal{R}$ if either t_1 or t_2 includes data constructors. The primitive predicates encoding the other inequality relations, namely $>$, $\geq$ and $>$, are defined analogously. □

The domain $\mathcal{R}$ is well known as the basis of the CLP($\mathcal{R}$) language and system (Jaffar et al. 1992). Some presentations of $\mathcal{R}$ known in the literature represent the arithmetical operations by using primitive functions instead of primitive predicates. In this paper we have chosen to work in a purely relational framework in order to simplify some technicalities without loss of real expressivity.

Other useful instances of constraint domains are known in the Constraint Programming literature; see e.g. (Jaffar and Maher 1994; López-Fraguas et al. 2007). In particular, the *Herbrand* domain $\mathcal{H}$ is intended to work just with equality constraints, while $\mathcal{FD}$ allows to work with constraints involving *finite domain variables*. The set of basic values of $\mathcal{FD}$ is $\mathbb{Z}$. There are also known techniques for combining several given constraint domains into a more expressive one; see e.g. the *coordination domains* defined in (Estévez-Martín et al. 2009).

The following definition introduces constraints over a given domain:

Definition 2.7 (*Constraints and Their Solutions*)
Given a constraint domain $\mathcal{C}$ of signature Σ:

1. *Atomic constraints* over $\mathcal{C}$ are of two kinds: primitive atoms $p(\bar{t}_n)$ and equations $t_1 == t_2$.
2. *Compound constraints* are built from atomic constraints using logical conjunction $\wedge$, existential quantification $\exists$, and sometimes other logical operations. Constraints of the form $\exists X_1 \ldots \exists X_n (B_1 \wedge \ldots \wedge B_m)$ –where B_j $(1 \leq j \leq m)$ are atomic– are called *existential*. The set of all constraints over $\mathcal{C}$ is noted $\mathrm{Con}_{\mathcal{C}}$.
3. Substitutions $\sigma : \mathcal{V}ar \to \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ where $\mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ is built using the set $B_{\mathcal{C}}$ of basic values are called $\mathcal{C}$-*substitutions*. Ground substitutions $\eta \in \mathrm{GSubst}(\Sigma, B)$ are called *variable valuations*. The set of all possible variable valuations is noted $\mathrm{Val}_{\mathcal{C}}$.
4. *The solution set* $\mathrm{Sol}_{\mathcal{C}}(\pi)$ of a constraint $\pi \in \mathrm{Con}_{\mathcal{C}}$ is defined by recursion on π's syntactic structure as follows:
   - If π is a primitive atom $p(\bar{t}_n)$, then $\mathrm{Sol}_{\mathcal{C}}(\pi)$ is the set of all $\eta \in \mathrm{Val}_{\mathcal{C}}$ such that $p(\bar{t}_n)\eta$ is ground and true in $\mathcal{C}$.
   - If π is an equation $t_1 == t_2$, then $\mathrm{Sol}_{\mathcal{C}}(\pi)$ is the set of all $\eta \in \mathrm{Val}_{\mathcal{C}}$ such that $t_1\eta$ and $t_2\eta$ are ground and syntactically identical terms.
   - If π is $\pi_1 \wedge \pi_2$ then $\mathrm{Sol}_{\mathcal{C}}(\pi) = \mathrm{Sol}_{\mathcal{C}}(\pi_1) \cap \mathrm{Sol}_{\mathcal{C}}(\pi_2)$.
   - If π is $\exists X \pi'$ then $\mathrm{Sol}_{\mathcal{C}}(\pi)$ is the set of all $\eta \in \mathrm{Val}_{\mathcal{C}}$ such that $\eta' \in \mathrm{Sol}_{\mathcal{C}}(\pi')$ holds for some $\eta' \in \mathrm{Val}_{\mathcal{C}}$ verifying $\eta =_{\setminus X} \eta'$.

   π is called *satisfiable* over $\mathcal{C}$ iff $\mathrm{Sol}_{\mathcal{C}}(\pi) \neq \emptyset$, and π is called *unsatisfiable* over $\mathcal{C}$ iff $\mathrm{Sol}_{\mathcal{C}}(\pi) = \emptyset$.
5. *The solution set* $\mathrm{Sol}_{\mathcal{C}}(\Pi)$ of a set Π of constraints is defined as $\bigcap_{\pi \in \Pi} \mathrm{Sol}_{\mathcal{C}}(\pi)$. In this way, finite sets of constraints are interpreted as the conjunction of their members. Π is called *satisfiable* over $\mathcal{C}$ iff $\mathrm{Sol}_{\mathcal{C}}(\Pi) \neq \emptyset$, and Π is called *unsatisfiable* over $\mathcal{C}$ iff $\mathrm{Sol}_{\mathcal{C}}(\Pi) = \emptyset$.
6. A constraint π *is entailed* by a set of constraints Π (in symbols, $\Pi \models_{\mathcal{C}} \pi$) iff $\mathrm{Sol}_{\mathcal{C}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{C}}(\pi)$. □

The following example illustrates the previous definition:

Example 2.1 (*Constraint solutions and constraint entailment over* $\mathcal{R}$)
Consider the set of constraints $\Pi = \{cp_{\geq}(A, 3.0),\ op_{+}(A, A, X), op_{\times}(2.0, A, Y)\} \subseteq \mathrm{Con}_{\mathcal{R}}$. Then:

1. For any valuation $\eta \in \mathrm{Val}_{\mathcal{R}}$: $\eta \in \mathrm{Sol}_{\mathcal{R}}(\Pi)$ holds iff $\eta(A)$, $\eta(X)$ and $\eta(Y)$ are real numbers $a, x, y \in \mathbb{R}$ such that $a \geq 3.0$, $a + a = x$ and $2.0 \times a = y$.
2. Due to the previous item, the following $\mathcal{R}$-entailments are valid:
   (a) $\Pi \models_{\mathcal{R}} cp_{>}(X, 5.5)$, because $\mathrm{Sol}_{\mathcal{R}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{R}}(cp_{>}(X, 5.5))$.
   (b) $\Pi \models_{\mathcal{R}} X == Y$, because $\mathrm{Sol}_{\mathcal{R}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{R}}(\mathrm{X} == \mathrm{Y})$.
   (c) $\Pi \models_{\mathcal{R}} c(X) == c(Y)$, because $\mathrm{Sol}_{\mathcal{R}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{R}}(\mathrm{c(X)} == \mathrm{c(Y)})$. Here we assume $c \in DC^1$.

(d) $\Pi \models_{\mathcal{R}} [X,Y] == [Y,X]$, because $\mathrm{Sol}_{\mathcal{R}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{R}}([\mathrm{X},\mathrm{Y}] == [\mathrm{Y},\mathrm{X}])$. Here, the terms $[X,Y]$ and $[Y,X]$ are built from variables and list constructors. □

The next technical result will be useful later on:

Lemma 2.1 (*Substitution Lemma*)
Assume a set of constraints $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$ and a $\mathcal{C}$-substitution σ. Then:

1. For any valuation $\eta \in \mathrm{Val}_{\mathcal{C}}$: $\eta \in \mathrm{Sol}_{\mathcal{C}}(\Pi\sigma) \iff \sigma\eta \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$.
2. For any constraint $\pi \in \mathrm{Con}_{\mathcal{C}}$: $\Pi \models_{\mathcal{C}} \pi \Longrightarrow \Pi\sigma \models_{\mathcal{C}} \pi\sigma$.

Proof
Let us give a separate reasoning for each item.

1. The following statement holds for any constraint $\pi \in \mathrm{Con}_{\mathcal{C}}$:

$$(\star)\ \ \eta \in \mathrm{Sol}_{\mathcal{C}}(\pi\sigma) \iff \sigma\eta \in \mathrm{Sol}_{\mathcal{C}}(\pi)$$

In fact, $(\star)$ can can be easily proved reasoning by induction on the syntactic structure of π. Now, using $(\star)$ we can reason as follows:

$$\eta \in \mathrm{Sol}_{\mathcal{C}}(\Pi\sigma) \iff \eta \in \mathrm{Sol}_{\mathcal{C}}(\pi\sigma) \text{ for all } \pi \in \Pi \iff_{(\star)}$$
$$\sigma\eta \in \mathrm{Sol}_{\mathcal{C}}(\pi) \text{ for all } \pi \in \Pi \iff \sigma\eta \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$$

2. Assume $\Pi \models_{\mathcal{C}} \pi$. For the sake of proving $\Pi\sigma \models_{\mathcal{C}} \pi\sigma$, also assume an arbitrary $\eta \in \mathrm{Sol}_{\mathcal{C}}(\Pi\sigma)$. Then we get $\sigma\eta \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ because of item 1 and $\sigma\eta \in \mathrm{Sol}_{\mathcal{C}}(\pi)$ due to the assumption $\Pi \models_{\mathcal{C}} \pi$, which implies $\eta \in \mathrm{Sol}_{\mathcal{C}}(\pi\sigma)$ again because of item 1. Since η is arbitrary, we have proved $\mathrm{Sol}_{\mathcal{C}}(\Pi\sigma) \subseteq \mathrm{Sol}_{\mathcal{C}}(\pi\sigma)$, i.e. $\Pi\sigma \models_{\mathcal{C}} \pi\sigma$. □

2.1.3 Term equivalence w.r.t. a given constraint set

Given two terms t, s we will use the notation $t \approx_{\Pi} s$ (read as t and s are Π-*equivalent*) as an abbreviation of $\Pi \models_{\mathcal{C}} t == s$, assuming that the constraint domain $\mathcal{C}$ and the constraint set $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$ are known. For the sake of simplicity, $\mathcal{C}$ is not made explicit in the $\approx_{\Pi}$ notation. In this subsection we present some properties related to $\approx_{\Pi}$ which will be needed later. First, we prove that $\approx_{\Pi}$ is an equivalence relation with a natural characterization.

Lemma 2.2 (Π-*Equivalence Lemma*)
1. $\approx_{\Pi}$ is an equivalence relation over $\mathrm{Term}(\Sigma, B, \mathcal{V}ar)$.
2. For any given terms t and s the following two statements are equivalent:
   (a) $t \approx_{\Pi} s$.
   (b) For any common position $p \in \mathrm{pos}(t) \cap \mathrm{pos}(s)$ some of the cases below holds:
      i $t \circ p$ or $s \circ p$ is a variable, and moreover $t|_p \approx_{\Pi} s|_p$.
      ii $t \circ p = s \circ p = u$ for some $u \in B_{\mathcal{C}}$.
      iii $t \circ p = s \circ p = c$ for some $n \in \mathbb{N}$ and some $c \in DC^n$.
3. $\approx_{\Pi}$ boils down to the syntactic equality relation $=$ when Π is the empty set.

Proof
We give a separate reasoning for each item.

1. Checking that $\approx_\Pi$ satisfies the axioms of an equivalence relation (i.e. *reflexivity*, *symmetry* and *transitivity*) is quite obvious.
2. Due to Definition 2.7, $t \approx_\Pi s$ holds iff $t\eta$ and $s\eta$ are identical ground terms for each $\eta \in \mathrm{Sol}_\mathcal{C}(\Pi)$. This statement can be proved equivalent to condition 2.(b) reasoning by induction on $\|t\| + \|s\|$.
3. Note that $t \approx_\emptyset s$ holds iff $t\eta$ and $s\eta$ are identical ground terms for each $\eta \in \mathrm{Sol}_\mathcal{C}(\emptyset) = \mathrm{Val}_\mathcal{C}$. This can happen iff t and s are syntactically identical. □

Since the set $\mathcal{V}ar$ of all variables is countably infinite, we can assume an arbitrarily fixed bijective mapping $\mathrm{ord} : \mathcal{V}ar \to \mathbb{N}$. By convention, $\mathrm{ord}(X)$ is called the ordinal number of X. The notions defined below rely on this convention.

Definition 2.8 (Π-*Canonical Variables and Terms*)
1. A variable X is called Π-*canonical* iff there is no other variable X' such that $X \approx_\Pi X'$ and $\mathrm{ord}(X') < \mathrm{ord}(X)$.
2. For each variable X its Π-*canonical form* $\mathrm{cf}_\Pi(X)$ is defined as the member of the set $\{X' \in \mathcal{V}ar \mid X \approx_\Pi X'\}$ with the least ordinal number.
3. A term t is called Π-*canonical* iff all the variables occurring in t are Π-canonical.
4. For each term t its Π-*canonical form* $\mathrm{cf}_\Pi(t)$ is defined as the result of replacing $\mathrm{cf}_\Pi(X)$ for each variable X occurring in t. □

The following lemma states some obvious properties of terms in canonical form:

Lemma 2.3 (Π-*Canonicity Lemma*)
For each term t, $\mathrm{cf}_\Pi(t)$ is Π-canonical and such that $t \approx_\Pi \mathrm{cf}_\Pi(t)$. Moreover, t and $\mathrm{cf}_\Pi(t)$ have the same positions and structure, except that each variable X occurring at some position $p \in \mathrm{vpos}(t)$ is replaced by an occurrence of $\mathrm{cf}_\Pi(X)$ at the same position p in $\mathrm{cf}_\Pi(t)$.

Proof
Straightforward consequence of the construction of $\mathrm{cf}_\Pi(t)$ from t and the Π-Equivalence Lemma 2.2. □

Given two terms t and s, the term built from t by replacing within t each variable X occurring at some position $p \in \mathrm{vpos}(t) \cap \mathrm{pos}(s)$ by the subterm $s|_p$ is called the *extension of* t *w.r.t. to* s and noted as $t \ll s$ (or equivalently, $s \gg t$). A more precise definition of this notion and some related properties are given below.

Definition 2.9 (*Term extension*)
Given any two terms t and s, the *extension of* t *w.r.t.* s is defined by recursion on the syntactical structure of t:

- $X \ll s = s$ for each variable $X \in \mathcal{V}ar$.
- $u \ll s = u$ for each basic value $u \in B$.

- $c(t_1,\ldots,t_n) \ll s = c(t_1 \ll s_1,\ldots,t_n \ll s_n)$ if $c \in DC^n$ and there is some $c' \in DC^n$ such that $s = c'(s_1,\ldots,s_n)$.
- $c(t_1,\ldots,t_n) \ll s = c(t_1,\ldots,t_n)$ if $c \in DC^n$ and there is no $c' \in DC^n$ such that $s = c'(s_1,\ldots,s_n)$. □

Lemma 2.4 (*Extension Lemma*)
The term extension operation $\ll$ enjoys the two following properties:

1. *Symmetrical Extension Property:*
   Let t', t'' be Π-canonical terms such that $t' \approx_\Pi t''$. Under this assumption $(t' \ll t'') = (t'' \ll t')$.
2. Π-*Equivalence Extension Property:*
   Let the terms t, s be such that for any $p \in \mathrm{pos}(t)$ with $t|_p = X \in \mathcal{V}ar$ one has $p \in \mathrm{pos}(s)$ and $X \approx_\Pi s|_p$. Under this assumption $t \approx_\Pi (t \ll s)$.

Proof of Symmetrical Extension Property
Recall that the hypothesis $t' \approx_\Pi t''$ means that $\Pi \models_{\mathcal{C}} t' == t''$. We reason by complete induction on $\|t'\| + \|t''\|$. There are five possible cases:

1. $t' == t''$ is $c'(\overline{t'}_n) == c''(\overline{t''}_n)$ for some $n \in \mathbb{N}$, $c', c'' \in DC^n$. In this case, the Π-Equivalence Lemma 2.2 ensures that $c' = c'' = c \in DC^n$ and $t'_i \approx_\Pi t''_i$ holds for all $1 \leq i \leq n$. Clearly, the terms t'_i, t''_i are Π-canonical. Therefore, by induction hypothesis we can assume $(t'_i \ll t''_i) = (t''_i \ll t'_i)$ for all $1 \leq i \leq n$. Then, by definition of $\ll$ we get $t' \ll t'' = c(t'_1 \ll t''_1,\ldots,t'_n \ll t''_n) = c(t''_1 \ll t'_1,\ldots,t''_n \ll t'_n) = t'' \ll t'$.
2. $t' == t''$ is $u' == u''$ for some $u', u'' \in B$. In this case, $u' \approx_\Pi u''$ implies that $u' = u'' = u \in B$, and by definition of $\ll$ we get $t' \ll t'' = t'' \ll t' = u \ll u = u$.
3. $t' == t''$ is $X == Y$ for some $X, Y \in \mathcal{V}ar$. In this case, $X \approx_\Pi Y$ and X, Y Π-canonical implies that X, Y must be identical variables. By definition of $\ll$ we get $t' \ll t'' = t'' \ll t' = X \ll X = X$.
4. $t' == t''$ is $X == t''$ with $X \in \mathcal{V}ar$, $t'' \notin \mathcal{V}ar$. In this case, by definition of $\ll$ we get $t' \ll t'' = X \ll t'' = t''$ and $t'' \ll t' = t'' \ll X = t''$.
5. $t' == t''$ is $t' == Y$ with $Y \in \mathcal{V}ar$, $t' \notin \mathcal{V}ar$. In this case, by definition of $\ll$ we get $t' \ll t'' = t' \ll Y = t' = t''$ and $t'' \ll t' = Y \ll t' = t'$. □

Proof of Π-*Equivalence Extension Property*
Recall that the thesis $t \approx_\Pi (t \ll s)$ means that $\Pi \models_{\mathcal{C}} t == (t \ll s)$. We reason by complete induction on $\|t\|$. There are four possible cases:

1. t is a variable $X \in \mathcal{V}ar$. In this case, $X \ll s = s$ by definition of $\ll$, and $X \approx_\Pi s$ holds by hypothesis.
2. t is a basic value $u \in B$. In this case, $u \ll s = u$ by definition of $\ll$, and $u \approx_\Pi u$ holds trivially.
3. t is $c(\overline{t}_n)$ for some $c \in DC^n$ and there is no $c' \in DC^n$ such that s has the form $c'(\overline{s}_n)$. In this case, $c(\overline{t}_n) \ll s = c(\overline{t}_n)$ by definition of $\ll$, and $c(\overline{t}_n) \approx_\Pi c(\overline{t}_n)$ holds trivially.

4. t is $c(\overline{t}_n)$ for some $c \in DC^n$ and s is $c'(\overline{s}_n)$ for some $c' \in DC^n$. In this case $c(\overline{t}_n) \ll c'(\overline{s}_n) = c(t_1 \ll s_1, \ldots, t_n \ll s_n)$ by definition of $\ll$. Moreover, the assumptions of the Π-Equivalent Extension Property hold for the smaller terms t_i, s_i $(1 \leq i \leq n)$. By induction hypothesis we can assume $t_i \approx_\Pi (t_i \ll s_i)$ for all $1 \leq i \leq n$. Therefore, $c(\overline{t}_n) \approx_\Pi c(t_1 \ll s_1, \ldots, t_n \ll s_n)$ due to the Π-Equivalence Lemma 2.2. □

2.2 Qualification Domains

The intended role of *Qualification Domains* in an extended logic programming scheme SQCLP have been already explained in the Introduction. They were originally introduced in (Rodríguez-Artalejo and Romero-Díaz 2008b) and their axiomatic definition was extended with axioms for an additional operation $\oslash$ in (Rodríguez-Artalejo and Romero-Díaz 2009) in order to enable a particular implementation technique for program clauses with threshold conditions in their bodies. The definition given below is again closer to the original one: $\oslash$ is omitted and the axioms of the operator $\circ$ are slightly refined.

Definition 2.10 (*Qualification Domains*)
A *Qualification Domain* is any structure $\mathcal{D} = \langle D, \preccurlyeq, \mathbf{b}, \mathbf{t}, \circ \rangle$ verifying the following requirements:

1. D, noted as $D_\mathcal{D}$ when convenient, is a set of elements called *qualification values*.
2. $\langle D, \preccurlyeq, \mathbf{b}, \mathbf{t} \rangle$ is a lattice with extreme points $\mathbf{b}$ (called *infimum* or *bottom* element) and $\mathbf{t}$ (called *maximum* or *top* element) w.r.t. the partial ordering $\preccurlyeq$, called *qualification ordering*. For given elements $d, e \in D$, we write $d \sqcap e$ for the *greatest lower bound* (*glb*) of d and e, and $d \sqcup e$ for the *least upper bound* (*lub*) of d and e. We also write $d \lhd e$ as abbreviation for $d \preccurlyeq e \wedge d \neq e$.
3. $\circ : D \times D \to D$, called *attenuation operation*, verifies the following axioms:
   (a) $\circ$ is associative, commutative and monotonic w.r.t. $\preccurlyeq$.
   (b) $\forall d \in D : d \circ \mathbf{t} = d$.
   (c) $\forall d \in D : d \circ \mathbf{b} = \mathbf{b}$.
   (d) $\forall d, e \in D : d \circ e \preccurlyeq e$.
   (e) $\forall d, e_1, e_2 \in D : d \circ (e_1 \sqcap e_2) = (d \circ e_1) \sqcap (d \circ e_2)$. □

Actually, some of the properties of $\circ$ postulated as axioms in the previous definition are redundant.[2] More precisely:

Proposition 2.1 (*Redundant postulates of Qualification Domains*)
The properties (3)(c) and (3)(d) are redundant and can be derived from the other axioms in Definition 2.10.

[2] The authors are thankful to G. Gerla for pointing out this fact.

Proof
Note that $\circ$ is commutative and monotonic w.r.t. $\preccurlyeq$ because of axiom (3)(a). Since $\mathbf{t}$ is the top element of the lattice, $d \preccurlyeq \mathbf{t}$ holds for any $d \in D$. By monotonicity of $\circ$, $d \circ e \preccurlyeq \mathbf{t} \circ e$ also holds for any $e \in D$. By commutativity of $\circ$ and axiom (3)(b), $d \circ e \preccurlyeq \mathbf{t} \circ e$ is the same as $d \circ e \preccurlyeq e$. Therefore (3)(d) is a consequence of the other axioms postulated for $\circ$. In particular, taking $e = \mathbf{b}$ we get $d \circ \mathbf{b} \preccurlyeq \mathbf{b}$, which implies $d \circ \mathbf{b} = \mathbf{b}$ because $\mathbf{b}$ is the bottom element of the lattice. Hence, (3)(c) also follows form the other axioms. □

In the rest of the report, $\mathcal{D}$ will generally denote an arbitrary qualification domain. For any finite $S = \{e_1, e_2, \ldots, e_n\} \subseteq D$, the *greatest lower bound* (also called *infimum* of S and noted as $\bigsqcap S$) exists and can be computed as $e_1 \sqcap e_2 \sqcap \cdots \sqcap e_n$ (which reduces to $\top$ in the case $n = 0$). The dual claim concerning *least upper bounds* is also true. As an easy consequence of the axioms, one gets the identity $d \circ \bigsqcap S = \bigsqcap\{d \circ e \mid e \in S\}$.

Many useful qualification domains are such that $\forall d, e \in D \setminus \{\mathbf{b}\} : d \circ e \neq \mathbf{b}$. In the sequel, any qualification domain $\mathcal{D}$ that verifies this property will be called *stable*. Below we present some basic qualification domains which are clearly stable, along with brief explanations of their role for building extended CLP languages as instances of the SQCLP scheme proposed in this report. Checking that these domains satisfy the axioms given in Def. 2.10 is left as an easy exercise. In fact, the axioms have been chosen as a natural generalization of some basic properties satisfied by the ordering $\leq$ and the operation $\times$ over the real interval $[0, 1]$.

2.2.1 The Domain $\mathcal{B}$ of Classical Boolean Values

This domain is $\mathcal{B} =_{\text{def}} \langle\{0, 1\}, \leq, 0, 1, \wedge\rangle$, 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 $\{0, 1\}$.

2.2.2 The Domain $\mathcal{U}$ of Uncertainty Values and its variant $\mathcal{U}'$

This domain is $\mathcal{U} =_{\text{def}} \langle \mathrm{U}, \leq, 0, 1, \times\rangle$, where $\mathrm{U} = [0, 1] = \{d \in \mathbb{R} \mid 0 \leq d \leq 1\}$, $\leq$ is the usual numerical ordering, and $\times$ is the multiplication operation. The top element $\mathbf{t}$ is 1 and the greatest lower bound $\bigsqcap S$ of a finite $S \subseteq \mathrm{U}$ is the minimum value $\min(S)$, which is 1 if $S = \emptyset$. Elements of $\mathcal{U}$ are intended to represent certainty degrees as used in (van Emden 1986).

A slightly different domain $\mathcal{U}'$ can be defined as $\langle \mathrm{U}, \leq, 0, 1, \min\rangle$ where the only difference with respect to $\mathcal{U}$ is that in the case of $\mathcal{U}'$, $\circ = \min$.

2.2.3 The Domain $\mathcal{W}$ of Weight Values and related variants

This domain is $\mathcal{W} =_{\text{def}} \langle \mathrm{P}, \geq, \infty, 0, +\rangle$, where $\mathrm{P} = [0, \infty] = \{d \in \mathbb{R} \cup \{\infty\} \mid d \geq 0\}$, $\geq$ is the reverse of the usual numerical ordering (with $\infty \geq d$ for any $d \in \mathrm{P}$), and $+$ is the addition operation (with $\infty + d = d + \infty = \infty$ for any $d \in \mathrm{P}$). The top

element $\mathbf{t}$ is 0 and the greatest lower bound $\bigsqcap S$ of a finite $S \subseteq \mathrm{P}$ is the maximum value $\max(S)$, which is 0 if $S = \emptyset$. Elements of $\mathcal{W}$ are intended to represent proof costs, measured as the weighted depth of proof trees.

In analogy to the definition of $\mathcal{U}'$ as a variant of $\mathcal{U}$, we can define a qualification domain $\mathcal{W}'$ as $\langle \mathrm{P}, \geq, \infty, 0, \max \rangle$ with $\circ = \max$. Also, as a discrete variant of $\mathcal{W}$, we define the qualification domain $\mathcal{W}_d =_{\mathrm{def}} \langle \mathrm{P}, \geq, \infty, 0, + \rangle$ with the only difference w.r.t. $\mathcal{W}$ being that $\mathrm{P} = \mathbb{N} \cup \{\infty\}$. Elements of $\mathcal{W}_d$ are also intended to represent proof costs (represented by natural numbers in this case). Finally, a variant $\mathcal{W}'_d$ of $\mathcal{W}_d$ can be defined by replacing the attenuation operation in $\mathcal{W}_d$ by max.

2.2.4 Two product constructions

To close this section, we present two product constructions that can be used to build compound qualification domains. The mathematical definition is as follows:

Definition 2.11 (*Products of Qualification Domains*)
Let two qualification domains $\mathcal{D}_i = \langle D_i, \preccurlyeq_i, \mathbf{b}_i, \mathbf{t}_i, \circ_i \rangle$ $(i \in \{1, 2\})$ be given.

1. The *cartesian product* $\mathcal{D}_1 \times \mathcal{D}_2$ is defined as $\mathcal{D} =_{\mathrm{def}} \langle D, \preccurlyeq, \mathbf{b}, \mathbf{t}, \circ \rangle$ where $D =_{\mathrm{def}} D_1 \times D_2$, the partial ordering $\preccurlyeq$ is defined as $(d_1, d_2) \preccurlyeq (e_1, e_2) \Longleftrightarrow_{\mathrm{def}} d_1 \preccurlyeq_1 e_1$ and $d_2 \preccurlyeq_2 e_2$, $\mathbf{b} =_{\mathrm{def}} (\mathbf{b}_1, \mathbf{b}_2)$, $\mathbf{t} =_{\mathrm{def}} (\mathbf{t}_1, \mathbf{t}_2)$ and the attenuation operator $\circ$ is defined as $(d_1, d_2) \circ (e_1, e_2) =_{\mathrm{def}} (d_1 \circ_1 e_1, d_2 \circ_2 e_2)$.
2. Given two elements $d_1 \in D_1$ and $d_2 \in D_2$, the *strict pair* $(\!|\, d_1, d_2 \,|\!)$ is defined by case distinction as follows: if $d_1 \neq \mathbf{b}_1$ and $d_2 \neq \mathbf{b}_2$, then $(\!|\, d_1, d_2 \,|\!) = (d_1, d_2)$; if $d_1 = \mathbf{b}_1$ or $d_2 = \mathbf{b}_2$, then $(\!|\, d_1, d_2 \,|\!) = (\mathbf{b}_1, \mathbf{b}_2)$. In both cases, $(\!|\, d_1, d_2 \,|\!) \in D_1 \times D_2$.
3. The *strict cartesian product* $\mathcal{D}_1 \otimes \mathcal{D}_2$ is defined as $\mathcal{D} =_{\mathrm{def}} \langle D, \preccurlyeq, \mathbf{b}, \mathbf{t}, \circ \rangle$ where $D = D_1 \otimes D_2 =_{\mathrm{def}} \{(\!|\, d_1, d_2 \,|\!) \mid d_1 \in D_1, d_2 \in D_2\}$ (or equivalently, $D = ((D_1 \setminus \{\mathbf{b}_1\}) \times (D_2 \setminus \{\mathbf{b}_2\})) \cup \{(\mathbf{b}_1, \mathbf{b}_2)\}$), the partial ordering $\preccurlyeq$ is defined as $(d_1, d_2) \preccurlyeq (e_1, e_2) \Longleftrightarrow_{\mathrm{def}} d_1 \preccurlyeq_1 e_1$ and $d_2 \preccurlyeq_2 e_2$, $\mathbf{b} =_{\mathrm{def}} (\!|\, \mathbf{b}_1, \mathbf{b}_2 \,|\!) = (\mathbf{b}_1, \mathbf{b}_2)$, $\mathbf{t} =_{\mathrm{def}} (\!|\, \mathbf{t}_1, \mathbf{t}_2 \,|\!)$, and the attenuation operator $\circ$ is defined as $(d_1, d_2) \circ (e_1, e_2) =_{\mathrm{def}} (d_1 \circ_1 e_1, d_2 \circ_2 e_2)$. Note the special case when D_1 or D_2 is a singleton set. Then, D is the singleton set $\{(\mathbf{b}_1, \mathbf{b}_2)\}$, $(\!|\, \mathbf{t}_1, \mathbf{t}_2 \,|\!) = (\mathbf{b}_1, \mathbf{b}_2)$, and $(\mathbf{t}_1, \mathbf{t}_2) \in D$ happens to be false if one of the two sets D_1, D_2 is not a singleton. □

Intuitively, each value (d_1, d_2) belonging to a product domain $\mathcal{D}_1 \times \mathcal{D}_2$ or $\mathcal{D}_1 \otimes \mathcal{D}_2$ imposes the qualification d_1 *and also* the qualification d_2. In particular, values (c, d) belonging to the product domains $\mathcal{U} \times \mathcal{W}$ and $\mathcal{U} \otimes \mathcal{W}$ impose two qualifications, namely: a certainty value greater or equal than c and a proof tree with weighted depth less or equal than d. This intuition indeed corresponds to the declarative semantics formally defined in Section 3.

The next theorem shows that the class of the qualification domains is closed under ordinary cartesian products, while the subclass of stable qualification domains is closed under strict cartesian products. We are particularly interested in stable

qualification domains built from basic domains by reiterated strict products, because they can be encoded into into constraint domains in the sense explained in Subsection 2.2.5 below.

Theorem 2.1
Assume two given qualification domains $\mathcal{D}_1$ and $\mathcal{D}_2$. Then the ordinary cartesian product $\mathcal{D}_1 \times \mathcal{D}_2$ is always a qualification domain. Moreover, if $\mathcal{D}_1$ and $\mathcal{D}_2$ are stable, then the strict cartesian product $\mathcal{D}_1 \otimes \mathcal{D}_2$ is a stable qualification domain.

Proof
Here we reason only for the case of the strict cartesian product since the reasonings needed for the ordinary cartesian product are very similar and even simpler. Assume that $\mathcal{D}_1$ and $\mathcal{D}_2$ are stable qualification domains, and let $\mathcal{D} = \mathcal{D}_1 \otimes \mathcal{D}_2$ be constructed as in Definition 2.11. In order to show that $\mathcal{D}$ is a stable qualification domain, we prove the four items below. The assumption that $\mathcal{D}_1$ and $\mathcal{D}_2$ satisfy all the axioms from Definition 2.10 is used in all the reasonings, often implicitly.

1. The attenuation operator $\circ$ of $\mathcal{D}$ is well defined. Assume $(d_1, d_2), (e_1, e_2) \in D$. According to Definition 2.11, $(d_1, d_2) \circ (e_1, e_2)$ is defined as $(d_1 \circ_1 e_1, d_2 \circ_2 e_2)$. Since $D = D_1 \otimes D_2$ is a strict subset of $D_1 \times D_2$, we must prove that $(d_1 \circ_1 e_1, d_2 \circ_2 e_2) \in D$. We reason by distinction of cases:
   1.1. $(d_1, d_2) = (\mathbf{b}_1, \mathbf{b}_2)$ or $(e_1, e_2) = (\mathbf{b}_1, \mathbf{b}_2)$. In this case, $(d_1 \circ_1 e_1, d_2 \circ_2 e_2) = (\mathbf{b}_1, \mathbf{b}_2) \in D$.
   1.2. $(d_1, d_2) \neq (\mathbf{b}_1, \mathbf{b}_2)$ and $(e_1, e_2) \neq (\mathbf{b}_1, \mathbf{b}_2)$. In this case, $d_1, e_1 \in D_1 \setminus \{\mathbf{b}_1\}$ and $d_2, e_2 \in D_2 \setminus \{\mathbf{b}_2\}$. The assumption that $\mathcal{D}_1$ and $\mathcal{D}_2$ are stable ensures $d_1 \circ_1 e_1 \neq \mathbf{b}_1$ and $d_2 \circ_2 e_2 \neq \mathbf{b}_2$, and therefore $(d_1 \circ_1 e_1, d_2 \circ_2 e_2) \in D$.

2. $\langle D, \preccurlyeq, \mathbf{b}, \mathbf{t} \rangle$ is a lattice with extreme points $\mathbf{b} = (\!|\, \mathbf{b}_1, \mathbf{b}_2 \,|\!) = (\mathbf{b}_1, \mathbf{b}_2)$ and $\mathbf{t} =_{\text{def}} (\!|\, \mathbf{t}_1, \mathbf{t}_2 \,|\!)$ w.r.t. the partial ordering $\preccurlyeq$. By definition, $(d_1, d_2) \preccurlyeq (e_1, e_2) \iff d_1 \preccurlyeq_1 e_1 \wedge d_2 \preccurlyeq_2 e_2$. The fact that $\preccurlyeq$ is a partial ordering with minimum (bottom) element $\mathbf{b}$ is an obvious connsequence. To prove that $\mathbf{t}$ is the maximum (top) element, we reason by case distinction. If D_1 is a singleton set, then $D_1 = \{\mathbf{b}_1\}$, $\mathbf{t}_1 = \mathbf{b}_1$, $D = \{(\mathbf{b}_1, \mathbf{b}_2)\}$, and $(\!|\, \mathbf{t}_1, \mathbf{t}_2 \,|\!) = (\mathbf{b}_1, \mathbf{b}_2)$ is obviously the top element. The case that D_2 is a singleton set is argued similarly. Finally, if neither D_1 nor D_2 are singleton, we have $\mathbf{t}_1 \neq \mathbf{b}_1$, $\mathbf{t}_2 \neq \mathbf{b}_2$, and $\mathbf{t} = (\!|\, \mathbf{t}_1, \mathbf{t}_2 \,|\!) = (\mathbf{t}_1, \mathbf{t}_2)$ is clearly the top element.
   To show that $\langle D, \preccurlyeq, \mathbf{b}, \mathbf{t} \rangle$ is a lattice, we assume two arbitrary elements (d_1, d_2), $(e_1, e_2) \in D$, and we prove:
   2.1. There is a *lub* $(d_1, d_2) \sqcup (e_1, e_2) \in D$. The *lubs* $d_1 \sqcup_1 e_1 \in D_1$ and $d_2 \sqcup_2 e_2 \in D_2$ are known to exist. We claim that $(d_1, d_2) \sqcup (e_1, e_2) = (d_1 \sqcup_1 e_1, d_2 \sqcup_2 e_2)$. Due to the component-wise definition of $\preccurlyeq$, it suffices to show that $(d_1 \sqcup_1 e_1, d_2 \sqcup_2 e_2) \in D$. We prove this by case distinction:
      2.1.1. If $(d_1, d_2) = (\mathbf{b}_1, \mathbf{b}_2)$ then $(d_1 \sqcup_1 e_1, d_2 \sqcup_2 e_2) = (e_1, e_2) \in D$.
      2.1.2. If $(e_1, e_2) = (\mathbf{b}_1, \mathbf{b}_2)$ then $(d_1 \sqcup_1 e_1, d_2 \sqcup_2 e_2) = (d_1, d_2) \in D$.
      2.1.3. If $(d_1, d_2) \neq (\mathbf{b}_1, \mathbf{b}_2)$ and $(e_1, e_2) \neq (\mathbf{b}_1, \mathbf{b}_2)$ then the construction of D ensures that $d_1, e_1 \in D_1 \setminus \{\mathbf{b}_1\}$ and $d_2, e_2 \in D_2 \setminus \{\mathbf{b}_2\}$. This implies $d_1 \sqcup_1 e_1 \neq \mathbf{b}_1$ and $d_2 \sqcup_2 e_2 \neq \mathbf{b}_2$, which guarantees $(d_1 \sqcup_1 e_1, d_2 \sqcup_2 e_2) \in D$.

2.2. There is a *glb* $(d_1, d_2) \sqcap (e_1, e_2) \in D$. The *glbs* $d_1 \sqcap_1 e_1 \in D_1$ and $d_2 \sqcap_2 e_2 \in D_2$ are known to exist. We claim that $(d_1, d_2) \sqcap (e_1, e_2) = (\!| d_1 \sqcap_1 e_1, d_2 \sqcap_2 e_2 |\!)$. We prove the claim by case distinction:

2.2.1. If $d_1 \sqcap_1 e_1 \neq \mathbf{b}_1$ and $d_2 \sqcap_2 e_2 \neq \mathbf{b}_2$, then $(\!| d_1 \sqcap_1 e_1, d_2 \sqcap_2 e_2 |\!)$ is the same as $(d_1 \sqcap_1 e_1, d_2 \sqcap e_2) \in D$, and this pair is the *glb* of (d_1, d_2) and (e_1, e_2) due to the component-wise definition of $\preccurlyeq$.

2.2.2. If $d_1 \sqcap_1 e_1 = \mathbf{b}_1$ or $d_2 \sqcap_2 e_2 = \mathbf{b}_2$, then $(\!| d_1 \sqcap_1 e_1, d_2 \sqcap_2 e_2 |\!) = (\mathbf{b}_1, \mathbf{b}_2)$ is obviously a common lower bound of (d_1, d_2) and (e_1, e_2). In order to conclude that $(\mathbf{b}_1, \mathbf{b}_2)$ is the *glb* of (d_1, d_2) and (e_1, e_2), we show that $(\mathbf{b}_1, \mathbf{b}_2)$ is the only common lower bound of (d_1, d_2) and (e_1, e_2) by the following reasoning: assume an arbitrary $(x, y) \in D$ such that $(x, y) \preccurlyeq (d_1, d_2)$ and $(x, y) \preccurlyeq (e_1, e_2)$. Then $x \preccurlyeq d_1 \sqcap_1 e_1$ and $y \preccurlyeq d_2 \sqcap_2 e_2$. Since $d_1 \sqcap_1 e_1 = \mathbf{b}_1$ or $d_2 \sqcap_2 e_2 = \mathbf{b}_2$, it follows that $x = \mathbf{b}_1$ or $y = \mathbf{b}_2$. By construction of D, it must be the case that $x = \mathbf{b}_1$ *and* $y = \mathbf{b}_2$, because otherwise (x, y) would not belong to D. Therefore $(x, y) = (\mathbf{b}_1, \mathbf{b}_2)$, as desired.

3. $\circ$ satisfies axioms required for attenuation operators in Definition 2.10. By definition of $\circ$ we know

$$(\star)\ (d_1, d_2) \circ (e_1, e_2) = (d_1 \circ_1 e_1, d_2 \circ_2 e_2)$$

which always belongs to D as already proved in item (1) above. All the axioms listed under item (3) of Definition 2.10 except (3)(e) follow easily from the equation $(\star)$ and the corresponding axioms for $\circ_1$ and $\circ_2$. In order to verify axiom (3)(e) for $\circ$, we assume three pairs $(d_1, d_2), (e_1, e_2), (e'_1, e'_2) \in D$. We must prove the equation

$$(\dagger)\ (d_1, d_2) \circ ((e_1, e_2) \sqcap (e'_1, e'_2)) = (d_1, d_2) \circ (e_1, e_2) \sqcap (d_1, d_2) \circ (e'_1, e'_2)\ .$$

We reason by case distinction:

3.1. If $d_1 = \mathbf{b}_1$ and $d_2 = \mathbf{b}_2$ then both sides of $(\dagger)$ are equal to $(\mathbf{b}_1, \mathbf{b}_2)$, as shown by the following calculations:

$$(d_1, d_2) \circ ((e_1, e_2) \sqcap (e'_1, e'_2)) = (\mathbf{b}_1, \mathbf{b}_2) \circ ((e_1, e_2) \sqcap (e'_1, e'_2)) = (\mathbf{b}_1, \mathbf{b}_2)$$

$$(d_1, d_2) \circ (e_1, e_2) \sqcap (d_1, d_2) \circ (e'_1, e'_2) = (\mathbf{b}_1, \mathbf{b}_2) \circ (e_1, e_2) \sqcap (\mathbf{b}_1, \mathbf{b}_2) \circ (e'_1, e'_2) = (\mathbf{b}_1, \mathbf{b}_2) \sqcap (\mathbf{b}_1, \mathbf{b}_2) = (\mathbf{b}_1, \mathbf{b}_2)$$

3.2. If the previous case does not apply, the construction of D ensures that $d_1 \neq \mathbf{b}_1$ and $d_2 \neq \mathbf{b}_2$. We distiguish two subcases:

3.2.1. If $e_1 \sqcap_1 e'_1 = \mathbf{b}_1$ or $e_2 \sqcap_2 e'_2 = \mathbf{b}_2$ we get also $d_1 \circ_1 (e_1 \sqcap_1 e'_1) = \mathbf{b}_1$ or $d_2 \circ_2 (e_2 \sqcap_2 e'_2) = \mathbf{b}_2$, and we can assume the following:

$$(\clubsuit)\quad (\!| e_1 \sqcap_1 e'_1, e_2 \sqcap_2 e'_2 |\!) = (\mathbf{b}_1, \mathbf{b}_2)$$
$$(\spadesuit)\quad (\!| d_1 \circ_1 (e_1 \sqcap_1 e'_1), d_2 \circ_2 (e_2 \sqcap_2 e'_2) |\!) = (\mathbf{b}_1, \mathbf{b}_2)$$

We can now prove that both sides of (†) are equal to $(\mathbf{b}_1, \mathbf{b}_2)$ as follows:

$$(d_1, d_2) \circ ((e_1, e_2) \sqcap (e_1', e_2')) = \\ (d_1, d_2) \circ (\!| e_1 \sqcap_1 e_1', e_2 \sqcap_2 e_2' |\!) =_{\clubsuit} (d_1, d_2) \circ (\mathbf{b}_1, \mathbf{b}_2) = (\mathbf{b}_1, \mathbf{b}_2)$$

$$(d_1, d_2) \circ (e_1, e_2) \sqcap (d_1, d_2) \circ (e_1', e_2') = \\ (d_1 \circ_1 e_1, d_2 \circ_2 e_2) \sqcap (d_1 \circ_1 e_1', d_2 \circ_2 e_2') = \\ (\!| d_1 \circ_1 e_1 \sqcap_1 d_1 \circ_1 e_1', d_2 \circ_2 e_2 \sqcap_2 d_2 \circ_2 e_2' |\!) = \\ (\!| d_1 \circ_1 (e_1 \sqcap_1 e_1'), d_2 \circ_2 (e_2 \sqcap_2 e_2') |\!) =_{\spadesuit} (\mathbf{b}_1, \mathbf{b}_2)$$

3.2.2. If $e_1 \sqcap_1 e_1' \neq \mathbf{b}_1$ and $e_2 \sqcap_2 e_2' \neq \mathbf{b}_2$ then the stability assumption made for $\mathcal{D}_1$ and $\mathcal{D}_2$ ensures $d_1 \circ_1 (e_1 \sqcap_1 e_1') \neq \mathbf{b}_1$ and $d_2 \circ_2 (e_2 \sqcap_2 e_2') \neq \mathbf{b}_2$, and we can assume the following:

$$(\diamondsuit) \quad (\!| e_1 \sqcap_1 e_1', e_2 \sqcap_2 e_2' |\!) = (e_1 \sqcap_1 e_1', e_2 \sqcap_2 e_2')$$
$$(\heartsuit) \quad (\!| d_1 \circ_1 (e_1 \sqcap_1 e_1'), d_2 \circ_2 (e_2 \sqcap_2 e_2') |\!) = (d_1 \circ_1 (e_1 \sqcap_1 e_1'), d_2 \circ_2 (e_2 \sqcap_2 e_2'))$$

Then, (†) is proved by the following calculations:

$$(d_1, d_2) \circ ((e_1, e_2) \sqcap (e_1', e_2')) = \\ (d_1, d_2) \circ (\!| e_1 \sqcap_1 e_1', e_2 \sqcap_2 e_2' |\!) =_{\diamondsuit} (d_1, d_2) \circ (e_1 \sqcap_1 e_1', e_2 \sqcap_2 e_2') = \\ (d_1 \circ_1 (e_1 \sqcap_1 e_1'), d_2 \circ_2 (e_2 \sqcap_2 e_2'))$$

$$(d_1, d_2) \circ (e_1, e_2) \sqcap (d_1, d_2) \circ (e_1', e_2') = \\ (d_1 \circ_1 e_1, d_2 \circ_2 e_2) \sqcap (d_1 \circ_1 e_1', d_2 \circ_2 e_2') = \\ (\!| d_1 \circ_1 e_1 \sqcap_1 d_1 \circ_1 e_1', d_2 \circ_2 e_2 \sqcap_2 d_2 \circ_2 e_2' |\!) = \\ (\!| d_1 \circ_1 (e_1 \sqcap_1 e_1'), d_2 \circ_2 (e_2 \sqcap_2 e_2') |\!) =_{\heartsuit} \\ (d_1 \circ_1 (e_1 \sqcap_1 e_1'), d_2 \circ_2 (e_2 \sqcap_2 e_2'))$$

4. $\mathcal{D}_1 \otimes \mathcal{D}_2$ is stable. To prove this let us assume $(d_1, d_2), (e_1, e_2) \in D_1 \otimes D_2 \setminus \{(\mathbf{b}_1, \mathbf{b}_2)\}$. Then $d_1, e_1 \in D_1 \setminus \{\mathbf{b}_1\}$ and $d_2, e_2 \in D_2 \setminus \{\mathbf{b}_2\}$. Since $\mathcal{D}_1$ and $\mathcal{D}_2$ are stable qualification domains, we can infer that $d_1 \circ_1 e_1 \neq \mathbf{b}_1$ and $d_2 \circ_2 e_2 \neq \mathbf{b}_2$, which implies $(d_1, d_2) \circ (e_1, e_2) = (d_1 \circ_1 e_1, d_2 \circ_2 e_2) \neq (\mathbf{b}_1, \mathbf{b}_2)$. □

2.2.5 Encoding Qualification Domains into Constraint Domains

In this subsection we investigate a technical relationship between qualification domains and constraint domains which will play a key role in the rest of the report.

Definition 2.12 (*Expressing $\mathcal{D}$ in $\mathcal{C}$*)
A qualification domain $\mathcal{D}$ with carrier set $D_{\mathcal{D}}$ is expressible in a constraint domain $\mathcal{C}$ with carrier set $C_{\mathcal{C}}$ if there is an injective mapping $\imath : D_{\mathcal{D}} \setminus \{\mathbf{b}\} \to C_{\mathcal{C}}$ embedding $D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ into $C_{\mathcal{C}}$, and the two following requirements are satisfied:

1. There is a $\mathcal{C}$-constraint $\mathsf{qVal}(X)$ such that $\mathrm{Sol}_{\mathcal{C}}(\mathsf{qVal}(X))$ is the set of all $\eta \in \mathrm{Val}_{\mathcal{C}}$ such that $\eta(X)$ belongs to the range of $\imath$.
2. There is a $\mathcal{C}$-constraint $\mathsf{qBound}(X, Y, Z)$ encoding "$x \trianglelefteq y \circ z$" in the following sense: any $\eta \in \mathrm{Val}_{\mathcal{C}}$ satisfying $\eta(X) = \imath(x)$, $\eta(Y) = \imath(y)$ and $\eta(Z) = \imath(z)$ for some $x, y, z \in D \setminus \{\mathbf{b}\}$ verifies $\eta \in \mathrm{Sol}_{\mathcal{C}}(\mathsf{qBound}(X, Y, Z)) \Longleftrightarrow x \trianglelefteq y \circ z$.

Moreover, if $\mathsf{qVal}(X)$ and $\mathsf{qBound}(X, Y, Z)$ can be chosen as existential constraints, we say that $\mathcal{D}$ is *existentially expressable* in $\mathcal{C}$. □

In the sequel, $\mathcal{C}$-constraints built as instances of $\mathsf{qVal}(X)$ and $\mathsf{qBound}(X, Y, Z)$ are called *qualification constraints*, and Ω is used as notation for sets of qualification constraints. The following result ensures that several qualification domains built with the techniques presented in Subsection 2.2 are existentially expressible in $\mathcal{H}$, $\mathcal{R}$ or $\mathcal{FD}$, according to the case.

Proposition 2.2 (*Expressing Qualification Domains in Constraint Domains*)

1. The domain $\mathcal{B}$ is existentially expressible in any given constraint domain $\mathcal{C}$.
2. The domains $\mathcal{U}$, $\mathcal{U}'$, $\mathcal{W}$ and $\mathcal{W}'$ are existentially expressible in $\mathcal{R}$ (or any other constraint domain that supports the expressivity of $\mathcal{R}$).
3. The domains $\mathcal{W}_d$ and $\mathcal{W}'_d$ are existentially expressible in $\mathcal{FD}$ (or any other constraint domain that supports the expressivity of $\mathcal{FD}$).
4. Assume that the two qualification domains $\mathcal{D}_1$ and $\mathcal{D}_2$ are stable and (existentially) expressible in a constraint domain $\mathcal{C}$. Then, $\mathcal{D}_1 \otimes \mathcal{D}_2$ is also (existentially) expressible in $\mathcal{C}$.

Proof

1. Straightforward, due to the fact that $D_{\mathcal{B}} \setminus \{\mathbf{b}\}$ is the singleton set $\{\mathbf{t}\} = \{true\}$.
2. We prove that $\mathcal{U}$ can be existentially expressed in $\mathcal{R}$ as follows: $D_{\mathcal{U}} \setminus \{\mathbf{b}\} = D_{\mathcal{U}} \setminus \{0\} = (0, 1] \subseteq \mathbb{R} \subseteq C_{\mathcal{R}}$; therefore $\imath$ can be taken as the identity embedding mapping from $(0, 1]$ into $\mathbb{R}$. Moreover, $\mathsf{qVal}(X)$ can be built as the existential $\mathcal{R}$-constraint $cp_<(0, X) \wedge cp_\leq(X, 1)$ and $\mathsf{qBound}(X, Y, Z)$ can be built as the existential $\mathcal{R}$-constraint $\exists X_1(op_\times(Y, Z, X_1) \wedge cp_\leq(X, X_1))$. By very similar reasonings it is easy to check that $\mathcal{U}'$, $\mathcal{W}$ and $\mathcal{W}'$ can also be existentially expressed in $\mathcal{R}$. Note that in the cases of $\mathcal{W}$ and $\mathcal{W}'$ there is no reasonable way to define $\imath(\infty)$. This is the reason why the domain of $\imath$ is required to be $D \setminus \{\mathbf{b}\}$ in Definition 2.12.
3. Note that $D_{\mathcal{W}_d} \setminus \{\mathbf{b}\} = D_{\mathcal{W}_d} \setminus \{\infty\} = \mathbb{N}$. Moreover, $\preccurlyeq$ is $\geq$ and $\circ$ is $+$ in $\mathcal{W}_d$. Therefore, $\mathcal{W}_d$ can be expressed in $\mathcal{FD}$ by taking $\imath$ as the identity embedding mapping, building $\mathsf{qVal}(X)$ as an existential $\mathcal{FD}$ constraint that requires the value of X to be an integer $x \geq 0$, and building $\mathsf{qBound}(X, Y, Z)$ as an existential $\mathcal{FD}$ constraint that requires the values of X, Y and Z to be integers x, y and z such that $x \geq y + z$. A similar reasoning proves that $\mathcal{W}'_d$ is existentially expressible in $\mathcal{FD}$ also.
4. For $j = 1, 2$ assume the existence of injective embedding mappings $\imath_j$ and $\mathcal{C}$-constraints $\mathsf{qVal}_j(X)$, $\mathsf{qBound}_j(X, Y, Z)$ that can be used to (existentially) express $\mathcal{D}_j$ in $\mathcal{C}$. Due to Theorem 2.1 we know that $\mathcal{D}_1 \otimes \mathcal{D}_2$ is a stable qualification domain. Moreover, because of the construction of $\mathcal{D} = \mathcal{D}_1 \otimes \mathcal{D}_2$ given in Definition 2.11, we know that $D \setminus \{\mathbf{b}\} = (D_1 \setminus \{\mathbf{b}_1\}) \times (D_2 \setminus \{\mathbf{b}_2\})$. We also know that $\preccurlyeq$ is defined component-wise from $\preccurlyeq_1$, and $\preccurlyeq_2$, and analogously for $\circ$. Therefore, $\mathcal{D}$ can be (existentially) expressed in $\mathcal{C}$ by taking:

   - $\imath$ defined by $\imath(d_1, d_2) =_{\text{def}} (\imath_1(d_1), \imath_2(d_2))$.
   - $\mathsf{qVal}(X)$ built as the prenex form of the constraint

     $$\exists X_1 \exists X_2 (X == (X_1, X_2) \wedge \mathsf{qVal}_1(X_1) \wedge \mathsf{qVal}_2(X_2))$$

     which is existential if $\mathsf{qVal}_1(X_1)$ and $\mathsf{qVal}_2(X_2)$ are both existential.

- $\mathsf{qBound}(X, Y, Z)$ built as the prenex form of the constraint

$$\exists X_1 \exists X_2 \exists Y_1 \exists Y_2 \exists Z_1 \exists Z_2 (X == (X_1, X_2) \wedge Y == (Y_1, Y_2) \wedge Z == (Z_1, Z_2) \wedge \mathsf{qBound}_1(X_1, Y_1, Z_1) \wedge \mathsf{qBound}_2(X_2, Y_2, Z_2))$$

  which is existential if $\mathsf{qBound}_1(X_1, Y_1, Z_1)$ and $\mathsf{qBound}_2(X_2, Y_2, Z_2)$ are both existential.

Note that this reasoning does not work for the non-strict cartesian product $\mathcal{D} = \mathcal{D}_1 \times \mathcal{D}_2$, because in this case $D \setminus \{\mathbf{b}\} = (D_1 \times D_2) \setminus \{(\mathbf{b}_1, \mathbf{b}_2)\}$ includes some pairs (d_1, d_2) such that either $d_1 = \mathbf{b}_1$ or $d_2 = \mathbf{b}_2$ (but not both), and the given mappings $\imath_1$, $\imath_2$ cannot be used to embed such pairs into $C_{\mathcal{C}}$. □

2.3 Similarity and Proximity Relations

Similarity relations over a given set S have been defined in (Zadeh 1971; Sessa 2002) and related literature as mappings $\mathcal{S} : S \times S \to [0, 1]$ that satisfy reflexivity, symmetry and transitivity axioms analogous to those required for classical equivalence relations. A more general notion called *proximity relation* has been defined in (Dubois and Prade 1980) by omitting the transitivity axiom. Each value $\mathcal{S}(x, y)$ computed by a similarity (resp. proximity) relation $\mathcal{S}$ is called the *similarity degree* (resp. *proximity degree*) between x and y. In our previous paper (Caballero et al. 2008), we proposed to generalize similarity relations by allowing elements of an arbitrary qualification domain $\mathcal{D}$ to serve as proximity degrees. The definition below further generalizes this approach by considering proximity relations.

Definition 2.13 (*Proximity and similarity relations*)
Let a qualification domain $\mathcal{D}$ with carrier set D and a set S be given.

1. A *$\mathcal{D}$-valued relation* over S is any mapping $\mathcal{S} : S \times S \to D$.
2. A $\mathcal{D}$-valued relation $\mathcal{S}$ over S is called
   (a) *Reflexive* iff $\forall x \in S : \mathcal{S}(x, x) = \mathbf{t}$.
   (b) *Symmetrical* iff $\forall x, y \in S : \mathcal{S}(x, y) = \mathcal{S}(y, x)$.
   (c) *Transitive* iff $\forall x, y, z \in S : \mathcal{S}(x, z) \trianglerighteq \mathcal{S}(x, y) \sqcap \mathcal{S}(y, z)$.
3. $\mathcal{S}$ is called a *$\mathcal{D}$-valued proximity relation* iff $\mathcal{S}$ is reflexive and symmetrical.
4. If $\mathcal{S}$ is also transitive, then it is called a *$\mathcal{D}$-valued similarity relation.*
5. $\mathcal{S}$ is called *finitary* iff there are only finitely many choices of elements $x, y \in S$ such that $x \neq y$ and $\mathcal{S}(x, y) \neq \mathbf{b}$. From a practical viewpoint, this is a very natural requirement. □

Obviously, $\mathcal{D}$-valued similarity relations are a particular case of $\mathcal{D}$-valued proximity relations. Moreover, when $\mathcal{D}$ is chosen as the qualification domain $\mathcal{U}$, the previous definition provides proximity and similarity relations in the sense of (Zadeh 1971; Dubois and Prade 1980). In this case, a proximity degree $\mathcal{S}(x, y) = d \in [0, 1]$ can be naturally interpreted as a *certainty degree* for the assertion that x and y are interchangeable. On the other hand, if $\mathcal{S}$ is $\mathcal{W}$-valued, then $\mathcal{S}(x, y) = d \in [0, \infty]$ can be interpreted as a *cost* to be paid for y to play the role of x. More generally, the

proximity degrees computed by a $\mathcal{D}$-valued proximity relation may have different interpretations according to the intended role of $\mathcal{D}$-elements as qualification values.

In contrast to previous works such as (Sessa 2002; Caballero et al. 2008), in the rest of this report we will work with $\mathcal{D}$-valued proximity rather than similarity relations. Formally, this leads to more general results. Moreover, as already noted by (Shenoi and Melton 1999) and other authors, the transitivity property required for similarity relations may be counterintuitive in some cases. For instance, assume nullary constructors `colt`, `cold` and `gold` intended to represent words composed of four letters. Then, measuring the proximity between such words might reasonably lead to a $\mathcal{U}$-valued proximity relation $\mathcal{S}$ such that $\mathcal{S}(\texttt{colt}, \texttt{cold}) = 0.9$, $\mathcal{S}(\texttt{cold}, \texttt{gold}) = 0.9$ and $\mathcal{S}(\texttt{colt}, \texttt{gold}) = 0.4$. On the other hand, insisting on $\mathcal{S}$ to be transitive would enforce the unreasonable condition $\mathcal{S}(\texttt{colt}, \texttt{gold}) \geq 0.9$. Therefore, a similarity relation would be not appropriate in this case.

The special mapping $\mathcal{S}_{\mathrm{id}} : S \times S \to D$ defined as $\mathcal{S}_{\mathrm{id}}(x, x) = \mathbf{t}$ for all $x \in S$ and $\mathcal{S}_{\mathrm{id}}(x, y) = \mathbf{b}$ for all $x, y \in S$, $x \neq y$ is trivially a $\mathcal{D}$-valued similarity (and therefore, also proximity) relation called the *identity*.

2.3.1 Admissible triples and proximity relations

From now on, we will focus on proximity relations that are related to constraint domains in the following sense:

Definition 2.14 (*Admissible triples*)
$\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ is called an *admissible triple* iff the following requirements are fulfilled:

1. $\mathcal{C}$ is a constraint domain with signature $\Sigma = \langle DC, DP, PP \rangle$ and set of basic values $B_{\mathcal{C}}$, and $\mathcal{D}$ is a qualification domain expressible in $\mathcal{C}$ in the sense of Definition 2.12.
2. $\mathcal{S}$ is a $\mathcal{D}$-valued proximity relation over $S = \mathcal{V}ar \uplus B_{\mathcal{C}} \uplus DC \uplus DP \uplus PP$.
3. $\mathcal{S}$ restricted to $\mathcal{V}ar$ behaves as the identity, i.e. $\mathcal{S}(X, X) = \mathbf{t}$ for all $X \in \mathcal{V}ar$ and $\mathcal{S}(X, Y) = \mathbf{b}$ for all $X, Y \in \mathcal{V}ar$ such that $X \neq Y$.
4. For any $x, y \in S$, $\mathcal{S}(x, y) \neq \mathbf{b}$ can happen only if some of the following cases holds:
   (a) $x = y$ are identical.
   (b) $x, y \in B_{\mathcal{C}}$ are basic values.
   (c) $x, y \in DC$ are data constructor symbols with the same arity.
   (d) $x, y \in DP$ are defined predicate symbols with the same arity.

   In particular, $\mathcal{S}(p, p') \neq \mathbf{b}$ cannot happen if $p, p' \in PP$ are syntactically different primitive predicate symbols. □

In the rest of the report, our notions and results are valid for any choice of an admissible triple $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$. Proposition 2.2 provides useful information for building admissible triples. For any given admissible triple, $\mathcal{S}$ can be naturally extended to act over terms and atoms over $\mathcal{C}$. The extension, also noted $\mathcal{S}$, works as specified in the recursive definition below. An analogous definition for the case of $\mathcal{U}$-valued similarity relations can be found in (Sessa 2002).

Definition 2.15 (*$\mathcal{S}$ acting over terms and atoms*)
For any given admissible triple, $\mathcal{S}$ is extended to work over $\mathcal{C}$-terms and $\mathcal{C}$-atoms as follows:

1. For any $t \in \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$:
   $\mathcal{S}(t,t) = \mathbf{t}$.
2. For $X \in \mathcal{V}ar$ and for any term t different from X:
   $\mathcal{S}(X,t) = \mathcal{S}(t,X) = \mathbf{b}$.
3. For $c, c' \in DC$ with different arities n, m:
   $\mathcal{S}(c(\overline{t}_n), c'(\overline{t'}_m)) = \mathbf{b}$.
4. For $c, c' \in DC$ with the same arity n:
   $\mathcal{S}(c(\overline{t}_n), c'(\overline{t'}_n)) = \mathcal{S}(c,c') \sqcap \mathcal{S}(t_1, t'_1) \sqcap \ldots \sqcap \mathcal{S}(t_n, t'_n)$.
5. For $r, r' \in DP \cup PP$ with different arities n, m:
   $\mathcal{S}(r(\overline{t}_n), r'(\overline{t'}_m)) = \mathbf{b}$.
6. For $r, r' \in DP \cup PP$ with the same arity n:
   $\mathcal{S}(r(\overline{t}_n), r'(\overline{t'}_n) = \mathcal{S}(r,r') \sqcap \mathcal{S}(t_1, t'_1) \sqcap \ldots \sqcap \mathcal{S}(t_n, t'_n)$. $\square$

Given two terms t, t' and some fixed qualification value $\lambda \in D \setminus \{\mathbf{b}\}$ we will use the notation $t \approx_\lambda t'$ (read as t and t' are *$\mathcal{S}$-close at level* λ) as an abbreviation of $\lambda \trianglelefteq \mathcal{S}(t,t')$. For the sake of simplicity, $\mathcal{S}$ is not made explicit in the $\approx_\lambda$ notation. The following lemma provides a natural characterization of $\approx_\lambda$. A similar result was given in (Sessa 2002) for the case of case of $\mathcal{U}$-valued similarity relations.

Lemma 2.5 (*Proximity Lemma*)

1. $\approx_\lambda$ is a reflexive and symmetric equivalence relation over terms, which is also transitive (and hence an equivalence relation) in the case that $\mathcal{S}$ is a similarity relation.
2. For any given terms t and t' the following two statements are equivalent:
   (a) $t \approx_\lambda t'$.
   (b) $\mathrm{pos}(t) = \mathrm{pos}(t')$, and for each $p \in \mathrm{pos}(t) \cap \mathrm{pos}(t')$ some of the cases below holds:
      i $t \circ p = t' \circ p = X$ for some $X \in \mathcal{V}ar$.
      ii $t \circ p = s \circ p = u$ for some $u \in B_{\mathcal{C}}$.
      iii $t \circ p = c$ and $t' \circ p = c'$ for some $n \in \mathbb{N}$ and some $c, c' \in DC^n$ such that $\lambda \trianglelefteq \mathcal{S}(c,c')$.
3. Any given terms t and t' such that $t \approx_\lambda t'$ are *quasi-identical* in the following sense: $\mathrm{pos}(t) = \mathrm{pos}(t')$, and for each $p \in \mathrm{pos}(t) = \mathrm{pos}(t')$ either $t \circ p = t' \circ p$ or else $t \circ p$ and $t' \circ p$ are two data constructors of the same arity.
4. $\approx_\lambda$ boils down to the syntactic equality relation '=' when $\mathcal{S}$ is the identity proximity relation $\mathcal{S}_{\mathrm{id}}$.

Proof
We give a separate reasoning for each item.

1. Note that the reflexivity and symmetry of $\approx_\lambda$ are a trivial consequence of the reflexivity and symmetry of $\mathcal{S}$, as formulated in Definition 2.13. In the case that $\mathcal{S}$ is a similarity relation, transitivity of $\approx_\lambda$ follows from transitivity of $\mathcal{S}$ and the obvious fact that $\lambda \sqcap \lambda = \lambda$.
2. The claimed equivalence between conditions 2(a) and 2(b) can be proved reasoning by induction on $\|t\| + \|t'\|$.
3. This item is an obvious consequence of the previous one.
4. Assume $\mathcal{S} = \mathcal{S}_{\mathrm{id}}$. Then, as a trivial consequence of Definition 2.15, the value of $\mathcal{S}(t, t')$ is $\mathbf{t}$ if $t = t'$ and $\mathbf{b}$ otherwise. Since $\lambda \neq \mathbf{b}$, it follows that $t \approx_\lambda t'$ iff $t = t'$, as desired. □

The following result shows that $\approx_\lambda$ is compatible with the term extension operation in a natural way:

Lemma 2.6 (*Proximity Preservation Lemma*)
Assume terms t, t' and $\lambda \in D \setminus \{\mathbf{b}\}$ such that $t \approx_\lambda t'$. Then $(t \ll s) \approx_\lambda (t' \ll s)$ holds also for any term s.

Proof
Due to the assumption, $t \approx_\lambda t'$ are quasi-identical and satisfy condition 2(b) as stated in the Proximity Lemma 2.5. From this fact and Definition 2.9 it is quite clear that the same condition 2(b) holds also for $t \ll s$, $t' \ll s$ and λ. Therefore, the Proximity Lemma allows to conclude $t \approx_\lambda t'$ as desired. □

2.3.2 Term proximity w.r.t. a given constraint set

Reasoning with equations between $\mathcal{C}$-terms will require to infer information both from $\mathcal{S}$ and for some fixed constraint set $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$. This leads to a generalization of $\approx_\lambda$ formally defined as follows:

Definition 2.16 (*Term proximity w.r.t. a given constraint set*)
Let $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ be any admissible triple. Assume $\lambda \in D \setminus \{\mathbf{b}\}$ and $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$. We will say that t and s are *$\mathcal{S}$-close at level λ w.r.t. Π* (in symbols, $t \approx_{\lambda,\Pi} s$) iff there are two terms $\hat{t}$, $\hat{s}$ such that $t \approx_\Pi \hat{t}$, $s \approx_\Pi \hat{s}$ and $\hat{t} \approx_\lambda \hat{s}$. For the sake of simplicity neither $\mathcal{S}$ nor $\mathcal{C}$ are made explicit in the notation. □

As illustration, let us present an example using the constraint domain $\mathcal{R}$ and the qualification domain $\mathcal{U}$:

Example 2.2 (*Term proximity w.r.t. $\mathcal{R}$ constraints*)
Consider $\Pi = \{op_+(A, A, X), op_\times(2.0, A, Y), Z == c(X, Y)\} \subseteq \mathrm{Con}_{\mathcal{R}}$. Note that this choice of Π ensures $X \approx_\Pi Y$. Assume $c, c', c'' \in DC^2$ and an $\mathcal{U}$-valued proximity relation $\mathcal{S}$ such that $\mathcal{S}(c', c) = \mathcal{S}(c, c'') = 0.8$ and $\mathcal{S}(c', c'') = 0.6$. Then:

1. $c(Y, X) \approx_\Pi Z$ holds, but $c'(Y, X) \approx_\Pi Z$ is false.
2. $c'(Y, X) \approx_{0.7,\Pi} Z$ holds, because $c'(Y, X) \approx_\Pi c'(X, X)$, $Z \approx_\Pi c(X, X)$ and $c'(X, X) \approx_{0.7} c(X, X)$.
3. $Z \approx_{0.7,\Pi} c''(X, Y)$ is also true, for similar reasons.

4. $c'(Y,X) \approx_{0.7,\Pi} c''(X,Y)$ is false, because there is no possible choice of terms $\hat{t}$ and $\hat{s}$ such that $c'(Y,X) \approx_\Pi \hat{t}$, $c''(X,Y) \approx_\Pi \hat{s}$ and $\hat{t} \approx_{0.7} \hat{s}$. □

The next result states some basic properties of relations $\approx_{\lambda,\Pi}$.

Lemma 2.7 (Π-*Proximity Lemma*)

1. $\approx_{\lambda,\Pi}$ is invariant w.r.t. $\approx_\Pi$ in the following sense: $t \approx_{\lambda,\Pi} s$ implies $t' \approx_{\lambda,\Pi} s'$ for all terms t', s' such that $t' \approx_\Pi t$ and $s' \approx_\Pi s$.
2. $\approx_{\lambda,\Pi}$ is a reflexive and symmetric relation over terms, which is also transitive (and hence an equivalence relation) in the case that $\mathcal{S}$ is a similarity relation.
3. For any given terms t and t' the following two statements are equivalent:
   (a) $t \approx_{\lambda,\Pi} t'$.
   (b) For any common position $p \in \mathrm{pos}(t) \cap \mathrm{pos}(t')$ some of the cases below holds:
      i $t \circ p$ or $t' \circ p$ is a variable, and moreover $t|_p \approx_{\lambda,\Pi} t'|_p$.
      ii $t \circ p = s \circ p = u$ for some $u \in B_\mathcal{C}$.
      iii $t \circ p = c$ and $t' \circ p = c'$ for some $n \in \mathbb{N}$ and some $c, c' \in DC^n$ such that $\lambda \trianglelefteq \mathcal{S}(c,c')$.
4. $\approx_{\lambda,\Pi}$ boils down to $\approx_\lambda$ when Π is the empty set, and $\approx_{\lambda,\Pi}$ boils down to $\approx_\Pi$ when $\mathcal{S}$ is the identity proximity relation $\mathcal{S}_{\mathrm{id}}$.

Proof
We give a separate reasoning for each item. Definition 2.16 and Lemmata 2.2 and 2.5 are implicitely used at some points.

1. By definition, $t \approx_{\lambda,\Pi} s$ means the existence of terms $\hat{t}, \hat{s}$ such that $t \approx_\Pi \hat{t}$, $s \approx_\Pi \hat{s}$ and $\hat{t} \approx_\lambda \hat{s}$. In case that $t' \approx_\Pi t$ and $s' \approx_\Pi s$, the same terms $\hat{t}, \hat{s}$ verify $t' \approx_\Pi \hat{t}$, $s' \approx_\Pi \hat{s}$ (since $\approx_\Pi$ is an equivalence relation) and $\hat{t} \approx_\lambda \hat{s}$. Therefore $t' \approx_{\lambda,\Pi} s'$.
2. Let us consider the three properties in turn:
   Reflexivity: $t \approx_{\lambda,\Pi} t$ holds because $\hat{t} = t$ trivially verifies $t \approx_\Pi \hat{t}$ and $\hat{t} \approx_\lambda \hat{t}$.
   Symmetry: Assume $t \approx_{\lambda,\Pi} s$. Then there are terms $\hat{t}, \hat{s}$ such that $t \approx_\Pi \hat{t}$, $s \approx_\Pi \hat{s}$ and $\hat{t} \approx_\lambda \hat{s}$. Due to the symmetry of $\approx_\lambda$ we get $\hat{s} \approx_\lambda \hat{t}$ and hence $s \approx_{\lambda,\Pi} t$.
   Transitivity: Example 2.2 above shows that $\approx_{\lambda,\Pi}$ is not transitive in general. Here we prove transitivity of $\approx_{\lambda,\Pi}$ under the assumption that $\mathcal{S}$ is a similarity relation fulfilling the transitive property stated in Definition 2.13.
   Assume terms t_1, t_2 and t_3 such that $t_1 \approx_{\lambda,\Pi} t_2$ and $t_2 \approx_{\lambda,\Pi} t_3$. Then there are terms t_1', t_2', t_2'' and t_3'' such that

$$(a)\ t_1 \approx_\Pi t_1',\ t_2 \approx_\Pi t_2',\ t_1' \approx_\lambda t_2' \quad \text{and} \quad (b)\ t_2 \approx_\Pi t_2'',\ t_3 \approx_\Pi t_3'',\ t_2'' \approx_\lambda t_3'' \ .$$

   Without loss of generality, t_1', t_2', t_2'' and t_3'' can be assumed to be Π-canonical terms. If they were not, it would suffice to to replace each of them by its Π-canonical form, built as explained in Definition 2.8. This replacement would preserve properties (a) and (b) thanks to the Canonicity Lemma 2.3.
   We claim that there are three terms $\hat{t_1}$, $\hat{t_2}$, and $\hat{t_3}$ such that

$$(c)\ t_1 \approx_\Pi \hat{t_1},\ t_2 \approx_\Pi \hat{t_2},\ t_3 \approx_\Pi \hat{t_3} \quad \text{and} \quad (d)\ \hat{t_1} \approx_\lambda \hat{t_2},\ \hat{t_2} \approx_\lambda \hat{t_3} \ .$$

Conditions (c) and (d) imply $t_1 \approx_{\lambda,\Pi} t_3$ due to Definition 2.16 and the transivity property of $\approx_\lambda$, which is ensured by the transitivity of $\mathcal{S}$ and the Proximity Lemma 2.5. In the rest of this item we prove the claim by assuming (a) and (b) and showing how to build $\hat{t_1}$, $\hat{t_2}$, and $\hat{t_3}$ fulfilling (c) and (d).
Note that the assumption $t'_1 \approx_\lambda t'_2$ implies that t'_1 and t'_2 are quasi-identical terms, due Proximity Lemma 2.5(3). Analogously, terms t''_2 and t''_3 must be also quasi-identical due to the assumption $t''_2 \approx_\lambda t''_3$, and the target condition (d) requires that $\hat{t_1}$, $\hat{t_2}$, $\hat{t_3}$ are constructed as quasi-identical terms. Since our assumptions do not guarantee quasi-identity of terms t'_2 and t''_2, we resort to the term extension construction from Definition 2.9 for building the terms $\hat{t_i}$. More precisely, we build:

$$\hat{t_1} =_{\text{def}} (t'_1 \ll t''_2); \quad \hat{t_2} =_{\text{def}} (t'_2 \ll t''_2) = (t''_2 \ll t'_2); \quad \text{and } \hat{t_3} =_{\text{def}} (t''_3 \ll t'_2)$$

where the identity $(t'_2 \ll t''_2) = (t''_2 \ll t'_2)$ is a consequence of the Symmetrical Extension Property from Lemma 2.4, which can be applied because t'_2 and t''_2 are Π-canonical and assumptions (a), (b) imply $t'_2 \sim_\Pi t''_2$. We argue that conditions (c) and (d) are satisfied as follows:
— Condition (c), $t_1 \approx_\Pi \hat{t_1}$: By assumptions (a), (b) we know $t_1 \approx_\Pi t'_1$ and $t'_2 \approx_\Pi t''_2$. It suffices to prove $t'_1 \approx_\Pi \hat{t_1}$. For each $p \in \text{pos}(t'_1)$ with $t'_1|_p = X \in \mathcal{V}ar$ we have $t'_2|_p = X$ because t'_1 and t'_2 are quasi-identical. Moreover, $t'_2 \approx_\Pi t''_2$ implies that $p \in \text{pos}(t''_2)$ and $X \sim_\Pi t''_2|_p$, due to the Π-Equivalence Lemma 2.2. In these conditions, $t'_1 \approx_\Pi \hat{t_1}$ follows from $\hat{t_1} = (t'_1 \ll t''_2)$ and the Equivalent Extension Property from Lemma 2.4.
— Condition (c), $t_3 \approx_\Pi \hat{t_3}$: The proof for this is analogous to the previous one. Since $t_3 \approx_\Pi t''_3$ and $\hat{t_3} = (t''_3 \ll t'_2)$ it suffices to prove $t''_3 \approx_\Pi (t''_3 \ll t'_2)$, which can be done with the help of the Equivalent Extension Property.
— Condition (c), $t_2 \approx_\Pi \hat{t_2}$: By assumptions (a), (b) we know $t_2 \approx_\Pi t'_2$ and $t'_2 \approx_\Pi t''_2$. It suffices to prove $t'_2 \approx_\Pi \hat{t_2}$. For each $p \in \text{pos}(t'_2)$ with $t'_2|_p = X \in \mathcal{V}ar$ we have $p \in \text{pos}(t''_2)$ and $X \sim_\Pi t''_2|_p$, due to $t'_2 \approx_\Pi t''_2$ and the Π-Equivalence Lemma 2.2. In these conditions, $t'_2 \approx_\Pi \hat{t_2}$ follows from $\hat{t_2} = (t'_2 \ll t''_2)$ and the Equivalent Extension Property.
— Condition (d), $\hat{t_1} \approx_\lambda \hat{t_2}$: By assumption (a) we have $t'_1 \approx_\lambda t'_2$. By the Proximity Preservation Lemma 2.6 this implies $(t'_1 \ll t''_2) \approx_\lambda (t'_2 \ll t''_2)$. By construction of the terms $\hat{t_i}$, this is the same as $\hat{t_1} \approx_\lambda \hat{t_2}$.
— Condition (d), $\hat{t_2} \approx_\lambda \hat{t_3}$: The proof for this is analogous to the previous one. Assumption (b) provides $t''_2 \approx_\lambda t''_3$. Then, the Proximity Preservation Lemma guarantees $(t''_2 \ll t'_2) \approx_\lambda (t''_3 \ll t'_2)$, which is the same as $\hat{t_2} \approx_\lambda \hat{t_3}$ by construction of the terms $\hat{t_i}$ (this time viewing $\hat{t_2}$ as $(t''_2 \ll t'_2)$ rather than $(t'_2 \ll t''_2)$ as in the previous argumentation).

3. The claimed equivalence between conditions 3(a) and 3(b) can proved reasoning by induction on $\|t\| + \|t'\|$.
4. According to Definition 2.16, $t \approx_{\lambda,\Pi} s$ is true iff $(\star)$ holds, where:

$$(\star) \text{ there are terms } \hat{t}, \hat{s} \text{ such that } t \approx_\Pi \hat{t},\ s \approx_\Pi \hat{s} \text{ and } \hat{t} \approx_\lambda \hat{s}.$$

Let us argue for the two cases $\Pi = \emptyset$ and $\mathcal{S} = \mathcal{S}_{\text{id}}$ separately:

- Assume that $\Pi = \emptyset$. Then, due to Π-Equivalence Lemma 2.2(3), $(\star)$ can be

rewritten as

$$\text{there are terms } \hat{t}, \hat{s} \text{ such that } t = \hat{t},\ s = \hat{s} \text{ and } \hat{t} \approx_\lambda \hat{s}$$

which is equivalent to $t \approx_\lambda s$.

- Assume now that $\mathcal{S} = \mathcal{S}_{\mathrm{id}}$. Then, due to Proximity Lemma 2.5(4), $(\star)$ can be rewritten as

$$\text{there are terms } \hat{t}, \hat{s} \text{ such that } t \approx_\Pi \hat{t},\ s \approx_\Pi \hat{s} \text{ and } \hat{t} = \hat{s}$$

which is equivalent to $t \approx_\Pi s$. □

The following technical lemma will be needed later on. Although it is closely related to Lemma 2.1(2), it needs a separate proof because statements of the form $t \approx_{\lambda,\Pi} s$ are not $\mathcal{C}$-constraints.

Lemma 2.8 (*Substitution Lemma for* $\approx_{\lambda,\Pi}$)
Let $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ be any admissible triple. Assume $\lambda \in D \setminus \{\mathbf{b}\}$, $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$, and two terms t, s such that $t \approx_{\lambda,\Pi} s$. Then $t\sigma \approx_{\lambda,\Pi\sigma} s\sigma$ holds for every $\mathcal{C}$-substitution σ.

Proof
Because of the assumptions and Definition 2.16, there are terms $\hat{t}, \hat{s}$ such that $t \approx_\Pi \hat{t}$ (i.e. $\Pi \models_{\mathcal{C}} t == \hat{t}$), $s \approx_\Pi \hat{s}$ (i.e. $\Pi \models_{\mathcal{C}} s == \hat{s}$) and $\hat{t} \approx_\lambda \hat{s}$. Consider now any substitution σ. Due to Lemma 2.1(2), we get $\Pi\sigma \models_{\mathcal{C}} t\sigma == \hat{t\sigma}$ (i.e. $t\sigma \approx_{\Pi\sigma} \hat{t\sigma}$) and $\Pi\sigma \models_{\mathcal{C}} s\sigma == \hat{s\sigma}$ (i.e. $s\sigma \approx_{\Pi\sigma} \hat{s\sigma}$). Moreover, $\hat{t\sigma} \approx_\lambda \hat{s\sigma}$ is an easy consequence of $\hat{t} \approx_\lambda \hat{s}$ and Proximity Lemma 2.5(2). Then, Definition 2.16 allows to conclude $t\sigma \approx_{\lambda,\Pi\sigma} s\sigma$ simply by taking $\hat{t\sigma}$ as $\hat{t\sigma}$ and $\hat{s\sigma}$ as $\hat{s\sigma}$. □

3 The SQCLP Programming Scheme

In this section we develop the SQCLP scheme with instances SQCLP$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ announced in the introduction. The parameters $\mathcal{S}$, $\mathcal{D}$ and $\mathcal{C}$ stand for an admissible proximity relation, a qualification domain and a constraint domain with a certain signature Σ, respectively. By convention, we consider only those instances of the scheme whose parameters are chosen to constitute an *admissible triple* in the sense of Definition 2.14. We focus on declarative semantics, using an interpretation transformer and a logical inference system to provide alternative characterizations of least program models. We also discuss declarative semantics of goals and related approaches.

A brief remark regarding notation is in place here. For the sake of notational consistency with previous works (either by us or other authors) where similarity rather than proximity relations were used, we keep the symbol $\mathcal{S}$ for proximity relations and the uppercase letter `S` in the names of programming schemes. Our results, however, do not rely on the transitivity property from Definition 2.13.

3.1 Programs and their Declarative Semantics

A SQCLP$(\mathcal{S}, \mathcal{D}, \mathcal{C})$-program is a set $\mathcal{P}$ of *qualified program rules* (also called *qualified clauses*) of the form $C : A \overset{\alpha}{\leftarrow} B_1 \sharp w_1, \ldots, B_m \sharp w_m$, where A is a defined atom, $\alpha \in$

$D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ is called the *attenuation factor* of the clause and each $B_j \sharp w_j$ $(1 \leq j \leq m)$ is an atom B_j annotated with a so-called *threshold value* $w_j \in (D_{\mathcal{D}} \setminus \{\mathbf{b}\}) \uplus \{?\}$. The intended meaning of C is as follows: if for all $1 \leq j \leq m$ one has $B_j \sharp e_j$ (meaning that B_j holds with qualification value e_j) for some $e_j \trianglerighteq^? w_j$, then $A \sharp d$ (meaning that A holds with qualification value d) can be inferred for any $d \in D \setminus \{\mathbf{b}\}$ such that $d \trianglelefteq \alpha \circ \bigsqcap_{j=1}^{m} e_j$. By convention, $e_j \trianglerighteq^? w_j$ means $e_j \trianglerighteq w_j$ if $w_j \neq ?$ and is identically true otherwise. In practice threshold values equal to '?' and attenuation values equal to $\mathbf{t}$ can be omitted.

As motivating example, consider a SQCLP($\mathcal{S}$,$\mathcal{U}\otimes\mathcal{W}$,$\mathcal{R}$)-program $\mathcal{P}$ including the clauses and equations for $\mathcal{S}$ displayed in Figure 1. From Subsection 2.2 recall that qualification values in $\mathcal{U}\otimes\mathcal{W}$ are pairs (d, e) (where d represents a certainty degree and e represents a proof cost), as well as the behavior of $\trianglelefteq$ and $\circ$ in $\mathcal{U}\otimes\mathcal{W}$. Consider the problem of proving `goodWork(king_liar)`$\sharp(d, e)$ from $\mathcal{P}$. This can be achieved for $d = 0.75 \times \min\{d_1, d_2\}$, $e = 3 + \max\{e_1, e_2\}$ by using R_1 instantiated by $\{\texttt{X} \mapsto \texttt{king\_liar}, \texttt{Y} \mapsto \texttt{shakespeare}\}$, and going on to prove `famousAuthor(shakespeare)`$\sharp(d_1, e_1)$ for some $d_1 \geq 0.5$, $e_1 \leq 100$ and `wrote(shakespeare,king_liar)`$\sharp(d_2, e_2)$ for some d_2, e_2. Thanks to R_2, R_3 and $\mathcal{S}$, these proofs succeed with $(d_1, e_1) = (0.9, 1)$ and $(d_2, e_2) = (0.8, 2)$. Therefore, the desired proof succeeds with certainty degree $d = 0.75 \times \min\{0.9, 0.8\} = 0.6$, and proof cost $e = 3 + \max\{1, 2\} = 5$.

R_1 : `goodWork(X) <-(0.75,3)- famousAuthor(Y)#(0.5,100), wrote(Y,X)#?`
R_2 : `famousAuthor(shakespeare) <-(0.9,1)-`
R_3 : `wrote(shakespeare,king_lear) <-(1,1)-`

`S(king_lear,king_liar) = (0.8,2)`

Fig. 1. SQCLP($\mathcal{S}$, $\mathcal{U}\otimes\mathcal{W}$, $\mathcal{R}$) Program Fragment

---

It is useful to define some special types of program clauses and programs, as follows:

- A clause is called *attenuation-free* iff $\alpha = \mathbf{t}$. The name is justified because $\mathbf{t}$ is an identity element for the attenuation operator $\circ$, as explained in Subsection 2.2. By convention, attenuation-free clauses may be written with the simplified syntax $A \leftarrow B_1 \sharp w_1, \ldots, B_m \sharp w_m$.
- A clause is called *threshold-free* iff $w_j = ?$ for all $j = 1 \ldots m$. The name is justified because the threshold value $w_j = ?$ occurring as annotation of a body atom B_j does not impose any particular requirement to the qualification value of B_j. Threshold-free clauses may be written with the simplified syntax $A \xleftarrow{\alpha} B_1, \ldots, B_m$.
- A clause is called *qualification-free* iff it is both attenuation-free and threshold-free. These clauses may be written with the simplified syntax $A \leftarrow B_1, \ldots, B_m$. They behave just like those used in the classical CLP scheme.
- A clause is called *constraint-free* iff all its body atoms are defined.

- A program is called attenuation-free iff all its clauses are of this type. Threshold-free, qualification-free and constraint-free programs are defined similarly.

The more technical SQCLP($\mathcal{S},\mathcal{U},\mathcal{R}$)-program $\mathcal{P}$ presented below will serve as a *running example* to illustrate various points in the rest of the report.

Example 3.1 (*Running example*)
Assume unary constructors $c, c' \in DC^1$, binary predicate symbols $p, p', q \in DP^2$ and a ternary predicate symbol $r \in DP^3$. Consider the admissible triple $\langle \mathcal{S},\mathcal{U},\mathcal{R}\rangle$, where $\mathcal{S}$ is an $\mathcal{U}$-valued proximity relation such that $\mathcal{S}(c,c') = 0.9$ and $\mathcal{S}(p,p') = 0.8$. Let $\mathcal{P}$ be the SQCLP($\mathcal{S},\mathcal{U},\mathcal{R}$)-program consisting of the qualified clauses R_1, R_2 and R_3 listed below:

$R_1 : q(X, c(X)) \xleftarrow{1.0}$
$R_2 : p(c(X), Y) \xleftarrow{0.9} q(X, Y)\sharp 0.8$
$R_3 : r(c(X), Y, Z) \xleftarrow{0.9} q(X, Y)\sharp 0.8, cp_{\geq}(X, 0.0)\sharp ?$ □

As we will see in the Conclusions, the classical CLP scheme for Constraint Logic Programming originally introduced in (Jaffar and Lassez 1987) can be seen as a particular case of the SQCLP scheme. In the rest of this subsection we present a declarative semantics for SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-programs inspired by (Gabbrielli and Levi 1991; Gabbrielli et al. 1995). These papers provided three different program semantics $\mathcal{S}_i$ ($i = 1, 2, 3$) characterizing *valid ground solutions for goals*, *valid open solutions for goals* and *computed answers for goals* in CLP, respectively. In fact, the $\mathcal{S}_i$ semantics in (Gabbrielli and Levi 1991; Gabbrielli et al. 1995) were conceived as the CLP counterpart of previously known semantics for logic programming, namely the least ground Herbrand model semantics (Apt 1990; Lloyd 1987), the open Herbrand model semantics, also known as $\mathcal{C}$-semantics (Clark 1979; Falaschi et al. 1993), and the $\mathcal{S}$-semantics (Falaschi et al. 1989; Bossi et al. 1994); see (Apt and Gabbrielli 1994) for a very concise and readable overview.

In this report we restrict ourselves to develop a $\mathcal{S}_2$-like semantics which can be used to characterize valid open solutions for SQCLP goals as we will see in Subsection 3.2. As a basis for our semantics we use so-called *qc-atoms* of the form $A\sharp d \Leftarrow \Pi$, intended to assert that the atom A is entailed by the constraint set Π with qualification degree d. We also use a special entailment relation $\succcurlyeq_{\mathcal{D},\mathcal{C}}$ intended to capture some implications between qc-atoms whose validity depends neither on the proximity relation $\mathcal{S}$ nor on the semantics of defined predicates. A formal definition of these notions is as follows:

Definition 3.1 (*qc-atoms, observables and* ($\mathcal{D},\mathcal{C}$)*-entailment*)

1. *Qualified constrained atoms* (or simply *qc-atoms*) are statements of the form $A\sharp d \Leftarrow \Pi$, where $A \in \mathrm{At}(\Sigma, B, \mathcal{V}ar)$ is an atom, $d \in D$ is a qualification value, and $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$ is a finite set of constraints.
2. A qc-atom $A\sharp d \Leftarrow \Pi$ is called *defined*, *primitive* or *equational* according to the syntactic form of A.
3. A qc-atom $A\sharp d \Leftarrow \Pi$ is called *observable* iff $d \in D \setminus \{\mathbf{b}\}$ and Π is satisfiable.

4. Given two qc-atoms $\varphi : A\sharp d \Leftarrow \Pi$ and $\varphi' : A'\sharp d' \Leftarrow \Pi'$, we say that φ $(\mathcal{D},\mathcal{C})$-*entails* φ' (in symbols, $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$) iff there is some $\mathcal{C}$-substitution θ satisfying $A' = A\theta$, $d' \preccurlyeq d$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$. □

We will focus our attention on observable qc-atoms because they can be interpreted as observations of valid open solutions for atomic goals in SQCLP$(\mathcal{S},\mathcal{D},\mathcal{C})$ as we will see in Subsection 3.2. The example below illustrates the main technical ideas from Definition 3.1.

Example 3.2 (*Observable qc-atoms and* $(\mathcal{D},\mathcal{C})$*-entailment*)
Consider the admissible triple underlying Example 3.1 and the sets of $\mathcal{R}$-constraints:

$$\Pi = \{cp_{>}(X, 1.0),\ op_{+}(A, A, X),\ op_{\times}(2.0, A, Y)\}$$
$$\Pi' = \{cp_{\geq}(A, 3.0),\ op_{\times}(2.0, A, X),\ op_{+}(A, A, Y)\}$$

Then, the following are observable qc-atoms:

$$\varphi_1 = q(X, c'(Y))\sharp 0.9 \Leftarrow \Pi \qquad \varphi_3 = r(c'(Y), c(X), Z)\sharp 0.8 \Leftarrow \Pi$$
$$\varphi_2 = p'(c'(Y), c(X))\sharp 0.8 \Leftarrow \Pi \qquad \varphi'_3 = r(c'(Y), c(X), c(Z'))\sharp 0.7 \Leftarrow \Pi'$$

and the $(\mathcal{U},\mathcal{R})$-entailment $\varphi_3 \succcurlyeq_{\mathcal{U},\mathcal{R}} \varphi'_3$ is valid thanks to $\theta = \{Z \mapsto c(Z')\}$, which satisfies $r(c'(Y), c(X), c(Z')) = r(c'(Y), c(X), Z)\theta$, $0.7 \leq 0.8$ and $\Pi' \models_{\mathcal{R}} \Pi\theta$. □

The intended meaning of $\succcurlyeq_{\mathcal{D},\mathcal{C}}$ as an entailment relation not depending on the meanings of defined predicates motivates the first item in the next definition.

Definition 3.2 (*Interpretations*)
Let $\langle \mathcal{S},\mathcal{D},\mathcal{C} \rangle$ be any given admissible triple. Then:

1. A *qualified constrained interpretation* (or *qc-interpretation*) is a set $\mathcal{I}$ of observable defined qc-atoms closed under $(\mathcal{D},\mathcal{C})$-entailment. In other words, a set $\mathcal{I}$ of qc-atoms which satisfies the following two conditions:
   (a) Each $\varphi \in \mathcal{I}$ is an observable defined qc-atom.
   (b) If $\varphi \in \mathcal{I}$ and φ' is another defined observable qc-atom such that $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, then also $\varphi' \in \mathcal{I}$.
2. Assume any given qc-interpretation $\mathcal{I}$. For any observable qc-atom φ, we say that φ is valid in $\mathcal{I}$ modulo $\mathcal{S}$ (in symbols, $\mathcal{I} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$) iff some of the three cases below holds:
   (a) φ is defined and $\varphi \in \mathcal{I}$.
   (b) $\varphi : (t == s)\sharp d \Leftarrow \Pi$ is equational and $t \approx_{d,\Pi} s$.
   (c) $\varphi : \kappa\sharp d \Leftarrow \Pi$ is primitive and $\Pi \models_{\mathcal{C}} \kappa$. □

Note that a given interpretation $\mathcal{I}$ can include several observables $A\sharp d_i \Leftarrow \Pi$ for the same (possibly not ground) atom A but is not required to include on "optimal" observable $A\sharp d \Leftarrow \Pi$ with d computed as the *lub* of all d_i. By contrast, the other related works discussed in the Introduction view program interpretations as mappings $\mathcal{I}$ from the ground Herbrand base into some set of lattice elements (the real interval $[0, 1]$ in many cases). In such interpretations, each ground atom A has attached one single lattice element $d = \mathcal{I}(A)$ intended as "the optimal qualification"

for A. Our view of interpretations is closer to the expected operational behavior of goal solving systems and can be used to characterize the validity of solutions computed by such systems, as we will see in Subsection 3.2.

Note also that the notation $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ is defined only for the case that φ is observable. In the sequel, we will implicitly assume that φ is observable in any context where the notation $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ is used. The next technical result shows that validity in any given interpretation is closed under entailment.

Proposition 3.1 (*Entailment Property for Interpretations*)
Assume that $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ and $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$. Then $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi'$.

Proof
Due to the hypothesis $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$ we can assume $\varphi = (A \sharp d \Leftarrow \Pi)$, $\varphi' = (A' \sharp d' \Leftarrow \Pi')$ and some $\mathcal{C}$-substitution θ such that $A' = A\theta$, $d' \trianglelefteq d$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$. We now distinguish cases according to the syntactic form of φ:

1. φ is defined. In this case, φ' is also defined. Moreover, $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ is equivalent to $\varphi \in \mathcal{I}$ because of Definition 3.2, which implies $\varphi' \in \mathcal{I}$ because qc-interpretations are closed under $\succcurlyeq_{\mathcal{D},\mathcal{C}}$, which is equivalent to $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi'$ because of Definition 3.2.
2. φ is equational. In this case A and A' have the form $t == s$ and $t\theta == s\theta$, respectively. Moreover, $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ is equivalent to $t \approx_{d,\Pi} s$ because of Definition 3.2, which implies $t\theta \approx_{d,\Pi\theta} s\theta$ because of Lemma 2.8, which trivially implies $t\theta \approx_{d',\Pi'} s\theta$ because of $\Pi' \models_{\mathcal{C}} \Pi\theta$ and $d' \trianglelefteq d$, which is equivalent to $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi'$ because of Definition 3.2.
3. φ is primitive. In this case A and A' have the form κ and $\kappa\theta$, respectively. Moreover, $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ is equivalent to $\Pi \models_{\mathcal{C}} \kappa$ because of Definition 3.2, which implies $\Pi\theta \models_{\mathcal{C}} \kappa\theta$ because of Lemma 2.1, which implies $\Pi' \models_{\mathcal{C}} \kappa\theta$ because of $\Pi' \models_{\mathcal{C}} \Pi\theta$, which is equivalent to $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi'$ because of Definition 3.2. □

The definition below explains when a given interpretation is regarded as a model of a given program, as well as the related notion of semantic consequence.

Definition 3.3 (*Models and semantic consequence*)
Let a SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$ and an observable qc-atom $\varphi : p'(\overline{t'}_n) \sharp d \Leftarrow \Pi$ be given. φ is an *immediate consequence* of a qc-interpretation $\mathcal{I}$ via a program rule $(R_l : p(\overline{t}_n) \stackrel{\alpha}{\Leftarrow} B_1 \sharp w_1, \ldots, B_m \sharp w_m) \in \mathcal{P}$ iff there exist a $\mathcal{C}$-substitution θ and a choice of qualification values $d_0, d_1, \ldots, d_n, e_1, \ldots, e_m \in D \setminus \{\mathbf{b}\}$ such that:

(a) $\mathcal{S}(p', p) = d_0$
(b) $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t'_i == t_i\theta) \sharp d_i \Leftarrow \Pi$ (i.e. $t'_i \approx_{d_i,\Pi} t_i\theta$) for $i = 1 \ldots n$
(c) $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\theta \sharp e_j \Leftarrow \Pi$ with $e_j \trianglerighteq^{?} w_j$ for $j = 1 \ldots m$
(d) $d \trianglelefteq \bigsqcap_{i=0}^{n} d_i \sqcap \alpha \circ \bigsqcap_{j=1}^{m} e_j$ [i.e., $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha \circ e_j$ $(1 \leq j \leq m)$]

Note that the qualification value d attached to φ is limited by two kinds of upper bounds: d_i $(0 \leq i \leq n)$, i.e. the $\mathcal{S}$-proximity between $p'(\overline{t'}_n)$ and the head of $R_l\theta$; and $\alpha \circ e_j$ $(1 \leq j \leq m)$, i.e. the qualification values of the atoms in the body of $R_l\theta$ attenuated w.r.t. R_l's attenuation factor α. Moreover, the inequalities

$e_j \trianglerighteq^? w_j$ $(1 \leq j \leq m)$ are required in order to impose the threshold conditions within R_l's body. As already explained at the beginning of this subsection, $e_j \trianglerighteq^? w_j$ means that either $w_j = ?$ or else $w_j \in D \setminus \{\mathbf{b}\}$ and $e_j \trianglerighteq w_j$. Now we can define:

1. $\mathcal{I}$ is a *model* of a program rule $R_l \in \mathcal{P}$ (in symbols, $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} R_l$) iff every defined observable qc-atom φ which is an immediate consequence of $\mathcal{I}$ via R_l verifies $\varphi \in \mathcal{I}$; and $\mathcal{I}$ is a *model* of $\mathcal{P}$ (in symbols, $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P}$) iff $\mathcal{I}$ is a model of every program rule $R_l \in \mathcal{P}$.
2. φ is a *semantic consequence* of $\mathcal{P}$ (in symbols, $\mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$) iff $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ for every qc-interpretation $\mathcal{I}$ such that $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P}$. □

The next example may serve as a concrete illustration:

Example 3.3 (*Models and semantic consequence*)
Recall the SQCLP($\mathcal{S},\mathcal{U},\mathcal{R}$)-program $\mathcal{P}$ from Example 3.1. Let us show that the three qc-atoms φ_1, φ_2 and φ_3 from Example 3.2 are semantic consequences of $\mathcal{P}$:

1. Assume an arbitrary model $\mathcal{I} \models_{\mathcal{S},\mathcal{U},\mathcal{R}} \mathcal{P}$. Note that the atom underlying φ_1 and the head atom of R_1 are $q(X, c'(Y))$ and $q(X, c(X))$, respectively. Since $\mathcal{S}(c, c') = 0.9$ and $\Pi \models_{\mathcal{C}} X == Y$, φ_1 can be obtained as an immediate consequence of $\mathcal{I}$ via R_1 using $\theta = \varepsilon$. Therefore $\varphi_1 \in \mathcal{I}$ and we can conclude that $\mathcal{P} \models_{\mathcal{S},\mathcal{U},\mathcal{R}} \varphi_1$.
2. Assume an arbitrary model $\mathcal{I} \models_{\mathcal{S},\mathcal{U},\mathcal{R}} \mathcal{P}$. Consider the substitution $\theta = \{Y \mapsto c'(Y)\}$. Note that the atom underlying φ_2 and the head atom of $R_2\theta$ are $p'(c'(Y), c(X))$ and $p(c(X), c'(Y))$, respectively. Moreover, $\varphi_1 \in \mathcal{I}$ (due to the previous item) and the atom $q(X, c'(Y))$ underlying φ_1 is the same as the atom in the body of $R_2\theta$. These facts together with $\mathcal{S}(p, p') = 0.8$, $\mathcal{S}(c, c') = 0.9$ and $\Pi \models_{\mathcal{C}} X == Y$ allow to obtain φ_2 as an immediate consequence of $\mathcal{I}$ via R_2. Therefore $\varphi_2 \in \mathcal{I}$ and we can conclude that $\mathcal{P} \models_{\mathcal{S},\mathcal{U},\mathcal{R}} \varphi_2$.
3. Assume an arbitrary model $\mathcal{I} \models_{\mathcal{S},\mathcal{U},\mathcal{R}} \mathcal{P}$. Consider again the substitution $\theta = \{Y \mapsto c'(Y)\}$. Note that the atom underlying φ_3 and the head atom of $R_3\theta$ are $r(c'(Y), c(X), Z)$ and $r(c(X), c'(Y), Z)$, respectively. Moreover, the two annotated atoms $B_j\theta \sharp w_j$ $(1 \leq j \leq 2)$ occurring in the body of $R_3\theta$ are such that $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\theta \sharp e_j \Leftarrow \Pi$ for suitable values $e_j \geq^? w_j$, namely $e_1 = 0.9$ and $e_2 = 1.0$. Note that $e_1 = 0.9$ works because $B_1\theta$ is the atom $q(X, c'(Y))$ underlying φ_1 and $\varphi_1 \in \mathcal{I}$, as proved in the first item of this example. On the other hand, $e_2 = 1.0$ works because $B_2\theta$ is the primitive atom $cp_{\geq}(X, 0.0)$ which is trivially entailed by Π. All these facts, together with $\mathcal{S}(c, c') = 0.9$, $0.8 \leq 0.9 \times 0.9$ and $\Pi \models_{\mathcal{C}} X == Y$ allow to obtain φ_3 as an immediate consequence of $\mathcal{I}$ via R_3. Therefore $\varphi_3 \in \mathcal{I}$ and we can conclude that $\mathcal{P} \models_{\mathcal{S},\mathcal{U},\mathcal{R}} \varphi_3$. □

Now we are ready to obtain results on the declarative semantics of programs in the SQCLP scheme. We will characterize the observable consequences of a given program $\mathcal{P}$ in two different, but equivalent, ways: either using the interpretation transformer presented in Subsection 3.1.1, or using the extension of Horn Logic presented in Subsection 3.1.2. In both approaches, we will prove the existence of a least model $\mathcal{M}_{\mathcal{P}}$ for each given program $\mathcal{P}$.

3.1.1 A Fixpoint Semantics

A well-known way of characterizing models and least models of programs in declarative languages proceeds by considering a lattice structure for the family of all program interpretations, and using an interpretation transformer to compute the immediate consequences obtained from program rules. This kind of approach is well known for logic programming (van Emden and Kowalski 1976; Apt and van Emden 1982; Lloyd 1987; Apt 1990) and constraint logic programming (Gabbrielli and Levi 1991; Gabbrielli et al. 1995; Jaffar et al. 1998). It has been used also in various extensions of logic programming designed to support uncertain reasoning, such as quantitative logic programming (van Emden 1986), its extension to qualified logic programming (Rodríguez-Artalejo and Romero-Díaz 2008b) quantitative constraint logic programming (Riezler 1996; Riezler 1998), similarity-based logic programming (Sessa 2002) and proximity-based logic programming in the sense of Bousi∼Prolog (Julián-Iranzo and Rubio-Manzano 2009a).

The SQCLP scheme is intended to unify all these logic programming extensions in a common framework. This subsection is based on the declarative semantics given in (Rodríguez-Artalejo and Romero-Díaz 2008b; Rodríguez-Artalejo and Romero-Díaz 2008a), extended to deal with constraints and proximity relations. Our first result provides a lattice of program interpretations.

Proposition 3.2 (*Lattice of Interpretations*)
$\mathrm{Int}_{\mathcal{D},\mathcal{C}}$, defined as the set of all qc-interpretations over the qualification domain $\mathcal{D}$ and the constraint domain $\mathcal{C}$, is a complete lattice w.r.t. the set inclusion ordering $\subseteq$. Moreover, the bottom element $\perp\!\!\!\perp$ and the top element $\top\!\!\!\top$ of this lattice are characterized as $\perp\!\!\!\perp = \emptyset$ and $\top\!\!\!\top = \{\varphi \mid \varphi \text{ is a defined observable qc-atom}\}$ and for any subset $I \subseteq \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ its greatest lower bound (glb) and least upper bound (lub) are characterized as follows:

1. The glb of I (written as $\bigsqcap I$) is $\bigcap_{\mathcal{I}\in I} \mathcal{I}$, understood as $\top\!\!\!\top$ if $I = \emptyset$; and
2. The lub of I (written as $\bigsqcup I$) is $\bigcup_{\mathcal{I}\in I} \mathcal{I}$, understood as $\perp\!\!\!\perp$ if $I = \emptyset$.

Proof
Both $\perp\!\!\!\perp$ and $\top\!\!\!\top$ are qc-interpretations because they are sets of defined observable qc-atoms and they are closed under $(\mathcal{D},\mathcal{C})$-entailment for trivial reasons, namely: $\perp\!\!\!\perp$ is empty and $\top\!\!\!\top$ includes all the defined observables. Moreover, they are the minimum and the maximum of $\mathrm{Int}_{\mathcal{D},\mathcal{C}}$ w.r.t. $\subseteq$ because $\perp\!\!\!\perp \subseteq \mathcal{I} \subseteq \top\!\!\!\top$ is trivially true for each $\mathcal{I} \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$. Thus, we have only left to prove 1. and 2.:

1. $\bigcap_{\mathcal{I}\in I} \mathcal{I}$ is obviously a set of defined observable qc-atoms because this is the case for each $\mathcal{I} \in I$. Given any $\varphi \in \bigcap_{\mathcal{I}\in I}$ and any observable defined qc-atom φ' such that $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, we get $\varphi' \in \bigcap_{\mathcal{I}\in I} \mathcal{I}$ as an obvious consequence of the fact that each $\mathcal{I} \in I$ is closed under $(\mathcal{D},\mathcal{C})$-entailment. Therefore, $\bigcap_{\mathcal{I}\in I} \mathcal{I} \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$. Obviously, $\bigcap_{\mathcal{I}\in I} \mathcal{I}$ is trivially a lower bound of I w.r.t. $\subseteq$. Moreover, $\bigcap_{\mathcal{I}\in I} \mathcal{I}$ is the glb of I, because any given lower bound $\mathcal{J}$ of I verifies $\mathcal{J} \subseteq \mathcal{I}$ for every $\mathcal{I} \in I$ and thus $\mathcal{J} \subseteq \bigcap_{\mathcal{I}\in I} \mathcal{I}$. Therefore, $\bigcap_{\mathcal{I}\in I} \mathcal{I} = \bigsqcap I$.

2. Using the properties of the union of a family of sets it is easy to prove that $\bigcup_{\mathcal{I}\in I}\mathcal{I} \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ and also that $\bigcup_{\mathcal{I}\in I}\mathcal{I}$ is the lub of I w.r.t. $\subseteq$. A more detailed reasoning would be similar to the previous item. Therefore, $\bigcup_{\mathcal{I}\in I}\mathcal{I} = \bigsqcup I$. □

Next we define an *interpretation transformer* $\mathrm{T}_{\mathcal{P}}$, intended to compute the immediate consequences obtained from a given qc-interpretation via the program rules belonging to $\mathcal{P}$.

Definition 3.4 (*Interpretations Transformer*)
Let $\mathcal{P}$ be a fixed SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program. The interpretations transformer $\mathrm{T}_{\mathcal{P}} : \mathrm{Int}_{\mathcal{D},\mathcal{C}} \to \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ is defined by the condition:

$$\mathrm{T}_{\mathcal{P}}(\mathcal{I}) =_{\mathrm{def}} \{\varphi \mid \varphi \text{ is an immediate consequence of } \mathcal{I} \text{ via some } R_l \in \mathcal{P}\} \ . \quad \square$$

The computation of immediate consequences of a given qc-interpretation $\mathcal{I}$ via a given program rule R_l has been already explained in Definition 3.3. The following example illustrates the workings of $\mathrm{T}_{\mathcal{P}}$.

Example 3.4 (*Interpretation transformer in action*)
Recall again the SQCLP($\mathcal{S},\mathcal{U},\mathcal{R}$)-program $\mathcal{P}$ from Example 3.1 and the observable defined qc-atoms φ_1, φ_2 and φ_3 from Example 3.2. Then:

1. The arguments given in Example 3.3(1) can be easily reused to show that φ_1 is an immediate consequence of the empty interpretation $\perp\!\!\!\perp$ via the program rule R_1. Therefore, $\varphi_1 \in \mathrm{T}_{\mathcal{P}}(\perp\!\!\!\perp)$.
2. The arguments given in Example 3.3(2) can be easily reused to show that φ_1 is an immediate consequence of $\mathcal{I}$ via the program rule R_2, provided that $\varphi_1 \in \mathcal{I}$. Therefore, $\varphi_2 \in \mathrm{T}_{\mathcal{P}}(\mathrm{T}_{\mathcal{P}}(\perp\!\!\!\perp))$.
3. The arguments given in Example 3.3(3) can be easily reused to show that φ_3 is an immediate consequence of $\mathcal{I}$ via the program rule R_3, provided that $\varphi_1 \in \mathcal{I}$. Therefore, $\varphi_3 \in \mathrm{T}_{\mathcal{P}}(\mathrm{T}_{\mathcal{P}}(\perp\!\!\!\perp))$. □

The next proposition states the main properties of interpretation transformers.

Proposition 3.3 (*Properties of interpretation transformers*)
Let $\mathcal{P}$ be any fixed SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program. Then:

1. $\mathrm{T}_{\mathcal{P}}$ is a well defined mapping, i.e. for all $\mathcal{I} \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ one has $\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$.
2. $\mathrm{T}_{\mathcal{P}}$ is monotonic and continuous.
3. For all $\mathcal{I} \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ one has: $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P} \iff \mathrm{T}_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$, That is, the models of $\mathcal{P}$ are precisely the pre-fixpoints of $\mathrm{T}_{\mathcal{P}}$.

Proof

1. By definition, $\mathrm{T}_{\mathcal{P}}(\mathcal{I})$ is a set of observable defined qc-atoms. It is sufficient to prove that it is closed under $(\mathcal{D},\mathcal{C})$-entailment. Let us assume two observable defined qc-atoms φ and φ' such that $\varphi \in \mathrm{T}_{\mathcal{P}}(\mathcal{I})$ and $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$. Because of $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$ we can assume $\varphi : p(\overline{t}_n)\sharp d \Leftarrow \Pi$, $\varphi' : p(\overline{t'}_n)\sharp d' \Leftarrow \Pi'$ and some substitution θ such that $p(\overline{t'}_n) = p(\overline{t}_n)\theta$, $d' \trianglelefteq d$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$. Because of $\varphi \in \mathrm{T}_{\mathcal{P}}(\mathcal{I})$, we can assume that φ is an immediate consequence of $\mathcal{I}$ via some $R_l \in \mathcal{P}$. More precisely, we can assume $(R_l : q(\overline{s}_n) \stackrel{\alpha}{\Leftarrow} B_1\sharp w_1, \ldots, B_m\sharp w_m) \in \mathcal{P}$, some substitution σ and some qualification values $d_0, d_1, \ldots, d_n, e_1, \ldots, e_m \in D \setminus \{\mathbf{b}\}$ such that

(a) $\mathcal{S}(p,q) = d_0$,
(b) $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t_i == s_i\sigma)\sharp d_i \Leftarrow \Pi$ for $i = 1 \ldots n$,
(c) $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\sigma\sharp e_j \Leftarrow \Pi$ with $e_j \trianglerighteq^? w_j$ for $j = 1 \ldots m$,
(d) $d \trianglelefteqslant \bigsqcap_{i=0}^{n} d_i \sqcap \alpha \circ \bigsqcap_{j=1}^{m} e_j$ [i.e., $d \trianglelefteqslant d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteqslant \alpha \circ e_j$ $(1 \leq j \leq m)$].

In order to show that $\varphi' \in \mathrm{T}_{\mathcal{P}}(\mathcal{I})$, we claim that φ' can be computed as an immediate consequence of $\mathcal{I}$ via the same program rule R_l, using the substitution $\sigma\theta$ and the qualification values $d_0, d_1, \ldots, d_n, e_1, \ldots, e_m \in D \setminus \{\mathbf{b}\}$. To justify this claim it is enough to check the following items:

(a') $\mathcal{S}(p,q) = d_0$,
(b') $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t'_i == s_i\sigma\theta)\sharp d_i \Leftarrow \Pi'$ for $i = 1 \ldots n$,
(c') $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\sigma\theta\sharp e_j \Leftarrow \Pi'$ with $e_j \trianglerighteq^? w_j$ for $j = 1 \ldots m$,
(d') $d \trianglelefteqslant \bigsqcap_{i=0}^{n} d_i \sqcap \alpha \circ \bigsqcap_{j=1}^{m} e_j$ [i.e., $d \trianglelefteqslant d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteqslant \alpha \circ e_j$ $(1 \leq j \leq m)$].

These four items closely correspond to items (a)-(d) above. More specifically:
— Items (a') and (d') are identical to items (a) and (d), respectively.
— Regarding item (b'): For $i = 1 \ldots n$, $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t'_i == s_i\sigma\theta)\sharp d_i \Leftarrow \Pi$ is the same as $t_i\theta \approx_{d_i,\Pi'} s_i\sigma\theta$. Because of Lemma 2.8, this is a consequence of $\Pi' \models_{\mathcal{C}} \Pi\theta$ and $t_i \approx_{d_i,\Pi} s_i\sigma$, which is ensured by item (b).
— Regarding item (c'): For $j = 1 \ldots m$, $e_j \trianglerighteq^? w_j$ is ensured by item (c), and $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\sigma\theta\sharp e_j \Leftarrow \Pi'$ follows from $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\sigma\sharp e_j \Leftarrow \Pi$ –also ensured by item (c)– and the entailment property for interpretations (Proposition 3.1), which can be applied because $B_j\sigma\sharp e_j \Leftarrow \Pi \succcurlyeq_{\mathcal{D},\mathcal{C}} B_j\sigma\theta\sharp e_j \Leftarrow \Pi'$.

2. Monotonicity means that the inclusion $\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \subseteq \mathrm{T}_{\mathcal{P}}(\mathcal{J})$ holds whenever $\mathcal{I} \subseteq \mathcal{J}$. This follows very easily from

$$(\spadesuit) \quad \mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \text{ and } \mathcal{I} \subseteq \mathcal{J} \Longrightarrow \mathcal{J} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$$

which is a trivial consequence of Definition 3.2.
Continuity means that the equation $\mathrm{T}_{\mathcal{P}}(\bigsqcup I) = \bigsqcup\{\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \mid \mathcal{I} \in I\}$ holds for any directed set $I \subseteq \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ of qc-interpretations. Recall that $I \subseteq \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ is called directed iff every finite subset $I_0 \subseteq I$ has some upper bound $\mathcal{I} \in I$. We show that $\mathrm{T}_{\mathcal{P}}(\bigsqcup I) = \bigsqcup\{Tp(\mathcal{I}) \mid \mathcal{I} \in I\}$ holds by proving the two inclusions separately:

(a) For each fixed $\mathcal{I}_0 \in I$, $\mathrm{T}_{\mathcal{P}}(\mathcal{I}_0) \subseteq \mathrm{T}_{\mathcal{P}}(\bigsqcup I)$ follows from $\mathcal{I}_0 \subseteq \bigsqcup I$ and monotonicity of $\mathrm{T}_{\mathcal{P}}$. Then, the inclusion $\bigsqcup\{\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \mid \mathcal{I} \in I\} \subseteq \mathrm{T}_{\mathcal{P}}(\bigsqcup I)$ holds by definition of supremum.
(b) In order to prove the opposite inclusion $\mathrm{T}_{\mathcal{P}}(\bigsqcup I) \subseteq \bigsqcup\{\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \mid \mathcal{I} \in I\}$, consider an arbitrary $\varphi \in \mathrm{T}_{\mathcal{P}}(\bigsqcup I)$. Due to Definition 3.4, φ is an immediate consequence of $\bigsqcup I$ via some program rule $R_l \in \mathcal{P}$. Because of the first item of Definition 3.3, φ is an immediate consequence of $\bigsqcup I$ via R_l due to finitely many qc-facts of the form $B_j\theta\sharp e_j \Leftarrow \Pi$ (coming from the body of a suitable instance of R_l) that are valid in $\bigsqcup I$. Because of ($\spadesuit$) and the assumption that I is a directed set, it is possible to choose some $\mathcal{I}_0 \in I$ such that all the qc-facts $B_j\theta\sharp e_j \Leftarrow \Pi$ are valid in $\mathcal{I}_0$. Then, φ is an immediate consequence of this particular $\mathcal{I}_0 \in I$ via R_l. Therefore, $\varphi \in \mathrm{T}_{\mathcal{P}}(\mathcal{I}_0) \subseteq \bigsqcup\{\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \mid \mathcal{I} \in I\}$.

3. According to Definition 3.3, $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P}$ holds iff every observable defined qc-atom φ which is an immediate consequence of $\mathcal{I}$ via the program rules $R_l \in \mathcal{P}$ verifies $\varphi \in \mathcal{I}$. According to Definition 3.4, $\mathrm{T}_{\mathcal{P}}(\mathcal{I})$ is just the set of all the defined observable qc-atoms φ that can be obtained as immediate consequences of $\mathcal{I}$ via the program rules $R_l \in \mathcal{P}$. Consequently, $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P}$ holds iff $\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$. □

The theorem below is the main result in this subsection.

Theorem 3.1 (*Fixpoint characterization of least program models*)
Every SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$ has a *least model* $\mathcal{M}_{\mathcal{P}}$, smaller than any other model of $\mathcal{P}$ w.r.t. the set inclusion ordering of the interpretation lattice $\mathrm{Int}_{\mathcal{D},\mathcal{C}}$. Moreover, $\mathcal{M}_{\mathcal{P}}$ can be characterized as the *least fixpoint* of $\mathrm{T}_{\mathcal{P}}$ as follows:

$$\mathcal{M}_{\mathcal{P}} = \mathit{lfp}(\mathrm{T}_{\mathcal{P}}) = \bigcup_{k \in \mathbb{N}} \mathrm{T}_{\mathcal{P}}{\uparrow}^k(\perp\!\!\!\perp) \ . \quad \square$$

Proof
As usual, a given $\mathcal{I} \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ is called a fixpoint of $\mathrm{T}_{\mathcal{P}}$ iff $\mathrm{T}_{\mathcal{P}}(\mathcal{I}) = \mathcal{I}$, and $\mathcal{I}$ is called a pre-fixpoint of $\mathrm{T}_{\mathcal{P}}$ iff $\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$. Due to a well-known theorem by Knaster and Tarski, see (Tarski 1955), a monotonic mapping from a complete lattice into itself always has a least fixpoint which is also its least pre-fixpoint. In the case that the mapping is continuous, its least fixpoint can be characterized as the lub of the sequence of lattice elements obtained by reiterated application of the mapping to the bottom element. Combining these results with Proposition 3.3 trivially proves the theorem. □

3.1.2 An equivalent Proof-theoretic Semantics

In order to give a logical view of program semantics and an alternative characterization of least program models, we define the *Proximity-based Qualified Constrained Horn Logic* SQCHL($\mathcal{S},\mathcal{D},\mathcal{C}$) as a formal inference system consisting of the three inference rules displayed in Figure 2.

SQDA $$\frac{(\ (t'_i == t_i\theta)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n} \quad (\ B_j\theta\sharp e_j \Leftarrow \Pi\ )_{j=1\ldots m}}{p'(\overline{t'}_n)\sharp d \Leftarrow \Pi}$$

if $(p(\overline{t}_n) \overset{\alpha}{\leftarrow} B_1\sharp w_1, \ldots, B_m\sharp w_m) \in \mathcal{P}$, θ subst., $\mathcal{S}(p',p) = d_0 \neq \mathbf{b}$, $e_j \trianglerighteq^? w_j$ $(1 \leq j \leq m)$ and $d \trianglelefteq \bigsqcap_{i=0}^{n} d_i \sqcap \alpha \circ \bigsqcap_{j=1}^{m} e_j$.

SQEA $$\frac{}{(t == s)\sharp d \Leftarrow \Pi}$$ if $t \approx_{d,\Pi} s$.

SQPA $$\frac{}{\kappa\sharp d \Leftarrow \Pi}$$ if $\Pi \models_{\mathcal{C}} \kappa$.

Fig. 2. Proximity-based Qualified Constrained Horn Logic

The three inference rules are intended to work with observable qc-atoms. Rule **SQDA** is used to infer defined qc-atoms. It formalizes an extension of the classical *Modus Ponens* inference, allowing to infer a defined qc-atom $p'(\overline{t'}_n)\sharp d \Leftarrow \Pi$ by means of an instance of a program clause with head $p(\overline{t}_n)\theta$ and body atoms $B_j\theta\sharp w_j$. The n premises $(t'_i == t_i\theta)\sharp d_i \Leftarrow \Pi$ combined with the side condition $\mathcal{S}(p', p) = d_0 \neq \mathbf{b}$ ensure the "equality" between $p'(\overline{t'}_n)$ and $p(\overline{t}_n)\theta$ modulo $\mathcal{S}$; the m premises $B_j\theta\sharp e_j \Leftarrow \Pi$ require to prove the body atoms; and the side conditions $e_j \trianglerighteq^? w_j$ and $d \trianglelefteq \bigsqcap_{i=0}^{n} d_i \sqcap \alpha \circ \bigsqcap_{j=1}^{m} e_j$ check the threshold conditions of the body atoms and impose the proper relationships between the qualification value attached to the conclusion and the qualification values attached to the premises. In particular, the inequality $d \trianglelefteq \alpha \circ \bigsqcap_{j=1}^{m} e_j$ is imposed, meaning that the qualification value attached to a clause's head cannot exceed the glb of the qualification values attached to the body atoms attenuated by the clause's attenuation factor. Rules **SQEA** and **SQPA** are used to infer equational and primitive qc-atoms, respectively. Rule **SQEA** is designed to work with term proximity w.r.t. Π in the sense of Definition 2.16, inferring $(t == s)\sharp d \Leftarrow \Pi$ just in the case that $t \approx_{d,\Pi} s$ holds. Rule **SQPA** infers $\kappa\sharp d \Leftarrow \Pi$ for an arbitrary $d \in D \setminus \{\mathbf{b}\}$, provided that $\Pi \models_{\mathcal{C}} \kappa$ holds. This makes sense because the requirements for admissible triples in Definition 2.14 include the assumption that $\mathcal{S}(p, p') \neq \mathbf{b}$ cannot happen if $p, p' \in PP$ are syntactically different primitive predicate symbols.

As usual in formal inference systems, SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proofs can be represented as *proof trees* T whose nodes correspond to qc-atoms, each node being inferred from its children by means of some SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ inference step. In the rest of the report we will use the following notations:

- $\|T\|$ will denote the *size* of the proof tree T, measured as its number of nodes, which equals the number of inference steps in the SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proof represented by T.
- $\|T\|_d$ will denote the number of nodes of the proof tree T that represent conclusions of **SQDA** inference steps. Obviously, $\|T\|_d \leq \|T\|$.
- $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ will indicate that φ can be inferred from $\mathcal{P}$ in SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$.
- $\mathcal{P} \vdash^{k}_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ will indicate that φ can be inferred from $\mathcal{P}$ in SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ using some proof tree T such that $\|T\|_d = k$.

The next example shows a SQCHL$(\mathcal{S}, \mathcal{U}, \mathcal{R})$ proof tree.

Example 3.5 (SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proof tree)
Recall the proximity relation $\mathcal{S}$ and the program $\mathcal{P}$ from our running Example 3.1, as well as the observable qc-statement $\varphi_2 = p'(c'(Y), c(X))\sharp 0.8 \Leftarrow \Pi$ already known from Example 3.2. A SQCHL$(\mathcal{S}, \mathcal{U}, \mathcal{R})$ proof tree witnessing $\mathcal{P} \vdash_{\mathcal{S},\mathcal{U},\mathcal{R}} \varphi_2$ can be displayed as follows:

$$\spadesuit = \dfrac{\dfrac{}{(Y == Y)\sharp 1.0 \Leftarrow \Pi}\,(5) \qquad \dfrac{}{(c(X) == c(Y))\sharp 1.0 \Leftarrow \Pi}\,(6)}{q(Y, c(X))\sharp 1.0 \Leftarrow \Pi}\,(4)$$

$$\dfrac{\dfrac{}{(c'(Y) == c(Y))\sharp 0.8 \Leftarrow \Pi}\ (2) \qquad \dfrac{}{(c(X) == c(X))\sharp 1.0 \Leftarrow \Pi}\ (3) \qquad \spadesuit\ (4)}{p'(c'(Y), c(X))\sharp 0.8 \Leftarrow \Pi}\ (1)$$

The inference steps in this proof are commented below. For the sake of clarity, we have used a different variant of the corresponding program clause for each each application of the inference rule **SQDA**.

(1) **SQDA** step with clause $R_1 = (\ p(c(X_1), Y_1) \xleftarrow{0.9} q(X_1, Y_1)\ )$ instantiated by substitution $\theta_1 = \{X_1 \mapsto Y, Y_1 \mapsto c(X)\}$. Note that 0.8 satisfies $0.8 \leq \mathcal{S}(p, p') = 0.8$, $0.8 \leq 0.8$, $0.8 \leq 1.0$, $0.8 \leq 0.9 \times 1.0$.
(2) **SQEA** step. $c'(Y) \approx_{0.8,\Pi} c(Y)$ holds due to $c'(Y) \approx_\Pi c'(Y)$, $c(Y) \approx_\Pi c(Y)$ and $c'(Y) \approx_{0.8} c(Y)$.
(3) **SQEA** step. $c(X) \approx_{1.0,\Pi} c(X)$ holds for trivial reasons.
(4) **SQDA** step with clause $R_2 = (\ q(X_2, c(X_2)) \xleftarrow{1.0}\ )$ instantiated by substitution $\theta_2 = \{X_2 \mapsto Y\}$. Note that 1.0 satisfies $1.0 \leq \mathcal{S}(q, q) = 1.0$ and $1.0 \leq 1.0$.
(5) **SQEA** step. $Y \approx_{1.0,\Pi} Y$ holds for trivial reasons.
(6) **SQEA** step. $c(X) \approx_{1.0,\Pi} c(Y)$ holds due to $c(X) \approx_\Pi c(Y)$ (which follows from $\Pi \models_\mathcal{R} X == Y$) and $c(X) \approx_{1.0} c(X)$. □

The next technical lemma establishes two basic properties of formal inference in the SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$) logic.

Lemma 3.1 (*Properties of SQCHL*($\mathcal{S}, \mathcal{D}, \mathcal{C}$) *derivability*)
Let $\mathcal{P}$ be any SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program. Then:

1. *$\mathcal{P}$-independent Inferences*:
   Given any $\mathcal{C}$-based qc-atom φ and any qc-interpretation $\mathcal{I}$, one has:
   $$\mathcal{P} \vdash^0_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \iff \mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \iff \mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi\ .$$
2. *Entailment Property for Programs*:
   Given any pair of qc-atoms φ and φ' such that $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ with inference proof tree T and $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, then $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi'$ with an inference proof tree T' of the same size and structure as T.

Proof of $\mathcal{P}$-independent Inferences
Since φ is $\mathcal{C}$-based, we can assume $\varphi = A\sharp d \Leftarrow \Pi$ where A is either an equation or a primitive atom. In both cases the equivalence $\mathcal{P} \vdash^0_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \iff \mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ is obvious. In order to prove the equivalence $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \iff \mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ we distinguish the two cases:

1. φ is equational. Then A has the form $t == s$. Considering the SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-inference rule **SQEA** and the second item of Definition 3.2, we get
   $$\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \iff s \approx_{d,\Pi} t \iff \mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi\ .$$

2. φ is primitive. Then A is a primitive atom κ. Considering the SQCHL($\mathcal{S},\mathcal{D},\mathcal{C}$)-inference rule **SQPA** and the second item of Definition 3.2, we get

$$\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longleftrightarrow \Pi \models_{\mathcal{C}} \kappa \Longleftrightarrow \mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \ . \quad \square$$

Proof of Entailment Property for Programs
Due to the hypothesis $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$ and Definition 3.1, we can assume $\varphi = A\sharp d \Leftarrow \Pi$ and $\varphi' = A'\sharp d' \Leftarrow \Pi'$ with $A' = A\theta$, $d' \trianglelefteq d$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$ for some substitution θ. We reason by complete induction on $\|T\|$. There are three possible cases, according to the the syntactic form of the atom A. In each case we argue how to build the desired proof tree T'.

1. A is a defined atom: In this case, A is $p(\overline{t}_n)$ with $p \in DP^n$, and A': is $p(\overline{t'}_n)$ with $p(\overline{t'}_n) = p(\overline{t}_n)\theta$. Moreover, T must be a proof tree of the following form:

$$T : \frac{\left(\dfrac{}{(t_i == s_i\sigma)\sharp d_i \Leftarrow \Pi}\right)_{i=1\ldots n} \quad \left(\dfrac{\cdots}{B_j\sigma\sharp e_j \Leftarrow \Pi}\right)_{j=1\ldots m}}{p(\overline{t}_n)\sharp d \Leftarrow \Pi}\ \textbf{SQDA}$$

where:

- The **SQDA** root inference uses some $R_l : (q(\overline{s}_n) \xleftarrow{\alpha} B_1\sharp w_1, \ldots, B_m\sharp w_m) \in \mathcal{P}$, some substitution σ and some qualification values $d_0, d_1, \ldots, d_n, e_1, \ldots e_m \in D \setminus \{\mathbf{b}\}$ such that $\mathcal{S}(p,q) = d_0 \neq \mathbf{b}$, $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha \circ e_j$ $(1 \leq j \leq m)$.
- For $i = 1 \ldots n$, $(t_i == s_i\sigma)\sharp d_i \Leftarrow \Pi$ has a proof tree T_i^h with $\|T_i^h\| < \|T\|$.
- For $j = 1 \ldots m$, $B_j\sigma\sharp e_j \Leftarrow \Pi$ has a proof tree T_j^b with $\|T_j^b\| < \|T\|$.

Then, T' can be built as a proof tree of the form:

$$T' : \frac{\left(\dfrac{}{(t'_i == s_i\sigma\theta)\sharp d_i \Leftarrow \Pi'}\right)_{i=1\ldots n} \quad \left(\dfrac{\cdots}{B_j\sigma\theta\sharp e_j \Leftarrow \Pi'}\right)_{j=1\ldots m}}{p(\overline{t'}_n)\sharp d' \Leftarrow \Pi'}\ \textbf{SQDA}$$

where:

- The **SQDA** root inference uses the same program clause $R_l \in \mathcal{P}$, the substitution $\sigma\theta$ and the same qualification values d_i $(0 \leq i \leq n)$ and e_j $(1 \leq j \leq m)$, satisfying $\mathcal{S}(p,q) = d_0 \neq \mathbf{b}$, $d' \trianglelefteq d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d' \trianglelefteq d \trianglelefteq \alpha \circ e_j$ $(1 \leq j \leq m)$.
- For $i = 1 \ldots n$, $(t'_i == s_i\sigma\theta)\sharp d_i \Leftarrow \Pi'$ has a proof tree $T_i'^h$ of the same size and structure as T_i^h. In fact, $T_i'^h$ can be obtained by induction hypothesis applied to T_i^h, which is allowed because $\|T_i^h\| < \|T\|$ and $(t_i == s_i\sigma)\sharp d_i \Leftarrow \Pi \succcurlyeq_{\mathcal{D},\mathcal{C}} (t'_i == s_i\sigma\theta)\sharp d_i \Leftarrow \Pi'$. Note that this entailment holds thanks to substitution θ, since $t'_i = t_i\theta$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$.
- For $j = 1 \ldots m$, $B_j\sigma\theta\sharp e_j \Leftarrow \Pi'$ has a proof tree $T_j'^b$ of the same size and structure as T_j^b. In fact, $T_j'^b$ can be obtained by induction hypothesis applied to T_j^b, which is allowed because $\|T_j^b\| < \|T\|$ and $B_j\sigma\sharp e_j \Leftarrow \Pi \succcurlyeq_{\mathcal{D},\mathcal{C}} B_j\sigma\theta\sharp e_j \Leftarrow \Pi'$. Note that this entailment holds thanks to substitution θ, since $\Pi' \models_{\mathcal{C}} \Pi\theta$.

By construction, T' has the same size and structure as T, as desired.

2. A is an equation: In this case, $A : t == s$ and $A' : t' == s'$ with $t' = t\theta$, $s' = s\theta$. Moreover, T must consist of one single node $(t == s)\sharp d \Leftarrow \Pi$ inferred by means of **SQEA**. Therefore, $t \approx_{d,\Pi} s$ holds. This implies $t\theta \approx_{d,\Pi\theta} s\theta$ (i.e. $t' \approx_{d,\Pi\theta} s'$) due to the Substitution Lemma 2.8. From this we conclude $t' \approx_{\Pi'} s'$ due to $d' \trianglelefteq d$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$. Therefore, T' can be built as a proof tree consisting of one single node $(t' == s')\sharp d' \Leftarrow \Pi'$ inferred by means of **SQEA**.
3. A is a primitive atom: In this case, $A : \kappa$ and $A' : \kappa' = \kappa\theta$. Moreover, T must consist of one single node $\kappa\sharp d \Leftarrow \Pi$ inferred by means of **SQPA**. Therefore, $\Pi \models_{\mathcal{C}} \kappa$ holds. This implies $\Pi\theta \models_{\mathcal{C}} \kappa\theta$ due to the Substitution Lemma 2.1. From this we conclude $\Pi' \models_{\mathcal{C}} \kappa'$ due to $\kappa' = \kappa\theta$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$. Therefore, T' can be built as a proof tree consisting of one single node $\kappa'\sharp d' \Leftarrow \Pi'$ inferred by means of **SQPA**. □

The following theorem is the main result in this subsection. It characterizes the least model of a SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program $\mathcal{P}$ w.r.t. the logic SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$):

Theorem 3.2 (*Logical characterization of least program models*)
For any SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program $\mathcal{P}$, its least model can be characterized as:

$$\mathcal{M}_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is a defined observable qc-atom and } \mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi\} \ .$$

Proof
By Theorem 3.1, we already know that $\mathcal{M}_{\mathcal{P}} = \bigcup_{k\in\mathbb{N}} \mathrm{T}_{\mathcal{P}}\uparrow^k(\perp\!\!\!\perp)$. Therefore, it is sufficient to prove that the two implications

1. $\mathcal{P} \vdash^k_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longrightarrow \exists k' : \varphi \in \mathrm{T}_{\mathcal{P}}\uparrow^{k'}(\perp\!\!\!\perp)$
2. $\varphi \in \mathrm{T}_{\mathcal{P}}\uparrow^k(\perp\!\!\!\perp) \Longrightarrow \exists k' : \mathcal{P} \vdash^{k'}_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$

hold for any defined observable qc-atom $\varphi = p(\overline{t}_n)\sharp d \Leftarrow \Pi$ and for any integer value $k \geq 1$. We prove both implications within one single inductive reasoning on k.

Basis ($k = 1$).
— *Implication 1.* Assume $\mathcal{P} \vdash^1_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$. Then, due to the single **SQDA** inference, there must exist some $R_l = (q(\overline{s}_n) \xleftarrow{\alpha}) \in \mathcal{P}$ with empty body, some substitution θ and some $d_0, d_1, \ldots, d_n \in D \setminus \{\mathbf{b}\}$ such that $\mathcal{P} \vdash^0_{\mathcal{S},\mathcal{D},\mathcal{C}} (t_i == s_i\theta)\sharp d_i \Leftarrow \Pi$ for $i = 1 \ldots n$, $\mathcal{S}(p, q) = d_0 \neq \mathbf{b}$, $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha$. Then $\perp\!\!\!\perp \vdash^0_{\mathcal{S},\mathcal{D},\mathcal{C}} (t_i == s_i\theta)\sharp d_i \Leftarrow \Pi$ holds for $i = 1 \ldots n$, because of Lemma 3.1(1). Therefore φ is an immediate consequence of $\perp\!\!\!\perp$ via R_l, which guarantees $\varphi \in \mathrm{T}_{\mathcal{P}}\uparrow^1(\perp\!\!\!\perp)$.
— *Implication 2.* Assume now $\varphi \in \mathrm{T}_{\mathcal{P}}\uparrow^1(\perp\!\!\!\perp)$. Then φ must be an immediate consequence of $\perp\!\!\!\perp$ via some $R_l = (q(\overline{s}_n) \xleftarrow{\alpha}) \in \mathcal{P}$ with empty body. Then there are some substitution θ and some $d_0, d_1, \ldots, d_n \in D \setminus \{\mathbf{b}\}$ such that $\perp\!\!\!\perp \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t_i == s_i\theta)\sharp d_i \Leftarrow \Pi$ for $i = 1 \ldots n$, $\mathcal{S}(p, q) = d_0 \neq \mathbf{b}$, $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha$. Again because of Lemma 3.1(1), we get $\mathcal{P} \vdash^0_{\mathcal{S},\mathcal{D},\mathcal{C}} (t_i == s_i\theta)\sharp d_i \Leftarrow \Pi$ for $i = 1 \ldots n$, which guarantees $\mathcal{P} \vdash^1_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ with one single **SQDA** inference using R_l instantiated by θ.

Inductive step ($k > 1$).

— *Implication 1.* Assume $\mathcal{P} \vdash^{k}_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$. Since the root inference must be **SQDA**, there must exist some program rule $(R_l : q(\overline{s}_n) \overset{\alpha}{\Leftarrow} B_1 \sharp w_1, \ldots, B_m \sharp w_m) \in \mathcal{P}$, some substitution θ and some qualification values $d_0, d_1, \ldots, d_n, e_1, \ldots, e_m \in D \setminus \{\mathbf{b}\}$ such that

- $\mathcal{P} \vdash^{0}_{\mathcal{S},\mathcal{D},\mathcal{C}} \phi_i = ((t_i == s_i\theta)\sharp d_i \Leftarrow \Pi)$ for $i = 1 \ldots n$,
- $\mathcal{P} \vdash^{k_j}_{\mathcal{S},\mathcal{D},\mathcal{C}} \psi_j = (B_j\theta\sharp e_j \Leftarrow \Pi)$ with $e_j \trianglerighteq^{?} w_j$ for $j = 1 \ldots m$, and
- $\mathcal{S}(p,q) = d_0 \neq \mathbf{b}$, $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha \circ e_j$ $(1 \leq j \leq m)$

where $\Sigma_{j=1}^{m} k_j = k - 1$. For each $j = 1 \ldots m$, either ψ_j is defined, and then induction hypothesis yields some k'_j such that $\psi_j \in \mathrm{T}_{\mathcal{P}}{\uparrow}^{k'_j}(\perp\!\!\!\perp)$ and therefore also $\mathrm{T}_{\mathcal{P}}{\uparrow}^{k'_j}(\perp\!\!\!\perp) \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \psi_j$; or else ψ_j is not defined and then $\mathrm{T}_{\mathcal{P}}{\uparrow}^{k'_j}(\perp\!\!\!\perp) \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \psi_j$ for any arbitrarily chosen k'_j, by Lemma 3.1(1). Then $l = max\{k'_j \mid 1 \leq j \leq m\}$ verifies that φ is an immediate consequence of $\mathrm{T}_{\mathcal{P}}{\uparrow}^{l}(\perp\!\!\!\perp)$ via R_l, which implies $\varphi \in \mathrm{T}_{\mathcal{P}}{\uparrow}^{k'}(\perp\!\!\!\perp)$ for $k' = l + 1$.

— *Implication 2.* Assume $\varphi \in \mathrm{T}_{\mathcal{P}}{\uparrow}^{k}(\perp\!\!\!\perp) = \mathrm{T}_{\mathcal{P}}(\mathrm{T}_{\mathcal{P}}{\uparrow}^{k-1}(\perp\!\!\!\perp))$. Then φ is an immediate consequence of $Tp{\uparrow}^{k-1}(\perp\!\!\!\perp)$ via some clause $(R_l : q(\overline{s}_n) \overset{\alpha}{\Leftarrow} B_1 \sharp w_1, \ldots, B_m \sharp w_m) \in \mathcal{P}$. Therefore, there exist some substitution θ and some qualification values $d_0, d_1, \ldots, d_n, e_1, \ldots, e_m \in D \setminus \{\mathbf{b}\}$ such that:

- $\mathrm{T}_{\mathcal{P}}{\uparrow}^{k-1}(\perp\!\!\!\perp) \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \phi_i = ((t_i == s_i\theta)\sharp d_i \Leftarrow \Pi)$ for $i = 1 \ldots n$,
- $\mathrm{T}_{\mathcal{P}}{\uparrow}^{k-1}(\perp\!\!\!\perp) \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \psi_j = (B_j\theta\sharp e_j \Leftarrow \Pi)$ with $e_j \trianglerighteq^{?} w_j$ for $j = 1 \ldots m$, and
- $\mathcal{S}(p,q) = d_0 \neq \mathbf{b}$, $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha \circ e_j$ $(1 \leq j \leq m)$.

For each $i = 1 \ldots n$, Lemma 3.1(1) yields $\mathcal{P} \vdash^{0}_{\mathcal{S},\mathcal{D},\mathcal{C}} \phi_i$. For each $j = 1 \ldots m$, either ψ_j is defined, in which case $\psi_j \in \mathrm{T}_{\mathcal{P}}{\uparrow}^{k-1}(\perp\!\!\!\perp)$, $k - 1 \geq 1$, and induction hypothesis yields some k'_j such that $\mathcal{P} \vdash^{k'_j}_{\mathcal{S},\mathcal{D},\mathcal{C}} \psi_j$; or else ψ_j is not defined, in which case $\mathcal{P} \vdash^{k'_j}_{\mathcal{S},\mathcal{D},\mathcal{C}} \psi_j$ for $k'_j = 0$, by Lemma 3.1(1). In these conditions, $\mathcal{P} \vdash^{k'}_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ holds for $k' = 1 + \Sigma_{j=1}^{m} k'_j$, with a proof tree using a **SQDA** root inference based on R_l instantiated by θ. □

As an easy consequence of the previous theorem we get:

Corollary 3.1 (SQCHL($\mathcal{S},\mathcal{D},\mathcal{C}$) is sound and complete)
For any SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$ and any observable qc-atom φ, the following three statements are equivalent:

$$(a)\ \mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \qquad (b)\ \mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \qquad (c)\ \mathcal{M}_{\mathcal{P}} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$$

Moreover, we also have:

1. *Soundness*: $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longrightarrow \mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$.
2. *Completeness*: $\mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longrightarrow \mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$.

Proof
Soundness and completeness are just a trivial consequence of $(a) \Leftrightarrow (b)$. To finish the proof it suffices to prove the two equivalences $(a) \Leftrightarrow (c)$ and $(b) \Leftrightarrow (c)$. This is done as follows:

$[(a) \Leftrightarrow (c)]$ In the case that φ is a defined qc-atom, $\mathcal{M}_\mathcal{P} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ reduces to $\varphi \in \mathcal{M}_\mathcal{P}$ which is equivalent to $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ by Theorem 3.2. Otherwise, $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \iff \mathcal{M}_\mathcal{P} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ holds because of Lemma 3.1(1).

$[(b) \Rightarrow (c)]$ Assume $\mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ and recall Definition 3.3. Then $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ for every qc-interpretation $\mathcal{I}$ such that $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P}$. In particular, $\mathcal{M}_\mathcal{P} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$, since $\mathcal{M}_\mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P}$ was proved in Theorem 3.1.

$[(c) \Rightarrow (b)]$ Assume $\mathcal{M}_\mathcal{P} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$. In order to obtain $\mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ we must prove:

$$(\star) \quad \mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \text{ holds for any qc-interpretation } \mathcal{I} \text{ such that } \mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P} \ .$$

In the case that φ is a defined qc-atom, $\mathcal{M}_\mathcal{P} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ reduces to $\varphi \in \mathcal{M}_\mathcal{P}$, which implies $(\star)$ because $\mathcal{M}_\mathcal{P}$ is the least model of $\mathcal{P}$, as proved in Theorem 3.1. In the case that φ is not defined but $\mathcal{C}$-based, $(\star)$ follows form the fact that $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ holds for any arbitrary qc-interpretation $\mathcal{I}$, as proved in Lemma 3.1(1). □

We close this subsection with a brief discussion on the relationship between the entailment relation $\succcurlyeq_{\mathcal{D},\mathcal{C}}$ used in this report and a different one that was proposed in (Caballero et al. 2008) and noted $\succcurlyeq_{\mathcal{S},\mathcal{D}}$. In contrast to $\succcurlyeq_{\mathcal{D},\mathcal{C}}$, the entailment $\succcurlyeq_{\mathcal{S},\mathcal{D}}$ depended on a given *similarity* relation $\mathcal{S}$. In the context of the SQCLP scheme, one could think of an entailment $\succcurlyeq_{\mathcal{S},\mathcal{D},\mathcal{C}}$ depending on $\mathcal{S}$ and defined in the following way: given two qc-atoms φ and φ', we could say that φ $(\mathcal{S},\mathcal{D},\mathcal{C})$-entails φ' (in symbols, $\varphi \succcurlyeq_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi'$) iff $\varphi : A\sharp d \Leftarrow \Pi$ and $\varphi' : A'\sharp d' \Leftarrow \Pi'$ such that there is some substitution θ satisfying $\mathcal{S}(A', A\theta) = \lambda \neq \mathbf{b}$, $d' \trianglelefteq \lambda$, $d' \trianglelefteq d$ and $\Pi' \models_\mathcal{C} \Pi\theta$.

However, $\succcurlyeq_{\mathcal{S},\mathcal{D},\mathcal{C}}$ would not work properly in the case that $\mathcal{S}$ is not transitive, as shown by the following simple example: think of a SQCLP$(\mathcal{S},\mathcal{U},\mathcal{R})$-program $\mathcal{P}$ including just a clause

$$R_1 : p_1 \xleftarrow{1.0}$$

and assume that $\mathcal{S}$ verifies $\mathcal{S}(p_1, p_2) = 0.9$, $\mathcal{S}(p_2, p_3) = 0.9$ and $\mathcal{S}(p_1, p_3) = 0.4$ where $p_1, p_2, p_3 \in DP^0$. Then, $\mathcal{P} \vdash_{\mathcal{S},\mathcal{U},\mathcal{R}} p_2\sharp 0.9 \Leftarrow \emptyset$ can be easily proved with the SQCHL rule **SQDA** and $p_2\sharp 0.9 \Leftarrow \emptyset \succcurlyeq_{\mathcal{S},\mathcal{U},\mathcal{R}} p_3\sharp 0.9 \Leftarrow \emptyset$ holds because of $\mathcal{S}(p_2, p_3) = 0.9$, but $\mathcal{P} \vdash_{\mathcal{S},\mathcal{U},\mathcal{R}} p_3\sharp 0.9 \Leftarrow \emptyset$ does not hold. Therefore, the Entailment Property for Programs (Lemma 3.1(2)) would fail if the entailment $\succcurlyeq_{\mathcal{S},\mathcal{D},\mathcal{C}}$ were adopted in place of $\succcurlyeq_{\mathcal{D},\mathcal{C}}$.

Since the Entailment Property for Programs is a very natural condition that must be preserved, we conclude that the entailment relation $\succcurlyeq_{\mathcal{D},\mathcal{C}}$ used in this report is the right choice in a framework where the underlaying proximity relation is not guaranteed to be a similarity.

3.2 Goals and their Solutions

In this brief subsection we present the syntax and declarative semantics of goals in the SQCLP scheme, and we define natural soundness and completeness properties

which are expected to be fulfilled by goal solving devices. These notions are intended as a useful tool to reason about the correctness of SQCLP implementations to be developed in the future.

In order to build goals for SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-programs, we assume a countably infinite set $\mathcal{W}ar$ of so-called *qualification variables* W, disjoint from $\mathcal{V}ar$ and $\mathcal{C}$'s signature Σ. Goals for a given program $\mathcal{P}$ have the form

$$G \ : \ A_1 \sharp W_1, \ \ldots, \ A_m \sharp W_m \ [] \ W_1 \trianglerighteq^? \beta_1, \ \ldots, \ W_m \trianglerighteq^? \beta_m$$

abbreviated as $(A_i \sharp W_i, \ W_i \trianglerighteq^? \beta_i)_{i=1\ldots m}$, where $A_i \sharp W_i$ $(1 \leq i \leq m)$ are atoms annotated with different qualification variables W_i; and $W_i \trianglerighteq^? \beta_i$ are so-called *threshold conditions* with $\beta_i \in (D \setminus \{\mathbf{b}\}) \uplus \{?\}$ $(1 \leq i \leq m)$. The notations ? and $\trianglerighteq^?$ have been already explained in Section 3.1.

In the sequel, the notation war(o) will denote the set of all qualification variables occurring in the syntactic object o. In particular, for a goal G as displayed above, war(G) denotes the set $\{W_i \mid 1 \leq i \leq m\}$. In the case $m = 1$ the goal is called *atomic*. The declarative semantics of goals is provided by their solutions, that are defined as follows:

Definition 3.5 (*Goal Solutions*)
Assume a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program $\mathcal{P}$ and a goal G for the program $\mathcal{P}$ with the syntax displayed above. Then:

1. A *solution* for G is any triple $\langle \sigma, \mu, \Pi \rangle$ such that σ is a $\mathcal{C}$-substitution, μ : war(G) $\rightarrow D \setminus \{\mathbf{b}\}$, Π is a satisfiable and finite set of atomic $\mathcal{C}$-constraints and the following two conditions hold for all $i = 1 \ldots m$:

   (a) $W_i\mu = d_i \trianglerighteq^? \beta_i$ and
   (b) $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} A_i\sigma \sharp W_i\mu \Leftarrow \Pi$.

   The set of all solutions for G is noted $\mathrm{Sol}_{\mathcal{P}}(G)$. Note that solutions are *open* in the sense that the substitution σ is not required to be ground.
2. A solution $\langle \eta, \rho, \Pi \rangle$ for G is called *ground* iff $\Pi = \emptyset$ and $\eta \in \mathrm{Val}_{\mathcal{C}}$ is a variable valuation such that $A_i\eta$ is a ground atom for all $i = 1 \ldots m$. The set of all ground solutions for G is noted $\mathrm{GSol}_{\mathcal{P}}(G)$. Obviously, $\mathrm{GSol}_{\mathcal{P}}(G) \subseteq \mathrm{Sol}_{\mathcal{P}}(G)$.
3. A ground solution $\langle \eta, \rho, \emptyset \rangle \in \mathrm{GSol}_{\mathcal{P}}(G)$ is *subsumed* by $\langle \sigma, \mu, \Pi \rangle$ iff there is some $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ s.t. $\eta =_{\mathrm{var}(G)} \sigma\nu$ and $W_i\rho \trianglelefteq W_i\mu$ for $i = 1 \ldots m$. $\square$

Implicitly, the first item in the previous definition requires $A_i\sigma \sharp W_i\mu \Leftarrow \Pi$ to be observable qc-atoms in the sense of Definition 3.1, which is trivially true because $W_i\mu = d_i \in D \setminus \{\mathbf{b}\}$ and Π is satisfiable. In fact, Definition 3.1 was designed with the aim of using observable qc-atoms as observations of valid open solutions for atomic goals. The next example illustrates the definition:

Example 3.6 (*Solutions for an atomic goals*)

1. G : `goodWork(X)#W [] W |> (0.55,30)` is a goal for the program fragment $\mathcal{P}$ shown in Figure 1, and the arguments given near the beginning of Subsection 3.1 can be formalized to prove that $\langle \{\mathtt{X} \mapsto \mathtt{king\_liar}\}, \{\mathtt{W} \mapsto \mathtt{(0.6,5)}\}, \emptyset \rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$.

2. As an additional example involving constraints, recall the SQCLP($\mathcal{S},\mathcal{U},\mathcal{R}$)-program $\mathcal{P}$ presented in Example 3.1. An atomic goal G for this program is $p'(c'(Y),Z)\sharp W \,[\!]\, W \geq^{?} 0.75$. Consider $\sigma = \{Z \mapsto c(X)\}$, $\mu = \{W \mapsto 0.8\}$ and $\Pi = \{cp_{>}(X, 1.0), op_{+}(A, A,\ X), op_{\times}(2.0, A, Y)\}$. Note that $0.8 \geq 0.75$ and $\mathcal{P} \vdash_{\mathcal{S},\mathcal{U},\mathcal{R}} p'(c'(Y),Z)\sigma\sharp W\mu \Leftarrow \Pi$, as we have seen in Example 3.5. Therefore, the requirements of Definition 3.5 are fulfilled, and $\langle\sigma,\mu,\Pi\rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$. □

In practice, users of SQCLP languages will rely on some available *goal solving system* for computing goal solutions. The following definition specifies two important properties of goal solving systems:

Definition 3.6 (*Correct Goal Solving Systems*)
At a high abstraction level, a *goal solving system* for SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$) can be thought as a device that takes a program $\mathcal{P}$ and a goal G as input and yields various triples $\langle\sigma,\mu,\Pi\rangle$, called *computed answers*, as outputs. Such a goal solving system is called:

1. *Sound* iff every computed answer is a solution $\langle\sigma,\mu,\Pi\rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$.
2. *Weakly complete* iff every ground solution $\langle\eta,\rho,\emptyset\rangle \in \mathrm{GSol}_{\mathcal{P}}(G)$ is subssumed by some computed answer.
3. *Correct* iff it is both sound and weakly complete. □

Every goal solving system for a SQCLP instance should be sound and ideally also weakly complete. Implementing such systems is one of the major lines of future research mentioned in the Conclusions of this report.

4 Conclusions

We have extended the classical CLP scheme to a new scheme SQCLP whose instances SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$) are parameterized by a proximity relation $\mathcal{S}$, a qualification domain $\mathcal{D}$ and a constraint domain $\mathcal{C}$. In addition to the known features of CLP programming, the new scheme offers extra facilities for dealing with expert knowledge representation and flexible query answering. Inspired by the observable CLP semantics in (Gabbrielli and Levi 1991; Gabbrielli et al. 1995), we have presented a declarative semantics for SQCLP that provides fixpoint and proof-theoretical characterizations of least program models as well as an implementation-independent notion of goal solutions.

SQCLP is a quite general scheme. Different partial instantiations of its three parameters lead to more particular schemes, most of which can be placed in close correspondence to previous proposals. The items below present seven particularizations, along with some comments which make use of the notions *threshold-free*, *attenuation-free* and *constraint-free* which have been explained at the beginning of Section 3.1.

1. By definition, QCLP has instances $\mathrm{QCLP}(\mathcal{D},\mathcal{C}) =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S}_{\mathrm{id}},\mathcal{D},\mathcal{C})$, where $\mathcal{S}_{\mathrm{id}}$ is the *identity* proximity relation. The *quantitative* CLP scheme proposed in (Riezler 1998) can be understood as a further particularization of QCLP that works with threshold-free QCLP($\mathcal{U},\mathcal{C}$) programs, where $\mathcal{U}$ is the qualification domain of uncertainty values (see Subsection 2.2.2).

2. By definition, SQLP has instances $\mathrm{SQLP}(\mathcal{S},\mathcal{D}) =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S},\mathcal{D},\mathcal{R})$, where $\mathcal{R}$ is the real constraint domain (see Subsection 2.1.2). The scheme with the same name originally proposed in (Caballero et al. 2008) can be understood as a restricted form of the present formulation; it worked with threshold-free and constraint-free $\mathrm{SQLP}(\mathcal{S},\mathcal{D})$ programs and it restricted the choice of the $\mathcal{S}$ parameter to transitive proximity (i.e. similarity) relations.
3. By definition, SCLP[3] has instances $\mathrm{SCLP}(\mathcal{S},\mathcal{C}) =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S},\mathcal{B},\mathcal{C})$, where $\mathcal{B}$ is the qualification domain of classical boolean values (see Subsection 2.2.1). Due to the fixed parameter choice $\mathcal{D} = \mathcal{B}$, both attenuation values and threshold values become useless, and each choice of $\mathcal{S}$ must necessarily represent a crisp reflexive and symmetric relation. Therefore, this new scheme is not so interesting from the viewpoint of uncertain and qualified reasoning.
4. By definition, QLP has instances $\mathrm{QLP}(\mathcal{D}) =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S}_{\mathrm{id}},\mathcal{D},\mathcal{R})$. The scheme with the same name originally proposed in (Rodríguez-Artalejo and Romero-Díaz 2008b) can be understood as a restricted form of the present formulation; it worked with threshold-free and constraint-free $\mathrm{QLP}(\mathcal{D})$ programs.
5. By definition, SLP has instances $\mathrm{SLP}(\mathcal{S}) =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S},\mathcal{U},\mathcal{R})$. The pure fragment of Bousi~Prolog (Julián-Iranzo and Rubio-Manzano 2009a) can be understood as a restricted form of SLP in the present formulation; it works with threshold-free, attenuation-free and constraint-free $\mathrm{SLP}(\mathcal{S})$ programs. Moreover, restricting the choice of $\mathcal{S}$ to similarity relations leads to SLP in the sense of (Sessa 2002) and related papers.
6. The CLP scheme can be defined by instances $\mathrm{CLP}(\mathcal{C}) =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S}_{\mathrm{id}},\mathcal{B},\mathcal{C})$. Both attenuation values and threshold values are useless in CLP programs, due to the fixed parameter choice $\mathcal{D} = \mathcal{B}$.
7. Finally, the pure LP paradigm can be defined as $\mathrm{LP} =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S}_{\mathrm{id}},\mathcal{B},\mathcal{H})$, where $\mathcal{H}$ is the *Herbrand* constraint domain. Again, attenuation values and threshold values are useless in LP due to the fixed parameter choice $\mathcal{D} = \mathcal{B}$.

In all the previous items, the schemes obtained by partial instantiation inherit the declarative semantics from SQCLP, using sets of observables of the form $A\sharp d \Leftarrow \Pi$ as interpretations. A similar semantic approach were used in our previous papers (Rodríguez-Artalejo and Romero-Díaz 2008b; Caballero et al. 2008), except that Π and equations were absent due to the lack of CLP features. The other related works discussed in the Introduction view program interpretations as mappings $\mathcal{I}$ from the ground Herbrand base into some set of lattice elements (the real interval $[0,1]$ in many cases), as already discussed in the explanations following Definition 3.2.

As seen in Subsection 3.2, SQCLP's semantics enables a declarative characterization of valid goal solutions. This fact is relevant for modeling the expected behavior of goal solving devices and reasoning about their correctness. Moreover, the relations $\approx_{\lambda,\Pi}$ introduced for the first time in the present paper (see Definition 2.16) allow to specify the semantic role of $\mathcal{S}$ in a constraint-based framework, with less technical overhead than in previous related approaches.

[3] Not to be confused with SCLP in the sense of (Bistarelli et al. 2001), discussed below.

A related work not mentioned in items 1–7 above is the semiring-based CLP of (Bistarelli et al. 2001), a scheme with instances SCLP(S) parameterized by a semiring $\mathrm{S} = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ whose elements are used to represent consistency levels in soft constraint solving. The semirings used in this approach can be equipped with a lattice structure whose *lub* operation is always $+$, but whose *glb* operation may be different from $\times$. On the other hand, our qualification domains are defined as lattices with an additional attenuation operation $\circ$. It turns out that the kind of semirings used in SCLP(S) correspond to qualification domains only in some cases. Moreover, $\times$ is used in SCLP(S) to interpret logical conjunction in clause bodies and goals, while the *glb* operation is used in SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) for the same purpose. For this reason, even if $\mathcal{D}$ is "equivalent" to S, SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) cannot be naturally used to express SCLP(S) in the case that $\times$ is not the *glb*. Assuming that $\mathcal{D}$ is "equivalent" to S and that $\times$ behaves as the *glb* in S, program clauses in SCLP(S) can be viewed as a particular case of program clauses in SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) which use an attenuation factor different from $\mathbf{t}$ only for facts. Other relevant differences between SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) and SCLP(S) can be explained by comparing the parameters. As said before $\mathcal{D}$ may be "equivalent" to S in some cases, but $\mathcal{S}$ is absent and $\mathcal{C}$ is not made explicit in SCLP(S). Seemingly, the intended use of SCLP(S) is related to finite domain constraints and no parametrically given constraint domain is provided.

In the future we plan to implement some SQCLP instances by extending the semantically correct program transformation techniques from (Caballero et al. 2008), and to investigate applications which can profit from flexible query answering. Other interesting lines of future work include: a) extension of the qualified SLD resolution presented in (Rodríguez-Artalejo and Romero-Díaz 2008b) to a SQCLP goal solving procedure able to work with constraints and a proximity relation; and b) extension of the QCFLP scheme in (Caballero et al. 2009) to work with a proximity relation and higher-order functions.

Acknowledgements

This report is a widely extended version of (Rodríguez-Artalejo and Romero-Díaz 2010). The authors are thankful to the anonymous referees of (Rodríguez-Artalejo and Romero-Díaz 2010) for constructive remarks and suggestions which helped to improve the presentation. They are also thankful to Rafael Caballero for useful discussions on the report's topics and to Jesús Almendros for pointing to bibliographic references in the area of flexible query answering.

References

Apt, K. R. 1990. Logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B: Formal Models and Semantics. Elsevier and The MIT Press, 493–574.

Apt, K. R. and Gabbrielli, M. 1994. Declarative interpretations reconsidered. In *Proceedings of the 11th International Conference on Logic Programming (ICLP'94)*, P. van Hentenryck, Ed. The MIT Press, 74–89.

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

Arcelli, F. and Formato, F. 1999. Likelog: a logic programming language for flexible data retrieval. In *Proceedings of the 1999 ACM Symposium on Applied computing (SAC'99).* ACM Press, New York, NY, USA, 260–267.

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

Baader, F. and Nipkow, T. 1998. *Term Rewriting and All That.* Cambridge University Press.

Bistarelli, S., Montanari, U., and Rossi, F. 2001. Semiring-based constraint logic programming: Syntax and semantics. *ACM Transactions on Programming Languages and Systems 3,* 1 (January), 1–29.

Bossi, A., Gabbrielli, M., Levi, G., and Martelli, M. 1994. The s-semantics approach: Theory and applications. *Journal of Logic Programming 19/20*, 149–197.

Caballero, R., Rodríguez-Artalejo, M., and Romero-Díaz, C. A. 2008. 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.* ACM, Valencia, Spain, 185–194.

Caballero, R., Rodríguez-Artalejo, M., and Romero-Díaz, C. A. 2009. Qualified computations in functional logic programming. In *Logic Programming (ICLP'09)*, P. Hill and D. Warren, Eds. LNCS, vol. 5649. Springer-Verlag Berlin Heidelberg, Pasadena, CA, USA, 449–463.

Clark, K. L. 1979. Predicate logic as a computational formalism (res. report doc 79/59). Tech. rep., Imperial College, Dept. of Computing, London.

Dubois, D. and Prade, H. 1980. *Fuzzy Sets and Systems: Theory and Applications.* Academic Press, New York, NY, USA.

Estévez-Martín, S., Hortalá-González, T., Rodríguez-Artalejo, M., del Vado Vírseda, R., Sáenz-Pérez, F., and Fernández, A. J. 2009. On the cooperation of the constraint domains $\mathcal{H}$, $\mathcal{R}$ and $\mathcal{FD}$ in $cflp$. *Theory and Practice of Logic Programming 9,* 4, 415–527.

Falaschi, M., Levi, G., Martelli, M., and Palamidessi, C. 1993. A model-theoretic reconstruction of the operational semantics of logic programs. *Information and Computation 102,* 1, 86–113.

Falaschi, M., Levi, G., Palamidessi, C., and Martelli, M. 1989. Declarative modeling of the operational behavior of logic languages. *Theoretical Computer Science 69,* 3 (December), 289–318.

Freuder, E. C. and Wallace, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence 58,* 1–3, 21–70.

Gabbrielli, M., Dore, G. M., and Levi, G. 1995. Observable semantics for constraint logic programs. *Journal of Logic and Computation 5,* 2, 133–171.

Gabbrielli, M. and Levi, G. 1991. Modeling answer constraints in constraint logic programs. In *Proceedings of the 8th International Conference on Logic Programming (ICLP'91).* The MIT Press, 238–252.

Georget, Y. and Codognet, P. 1998. Compiling semiring-based constraints with CLP(FD,S). In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming.* LNCS, vol. 1520. Springer-Verlag, 205–219.

Guadarrama, S., Muñoz, S., and Vaucheret, C. 2004. Fuzzy prolog: A new approach using soft constraint propagation. *Fuzzy Sets and Systems 144,* 1, 127–150.

Hájek, P. 1998. *Metamathematics of Fuzzy Logic.* Dordrecht: Kluwer.

Hanus, Ed., M. Curry: An integrated functional logic language (vers. 0.8.2, 2006); `http://www.curry-language.org`.

Höhfeld, M. and Smolka, G. 1988. Definite relations over constraint languages. Tech. Rep. LILOG Report 53, IBM Deutschland.

Jaffar, J. and Lassez, J. L. 1987. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages (POPL'87)*. ACM New York, NY, USA, Munich, West Germany, 111–119.

Jaffar, J. and Maher, M. 1994. Constraint logic programming: a survey. *Journal of Logic Programming 19&20*, 503–581.

Jaffar, J., Maher, M., Marriott, K., and Stuckey, P. J. 1998. Semantics of constraints logic programs. *Journal of Logic Programming 37,* 1-3, 1–46.

Jaffar, J., Michaylov, S., Stuckey, P. J., and Yap, R. H. C. 1992. The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems 14(3)*, 339–395.

Julián-Iranzo, P., Rubio, C., and Gallardo, J. 2009. Bousi∼Prolog: a prolog extension language for flexible query answering. In *Proceedings of the Eighth Spanish Conference on Programming and Computer Languages (PROLE 2008)*, J. M. Almendros-Jiménez, Ed. ENTCS, vol. 248. Elsevier, Gijón, Spain, 131–147.

Julián-Iranzo, P. and Rubio-Manzano, C. 2009a. A declarative semantics for Bousi∼Prolog. In *PPDP'09: Proceedings of the 11th ACM SIGPLAN conference on Principles and practice of declarative programming.* ACM, Coimbra, Portugal, 149–160.

Julián-Iranzo, P. and Rubio-Manzano, C. 2009b. A similarity-based WAM for Bousi∼Prolog. In *Bio-Inspired Systems: Computational and Ambient Intelligence (IWANN 2009)*. LNCS, vol. 5517. Springer Berlin / Heidelberg, Salamanca, Spain, 245–252.

Kifer, M. and Subrahmanian, V. S. 1992. Theory of generalized annotated logic programs and their applications. *Journal of Logic Programming 12,* 3&4, 335–367.

Krajči, S., Lencses, R., and Vojtáš, P. 2004. A comparison of fuzzy and annotated logic programming. *Fuzzy Sets and Systems 144*, 173–192.

Lloyd, J. W. 1987. *Foundations of Logic Programming, Second Edition.* Springer.

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

López-Fraguas, F. J., Rodríguez-Artalejo, M., and del Vado-Vírseda, R. 2007. A new generic scheme for functional logic programming with constraints. *Journal of Higher-Order and Symbolic Computation 20,* 1&2, 73–122.

Lucio, P., Orejas, F., Pasarella, E., and Pino, E. 2008. A functorial framework for constraint normal logic programming. *Applied Categoriacal Structures 16,* 3, 421–450.

Medina, J., Ojeda-Aciego, M., and Vojtáš, P. 2001a. Multi-adjoint logic programming with continuous semantics. In *Logic Programming and Non-Monotonic Reasoning (LPNMR'01)*, T. Eiter, W. Faber, and M. Truszczyinski, Eds. LNAI, vol. 2173. Springer-Verlag, 351–364.

Medina, J., Ojeda-Aciego, M., and Vojtáš, P. 2001b. A procedural semantics for multi-adjoint logic programming. In *Progress in Artificial Intelligence (EPIA'01)*, P. Brazdil and A. Jorge, Eds. LNAI, vol. 2258. Springer-Verlag, 290–297.

Moreno, G. and Pascual, V. 2007. Formal properties of needed narrowing with similarity relations. *Electronic Notes in Theoretical Computer Science 188*, 21–35.

Ng, R. T. and Subrahmanian, V. S. 1992. Probabilistic logic programming. *Information and Computation 101,* 2, 150–201.

Riezler, S. 1996. Quantitative constraint logic programming for weighted grammar applications. In *Proceedings of the Logical Aspects of Computational Linguistics (LACL'96)*, C. Retoré, Ed. LNCS, vol. 1328. Springer-Verlag, 346–365.

Riezler, S. 1998. Probabilistic constraint logic programming. Ph.D. thesis, Neuphilologischen Fakultät del Universität Tübingen.

Rodríguez-Artalejo, M. and Romero-Díaz, C. A. 2008a. A generic scheme for qualified logic programming. Tech. Rep. SIC-1-08 (CoRR abs/1008.3863), Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain.

Rodríguez-Artalejo, M. and Romero-Díaz, C. A. 2008b. Quantitative logic programming revisited. In *Functional and Logic Programming (FLOPS'08)*, J. Garrigue and M. Hermenegildo, Eds. LNCS, vol. 4989. Springer-Verlag, Ise, Japan, 272–288.

Rodríguez-Artalejo, M. and Romero-Díaz, C. A. 2009. Qualified logic programming with bivalued predicates. In *Proceedings of the Eighth Spanish Conference on Programming and Computer Languages (PROLE 2008)*, J. M. Almendros-Jiménez, Ed. ENTCS, vol. 248. Elsevier, Gijón, Spain, 67–82.

Rodríguez-Artalejo, M. and Romero-Díaz, C. A. 2010. A declarative semantics for CLP with qualification and proximity. *Theory and Practice of Logic Programming, 26th Int'l. Conference on Logic Programming (ICLP'10) Special Issue 10,* 4–6, 627–642.

Saraswat, V. A. 1992. The category of constraint systems is cartesian-closed. In *Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science (LICS '92)*. 341–345.

Sessa, M. I. 2002. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science 275,* 1-2, 389–426.

Shenoi, S. and Melton, A. 1999. Proximity relations in the fuzzy relational database model. *Fuzzy Sets and Systems 100,* supl., 51–62.

Tarski, A. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics 5,* 2, 285–309.

van Emden, M. H. 1986. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming 3,* 1, 37–53.

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

Vaucheret, C., Guadarrama, S., and Muñoz, S. 2002. Fuzzy prolog: A simple general implementation using CLP($\mathcal{R}$). In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'02)*, M. Baaz and A. Voronkov, Eds. LNCS, vol. 2514. Springer Berlin / Heidelberg, Tbilisi, Georgia, 450–463.

Vojtáš, P. 2001. Fuzzy logic programming. *Fuzzy Sets and Systems 124*, 361–370.

Zadeh, L. A. 1965. Fuzzy sets. *Information and Control 8,* 3, 338–353.

Zadeh, L. A. 1971. Similarity relations and fuzzy orderings. *Information Sciences 3,* 2, 177–200.