Electronic Notes in Theoretical Computer Science 4 (1996)

# Bi-rewriting Rewriting Logic*

W. Marco Schorlemmer

*Institut d'Investigació en Intel·ligència Artificial*
*Consell Superior d'Investigacions Científiques*
*Campus UAB, E-08193 Bellaterra, Catalunya*
marco@iiia.csic.es

**Abstract**

Rewriting logic appears to have good properties as logical framework, and can be useful for the development of programming languages which attempt to integrate various paradigms of declarative programming. In this paper I propose to tend towards the operational semantics for such languages by basing it on *bi-rewrite systems* and *ordered chaining calculi* which apply rewrite techniques to first-order theories with arbitrary possibly non-symmetric transitive relations, because this was an important breakthrough for the automation of deduction in these kind of theories. I show that a proof calculus based on the bi-rewriting technique may serve as framework of different proof calculi, by analizing those of equational logic and Horn logic, and presenting them as specific cases of bi-rewrite systems. Deduction is then essentially bi-rewriting a theory of rewriting logic. Since recently the interest in specifications based on theories with transitive relations has arisen, the result of this research towards a general framework for bi-rewriting based operational semantics of several programming paradigms will also be very useful for the development of rapid prototyping tools for these kind of specifications.

## 1 Introduction

Term rewriting has been mainly used as a technique for the deduction in equational theories, and was studied thoroughly in the context of rewrite systems [10,40,20]. But recently it has been noticed that, since rewriting is done only in one direction, it is not limited to equivalence relations, but also applicable on arbitrary transitive relations. Indeed, Meseguer showed that the implicit logic underlying rewrite systems is not equational logic, but *rewriting logic* [31]. Meseguer put the strength of his research in developing a strong mathematical semantics of rewriting logic by formulating it as a logic of action and concurrent change.

Similar observation were made independently by Levy and Agustí, as they studied mechanisms for automating the deduction in theories involving sub-

* Supported by project DISCOR (TIC 94-0847-C02-01) funded by the CICYT

set inclusions. They applied rewrite techniques to inclusional theories [25] and generalized the notions of Church-Rosser and termination of rewrite systems to the more general framework called *bi-rewrite systems* [26]. This was an important breakthrough in automated deduction with arbitrary transitive relations: Bachmair and Ganzinger based on Levy and Agustí's work their generalization from *superposition calculi* for full first-order theories with equality [5] to *ordered chaining calculi* for theories with arbitrary transitive relations, besides equality [6]. Actually their calculi apply rewrite techniques (i.e. the use of ordering restrictions on terms and atoms involved in inferences) to the original chaining inference first stated by Slagle [44].

Meseguer's rewriting logic appears to have good properties as logical framework, and, following its approach on 'general logics' [30], different logics of interest have been mapped to it [28]. Therefore a proof calculus for rewriting logic may be useful as framework for a variety of other proof calculi, which can also be mapped to it, specially if such a proof calculus is an effective and, even better, a very efficient one. That's why rewriting logic serves as basis for the development of programming languages like Maude [32], which attempt to unify the paradigms of functional, relational and concurrent object-oriented programming. It was Parker who also advocated programming on non-symmetric transitive relations like preorder or partial order relations for generalizing and subsequently combining several different programming paradigms, symbolic or numeric, like functional and logic programming among others [37,38]. Another recent approach for integrating functional and logic programming, based on rewriting logic, but taking possibly non-deterministic lazy functions as the fundamental notion, has been done by González-Moreno et al. [13].

In order to deal in practice with such multi-paradigm languages like e.g. Maude it is necessary to provide them with an efficient operational semantics. Therefore, instead of formulating it on the straightforward proof calculus defined by the deduction rules of rewriting logic, I argue that by applying the known results about automated deduction in theories with transitive relations, we will be able to define a general framework for the integration of different operational semantics in a more promising way, from the efficiency point of view. In this paper I conjecture that, since the work on bi-rewriting and ordered chaining done by Levy and Agustí, and Bachmair and Ganzinger respectively is suitable for mechanization, their results will be useful for stating such operational semantics framework.

## 2 Preliminaries

In rewriting logic, a rewrite theory $\mathcal{R}$ can be described as a 4-tuple $(F, A, L, R)$, where $(F, A)$ is a signature consisting of a set $F$ of function symbols and a set $A$ of structural axioms ($F$-equations like associativity or commutativity), $L$ is a set of labels, and $R$ is a set of sentences of the form $r : [s]_A \Rightarrow [t]_A$ (i.e. labeled rules with $r \in L$) among $A$-equivalence classes of first-order terms

**Reflexivity**:

$$\frac{}{[t] \Rightarrow [t]}$$

**Congruence**: For each $f \in F$,

$$\frac{[s_1] \Rightarrow [t_1] \cdots [s_n] \Rightarrow [t_n]}{[f(s_1, \ldots, s_n)] \Rightarrow [f(t_1, \ldots, t_n)]}$$

**Replacement**: For each rule $[s] \Rightarrow [t] \in R$,

$$\frac{[u_1] \Rightarrow [v_1] \cdots [u_n] \Rightarrow [v_n]}{[s\langle x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\rangle] \Rightarrow [t\langle x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\rangle]}$$

where $x_1, \ldots, x_n$ are the variables occurring in either $s$ or $t$.

**Transitivity**:

$$\frac{[s] \Rightarrow [t] \cdots [t] \Rightarrow [u]}{[s] \Rightarrow [u]}$$

Fig. 1. Deduction rules of rewriting logic

$s, t \in \mathcal{T}(F, X)$ over a denumerable set $X$ of variables [1].

In order to simplify the exposition of the ideas presented in this paper, I will only consider unlabeled rewrite theories, i.e. where rules in $R$ are of the form $[s]_A \Rightarrow [t]_A$. Therefore we can describe such a rewrite theory by means of the triple $(F, A, R)$. When the set of axioms $A$ is clear from the context I will denote the equivalence class of a term $t$ with $[t]$ instead of $[t]_A$.

Given a term expression $t$, $t|_p$ denotes the subterm occurring at position $p$. If this occurrence is replaced by term $v$, we will denote it with $t[v]_p$. A *substitution* $\sigma = \langle x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\rangle$ is a mapping from a finite set $\{x_1, \ldots, x_n\} \subseteq X$ of variables to $\mathcal{T}(F, X)$, extended as a morphism to a mapping from $\mathcal{T}(F, X) \longrightarrow \mathcal{T}(F, X)$. I will use substitutions in postfix notation.

The entailment of sentences $[s] \Rightarrow [t]$ from a rewrite theory $\mathcal{R}$, denoted $\mathcal{R} \vdash_{RWL} [s] \Rightarrow [t]$ is defined by the set of deduction rules given in Figure 1. A rewrite theory $\mathcal{R}$ induces the reachability relation '$\rightarrow_{\mathcal{R}}$', such that $[s] \rightarrow_{\mathcal{R}} [t]$ if we can obtain $[t]$ from $[s]$ by a finite amount of applications of the deduction rules of Figure 1.

An *ordering* $\succ$ is an irreflexive, transitive binary relation. It is a *reduction ordering* if additionally it is well-founded (no infinite sequences of the form $t_1 \succ t_2 \succ \cdots$ exist), monotonic ($u \succ v$ implies $s[u]_p \succ s[v]_p$) and stable under substitutions ($s \succ t$ implies $s\sigma \succ t\sigma$). *Path orderings* define reduction orderings [2] constructing them directly from a well-founded ordering over the symbols of the signature —the *precedence*— by exploring paths in the tree structure of the terms. An example of path ordering is the *lexicographic path*

[1] Actually sentences of rewriting logic are conditional rules [31], but here I will only consider unconditional ones.

[2] Actually they define *simplification orderings* which are reduction orderings satisfying the subterm property $t \succ t|_p$.

*ordering*. For a complete survey on termination orderings we refer to [9].

## 3 Proof Calculi for Rewriting Logic

A straightforward proof calculus for rewriting logic is defined by the category with equivalent classes of terms as objects and proof terms as morphisms [28]. Proof terms are built by the deduction rules defining the entailment relation of rewriting logic given in Figure 1 modulo those equations on proof terms, which identify equivalent proofs. Such a proof calculus is based on the following variant of Birkhoff's theorem [8] for the non-symmetric relation '$\Rightarrow$' of rewriting logic:

**Lemma 3.1** *Given a rewrite theory $\mathcal{R}$, if '$\rightarrow_{\mathcal{R}}$' denotes the reachability relation induced by the rules of $\mathcal{R}$, then $\mathcal{R} \vdash_{RWL} [s] \Rightarrow [t]$ if and only if $[s] \rightarrow_{\mathcal{R}} [t]$.*

Though for finite theory presentations a decision procedure based on Birkhoff's theorem is implementable (since the set of all theorems of $\mathcal{R}$ is recursively enumerable), it is well known, from equational logic [3], that such a procedure is absolutely intractable and awkward to implement. By first orienting the equations of a theory presentation following a reduction ordering on terms, and subsequently completing such a presentation in order to satisfy the Church-Rosser property, a very efficient proof calculus for equational logic based on normal form computation can be given. But, though normal form computation doesn't have any sense within the more general rewrite theories, we still should consider a proof calculus for rewriting logic which takes such a reduction ordering on terms into account. The fact that sentences of rewriting logic have already an orientation does not imply that such orientation coincides with the direction of term reduction, i.e. $\rightarrow_{\mathcal{R}} \not\subseteq \succ$ in general.

### *3.1 Bi-rewrite systems*

By orienting the sentences $[s] \Rightarrow [t]$ of a given rewrite theory $\mathcal{R} = (F, A, R)$ following an ordering $\succ$ on terms, we obtain *two* separate rewrite relations $\Rightarrow \cap \succ$ and $\Rightarrow \cap \prec$, which I will denote '$\xrightarrow{\Rightarrow}$' and '$\xrightarrow{\Leftarrow}$', respectively, where '$\Rightarrow$' is the direction of the rules in $R$, and '$\longrightarrow$' is the direction of reduction of terms. We obtain in this way two separate rewrite systems, which form together a *bi-rewrite system* $\langle R_{\Rightarrow}, R_{\Leftarrow} \rangle$.

**Example 3.2** *Consider the rewrite theory $\mathcal{R} = (\{a, b, c, f\}, \emptyset, R)$, where $R$ is given below:*

$$R = \begin{cases} f(a, x) \Rightarrow x \\ f(x, c) \Rightarrow x \\ b \Rightarrow f(a, c) \end{cases}$$

[3] As pointed out by Meseguer in [31] equational logic is obtained from rewriting logic by adding the symmetry rule to its deduction rules.

*Orienting these rules, following e.g. a lexicographic path ordering based on signature precedence* $f \succ c \succ b \succ a$*, we obtain the following two rewrite systems* $R_{\Rightarrow}$ *and* $R_{\Leftarrow}$*:*

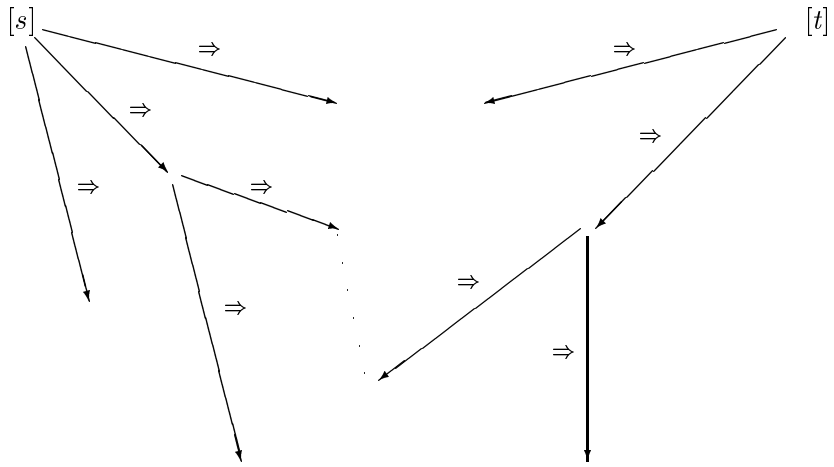
$$R_{\Rightarrow} = \begin{cases} f(a,x) \xrightarrow{\Rightarrow} x \\ f(x,c) \xrightarrow{\Rightarrow} x \end{cases} \qquad R_{\Leftarrow} = \left\{ f(a,c) \xrightarrow{\Leftarrow} b \right.$$

In order to have a decision algorithm for the word problem in a rewrite theory the bi-rewrite system needs to be *convergent*, i.e. it has to satisfy two properties: *Church-Rosser* and *termination*[4]. The system is Church-Rosser if whenever we have two equivalent classes of terms $[s]$ and $[t]$ such that $\mathcal{R} \vdash_{RWL} [s] \Rightarrow [t]$ a *bi-rewrite proof* between these equivalent classes exists, consisting of two paths, one using rules of $R_{\Rightarrow}$ and the other using rules of $R_{\Leftarrow}$, which join together in a common equivalent class:

$$[s] \xrightarrow{\Rightarrow} \cdots \xrightarrow{\Rightarrow} [u] \xleftarrow{\Rightarrow} \cdots \xleftarrow{\Rightarrow} [t]$$

The system is terminating, if no infinite sequences of rewrites with rules in $R_{\Rightarrow}$ (or $R_{\Leftarrow}$) can be built. Termination is guaranteed when the rewrite orderings defined by $R_{\Rightarrow}$ and $R_{\Leftarrow}$ respectively are contained in a unique reduction ordering on terms.

A decision algorithm for the word problem in convergent bi-rewrite systems is then straightforward: To check if $\mathcal{R} \vdash_{RWL} [s] \Rightarrow [t]$ we reduce $[s]$ and $[t]$ applying rewrite rules of each rewrite system, exploring all possible paths, until a common equivalent class of terms is reached:




The conditions put on the rewrite relations in order to guarantee termination also avoid the possibility of infinite branching.

Finite convergent bi-rewrite systems encode the reflexive, transitive and monotone closure of rewrite relation '⇒': All possible consequences of a rewrite theory $\mathcal{R}$ using the deduction rules of rewriting logic can be represented by a bi-rewrite proof.

An arbitrary bi-rewrite system, obtained by orienting the sentences of a rewrite theory $\mathcal{R}$ is non-convergent in general. But, like in the equational case,

[4] To be rigorous we only need quasi-termination [25], but for the sake of simplicity, in this case I require termination.

there exist necessary and sufficient conditions for a terminating bi-rewrite system to be Church-Rosser, which were stated by Levy and Agustí adapting the original results of Knuth and Bendix [21]. First of all we give two definitions and then the theorem which summarizes this result [5]:

**Definition 3.3** *Given a bi-rewrite system* $\langle R_{\Rightarrow}, R_{\Leftarrow} \rangle$ *and two rules* $l_1 \stackrel{\Rightarrow}{\longrightarrow} r_1 \in R_{\Rightarrow}$, $l_2 \stackrel{\Leftarrow}{\longrightarrow} r_2 \in R_{\Leftarrow}$ *(or vice versa), and a non-variable subterm* $l_2|_p$, *if* $\sigma$ *is a most general unifier of* $l_1$ *and* $l_2|_p$, *then* $\langle l_2[r_1]_p\sigma, r_2\sigma \rangle$ *is called a* critical pair.

**Definition 3.4** *Given a bi-rewrite system* $\langle R_{\Rightarrow}, R_{\Leftarrow} \rangle$ *and a rule* $l_1 \stackrel{\Rightarrow}{\longrightarrow} r_1 \in R_{\Rightarrow}$ *and an instance* $l_2\sigma \stackrel{\Leftarrow}{\longrightarrow} r_2\sigma$ *of a rewrite rule* $l_2 \stackrel{\Leftarrow}{\longrightarrow} r_2 \in R_{\Leftarrow}$ *(or vice versa), where* $\sigma$ *is such that, for some term* $v$ *with subterm* $v|_q = l_1$, *and some variable* $x$ *at position* $p$ *that appears more than once in* $l_2$, $x\sigma = v$ *and* $y\sigma = y$, *whenever* $y \neq x$, *then the critical pair* $\langle l_2[v[r_1]_q]_p\sigma, r_2\sigma \rangle$ *is called a* variable instance pair [6].

A critical or variable instance pair is said to be *convergent* if it has a bi-rewrite proof, and *divergent* otherwise.

**Theorem 3.5 (Levy and Agustí [25])** *A terminating bi-rewrite system* $\langle R_{\Rightarrow}, R_{\Leftarrow} \rangle$ *is Church-Rosser (and thus, convergent) if and only if there are no divergent critical or variable instance pairs between the rules of* $R_{\Rightarrow}$ *and the rules of* $R_{\Leftarrow}$.

Following the same ideas proposed by Knuth and Bendix, one can attempt to complete a non-convergent terminating bi-rewrite system, by means of adding divergent critical and variable instance pairs as new rewrite rules to the systems $R_{\Rightarrow}$ or $R_{\Leftarrow}$. Notice that the number of critical pairs among rewrite rules of sets $R_{\Rightarrow}$ and $R_{\Leftarrow}$ is always finite. But from the definition of variable instance pairs, we can observe that the overlap of term $l_1$ on $l_2$ is done *below* a variable position of $l_2$, and therefore unification always succeeds. Furthermore, term $v$ is arbitrary, which means that if a variable instance pair exists between two rewrite rules then there are an infinite number of them. As we will see later, this is one of the major drawbacks for the tractability of the generalization of rewrite techniques to arbitrary transitive relations, because a completion procedure which attempts to add variable instance pairs as new rewrite rules is impossible to manage in general.

We know from the completion of equational theories, that the process may fail to orient a critical pair with the given reduction ordering. There have been various variants of completion to overcome this situation [22,39,18], which have been also generalized to bi-rewrite systems [26].

[5] For the sake of simplicity I present Levy and Agustí's results for the case where no structural axioms are considered, i.e. $A = \emptyset$.

[6] Variable instance pairs also appear in the context of rewriting modulo a congruence [2].

### *3.2 Ordered chaining*

During the last decade and the beginning of the present it has been shown that the process of completion of rewrite systems can be seen as a process of refutation in the context of resolution-based theorem proving [15,3]. The principle of refutation by means of resolution is the core of the operational semantics of the logic programming paradigm [27].

Completion as a refutation process was later generalized for full first-order theories with equality [14,5] and has been further improved [35,7]. This generalization is also applicable to completion of bi-rewrite systems, and consequently we can prove theorems of a theory in rewriting logic applying a process of refutation captured by the *ordered chaining calculus* of Bachmair and Ganzinger [6]. It is based on the ordered chaining inference rule between two clauses and in essence generalizes the critical pair and variable instance pair computation during completion of bi-rewrite systems. The inference rule is stated as follows:

$$\textbf{Ordered Chaining:}\quad \frac{C \vee s \Rightarrow t \quad D \vee u \Rightarrow v}{C\sigma \vee D\sigma \vee u[s]_p\sigma \Rightarrow v\sigma}$$

where $\sigma$ is a most general unifier of $t$ and $u|_p$, $p$ being a subterm position in $u$, and the following ordering restrictions between terms, and literals hold: $s\sigma \not\succeq t\sigma$, $v\sigma \not\succeq u\sigma$, $s\sigma \Rightarrow t\sigma$ is the strictly maximal literal with respect to the remaining disjunction $C\sigma$ of the first clause, and $u\sigma \Rightarrow v\sigma$ is the strictly maximal literal with respect to the remaining disjunction $D\sigma$ of the second clause. In this context, as in the equational case, the process of completion, is known as *saturation*.

The complete calculus for full first-order clauses with transitive relations is formed of the ordered chaining inference rule together with several other inference rules —negative chaining, ordered resolution, ordered factoring and transitivity resolution—, which also put ordering restriction on the terms and atoms participating in the inference, in order to prune the search space to be explored (see [6] for further details). Bachmair and Ganzinger proved the refutational completeness of the calculus by means of their 'model construction method': Given a *saturated set* [7] of clauses they inductively construct —over an ordering on clauses— a Herbrand interpretation which is the minimal model of the saturated set. This model is then a preordered set. They also gave an intuitive notion of *redundant* clauses and inferences within the context of this model construction method. This notion is very important, since in analogy to a completion procedure, which attempts to produce a convergent bi-rewrite system in which all critical pairs and variable instance pairs are convergent (have a bi-rewrite proof), the saturation process attempts to provide us a set of clauses in which all inferences are redundant. We say in this case that the set of clauses is *saturated*, i.e. closed up to redundancy. Notice that this is the criterion in order to finish the process of completion, or saturation respectively. In the same manner as during the completion process rewrite rules are kept as interreduced as possible, during saturation redundant

[7] I give the meaning of saturated set below.

clauses are deleted, and redundant inferences avoided, by means of so called *redundancy provers*. Unfortunately, unlike the equational case, there is a lack of powerful redundancy proving techniques that can be used within a theorem prover dealing with arbitrary transitive relations.

### *3.3 Drawbacks of the general ordered chaining calculus*

We have seen that calculi based on bi-rewriting, like ordered chaining, are suitable as proof calculi for rewriting logic, since ordering restriction on terms and atoms significantly prune the search space of the prover. But these calculi are still highly prolific in the general case [42]. Inferences require unification on variable positions, although only when they appear repeated in the same term (see Definition 3.4), and, if the operators are monotonic with respect to the transitive relation (e.g. the rewrite relation '$\Rightarrow$' in rewrite theories) functional reflexive axioms are explicitly needed, in order to make variable instance pairs convergent. On the other hand, no rewriting within equivalence classes of terms is done, making a notion of unique normal form, on which equational term rewriting is based, meaningless. Consequently the order of application of rewrite rules is now significant, making term rewriting shift from *don't care* nondeterminism to *don't know* nondeterminism: Backtracking is needed for a rewrite proof to be found [8]. But by restricting these calculi to special theories, or by limiting the kind of axioms we use, it is possible to provide rewriting logic with interesting subcalculi. It is known, e.g. that in dense total orderings without endpoints, variable chaining can be avoided completely [4]. Furthermore completion of the inclusional theory of lattices to a finite and convergent bi-rewrite system is possible [24] (though no finite term rewrite system for the equational theory of lattices exists [11]) and this fact suggests to consider the properties of specific algebraic structures for improving deduction in rewriting logic.

## 4 A Framework for Proof Calculi

In this section I present the idea that a proof calculus based on the bi-rewriting technique may serve as framework of different proof calculi. I will sketch this on two very intuitive and well-known logics, following Martí-Oliet and Meseguer's approach in [28], mapping them to rewriting logic.

I am going to present the proof calculi of equational logic and Horn clause logic, from the perspective of bi-rewriting. This may appear strange or even absurd in a first sight, but my purpose is to show that these operational semantics are in fact specific cases of bi-rewrite system, and that their special nature restrict significantly the general proof calculus based on bi-rewriting. Furthermore, these restrictions act upon the drawbacks I just mentioned in Section 3.3.

[8] In spite of these general drawbacks, there exists an implementation in Prolog of a theorem prover based on ordered chaining, done by Nivela, Nieuwenhuis and Ganzinger, called *Saturate* [36,12], for which currently better implementation techniques are studied [34].

*4.1 Bi-rewriting equational logic*

An equational theory $\mathcal{E}$ can be described as a triple $(F, A, E)$, where $(F, A)$ is a signature consisting of a set $F$ of function symbols and a set $A$ of structural axioms ($F$-equations), and $E$ is a set of equations of the form $[s]_A = [t]_A$ between equivalence classes of terms. Note that if $A$ is the empty set, the equations in $E$ are between terms.

An equational theory $\mathcal{E} = (F, A, E)$ is mapped to a rewrite theory $\mathcal{R} = (F, A, R)$, such that for every equation $[s] = [t]$ in $E$, two rules $[s] \Rightarrow [t]$ and $[t] \Rightarrow [s]$ are in $R$, in order to make explicit the property of symmetry. The bi-rewrite system $\langle R_{\Rightarrow}, R_{\Leftarrow} \rangle$ resulting from orienting the rules of $R$ has for every rule $[s] \xrightarrow{\Rightarrow} [t]$ in $R_{\Rightarrow}$ also a rule $[s] \xrightarrow{\Leftarrow} [t]$ in $R_{\Leftarrow}$, i.e. each former equation appears as a rewrite rule in both rewrite systems.

**Example 4.1** *Let's consider the map of equational theory* $\mathcal{E} = (\{+, s, 0\}, \emptyset, E)$ *—which specifies the non-associative/commutative sum operator— into rewrite theory* $\mathcal{R} = (\{+, s, 0\}, \emptyset, R)$ *given below:*

$$E = \begin{cases} x + 0 = x \\ x + s(y) = s(x + y) \end{cases} \quad \longmapsto \quad R = \begin{cases} x + 0 & \Rightarrow x \\ x & \Rightarrow x + 0 \\ x + s(y) & \Rightarrow s(x + y) \\ s(x + y) & \Rightarrow x + s(y) \end{cases}$$

*Orienting the rules in* $R$*, following e.g. a lexicographic path ordering based on the signature precedence* $+ \succ s \succ 0$*, we get the following bi-rewrite system:*

$$R_{\Rightarrow} = \begin{cases} x + 0 & \xrightarrow{\Rightarrow} x \\ x + s(y) & \xrightarrow{\Rightarrow} s(x + y) \end{cases} \qquad R_{\Leftarrow} = \begin{cases} x + 0 & \xrightarrow{\Leftarrow} x \\ x + s(y) & \xrightarrow{\Leftarrow} s(x + y) \end{cases}$$

Due to symmetry, we actually are duplicating each rewrite rule. Note that since generation of critical pairs is done by looking for overlaps between left-hand sides of two rules, one of each rewrite system, in this case this is equivalent to look for overlaps among the rules of one unique rewrite system, i.e. rules that actually rewrite on equations. When dealing with equational theories, bi-rewrite systems can be 'simplified' to standard rewrite systems, as we are familiar, as for instance the following equational term rewrite system for the equational theory of Example 4.1:

$$R' = \begin{cases} x + 0 & \xrightarrow{=} x \\ x + s(y) & \xrightarrow{=} s(x + y) \end{cases}$$

Such rewrite systems correspond e.g. to the semantics of Maude's functional modules [32].

Overlaps on variable positions and the functional reflexive axioms are not needed: All those overlaps are convergent, because rewrite rules appear in both rewrite systems (see [42]). If the set of equations $E$ is Church-Rosser (in the 'traditional' sense of equational rewrite systems, for instance see [10]), the bi-rewrite system $\langle R_{\Rightarrow}, R_{\Leftarrow} \rangle$ obtained from set of rules $R$ in which $E$ is

mapped to is also Church-Rosser (in the sense of Theorem 3.5), as well as each of both rewrite systems $R_{\Rightarrow}$ and $R_{\Leftarrow}$ (again in the equational sense).

In the case $A$ is not empty, rewriting must be done modulo the set of axioms in $A$. As mentioned in Section 3.1 this has been thoroughly studied by the rewriting community, and their results can be applied also to bi-rewrite systems. This suggests that Patrick Viry's notion of *coherence completion* [45] for the implementation of rewriting in rewriting logic by using standard rewriting instead of rewriting modulo, should be also applicable to bi-rewrite systems.

Symmetry plays an important role, because when reasoning with equivalence relations, we can deal with the notion of *equivalence class*. Since we do not have two different rewrite systems any more, critical pairs are computed by overlapping left-hand sides of rules of one unique rewrite system. If such rewrite system is convergent this has important practical consequences: Each term not only has an irreducible term, the so called *normal form*, but this normal form is also unique for each term. Rewriting is done within an equivalence class, and all the members of this class share the same normal form. A decision procedure for the word problem in equational theories, based on convergent rewriting systems, is much simpler than in arbitrary rewrite theories. Just the normal forms of the two terms of the equation we want to validate are computed and checked for identity. Furthermore the property of *don't care* nondeterminism of theorem proving in convergent equational theories is kept.

### *4.2 Bi-rewriting Horn logic*

A Horn theory $\mathcal{H}$ can be described as a 4-tuple $(F, P, A, H)$. The triple $(F, P, A)$ is the signature, consisting of a set $F$ of function symbols, a set $P$ of predicate symbols, and a set $A$ of structural axioms (i.e. $F$-equations). $H$ is a set of Horn clauses of the form $[s]_A \longleftarrow [t_1]_A, \ldots, [t_n]_A$. A Horn theory $\mathcal{H} = (F, P, A, H)$ is mapped to a two-sorted rewrite theory $\mathcal{R} = (F \cup P', A \cup A', R)$ with sorts **term** and **prop**. All functions symbols in $F$ take arguments of sort **term** and are themselves of sort **term**, and set $P'$ contains a constant *true* of sort **prop**, a binary infix operator '$\wedge$' of sort **prop** taking as argument two elements of sort **prop**, and for each n-ary predicate $p$ in $P$, an n-ary function symbol $p$ of sort **prop** taking as arguments $n$ elements of sort **term**. $A'$ is the set containing the associativity, commutativity and identity law (with respect to constant *true*) of operator '$\wedge$', and $R$ is the set of rules obtained by mapping each clause $[s]_A \longleftarrow [t_1]_A, \ldots, [t_n]_A$ to the rule $[s]_{A \cup A'} \Rightarrow [t_1 \wedge \cdots \wedge t_n]_{A \cup A'}$, and each unit clause $[s]_A$ to the rule $[s]_{A \cup A'} \Rightarrow [true]_{A \cup A'}$.

**Example 4.2** *Horn theory* $\mathcal{H} = (\{ann, bob, tom\}, \{par, anc\}, \emptyset, H)$ *—which specifies the parent (par) and ancestor (anc) relation— is mapped to rewrite theory* $\mathcal{R} = (\{ann, bob, tom, par, anc, true, \wedge\}, A', R)$ *as follows,* $A'$ *being the set defined above:*

$$H = \begin{cases} par(ann, bob) \\ par(bob, tom) \\ anc(x,y) \leftarrow par(x,y) \\ anc(x,y) \leftarrow par(x,z), anc(z,y) \end{cases} \mapsto R = \begin{cases} par(ann, bob) \Rightarrow true \\ par(bob, tom) \Rightarrow true \\ anc(x,y) \Rightarrow par(x,y) \\ anc(x,y) \Rightarrow par(x,z) \wedge anc(z,y) \end{cases}$$

*4.2.1 SLD-resolution is not bi-rewriting*

It is well-known that a proof calculus based on the resolution inference is efficient as operational semantics for Horn logic programming: Queries to a program are existentially quantified formulas $\exists \bar{x}\ u_1, \ldots, u_m$ [9], and are solved by refuting its negation. A resolution step is then as follows [10]:

$$\frac{\longleftarrow u_1, u_2, \ldots, u_m \qquad s \longleftarrow t_1, \ldots, t_n}{\longleftarrow t_1\sigma, \ldots, t_n\sigma, u_2\sigma, \ldots, u_m\sigma}$$

where $\sigma$ is a most general unifier of $u_1$ and $s$.

A query in its correspondent rewrite theory reads then $\exists \bar{x}\ [u_1 \wedge u_2 \wedge \cdots \wedge u_m]_{A'} \Rightarrow [true]_{A'}$, which is solved also by refuting its negation. The inference step which corresponds to the above resolution step reads:

$$\frac{[u_1 \wedge u_2 \wedge \cdots \wedge u_m] \not\Rightarrow [true] \qquad [s] \Rightarrow [t_1 \wedge \cdots \wedge t_n]}{[t_1\sigma \wedge \cdots \wedge t_n\sigma \wedge u_2\sigma \wedge \cdots u_m\sigma] \not\Rightarrow [true]} \quad (1)$$

where $\sigma$ is, as before, a most general unifier of $u_1$ and $s$. This inference step is actually a *negative chaining* (see [6]).

Since chaining is only done through the term on the left-hand side of the rule representing the negated query, until a term in the $A'$-equivalence class of $true$ is reached, we can see this inference also as applying rule $[s] \Rightarrow [t_1 \wedge \cdots \wedge t_n]$ in order to *narrow* [11] the 'query term' $[u_1 \wedge u_2 \wedge \cdots \wedge u_m]$:

$$[u_1 \wedge u_2 \wedge \cdots \wedge u_m] \rightsquigarrow [t_1\sigma \wedge \cdots \wedge t_n\sigma \wedge u_2\sigma \wedge \cdots u_m\sigma]$$

Here $\sigma$ is, again, a most general unifier of $u_1$ and $s$. This is the approach followed by C. Kirchner, H. Kirchner and Vittek in [19], who also studied the map of proofs in Horn theories to proofs in rewrite theories. They map Horn clauses to narrowing rules, and the proof-theoretic structure of Horn logic, based on SLD-resolution, is therefore captured by the straightforward application of the deduction rules of rewriting logic. They further add to the rewrite theory a notion of strategy to efficiently compute with the given rewrite rules and call such a rewrite theory plus strategy a *computational system*.

Negative chaining —Inference 1 above— is ordered if rules $[s] \Rightarrow [t_1 \wedge \cdots \wedge t_n]$ of rewrite theory $\mathcal{R}$ are oriented from left to right, i.e. $[s] \succ [t_1 \wedge \cdots \wedge t_n]$. Indeed, the operational behavior of query solving in Horn theories following resolution strategies known from logic programming, like Prolog's SLD-resolution, is captured by the trivial 'bi-rewrite' system $(R_{\Rightarrow}, \emptyset)$, where $\Rightarrow \subseteq \xrightarrow{\Rightarrow}$. This 'bi-

[9] $\bar{x}$ denotes the free variables of terms $u_1, \ldots, u_m$.

[10] For the sake of simplicity this inference is shown for Horn theories with no structural axioms, i.e. $A = \emptyset$.

[11] Narrowing was originally devised as an efficient E-unification procedure using convergent sets of rewrite rules [16].

rewrite' system is actually a standard rewrite system since we are not rewriting in two directions, and its operational behavior corresponds to standard deduction in rewriting logic. But, as said in Section 3 the ordering induced by these rules will not be in general a reduction ordering, and therefore this 'bi-rewrite' system will in general be non-terminating.

*4.2.2 Ordered chaining for Horn theories*

When taking a reduction ordering on terms into account, the process of theorem proving in Horn logic maps to an ordered chaining inference tree. I will show this through an example.

**Example 4.3** *If we orient the rules of the rewrite theory obtained in Example 4.2 following e.g. a lexicographic path ordering based on the signature precedence* $\wedge \succ anc \succ par \succ tom \succ bob \succ ann \succ true$*, we get the following bi-rewrite system:*

$$R_{\Rightarrow} = \begin{cases} par(ann, bob) \xrightarrow{\Rightarrow} true \\ par(bob, tom) \xrightarrow{\Rightarrow} true \\ anc(x, y) \xrightarrow{\Rightarrow} par(x, y) \end{cases}$$

$$R_{\Leftarrow} = \left\{ par(x, z) \wedge anc(z, y) \xrightarrow{\Leftarrow} anc(x, y) \right.$$

As said in Section 3.1, by orienting the rules of a rewrite theory by means of a reduction ordering on terms, critical pairs (or even variable instance pairs) among the rules of both rewrite systems can arise: We need to start a process of completion for proving theorems, by generating new rules, i.e. our proof calculus will be based on ordered chaining (see Section 3.2). The interesting point is that, since the unique operator of the signature which is monotonic with respect to the relation '$\Rightarrow$' is the the conjunction operator '$\wedge$', the overlap required for generating new rules is only needed on whole propositions and not on terms within them. Furthermore, since the map of Horn to rewrite theories does not introduce variables as arguments of '$\wedge$', unification on variable positions is not needed, and the intractable variable instance pair generation can be completely avoided (and therefore functional reflexive axioms are superfluous).

Figure 2 shows the ordered chaining inference tree for proving theorem $anc(ann, tom) \Rightarrow true$ in rewrite theory $\mathcal{R}$ of Example 4.2. The leaf with the framed sentence is the negation of the theorem. All other leafs are sentences of the rewrite theory. Inference steps are labeled with *(OC)* if it is a ordered chaining step, with *(NC)* if it is a negative chaining step and with *(OR)* if it is a ordered resolution step (see [6] for further details). Bold faced terms are the ones who are unified (i.e. chained through). For instance the top most inference step of Figure 2 corresponds to the generation of a critical pair among rewrite rules $par(x, z) \wedge anc(z, y) \xrightarrow{\Leftarrow} anc(x, y)$ and $anc(x', y') \xrightarrow{\Rightarrow} par(x', y')$.

$$\dfrac{\dfrac{\dfrac{anc(x,y) \Rightarrow par(x,z) \wedge \mathbf{anc(z,y)} \qquad \mathbf{anc(x',y')} \Rightarrow par(x',y')}{anc(x,y) \Rightarrow \mathbf{par(x,z)} \wedge par(z,y) \qquad \mathbf{par(ann,bob)} \Rightarrow true}\,(\text{OC})}{\boxed{\mathbf{anc(ann,tom)} \not\Rightarrow true} \qquad \begin{array}{c} anc(ann,y) \Rightarrow true \wedge par(bob,y) \\ \| \| \; A' \\ \mathbf{anc(ann,y)} \Rightarrow par(bob,y) \end{array}}\,(\text{OC})}{par(bob,tom) \not\Rightarrow true \qquad par(bob,tom) \Rightarrow true}\,(\text{NC})$$

$$\square \qquad (\text{OR})$$

Fig. 2. Ordered chaining inference tree

Unfortunately, as we can observe from Figure 2, the linear strategy of resolution in Horn theories must be —for completeness— abandoned, since the generation of rules from critical pairs (i.e. ordered chaining inference steps) correspond to resolution among clauses of the given theory. But the advantages of the use of term ordering arise, when it is possible to saturate (i.e. to complete) a bi-rewrite system obtained from the previously explained map: The search for proofs by SLD-resolution (or straightforward deduction in rewriting logic, see Section 4.2.1), which could have been non-terminating, is now 'replaced' by terminating bi-rewriting (because of the reduction ordering on terms).

**Example 4.4** *Given the following set of Horn clauses:*

$$q(x) \longleftarrow p(x)$$
$$p(x) \longleftarrow q(x)$$

*and the (negated) query:*

$$\longleftarrow q(a)$$

*Though it is evident that we cannot refute it, the process of applying SLD-resolution will never terminate. Instead, given a signature precedence $q \succ p$, the rewrite theory to which this Horn theory is mapped, forms a convergent bi-rewrite system:*

$$R_{\Rightarrow} = \{q(x) \xrightarrow{\Rightarrow} p(x)\}$$
$$R_{\Leftarrow} = \{p(x) \xrightarrow{\Leftarrow} q(x)\}$$

*Now we can proof, in a finite amount of time, that $q(a) \not\Rightarrow true$, because $q(a) \xrightarrow{\Rightarrow} p(a)$ is the only rewrite step that can be performed.*

Further work I want to do in this direction is to study the results about termination of Horn clause programs from this point of view, and to reformu-

late the conditions of termination as restrictions on proof calculi of rewriting logic.

## 5 Towards a Framework for the Operational Semantics of Logic Programs

Martí-Oliet and Meseguer conjecture in [29], that rewriting logic can be useful as logical framework, at least for those logics we can consider of 'practical interest', and whose proof calculi correspond to the operational semantics of programming languages based on these logics. In this paper I have made a first step towards the study of specific restrictions on bi-rewriting based calculi by analizing mappings between proof calculi, which I think will be useful for defining a general notion of operational semantics: Research in this direction will be promising.

Furthermore, recently the interest in specifications based on logics with transitive relations has arisen. Mosses introduced *unified algebras* [33], a framework for the algebraic specification of abstract data types, where sorts are treated as values, so that operations may be applied to sorts as well as to the elements that they classify. This framework is based on a partial order of a distributive lattice with a bottom. Similar intuitions were followed by Levy and Agustí, who proposed the *Calculus of Refinements* [24], a formal specification model based on inclusions. Their approach showed to be useful for the preliminary specification and further stepwise refinement of complex systems [41]. Rewriting logic itself and its embodiment in Maude has served as prototyping language for the specification of complex systems [23]. Therefore the result of this research towards the design of a multi-paradigm programming language dealing with arbitrary transitive relations may also be very useful for developing rapid prototyping tools for these kind of specifications [43].

Besides these general specification frameworks, partial orders also play a central role in a variety of much more concrete logic programming languages. For example, Aït-Kaci and Podelski make use of order-sorted feature terms as basic data structure of the programming language LIFE [1], generalizing in this way the flat first-order terms normally used as unique data structure in logic programming. An order-sorted feature term is a compact way to represent the collection of elements of a given non-empty domain which satisfy the constraint encoded by the term, and therefore may be interpreted itself as a sort, like in 'unified algebras' or in the 'Calculus of Refinements', being LIFE one of the first proposals of sorts as values. Algebraically, a term denotes an element of a meet semi-lattice with a top $\top$ and a bottom $\bot$, which in essence is a subalgebra of the power set of the considered domain. But, deduction in LIFE is quite poor, because of the restricted use of terms within the definition of the partial order. Deduction reduces to unification of order-sorted feature terms and can be seen as the meet operation in the semi-lattice. It is performed by normalizing the conjunction of the constraints encoded in the terms to be unified, and is equivalent to intersecting the collections of elements the terms represent.

Also Jayaraman, Osorio and Moon base their *partial order programming* paradigm on a lattice structure, and are specially interested on the complete lattice of finite sets [17]. In their paradigm they pursue the aim to integrate sets into logic programming, and to consider them as basic data structure on which the paradigm relies. But in this framework no deduction mechanisms are given to validate order related functional expressions.

To summarize, in a future work it is necessary to analize proof calculi and theorem proving strategies of different interesting logics, and to study the map of their proof calculi to bi-rewriting. This will clarify how efficiency issues and strategies of these calculi are captured by restrictions on general calculi based on bi-rewriting, so that a sufficiently general proof calculus of rewriting logic based on bi-rewriting and ordered chaining can be stated, which may serve as general framework for the operational semantics of interesting logic programming and specification paradigms I have just mentioned. The knowledge about these restrictions translated to efficiency aspects of proof calculi will help to find an optimal balance between generality and efficiency.

## Acknowledgement

I am specially grateful to Jaume Agustí for his valuable comments and helpful suggestions on previous versions of this paper.

## References

[1] H. Aït-Kaci and A. Podelski. Towards a meaning of LIFE. *Journal of Logic Programming*, 16:195–234, 1993.

[2] L. Bachmair, N. Dershowitz, and J. Hsiang. Orderings for equational proofs. In *Symposium of Logic in Computer Science*, pages 346–357, 1986.

[3] L. Bachmair, N. Dershowitz, and D. A. Plaisted. Completion without failure. In *Resolution of Equations in Algebraic Structures*, volume 2. Academic Press, 1989.

[4] L. Bachmair and H. Ganzinger. Ordered chaining for total orderings. In A. Bundy, editor, *Automated Deduction — CADE'12*, volume 814 of *LNAI*, pages 435–450. Springer-Verlag, 1994.

[5] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):1–31, 1994.

[6] L. Bachmair and H. Ganzinger. Rewrite techniques for transitive relations. In *Proc., Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 384–393, 1994.

[7] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and superposition. In D. Kapur, editor, *Automated Deduction — CADE-11*, volume 607 of *LNAI*, pages 462–476. Springer-Verlag, 1992.

[8] G. Birkhoff. On the structure of abstract algebras. *Proc. Cambridge Philos. Soc.*, 31:433–454, 1935.

[9] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[10] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B. Elsevier Science Publishers, 1990.

[11] R. Freese, J. Ježek, and J. Nation. Term rewrite systems for lattice theory. *Journal of Symbolic Computation*, 16:279–288, 1993.

[12] H. Ganzinger, R. Nieuwenhuis, and P. Nivela. The Saturate system. `http://www.mpi-sb.mpg.de/SATURATE/Saturate.html`, 1995.

[13] J. C. González-Moreno, T. Hortalá-González, F. López-Fraguas, and M. Rodríguez-Artalejo. A rewriting logic for declarative programming. In H. R. Nielson, editor, *Programming Languages and Systems — ESOP '96*, LNCS 1058. Springer-Verlag, 1996.

[14] J. Hsiang and M. Rusinowitch. Proving refutational completeness of theorem proving strategies: The transfinite semantic tree method. *Journal of the ACM*, 38(3):559–587, 1991.

[15] G. Huet. A complete proof of correctness of the Knuth-Bendix completion algorithm. *Journal of Computation and System Sciences*, 23:11–21, 1981.

[16] J. M. Hullot. Canonical forms and unification. In *Proc. 4th International Conference on Automated Deduction*, LNCS 87, 1980.

[17] B. Jayaraman, M. Osorio, and K. Moon. Partial order programming (revisited). In *Proc. Algebraic Methodology and Software Technology (AMAST)*, pages 561–575, 1995.

[18] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15:1155–1194, 1986.

[19] C. Kirchner, H. Kirchner, and M. Vittek. Designing constraint logic programming languages using computational systems. In P. van Hentenryck and S. Saraswat, editors, *Principles and Practice of Constraint Programming.* MIT Press, 1995.

[20] J. W. Klop. Term rewriting systems. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, 1992.

[21] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[22] D. S. Lankford and A. Ballantyne. Decision procedures for simple equational theories with permutative axioms: Complete sets of permutative reductions. Technical Report ATP-37, Department of Mathematics and Computer Science, University of Texas, 1977.

[23] U. Lechner, C. Lengauer, and M. Wirsing. An object-oriented airport: Specification and refinement in Maude. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Types Specification*, LNCS 906. Springer Verlag, 1995.

[24] J. Levy. *The Calculus of Refinements: a Formal Specification Model Based on Inclusions.* PhD thesis, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 1994.

[25] J. Levy and J. Agustí. Bi-rewriting, a term rewriting technique for monotonic order relations. In C. Kirchner, editor, *Rewriting Techniques and Applications*, LNCS 690, pages 17–31. Springer-Verlag, 1993.

[26] J. Levy and J. Agustí. Bi-rewrite systems. *Journal of Symbolic Computation*, 1996. To be published.

[27] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1984.

[28] N. Martí-Oliet and J. Meseguer. Rewriting logic as logical and semantic framework. Technical Report SRI-CSL-93-05, Computer Science Laboratory, SRI International, August 1993.

[29] N. Martí-Oliet and J. Meseguer. General logics and logical frameworks. In D. M. Gabbay, editor, *What is a Logical System?*, pages 355–391. Clarendon Press, 1994.

[30] J. Meseguer. General logics. In H. D. Ebbinghaus et al., editors, *Logic Colloquium '87*, pages 275–329. Elsevier Science Publishers, 1989.

[31] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Journal of Theoretical Computer Science*, 96:73–155, 1992.

[32] J. Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In G. Agha et al., editors, *Research Directions in Concurrent Object-Oriented Programming*, pages 315–390. MIT Press, 1993.

[33] P. Mosses. Unified algebras and institutions. In *Principles of Programming Languages Conference*, pages 304–312. ACM Press, 1989.

[34] R. Nieuwenhuis, J. M. Rivero, and M. A. Vallejo. An implementation kernel for theorem proving with equality clauses. In *Proc. of the 1996 Joint Conference on Declarative Programming APPIA-GULP-PRODE'96*, pages 89–103, July 1996.

[35] R. Nieuwenhuis and A. Rubio. Basic superposition is complete. In *European Symposium on Programming*, 1992.

[36] P. Nivela and R. Nieuwenhuis. Saturation of first-order (constrained) clauses with the Saturate system. In C. Kirchner, editor, *Rewriting Techniques and Applications*, LNCS 690, pages 436–440. Springer-Verlag, 1993.

[37] D. S. Parker. Partial order programming. Unpublished monograph, 1987.

[38] D. S. Parker. Partial order programming. In *POPL'89: 16th ACM Symposium on Principles of Programming Languages*, pages 260–266. ACM Press, 1989.

[39] G. E. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, 1981.

[40] D. A. Plaisted. Equational reasoning and term rewriting systems. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1, pages 273–364. Oxford University Press, 1993.

[41] D. Robertson, J. Agustí, J. Hesketh, and J. Levy. Expressing program requirements using refinement lattices. *Fundamenta Informaticae*, 21:163–183, 1994.

[42] W. M. Schorlemmer and J. Agustí. Theorem proving with transitive relations from a practical point of view. Research Report IIIA 95/12, Institut d'Investigació en Intel·ligència Artificial (CSIC), May 1995.

[43] W. M. Schorlemmer and J. Agustí. Inclusional theories in declarative programming. In *Proc. of the 1996 Joint Conference on Declarative Programming APPIA-GULP-PRODE'96*, pages 167–178, July 1996.

[44] J. R. Slagle. Automated theorem proving for theories with built-in theories including equality, partial orderings and sets. *Journal of the ACM*, 19:120–135, 1972.

[45] P. Viry. Rewriting: An effective model of concurrency. In C. Halatsis et al., editors, *PARLE '94, Proc. Sixth Int. Conf. on Parallel Architectures and Languages Europe*, LNCS 817, pages 648–660. Springer-Verlag, July 1994.