Translating logic programs into conditional rewriting systems

F. van Raamsdonk

Probability, Networks and Algorithms (PNA)

PNA-R9803 April 30, 1998

Report PNA-R9803
ISSN 1386-3711

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Translating Logic Programs into Conditional Rewriting Systems

Femke van Raamsdonk
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

ABSTRACT
In this paper a translation from a subclass of logic programs consisting of the simply moded logic programs into rewriting systems is defined. In these rewriting systems conditions and explicit substitutions may be present. We argue that our translation is more natural than previously studied ones and establish a result showing its correctness.

1991 Mathematics Subject Classification: 68Q42, 16S15.
1991 Computing Reviews Classification System: D.1.6, I.2.2, I.2.3, F.3.2.
Keywords and Phrases: logic programming, rewriting, program transformation.
Note: This research was supported by NWO/SION project number 612-33-003, entitled 'Parallel declarative programming: transforming logic programs to lazy functional programs'. A shorter version of this paper appeared as [16].

1. Introduction

Logic and functional programming are both instances of declarative programming and hence it is not surprising that the relationship between them has been studied. However, the work so far has in our opinion not yet resulted in clear cut and simple to state results clarifying this relationship. Moreover, most of the work in the area concerns only termination of logic programming, via a translation into term rewriting systems. See Section 5 for a discussion of related work.

The aim of the present paper is to relate in a precise way the operational semantics of logic programming, resolution, to the operational semantics of functional programming, rewriting, thus abstracting from the syntactic details of particular programming languages. We discuss extensively the merits and deficiencies of possible translations and argue that the use of *conditions* and *explicit substitutions* makes it possible to design a natural and intuitive translation. Our translation can be used as a basis for an alternative implementation of a subset of logic programming via a translation to functional programs.

We provide a rigorous result showing the correctness of our translation. This result states that one resolution step using a clause C is translated into one or more rewrite steps, all using the rewrite rule C^*, which is the translation of C. Hence in particular termination of a logic program is implied by termination of its translation. Moreover, a successful resolution sequence is translated into a rewriting sequence that ends in an expression in normal form, from which the computed answer substitution can be read immediately.

2. Preliminaries

We assume the reader to be familiar with logic programming and refer to [2] for an overview. In this section we fix the notation and give the definitions that are less well-known.

We assume a set $\mathcal{V}$ consisting of infinitely many *variables* written as $x, y, z, \ldots$. A *logic program* is a triple of the form $(\mathcal{F}, \mathcal{R}, \mathcal{C})$ with $(f, g \in)$ $\mathcal{F}$ a set of *function symbols*, $(r, f, g \in)$ $\mathcal{R}$ a set of *relation symbols* and $(C, C' \in)$ $\mathcal{C}$ a set of *clauses* over $(\mathcal{F}, \mathcal{R})$. Queries are denoted by $Q, Q', \ldots$, and the empty query is written as $\Box$. Terms are denoted by $s, t, \ldots$ and atoms by $a, b, \ldots$.

Substitutions are denoted by $\sigma, \tau, \dots$. The identity substitution is denoted by ϵ, and the composition of substitutions σ and τ is denoted by $\sigma\tau$. The result of applying a substitution σ to a term s is denoted by $s\sigma$.

The set of free variables occurring in an expression X is denoted by $\mathsf{V}(X)$. We denote the union of the variables in the domain and the variables in the codomain of a substitution σ by $\mathsf{V}(\sigma)$.

The relation symbols of the logic programs considered in this paper use some arguments as input and some arguments as output. This is formalized using the notion of modes. Modes were introduced by Mellish [14] and further studied by Reddy [17]. A *base mode* is either *input*, denoted by $\downarrow$, or *output*, denoted by $\uparrow$. An *m-ary mode* is a product, denoted using $\times$, of m base modes. Without loss of generality, an m-ary mode is of the form $\downarrow \times \dots \times \downarrow \times \uparrow \times \dots \times \uparrow$ with first p times $\downarrow$ and then q times $\uparrow$, and $p + q = m$. Such a mode is denoted by (p, q). In the remainder of this paper, every relation symbol is supposed to have a fixed mode. The following convention will be used.

Notation 2.1. If r is a relation symbol of mode (p, q), then $r(\vec{s}, \vec{t})$ denotes the atom with terms $\vec{s} = s_1, \dots, s_p$ in the input positions of r and terms $\vec{t} = t_1, \dots, t_q$ in the output positions of r. Note that p and q may be zero. The length of a sequence $\vec{s}$ is denoted by $|\vec{s}|$.

One of the main differences between logic programming and rewriting is that the resolution relation of a logic program is defined using unification whereas the rewrite relating of a rewriting system is defined using matching. Apt and Etalle identify in [3] several classes of Prolog programs for which unification can be replaced by iterated matching. One of these classes consists of programs that are well-moded and satisfy in addition another restriction; in the present paper these programs are said to be *simply moded*. The class of simply moded logic programs is used as the domain of the translation defined in Section 4. For the definition of simply modedness we first need the definition of well-modedness, which is originally due to Dembiński and Małuszyński [9]. The following definition is taken from [2]. Intuitively, well-modedness is a restriction concerning the flow of information in a program.

Definition 2.2.

1. A query $r_1(\vec{s}_1, \vec{t}_1), \dots, r_m(\vec{s}_m, \vec{t}_m)$ is said to be *well-moded* if

$$\mathsf{V}(\vec{s}_i) \subseteq \bigcup_{j=1}^{i-1} \mathsf{V}(\vec{t}_j)$$

   for every $i \in \{1, \dots, m\}$.

2. A clause $r_0(\vec{t}_0, \vec{s}_{m+1}) \leftarrow r_1(\vec{s}_1, \vec{t}_1), \dots, r_m(\vec{s}_m, \vec{t}_m)$ is *well-moded* if

$$\mathsf{V}(\vec{s}_i) \subseteq \bigcup_{j=0}^{i-1} \mathsf{V}(\vec{t}_j)$$

   for every $i \in \{1, \dots, m + 1\}$.

3. A logic program $(\mathcal{F}, \mathcal{R}, \mathcal{C})$ is *well-moded* if every clause in $\mathcal{C}$ is well-moded.

Note that if $r_1(\vec{s}_1, \vec{t}_1), \dots, r_m(\vec{s}_m, \vec{t}_m)$ is a well-moded query, then $\mathsf{V}(\vec{s}_1) = \emptyset$. The concept of well-modedness is important because computing well-moded queries in well-moded programs yields computed answer substitutions that are ground (see [2]). Modes play an important rôle in the programming language Mercury [18]. Now we impose further restrictions on programs and queries as follows.

Definition 2.3.

1. A query $r_1(\vec{s}_1, \vec{t}_1), \dots, r_m(\vec{s}_m, \vec{t}_m)$ is said to be *simply moded* if

   (a) it is well-moded,
   (b) the terms $\vec{t}_1, \dots, \vec{t}_m$ are distinct variables.

2. A clause $r_0(\vec{s}_0, \vec{t}_0) \leftarrow r_1(\vec{s}_1, \vec{t}_1), \dots, r_m(\vec{s}_m, \vec{t}_m)$ is *simply moded* if
   (a) it is well-moded,
   (b) the terms $\vec{t}_1, \dots, \vec{t}_m$ are distinct variables not occurring in $\vec{s}_0$.
3. A logic program $(\mathcal{F}, \mathcal{R}, \mathcal{C})$ is *simply moded* if every clause in $\mathcal{C}$ is simply moded.

Note that our terminology differs from the one used in [3]: simply moded in the sense of Definition 2.3 is equivalent to the conjunction of well-modedness and simply modedness in the sense of [3]. It is possible to transform well-moded programs into simply moded programs. In the present paper we won't discuss this issue; we just restrict attention to simply moded logic programs and queries. The following notion of resolution will be used.

Definition 2.4. Let $(\mathcal{F}, \mathcal{R}, \mathcal{C})$ be a simply moded logic program. A *resolution step* is defined as a pair written as

$$\langle Q; \sigma \rangle \Rightarrow_C \langle Q'; \sigma' \rangle$$

with Q, Q' queries, σ, σ', τ substitutions and C a clause in $\mathcal{C}$ such that:

1. $\mathsf{V}(C) \cap \mathsf{V}(\langle Q; \sigma \rangle) = \emptyset$,
2. $Q = a_1, a_2, \dots, a_n$,
3. $C = h \leftarrow b_1, \dots, b_m$,
4. τ is a most general unifier of h and a_1,
5. $Q' = b_1\tau_1, \dots, b_m\tau_1, a_2\tau_2, \dots, a_n\tau_2$,
6. $\sigma' = \sigma\tau_2$,

with τ_1 the substitution τ restricted to $\mathsf{V}(h)$ and τ_2 the substitution τ restricted to $\mathsf{V}(a_1)$.

A sequence of resolution steps is called a *resolution sequence*. A resolution sequence is *successful* if it ends in $\langle \Box; \sigma \rangle$, for some substitution σ. We write $\Rightarrow$ instead of $\Rightarrow_C$ if it is clear or irrelevant which clause is used in the resolution step.

A few remarks concerning the previous definition are appropriate. First, note that always the leftmost atom is selected. Second, the expressions that are transformed are pairs consisting of a query and a substitution. In some other definitions of the resolution relation, see for instance [2], the expressions that are transformed are queries. We consider the first option to be more natural; moreover it is closer to actual implementations. Third, we make essential use of the form of simply moded clauses and programs, which also ensures that instead of unification iterated matching can be used, as shown by Apt and Etalle in [3]. Suppose, using the notation of Definition 2.4, that $a_1 = r(\vec{s}_1, \vec{t}_1)$ and $h = r(\vec{s}, \vec{t})$ are unifiable. Then to start with $\vec{s}_1$ and $\vec{s}$ are unifiable, which means, since $\mathsf{V}(\vec{s}_1) = \emptyset$, that there is a substitution τ_1 such that $\vec{s}\tau_1 = \vec{s}_1$. We call this substitution the *matching substitution* since it matches the input part of the head of a clause with the input part of an atom. Second, since $\vec{t}_1$ are distinct variables, the substitution τ_2 that assigns $\vec{t}\tau_1$ to $\vec{t}_1$ is a unifier of $\vec{t}_1$ and $\vec{t}\tau_1$. Because τ_2 expresses which values are computed for the variables $\vec{t}_1$, we call τ_2 the *computation substitution*. Note that the domains of τ_1 and τ_2 are disjoint.

In the remainder of the paper we will tacitly make use of the following result, which combines results of [2] and [3]. The definition of resolution we employ in this paper permits to give a slightly simpler proof.

Theorem 2.5. *Let $(\mathcal{F},\mathcal{R},\mathcal{C})$ be a simply moded logic program and let Q be a simply moded query. If $\langle Q;\sigma\rangle \Rightarrow \langle Q';\sigma'\rangle$, then the query Q' is also simply moded.*

Proof. Let $(\mathcal{F},\mathcal{R},\mathcal{C})$ be a simply moded logic program. Let $Q = a_1,\ldots,a_n$ be a simply moded query with $a_i = r_i(\vec{s}_i,\vec{t}_i)$ for every $i \in \{1,\ldots,n\}$. Let $\langle Q;\sigma\rangle \Rightarrow \langle Q';\sigma'\rangle$ be a resolution step using a clause $C = h \leftarrow b_1,\ldots,b_m$ with $h = r_1(\vec{u},\vec{v})$ and $b_i = r'_i(\vec{u}_i,\vec{v}_i)$ for every $i \in \{1,\ldots,m\}$. We have $Q' = b_1\tau_1,\ldots,b_m\tau_1,a_2\tau_2,\ldots,a_n\tau_2$ with τ a most general unifier of a_1 and h, split in a matching substitution τ_1 and a computation substitution τ_2.

Since C is well-moded, we have for every $i \in \{1,\ldots,m\}$:

$$\mathsf{V}(\vec{u}_i) \subseteq \bigcup_{j=1}^{i-1} \mathsf{V}(\vec{v}_j) \cup \mathsf{V}(\vec{u}).$$

Hence we have for $i \in \{1,\ldots,m\}$:

$$\begin{aligned}\mathsf{V}(\vec{u}_i\tau_1) &\subseteq \bigcup_{j=1}^{i-1} \mathsf{V}(\vec{v}_j\tau_1) \cup \mathsf{V}(\vec{u}\tau_1)\\ &= \bigcup_{j=1}^{i-1} \mathsf{V}(\vec{v}_j\tau_1).\end{aligned}$$

Further, we have

$$\mathsf{V}(\vec{v}) \subseteq \bigcup_{j=1}^{m} \mathsf{V}(\vec{v}_j) \cup \mathsf{V}(\vec{u}).$$

Moreover, since Q is well-moded, we have for every $i \in \{1,\ldots,n\}$:

$$\mathsf{V}(\vec{s}_i) \subseteq \bigcup_{j=1}^{i-1} \mathsf{V}(\vec{t}_j).$$

This yields for $i \in \{2,\ldots,n\}$:

$$\begin{aligned}\mathsf{V}(\vec{s}_i\tau_2) &\subseteq \bigcup_{j=1}^{i-1} \mathsf{V}(\vec{t}_j\tau_2)\\ &= \mathsf{V}(\vec{t}_1\tau_2) \cup \bigcup_{j=2}^{i-1} \mathsf{V}(\vec{t}_j\tau_2)\\ &= \mathsf{V}(\vec{v}\tau_1) \cup \bigcup_{j=2}^{i-1} \mathsf{V}(\vec{t}_j\tau_2)\\ &\subseteq \bigcup_{j=1}^{m} \mathsf{V}(\vec{v}_j\tau_1) \cup \mathsf{V}(\vec{u}\tau_1) \cup \bigcup_{j=2}^{i-1} \mathsf{V}(\vec{t}_j\tau_2)\\ &= \bigcup_{j=1}^{m} \mathsf{V}(\vec{v}_j\tau_1) \cup \bigcup_{j=2}^{i-1} \mathsf{V}(\vec{t}_j\tau_2).\end{aligned}$$

Hence Q' is well-moded.

Since C is simply moded, the variables $\vec{v}_1,\ldots,\vec{v}_m$ are not in the domain of the assignment τ_1. Since Q is simply moded, the variables $\vec{t}_2,\ldots,\vec{t}_n$ are not in the domain of the substitution τ_2. Because moreover C and Q have no variables in common, terms in output positions of atoms in Q' are different variables. We conclude that Q' is simply moded. □

3. Conditional Rewriting

In this section we define conditional rewriting systems that will serve as the codomain of the translation defined in Section 4. They differ from the usual ones in several respects. First, conditions are treated on an object-level instead of on a meta-level. Second, the expressions that are transformed are not terms, but environments with conditions. Moreover, the conditions may contain explicit substitutions.

We assume a set $\mathcal{V}$ consisting of infinitely many *variables* written as $x, y, z, \dots$ and a set $\mathcal{T} = \{\mathsf{T}_i \mid i \geq 0\}$ of symbols for *tupling.* The arity of $\mathsf{T}_i \in \mathcal{T}$ is i. A conditional rewriting system is specified by a pair $(\mathcal{F}, \mathcal{RR})$ consisting of a set of *function symbols* and a set of *conditional rewrite rules.* We assume that $\mathcal{T} \subseteq \mathcal{F}$. The set of terms is denoted by Terms and terms are denoted by $s, t, \dots$ as in the previous section. We write T instead of T_0 and s instead of $\mathsf{T}_1(s)$.

We assume further a binary operator $\otimes$ on the set of terms. A *condition* is a term or $c \otimes d$ with c and d conditions. The set of conditions is denoted by Con and conditions are denoted by $c, d, \dots$. Note that $\mathsf{Terms} \subseteq \mathsf{Con}$. We will make use of the following syntactic constructs.

Definition 3.1.

1. An *environment* is inductively defined as follows.
   (a) $[\,]$ is an environment (the empty environment),
   (b) if e is an environment, $x_1, \dots, x_m$ (for some $m > 0$) are variables and s is a term then $e\,[\mathsf{T}_m(x_1, \dots, x_m) := s]$ is an environment.

   Environments are denoted by $e, e', \dots$.
2. A *condition with explicit substitutions* is a pair consisting of a condition c and an environment e, denoted using juxtaposition by ce. Conditions with explicit substitutions are like conditions denoted by $c, c', \dots$. Terms with explicit substitutions are denoted as terms without explicit substitutions.
3. A *conditional term* is a pair consisting of a term and a condition, where both the term and the condition may contain explicit substitutions. A conditional term is denoted by $s \Leftarrow c$.
4. A *conditional environment* is a pair consisting of an environment and a condition, where the condition may contain explicit substitutions. A conditional environment is denoted by $e \Leftarrow c$.

We write s instead of $s \Leftarrow \mathsf{T}$ and e instead of $e \Leftarrow \mathsf{T}$. Further we will work modulo the following equations:

$$\boxed{\begin{array}{rcl}
c \otimes c' & = & c' \otimes c \\
(c \otimes c') \otimes c'' & = & c \otimes (c' \otimes c'') \\
c \otimes c & = & c \\
c \otimes \mathsf{T} & = & c \\[1em]
ce \otimes c'e & = & (c \otimes c')e \\
ce \otimes c' & = & (c \otimes c')e \\[1em]
e[\mathsf{T}_i(\vec{x}) := s\tilde{e}] & = & e[\mathsf{T}_i(\vec{x}) := s]\tilde{e} \\
[\mathsf{T}_i(\vec{x}) := \mathsf{T}_i(\vec{s})] & = & [x_1 := s_1] \dots [x_i := s_i]
\end{array}}$$

We assume a hygienic treatment of variables, for instance in $ce \otimes c' = (c \otimes c')e$ variables bound by e are supposed not to occur in c'. The definition of a conditional rewrite rule is as follows.

Definition 3.2. A *conditional rewrite rule* is a pair $l \to (r \Leftarrow c)$ such that

1. l is a non-variable term,
2. $r \Leftarrow c$ is a conditional term,
3. $\mathsf{V}(r \Leftarrow c) \subseteq \mathsf{V}(l)$.

A function symbol that occurs as the head-symbol of the left-hand side of a conditional rewrite rule is said to be a *defined symbol.* A function symbol that is not a defined symbol is a *constructor symbol.* We assume the symbols in $\mathcal{T}$ to be constructor symbols. A *context* is an expression with one hole in it. The result of replacing the hole $[\,]$ in a context $C[\,]$ by an expression X is denoted by $C[X]$.

Definition 3.3. Let $(\mathcal{F}, \mathcal{RR})$ be a conditional rewriting system. The rewrite relation $\to$ on conditional environments is defined as follows. We have

$$(e \Leftarrow c) \ \to_\rho \ (e' \Leftarrow d\sigma \otimes c')$$

if there is a rewrite rule $\rho = l \to (r \Leftarrow d)$ in $\mathcal{RR}$, a substitution σ and a context $C[\,]$ such that

1. $(e \Leftarrow c) = C[l\sigma]$,
2. $(e' \Leftarrow c') = C[r\sigma]$.

The relation $\to$ is defined as the union $\cup_{\rho \in \mathcal{RR}} \to_\rho$.

The transitive closure of a relation $\to$ is denoted by $\to^+$ and the reflexive-transitive closure is denoted by $\to^*$. Note that we consider only one level of conditions. For instance, we have $e \Leftarrow c_1 \otimes c_2$ instead of $(e \Leftarrow c_1) \Leftarrow c_2$. A conditional environment that cannot be rewritten is said to be in *normal form.* Note that a conditional environment without defined symbols is in normal form.

Example 3.4. As an example, we consider the conditional rewriting system defined by the following rewrite rules:

$$\begin{array}{lcl} a & \to & (b \Leftarrow c) \\ c & \to & \mathsf{T} \end{array}$$

We have the following rewrite sequence:

$$[x := a] \to ([x := b] \Leftarrow c) \to [x := b].$$

We consider environments instead of simply terms because this permits to define a natural translation of substitutions in logic programming. The choice to consider conditions on object- instead of on a meta-level is also motivated by the translation; this permits to translate one resolution step into one or two rewrite steps. However we claim that also from different perspectives it can be useful to consider conditions on object-level, for instance if one is interested in the cost of a computation in terms of the number of rewrite steps. Then it is clearly important to take also the steps performed to check a condition into account. A similar form of condition rewriting, called reduction without evaluation of the premise, is introduced by Bockmayr in [7] (see also [8]) in order to relate conditional rewriting and conditional narrowing. A variant of the relation introduced by Bockmayr is used by Middeldorp and Hamoen in [15].

4. Translation

In this section we define the translation from simply moded logic programs into conditional rewriting systems as defined in the previous section.

Translating the Alphabet. The symbols used in a logic program are variables, function symbols with a fixed arity and relation symbols with a fixed mode. These symbols should be translated into symbols used in a conditional rewriting system. We suppose variables to be universal. Both the function symbols and the relation symbols of a logic program are translated into function symbols as follows.

Definition 4.1.

1. A variable x is translated into itself.
2. A function symbol $f \in \mathcal{F}$ of arity m is translated into a function symbol f^* of arity m.
3. A relation symbol $r \in \mathcal{R}$ of mode (p,q) is translated into a function symbol r^* of arity p.

The set $\{f^* \mid f \in \mathcal{F}\}$ is denoted by $\mathcal{F}^*$ and the set $\{r^* \mid r \in \mathcal{R}\}$ is denoted by $\mathcal{R}^*$. We will write f and r instead of f^* and r^*. The translation of the alphabet is extended homomorphically to the set of atoms. Note that the translation is the identity on the set of terms of a logic program.

Translating Clauses. The dynamics of a logic program is prescribed by its set of clauses. In a rewriting system, the rewrite rules determine how expressions can be transformed. We will define how to translate simply moded clauses into conditional rewriting rules with explicit substitution as defined in Section 3. First we motivate the use of the three main particularities of the class of rewriting systems used as codomain: tupling, conditions and explicit substitutions.

First we discuss the use of *tupling.* Consider a clause of the form $h \leftarrow$, with an empty body. Such a clause is of the form

$$f(s_1, \dots, s_p, t_1, \dots, t_q) \leftarrow$$

with the first p arguments input and the last q arguments output. The natural translation of such a clause is

$$f(s_1, \dots, s_p) \to \mathsf{T}_q(t_1, \dots t_q)$$

with T_q a symbol for tupling of arity q as defined in Section 3. We identify the tuple of arity 0 (T_0) with 'true'. Hence a simply moded clause $(\mathcal{F}, \mathcal{R})$ will be translated into a rewrite rule over the alphabet $\mathcal{F}^* \cup \mathcal{R}^* \cup \mathcal{T}$ with $\mathcal{T} = \{\mathsf{T}_i \mid i \geq 0\}$ as defined in Section 2.

Second we discuss the use of *conditions.* As an example we consider the following logic program:

$$\begin{array}{rcl} even(0) & \leftarrow & \\ odd(s(x)) & \leftarrow & even(x) \end{array}$$

with modes $even, odd$: $\downarrow$. This program is translated into the following conditional rewriting system:

$$\begin{array}{lclcl} even(0) & \to & \mathsf{T} & & \\ odd(s(x)) & \to & \mathsf{T} & \Leftarrow & even(x) \end{array}$$

Intuitively, the resolution sequence

$$\langle odd(s(0)); \epsilon \rangle \Rightarrow \langle even(0); \epsilon \rangle \Rightarrow \langle \Box; \epsilon \rangle$$

corresponds to the rewrite sequence

$$odd(s(0)) \rightarrow (\mathsf{T} \Leftarrow even(0)) \rightarrow \mathsf{T}.$$

Note that the use of conditions on an object level is essential, if we want every resolution step to correspond to at least one rewriting step. Using 'normal' conditional rewriting, the second rewrite step would take place on a meta-level and would hence not be observable.

Third we discuss the use of *explicit substitutions*. As an example we consider the logic program for addition of natural numbers:

$$\begin{array}{rcl} add(0,x,x) & \leftarrow & \\ add(s(x),y,s(z)) & \leftarrow & add(x,y,z) \end{array}$$

with mode *add*: $\downarrow \times \downarrow \times \uparrow$. A naive but elegant way of translating this logic program yields the following conditional rewriting system:

$$\begin{array}{lclcl} add(0,x) & \rightarrow & x & & \\ add(s(x),y) & \rightarrow & s(z) & \Leftarrow & add(x,y) \rightarrow^* z \end{array}$$

Note that this conditional rewriting system is not in the format defined in Section 2; it is in fact a conditional rewriting system with so-called *extra variables*, since the variable z in the last rewrite rule does not appear in the left-hand side. This is, ignoring some notational differences, the translation used by Ganzinger and Waldmann in [10]. Although for the purpose of that paper, which is to provide a method to prove termination of logic programs, this translation is satisfactory, for the purposes of the present paper it is not, for the following reasons. First, every successful resolution sequence starting in a simply moded query is translated into a rewrite sequence consisting of one rewrite step, so the translation does not give any indication of the cost of a computation expressed in the number of transformation steps. This is the case since resolution steps at object-level are translated into rewriting steps at meta-level. Second, we think that the conditional part of a rewriting rule should be used to check a condition and not to calculate the value of a variable. Reconsidering the second rewrite rule reveals that the problems mentioned above can be solved by turning the condition of the second rewrite rule into a substitution, yielding the following rewrite rule:

$$add(s(x),y) \rightarrow s(z)[z := add(x,y)].$$

If the condition is evaluated in the usual way, then this rewrite rule takes the following form:

$$add(s(x),y) \rightarrow s(add(x,y)).$$

In this way the logic program for addition is translated into the usual rewriting system for addition. However, we choose not to evaluate the conditions in right-hand sides of rewrite rules for the following reason. Translating the clause

$$f(x,a) \quad \leftarrow \quad f(a,y)$$

with mode f: $\downarrow \times \uparrow$ into a rewrite rule with explicit conditions yields

$$f(x) \quad \rightarrow \quad a[y := f(a)].$$

If the condition is evaluated, the rule takes the form

$$f(x) \quad \rightarrow \quad a.$$

Hence if the substitution is evaluated, a non-terminating logic program is translated into a terminating rewriting system: the resolution sequence

$$\langle f(a,a);\epsilon\rangle \Rightarrow \langle f(a,a);\epsilon\rangle \Rightarrow \ldots$$

corresponds intuitively to the rewriting sequence

$$f(a) \to a$$

which ends in normal form after one rewrite step. If the substitution is not evaluated, the infinite resolution sequence above corresponds intuitively to the infinite rewrite sequence

$$f(a) \to a[y := f(a)] \to a[y := a[y' := f(a)]] \ldots$$

in which almost all rewrite steps take place in 'garbage'. Finally, the order in which the explicit substitutions occur in the translation of a clause is determined by the flow of information, which is in simply moded clauses from left to right. See for an illustration Example 4.3.

This discussion motivates the following definition of the translation of a moded (not necessarily simply moded) clause.

Definition 4.2. Let

$$C = h \leftarrow b_1, \ldots, b_m$$

with $h = r(\vec{s}, \vec{t})$ and $m \geq 0$ be a moded clause over $(\mathcal{F}, \mathcal{R})$ with distinct variables in the output positions of the body. Define for every $p \in \{1, \ldots, m+1\}$: $C_p = h \leftarrow b_p, \ldots, b_m$.

1. The translation of C_p, denoted by C_p^*, is defined by induction on $m + 1 - p$.
   (a) Suppose that $p = m + 1$. The translation of C_p is defined as follows:
   $$C_p^* = r(\vec{s}) \to \mathsf{T}_i(\vec{t}) \quad \text{with } i = |\vec{t}|.$$
   (b) Suppose that $1 \leq p < m + 1$ and let $C_{p+1}^* = l_{p+1} \to (r_{p+1} \Leftarrow c_{p+1})$.
      i. If $b_p = r_p(\vec{s}_p, \vec{t}_p)$ with $|\vec{t}_p| = i > 0$, then
      $$C_p^* = l_{p+1} \to r_{p+1}[\mathsf{T}_i(\vec{t}_p) := r_p(\vec{s}_p)] \Leftarrow c_{p+1}[\mathsf{T}_i(\vec{t}_p) := r_p(\vec{s}_p)].$$
      ii. If $b_p = r_p(\vec{s}_p)$, then
      $$C_p^* = l_{p+1} \to (r_{p+1} \Leftarrow r_p(\vec{s}_p) \otimes c_{p+1}).$$
2. The translation of C, denoted by C^*, is defined as follows: $C^* = C_1^*$.

Note that we have $l_p = r(\vec{s})$ for every $p \in \{1, \ldots, m+1\}$ in the previous definition. Another observation is that the relation symbols of a logic program which are defined by clauses are translated into defined symbols, and that its function symbols are translated into constructor symbols.

Example 4.3. Consider the simply moded clause

$$C = f(x, z) \leftarrow g(x, y), h(y), g'(y, z)$$

with modes f, g, g': $\downarrow \times \uparrow$ and h: $\downarrow$. Following Definition 4.2, we find the following:

$C_4 = f(x, z) \leftarrow$	$C_4^* = f(x) \to z$
$C_3 = f(x, z) \leftarrow g'(y, z)$	$C_3^* = f(x) \to z[z := g'(y)]$
$C_2 = f(x, z) \leftarrow h(y), g'(y, z)$	$C_2^* = f(x) \to z[z := g'(y)] \Leftarrow h(y)$
$C_1 = f(x, z) \leftarrow g(x, y), h(y), g'(y, z)$	$C_1^* = f(x) \to z[z := g'(y)][y := g(x)] \Leftarrow h(y)[y := g(x)]$

Hence $C^* = f(x) \to z[z := g'(y)][y := g(x)] \Leftarrow h(y)[y := g(x)]$.

The following result states that the translation of a simply moded clause is a conditional rewrite rule.

Proposition 4.4. *Let C be a simply moded clause. Then C^* is a conditional rewrite rule.*

Proof. Let $C = h \leftarrow b_1, \ldots, b_m$ with $h = r(\vec{s}, \vec{t})$ and $m \geq 0$ be a simply moded clause. Its translation C^* is of the form $l \rightarrow (r \Leftarrow c)$ with $l = r(\vec{s})$. It is clear that $l \notin \mathcal{V}$. It remains to be shown that $\mathsf{V}(r \Leftarrow c) \subseteq \mathsf{V}(l)$. This is proved by induction on the definition of the translation of C. Let $C_p = h \leftarrow b_p, \ldots, b_m$ and $C_p^* = l_p \rightarrow (r_p \Leftarrow c_p)$ for every $p \in \{1, \ldots, m+1\}$. Let further $b_p = r_p(\vec{s}_p, \vec{t}_p)$, possibly with $|\vec{t}_p| = 0$, for every $p \in \{1, \ldots, m\}$. We prove the following:

$$\mathsf{V}(r_p \Leftarrow c_p) \subseteq \mathsf{V}(\vec{t}_1) \cup \ldots \cup \mathsf{V}(\vec{t}_{p-1}) \cup \mathsf{V}(\vec{s})$$

for every $p \in \{1, \ldots, m+1\}$.

1. Suppose that $p = m+1$. We have $C_p = r(\vec{s}, \vec{t}) \leftarrow$ and

$$C_p^* = r(\vec{s}) \rightarrow (\mathsf{T}_i(\vec{t}) \Leftarrow \mathsf{T})$$

with $i = |\vec{t}|$. Since C is well-moded, we have

$$\begin{aligned} \mathsf{V}(\vec{t}) &\subseteq \mathsf{V}(\vec{t}_1) \cup \ldots \cup \mathsf{V}(\vec{t}_m) \cup \mathsf{V}(\vec{s}) \\ &= \mathsf{V}(\vec{t}_1) \cup \ldots \cup \mathsf{V}(\vec{t}_{p-1}) \cup \mathsf{V}(\vec{s}). \end{aligned}$$

Hence $\mathsf{V}(r_p \Leftarrow c_p) \subseteq \mathsf{V}(\vec{t}_1) \cup \ldots \cup \mathsf{V}(\vec{t}_{p-1}) \cup \mathsf{V}(\vec{s})$.

2. Suppose that $1 \leq p < m+1$. The induction hypothesis is:

$$\mathsf{V}(r_{p+1} \Leftarrow c_{p+1}) \subseteq \mathsf{V}(\vec{t}_1) \cup \ldots \cup \mathsf{V}(\vec{t}_p) \cup \mathsf{V}(\vec{s}).$$

Consider $b_p = r_p(\vec{s}_p, \vec{t}_p)$ possibly with $|\vec{t}_p| = 0$. Since C is well-moded, we have that

$$\mathsf{V}(\vec{s}_p) \subseteq \mathsf{V}(\vec{t}_1) \cup \ldots \cup \mathsf{V}(\vec{t}_{p-1}) \cup \mathsf{V}(\vec{s}). \tag{4.1}$$

Two cases are distinguished.

(a) $|\vec{t}_p| = 0$. Then

$$C_p^* = l_{p+1} \rightarrow (r_{p+1} \Leftarrow r_p(\vec{s}_p) \otimes c_{p+1}).$$

Because $|\vec{t}_p| = 0$, we have $\mathsf{V}(r_{p+1} \Leftarrow c_{p+1}) \subseteq \mathsf{V}(\vec{t}_1) \cup \ldots \cup \mathsf{V}(\vec{t}_{p-1}) \cup \mathsf{V}(\vec{s})$. Together with 4.1, this yields $\mathsf{V}(r_p \Leftarrow c_p) \subseteq \mathsf{V}(\vec{t}_1) \cup \ldots \cup \mathsf{V}(\vec{t}_{p-1}) \cup \mathsf{V}(\vec{s})$.

(b) $|\vec{t}_p| = i > 0$. Then

$$C_p^* = l_{p+1} \rightarrow r_{p+1}[\mathsf{T}_i(\vec{t}_p) := r_p(\vec{s}_p)] \Leftarrow c_{p+1}[\mathsf{T}_i(\vec{t}_p) := r_p(\vec{s}_p)].$$

The combination of the induction hypothesis and 4.1 yields that $\mathsf{V}(r_p \Leftarrow c_p) \subseteq \mathsf{V}(\vec{t}_1) \cup \ldots \cup \mathsf{V}(\vec{t}_{p-1}) \cup \mathsf{V}(\vec{s})$.

Finally note that $C^* = C_1^* = l_1 \rightarrow (r_1 \Leftarrow c_1)$ and $\mathsf{V}(r_1 \Leftarrow c_1) \subseteq \mathsf{V}(\vec{s}) = \mathsf{V}(l_1)$. □

Translating Queries and Substitutions. In a resolution step, a pair consisting of a query and a substitution is transformed into another pair consisting of a query and a substitution. We now define the translation of such pairs into conditional environments. First, a substitution is translated into a conditional environment as follows.

Definition 4.5. Let $\sigma = \{x_1 \mapsto s_1, \ldots, x_m \mapsto s_m\}$ be a substitution. Its translation, denoted by σ^*, is defined as follows:

$$\sigma^* = [x_1 := s_1] \ldots [x_m := s_m].$$

Note that the translation of a substitution is a conditional environment in normal form, since a term in a logic program is translated into a term not containing defined symbols.

A moded query is also translated into a conditional environment.

Definition 4.6. Let $Q = a_1, \ldots, a_m$ with $m \geq 0$ be a moded query over $(\mathcal{F}, \mathcal{R})$ with distinct variables in the output positions. Define for every $p \in \{1, \ldots, m+1\}$: $Q_p = a_p, \ldots, a_m$.

1. The translation of Q_p, denoted by Q_p^*, is defined by induction on $m + 1 - p$ as follows.
   (a) Suppose that $p = m + 1$. Then:
   $$Q_p^* = [\,].$$
   (b) Suppose that $1 \leq p < m + 1$. Suppose that $Q_{p+1}^* = e_{p+1} \Leftarrow c_{p+1}$.
      i. If $a_p = r_p(\vec{s}_p, \vec{t}_p)$ with $|\vec{t}_p| = i > 0$, then
      $$Q_p^* = e_{p+1}[\mathsf{T}_i(\vec{t}_p) := r_p(\vec{s}_p)] \Leftarrow c_{p+1}[\mathsf{T}_i(\vec{t}_p) := r_p(\vec{s}_p)].$$
      ii. If $a_p = r_p(\vec{s}_p)$, then
      $$Q_p^* = e_{p+1} \Leftarrow r_p(\vec{s}_p) \otimes c_{p+1}.$$
2. The translation of Q, denoted by Q^*, is defined as follows: $Q^* = Q_1^*$.

Using the previous two definitions, the translation of a pair consisting of a query and a substitution is defined as follows.

Definition 4.7. Let Q be a query with translation $Q^* = e \Leftarrow c$ and let σ be an substitution with translation $\sigma^* = \tilde{e}$. Then: $\langle Q; \sigma\rangle^* = \tilde{e}\, e \Leftarrow c$.

One might ask whether the order in which the environments $\tilde{e}$ and e are concatenated in the previous definition is essential. This is indeed the case, as shown in Example 4.8. The reason is that we do not consider concatenation of environments to be commutative; that is, $e\,\tilde{e}$ is not the same as $\tilde{e}\,e$. Another natural question is whether splitting the substitution used in a resolution step in a matching substitution and a computation substitution, as defined in Definition 2.4 is necessary. This is indeed essential, which is also shown in Example 4.8. In this example we further discuss some slight variations of the definition of resolution and explain why they do not (or do not easily) permit to translate a resolution step into a non-empty sequence of rewrite steps.

Example 4.8. In this example we make use of the following two clauses:

$$\begin{array}{rcll} C_1 & = & f(x, g(y)) & \leftarrow \quad h(x, y) \\ C_2 & = & f(x, g(x)) & \leftarrow \end{array}$$

with modes $f, h : \downarrow \times \uparrow$. Their translations are as follows:

$$\begin{array}{rcll} C_1^* & = & f(x) & \rightarrow \quad g(y)[y := h(x)] \\ C_2^* & = & f(x) & \rightarrow \quad g(x) \end{array}$$

1. First we show that the order in which the environments e and $\tilde{e}$ are concatenated in Definition 4.7 cannot be changed. We have a resolution step

   $$\langle f(a,z);\epsilon\rangle \Rightarrow_{C_1} \langle h(a,y);\{z \mapsto g(y)\}\rangle.$$

   Using Definition 4.7, this resolution step is translated into the following rewrite step:

   $$[z := f(a)] \rightarrow_{C_1^*} [z := g(y)][y := h(a)].$$

   The translation is not correct if $\langle h(a,y);\{z \mapsto g(y)\}\rangle$ is translated into $[y := h(a)][z := g(y)]$.

2. We consider an alternative definition of resolution by replacing clause 6 of Definition 2.4 by

   $$\sigma' = \sigma\tau.$$

   Using this alternative definition, we have a resolution step

   $$\langle f(a,z);\epsilon\rangle \Rightarrow_{C_1} \langle h(a,y);\{x \mapsto a, z \mapsto g(y)\}\rangle.$$

   The matching substitution $x \mapsto a$ is not observable in a rewrite step using this matching substitution.

3. A second alternative definition of resolution is obtained by replacing clause 5 of Definition 2.4 by

   $$Q' = b_1, \ldots, b_m, a_1, \ldots a_n.$$

   Using this definition of resolution, we have the resolution step

   $$\langle f(a,z), h(z,z');\epsilon\rangle \Rightarrow_{C_2} \langle h(z,z');\{z \mapsto g(a)\}\rangle.$$

   We need to define $\langle Q;\sigma\rangle^*$ as $e\tilde{e} \Leftarrow c$ with $Q^* = e \Leftarrow c$ and $\sigma^* = \tilde{e}$ in order to have a correct translation. However, connecting the translation of the query and the translation of the substitution in this order is in general not possible, as is shown by considering again 1 above.

4. A third alternative definition of resolution is obtained by replacing clause 5 of Definition 2.4 by

   $$Q' = b_1, \ldots, b_m, a_1, \ldots a_n$$

   and clause 6 by

   $$\sigma' = \sigma\tau.$$

   Using this definition, we have a resolution step

   $$\langle f(a,z);\epsilon\rangle \Rightarrow_{C_1} \langle h(x,y);\{x \mapsto a, z \mapsto g(y)\}\rangle.$$

   The translation of $\langle f(a,z);\epsilon\rangle$ is $[z := f(a)]$. For the translation of $\langle h(x,y);\{x \mapsto a, z \mapsto g(y)\}\rangle$, there are two alternatives. If the environment is built from the translation of the substitution and the translation of the query, in that order, we obtain $\langle h(x,y);\{x \mapsto a, z \mapsto g(y)\}\rangle^* = [x := a][z := g(y)][y := h(x)]$. Otherwise, in the reverse order, we obtain $\langle h(x,y);\{x \mapsto a, z \mapsto g(y)\}\rangle^* = [y := h(x)][x := a][z := g(y)]$. Neither of them is the desired $[z := g(y)][y := h(x)][x := a]$. Note that the problem is present since we consider environments to be lists of declarations; if the declarations are unordered the problem disappears and this second alternative definition of resolution can be used.

This example of course doesn't show that the translation we employ is the only possible one. However it illustrates that some subtleties have to be taken into account.

The Main Result. The main result is that using the translation of the present paper, a resolution step using a clause C corresponds to a rewrite sequence consisting of at least one step using the translation of C. In a diagram:

$$\begin{tikzcd}
\langle Q;\sigma\rangle \arrow[r, phantom, "\Rightarrow_C"] \arrow[d] & \langle Q';\sigma'\rangle \arrow[d] \\
\langle Q;\sigma\rangle^* \arrow[r, phantom, "\to^+_{C^*}"] & \langle Q';\sigma'\rangle^*
\end{tikzcd}$$

This is expressed in Theorem 4.9 which is proved by induction on the translation of the clause C. In the proof of this theorem we work modulo the following equations concerning conditional environments:

$$e_1e_2[\mathsf{T}_i(\vec{x}) := \mathsf{T}_i(\vec{s})]e_3 \Leftarrow c[\mathsf{T}_i(\vec{x}) := \mathsf{T}_i(\vec{s})]e_4 = e_1[\mathsf{T}_i(\vec{x}) := \mathsf{T}_i(\vec{s})](e_2\sigma)e_3 \Leftarrow (c\sigma)e_4 \tag{4.2}$$

with $\sigma = \{\vec{x} \mapsto \vec{s}\}$.

$$e_1 \Leftarrow ce_2 = e_1e_2 \Leftarrow ce_2 \tag{4.3}$$

Theorem 4.9. *If $\langle Q;\sigma\rangle \Rightarrow_C \langle Q';\sigma'\rangle$, then $\langle Q;\sigma\rangle^* \to^+_{C^*} \langle Q';\sigma'\rangle^*$.*

Proof. See the Appendix. □

Properties of the Translation. A consequence of Theorem 4.9 is that an infinite resolution sequence is translated into an infinite rewrite sequence. Hence termination of a logic program is implied by termination of its translation.

Further, the translation of $\langle\Box;\sigma\rangle$ is a conditional environment in normal form. Hence a successful resolution sequence is translated into a rewrite sequence ending in an conditional environment in normal form.

A resolution step concerns by definition the leftmost atom of a query, whereas in the codomain of the translation no rewriting strategy is imposed. As a consequence, the translation of a query that cannot perform a resolution step is not necessarily a conditional environment in normal form. Consider for instance the logic program consisting of one clause:

$$C = b \leftarrow b.$$

The query a, b cannot perform a resolution step. However, its translation $[\,] \Leftarrow a \otimes b$ rewrites in one step to itself using $C^* = b \to \mathsf{T} \Leftarrow b$. This means that termination with failure cannot be detected in the translation.

This problem can be solved by enriching the translation with a marking device. An idea to do this is as follows. The subterm of the translation of a query corresponding to the ith atom of the query (from the left) is labelled by i. In the translation of a clause, a variable is added that is meant to range over labels. The right-hand side of the translation of a clause is labelled as follows: the subterm corresponding to the ith atom in the body of the clause is labelled by the sequence consisting of the extra variable followed by i. Then, we impose the strategy that only a subterm with a minimal label (with respect to the natural extension of the ordering on natural numbers to sequences of natural numbers) can be rewritten. In fact, this is not a rewrite strategy according to the usual definition. An example clarifying this idea is as follows.

$$\begin{aligned} f(x,y) &\leftarrow g(x), h(x,y) \\ g(a) &\leftarrow \\ h(a,c) &\leftarrow \\ h(b,c) &\leftarrow \end{aligned}$$

with modes $f, h :\downarrow \times \uparrow$ and $g :\downarrow$. Its labelled translation is the following conditional rewriting system:

$$\begin{aligned} f(x)^l &\rightarrow y[y := h(x)^{l2}] \Leftarrow g(x)^{l1} \\ g(a)^l &\rightarrow \mathsf{T} \\ h(a)^l &\rightarrow c \\ h(b)^l &\rightarrow c \end{aligned}$$

with l a variable that is meant to range over labels. The resolution sequence

$$\langle f(b,y);\epsilon\rangle \Rightarrow \langle g(b), h(b,y);\epsilon\rangle$$

ends with failure. It is translated into the following rewrite sequence:

$$\begin{aligned} [y := f(b)^1] \Leftarrow \mathsf{T} &\quad \rightarrow \\ [y := z[z := h(b)^{12}]] \Leftarrow g(b)^{11} &\quad = \\ [y := z][z := h(b)^{12}] \Leftarrow g(b)^{11} & \end{aligned}$$

The conditional environment $[y := h(b)^{12}] \Leftarrow g(b)^{11}$ cannot be rewritten since the subterm with minimal label, $g(b)^{11}$, is in normal form.

Note that the successful resolution sequence

$$\langle f(a,y);\epsilon\rangle \Rightarrow \langle g(a), h(a,y);\epsilon\rangle \Rightarrow \langle h(a,y);\epsilon\rangle \Rightarrow \langle \Box; y \mapsto c\rangle$$

is translated into the following rewrite sequence ending in a conditional environment in normal form:

$$[y := f(a)^1] \Leftarrow \mathsf{T} \rightarrow [y := h(b)^{12}] \Leftarrow g(b)^{11} \rightarrow [y := h(b)^{12}] \Leftarrow \mathsf{T} \rightarrow [y := c] \Leftarrow \mathsf{T}.$$

If we use the labelled translation as informally described above, a rewrite sequence starting in the translation of a query Q corresponds to a resolution sequence starting in Q.

We summarize the properties of the translation as discussed above:

- A successful resolution sequence is translated into a rewrite sequence ending in a conditional environment in normal form.
- An infinite resolution sequence in translated into an infinite rewrite sequence.
- Using a marking device it is possible to translate a resolution sequence ending with failure in a "failing" rewrite sequence.

Optimizing the translation. Finally, the translation of the clauses of a logic program can be put into a more readable form by evaluating the explicit substitutions in the usual way. Then the statement of Theorem 4.9 doesn't hold anymore; instead we obtain the weaker result that $\langle Q;\sigma\rangle \Rightarrow_C \langle Q';\sigma'\rangle$ implies $\langle Q;\sigma\rangle^* \rightarrow^*_{C^*} \langle Q';\sigma'\rangle^*$, that is, one resolution step is translated into a (possibly empty) rewrite sequence.

Example. As an example we consider quicksort. Using the notational conventions of Prolog, this program consists of the following clauses:

```
q([], []).
q([X | Xs], Ys) :-
  p(X, Xs, Ls, Bs),
  q(Ls, Ls'), q(Bs, Bs'),
  app(Ls', [X | Bs'], Ys).
p(X, [], [], []).
p(X, [Y | Ys], [Y | Ls], Bs) :-
  X >= Y, p(X, Ys, Ls, Bs).
p(X, [Y | Ys], Ls, [Y | Bs]) :-
  X < Y, p(X, Ys, Ls, Bs).
app([], Xs, Xs).
app([X | Xs], Ys, [X | Zs]) :-
  app(Xs, Ys, Zs).
```

with modes q: $\downarrow \times \uparrow$, p: $\downarrow \times \downarrow \times \uparrow \times \uparrow$, app: $\downarrow \times \downarrow \times \uparrow$. Translating this program yields a conditional rewriting system. If we evaluate some of the explicit conditions in this conditional rewriting system, and use the notational conventions of Gofer, then we obtain the following functional program.

```
q([])           =  []
q(x:xs)         =  app(q(ls), (x:q(bs)))
                   where (ls, bs) = p(x, xs)
p(x, [])        =  ([], [])
p(x, (y:ys))    =  ((y:ls), bs) if x >= y
                =  (ls, (y:bs)) if x < y
                   where (ls, bs) = p(x, ys)
app([], xs)     =  xs
app((x:xs), ys) =  (x:app(xs, ys))
```

5. Related Work

The first to study the relationship between logic programming and functional programming is Reddy. In [17], he presents an extensive study concerning modes and modes assignments. Further, he presents techniques to infer the moding of a logic program given the modes of a goal, and to infer whether a moding yields ground answer substitutions. This work is later on extended and improved in various studies of well-moded, simply moded and nicely moded logic programs (see [2]). Moreover, Reddy defines a translation from logic programs to functional programs. The main difference between a logic program and its translation is the notation.

Later work concerning translations from logic programming to functional programming is mainly concerned with termination of logic programming. Various papers present a translation from logic programs to rewriting system that permits to derive termination of the logic program from termination of its translation.

A first such result is due to Krishna Rao, Kapur and Shyamasundar. In [11], they define a translation from well-moded logic programs into (unconditional) term rewriting systems. The translation is defined by means of an algorithm and differs from the one presented in the present paper. For instance, the translation of a relation symbol with more than one output position is not unique. The translation might transform a terminating logic program into a non-terminating rewriting system.

Ganzinger and Waldmann present in [10] an elegant and simple translation from well-moded logic programs to conditional term rewriting systems with extra variables. A clause

$$r_0(\vec{s}_0, \vec{t}_0) \leftarrow r_1(\vec{s}_1, \vec{t}_1), \ldots, r_m(\vec{s}_m, \vec{t}_m)$$

in the notation as used in the present paper is translated into the conditional rewrite rule

$$r_0^{in}(\vec{s}_0) \to r_0^{out}(\vec{t}_0) \Leftarrow r_1^{in}(\vec{s}_1) \to r_1^{out}(\vec{t}_1) \ldots r_m^{in}(\vec{s}_m) \to r_m^{out}(\vec{t}_m).$$

We argued in Section 4 that this translation, although elegant, is not satisfactory for the purposes of the present paper. Ganzinger and Waldmann use an earlier result by Ganzinger stating that conditional rewriting systems satisfying some requirement corresponding to well-modedness for logic programs, are terminating if they are *quasi-reductive*. In this way a method to infer termination of logic programs: a logic program is terminating if its translation is quasi-reductive. This method can be used to prove termination of some logic programs for which the method due to Krishna Rao, Kapur and Shyamasundar fails. On the other hand, the method due to Ganzinger and Waldmann doesn't yield termination of all terminating logic programs.

The work by Ganzinger and Waldmann is extended by Avenhaus and Loría-Sáenz. In [6], they show that a quasi-reductive conditional term rewriting system is confluent if every critical pair is either *unfeasible* or *context-joinable*. Together with the earlier work by Ganzinger and Waldmann, this yields a method to derive whether a logic program terminates in a unique result.

Another method to derive termination of a logic program by means of a translation to a term rewriting system is due to Aguzzi and Modigliani [1].

Massimo Marchiori describes in [12] a translation from well-moded and simply moded (hence in the terminology of the present paper simply moded) logic programs to term rewriting systems. The translation satisfies the property that a logic program is terminating if and only if its translation is terminating.

Arts and Zantema present in [5] (see also [4]) an algorithm to translate logic programs into (unconditional) constructor systems. A clause is translated into possibly more than one rewrite rule. The first rewrite rule has as left-hand side the input part of the head of the clause, and as right-hand side the output part of the head of the clause, plus all variables occurring in the left-hand side. Then, the body atoms of a clause give rise to additional rewrite rules 'connected' to the first rewrite rule and to each other by passing variables. Arts and Zantema present a technique to prove termination of constructor systems that is particularly suitable to prove termination of the translation of logic programs, using their translation.

All translations mentioned above are different from the translation defined in the present paper. None of them presents a translation of a pairs consisting of a query and a substitution, or of a query alone. A detailed comparison of all the methods to infer termination of a logic program by means of a translation into a (conditional) term rewriting system is not available.

Massimo Marchiori presents in [13] a transformation from join conditional rewriting systems, where conditions have the form $s_1 \downarrow t_1, \ldots, s_m \downarrow t_m$ to unconditional rewriting systems, and a transformation from normal conditional rewriting systems, where conditions have the form $s_1 \to^* t_1, \ldots, s_m \to^* t_m$ with $t_1, \ldots, t_m$ closed normal forms, to unconditional rewriting systems. Since the translation of a well-moded logic program as defined by Ganzinger and Waldmann yields a conditional rewriting system with extra variables, the transformations defined by Marchiori cannot be applied directly to obtain an unconditional term rewriting system. However, using the basic idea of the transformations defined by Marchiori, the translation as defined by Ganzinger and Waldmann can be transformed into an unconditional term rewriting system which seems very close to the translation defined by Arts and Zantema.

The translation defined in the present paper can be used to define a transformation from conditional rewriting systems with extra variables, satisfying certain properties, to conditional rewriting systems without extra variables.

Acknowledgements
I gratefully acknowledge helpful and motivating discussions with Krzysztof Apt, Gilles Barthe, Sandro Etalle, Pieter Hartel, Aart Middeldorp, Vincent van Oostrom, Wim Vree and Mark Wielaard.

References

1. G. Aguzzi and U. Modigliani. Proving termination of logic programs by transforming them into equivalent term rewriting systems. In R.K. Shyamasundar, editor, *Proceedings of the 13th conference on Foundations of Software Technology and Theoretical Computer Science*, number 761 in Lecture Notes in Computer Science, Bombay, India, December 1993.
2. K.R. Apt. *From Logic Programming to Prolog.* Prentice-Hall, 1997.
3. K.R. Apt and S. Etalle. On the unification free prolog programs. In A. Borzyszkowski and S. Sokolowski, editors, *Proceedings of the Conference on Mathematical Foundations of Computer Science (MFCS '93)*, number 711 in Lecture Notes in Computer Science, pages 1–19, Gdańsk, Poland, 1993. Springer Verlag.
4. T. Arts. *Automatically proving termination and innermost termination of term rewriting systems.* PhD thesis, Universiteit Utrecht, May 1997.
5. T. Arts and H. Zantema. Termination of logic programs using semantic unificition. In M. Proietti, editor, *Proceedings of the 5th International Workshop on Logic Program Synthesis and Transformation (LOPSTR '95)*, volume 1048 of *Lecture Notes in Computer Science*, pages 219–233, Utrecht, The Netherlands, September 1995. Springer Verlag.
6. J. Avenhaus and C. Loría-Sáenz. On conditional rewrite systems with extra variables and deterministic logic programs. In F. Pfenning, editor, *Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning (LPAR '94)*, volume 822 of *Lecture Notes in Artificial Intelligence*, pages 215–229, Kiev, Ukraine, July 1994. Springer Verlag.
7. A. Bockmayr. *Contributions to the Theory of Logic-Functional Programming.* PhD thesis, University of Karlsruhe, Karlsruhe, Germany, 1990. (in German).
8. A. Bockmayr. Conditional narrowing modulo a set of equations. *Applicable Algebra in Engineering, Communication and Computing*, 4(3):147–168, 1993.
9. P. Dembiński and J. Małuszyński. AND-parallelism with intelligent backtracking for annotated logic programs. In *Proceedings of the International Symposium on Logic Programming*, Boston, USA, 1985.
10. H. Ganzinger and U. Waldmann. Termination proofs of well-moded logic programs via conditional rewrite systems. In M. Rusinowitch and J.L. Rémy, editors, *Proceedings of the third international workshop on conditional term rewriting systems (CTRS '92)*, number 656 in Lecture Notes in Computer Science, pages 430–437, Pont-à-Mousson, July 1993.
11. M.R.K. Krishna Rao, D. Kapur, and R.K. Shyamasundar. A transformation methodology for proving termination of logic programs. In E. Börger, G. Jäger, H. Kleine Büning, and M.M. Richter, editors, *Proceedings of the 5th Workshop on Computer Science Logic (CSL '91)*, pages 213–226, Berne, Switzerland, October 1991. Springer Verlag.
12. M. Marchiori. Logic programs as term rewriting systems. In G. Levi and M. Rodríguez-Artalejo, editors, *Proceedings of the 4th International Conference on Algebraic and Logic Programming (ALP '94)*, number 850 in Lecture Notes in Computer Science, pages 223–241, Madrid, Spain, September 1994.
13. M. Marchiori. Unravelings and ultra-properties. In M. Hanus and M. Rodríguez-Artelego, editors, *Proceedings of the 5th International Conference on Algebraic and Logic Programming*, number 1139 in Lecture Notes in Computer Science, pages 107–121, Aachen, Germany, September 1996.
14. C.S. Mellish. The automatic generation of mode declarations for Prolog programs. Technical Report DAI Research Paper 163, University of Edinburgh, 1981.
15. A. Middeldorp and E. Hamoen. Completeness results for basic narrowing. *AAECC*, 5:213–253, 1994.

16. F. van Raamsdonk. Translating logic programs into conditional rewriting systems. In L. Naish, editor, *Proceedings of the Fourteenth International Conference on Logic Programming (ICLP '97)*, pages 168–182, Leuven, Belgium, July 1997. MIT Press.

17. U.S. Reddy. Transformation of logic programs into functional programs. In *Proceedings of the Symposium on Logic Programming*, pages 187–196, Atlantic City, New Jersey, USA, February 1984. IEEE Computer Society, Silver Spring, MD.

18. Z. Somogyi, F. Henderson, and T. Conway. The execution algorithm of Mercury: an efficient purely declarative logic programming language. *Journal of Logic Programming*, 29(1-3):17–64, 1996.

Appendix

Proof of Theorem 4.9.

Let $\langle Q;\sigma\rangle \Rightarrow_C \langle Q';\sigma'\rangle$ be a resolution step in a simply moded program. Suppose that

$$C = h \leftarrow b_1,\dots,b_m$$

for some $m \geq 0$ with $h = r(\vec{s},\vec{t})$, and that

$$Q = a_1,\dots,a_n$$

for some $n > 0$ with $a_1 = r(\vec{u}_1,\vec{v}_1)$. Let $\tau = \tau_1\tau_2$ be a most general unifier of h and a_1 with $\vec{s}\tau_1 = \vec{u}_1$ and $\vec{v}_1\tau_2 = \vec{t}\tau_1$. Then

$$Q' = b_1\tau_1,\dots,b_m\tau_1, a_2\tau_2,\dots,a_n\tau_2$$

and

$$\sigma' = \sigma\tau_2.$$

Suppose that $\sigma^* = \tilde{e}$ and $(a_2,\dots,a_n)^* = e \Leftarrow c$, so

$$\langle a_2,\dots,a_n;\sigma\rangle^* = \tilde{e}\, e \Leftarrow c.$$

Let $C_p = h \leftarrow b_p,\dots,b_m$ for every $p \in \{1,\dots,m+1\}$. By induction on $m+1-p$ we prove the following:

$$\text{if } \langle Q;\sigma\rangle \Rightarrow_{C_p} \langle Q_p;\sigma_p\rangle \text{ then } \langle Q;\sigma\rangle^* \rightarrow^+_{C_p^*} \langle Q_p;\sigma_p\rangle^*.$$

Note that $\sigma_p = \sigma\tau_2$ for every $p \in \{1,\dots,m+1\}$.

1. Suppose that $p = m+1$. We have $C_p = r(\vec{s},\vec{t}) \leftarrow$ and

   $$Q_p = a_2\tau_2,\dots,a_n\tau_2.$$

   Further

   $$C_p^* = r(\vec{s}) \rightarrow (\mathsf{T}_i(\vec{t}) \Leftarrow \mathsf{T})$$

   with $i = |\vec{t}|$ $(= |\vec{v}_1|)$. Two cases are distinguished.

   (a) $i = 0$. Then we have

   $$Q^* = e \Leftarrow (r(\vec{u}_1) \otimes c)$$

and

$$Q_p^* = e \Leftarrow c.$$

Note that $\tau_2 = \epsilon$. We have

$$\begin{array}{rcl} \langle Q;\sigma\rangle^* & = & \tilde{e}\, e \Leftarrow (r(\vec{u}_1) \otimes c) \\ & = & \tilde{e}\, e \Leftarrow (r(\vec{s})\tau_1 \otimes c) \\ & \rightarrow_{C_p^*} & \tilde{e}\, e \Leftarrow (\mathsf{T} \otimes \mathsf{T} \otimes c) \\ & = & \tilde{e}\, e \Leftarrow c \\ & = & \langle Q_p;\sigma_p\rangle^*. \end{array}$$

(b) $i > 0$. Then we have

$$Q^* = e[\mathsf{T}_i(\vec{v}_1) := r(\vec{u}_1)] \Leftarrow c[\mathsf{T}_i(\vec{v}_1) := r(\vec{u}_1)]$$

and

$$Q_p^* = e\tau_2 \Leftarrow c\tau_2.$$

We have:

$$\begin{array}{rcl} \langle Q;\sigma\rangle^* & = & \tilde{e}\, e[\mathsf{T}_i(\vec{v}_1) := r(\vec{u}_1)] \Leftarrow c[\mathsf{T}_i(\vec{v}_1) := r(\vec{u}_1)] \\ & = & \tilde{e}\, e[\mathsf{T}_i(\vec{v}_1) := r(\vec{s})\tau_1] \Leftarrow c[\mathsf{T}_i(\vec{v}_1) := r(\vec{s})\tau_1] \\ & \rightarrow^+ & \tilde{e}\, e[\mathsf{T}_i(\vec{v}_1) := \mathsf{T}_i(\vec{t})\tau_1] \Leftarrow c[\mathsf{T}_i(\vec{v}_1) := \mathsf{T}_i(\vec{t})\tau_1] \\ & = & \tilde{e}\,[\mathsf{T}_i(\vec{v}_1) := \mathsf{T}_i(\vec{t})\tau_1]\,(e\tau_2) \Leftarrow c\tau_2 \\ & = & \langle Q_p;\sigma\tau_2\rangle^* \\ & = & \langle Q_p;\sigma_p\rangle^*. \end{array}$$

Here $\tau_2 = \{v_1 \mapsto t_1\tau_1, \ldots, v_i \mapsto t_1\tau_i\}$.

Note that we make use of equation 4.2.

2. Suppose that $1 \le p < m+1$. We have $C_p = h \leftarrow b_p, b_{p+1}, \ldots, b_m$ and

$$Q_p = b_p\tau_1, \ldots, b_m\tau_1, a_2\tau_2, \ldots, a_n\tau_2.$$

Let

$$\begin{array}{rcl} C_{p+1}^* & = & r(\vec{s}) \rightarrow (r_{p+1} \Leftarrow d_{p+1}), \\ Q_{p+1}^* & = & e_{p+1} \Leftarrow c_{p+1}. \end{array}$$

The induction hypothesis is:

$$\langle Q;\sigma\rangle^* \rightarrow^+_{C_{p+1}^*} \langle Q_{p+1};\sigma_{p+1}\rangle^*.$$

Let $b_p = r_p(\vec{s}_p, \vec{t}_p)$ and $i = |\vec{t}|$ $(= |\vec{v}_1|)$ and $j = |\vec{t}_p|$. Four cases are distinguished.

(a) $i = 0$ and $j = 0$. Then we have:

$$\begin{array}{rcl} Q^* & = & e \Leftarrow r(\vec{u}_1) \otimes c, \\ C_p^* & = & r(\vec{s}) \rightarrow r_{p+1} \Leftarrow r_p(\vec{s}_p) \otimes d_{p+1}, \\ Q_p^* & = & e_{p+1} \Leftarrow r_p(\vec{s}_p)\tau_1 \otimes c_{p+1}. \end{array}$$

The induction hypothesis in detail is:

$$\begin{array}{rcl} \langle Q;\sigma\rangle^* & = & \tilde{e}\, e \Leftarrow r(\vec{s})\tau_1 \otimes c \\ & \to_{C^*_{p+1}} & \tilde{e}\, e \Leftarrow d_{p+1}\tau_1 \otimes r_{p+1}\tau_1 \otimes c \\ & = & \tilde{e}_{p+1}\, e_{p+1} \Leftarrow c_{p+1}. \end{array}$$

We have:

$$\begin{array}{rcl} \langle Q;\sigma\rangle^* & = & \tilde{e}\, e \Leftarrow r(\vec{u}_1) \otimes c \\ & = & \tilde{e}\, e \Leftarrow r(\vec{s})\tau_1 \otimes c \\ & \to_{C^*_p} & \tilde{e}\, e \Leftarrow r_p(\vec{s}_p)\tau_1 \otimes d_{p+1}\tau_1 \otimes r_p\tau_1 \otimes c \\ & = & \tilde{e}_{p+1}\, e_{p+1} \Leftarrow r_p(\vec{s}_p)\tau_1 \otimes c_{p+1} \\ & = & \langle Q_p;\sigma\rangle^*. \end{array}$$

(b) $i = 0$ and $j > 0$. Then we have:

$$\begin{array}{rcl} Q^* & = & e \Leftarrow r(\vec{u}_1) \otimes c, \\ C^*_p & = & r(\vec{s}) \to r_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)] \Leftarrow d_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)], \\ Q^*_p & = & e_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1] \Leftarrow c_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1]. \end{array}$$

The induction hypothesis in detail is:

$$\begin{array}{rcl} (Q,\sigma)^* & = & \tilde{e}\, e \Leftarrow r(\vec{s})\tau_1 \otimes c \\ & \to_{C^*_{p+1}} & \tilde{e}\, e \Leftarrow d_{p+1}\tau_1 \otimes r_{p+1}\tau_1 \otimes c \\ & = & \tilde{e}_{p+1}\, e_{p+1} \Leftarrow c_{p+1}. \end{array}$$

We have:

$$\begin{array}{rcl} \langle Q;\sigma\rangle^* & = & \tilde{e}\, e \Leftarrow r(\vec{u}_1) \otimes c \\ & = & \tilde{e}\, e \Leftarrow r(\vec{s})\tau_1 \otimes c \\ & \to_{C^*_p} & \tilde{e}\, e \Leftarrow r_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)]\tau_1 \otimes \\ & & d_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)]\tau_1 \otimes c \\ & = & \tilde{e}\, e \Leftarrow r_{p+1}\tau_1[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1] \otimes \\ & & d_{p+1}\tau_1[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1] \otimes c \\ & = & \tilde{e}_{p+1}\, e_{p+1} \Leftarrow c_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1] \\ & = & \tilde{e}_{p+1}\, e_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1] \\ & & \Leftarrow c_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1] \\ & = & \langle Q_p;\sigma\rangle^*. \end{array}$$

Note that we make use of equation 4.3.

(c) $i > 0$ and $j = 0$. Then we have:

$$\begin{array}{rcl} Q^* & = & e[\mathsf{T}_i(\vec{v}_1) := r(\vec{u}_1)] \Leftarrow c[\mathsf{T}_i(\vec{v}_1) := r(\vec{u}_1)], \\ C^*_p & = & r(\vec{s}) \to r_{p+1} \Leftarrow r_p(\vec{s}_p) \otimes d_{p+1}, \\ Q^*_p & = & e_{p+1} \Leftarrow r_p(\vec{s}_p)\tau_1 \otimes c_{p+1}. \end{array}$$

The induction hypothesis is:

$$\begin{array}{rcl} \langle Q;\sigma\rangle^* & = & \tilde{e}\, e[\mathsf{T}_i(\vec{v}) := r(\vec{s})\tau_1] \Leftarrow c[\mathsf{T}_i(\vec{v}) := r(\vec{s})\tau_1] \\ & \to^+_{C^*_{p+1}} & \tilde{e}_{p+1}\, e_{p+1} \Leftarrow c_{p+1}. \end{array}$$

We have:

$$\begin{array}{rcl}
\langle Q;\sigma\rangle^* & = & \tilde{e}\, e[\mathsf{T}_i(\vec{v}_1) := r(\vec{u}_1)] \Leftarrow c[\mathsf{T}_i(\vec{v}_1) := r(\vec{u}_1)] \\
& = & \tilde{e}\, e[\mathsf{T}_i(\vec{v}_1) := r(\vec{s})\tau_1] \Leftarrow c[\mathsf{T}_i(\vec{v}_1) := r(\vec{s})\tau_1] \\
& \to^+_{C^*_p} & \tilde{e}_{p+1}\, e_{p+1} \Leftarrow r(\vec{s}_p)\tau_1 \otimes c_{p+1} \\
& = & \langle Q_p;\sigma\tau_2\rangle^*.
\end{array}$$

(d) $i > 0$ and $j > 0$. Then we have:

$$\begin{array}{rcl}
Q^* & = & e[\mathsf{T}_i(\vec{v}_1) := r(\vec{u}_1)] \Leftarrow c[\mathsf{T}_i(\vec{v}_1) := r(\vec{u}_1)], \\
C^*_p & = & r(\vec{s}) \to r_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)] \Leftarrow d_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)], \\
Q^*_p & = & e_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1] \Leftarrow c_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1].
\end{array}$$

Note that $\mathsf{Dom}(\tau_1) = \mathsf{V}(\vec{s})$ and since C is simply moded we have $\mathsf{V}(\vec{s}) \cap \vec{t}_p = \emptyset$. Hence $\vec{t}_p\tau_1 = \vec{t}_p$. The induction hypothesis in detail is:

$$\begin{array}{rcl}
(Q,\sigma)^* & = & \tilde{e}\, e[\mathsf{T}_i(\vec{v}) := r(\vec{s})\tau_1] \Leftarrow c[\mathsf{T}_i(\vec{v}) := r(\vec{s})\tau_1] \\
& \to^+_{C^*_{p+1}} & e\,\tilde{e}[\mathsf{T}_i(\vec{v}) := r_{p+1}\tau_1] \Leftarrow d_{p+1}\tau_1 \otimes c[\mathsf{T}_i(\vec{v}) := r_{p+1}\tau_1] \\
& = & \tilde{e}_{p+1}\, e_{p+1} \Leftarrow c_{p+1}.
\end{array}$$

We have:

$$\begin{array}{rcl}
\langle Q;\sigma\rangle^* & = & \tilde{e}\, e[\mathsf{T}_i(\vec{v}_1) := r(\vec{u}_1)] \Leftarrow c[\mathsf{T}_i(\vec{v}_1) := r(\vec{u}_1)] \\
& = & \tilde{e}\, e[\mathsf{T}_i(\vec{v}_1) := r(\vec{s})\tau_1] \Leftarrow c[\mathsf{T}_i(\vec{v}_1) := r(\vec{s})\tau_1] \\
& \to^+_{C^*_p} & \tilde{e}\, e[\mathsf{T}_i(\vec{v}) := r_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)]\tau_1] \\
& & \Leftarrow d_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)]\tau_1 \otimes c[\mathsf{T}_i(\vec{v}) := r_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)]\tau_1] \\
& = & \tilde{e}\, e[\mathsf{T}_i(\vec{v}) := r_{p+1}\tau_1][\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1] \Leftarrow \\
& & d_{p+1}\tau_1[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1] \otimes c[\mathsf{T}_i(\vec{v}) := r_{p+1}\tau_1][\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1] \\
& = & \tilde{e}_{p+1}\, e_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1] \Leftarrow c_{p+1}[\mathsf{T}_j(\vec{t}_p) := r_p(\vec{s}_p)\tau_1] \\
& = & \langle Q_p;\sigma\tau_2\rangle^*.
\end{array}$$

□