A Framework for Goal-Directed Bottom-Up Evaluation of Functional Logic Programs

Jesús M. Almendros-Jiménez* and Antonio Becerra-Terón
Dpto. de Lenguajes y Computación. Universidad de Almería.
04120-Almería. Spain. `email: {jalmen,abecerra}@ual.es`

Abstract. In this paper we start the design of a functional-logic deductive database language. Given that most logic deductive languages consider bottom-up evaluation as operational mechanism, here we will focus on the development of an operational semantics based on bottom-up evaluation for functional logic languages. As in the logic paradigm, the bottom-up evaluation will consist in a magic transformation for a given program-query into a magic program-query for which the bottom-up evaluation will simulate the top-down one of the original program.

1 Introduction

Deductive databases are database management systems whose query language and, usually, storage structure are designed around a logical data model. It is easy to see deductive database systems as an advanced form of relational systems, and they are best suited for applications in which a large amount of data must be accessed and complex queries must be supported. Deductive databases languages [21] have been influenced by work in logic programming and offer a rich query language, which extends *SQL* in many important directions (including support for *aggregation*, *negation*, and *recursion*). Deductive databases languages use logic programming concepts but do not share in most cases the evaluation mechanism of traditional logic programming languages as Prolog. There are three main reasons for it:

- Prolog's *depth-first* evaluation strategy leads to *infinite loops*, even for definite programs and even in the absence of function symbols or arithmetic. In the presence of large volumes of data, operational reasoning is not desirable, and a higher premium is placed upon *completeness* and *termination* of the evaluation method.
- In a typical database application, the amount of data is sufficiently large that much of it is on *secondary storage*. Prolog systems evaluate logic programs efficiently in *main-memory*, but are *tuple-at-a-time*; that is, they process one (sub) goal at a time, and thus inefficiently w.r.t. disk accesses. Efficient access to these data is crucial to good performance.
- When evaluating a database query, it is customary to want to compute *all of the answers* to the query. When writing a program for computation purposes, such as a typical Prolog program, it is common to only need *one answer* to the query.

* The author has been partially supported by the Spanish CICYT (project TIC 98-0445-C03-02 TREND)

One consequence of this is that most deductive database systems (for instance, DATALOG [25], CORAL [20], ADITI [26]) use *bottom-up* evaluation methods instead of *top-down* one. Bottom-up approach allows us to use *set-at-a-time* evaluation, i.e. it processes sets of goals, rather than proceeding one (sub) goal at a time, where operations like relational joins can be made for disk-resident data efficiently. In this sense, deductive systems are an attempt to adapt Prolog, which has a "small-data" view of the world, to a "large-data" world. On the other hand, the bottom-up evaluation avoids infinite loops over programs under certain conditions.

An important problem is that a query asks not for the entire relation corresponding to an intensional predicate but for a small subset. It is important that we answer a query by examining only the part of database that involves the predicates relevant to the instantiated or partially instantiated arguments in the query. The goal-directed bottom-up evaluation generates the subset of the *Herbrand model* of the program relevant to the query. With this aim, the bottom-up evaluation in such languages involves a query-program transformation termed *Magic Sets* [5].

The basic idea of this technique is that a logic program-query is transformed into a *magic* logic program-query whose bottom-up *fixed point evaluation* [3] is devised to simulate top-down evaluation of the original program and query. The program is evaluated using bottom-up evaluation until no new facts are generated or the answer to the query is found. The transformed program adds new predicates, called *magic predicates*, whose role is to pass information (instantiated and partially instantiated arguments in the predicates of the query) to the program in order to consider only those instances of the program rules relevant to the query solving.

Example 1. Given a set of facts for a relation `par` and the following logic rules:

```
anc(X,Y):-par(X,Y).
anc(X,Y):-par(X,Z),anc(Z,Y).
```

and a query `anc(john,X)`, the magic-sets transformation rewrites this program into the following logic program:

```
anc(X,Y):-mg_anc(X),par(X,Y).
anc(X,Y):-mg_anc(X),par(X,Z),anc(Z,Y).
mg_anc(Z):-mg_anc(X),par(X,Z).
mg_anc(john).
```

together with `par(`α`,`β`):-mg_par(`α`,`β`)` for each pair α,β in the relation `par`. The fixed point evaluation computes only the ancestors of `john` (and those intermediate results needed for it).

Several transformation magic methods have been widely studied in the past with the aim to deal with *recursive queries*, *non-ground facts*, *function symbols*, *partially instantiated terms*, *negation* and to provide optimizations such as avoiding the computation of *duplicated facts*, and the memoization of *intermediate results.*

As examples, we have the transformations called *Generalized Magis Sets* [5, 12], *Generalized Supplementary Magic Sets* [5], *Magic Templates* [19], *Alexander Templates* [24], *Counting Methods* [23] and recently *Induced Magic Sets* [6].

Although bottom-up evaluation based on magic-sets has been studied in the context of logic programming, there are some approaches (for instance [13]) which consider bottom-up evaluation of *constraint logic programs*, showing a new research area in the application of the CLP scheme for databases in the framework of *constraint query languages* [11]. Moreover, the bottom-up evaluation has been used in the context of semantic-based program analysis (for instance [4]).

Deductive databases have also been studied in a functional programming context (PFL [22], FDL [18]), enjoying the three following advantages, *ease of reasoning*: the reasoning can be used to transform queries into a more efficient form or to prove that transactions preserve consistency constraints; *freedom from a detailed execution order*: this freedom has been used to improve query evaluation by selecting an appropriate execution order for subqueries, in such a way that this freedom makes parallel execution natural in functional database query languages; and *freedom from side-effects*: because side-effects may not be anticipated by the programmer, they complicate the programmer's task, especially in large database applications.

On the other hand, the integration of functional and logic programming has been widely investigated during the last years, see [8] for a survey. It has leaded to the recent design of modern programming languages such as CURRY [9] and $\mathcal{TOY}$ [16] following the ideas of the predecessors BABEL and K-LEAF. The aim of such integration is to include features from functional (cfr. determinism, higher order functions, partial, non strict and lazy functions, possibly infinite data structures) and logic languages (cfr. logic variables, function inversion, non-determinism, built-in search). The basic ideas in functional-logic programming consist in *lazy narrowing* as operational mechanism, following some class of *narrowing strategy* [2, 14] combined with some kind of *constraints solving* [15] and *higher order* features [10].

The aim of this paper is to study a functional-logic deductive database language. Given that most logic deductive languages consider bottom-up evaluation as operational mechanism, here we focus on the development of an operational semantics based on bottom-up evaluation for functional logic languages. As in the logic paradigm, the bottom-up evaluation will consist in a magic transformation for a given program-query into a magic program-query for which the bottom-up evaluation will simulate the top-down one of the original program.

With respect to logic programming paradigm, we have to solve the old and new problems. In the earliest logic deductive databases, logic programs were constrained to be free of function-symbols, facts were grounds and deduction rules were well-formed, i.e. each variable that appears in the head of the rule also appears in its body [5]. These conditions ensure termination of the bottom-up evaluation based on magic-sets given that the least Herbrand model [3] is finite in the original and magic program, and thus bottom-up is a complete deduction procedure whereas Prolog is not.

Example 2. In the following logic program:

```
p(X):-p(X).
p(0).
```

the Prolog solving of the goal `p(0)` loops and no answer is found, however the bottom-up does not; the evaluation of the transformed magic program:

```
p(X):-mg_p(X),p(X).
p(0):-mg_p(0).
mg_p(0).
```

where `mg_p` is the magic predicate associated to `p`, ends and obtains `p(0)` as fact.

Later [19], the well-formed condition was removed but no general result of termination for bottom-up evaluation was presented by allowing function-symbols (and therefore for general Horn-logic programs), although some attempts were done [24, 19]. In the presence of function symbols, the Herbrand model of a logic program can be infinite and thus, in the general case, the bottom-up evaluation of the original and transformed program cannot end. Therefore, the introduction of function symbols causes uncompleteness for negative information.

In summary, the main aims of such works were to find a bottom-up evaluation method for restricted or general Horn-logic programs which simulates top-down evaluation, that is, a goal-directed and efficient evaluation method (in terms of intermediates facts (subgoals) that are generated [24, 6]) which retains the advantages of avoiding infinite loops and set-at-a-time computations. They also study semantic models and magic transformations for deductive database languages with negation [12]. Thus, we may say that the bottom-up evaluation is better than top-down one w.r.t. positive goals: bottom-up evaluation is a complete deduction procedure when top-down is not. In the presence of function symbols and negative goals, both bottom-up and top-down evaluation with negative goals are, in general, uncomplete procedures.

In our case, we will consider a functional logic language like $\mathcal{TOY}$ [7], that is, our programs will be conditional constructor-based rewriting rules in which no additional restrictions in the form of program rules will be required. Therefore, Horn-logic programs will be considered as particular cases of our functional-logic programs. It means that we will have the same problems due to the introduction of function symbols in deductive logic programs (called constructors in a functional-logic context). Given that in this paper we do not deal with negation, we will now not care about it. We are interested in the definition of an operational semantics for $\mathcal{TOY}$ programs based on bottom-up evaluation which simulates the top-down one of a $\mathcal{TOY}$ program. This bottom-up evaluation will be as goal-directed as top-down evaluation, that is, it generates the same intermediate results than the top-down evaluation, retaining the advantages of bottom-up evaluation. We will consider the standard magic transformation with left-to-right information passing strategy [5].

As new contributions of this paper, we have the dealing with functions instead of logic predicates, which can be lazy, partial and non-strict, and the dealing with possible infinite data, which introduces new problems in the bottom-up evaluation once we have to apply program rules lazily in every step of the fixed point operator application. The laziness of the evaluation will be driven by the goal evaluating program rules (arguments and conditions) as far as the goal requires. In this sense, our bottom-up evaluation is similar to the demand-driven [14] top-down evaluation of functional logic programs. As far as we know, this is the first time that such kind of evaluation method for functional logic languages has been presented. To put an end to the comparison of bottom-up and top-down evaluations in our framework, we will establish the equivalence results of our bottom-up evaluation and the conditional rewriting logic (CRWL) [7] which provides logic foundations to the $\mathcal{TOY}$ language, and therefore ensuring soundness and completeness of our bottom-up evaluation. Moreover, we will establish correspondences among proofs of a given goal in the cited logic and the "facts" computed by means of the bottom-up evaluation showing the optimality of our evaluation method.

The rest of the paper will be organized as follows. In section 2 we will introduce basic notions that will be used in the rest of the paper; in section 3 we present the language; the section 4 will define Herbrand models for our language; section 5 will present the magic transformation; section 6 will show the results of soundness, completeness and optimality and finally in section 7 we will describe the future work and conclusions. Due to the lack of space, the proofs of our results have not been included in this version but the full proofs can be found in [1].

2 Basic Notions

We assume the reader has familiarity with basic concepts of model theory on logic programming and functional-logic programming (see [3, 17, 7] for more details). We now point up some of the notions used in this paper.

Given S, a *partially ordered set* (in short, *poset*) with *bottom* $\perp$ (equipped with a partial order $\leq$ and a least element $\perp$), the set of all totally defined elements of S will be noted *Def(S)*. We write $\mathcal{C}(S)$, $\mathcal{I}(S)$ for the sets of cones and ideals of S respectively. The set $\bar{S} =_{def} \mathcal{I}(S)$ denotes the *ideal completion* of S, which is also a *poset* under the *set-inclusion* ordering $\subseteq$, and there is a natural order which maps each $x \in S$ into the principal ideal generated by x, $< x >=_{def} \{y \in S : y \leq x\} \in \bar{S}$. Furthermore, $\bar{S}$ is an algebraic *cpo* whose finite elements are precisely the principal ideals $< x >,\ x \in S$.

A *signature* is a set $\sum = \mathcal{C} \cup \mathcal{F}$, where $\mathcal{C} = \bigcup_{n \in I\!N} \mathcal{C}^n$ and $\mathcal{F} = \bigcup_{n \in I\!N} \mathcal{F}^n$ are disjoint sets of constructor and function symbols respectively, each of them with associated *arity*. $Expr_{\Sigma}$ and $Term_{\Sigma}$ denote the set of expressions built up from $\sum$ and a set of variables $\mathcal{V}$, and the subset of terms making only use of $\mathcal{C}$ and $\mathcal{V}$, respectively. We distinguish partial $Expr_{\Sigma_\perp}$ (resp. $Term_{\Sigma_\perp}$) and total expressions (resp. terms) depending whether they include $\perp$ or not. The *approximation ordering* $\leq$ for $Term_{\Sigma_\perp}$ can be defined as the least partial ordering satisfying the following properties: $\perp \leq t$, for every t, and if $t_1 \leq s_1, ..., t_n \leq s_n$ then $c(t_1, ..., t_n) \leq c(s_1, ..., s_n)$, for every $c \in \mathcal{C}^n$.

Substitutions are mappings $\theta : \mathcal{V} \rightarrow Term_{\Sigma_\perp}$ which have $\theta : Term_{\Sigma_\perp} \rightarrow Term_{\Sigma_\perp}$ as unique natural extension, also noted as θ. $Subst_{\Sigma}$ (resp. $Subst_{\Sigma_\perp}$) denotes the set of substitutions in total (resp. possibly partial) terms. We note as $t\theta$ the result of applying the substitutions θ to the term t.

3 A Functional Logic Language

A *program* $\mathcal{P}$ is a signature Σ together with a set of conditional constructor-based rewrite rules of the form: $f(t_1, \ldots, t_n) := r \Leftarrow C$ where $f \in \mathcal{F}$ of arity n,

$\bar{t}$ must be a linear tuple of terms $t_i \in Term_\Sigma$, and the condition C must consist of finitely many (possibly zero) of strict equalities $e == e'$ with $e,\ e' \in Expr_\Sigma$. $[\mathcal{P}] = \{(l := r \Leftarrow C)\ \theta \mid (l := r \Leftarrow C) \in \mathcal{P},\ \theta \in Subst_{\Sigma_\perp}\}$ denotes the set of (possibly partial) instances of the program rules of $\mathcal{P}$. We allow extra variables in the body and the condition. In fact, our rules can be used to define non-deterministic functions. A goal $\mathcal{G}$ is like the conditional part of a program rule.

Now we present the semantics of a language through a conditional rewriting logic (CRWL) as in [7]. This logic allows us to prove statements of the form $\mathcal{P} \vdash_{\mathcal{CRWL}} e \to t$, with $e \in Expr_{\Sigma_\perp}, t \in Term_{\Sigma_\perp}$, called *non-strict equalities*, whose meaning is that term t approximates the value of the expression e, and *strict equalities* $\mathcal{P} \vdash_{\mathcal{CRWL}} e == e',\ e, e' \in Expr_{\Sigma_\perp}$ whose meaning is that e and e' represent the same totally defined value. A *solution* for a goal $\mathcal{G}$ w.r.t. a program $\mathcal{P}$ is a substitution $\theta \in Subst_{\Sigma_\perp}$ such that $\mathcal{P} \vdash_{\mathcal{CRWL}} \mathcal{G}\theta$. The formal presentation of CRWL is as follows:

Definition 1 (Conditional Rewriting Logic (CRWL)).

(B) $\dfrac{}{e \to \perp}$

(R) $\dfrac{}{X \to X}\ X \in \mathcal{V}$ **(DC)** $\dfrac{e_1 \to t_1\ \dots\ e_n \to t_n}{c(e_1,\ \dots\ , e_n) \to c(t_1,\ \dots\ , t_n)}\ c \in \mathcal{C}^n$

(O) $\dfrac{e_1 \to t_1\ \dots\ e_n \to t_n \quad C \quad r \to t}{f(e_1,\ \dots\ , e_n) \to t}$ **(S)** $\dfrac{e \to t \quad e' \to t}{e\ ==\ e'}\ t \in Term_\Sigma$

$t \not\equiv \perp,\ f(t_1,\ \dots\ , t_n) := r \Leftarrow C \in [\mathcal{P}]$

4 Herbrand Models

4.1 CRWL-Algebras

In this section we present Herbrand algebras and models as a particular case of CRWL-algebras. Firstly, we need the following definitions.

Definition 2 (Non-deterministic and Deterministic Functions). *Given two posets D and E with $\perp$ we define:*

- *The set of all non-deterministic functions from D to E as:*
  $[D \to_n E] =_{def}\ \{f : D \to \mathcal{C}(E) \mid \forall\ u, u' \in D : (u \le u' \Rightarrow f(u) \subseteq f(u'))\}$
- *The set of all deterministic functions from D to E as:*
  $[D \to_d E] =_{def}\ \{f \in [D \to_n E] \mid \forall\ u \in D : f(u) \in \mathcal{I}(E)\}$

Now we can define the class of algebras which will be used as models for CRWL:

Definition 3 (CRWL-Algebras). *For any given signature, CRWL-algebras are algebraic structures of the form: $\mathcal{A} = (D_\mathcal{A}, \{c^\mathcal{A}\}_{c \in \mathcal{C}}, \{f^\mathcal{A}\}_{f \in \mathcal{F}})$ where $D_\mathcal{A}$ is a poset, $c^\mathcal{A} \in [D^n_\mathcal{A} \to_d D_\mathcal{A}]$ for $c \in \mathcal{C}^n$, and $f^\mathcal{A} \in [D^n_\mathcal{A} \to_n D_\mathcal{A}]$ for $f \in \mathcal{F}^n$. For $c_\mathcal{A}$ we still require the following additional condition: for all $u_1, \dots, u_n \in D_\mathcal{A}$ there is $v \in D_\mathcal{A}$ such that $c^\mathcal{A}(u_1, \dots, u_n) =< v >$. Moreover, $v \in Def(D_\mathcal{A})$ in case that all $u_i \in Def(D_\mathcal{A})$*

The next definition shows how to evaluate expressions in CRWL-algebras:

Definition 4 (Expression evaluation). *Let $\mathcal{A}$ be a CRWL-algebra of signature $\sum$. A evaluation over $\mathcal{A}$ is any mapping $\eta : \mathcal{V} \to \mathcal{D}_\mathcal{A}$, and we say that η is totally defined iff $\eta(X) \in Def(D_\mathcal{A})$ for all $X \in \mathcal{V}$. We denote by $Val(\mathcal{A})$ the set of all valuations, and by $DefVal(\mathcal{A})$ the set of all totally defined valuations. The evaluation of $e \in Expr_{\Sigma_\perp}$ in $\mathcal{A}$ under η yields $[\![e]\!]^\mathcal{A}\eta \in \mathcal{C}(D_\mathcal{A})$ which is defined recursively as follows:*

- $[\![\perp]\!]^{\mathcal{A}}\eta =_{def} < \perp_{\mathcal{A}} >$.
- $[\![X]\!]^{\mathcal{A}}\eta =_{def} < \eta(X) >$, *for* $X \in \mathcal{V}$.
- $[\![h(e_1, \ \dots\ , e_n)]\!]^{\mathcal{A}}\eta =_{def} h^{\mathcal{A}}([\![e_1]\!]^{\mathcal{A}}\eta, \ \dots\ , [\![e_n]\!]^{\mathcal{A}}\eta)$, *for all* $h \in \mathcal{C}^n \cup \mathcal{F}^n$.

Due to non-determinism, the evaluation of an expression yields a cone rather than an element. However, this cone can still represent an element (in the ideal completion) in the case that it is an ideal. It can be proved that given a CRWL-algebra $\mathcal{A}$, for any $e \in Expr_{\Sigma_\perp}$ and any $\eta \in Val(\mathcal{A})$ then $[\![e]\!]^{\mathcal{A}}\eta \in \mathcal{C}(D_{\mathcal{A}})$, $[\![e]\!]^{\mathcal{A}}\eta \in \mathcal{I}(D_{\mathcal{A}})$ if $f^{\mathcal{A}}$ is deterministic for every defined function symbol f occurring in e, and $[\![e]\!]^{\mathcal{A}}\eta =< v >$ for some $v \in D_{\mathcal{A}}$, if $e \in Term_{\Sigma_\perp}$ and $v \in Def(D_{\mathcal{A}})$ if $e \in Term_{\Sigma}$ and $\eta \in DefVal(\mathcal{A})$. Moreover, given a valuation over a CRWL-algebra $\mathcal{A}$, for any $e \in Expr$ and any $\theta \in Subst$ we have $[\![e\theta]\!]^{\mathcal{A}}\eta = [\![e]\!]^{\mathcal{A}}\eta\theta$, where $\eta\theta$ is the uniquely determined valuation that satisfies $< \eta\theta(X) >= [\![X\theta]\!]^{\mathcal{A}}\eta$ *for all* $X \in \mathcal{V}$.

4.2 Herbrand Models

Now we can introduce Herbrand models. The main ideas are to interpret non-strict equalities in $\mathcal{A}$ as inclusions among cones and to interpret strict equalities as asserting the existence of some common, totally defined approximation.

Definition 5 (Models). *Assume a program $\mathcal{P}$ and a CRWL-algebra $\mathcal{A}$. We define:*

- *$\mathcal{A}$ satisfies a non-strict equality $e \to e'$ under a valuation η (in symbols, $(\mathcal{A}, \eta) \models e \to e'$) iff $[\![e]\!]^{\mathcal{A}}\eta \supseteq [\![e']\!]^{\mathcal{A}}\eta$.*
- *$\mathcal{A}$ satisfies a strict equality $e == e'$ under a valuation η (in symbols, $(\mathcal{A}, \eta) \models e == e'$) iff $[\![e]\!]^{\mathcal{A}}\eta \bigcap [\![e']\!]^{\mathcal{A}}\eta \bigcap Def(D_{\mathcal{A}}) \neq \emptyset$.*
- *$\mathcal{A}$ satisfies a rule $l := r \Leftarrow C$ iff every valuation η such that $(\mathcal{A}, \eta) \models C$ verifies: $(\mathcal{A}, \eta) \models l \to r$.*
- *$\mathcal{A}$ is a model of $\mathcal{P}$ (in symbols, $\mathcal{A} \models \mathcal{P}$) iff $\mathcal{A}$ satisfies all the rules in $\mathcal{P}$.*

Theorem 1 (Soundness). *For any program $\mathcal{P}$ and any non-strict or strict equality φ: $\mathcal{P} \vdash_{\mathcal{CRWL}} \varphi \Rightarrow (\mathcal{A}, \eta) \models \varphi$, for all $\mathcal{A} \models \mathcal{P}$ and all $\eta \in DefVal(\mathcal{A})$.*

Now we define Herbrand algebras as a particular case of CRWL-algebras in which the poset is the set of terms partially ordered by $\leq$ and the interpretation of the constructors consists of the ideal generated for the represented term.

Definition 6 (Herbrand Algebras). *Given a program $\mathcal{P}$, a Herbrand Algebra $\mathcal{A}$ is defined as follows. $D_{\mathcal{A}}$ is the poset $Term_{\Sigma_\perp}$ with the approximation ordering $\leq$ and $c^{\mathcal{A}}(t_1, \ \dots\ , t_n) =_{def} < c(t_1, \ \dots\ , t_n) >$ (principal ideal), for all $t_i \in Term_{\Sigma_\perp}$.*

From now on Val_H denotes the valuations in Herbrand algebras.

Definition 7 (Poset of Herbrand Algebras). *Given a program $\mathcal{P}$ we define the poset CRWLH of Herbrand algebras as follows: $\mathcal{A} \leq \mathcal{B}$ iff $f^{\mathcal{A}}(t_1, \dots, t_n) \subseteq f^{\mathcal{B}}(t_1, \dots, t_n)$ for every $f \in \mathcal{F}$ and $t_i \in Term_{\Sigma_\perp}, 1 \leq i \leq n$.*

Moreover, we can prove that the ideal completion of *CRWLH* is a cpo, and $[\![\ ]\!]$ is continuous w.r.t. the ideal completion of *CRWLH* and Val_H.

Definition 8 (Fix Point Operator). *Given $\mathcal{A} \in CRWLH$, $f \in \mathcal{F}$, we define the fix point operator as:*

$T_{\mathcal{P}}(\mathcal{A}, f)(s_1, \dots, s_n) =_{def} \{[\![r]\!]^{\mathcal{A}}_{\eta} \mid$ *there exists* $f(\bar{t}) := r \Leftarrow C \in \mathcal{P}$, *and* $\eta \in Val(\mathcal{A})$ *such that* $[\![t_i]\!]^{\mathcal{A}}_{\eta} = s_i, (\mathcal{A}, \eta) \models C\}$

Proposition 1. *Given $\mathcal{A} \in CRWLH$ there exists a unique $\mathcal{B} \in CRWLH$ denoted by $T_{\mathcal{P}}(\mathcal{A})$ such that $f^{\mathcal{B}}(t_1, \ldots, t_n) = T_P(\mathcal{A}, f)(t_1, \ldots, t_n)$ for every $f \in \mathcal{F}$ and $t_i \in Term_{\Sigma_\perp}, 1 \leq i \leq n$.*

With these definitions, we can ensure the following result which characterizes the least Herbrand model.

Theorem 2. *The fix point operator $T_{\mathcal{P}}$ is continuous and satisfies:*

1. *For every $\mathcal{A} \in CRWLH$: $\mathcal{A} \models \mathcal{P}$ iff $T_P(\mathcal{A}) \leq \mathcal{A}$.*
2. *$T_{\mathcal{P}}$ has a least fix point $\mathcal{M}_{\mathcal{P}} = \sqcup_{k \geq 0} {\mathcal{H}_{\mathcal{P}}}^k$ where ${\mathcal{H}_{\mathcal{P}}}^0$ is the bottom in CRWLH and ${\mathcal{H}_{\mathcal{P}}}^{k+1} = T_{\mathcal{P}}({\mathcal{H}_{\mathcal{P}}}^k)$*
3. *$\mathcal{M}_{\mathcal{P}}$ is the least Herbrand model of $\mathcal{P}$.*

Moreover we can see that satisfaction in $\mathcal{M}_{\mathcal{P}}$ can be characterized in terms of $\vdash_{\mathcal{CRWL}}$ provability.

Lemma 1 (Characterization Lemma). *Let id be the identity valuation over $\mathcal{M}_{\mathcal{P}}$, defined by $\mathsf{id}(X) = X$ for all $X \in \mathcal{V}$. For any non-strict or strict equality φ, we have $(\mathcal{M}_{\mathcal{P}}, \mathsf{id}) \models \varphi \Leftrightarrow \mathcal{P} \vdash_{\mathcal{CRWL}} \varphi$.*

As a consequence of the Characterization Lemma, we also get that for any substitution $\theta \in Subst_{\Sigma_\perp}$ (which is also a valuation over $\mathcal{M}_{\mathcal{P}}$) and any non-strict or strict equality φ, we have $(\mathcal{M}_{\mathcal{P}}, \theta) \models \varphi \Leftrightarrow \mathcal{P} \vdash_{\mathcal{CRWL}} \varphi\theta$.

Theorem 3 (Adequateness of $\mathcal{M}_{\mathcal{P}}$). *$\mathcal{M}_{\mathcal{P}}$ is a model of $\mathcal{P}$, for any non-strict or strict equality φ, the following conditions are equivalent:*

a) $\mathcal{P} \vdash_{\mathcal{CRWL}} \varphi$.
b) $(\mathcal{A}, \eta) \models \varphi$ for every $\mathcal{A} \models \mathcal{P}$, and every $\eta \in DefVal(\mathcal{A})$.
c) $(\mathcal{M}_{\mathcal{P}}, \mathsf{id}) \models \varphi$, where id is the identity valuation.

Note that the completeness of $\vdash_{\mathcal{CRWL}}$ also follows from Theorem 3. According to this result, $\mathcal{M}_{\mathcal{P}}$ can be regarded as the intended *(canonical) model of program* $\mathcal{P}$. In particular, a given $f \in \mathcal{F}$ will denote a deterministic function iff $f^{\mathcal{M}_{\mathcal{P}}}(t_1, \ \ldots \ , t_n)$ is an ideal for all $t_i \in Term_{\Sigma_\perp}$.

5 On the Power of Magic

In this section we will present the basic ideas of the bottom-up evaluation method for our functional logic language. The main idea in the goal solving of a lazy functional logic language under a top-down evaluation, is to select a subgoal (strict equation) at a time and to reduce, by applying narrowing steps, every side of the equation as far as needed until terms in both sides are found. The narrowing steps involve, as it is expressed in CRWL, to select a program rule for the outermost function symbol, to solve lazily non-strict equations as parameter passing in the function's calling and to solve the subgoals of the conditions of the applied rule.

Bottom-up evaluation will solve goals obtaining by means of the fix point operator a Herbrand algebra in which every strict equality in the goal is satisfied. Like logic programming, the so-called *passing magic (boolean) functions* will activate the evaluation of the functions through the fix point operator, whenever there exists a call, passing the arguments from the head to the body and conditions of every program rule for them. In every step of the fix-point operator application, new approximations to the value (or values, due to the non-determinism) of every function are computed.

Table 1. Example 1

```
(1) f(s(s(A)),B) := B ⇐ h(A,C) == 0, p(C) == s(0), t(0) == 0.
(2) g(s(D)) := s(g(D)).
(3) h(s(E),s(0)) := 0.
(4) p(s(I)) := s(p(I)).
(5) p(0) := 0.
(6) t(0) := 0.
(7) t(s(0)) := 0.
(8) k := 0.
(9) k := s(0).
(10) k := s(s(0)).
```

Let the program example in table 1[1] and the goal `f(g(X),Y)==t(k)` be, wherein program rules with conditions, terminating and non-terminating recursion, non-strictness, and non-determinism are mixed.

In a top-down evaluation, `f(g(X),Y)` is narrowed by applying a program rule for `f` like **(1)** shown in table 1. However, `g(X)` must be lazily unified previously with `s(s(A))`, and `Y` with `B` respectively. Once unified, the bindings for `A` and `B` instantiate the conditions ⇐ `h(A,C) == 0`, `p(C) == s(0)`, `t(0) == 0` which became subgoals; similarly with `t(k)`.

In functional-logic bottom-up evaluation the passing magic function for `f`, named $\texttt{mg\_f}^{\texttt{P}}$, activates the evaluation of the body and the conditions for the binding obtained by the lazy unification of `g(X)` with `s(s(A))` and `Y` with `B`, and therefore obtaining approximations for (instances of) `f(g(X),Y)` by means of the fix-point operator. Similarly with `t(k)`. The approximations computed for (instances of) `f(g(X),Y)` and `t(k)` are compared in every step of the fix-point operator application and the goal solving successes every time that some (due to non-determinism) of the approximations agrees, for a particular case of the goal, in a total term.

The magic transformation transforms every pair ($\mathcal{P}$,$\mathcal{G}$), where $\mathcal{P}$ is a program and $\mathcal{G}$ is a goal, into a pair ($\mathcal{P}^{\mathcal{MG}}$,$\mathcal{G}^{\mathcal{MG}}$) in such a way that the transformed program, evaluated by means of the fix point operator, computes the same solutions θ of the goal $\mathcal{G}$ w.r.t. the program $\mathcal{P}$.

The transformation process needs the use, in some cases like in our running example, of auxiliary intermediate functions whenever there exist nested constructors occurring in the head of the rules; for instance, `f(s(s(A)),B)` forces to use an auxiliary intermediate function, called $\texttt{f}_1$, and to add the rules $\texttt{f(s(A),B)} := \texttt{f}_1\texttt{(A,B)}$ and $\texttt{f}_1\texttt{(s(C),D)} := \texttt{D}$ replacing **(1)** shown in the table 1 which preserves the semantics of `f` in the original program.

Once the quoted intermediate functions have been introduced, the magic transformation transforms the program, aided by the passing magic boolean functions $\texttt{mg\_f}^{\texttt{P}}$, $\texttt{mg\_f}_1{}^{\texttt{P}}$, $\texttt{mg\_g}^{\texttt{P}}$, etc., into a filtered program (rules **(1)-(11)** shown in table 2) where passing magic functions $mg\_f^P(\bar{t}) == true$ will filter the fix point evaluation for those functions needed by the goal solving. The passing magic functions have its own program rules (see rules **(12)-(16)** shown in table 2) activating the evaluation of the body and the left-to-right evaluation of the conditions. In our example, the passing magic function for `f` will activate the evaluation of $\texttt{f}_1$ by rule **(1)**, the magic function for $\texttt{f}_1$ will activate the evaluation

[1] We rename program rules in our examples.

Table 2. Example 1

Filtered Rules
(**1**) $\mathtt{f(s(A),B) := f_1(A,B) \Leftarrow mg\_f^P(s(A),B) == true.}$
(**2**) $\mathtt{f_1(s(C),D) := D \Leftarrow mg\_f_1^P(s(C),D) == true, h(C,E) == 0,}$
$\mathtt{p(E) == s(0), t(0) == 0.}$
(**3**) $\mathtt{g(s(F)) := s(g(F)) \Leftarrow mg\_g^P(s(F)) == true.}$
(**4**) $\mathtt{h(s(G),s(0)) := 0 \Leftarrow mg\_h^P(s(G),s(0)) == true.}$
(**5**) $\mathtt{p(s(H)) := s(p(H)) \Leftarrow mg\_p^P(s(H)) = true.}$
(**6**) $\mathtt{p(0) := 0 \Leftarrow mg\_p^P(0) == true.}$
(**7**) $\mathtt{t(0) := 0 \Leftarrow mg\_t^P(0) == true.}$
(**8**) $\mathtt{t(s(0)) := 0 \Leftarrow mg\_t^P(s(0)) == true.}$
(**9**) $\mathtt{k := 0 \Leftarrow mg\_k^P == true.}$
(**10**) $\mathtt{k := s(0) \Leftarrow mg\_k^P == true.}$
(**11**) $\mathtt{k := s(s(0)) \Leftarrow mg\_k^P == true.}$
Magic Rules
(**12**) $\mathtt{mg\_f_1^P(I,J) := mg\_f^P(s(I),J).}$
(**13**) $\mathtt{mg\_h^P(K,L) := mg\_f_1^P(s(K),L_1).}$
(**14**) $\mathtt{mg\_p^P(M) := mg\_f_1^P(s(N),P) \Leftarrow h(N,M) == 0.}$
(**15**) $\mathtt{mg\_t^P(0) := mg\_f_1^P(s(R),S) \Leftarrow h(R,T) == 0, p(T) == s(0).}$
(**16**) $\mathtt{mg\_p^P(U) := mg\_p^P(s(U)).}$
(**17**) $\mathtt{mg\_f_1^P(s(mg\_g^N(V)),X_1) := mg\_f_1^P(mg\_g^N(s(V)),X_1).}$
(**18**) $\mathtt{mg\_f^P(s(mg\_g^N(Y_1)),Z) := mg\_f^P(mg\_g^N(s(Y_1)),Z).}$
(**19**) $\mathtt{mg\_t^P(0) := mg\_t^P(mg\_k^N).}$
(**20**) $\mathtt{mg\_t^P(s(0)) := mg\_t^P(mg\_k^N).}$
(**21**) $\mathtt{mg\_t^P(s(s(0))) := mg\_t^P(mg\_k^N).}$
(**22**) $\mathtt{mg\_h^P(s(mg\_g^N(A_1)),B_1) := mg\_h^P(mg\_g^N(s(A_1)),B_1).}$
(**23**) $\mathtt{mg\_f^P(mg\_g^N(X),Y) := true.}$
(**24**) $\mathtt{mg\_t^P(mg\_k^N) := true.}$
Goal Solving Rules
(**25**) $\mathtt{f(mg\_g^N(s(C_1)),D_1) := f(s(mg\_g^N(C_1)),D_1) \Leftarrow mg\_f^P(mg\_g^N(s(C_1)),D_1) == true.}$
(**26**) $\mathtt{f_1(mg\_g^N(s(E_1)),F_1) := f_1(s(mg\_g^N(E_1)),F_1) \Leftarrow mg\_f_1^P(mg\_g^N(s(E_1)),F_1) == true.}$
(**27**) $\mathtt{h(mg\_g^N(s(G_1)),H_1) := h(s(mg\_g^N(G_1)),H_1) \Leftarrow mg\_h^P(mg\_g^N(s(G_1)),H_1) == true.}$
(**28**) $\mathtt{t(mg\_k^N) := t(0) \Leftarrow mg\_t^P(mg\_k^N) == true.}$
(**29**) $\mathtt{t(mg\_k^N) := t(s(0)) \Leftarrow mg\_t^P(mg\_k^N) == true.}$
(**30**) $\mathtt{t(mg\_k^N) := t(s(s(0))) \Leftarrow mg\_t^P(mg\_k^N) == true.}$

of `h`, `p` and `t`, as they appear as outermost function symbols of the conditions of the rule **(2)** for $\mathtt{f_1}$ and finally, the magic function for `p` will trigger the recursion in `p` by rule **(5)**. In the general case, the transformation process adds only magic rules for the *outermost function symbols* (set denoted by $outer(\mathcal{P},\mathcal{G})$), which are defined as those ones occurring either in the goal, or in the body and conditions of some program rule of the outermost function symbols, or in the conditions of some program rule of function symbols occurring in the scope of an outermost function symbol.

Secondly, the idea is that whenever a function symbol is in the scope of an outermost function symbol, every program rule for the *inner function symbol* generates a rule for the passing magic function of the outermost one. In our example $\mathtt{g(s(D)) := s(g(D))}$, and $\mathtt{k := 0, k := s(0), k := s(s(0))}$, generate the rules **(17)-(21)** shown in table 2 for $\mathtt{f_1}$, `f`, and `t`, respectively. The head and the body of every program rule of each inner function symbol are "turned around" and introduced as arguments of the head and the body, respectively, of the magic rule for the outermost function symbol, occurring at the same position where they appear in the goal, and filling the rest of arguments with fresh variables. The conditions of the program rules for the inner function symbol, if any, are added as conditions of the magic rule for the outermost one. The inner function symbols are substituted in the magic rules by the so-called *nesting magic constructors*, given that patterns in program rules must be terms.

Moreover, like in our running example, given that g becomes an inner function symbol for h due to the information passing from the first argument of $\texttt{f}_1$ to h in the rule **(2)**, then the rule **(22)** shown in table 2 is also included. In the general case, for every information passing from the head to an outermost function symbol in the body and conditions, a magic rule of this kind for the outermost symbol is added.

Furthermore, the facts **(23)** and **(24)** shown in table 2 are introduced for the given goal, which activate the evaluation of the outermost symbols f and t in the goal. Therefore, the evaluation of the outermost function symbols, which appear in the bodies and conditions of every program rule for f and t, is activated by means of the corresponding magic rules. This is the same class of transformation of a program rule in the particular case of a goal.

Thirdly, the goal `f(g(X),Y)==t(k)` is transformed into a new magic goal $\texttt{f}(\texttt{mg\_g}^{\texttt{N}}(\texttt{X}), \texttt{Y}) == \texttt{t}(\texttt{mg\_k}^{\texttt{N}})$. In the general case, given a goal $\mathcal{G}$, every equation $e == e' \in \mathcal{G}$ is transformed into a new equation wherein each inner function symbol is replaced by nesting magic constructors.

Finally, the rules **(25)-(30)** shown in table 2 must be added. These rules, also named "goal solving rules", allow us to solve the new magic goal whenever the original one has been modified. In our case, the goal solving rules just are the corresponding ones for the magic rules **(17)-(22)**.

Once the program has been transformed, the operational mechanism of the bottom-up evaluation simulates the lazy unification, and the passing of the binding obtained by the unification up to the body and conditions of the program rules. With respect to lazy unification, and starting with the fact **(23)**, the rules **(18)**, **(12)** and **(17)** simulate the evaluation of g for unifying with the first argument of f. The magic rules **(13)**, **(14)** and **(15)** make the information passing up to the conditions of $\texttt{f}_1$. The rule **(22)** allows to evaluate lazily g for unifying with the first argument of h. Therefore g is evaluated three times in order to apply the rule of f. In general, the magic rules allow to pass the (partial) evaluation of the inner function to evaluate the outermost one for the given (partial) result.

Let remark us that given that our transformation cannot bring forward the required narrowing steps, some magic rules are introduced but not used. For instance, let the program $\texttt{f}(\texttt{s}(\texttt{X})) := \texttt{X} \Leftarrow \texttt{h}(\texttt{X}) == \texttt{X}$, $\texttt{h}(\texttt{s}(\texttt{X})) := \texttt{X}$ and $\texttt{g}(0) := \texttt{s}(\texttt{s}(0))$ and the goal $\texttt{f}(\texttt{g}(0)) == 0$ be, the transformation generates $\texttt{mg\_h}^{\texttt{P}}(\texttt{X}) := \texttt{mg\_f}^{\texttt{P}}(\texttt{s}(\texttt{X}))$, $\texttt{mg\_f}^{\texttt{P}}(\texttt{s}(\texttt{s}(0))) := \texttt{mg\_f}^{\texttt{P}}(\texttt{mg\_g}^{\texttt{N}}(0))$, $\texttt{mg\_h}^{\texttt{P}}(\texttt{s}(\texttt{s}(0))) := \texttt{mg\_h}^{\texttt{P}}(\texttt{mg\_g}^{\texttt{N}}(0))$, but given that g only needs to be evaluated once, the last magic rule is never applied, corresponding to lazy evaluation of g.

Note that the fact that every program rule of the inner functions is introduced as argument of the magic rule for the outermost function, is justified as a mechanism of a *demanded driven evaluation* of the outermost function symbol; that is, the inner symbol is evaluated as far as needed to unify lazily with the patterns of every rule of the outermost symbol. The magic transformation passes the inner symbols up to the conditions in order to check if the conditions demand the evaluation of some of the arguments. For instance, let the program $\texttt{h}(\texttt{Y}) := 0 \Leftarrow \texttt{s}(\texttt{Y}) == \texttt{s}(0)$, $\texttt{g}(0) := \texttt{s}(\texttt{f}(0))$ and $\texttt{f}(0) := 0$ and the goal $\texttt{h}(\texttt{g}(0)) == 0$ be,

Table 3. Bottom-Up Evaluation

$\odot\ \mathcal{H}^{0}_{\mathcal{PMG}} = \perp$

$\odot\ \mathcal{H}^{1}_{\mathcal{PMG}} = \mathcal{H}^{0}_{\mathcal{PMG}} \cup \{\mathtt{mg\_f}^{\mathtt{P}}(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{X}),\mathtt{Y}) = \mathtt{true}, \mathtt{mg\_t}^{\mathtt{P}}(\mathtt{mg\_k}^{\mathtt{N}}) = \mathtt{true}\}$

$\odot\ \theta_1 = \{\mathtt{X}/\mathtt{s}(\mathtt{Y}_1), \mathtt{Y}/\mathtt{Z}\}\ \mathcal{H}^{2}_{\mathcal{PMG}} = \mathcal{H}^{1}_{\mathcal{PMG}} \cup \{\mathtt{mg\_f}^{\mathtt{P}}(\mathtt{s}(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{Y}_1)),\mathtt{Z}) = \mathtt{true},$
$\mathtt{mg\_t}^{\mathtt{P}}(0) = \mathtt{true}, \mathtt{mg\_t}^{\mathtt{P}}(\mathtt{s}(0)) = \mathtt{true}, \mathtt{mg\_t}^{\mathtt{P}}(\mathtt{s}(\mathtt{s}(0))) = \mathtt{true}\}$

$\odot\ \theta_2 = \theta_1 \circ \{\mathtt{I}/\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{Y}_1), \mathtt{J}/\mathtt{Z}\}\ \mathcal{H}^{3}_{\mathcal{PMG}} = \mathcal{H}^{2}_{\mathcal{PMG}} \cup \{\mathtt{t}(\mathtt{s}(0)) = 0, \mathtt{t}(0) = 0,$
$\mathtt{mg\_f}^{\mathtt{P}}_{1}(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{Y}_1),\mathtt{Z}) = \mathtt{true}\}$

$\odot\ \theta_3 = \theta_2 \circ \{\mathtt{Y}_1/\mathtt{s}(\mathtt{V}), \mathtt{Z}/\mathtt{X}_1\}\ \ \mathcal{H}^{4}_{\mathcal{PMG}} = \mathcal{H}^{3}_{\mathcal{PMG}} \cup \{\mathtt{mg\_f}^{\mathtt{P}}_{1}(\mathtt{s}(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{V})),\mathtt{X}_1) = \mathtt{true}, \mathtt{t}(\mathtt{mg\_k}^{\mathtt{N}}) = 0\}$

$\odot\ \theta_4 = \theta_3 \circ \{\mathtt{K}/\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{V}), \mathtt{L}_1/\mathtt{X}_1\}\ \mathcal{H}^{5}_{\mathcal{PMG}} = \mathcal{H}^{4}_{\mathcal{PMG}} \cup \{\mathtt{mg\_h}^{\mathtt{P}}(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{V}),\mathtt{L}) = \mathtt{true}\}$

$\odot\ \theta_5 = \theta_4 \circ \{\mathtt{V}/\mathtt{s}(\mathtt{A}_1), \mathtt{L}/\mathtt{B}_1\}\mathcal{H}^{6}_{\mathcal{PMG}} = \mathcal{H}^{5}_{\mathcal{PMG}} \cup \{\mathtt{mg\_h}^{\mathtt{P}}(\mathtt{s}(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{A}_1)),\mathtt{B}_1) = \mathtt{true}\}$

$\odot\ \theta_6 = \theta_5 \circ \{\mathtt{G}/\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{A}_1), \mathtt{B}_1/\mathtt{s}(0)\}\ \mathcal{H}^{7}_{\mathcal{PMG}} = \mathcal{H}^{6}_{\mathcal{PMG}} \cup \{\mathtt{h}(\mathtt{s}(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{A}_1)),\mathtt{s}(0)) = 0\}$

$\odot\ \theta_7 = \theta_6 \circ \{\mathtt{G}_1/\mathtt{A}_1, \mathtt{H}_1/\mathtt{s}(0)\}\mathcal{H}^{8}_{\mathcal{PMG}} = \mathcal{H}^{7}_{\mathcal{PMG}} \cup \{\mathtt{h}(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{s}(\mathtt{A}_1)),\mathtt{s}(0)) = 0\}$

$\odot\ \theta_8 = \theta_7 \circ \{\mathtt{N}/\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{s}(\mathtt{A}_1)), \mathtt{M}/\mathtt{s}(0), \mathtt{P}/\mathtt{X}_1\}\ \mathcal{H}^{9}_{\mathcal{PMG}} = \mathcal{H}^{8}_{\mathcal{PMG}} \cup \{\mathtt{mg\_p}^{\mathtt{P}}(\mathtt{s}(0)) = \mathtt{true}\}$

$\odot\ \theta_9 = \theta_8 \circ \{\mathtt{U}/0\}\ \mathcal{H}^{10}_{\mathcal{PMG}} = \mathcal{H}^{9}_{\mathcal{PMG}} \cup \{\mathtt{mg\_p}^{\mathtt{P}}(0) = \mathtt{true}\}$

$\odot\ \mathcal{H}^{11}_{\mathcal{PMG}} = \mathcal{H}^{10}_{\mathcal{PMG}} \cup \{\mathtt{p}(0) = 0\}$

$\odot\ \theta_{10} = \theta_9 \circ \{\mathtt{H}/0\}\ \mathcal{H}^{12}_{\mathcal{PMG}} = \mathcal{H}^{11}_{\mathcal{PMG}} \cup \{\mathtt{p}(\mathtt{s}(0)) = \mathtt{s}(0)\}$

$\odot\ \theta_{11} = \theta_{10} \circ \{\mathtt{T}/\mathtt{s}(0), \mathtt{R}/\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{s}(\mathtt{A}_1)), \mathtt{S}/\mathtt{X}_1\}\ \mathcal{H}^{13}_{\mathcal{PMG}} = \mathcal{H}^{12}_{\mathcal{PMG}} \cup \{\mathtt{mg\_t}^{\mathtt{P}}(0) = 0\}$

$\odot\ \theta_{12} = \theta_{11} \circ \{\mathtt{C}/\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{s}(\mathtt{A}_1)), \mathtt{D}/\mathtt{X}_1, \mathtt{E}/\mathtt{s}(0)\}\ \mathcal{H}^{14}_{\mathcal{PMG}} = \mathcal{H}^{13}_{\mathcal{PMG}} \cup \{\mathtt{f}_1(\mathtt{s}(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{s}(\mathtt{A}_1))),\mathtt{X}_1) = \mathtt{X}_1\}$

$\odot\ \theta_{13} = \theta_{12} \circ \{\mathtt{E}_1/\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{s}(\mathtt{A}_1)), \mathtt{F}_1/\mathtt{X}_1\}\ \mathcal{H}^{15}_{\mathcal{PMG}} = \mathcal{H}^{14}_{\mathcal{PMG}} \cup \{\mathtt{f}_1(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{s}(\mathtt{s}(\mathtt{A}_1))),\mathtt{X}_1) = \mathtt{X}_1\}$

$\odot\ \theta_{14} = \theta_{13} \circ \{\mathtt{A}/\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{s}(\mathtt{s}(\mathtt{A}_1))), \mathtt{B}/\mathtt{X}_1\}\ \mathcal{H}^{16}_{\mathcal{PMG}} = \mathcal{H}^{15}_{\mathcal{PMG}} \cup \{\mathtt{f}(\mathtt{s}(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{s}(\mathtt{s}(\mathtt{A}_1)))),\mathtt{X}_1) = \mathtt{X}_1\}$

$\odot\ \theta_{15} = \theta_{14} \circ \{\mathtt{C}_1/\mathtt{s}(\mathtt{s}(\mathtt{A}_1)), \mathtt{D}_1/\mathtt{X}_1\}\ \mathcal{H}^{17}_{\mathcal{PMG}} = \mathcal{H}^{16}_{\mathcal{PMG}} \cup \{\mathtt{f}(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{A}_1)))),\mathtt{X}_1)) = \mathtt{X}_1,$
$\mathtt{t}(\mathtt{mg\_k}^{\mathtt{N}}) = 0\}$

then the magic transformation generates $\mathtt{mg\_h}^{\mathtt{P}}(\mathtt{s}(\mathtt{mg\_f}^{\mathtt{N}}(0))) := \mathtt{mg\_h}^{\mathtt{P}}(\mathtt{mg\_g}^{\mathtt{N}}(0))$, $\mathtt{mg\_h}^{\mathtt{P}}_{1}(\mathtt{Y}) := \mathtt{mg\_h}^{\mathtt{P}}(\mathtt{s}(\mathtt{Y}))$, and $\mathtt{mg\_h}^{\mathtt{P}}_{1}(0) := \mathtt{mg\_h}^{\mathtt{P}}_{1}(\mathtt{mg\_f}^{\mathtt{N}}(0))$. Therefore $\mathtt{g}(0)$ is demanded for satisfying the condition $\mathtt{s}(\mathtt{Y}) == \mathtt{s}(0)$.

In summary, in order to get laziness, there is an imposed condition to pass program rules (for inner symbols) as arguments of magic rules for outermost symbols: either the corresponding pattern of the outermost function is a constructor term or it is a variable occurring in a safe position (i.e. out of the scope of a function symbol) in the body or condition. Both cases ensure the laziness of the evaluation process. Otherwise, that is, if the variable occurs in the scope of a function symbol f then there could exist the corresponding magic rule for f.

For instance, let $\mathtt{f}(\mathtt{g}(0)) == 0$ the goal and the program $\mathtt{f}(\mathtt{s}(\mathtt{X})) := \mathtt{X}$, $\mathtt{g}(0) := \mathtt{s}(0)$ be, then there is a passing magic rule $\mathtt{mg\_f}^{\mathtt{P}}(\mathtt{s}(0)) := \mathtt{mg\_f}^{\mathtt{P}}(\mathtt{mg\_g}^{\mathtt{N}}(0))$, expressing that $\mathtt{f}$ demands the evaluation of $\mathtt{g}$ for lazily unifying with the pattern $\mathtt{s(X)}$. Now, let $\mathtt{f}(\mathtt{g}(0)) == \mathtt{s}(0)$ the goal and the program $\mathtt{f}(\mathtt{Y}) := \mathtt{s}(\mathtt{Y})$, $\mathtt{g}(0) := 0$ be, then there exists the same passing magic rule $\mathtt{mg\_f}^{\mathtt{P}}(0) := \mathtt{mg\_f}^{\mathtt{P}}(\mathtt{mg\_g}^{\mathtt{N}}(0))$, since $\mathtt{f}$ is an outermost symbol (thus occurring in the goal or in a subgoal), and for complying with the strict semantics, $\mathtt{Y}$ must be necessarily evaluated. However, there are no passing rules of this class whenever $\mathtt{f}(\mathtt{T}) := \mathtt{h}(\mathtt{T})$ and $\mathtt{h}(\mathtt{Y}) := 0$, given that both $\mathtt{f}$ and $\mathtt{h}$ are non-strict in $\mathtt{T}$ and $\mathtt{Y}$, respectively.

The bottom-up evaluation (shown in table 3) of the magic program in table 2 with the fix point operator of the definition 8 ends (that is, the Herbrand model for the magic program is computed in a finite number of steps) computing $\theta = \{\mathtt{X}/\mathtt{s}(\mathtt{s}(\mathtt{A}_1))), \mathtt{Y}/0\}$ as answer; that is, the instance $\mathtt{f}(\mathtt{mg\_g}^{\mathtt{N}}(\mathtt{s}(\mathtt{s}(\mathtt{A}_1)))), 0) == \mathtt{t}(\mathtt{mg\_k}^{\mathtt{N}})$ is satisfied in the computed Herbrand model.
The algorithm for the magic transformation is shown in tables 4 and 5, where

- $\bar{t}|_i$ represents the subterm of $\bar{t}$ at position i
- $\bar{e}[e']_i$ represents $\bar{e}$ replacing the subexpression at position i by e'

- safe(φ) represents the sub-expression of φ out of the scope of a function symbol
- e^N is defined as $X^N =_{def} X$, $c(\bar{e})^N =_{def} c(\overline{e^N})$ and $f(\bar{e})^N =_{def} f(\overline{e^{mg^N}})$ and $X^{mg^N} =_{def} X$, $c(\bar{e})^{mg^N} =_{def} c(\overline{e^{mg^N}})$ and $f(\bar{e})^{mg^N} =_{def} mg\_f^N(\overline{e^{mg^N}})$; that is, the inner function symbols are replaced by nesting magic constructors
- e^P is defined as $X^P =_{def} X$, $c(\bar{e})^P =_{def} c(\overline{e^P})$ and $f(\bar{e})^P =_{def} mg\_f^P(\overline{e^{mg^N}})$; that is, the outermost function symbols are replaced by passing magic functions and the inner function symbols by nesting magic constructors.
- The meaning of the algorithm parameters is as follows: $\bar{e}$ represents the expression or sequence of expressions to be considered (there can be subexpressions occurring in the goal or in some program rule); $h(\bar{t})$ is the head of a program rule; the boolean *Rule?* is true whenever the parameter $h(\bar{t})$ has been input; f is a function symbol representing that $\bar{e}$ is in the scope of the outermost symbol f; the boolean *Nested?* is true whenever the parameter f has been input; C represents conditions of some program rule or goal; M_g represents the computed set of magic rules; P_g represents the computed set of program rules; *ind* indicates the position of $\bar{e}$ in the scope of the outermost symbol f and G represents a set of triples (f, g, i) where each triple means that the function g is nested by f at position i.

The algorithm transforms every pair ($\mathcal{P}$,$\mathcal{G}$), where $\mathcal{P}$ is a program and $\mathcal{G}$ is a goal, into a pair ($\mathcal{P}^{\mathcal{MG}}$,$\mathcal{G}^{\mathcal{MG}}$) in such a way that the transformed program, evaluated by means of the fix point operator, computes the same solutions θ of the goal $\mathcal{G}$ w.r.t. the program $\mathcal{P}$. It is applied as follows:

```
$C := \emptyset; Mg := \emptyset; Pg := \mathcal{P}; G := \emptyset; ind := 0;$
**for every** $e == e' \in \mathcal{G}$ **do**
**Magic_Alg**$(e, \_, false, \_false, C, M_g, P_g, ind, G)$;
**Magic_Alg**$(e', \_, false, \_, false, C, M_g, P_g, ind, G)$;
$C := C \cup \{e == e'\}$
**endfor**
```

where "_" denotes arguments not needed for the calling.

Once the algorithm has been applied, $\mathcal{P}^{\mathcal{MG}}$ consists of $P_g^P \cup M_g$, where $(f(\bar{t}) := r \Leftarrow C)^P$ is of the form $f(\bar{t}) := r \Leftarrow mg\_f^P(\bar{t}) == true, C$, and $\Sigma^{\mathcal{MG}}$ consists of $\mathcal{C} \cup \mathcal{C}^{\mathcal{MG}}$ and $\mathcal{F} \cup \mathcal{F}^{\mathcal{MG}}$ where $\mathcal{C}^{\mathcal{MG}}$ and $\mathcal{F}^{\mathcal{MG}}$ denote the set of nesting magic constructors, and passing magic functions, respectively. $\mathcal{G}^{\mathcal{MG}}$ consists of the set $e^N == e'^N$ for each $e == e' \in \mathcal{G}$.

The basic idea of this algorithm is starting from the goal to traverse the program rules of the outermost function symbols in the goal. For every traversed program rule the algorithm includes in Mg the magic rules of the information passing from the outermost symbols to the body and conditions in its rules; and in Pg the new rules for the outermost symbols.

The parameter G includes the collection of occurrences of inner function symbols in outermost function symbols in order to use it in each information passing.

The main algorithm **Magic_Alg** uses an auxiliary algorithm **Nesting** which introduces the rules of inner function symbols as arguments of the outermost ones in both new and magic rules.

Table 4. Magic Algorithm

```
Magic_Alg(in $\bar{e}$ : $tuple(Expression)$; in $h(\bar{t})$ : $Expression$; in $Rule?$ : $Bool$;
 in $f$ : $FunctionSymbol$; in $Nested?$ : $Bool$; in $C$ : $Goal$; in/out $M_g$ : $Program$;
 in/out $P_g$ : $Program$; in/out $ind$ : $Index$; in/out $G$ : $set(tuple(FunctionSymbol,$
$FunctionSymbol, Index)))$
var $C''$ : $Goal$;
if $Nested?$ then
    if $Rule?$ then $M_g := M_g \cup \{f_{ind}(\bar{e})^P := mg\_h^P(\bar{t}) \Leftarrow C\}$
              else $M_g := M_g \cup \{f_{ind}(\bar{e})^P := true\}$
    endif;
    for every $e_1, \ldots, e_n$ do
    case $e_i$ of
      $X,\ X \in \mathcal{V}$ :
        if $Rule?$ then
          if $((h, k, j) \in G)$ and $(\bar{t}|_j \equiv X)$ then
            $G := G \cup \{(f_{ind}, k, i)\}$; Nesting($k, f, M_g, P_g, ind, i, G$);
          endif;
        endif;
      $c(\bar{e'}),\ c \in \mathcal{C}^{\mathcal{MG}}$ :
        for every $f_{ind}(\bar{t}) := r \Leftarrow C' \in P_g$ do
          if $(t_i \equiv c(\bar{t'})$ and not $(\bar{t'} \equiv \bar{X}$ and $\bar{X} \cap (safe(r) \cup safe(C')) = \emptyset))$ then
           $P_g := P_g \cup \{f_{ind+1}(\bar{t}[\bar{t'}]_i) := r \Leftarrow C',\ f_{ind}(\bar{X}[c(\bar{V})]_i)^N := f_{ind+1}(\bar{X}[\bar{V}]_i)^N\}$;
           $M_g := M_g \cup \{f_{ind+1}(\bar{X}[\bar{V}]_i)^P := f_{ind}(\bar{X}[c(\bar{V})]_i)^P\}$;
           endif;
           if $(t_i \in \mathcal{V}$ and $t_i \in safe(r) \cup safe(C'))$ then
           $P_g := P_g \cup \{f_{ind+1}(\bar{t}) := r \Leftarrow C',\ f_{ind}(\bar{X}[c(\bar{V})]_i)^N := f_{ind+1}(\bar{X}[\bar{V}]_i)^N\}$;
           $M_g := M_g \cup \{f_{ind+1}(\bar{X}[\bar{V}]_i)^P := f_{ind}(\bar{X}[c(\bar{V})]_i)^P\}$;
          endif;
        endfor;
        $ind := ind + 1$; Magic_Alg($\bar{e'}, h(\bar{t}), Rule?, f, true, C, M_g, P_g, ind, G$);
      $k(\bar{e'}),\ k \in \mathcal{F}$ :
        if $((f_{ind}, k, i) \notin G)$ then
          $G := G \cup \{(f_{ind}, k, i)\}$; Nesting($k, f, M_g, P_g, ind, i, G$);
          Magic_Alg($mg\_k^N(\bar{e'}), h(\bar{t}), Rule?, f, true, C, M_g, P_g, ind, G$);
        endif;
    endcase;
    endfor;
else
    for every $e_1, \ldots, e_n$ do
    case $e_i$ of
      $c(\bar{e'}),\ c \in \mathcal{C}$ :
        Magic_Alg($\bar{e'}, h(\bar{t}), Rule?, -, false, C, M_g, P_g, ind, G$);
      $k(\bar{e'}),\ k \in \mathcal{F}$ :
        Magic_Alg($\bar{e'}, h(\bar{t}), Rule?, k, true, C, M_g, P_g, ind, G$);
        for every $k(\bar{s}) := r \Leftarrow C' \in P_g$ do
          Magic_Alg($r, k(\bar{s}), true, -, false, C', M_g, P_g, ind, G$);
          $C'' := \emptyset$;
          for every $e == e' \in C'$ do
             Magic_Alg($e, k(\bar{s}), true, -, false, C'', M_g, P_g, ind, G$);
             Magic_Alg($e', k(\bar{s}), true, -, false, C'', M_g, P_g, ind, G$);
             $C'' := C'' \cup \{e == e'\}$
          endfor;
        endfor;
    endcase;
    endfor;
endif;
```

In the magic algorithm, we can distinguish the following main cases: (a) nested expressions (in the scope of an outermost function symbol) and (b) non-nested expressions (out of the scope of an outermost function symbol):

(a) in this case, let f the outermost function symbol and e the expression nested by f be, then a passing magic rule from the head of the rule to the nesting expression is included. In the particular case of the goal, the passing magic rule is a fact (see rules **(12)-(16)** and **(23)-(24)**).

Table 5. Nesting Transformation

```
Nesting(in $k$ : $FunctionSymbol$; in $f$ : $FunctionSymbol$; in/out $M_g$ : $Program$; in/out
 $P_g$ : $Program$; in $ind$ : $Index$; in $i$ : $Index$; in $G$ : $set(tuple(FunctionSymbol,$
 $FunctionSymbol, Index)))$;
var $C''$ : $Goal$;
for every $f_{ind}(\bar{t}) := r \Leftarrow C \in P_g$ do
        if $(t_i \notin \mathcal{V})$ or $((t_i \in \mathcal{V})$ and $(t_i \in safe(r) \cup safe(C)))$ then
          for every $k(\bar{s}) := r' \Leftarrow C' \in P_g$ do
                    $M_g := M_g \cup \{f_{ind}(\bar{X}[r']_i)^P := f_{ind}(\bar{X}[k(\bar{s})]_i)^P \Leftarrow C'\}$;
                    $P_g := P_g \cup \{f_{ind}(\bar{X}[k(\bar{s})]_i)^N := f_{ind}(\bar{X}[r']_i)^N \Leftarrow C'\}$;
                    Magic_Alg($r', k(\bar{s}), true, f, true, C', M_g, P_g, ind, G$);
                    $C'' := \emptyset$;
                    for every $e == e' \in C'$ do
                          Magic_Alg($e, f_{ind}(\bar{X}[k(\bar{s})]_i)^P, true, -, false, C'', M_g, P_g, ind, G$);
                          Magic_Alg($e', f_{ind}(\bar{X}[k(\bar{s})]_i)^P, true, -, false, C'', M_g, P_g, ind, G$);
                          $C'' := C'' \cup \{e == e'\}$;
                          $Mg := Mg \cup \{f_{ind}(\bar{X}[e]_i)^P := f_{ind}(\bar{X}[k(\bar{s})]_i)^P,$
                                $f_{ind}(\bar{X}[e']_i)^P := f_{ind}(\bar{X}[k(\bar{s})]_i)^P\}$;
                    endfor;
          endfor;
        endif;
endfor;
```

- If e is a variable occurring in a program rule for h, and it occurs in the head, then the algorithm checks using G if h nests some function in the position of the variable in the head; if any, the algorithm passes the nested function as argument of f (see rule **(22)**).
- If e is a constructor expression, the algorithm introduces the quoted auxiliary functions (see rules **(1)-(2)**).
- Finally, if e is a functional expression $k(\bar{e})$, the algorithm passes every rule of k as argument of the outermost symbol f in each rule of f, whenever the nested expression $k(\bar{e})$ is demanded by the rule; that is, the pattern of the rule is a constructor term or it is a variable occurring in a safe position of the body or conditions (see rules **(17)-(22)** and **(25)-(30)**).

(b) In the case of non-nested expression, the algorithm has only to decompose the arguments, so that if the expression is a functional one (there exists an outermost function symbol), then each rule for it must be analyzed.

6 Soundness, Completeness and Optimality

In this section we present our soundness, completeness and optimality results of our evaluation method based on the magic transformation proposed. From now on, we suppose there is a program $\mathcal{P}$, a goal $\mathcal{G}$ and $\mathcal{P}^{\mathcal{MG}}$, $\mathcal{G}^{\mathcal{MG}}$ represents the program and goal obtained by following the magic transformation. Our first result establishes the equivalence among both the original and transformed program w.r.t. the given goal.

Theorem 4 (Soundness and Completeness). *$\mathcal{P} \vdash_{\mathcal{CRWL}} \mathcal{G}\theta \Leftrightarrow \mathcal{P}^{\mathcal{MG}} \vdash_{\mathcal{CRWL}} \mathcal{G}^N\theta$.*

The second result ensures optimality of our bottom-up evaluation method, in the sense of, every value computed for a function symbol corresponds either with a subproof of a proof of some solution for the goal, or with a subproof for a non-strict equation involved in the lazy unification of an expression with the pattern of a rule of an outermost function symbol. Moreover, the result assures that each computed magic fact corresponds with one of the evaluated functions.

Theorem 5 (Optimality).

– *If $\mathcal{P}^{\mathcal{MG}} \vdash_{\mathcal{CRWL}} f(\bar{e})^N \rightarrow t$, $t \neq \bot$, then $f \in outer(\mathcal{P}, \mathcal{G})$ and either:*
  - *there exist θ, a proof $\mathcal{P} \vdash_{\mathcal{CRWL}} \mathcal{G}\theta$ of minimal size and a subproof of the form $\mathcal{P} \vdash_{\mathcal{CRWL}} f(\bar{e}) \rightarrow t$*
  - *or there exists $f(\bar{t}) := r \Leftarrow C \in [\mathcal{P}]$ such that $\mathcal{P} \vdash_{\mathcal{CRWL}} e_i \rightarrow t_i$ is a proof of minimal size.*

– *If $\mathcal{P}^{\mathcal{MG}} \vdash_{\mathcal{CRWL}} f_k(\bar{e})^N \rightarrow t, t \neq \bot$ then $f \in outer(\mathcal{P}, \mathcal{G})$ and either:*
  - *there exist θ, a proof $\mathcal{P} \vdash_{\mathcal{CRWL}} \mathcal{G}\theta$ of minimal size and a subproof of the form $\mathcal{P} \vdash_{\mathcal{CRWL}} f(\bar{e'}) \rightarrow t$ containing subproofs $\mathcal{P} \vdash_{\mathcal{CRWL}} e_i \rightarrow t_i$ for some $f(\bar{t}) := r \Leftarrow C \in [\mathcal{P}]$*
  - *or there exists $f(\bar{t}) := r \Leftarrow C \in [\mathcal{P}]$ such that $\mathcal{P} \vdash_{\mathcal{CRWL}} e_i \rightarrow t_i$ is a proof of minimal size.*

– *If $\mathcal{P}^{\mathcal{MG}} \vdash_{\mathcal{CRWL}} f(\bar{e})^P \rightarrow true$ then there exists a proof $\mathcal{P}^{\mathcal{MG}} \vdash_{\mathcal{CRWL}} f(\bar{e})^N \rightarrow t$ for some t.*

7 Conclusions and Future Work

In this paper, we have presented a framework for goal-directed bottom-up evaluation of functional logic programs. We have shown that our evaluation method is sound and complete w.r.t. a conditional rewriting logic which provides logic foundations to our functional logic language. We have also proved that our method is optimal, that is, the computed "facts" can be mapped with subproofs of a proof for some solution of the goal. We want to remark that our magic algorithm also provides a transformation for functional logic programs which can be used under a top-down evaluation; in fact a Prolog interpreter as:

$\mathtt{X} == \mathtt{X} : -\mathtt{var}(\mathtt{X}).$
$\mathtt{c}(\mathtt{X}_1, \ldots, \mathtt{X}_n) == \mathtt{c}(\mathtt{Y}_1, \ldots, \mathtt{Y}_n) : -\mathtt{X}_1 == \mathtt{Y}_1, \ldots, \mathtt{X}_n == \mathtt{Y}_n.$ % `for every c` $\in \mathcal{C}$
$\mathtt{X} == \mathtt{Y} : -\mathtt{X} := \mathtt{Z}, \mathtt{Z} == \mathtt{Y}.$
$\mathtt{X} == \mathtt{Y} : -\mathtt{Y} := \mathtt{Z}, \mathtt{X} == \mathtt{Z}.$

where every rule is defined as a Prolog rule of the form: $\mathtt{f}(\bar{\mathtt{t}}) := \mathtt{r} : -\mathtt{C}.$, evaluates goals with a top-down lazy narrowing strategy. Similarly, we could write a bottom-up interpreter in the line of the presented for logic programming in [6]. We expect that other magic transformations presented in the literature can be successfully applied in our framework. As future work, we will study how to consider, like the most deductive query languages, the dealing with negative goals in our language. Moreover, we would like to provide an implementation of our framework based on the indexation of the original and magic rules as well as the management of the secondary memory.

Acknowledgements: We would like to thank anonymous referees for their useful comments.

References

1. J. M. Almendros-Jiménez and A. Becerra-Terón. A Framework for Goal-Directed Bottom-Up Evaluation of Functional Logic Programs, available in `http://www.ual.es/~jalmen`. Technical report, Universidad de Almería, 2000.
2. S. Antoy, R. Echahed, and M. Hanus. A Needed Narrowing Strategy. In *Proc. of POPL'94*. pp. 268-279. ACM Press.

3. K. R. Apt. Logic programming. In *Handbook of Theoretical Computer Science*. vol B: Formal Models and Semantics, chapter 10, pp. 493–574. MIT Press, 1990.
4. R. Barbuti, R. Giacobazzi, and G. Levi. A General Framework for Semantics-based Bottom-up Abstract Interpretation of Logic Programs. *TOPLAS*, 15(1):133–181, 1993.
5. C. Beeri and R. Ramakrishnan. On the Power of Magic. *JLP*, 10(3,4):255–299, 1991.
6. M. Codish. Efficient Goal Directed Bottom-up Evaluation of Logic Programs. In *Proc. of ICLP'97*. pp. 422–439. MIT Press.
7. J. C. González-Moreno, M. T. Hortalá-González, F. López-Fraguas, and M. Rodríguez-Artalejo. An Approach to Declarative Programming Based on a Rewriting Logic. *JLP*, 1(40):47–87, 1999.
8. M. Hanus. The Integration of Functions into Logic Programming: From Theory to Practice. *JLP*, 19,20:583–628, 1994.
9. M. Hanus. Curry, An Integrated Functional Logic Language, Version 0.7.1. Technical report, University of Kiel, Germany, June 2000.
10. M. Hanus and C. Prehofer. Higher-order Narrowing with Definitional Trees. *JFP*, 9(1):33–75, 1999.
11. P. Kanellakis, G. Kuper, and P. Revesz. Constraint Query Languages. *JCSS*, 51(1):26–52, 1995.
12. D. Kemp, D. Srivastava, and P. Stuckey. Bottom-up Evaluation and Query Optimization of Well-founded Models. *TCS*, 146(1–2):145–184, 1995.
13. D. Kemp and P. Stuckey. Optimizing Bottom-up Evaluation of Constraint Queries. *JLP*, 26(1):1–30, 1996.
14. R. Loogen, F. J. López-Fraguas, and M. Rodríguez-Artalejo. A Demand Driven Computation Strategy for Lazy Narrowing. In *Proc. of PLILP'93*. pp. 184–200. LNCS 714.
15. F. J. Lopez-Fraguas. A General Scheme for Constraint Functional Logic Programming. In *Proc. of ALP'92*. pp. 213–227. LNCS 632.
16. F. J. López-Fraguas and J. Sánchez-Hernández. $\mathcal{TOY}$: A Multiparadigm Declarative System. In *Proc. of the RTA'99*. pp. 244–247. LNCS 1631.
17. J. J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic Programming with Functions and Predicates: The Language BABEL. *JLP*, 12(3):191–223, 1992.
18. A. Poulovasslis and C. Small. A Domain-theoretic Approach to Integrating Functional and Logic Database Languages. In *Proc. of VLDB'93*. pp. 416–428. Morgan Kaufmann.
19. R. Ramakrishnan. Magic Templates: A Spellbinding Approach to Logic Programs. *JLP*, 11(3,4):189–216, 1991.
20. R. Ramakrishnan, W. G. Roth, P. Seshadri, D. Srivastava, and S. Sudarshan. The CORAL Deductive Database System. In *Proc. of Inter. Conf. on Management of Data'93*. pp. 544–545. ACM Press.
21. R. Ramakrishnan and J. Ullman. A Survey of Deductive Database Systems. *JLP*, 23(2):125–149, 1995.
22. S. Reddi, A. Poulovassilis, and C. Small. PFL: An Active Functional DBPL. In *Active Rules in Database Systems*. chapter 16, pp. 297–308. Springer, 1999.
23. D. Sacca and C. Zaniolo. The Generalized Counting Method for Recursive Logic Queries. *TCS*, 62(1-2):187–220, 1988.
24. H. Seki. On the Power of Alexander Templates. In *Proc. of PODS'89*. pp. 150–159. ACM Press.
25. J. D. Ullman. Bottom-up Beats Top-down for Datalog. In *Procs. of PODS'89*. pp. 140–149. ACM Press.

26. J. Vaghani, K. Ramamohanarao, D. B. Kemp, Z. Somogyi, P. J. Stuckey, T. S. Leask, and J. Harland. The Aditi Deductive Database System. *VLDB*, 3(2):245–288, 1994.