Qualified Computations in Functional Logic Programming*

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

Departamento de Sistemas Informáticos y Computación, Universidad Complutense
Facultad de Informática, 28040 Madrid, Spain
{rafa,mario}@sip.ucm.es and cromdia@fdi.ucm.es

Abstract. Qualification has been recently introduced as a generalization of uncertainty in the field of Logic Programming. In this paper we investigate a more expressive language for First-Order Functional Logic Programming with Constraints and Qualification. We present a Rewriting Logic which characterizes the intended semantics of programs, and a prototype implementation based on a semantically correct program transformation. Potential applications of the resulting language include flexible information retrieval. As a concrete illustration, we show how to write program rules to compute qualified answers for user queries concerning the books available in a given library.

Keywords: Constraints, Functional Logic Programming, Program Transformation, Qualification, Rewriting Logic.

1 Introduction

Various extensions of Logic Programming with uncertain reasoning capabilities have been widely investigated during the last 25 years. The recent recollection [16] reviews the evolution of the subject from the viewpoint of a committed researcher. All the proposals in the field replace classical two-valued logic by some kind of many-valued logic with more than two truth values, which are attached to computed answers and interpreted as truth degrees.

In a recent paper [14] we have presented a *Qualified Logic Programming* scheme QLP($\mathcal{D}$) parameterized by a *qualification domain* $\mathcal{D}$, a lattice of so-called *qualification values* that are attached to computed answers and interpreted as a measure of the satisfaction of certain user's expectations. QLP($\mathcal{D}$)-programs are sets of clauses of the form $A \xleftarrow{\alpha} \overline{B}$, where the head A is an atom, the body $\overline{B}$ is a conjunction of atoms, and $\alpha \in \mathcal{D}$ is called *attenuation* factor. Intuitively, α measures the maximum confidence placed on an inference performed by the clause. More precisely, any successful application of the clause attaches to the head a qualification value which cannot exceed the infimum of $\alpha \circ \beta_i \in \mathcal{D}$, where β_i are the qualification values computed for the body atoms and $\circ$ is a so-called *attenuation operator*, provided by $\mathcal{D}$.

* Research partially supported by projects MERIT–FORMS (TIN2005-09027-C03-03), PROMESAS–CAM(S-0505/TIC/0407) and STAMP (TIN2008-06622-C03-01).

Uncertain Logic Programming can be expressed by particular instances of QLP($\mathcal{D}$), where the user's expectation is understood as a lower bound for the *truth degree* of the computed answer and $\mathcal{D}$ is chosen to formalize a lattice of non-classical truth values. Other choices of $\mathcal{D}$ can be designed to model other kinds of user expectations, as e.g. an upper bound for the *size* of the logical proof underlying the computed answer. As shown in [3], the QLP($\mathcal{D}$) scheme is also well suited to deal with Uncertain Logic Programming based on similarity relations in the line of [15]. Therefore, Qualified Logic Programming has a potential for flexible information retrieval applications, where the answers computed for user's queries may match the user's expectations only to some degree. As shown in [14], several useful instances of QLP($\mathcal{D}$) can be conveniently implemented by using constraint solving techniques.

In this paper we investigate an extension of QLP($\mathcal{D}$) to a more expressive scheme, supporting computation with first-order lazy functions and constraints. More precisely, we consider the first-order fragment of CFLP($\mathcal{C}$), a generic scheme for functional logic programming with constraints over a parametrically given domain $\mathcal{C}$ presented in [9]. We propose an extended scheme QCFLP($\mathcal{D}, \mathcal{C}$) where the additional parameter $\mathcal{D}$ stands for a qualification domain. QCFLP($\mathcal{D}, \mathcal{C}$)-programs are sets of conditional rewrite rules of the form $f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \Delta$, where the condition Δ is a conjunction of $\mathcal{C}$-constraints that may involve user defined functions, and $\alpha \in \mathcal{D}$ is an attenuation factor. As in the logic programming case, α measures the maximum confidence placed on an inference performed by the rule: any successful application of the rule attaches to the computed result a qualification value which cannot exceed the infimum of $\alpha \circ \beta_i \in \mathcal{D}$, where β_i are the qualification values computed for r and Δ, and $\circ$ is $\mathcal{D}$'s attenuation operator. QLP($\mathcal{D}$) program clauses can be easily formulated as a particular case of QCFLP($\mathcal{D}, \mathcal{C}$) program rules.

As far as we know, no related work covers the expressivity of our approach. Guadarrama et al. [6] have proposed to use real arithmetic constraints as an implementation tool for a Fuzzy Prolog, but their language does not support constraint programming as such. Starting from the field of natural language processing, Riezler [11,12] has developed quantitative and probabilistic extensions of the classical CLP($\mathcal{C}$) scheme with the aim of computing good parse trees for constraint logic grammars, but his work bears no relation to functional programming. Moreno and Pascual [10] have investigated similarity-based unification in the context of *needed narrowing* [1], a narrowing strategy using so-called *definitional trees* that underlies the operational semantics of functional logic languages such as Curry [7] and $\mathcal{TOY}$ [2], but they use neither constraints nor attenuation factors and they provide no declarative semantics.

Figure 1 below shows a small set of QCFLP($\mathcal{U}, \mathcal{R}$) program rules, called the *library program* in the rest of the paper. The concrete syntax is inspired by the functional logic language $\mathcal{TOY}$, but the ideas and results of this paper could be applied also to Curry and other similar languages. In this example, $\mathcal{U}$ stands for a particular qualification domain which supports uncertain truth values in the real interval $[0, 1]$, while $\mathcal{R}$ stands for a particular constraint domain which

```
%% Data types:
type pages, id = int
type title, author, language, genre = [char]
data vocabularyLevel = easy | medium | difficult
data readerLevel = basic | intermediate | upper | proficiency
data book = book(id, title, author, language, genre, vocabularyLevel, pages)

%% Simple library, represented as list of books:
library :: [book]
library --> [ book(1, "Tintin", "Herge", "French", "Comic", easy, 65),
              book(2, "Dune", "F. P. Herbert", "English", "SciFi", medium, 345),
              book(3, "Kritik der reinen Vernunft", "Immanuel Kant", "German",
                   "Philosophy", difficult, 1011),
              book(4, "Beim Hauten der Zwiebel", "Gunter Grass", "German",
                   "Biography", medium, 432) ]

%% Auxiliary function for computing list membership:
member(B,[]) --> false
member(B,H:_T) --> true <== B == H
member(B,H:T) --> member(B,T) <== B /= H

%% Functions for getting the explicit attributes of a given book:
getId(book(Id,_Title,_Author,_Lang,_Genre,_VocLvl,_Pages)) --> Id
getTitle(book(_Id,Title,_Author,_Lang,_Genre,_VocLvl,_Pages)) --> Title
getAuthor(book(_Id,_Title,Author,_Lang,_Genre,_VocLvl,_Pages)) --> Author
getLanguage(book(_Id,_Title,_Author,Lang,_Genre,_VocLvl,_Pages)) --> Lang
getGenre(book(_Id,_Title,_Author,_Lang,Genre,_VocLvl,_Pages)) --> Genre
getVocabularyLevel(book(_Id,_Title,_Author,_Lang,_Genre,VocLvl,_Pages)) --> VocLvl
getPages(book(_Id,_Title,_Author,_Lang,_Genre,_VocLvl,Pages)) --> Pages

%% Function for guessing the genre of a given book:
guessGenre(B) --> getGenre(B)
guessGenre(B) -0.9-> "Fantasy" <== guessGenre(B) == "SciFi"
guessGenre(B) -0.8-> "Essay" <== guessGenre(B) == "Philosophy"
guessGenre(B) -0.7-> "Essay" <== guessGenre(B) == "Biography"
guessGenre(B) -0.7-> "Adventure" <== guessGenre(B) == "Fantasy"

%% Function for guessing the reader level of a given book:
guessReaderLevel(B) --> basic <== getVocabularyLevel(B) == easy, getPages(B) < 50
guessReaderLevel(B) -0.8-> intermediate <== getVocabularyLevel(B) == easy, getPages(B) >= 50
guessReaderLevel(B) -0.9-> basic <== guessGenre(B) == "Children"
guessReaderLevel(B) -0.9-> proficiency <== getVocabularyLevel(B) == difficult,
                                           getPages(B) >= 200
guessReaderLevel(B) -0.8-> upper <== getVocabularyLevel(B) == difficult, getPages(B) < 200
guessReaderLevel(B) -0.8-> intermediate <== getVocabularyLevel(B) == medium
guessReaderLevel(B) -0.7-> upper <== getVocabularyLevel(B) == medium

%% Function for answering a particular kind of user's query:
search(Language,Genre,Level) --> getId(B) <== member(B,library),
                                              getLanguage(B) == Language,
                                              guessReaderLevel(B) == Level,
                                              guessGenre(B) == Genre
```

Fig. 1. Library with books in different languages

supports arithmetic constraints over the real numbers; see Section 2 for more details.

The program rules are intended to encode expert knowledge for computing qualified answers to user's queries concerning the books available in a simplified library, represented as a list of objects of type `book`. The various `get` functions extract the explicit values of book attributes. Functions `guessGenre` and `guessReaderLevel` perform qualified inferences relying on analogies between different genres and heuristic rules to estimate reader levels on the basis of other features of a given book, respectively. For instance, the second rule for `guessGenre` infers the genre `"Fantasy"` with attenuation `0.9` for a book `B` whose genre is already known to be `"SciFi"`. Some program rules, as e.g. those of the auxiliary function `member`, have attached no explicit attenuation factor. By convention, this is understood as the implicit attachment of the attenuation factor `1.0`, the top value of $\mathcal{U}$. For any instance of the QCFLP($\mathcal{D}, \mathcal{C}$) scheme, a similar convention allows to view CFLP($\mathcal{C}$)-program rules as QCFLP($\mathcal{D}, \mathcal{C}$)-program rules whose attached qualification is optimal.

The last rule for function `search` encodes a method for computing qualified answers to a particular kind of user's queries. Therefore, the queries can be formulated as goals to be solved by the program fragment. For instance, answering the query of a user who wants to find a book of genre `"Essay"`, language `"German"` and user level `intermediate` with a certainty degree of at least 0.65 can be formulated as the goal:

```
(search("German","Essay",intermediate) == R) # W | W >= 0.65
```

The techniques presented in Section 4 can be used to translate the QCFLP($\mathcal{U}, \mathcal{R}$) program rules and goal into the CFLP($\mathcal{R}$) language, which is implemented in the $\mathcal{TOY}$ system. Solving the translated goal in $\mathcal{TOY}$ computes the answer $\{R \mapsto 4\}\{0.65 \leq W, W \leq 0.7\}$, ensuring that the library book with `id` 4 satisfies the query's requirements with any certainty degree in the interval [0.65,0.7], in particular 0.7. The computation uses the 4th program rule of `guessGenre` to obtain `"Essay"` as the book's genre with qualification 0.7, and the 6th program rule of `guessReaderLevel` to obtain `intermediate` as the reader level with qualification 0.8.

The rest of the paper is organized as follows. In Section 2 we recall known proposals concerning qualification and constraint domains, and we introduce a technical notion needed to relate both kinds of domains for the purposes of this paper. In Section 3 we present the generic scheme QCFLP($\mathcal{D}, \mathcal{C}$) announced in this introduction, and we formalize a special Rewriting Logic which characterizes the declarative semantics of QCFLP($\mathcal{D}, \mathcal{C}$)-programs. In Section 4 we present a semantically correct program transformation converting QCFLP($\mathcal{D}, \mathcal{C}$) programs and goals into the qualification-free CFLP($\mathcal{C}$) programming scheme, which is supported by existing systems such as $\mathcal{TOY}$. Section 5 concludes and points to some lines of planned future work. The Technical Report [4] includes full proofs of the main results in this paper, as well as some additional results concerning alternative characterizations of program semantics.

2 Qualification and Constraint Domains

A *Qualification Domain* is any algebraic structure $\mathcal{D} = \langle D, \trianglelefteq, \mathbf{b}, \mathbf{t}, \circ \rangle$ such that D is a set of elements called *qualification values*, $\langle D, \trianglelefteq, \mathbf{b}, \mathbf{t} \rangle$ is a lattice with extreme points $\mathbf{b}$ and $\mathbf{t}$ w.r.t. the partial ordering $\trianglelefteq$ and $\circ : D \times D \to D$ is a so-called *attenuation operation* satisfying the axioms stated in [14]. When convenient, D will be also noted as $D_{\mathcal{D}}$.

The intended use of qualification domains has been explained in the introduction. The examples in this paper will use a particular qualification domain $\mathcal{U}$ whose values represent certainty degrees in the sense of fuzzy logic. Formally, $\mathcal{U} = \langle U, \leq, 0, 1, \times \rangle$, where $U = [0,1] = \{d \in \mathbb{R} \mid 0 \leq d \leq 1\}$, $\leq$ is the usual numerical ordering, and $\times$ is the multiplication operation. In this domain, the bottom and top elements are $\mathbf{b} = 0$ and $\mathbf{t} = 1$, and the infimum of a finite $S \subseteq U$ is the minimum value $\min(S)$, understood as 1 if $S = \emptyset$. The reader is referred to [14] for other useful instances of qualification domains.

Constraint domains are used in Constraint Logic Programming and its extensions as a tool to provide data values, primitive operations and constraints tailored to domain-oriented applications. Various formalizations of this notion are known. In this paper, constraint domains are related to signatures of the form $\Sigma = \langle DC, PF, DF \rangle$ where $DC = \bigcup_{n \in \mathbb{N}} DC^n$, $PF = \bigcup_{n \in \mathbb{N}} PF^n$ and $DF = \bigcup_{n \in \mathbb{N}} DF^n$ are mutually disjoint sets of *data constructor* symbols, *primitive function* symbols and *defined function* symbols, respectively, ranked by arities. Given a signature Σ, a symbol $\bot$ to note the *undefined value*, a set B of *basic values* u and a countably infinite set $\mathcal{V}ar$ of variables X, we define the notions listed below, where $\overline{o}_n$ abbreviates the n-tuple of syntactic objects $o_1, \ldots, o_n$.

- *Expressions* $e \in \mathrm{Exp}_\bot(\Sigma, B, \mathcal{V}ar)$ have the syntax $e ::= \bot | X | u | h(\overline{e}_n)$, where $h \in DC^n \cup PF^n \cup DF^n$. In the case $n = 0$, $h(\overline{e}_n)$ is written simply as h.
- *Constructor Terms* $t \in \mathrm{Term}_\bot(\Sigma, B, \mathcal{V}ar)$ have the syntax $e ::= \bot | X | u | c(\overline{t}_n)$, where $c \in DC^n$. They will be called just terms in the sequel.
- *Total Expressions* $e \in \mathrm{Exp}(\Sigma, B, \mathcal{V}ar)$ and *Total Terms* $t \in \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ have a similar syntax, with the $\bot$ case omitted.
- An expression or term (total or not) is called *ground* iff it includes no occurrences of variables. $\mathrm{Exp}_\bot(\Sigma, B)$ stands for the set of all ground expressions. The notations $\mathrm{Term}_\bot(\Sigma, B)$, $\mathrm{Exp}(\Sigma, B)$ and $\mathrm{Term}(\Sigma, B)$ have a similar meaning.
- We note as $\sqsubseteq$ the *information ordering*, defined as the least partial ordering over $\mathrm{Exp}_\bot(\Sigma, B, \mathcal{V}ar)$ compatible with contexts and verifying $\bot \sqsubseteq e$ for all $e \in \mathrm{Exp}_\bot(\Sigma, B, \mathcal{V}ar)$.
- Substitutions are defined as mappings $\sigma : \mathcal{V}ar \to \mathrm{Term}_\bot(\Sigma, B, \mathcal{V}ar)$ assigning not necessarily total terms to variables. They can be represented as sets of bindings $X \mapsto t$ and extended to act over other syntactic objects o. The *domain* $\mathrm{dom}(\sigma)$ and *variable range* $\mathrm{vran}(\sigma)$ are defined in the usual way. We will write $o\sigma$ for the result of applying σ to o. The *composition* $\sigma\sigma'$ of two substitutions is such that $o(\sigma\sigma')$ equals $(o\sigma)\sigma'$.

By adapting the definition found in Section 2.2 of [9] to a first-order setting, we formalize a constraint domain of signature Σ as any algebraic structure of the form $\mathcal{C} = \langle C, \{p^{\mathcal{C}} \mid p \in PF\} \rangle$ such that:

1. The carrier set C is $\mathrm{Term}_\bot(\Sigma, B)$ 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. $p^{\mathcal{C}} \subseteq C^n \times C$, written simply as $p^{\mathcal{C}} \subseteq C$ in the case $n = 0$, is called the *interpretation* of p in $\mathcal{C}$. We will write $p^{\mathcal{C}}(\overline{t}_n) \to t$ (or simply $p^{\mathcal{C}} \to t$ if $n = 0$) to indicate that $(\overline{t}_n, t) \in p^{\mathcal{C}}$.

3. Each primitive interpretation $p^{\mathcal{C}}$ has *monotonic* and *radical* behavior w.r.t. the information ordering $\sqsubseteq$, in the technical sense defined in [4].

Note that symbols $h \in DC \cup DF$ are given no interpretation in $\mathcal{C}$. As we will see in Section 3, symbols in $c \in DC$ are interpreted as free constructors, and the interpretation of symbols $f \in DF$ is program-dependent. We assume that any signature Σ includes two nullary constructors *true* and $false$ for the boolean values, and a binary symbol $== \in PF^2$ used in infix notation and interpreted as *strict equality*; see [9] for details. For the examples in this paper we will use a constraint domain $\mathcal{R}$ whose set of basic elements is $C_{\mathcal{R}} = \mathbb{R}$ and whose primitives functions correspond to the usual arithmetic operations $+, \times, \ldots$ and the usual boolean-valued comparison operations $\leq, <, \ldots$ over $\mathbb{R}$. Other useful instances of constraint domains can be found in [9].

Atomic constraints over $\mathcal{C}$ have the form $p(\overline{e}_n) == v$ with $p \in PF^n$, $e_i \in \mathrm{Exp}_\bot(\Sigma, B, \mathcal{V}ar)$ and $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$. Atomic constraints of the form $p(\overline{e}_n) == true$ are abbreviated as $p(\overline{e}_n)$. In particular, $(e_1 == e_2) == true$ is abbreviated as $e_1 == e_2$. Atomic constraints of the form $(e_1 == e_2) == false$ are abbreviated as $e_1 \mathrel{/=} e_2$.

Compound constraints are built from atomic constraints using logical conjunction, existential quantification, and sometimes other logical operations. Constraints without occurrences of symbols $f \in DF$ are called *primitive*. We will note atomic constraints as δ, sets of atomic constraints as Δ, atomic primitive constraints as π, and sets of atomic primitive constraints as Π. When interpreting sets of constraints, we will treat them as the conjunction of their members.

Ground substitutions η such that $X\eta \in \mathrm{Term}_\bot(\Sigma, B)$ for all $X \in \mathrm{dom}(\eta)$ are called *variable valuations* over $\mathcal{C}$. The set of all possible variable valuations is noted $\mathrm{Val}_{\mathcal{C}}$. The *solution set* $\mathrm{Sol}_{\mathcal{C}}(\Pi) \subseteq \mathrm{Val}_{\mathcal{C}}$ includes as members those valuations η such that $\pi\eta$ is true in $\mathcal{C}$ for all $\pi \in \Pi$; see [9] for a formal definition. In case that $\mathrm{Sol}_{\mathcal{C}}(\Pi) = \emptyset$ we say that Π is *unsatisfiable* and we write $\mathrm{Unsat}_{\mathcal{C}}(\Pi)$. In case that $\mathrm{Sol}_{\mathcal{C}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{C}}(\pi)$ we say that π is *entailed* by Π in $\mathcal{C}$ and we write $\Pi \models_{\mathcal{C}} \pi$. Note that the notions defined in this paragraph only make sense for primitive constraints.

In this paper we are interested in pairs consisting of a qualification domain and a constraint domain that are related in the following technical sense:

Definition 1 (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 $D_{\mathcal{D}} \setminus \{\mathbf{b}\} \subseteq C_{\mathcal{C}}$ and the two following requirements are satisfied:*

1. *There is a primitive $\mathcal{C}$-constraint $\mathsf{qVal}(X)$ depending on the variable X, such that $Sol_{\mathcal{C}}(\mathsf{qVal}(X)) = \{\eta \in Val_{\mathcal{C}} \mid \eta(X) \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}\}$.*
2. *There is a primitive $\mathcal{C}$-constraint $\mathsf{qBound}(X,Y,Z)$ depending on the variables X, Y, Z, such that any $\eta \in Val_{\mathcal{C}}$ such that $\eta(X)$, $\eta(Y)$, $\eta(Z) \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $\eta \in Sol_{\mathcal{C}}(\mathsf{qBound}(X,Y,Z)) \iff \eta(X) \trianglelefteq \eta(Y) \circ \eta(Z)$.* □

Intuitively, $\mathsf{qBound}(X,Y,Z)$ encodes the $\mathcal{D}$-statement $X \trianglelefteq Y \circ Z$ as a $\mathcal{C}$-constraint. As convenient notations, we will write $\ulcorner X \trianglelefteq Y \circ Z \urcorner$, $\ulcorner X \trianglelefteq Y \urcorner$ and $\ulcorner X \trianglerighteq Y \urcorner$ in place of $\mathsf{qBound}(X, Y, Z)$, $\mathsf{qBound}(X, \mathbf{t}, Y)$ and $\mathsf{qBound}(Y, \mathbf{t}, X)$, respectively. In the sequel, $\mathcal{C}$-constraints of the form $\ulcorner\kappa\urcorner$ are called *qualification constraints*, and Ω is used as notation for sets of qualification constraints. We also write $\mathrm{Val}_{\mathcal{D}}$ for the set of all $\mu \in \mathrm{Val}_{\mathcal{C}}$ such that $X\mu \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ for all $X \in \mathrm{dom}(\mu)$, called *$\mathcal{D}$-valuations.*

Note that $\mathcal{U}$ can be expressed in $\mathcal{R}$, because $D_{\mathcal{U}} \setminus \{0\} = (0, 1] \subseteq \mathbb{R} \subseteq C_{\mathcal{R}}$, $\mathsf{qVal}(X)$ can be built as the $\mathcal{R}$-constraint $0 < X \wedge X \leq 1$ and $\ulcorner X \trianglelefteq Y \circ Z\urcorner$ can be built as the $\mathcal{R}$-constraint $X \leq Y \times Z$. Other instances of qualification domains presented in [14] are also expressible in $\mathcal{R}$.

3 A Qualified Declarative Programming Scheme

In this section we present the scheme QCFLP($\mathcal{D}, \mathcal{C}$) announced in the Introduction and its declarative semantics. The parameters $\mathcal{D}$ and $\mathcal{C}$ stand for a qualification domain and a constraint domain with certain signature Σ, respectively. By convention, we only allow those instances of the scheme verifying that $\mathcal{D}$ is expressible in $\mathcal{C}$ in the sense of Definition 1. For example, QCFLP($\mathcal{U}, \mathcal{R}$) is an allowed instance.

A QCFLP($\mathcal{D}, \mathcal{C}$)-program is a set $\mathcal{P}$ of program rules. A program rule has the form $f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \Delta$ where $f \in DF^n$, $\overline{t}_n$ is a linear sequence of Σ-terms (where each variable occurs just once), $\alpha \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ is an attenuation factor, r is a Σ-expression and Δ is a sequence of atomic $\mathcal{C}$-constraints δ_j $(1 \leq j \leq m)$, interpreted as conjunction. The undefined symbol $\bot$ is not allowed to occur in program rules. The library program shown in Figure 1 serves as an example of QCFLP($\mathcal{U}, \mathcal{R}$)-program. Leaving aside the attenuation factors, this is clearly not a confluent conditional term rewriting system. Certain program rules, as e.g. those for `guessGenre`, are intended to specify the behavior of *non-deterministic functions.* As argued elsewhere [13], the semantics of non-deterministic functions for the purposes of Functional Logic Programming is not suitably described by ordinary rewriting. We overcome this difficulty by designing a formal inference system noted QCRWL($\mathcal{D}, \mathcal{C}$) and called *Qualified Constrained Rewriting Logic.* First, we define the kind of statements that can be inferred in this logic:

Definition 2 (qc-Statements). *Assume a partial Σ-expression e, partial Σ-terms $t, t', \overline{t}_n$, a qualification value $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$, an atomic $\mathcal{C}$-constraint δ and a finite set of atomic primitive $\mathcal{C}$-constraints Π. A* qualified constrained statement *(briefly, qc-statement) φ must have one of the following two forms:*

1. qc-production $(e \to t)\sharp d \Leftarrow \Pi$. *Such a qc-statement is called* trivial *iff either t is $\bot$ or else $\mathit{Unsat}_{\mathcal{C}}(\Pi)$. Its intuitive meaning is that a rewrite sequence $e \to^* t'$ using program rules and with attached qualification value d is allowed in our intended semantics for some $t' \sqsupseteq t$, under the assumption that Π holds. By convention, qc-productions of the form $(f(\overline{t}_n) \to t)\sharp d \Leftarrow \Pi$ with $f \in DF^n$ are called* qc-facts.
2. qc-atom $\delta\sharp d \Leftarrow \Pi$. *Such a qc-statement is called* trivial *iff $\mathit{Unsat}_{\mathcal{C}}(\Pi)$. Its intuitive meaning is that δ is entailed by the program rules with attached qualification value d, under the assumption that Π holds.* □

QTI $$\frac{}{\varphi}$$ if φ is a trivial qc-statement.

QRR $$\frac{}{(v \to v)\sharp d \Leftarrow \Pi}$$ if $v \in \mathcal{V}ar \cup B_{\mathcal{C}}$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$.

QDC $$\frac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\dots n}}{(c(\overline{e}_n) \to c(\overline{t}_n))\sharp d \Leftarrow \Pi}$$ if $c \in DC^n$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(1 \le i \le n)$.

QDF$_{\mathcal{P}}$ $$\frac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\dots n} \quad (r \to t)\sharp d'_0 \Leftarrow \Pi \quad (\delta_j \sharp d'_j \Leftarrow \Pi)_{j=1\dots m}}{(f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi}$$

if $f \in DF^n$ and $(f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \delta_1, \dots, \delta_m) \in [\mathcal{P}]_\perp$ where $[\mathcal{P}]_\perp = \{R_l\theta \mid R_l$ is a rule in $\mathcal{P}$ and θ is a substitution$\}$ is the set of program rule instances, and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(1 \le i \le n)$, $d \trianglelefteq \alpha \circ d'_j$ $(0 \le j \le m)$.

QPF $$\frac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\dots n}}{(p(\overline{e}_n) \to v)\sharp d \Leftarrow \Pi}$$ if $p \in PF^n$, $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$,

$\Pi \models_{\mathcal{C}} p(\overline{t}_n) \to v$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(1 \le i \le n)$.

QAC $$\frac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\dots n}}{(p(\overline{e}_n) == v)\sharp d \Leftarrow \Pi}$$ if $p \in PF^n$, $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$,

$\Pi \models_{\mathcal{C}} p(\overline{t}_n) == v$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(1 \le i \le n)$.

Fig. 2. Qualified Constrained Rewriting Logic

Next, we define QCRWL($\mathcal{D}, \mathcal{C}$) as the formal system consisting of the six inference rules displayed in Fig. 2. They are based on the first-order fragment of the Constrained Rewriting Logic presented in [9], suitably extended to manage attached qualification values. These inference rules formalize provability of qc-statements according to their intuitive meanings. In particular, **QDF$_{\mathcal{P}}$** formalizes the applications of a program rule instance to infer that $f(\overline{e}_n)$ returns a result t with qualification d. Note that d is bounded by the qualifications d_i corresponding to the evaluation of e_i, and also by $\alpha \circ d'_j$ corresponding to the evaluation of the right hand side and the conditions of the rule attenuated by α.

In the sequel we use the notation $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ to indicate that φ can be inferred from $\mathcal{P}$ in QCRWL($\mathcal{D}, \mathcal{C}$). By convention, we agree that no other inference rule is used whenever **QTI** is applicable. Therefore, trivial qc-statements can only be inferred by rule **QTI**. As usual in formal inference systems, QCRWL($\mathcal{D}, \mathcal{C}$) proofs can be represented as trees whose nodes correspond to inference steps. For example, if $\mathcal{P}$ is the library program, Π is empty, and ψ is

```
(guessGenre(book(4,"Beim Hauten der Zwiebel","Gunter Grass",
    "German","Biography", medium, 432)) --> "Essay")#0.7
```

then $\mathcal{P} \vdash_{\mathcal{U},\mathcal{R}} \psi \Leftarrow \Pi$ with a proof tree whose root inference can be chosen as $\mathbf{QDF}_{\mathcal{P}}$ using a suitable instance of the 4th program rule for `guessGenre`.

Extending ideas from [9], it is possible to define *qc-interpretations* as sets $\mathcal{I}$ of qc-facts that verify certain closure conditions. Moreover, *models* of $\mathcal{P}$ can be defined to be those interpretations that satisfy the program rules in a suitable sense. The following result can be proved:

Theorem 1 (Least Program Model). *For any QCFLP($\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$, $S_{\mathcal{P}} = \{\varphi \mid \varphi$ is a qc-fact and $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi\}$ is the least model of $\mathcal{P}$ w.r.t. set inclusion. An alternative characterization of $S_{\mathcal{P}}$ as least fixpoint is also possible.* □

Assume now a QCFLP($\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$ and a countable set $\mathcal{W}ar$ of so-called *qualification variables*, disjoint from $\mathcal{V}ar$ and $\mathcal{C}$'s signature Σ. Then:

Definition 3 (Goals and their Solutions).

1. *A* goal *G for $\mathcal{P}$ has the form $\delta_1\sharp W_1, \ldots, \delta_m\sharp W_m \;[]\; W_1 \trianglerighteq \beta_1, \ldots, W_m \trianglerighteq \beta_m$, abbreviated as $(\ \delta_i\sharp W_i,\ W_i \trianglerighteq \beta_i\ )_{i=1\ldots m}$, where $\delta_i\sharp W_i$ $(1 \le i \le m)$ are atomic $\mathcal{C}$-constraints annotated with different qualification variables W_i, and $W_i \trianglerighteq \beta_i$ are so-called* threshold conditions, *with $\beta_i \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ $(1 \le i \le m)$.*
2. *A* solution *for G is any triple $\langle \sigma, \mu, \Pi \rangle$ such that σ is a substitution, μ is a $\mathcal{D}$-valuation, Π is a finite set of atomic primitive $\mathcal{C}$-constraints, and the following two conditions hold for all $1 \le i \le m$: $W_i\mu = d_i \trianglerighteq \beta_i$, and $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (\delta_i\sigma)\sharp d_i \Leftarrow \Pi$. The set of all solutions for G is noted $Sol_{\mathcal{P}}(G)$.* □

Thanks to Theorem 1, solutions of $\mathcal{P}$ are valid in the least model $S_{\mathcal{P}}$ and hence in all models of $\mathcal{P}$. A goal for the library program and one solution for it have been presented in the Introduction. In this particular example, $\Pi = \emptyset$ and the QCRWL($\mathcal{U},\mathcal{R}$) proof needed to check the solution according to Definition 3 can be formalized by following the intuitive ideas sketched in the Introduction.

4 Implementation by Program Transformation

Goal solving in instances of the CFLP($\mathcal{C}$) scheme from [9] has been formalized by means of *constrained narrowing* procedures as e.g. [8,5], and is supported by systems such as Curry [7] and $\mathcal{TOY}$ [2]. In this section we present a semantically correct transformation from QCFLP($\mathcal{D},\mathcal{C}$) into the first-order fragment of CFLP($\mathcal{C}$) which can be used for implementing goal solving in QCFLP($\mathcal{D},\mathcal{C}$).

By abuse of notation, the first-order fragment of the CFLP($\mathcal{C}$) scheme will be noted simply as CFLP($\mathcal{C}$) in the sequel. A formal description of CFLP($\mathcal{C}$) is easily derived from the previous Section 3 by simply omitting everything related to qualification domains and values. Programs $\mathcal{P}$ are sets of program rules of the form $f(\bar{t}_n) \to r \Leftarrow \Delta$, with no attenuation factors attached. Program semantics is characterized by a Constrained Rewriting Logic CRWL($\mathcal{C}$) where c-statements can be derived from a given program. A c-statement may be a c-production $e \to t \Leftarrow \Pi$ or a c-atom $\delta \Leftarrow \Pi$. The six inference rules **RL** of CRWL($\mathcal{C}$) are easy to derive from the corresponding rules **QRL** of QCRWL($\mathcal{D},\mathcal{C}$). For instance, the CRWL($\mathcal{C}$) rule derived from **QAC** by forgetting qualifications is:

$$\mathbf{AC} \quad \frac{(\ e_i \to t_i \Leftarrow \Pi\ )_{i=1\ldots n}}{p(\overline{e}_n) == v \Leftarrow \Pi} \quad \begin{array}{l}\text{if } p \in PF^n,\ v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}} \\ \text{and } \Pi \models_{\mathcal{C}} p(\overline{t}_n) == v.\end{array}$$

The notation $\mathcal{P} \vdash_{\mathcal{C}} \varphi$ indicates that φ can be inferred from $\mathcal{P}$ in CRWL($\mathcal{C}$). In analogy to Theorem 1, it is possible to prove that the least model of $\mathcal{P}$ w.r.t. set inclusion can be characterized as $S_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is a c-fact and } \mathcal{P} \vdash_{\mathcal{C}} \varphi\}$. In analogy to Definition 3, goals G for a CFLP($\mathcal{C}$)-program $\mathcal{P}$ have the form $\delta_1, \ldots, \delta_m$ where δ_j are atomic $\mathcal{C}$-constraints, and $Sol_{\mathcal{P}}(G)$ is defined as the set of all the pairs $\langle\sigma, \Pi\rangle$ such that σ is a substitution, Π is a finite set of atomic primitive $\mathcal{C}$-constraints, and $\mathcal{P} \vdash_{\mathcal{C}} \delta_j\sigma \Leftarrow \Pi$ holds for $1 \le j \le m$.

The transformation goes from a source signature Σ into a target signature Σ' such that each $f \in DF^n$ in Σ becomes $f' \in DF^{n+1}$ in Σ', and all the other symbols in Σ remain the same in Σ'. It works by introducing fresh variables W to represent the qualification values attached to the results of calls to defined functions, as well as qualification constraints to be imposed on such variables. There are four groups of transformation rules displayed in Figure 3. Let us comment them in order.

Transforming an expression e yields a triple $e^{\mathcal{T}} = (e', \Omega, \mathcal{W})$, where Ω is a set of qualification constraints and $\mathcal{W}$ is the set of qualification variables occurring in e' at outermost positions. The qualification value attached to e cannot exceed the infimum in $\mathcal{D}$ of the values of the variables $W \in \mathcal{W}$, and $e^{\mathcal{T}}$ is computed by recursion on e's syntactic structure as specified by the transformation rules **TAE**, **TCE**$_1$ and **TCE**$_2$. Note that **TCE**$_2$ introduces a new qualification variable W for each call to a defined function $f \in DF^n$ and builds a set Ω' of qualification constraints ensuring that W must be interpreted as a qualification value not greater than the qualification values attached to f's arguments. **TCE**$_1$ deals with calls to constructors and primitive functions just by collecting information from the arguments, and **TAE** is self-explanatory.

Unconditional productions and atomic constraints are transformed by means of **TP** and **TA**, respectively, relying on the transformation of expressions in the obvious way. Relying on **TP** and **TA**, **TCS** transforms a qc-statement of the form $\psi\sharp d \Leftarrow \Pi$ into a c-statement whose conditional part includes, in addition to Π, the qualification constraints Ω coming from $\psi^{\mathcal{T}}$ and extra qualification constraints ensuring that d is not greater than allowed by ψ's qualification.

Program rules are transformed by **TPR**. Transforming the left-hand side $f(\overline{t}_n)$ introduces a fresh symbol $f' \in DF^{n+1}$ and a fresh qualification variable W. The transformed right-hand side r' comes from $r^{\mathcal{T}}$, and the transformed conditions are obtained from the constraints coming from $r^{\mathcal{T}}$ and ${\delta_i}^{\mathcal{T}}(1 \le i \le m)$ by adding extra qualification constraints to be imposed on W, namely $\mathsf{qVal}(W)$ and $(\ulcorner W \trianglelefteq \alpha \circ W' \urcorner)_{W' \in \mathcal{W}'}$, for $\mathcal{W}' = \mathcal{W}_r$ and $\mathcal{W}' = \mathcal{W}_i$ $(1 \le i \le m)$. By convention, $(\ulcorner W \trianglelefteq \alpha \circ W' \urcorner)_{W' \in \mathcal{W}'}$ is understood as $\ulcorner W \trianglelefteq \alpha \urcorner$ in case that $\mathcal{W}' = \emptyset$. The idea is that W's value cannot exceed the infimum in $\mathcal{D}$ of all the values $\alpha \circ \beta$, for the different β coming from the qualifications of r and δ_i $(1 \le i \le m)$. The result of applying **TPR** to all the program rules of a program $\mathcal{P}$ will be noted as $\mathcal{P}^{\mathcal{T}}$.

Transforming Expressions

TAE $$\frac{}{v^{\mathcal{T}} = (v, \emptyset, \emptyset)} \quad \text{if } v \in \mathcal{V}ar \cup B_{\mathcal{C}}$$

TCE$_1$ $$\frac{(\ e_i{}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)\ )_{i=1\ldots n}}{h(\overline{e}_n)^{\mathcal{T}} = (h(\overline{e'}_n),\ \bigcup_{i=1}^{n} \Omega_i,\ \bigcup_{i=1}^{n} \mathcal{W}_i)} \quad \text{if } h \in DC^n \cup PF^n$$

TCE$_2$ $$\frac{(\ e_i{}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)\ )_{i=1\ldots n}}{f(\overline{e}_n)^{\mathcal{T}} = (f'(\overline{e'}_n, W), \Omega', \{W\})}$$

if $f \in DF^n$ and W is a fresh variable,
where $\Omega' = (\bigcup_{i=1}^{n} \Omega_i) \cup \{\mathsf{qVal}(W)\} \cup \{\ulcorner W \trianglelefteq W' \urcorner \mid W' \in \bigcup_{i=1}^{n} \mathcal{W}_i\}$.

Transforming qc-Statements

TP $$\frac{e^{\mathcal{T}} = (e', \Omega, \mathcal{W})}{(e \to t)^{\mathcal{T}} = (e' \to t,\ \Omega, \mathcal{W})}$$

TA $$\frac{(\ e_i{}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)\ )_{i=1\ldots n}}{(p(\overline{e}_n) == v)^{\mathcal{T}} = (\ p(\overline{e'}_n) == v, \bigcup_{i=1}^{n} \Omega_i, \bigcup_{i=1}^{n} \mathcal{W}_i\ )}$$

if $p \in PF^n$, $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$.

TCS $$\frac{\psi^{\mathcal{T}} = (\psi', \Omega, \mathcal{W})}{(\psi \sharp d \Leftarrow \Pi)^{\mathcal{T}} = (\psi' \Leftarrow \Pi, \Omega \cup \{\ulcorner d \trianglelefteq W \urcorner \mid W \in \mathcal{W}\})}$$

if ψ is of the form $e \to t$ or $p(\overline{e}_n) == v$ and $d \in D_{\mathcal{D}}$.

Transforming Program Rules

TPR $$\frac{r^{\mathcal{T}} = (r', \Omega_r, \mathcal{W}_r) \qquad (\ \delta_i{}^{\mathcal{T}} = (\delta'_i, \Omega_i, \mathcal{W}_i)\ )_{i=1\ldots m}}{\begin{array}{l}(f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \delta_1, \ldots, \delta_m)^{\mathcal{T}} = \\ \quad f'(\overline{t}_n, W) \to r' \Leftarrow \mathsf{qVal}(W),\ \Omega_r,\ (\ulcorner W \trianglelefteq \alpha \circ W' \urcorner)_{W' \in \mathcal{W}_r}, \\ \qquad (\ \Omega_i,\ (\ulcorner W \trianglelefteq \alpha \circ W' \urcorner)_{W' \in \mathcal{W}_i},\ \delta'_i\ )_{i=1\ldots m}\end{array}}$$

where W is a fresh variable.

Transforming Goals

TG $$\frac{(\ \delta_i{}^{\mathcal{T}} = (\delta'_i, \Omega_i, \mathcal{W}_i)\ )_{i=1\ldots m}}{\begin{array}{l}((\ \delta_i \sharp W_i, W_i \trianglerighteq \beta_i\ )_{i=1\ldots m})^{\mathcal{T}} = \\ \quad (\ \Omega_i,\ \mathsf{qVal}(W_i),\ (\ulcorner W_i \trianglelefteq W \urcorner)_{W \in \mathcal{W}_i},\ \ulcorner W_i \trianglerighteq \beta_i \urcorner,\ \delta'_i\ )_{i=1\ldots m}\end{array}}$$

Fig. 3. Transformation rules

Finally, **TG** transforms a goal $(\ \delta_i \sharp W_i, W_i \trianglerighteq \beta_i\ )_{i=1\ldots m}$ by transforming each atomic constraint δ_i and adding $\mathsf{qVal}(W_i)$, $(\ulcorner W_i \trianglelefteq W' \urcorner)_{W' \in \mathcal{W}'_i}$ and $\ulcorner W_i \trianglerighteq \beta_i \urcorner$ $(1 \leq i \leq m)$ to ensure that each W_i is interpreted as a qualification value not bigger than the qualification computed for δ_i and satisfying the threshold condition $W_i \trianglerighteq \beta_i$. In case that $\mathcal{W}'_i = \emptyset$, $(\ulcorner W_i \trianglelefteq W' \urcorner)_{W' \in \mathcal{W}'_i}$ is understood as $\ulcorner W_i \trianglelefteq \mathbf{t} \urcorner$.

Program semantics in QCFLP($\mathcal{D},\mathcal{C}$) and CFLP($\mathcal{C}$) is characterized by derivability in QCRWL($\mathcal{D},\mathcal{C}$) and CRWL($\mathcal{C}$), respectively. Therefore, the following theorem proves the semantic correctness of the program transformation:

Theorem 2. *Let $\mathcal{P}$ be a QCFLP($\mathcal{D},\mathcal{C}$)-program and $\psi \sharp d \Leftarrow \Pi$ a qc-statement such that $(\psi \sharp d \Leftarrow \Pi)^{\mathcal{T}} = (\psi' \Leftarrow \Pi, \Omega')$. Then the two following statements are equivalent:*

1. *$\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \psi \sharp d \Leftarrow \Pi$.*
2. *$\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \psi' \rho \Leftarrow \Pi$ for some $\rho \in Sol_{\mathcal{C}}(\Omega')$ such that $dom(\rho) = var(\Omega')$.*

Proof. (Sketch; a full proof can be found in [4] as Proof of Theorem 3).

[1. $\Rightarrow$ 2.] *(Transformation completeness).* Assume $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \psi \sharp d \Leftarrow \Pi$ by means of a QCRWL($\mathcal{D},\mathcal{C}$) proof tree T with k nodes. By induction on k we show the existence of a CRWL($\mathcal{C}$) proof tree T' witnessing $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \psi' \rho \Leftarrow \Pi$ for some $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega)$ such that $\mathrm{dom}(\rho) = \mathrm{var}(\Omega')$. In the base case $k = 1$, T contains just one root node inferred by a QCRWL($\mathcal{D},\mathcal{C}$) inference rule **QRL** other than $\mathbf{QDF}_{\mathcal{P}}$ and with no premises. Then T' can be easily built as a proof tree which also contains just one root node inferred by the QCRWL($\mathcal{D},\mathcal{C}$) inference rule **RL** with no premises. In the inductive case $k > 1$ the QCRWL($\mathcal{D},\mathcal{C}$) inference rule **QRL** applied at T's root can be neither **QTI** nor **QRR**. Here we argue only for the case where **QRL** is **QAC**. In this case ψ has the form $p(\overline{e}_n) == v$ and according to Figure 2 the inference step at T's root has the form:

$$\frac{(\ (e_i \to t_i) \sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(p(\overline{e}_n) == v) \sharp d \Leftarrow \Pi}$$

where $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$, $\Pi \models_{\mathcal{C}} p(\overline{t}_n) == v$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(1 \leq i \leq n)$. Assume $(\ e_i{}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)\ )_{i=1\ldots n}$, using different fresh variables W in each case. Then the transformation rules **TA** and **TCS** yield $((p(\overline{e}_n) == v) \sharp d \Leftarrow \Pi)^{\mathcal{T}} = p(\overline{e'}_n) == v \Leftarrow \Pi, \Omega'$ and $((e_i \to t_i) \sharp d \Leftarrow \Pi)^{\mathcal{T}} = e'_i \to t_i \Leftarrow \Pi, \Omega'_i$, where $\Omega' = \bigcup_{i=1}^n \Omega_i \cup \{\ulcorner d \trianglelefteq W \urcorner \mid W \in \bigcup_{i=1}^n \mathcal{W}_i\}$ and $\Omega'_i = \Omega_i \cup \{\ulcorner d_i \trianglelefteq W \urcorner \mid W \in \mathcal{W}_i\}$. For each $1 \leq i \leq n$, $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (e_i \to t_i) \sharp d_i \Leftarrow \Pi$ is witnessed by a QCRWL($\mathcal{D},\mathcal{C}$) proof tree T_i which is subtree of T and has less than k nodes. Therefore, by induction hypothesis we get CRWL($\mathcal{C}$) proof trees T'_i $(1 \leq i \leq n)$ witnessing $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (e'_i \to t_i)\rho_i \Leftarrow \Pi$ for certain $\rho_i \in \mathrm{Sol}_{\mathcal{C}}(\Omega_i)$ such that $\mathrm{dom}(\rho_i) = \mathrm{var}(\Omega'_i)$. Consider $\rho = \biguplus_{i=1}^n \rho_i \in \mathrm{Val}_{\mathcal{D}}$, which is is well defined because the sets $\mathrm{var}(\Omega'_i), 1 \leq i \leq n$, are pairwise disjoint. Note that $\mathrm{dom}(\rho) = \bigcup_{i=1}^n \mathrm{dom}(\rho_i) = \bigcup_{i=1}^n \mathrm{var}(\Omega'_i) = \mathrm{war}(\Omega')$. Moreover, $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega')$. In fact, for each $1 \leq i \leq n$, $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega_i)$ follows from $\rho_i \in \mathrm{Sol}_{\mathcal{C}}(\Omega_i)$; and for each $1 \leq i \leq n$ and each $W \in \mathcal{W}_i$, $\rho \in \mathrm{Sol}_{\mathcal{C}}(\ulcorner d \trianglelefteq W \urcorner)$ follows from $d \trianglelefteq d_i$

(ensured by the **QAC** inference at T's root) and $\rho \in \mathrm{Sol}_{\mathcal{C}}(\ulcorner d_i \trianglelefteq W \urcorner)$ (ensured by $\rho_i \in \mathrm{Sol}_{\mathcal{C}}(\Omega'_i)$ and $\ulcorner d_i \trianglelefteq W \urcorner \in \Omega'_i$). Finally, $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} ((p(\overline{e'}_n) == v)\rho \Leftarrow \Pi$ is witnessed by a proof tree T' whose root inference using **AC** has the form

$$\frac{((\ e'_i \rightarrow t_i)\rho \Leftarrow \Pi\ )_{i=1\dots n}}{(p(\overline{e'}_n) == v)\rho \Leftarrow \Pi}$$

and where each premise $(e'_i \rightarrow t_i)\rho \Leftarrow \Pi$ is identical to $(e'_i \rightarrow t_i)\rho_i \Leftarrow \Pi$ and therefore proved by the CRWL($\mathcal{C}$) proof tree T'_i.

[2. $\Rightarrow$ 1.] *(Transformation soundness).* Assume $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega')$ such that $\mathrm{dom}(\rho) = \mathrm{var}(\Omega')$ and $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \psi'\rho \Leftarrow \Pi$ by means of a CRWL($\mathcal{C}$) proof tree T' with k nodes. Reasoning by induction on k we show the existence of a QCRWL($\mathcal{D},\mathcal{C}$) proof tree T witnessing $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \psi\sharp d \Leftarrow \Pi$. The base case $k = 1$ is easy. For the inductive case $k > 1$ we distinguish cases according to the CRWL($\mathcal{C}$) inference rule **RL** applied at the root of T'. Here we argue only for the case where **RL** is **AC**. In this case ψ, ψ', Ω', the proof tree T' and the subtrees T'_i of T' proving the premises of the **AC** inference at the root of T' have the forms described in the first part of the proof. For each $1 \le i \le n$, let $d_i = d$ and $\rho_i = \rho\restriction\mathrm{var}(\Omega'_i)$. Then $\rho_i \in \mathrm{Sol}_{\mathcal{C}}(\Omega'_i)$ follows from $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega')$. Moreover, $(e'_i \rightarrow t_i)\rho_i \Leftarrow \Pi$ is identical to the i-th premise of the **AC** inference at the root of T', and therefore $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (e'_i \rightarrow t_i)\rho_i \Leftarrow \Pi$ is witnessed by T'_i, which has less than k nodes. By induction hypothesis we can obtain QCRWL($\mathcal{D},\mathcal{C}$) proof trees T_i witnessing $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (e_i \rightarrow t_i)\sharp d_i \Leftarrow \Pi$. Since $d = d_i$, the conditions $d \trianglelefteq d_i$ $(1 \le i \le n)$ hold trivially, and T can be built as a QCRWL($\mathcal{D},\mathcal{C}$) proof tree having the form described in the beginning, with the inference rule **QAC** applied at the root and the proof trees T_i witnessing the premisses. $\square$

Using Theorem 2 we can prove that the transformation of goals specified in Figure 3 preserves solutions in the sense of the following result.

Theorem 3. *Let G be a goal for a given QCFLP($\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$. Then, the two following statements are equivalent:*

1. *$\langle \sigma, \mu, \Pi \rangle \in Sol_{\mathcal{P}}(G)$.*
2. *$\langle \sigma \uplus \mu \uplus \rho, \Pi \rangle \in Sol_{\mathcal{P}^{\mathcal{T}}}(G^{\mathcal{T}})$ for some $\rho \in Val_{\mathcal{D}}$ such that $dom(\rho)$ is the set of new variables W introduced by the transformation of G.*

Proof. Let $G = (\ \delta_i\sharp W_i, W_i \trianglerighteq \beta_i\ )_{i=1\dots m}$, σ and μ be given. For $i = 1 \dots m$, consider ${\delta_i}^{\mathcal{T}} = (\delta'_i, \Omega_i, \mathcal{W}_i)$ and $\Omega'_i = \Omega_i \cup \{\ulcorner W_i \trianglelefteq W \urcorner \mid W \in \mathcal{W}_i\}$. According to Fig. 3, $G^{\mathcal{T}} = (\Omega'_i, \mathsf{qVal}(W_i), \ulcorner W_i \trianglerighteq \beta_i \urcorner, \delta'_i)_{i=1\dots m}$. Then, because of Def. 3(2) and the analogous notion of solution for CFLP($\mathcal{C}$) goals explained in Sect. 3, the two statements of the theorem can be reformulated as follows:

(a) $W_i\mu \trianglerighteq \beta_i$ and $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \delta_i\sigma\sharp W_i\mu \Leftarrow \Pi$ hold for $i = 1 \dots m$.
(b) There exists $\rho \in \mathrm{Val}_{\mathcal{D}}$ with $\mathrm{dom}(\rho) = \bigcup_{i=1}^{m} \mathrm{var}(\Omega_i)$ such that $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega'_i\mu)$, $W_i\mu \trianglerighteq \beta_i$ and $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\delta'_i\sigma)\rho \Leftarrow \Pi$ hold for $i = 1 \dots m$.

$[(a) \Rightarrow (b)]$ Assume (a). Note that $\delta_i\sigma\sharp W_i\mu \Leftarrow \Pi^{\mathcal{T}}$ is $\delta'_i\sigma \Leftarrow \Pi, \Omega'_i\mu$. Applying Theorem 2 (with $\psi = \delta_i\sigma$, $d = W_i\mu$ and Π) we obtain $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\delta'_i\sigma)\rho_i \Leftarrow \Pi$ for some $\rho_i \in \mathrm{Sol}_{\mathcal{C}}(\Omega'_i\mu)$ with $\mathrm{dom}(\rho_i) = \mathrm{var}(\Omega'_i\mu) = \mathrm{var}(\Omega_i)$. Then (b) holds for $\rho = \biguplus_{i=1}^{m} \rho_i$.
$[(b) \Rightarrow (a)]$ Assume (b). Let $\rho_i = \rho\restriction\mathrm{var}(\Omega_i)$, $i = 1 \ldots m$. Note that (b) ensures $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\delta'_i\sigma)\rho_i \Leftarrow \Pi$ and $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega'_i\mu)$. Then Theorem 2 can be applied (again with $\psi = \delta_i\sigma$, $d = W_i\mu$ and Π) to obtain $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \delta_i\sigma\sharp W_i\mu \Leftarrow \Pi$. Therefore, (a) holds. $\square$

As an example of goal solving via the transformation, we consider again the *library program* $\mathcal{P}$ and the goal G discussed in the Introduction. Both belong to the instance QCFLP$(\mathcal{U}, \mathcal{R})$ of our scheme. Their translation into CFLP$(\mathcal{R})$ can be executed in the $\mathcal{TOY}$ system [2] after loading the Real Domain Constraints library (`cflpr`). The source and translated code are publicly available at `gpd.sip.ucm.es/cromdia/qlp`. Solving the transformed goal in $\mathcal{TOY}$ computes the answer announced in the Introduction as follows:

```
Toy(R)> qVal([W]), W>=0.65, search("German","Essay",intermediate,W) == R
      { R -> 4 }
      { W=<0.7, W>=0.65 }
sol.1, more solutions (y/n/d/a) [y]? no
```

The best qualification value for `W` provided by the answer constraints is `0.7`.

5 Conclusions and Future Work

The work in this paper is based on the scheme CFLP$(\mathcal{C})$ for functional logic programming with constraints presented in [9]. Our main results are: a new programming scheme QCFLP$(\mathcal{D}, \mathcal{C})$ extending the first-order fragment of CFLP$(\mathcal{C})$ with qualified computation capabilities; a rewriting logic QCRWL$(\mathcal{D}, \mathcal{C})$ characterizing QCFLP$(\mathcal{D}, \mathcal{C})$-program semantics; and a transformation of QCFLP$(\mathcal{D}, \mathcal{C})$ into CFLP$(\mathcal{C})$ preserving program semantics and goal solutions, that can be used as a correct implementation technique. Existing CFLP$(\mathcal{C})$ systems such as $\mathcal{TOY}$ [2] and Curry [7] that use definitional trees as an efficient implementation tool can easily adopt the implementation, since the structure of definitional trees is quite obviously preserved by the transformation.

As argued in the Introduction, our scheme is more expressive than the main related approaches we are aware of. By means of an example dealing with a simplified library, we have shown that instances of QCFLP$(\mathcal{D}, \mathcal{C})$ can serve as a declarative language for flexible information retrieval problems, where qualified (rather than exact) answers to user's queries can be helpful.

As future work we plan to extend QCFLP$(\mathcal{D}, \mathcal{C})$ and the program transformation in order to provide explicit support for similarity-based reasoning, as well as the higher-order programming features available in CFLP$(\mathcal{C})$. We also plan to analyze the complexity of the program transformation and to embed it as part of an enhanced version of the $\mathcal{TOY}$ system. Finally, we plan further research on flexible information retrieval applications, using different instances of our scheme.

References

1. Antoy, S., Echahed, R., Hanus, M.: A needed narrowing strategy. Journal of the ACM 47(4), 776–822 (2000)
2. Arenas, P., Fernández, A.J., Gil, A., López-Fraguas, F.J., Rodríguez-Artalejo, M., Sáenz-Pérez, F.: $\mathcal{TOY}$, a multiparadigm declarative language. version 2.3.1. Caballero, R., Sánchez, J. (eds.) (2007), `http://toy.sourceforge.net`
3. Caballero, R., Rodríguez-Artalejo, M., Romero-Díaz, C.A.: Similarity-based reasoning in qualified logic programming. In: PPDP 2008: Proceedings of the 10th international ACM SIGPLAN conference on Principles and Practice of Declarative Programming, pp. 185–194. ACM, New York (2008)
4. Caballero, R., Rodríguez-Artalejo, M., Romero-Díaz, C.A.: A generic scheme for qualified constraint functional logic programming. Technical Report SIC-1-09, Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain (2009), `http://gpd.sip.ucm.es/cromdia/works`
5. del Vado Vírseda, R.: Declarative constraint programming with definitional trees. In: Gramlich, B. (ed.) FroCos 2005. LNCS, vol. 3717, pp. 184–199. Springer, Heidelberg (2005)
6. Guadarrama, S., Muñoz, S., Vaucheret, C.: Fuzzy prolog: A new approach using soft constraint propagation. Fuzzy Sets and Systems 144(1), 127–150 (2004)
7. Hanus, M.: Curry: an integrated functional logic language, version 0.8.2. Hanus, M. (ed.) (2006), `http://www.informatik.uni-kiel.de/~curry/report.html`
8. López-Fraguas, F.J., Rodríguez-Artalejo, M., del Vado-Virseda, R.: A lazy narrowing calculus for declarative constraint programming. In: Proceedings of the 6th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP 2004), pp. 43–54. ACM Press, New York (2004)
9. López-Fraguas, F.J., Rodríguez-Artalejo, M., del Vado-Vírseda, R.: A new generic scheme for functional logic programming with constraints. Journal of Higher-Order and Symbolic Computation 20(1-2), 73–122 (2007)
10. Moreno, G., Pascual, V.: Formal properties of needed narrowing with similarity relations. Electronic Notes in Theoretical Computer Science 188, 21–35 (2007)
11. Riezler, S.: Quantitative constraint logic programming for weighted grammar applications. In: Retoré, C. (ed.) LACL 1996. LNCS, vol. 1328, pp. 346–365. Springer, Heidelberg (1997)
12. Riezler, S.: Probabilistic Constraint Logic Programming. PhD thesis, Neuphilologischen Fakultät del Universität Tübingen (1998)
13. Rodríguez-Artalejo, M.: Functional and constraint logic programming. In: Comon, H., Marché, C., Treinen, R. (eds.) CCL 1999. LNCS, vol. 2002, pp. 202–270. Springer, Heidelberg (2001)
14. Rodríguez-Artalejo, M., Romero-Díaz, C.A.: Quantitative logic programming revisited. In: Garrigue, J., Hermenegildo, M.V. (eds.) FLOPS 2008. LNCS, vol. 4989, pp. 272–288. Springer, Heidelberg (2008)
15. Sessa, M.I.: Approximate reasoning by similarity-based SLD resolution. Theoretical Computer Science 275(1-2), 389–426 (2002)
16. Subrahmanian, V.S.: Uncertainty in logic programming: Some recollections. Association for Logic Programming Newsletter 20(2) (2007)