Electronic Notes in Theoretical Computer Science 4 (1997)

Reflection and Strategies in Rewriting Logic [1]

Manuel Clavel and José Meseguer

Computer Science Laboratory
SRI International
Menlo Park, CA 94025, USA
{*clavel,meseguer*}*@csl.sri.com*

Abstract

After giving general metalogical axioms characterizing reflection in general logics in terms of the notion of a *universal theory*, this paper specifies a finitely presented universal theory for rewriting logic and gives a detailed proof of the claim made in [6] that rewriting logic is reflective. The paper also gives general axioms for the notion of a strategy language *internal* to a given logic. Exploiting the fact that rewriting logic is reflexive, a general method for defining internal strategy languages for it and proving their correctness is proposed and is illustrated with an example. The Maude language has been used as an experimental vehicle for the exploration of these techniques. They seem quite promising for applications such as metaprogramming and module composition, logical framework representations, development of formal programming and proving environments, supercompilation, and formal verification of strategies.

1 Introduction

Reflection is a very desirable property of computational systems, because a reflective system can access its own metalevel and can in this way be much more powerful, flexible, and adaptable than a nonreflective one. Many researchers have recognized the great importance and usefulness of reflection in programming languages [29,28,31,30,13,10,19], in theorem-proving [32,4,26,11,1,18,8,9], in concurrent and distributed computation [17,24,25], and in many other areas such as compilation, programming environments, operating systems, fault-tolerance, and databases (see [27,12] for recent snapshots of research in reflection).

[1] Supported by Office of Naval Research Contracts N00014-95-C-0225 and N00014-96-C-0114, National Science Foundation Grant CCR-9224005, and by the Information Technology Promotion Agency, Japan, as a part of the Industrial Scienc e and Technology Frontier Program "New Models for Software Architecture" sponsored by NEDO (New Energy and Industrial Technology Development Organization).

The goal of this paper is to prove in detail the claim made in [6], namely, that rewriting logic is reflective. We then use this fact to provide semantic foundations for internal strategy languages in rewriting logic, that is, languages that control the computations of rewrite theories but that are themselves definable within rewriting logic.

Since the field of reflection has a wealth of important examples but a dearth of general, formalism-independent, semantic foundations, the notion of "reflective logic" typically is not defined formally but is instead illustrated by example. Under such circumstances, a mathematical proof of our claim is not even a meaningful concept. Therefore, we must first make precise what we mean by a "reflective logic," as an essential prerequisite to the statement and proof of our claim. For this purpose we summarize in Section 2 the metalogical axioms of reflection based on the theory of general logics that we proposed in [6].

The key idea is that a class $\mathcal{C}$ of theories in a logic is reflective if we can find inside the class a *universal theory* U that can simulate all other theories in the class in the sense that there is a function, called a *representation function*,

$$\overline{(\_ \vdash \_)} : \bigcup_{T \in \mathcal{C}} \{T\} \times sen(T) \longrightarrow sen(U)$$

such that for each $T \in \mathcal{C},\ \varphi \in sen(T)$,

$$T \vdash \varphi \iff U \vdash \overline{T \vdash \varphi}.$$

For rewriting logic the class $\mathcal{C}$ of interest is that of all finitely presentable theories, that is, theories with a finite signature and a finite set of axioms. We construct a finitely presentable rewrite theory U and prove the above equivalence.

The correctness of U can then be used to provide semantic foundations for another topic of great importance in rewriting logic, namely *strategies* that can be used to control the rewriting process. Since the rules in rewrite theory need not be Church-Rosser and may not terminate, the need for controlling the rewrites is much stronger than for functional programs. The great opportunity offered by reflection is to make such strategies *internal* to rewriting logic. This means that strategies can themselves be defined by rewrite rules, and can be reasoned about inside the logic.

Again, to make this concept precise not just for rewriting logic but for a logic in general, in Section 4 we give a somewhat simpler version of the general, formalism-independent, notion of strategy already proposed in [6]. Basically, an *internal strategy language* is a theory-transforming function $\mathcal{S}$ that sends each theory T to another theory $\mathcal{S}(T)$ whose deductions simulate controlled deductions of T. In Section 5 we discuss reflection in rewriting logic and give a general method for defining internal strategy languages in a sound and extensible manner. These ideas have been applied to Maude, that is an explicitly reflective rewriting logic language [5].

In the concluding remarks we discuss several promising application areas that reflection and internal strategies open up for rewriting logic languages.

2 Reflection in General Logics

We give here a brief summary of the notion of a universal theory in a logic and of a reflective entailment system introduced in [6]. These notions axiomatize reflective logics within the theory of general logics [22]. We focus here on the simplest case, namely entailment systems. However, reflection at the proof calculus level—where not only sentences, but also proofs are reflected—is also very useful; the adequate definitions for that case are also in [6].

For our present purposes it will be the notions of syntax, of entailment system proposed in [22] that play a crucial role. We present below in summarized form the axioms characterizing these notions. The axioms use the language of category theory, but do not require any acquaintance with categories beyond the basic notions of category, functor, and natural transformation.

2.1 Syntax

Syntax can typically be given by a *signature* Σ providing a grammar on which to build *sentences*. For first order logic, a typical signature consists of a list of function symbols and a list of predicate symbols, each with a prescribed number of arguments, which are used to build up the usual sentences. It is enough to assume that for each logic there is a category **Sign** of possible signatures for it, and a functor *sen* assigning to each signature Σ the set $sen(\Sigma)$ of all its sentences. We call the pair $(\mathbf{Sign}, sen)$ a *syntax*.

2.2 Entailment systems

For a given signature Σ in **Sign**, *entailment* (also called *provability*) of a sentence $\varphi \in sen(\Sigma)$ from a set of axioms $\Gamma \subseteq sen(\Sigma)$ is a relation $\Gamma \vdash \varphi$ which holds if and only if we can prove φ from the axioms Γ using the rules of the logic. We make this relation relative to a signature.

In what follows, $|\mathcal{C}|$ denotes the collection of objects of a category $\mathcal{C}$.

Definition 2.1 [22] An *entailment system* is a triple $\mathcal{E} = (\mathbf{Sign}, sen, \vdash)$ such that

- $(\mathbf{Sign}, sen)$ is a syntax,
- $\vdash$ is a function associating to each $\Sigma \in |\mathbf{Sign}|$ a binary relation $\vdash_\Sigma \subseteq \mathcal{P}(sen(\Sigma)) \times sen(\Sigma)$ called Σ-*entailment* such that the following properties are satisfied:

(1) *reflexivity*: for any $\varphi \in sen(\Sigma)$, $\{\varphi\} \vdash_\Sigma \varphi$,
(2) *monotonicity*: if $\Gamma \vdash_\Sigma \varphi$ and $\Gamma' \supseteq \Gamma$ then $\Gamma' \vdash_\Sigma \varphi$,
(3) *transitivity*: if $\Gamma \vdash_\Sigma \varphi_i$, for all $i \in I$, and $\Gamma \cup \{\varphi_i \mid i \in I\} \vdash_\Sigma \psi$, then $\Gamma \vdash_\Sigma \psi$,
(4) $\vdash$-*translation*: if $\Gamma \vdash_\Sigma \varphi$, then for any $H : \Sigma \to \Sigma'$ in **Sign** we have $sen(H)(\Gamma) \vdash_{\Sigma'} sen(H)(\varphi)$. $\Box$

Definition 2.2 [22] Given an entailment system $\mathcal{E}$, its category **Th** of *theories* has as objects pairs $T = (\Sigma, \Gamma)$ with Σ a signature and $\Gamma \subseteq sen(\Sigma)$. A *theory* *morphism* $H : (\Sigma, \Gamma) \to (\Sigma', \Gamma')$ is a signature morphism $H : \Sigma \to \Sigma'$ such that if $\varphi \in \Gamma$, then $\Gamma' \vdash_{\Sigma'} sen(H)(\varphi)$.

A theory morphism $H : (\Sigma, \Gamma) \longrightarrow (\Sigma', \Gamma')$ is called *axiom-preserving* if it satisfies the condition that $sen(H)(\Gamma) \subseteq \Gamma'$. This defines a subcategory $\mathbf{Th}_0$ with the same objects as $\mathbf{Th}$ but with morphisms restricted to be axiom-preserving theory morphisms. $\Box$

Note that we can extend the functor *sen* to a functor on theories by taking $sen(\Sigma, \Gamma) = sen(\Sigma)$. Note that we have also a functor $thm : \mathbf{Th}_0 \longrightarrow \mathbf{Set}$ associating to each theory $T = (\Sigma, \Gamma)$ the set $thm(T) = \{\varphi \in sen(\Sigma) \mid \Gamma \vdash_\Sigma \varphi\}$ of its theorems.

2.3 Universal theories and reflective entailment systems

A reflective logic is a logic in which important aspects of its metatheory can be represented at the object level in a consistent way, so that the object-level representation correctly simulates the relevant metatheoretic aspects. Two obvious metatheoretic notions that can be so reflected are theories and the entailment relation $\vdash$. This leads us to the notion of a universal theory. However, universality may not be absolute, but only relative to a class $\mathcal{C}$ of *representable* theories. Typically, for a theory to be representable at the object level, it must have a finitary description in some way—say, being recursively enumerable—so that it can be represented as a piece of language.

Definition 2.3 Given an entailment system $\mathcal{E}$ and a set of theories $\mathcal{C} \subseteq |\mathbf{Th}|$, a theory U is $\mathcal{C}$*-universal* if there is a function, called a *representation function*,

$$\overline{(\_ \vdash \_)} : \bigcup_{T \in \mathcal{C}} \{T\} \times sen(T) \longrightarrow sen(U)$$

such that for each $T \in \mathcal{C}$, $\varphi \in sen(T)$,

$$T \vdash \varphi \iff U \vdash \overline{T \vdash \varphi}.$$

If, in addition, $U \in \mathcal{C}$, then the entailment system $\mathcal{E}$ is called $\mathcal{C}$*-reflective*. $\Box$

Note that in a reflective entailment system, since U itself is representable, representation can be iterated, so that we immediately have a "reflective tower"

$$T \vdash \varphi \iff U \vdash \overline{T \vdash \varphi} \iff U \vdash \overline{U \vdash \overline{T \vdash \varphi}} \ \ldots$$

3 Reflection in Rewriting Logic

In this section we give the rules of deduction for rewriting logic, define a universal theory for a class of finitely presentable rewrite theories, and prove the correctness of such a theory, establishing that indeed it satisfies all the formal requirements that we have stated for a reflective logic in Definition 2.3.

3.1 Rewriting logic

A *signature* in rewriting logic is a pair (Σ, E) formed by a ranked alphabet Σ of function symbols and a set E of Σ-equations. Given a signature (Σ, E), *sentences* of the logic are sequents of the form $[t]_E \longrightarrow [t']_E$, where $[t]$ and $[t']$ are E-equivalence classes of terms. A *theory* in this logic, called a rewrite theory, is a triple[2] $T = (\Sigma, E, R)$ with (Σ, E) a signature, and R a set of sequents called the *rewrite rules* of T.

Given a rewrite theory T, we say that T *entails* a sequent $[t] \longrightarrow [t']$, and write $T \vdash [t] \longrightarrow [t']$, iff $[t] \longrightarrow [t']$ can be obtained by finite application of the following rules of deduction. To indicate that $\{x_1, \ldots, x_n\}$ is the set of variables occurring in either t or t', we write $t(x_1, \ldots, x_n) \longrightarrow t'(x_1, \ldots, x_n)$; also $t(\vec{w}/\vec{x})$ denotes the simultaneous substitution of w_i for x_i in t.

(1) **Reflexivity**. For each $[t] \in T_{\Sigma,E}(X)$,

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

(2) **Congruence**. For each $f \in \Sigma_n$, $n \in \mathbb{N}$,

$$\frac{[t_1] \longrightarrow [t'_1] \quad \ldots \quad [t_n] \longrightarrow [t'_n]}{[f(t_1, \ldots, t_n)] \longrightarrow [f(t'_1, \ldots, t'_n)]}$$

(3) **Replacement**. For each rewrite rule $t(x_1, \ldots, x_n) \longrightarrow t'(x_1, \ldots, x_n)$ in T,

$$\frac{[w_1] \longrightarrow [w'_1] \quad \ldots \quad [w_n] \longrightarrow [w'_n]}{[t(\vec{w}/\vec{x})] \longrightarrow [t'(\vec{w'}/\vec{x})]}$$

(4) **Transitivity**.

$$\frac{[t_1] \longrightarrow [t_2] \qquad [t_2] \longrightarrow [t_3]}{[t_1] \longrightarrow [t_3]}$$

Lemma 3.1 *Given a theory* (Σ, E, R), t, $t' \in T_\Sigma$,

$$(\Sigma, E, R) \vdash [t]_E \longrightarrow [t']_E \iff (\Sigma, \emptyset, R \cup \overrightarrow{E} \cup \overleftarrow{E}) \vdash t \longrightarrow t',$$

where, by definition, $\overrightarrow{E} = \{t_1 \longrightarrow t_2 \mid t_1 = t_2 \in E\}$, $\overleftarrow{E} = \{t_2 \longrightarrow t_1 \mid t_1 = t_2 \in E\}$.

3.2 A universal theory for rewriting logic

In this section we introduce a theory U and a representation function $\overline{(\_ \vdash \_)}$ for encoding pairs consisting of a rewrite theory T in $\mathcal{C}$ and a sentence in it as sentences in U, for $\mathcal{C}$ the class of unconditional and unsorted finitely presentable rewrite theories—that is, theories whose ranked alphabet, and set of rules are all finite.

Without any essential loss of generality we assume that the syntax of those theories is given by operators and variables that are strings of ASCII characters. We also assume that all such theories have standard parenthesized notation. However, to ease readability, in the particular case of the theory U, we will adopt some extra notational conventions.

Moreover, for the sake of simplicity, using Lemma 3.1 we will consider the equations in a theory T in $\mathcal{C}$ as bidirectional sequents.

[2] In the standard treatment of rewriting logic, rules in a rewrite theory T have labels in a set L and are written $l : t \longrightarrow t'$. We omit labels in the present version to simplify the exposition. All that we say below has a straightforward extension to the labelled case.

We first introduce the ranked alphabet Σ_U of operation symbols of U and briefly explain how a representation function for U can be defined. Then, we give the set E_U of equations, and the set R_U of rules of U.

The operation symbols of Σ_U are as follows:

$\Sigma_0 = \mathrm{ASCII} \cup \{ \ \cdot\ ,\ \triangleright\ \}$, where ASCII denotes the set of ASCII characters.
$\Sigma_1 = \{\ \mathrm{op}\{\_\}\ ,\ \ \mathrm{var}\{\_\}\ \}$
$\Sigma_2 = \{\ \_.\_\ ,\ \_[\_]\ ,\ \_;\_\ ,\ \_[\rightarrow \_]\ ,\ \_[\hookleftarrow \_]\ ,\ \_[\hookrightarrow \_]\ ,\ \_\Rightarrow\_\ ,\ \_\mapsto\_\ ,\ \_/\_\ ,$
$\qquad \_@\_\ ,\ \_\ddot{@}\_\ ,\ \langle \_,\_ \rangle\ \}$
$\Sigma_3 = \{\ \_ :: \langle \_,\_ \rangle\ \}$
$\Sigma_5 = \{\ \_ :: \langle \_,\_,\_,\_ \rangle\ \}$

To ease readability we adopt the following notational convention: the occurrence of n underbar characters in an n-ary operator of Σ_U allows us to display the corresponding expressions with mix-fix syntax, adding parentheses when necessary. Thus, for the operator $\_ \mapsto \_$ we write $X \mapsto Y$ instead of $\_ \mapsto \_(X, Y)$. Also, we use in Σ_U some characters that are not strings of ASCII characters.

We next define a representation function $\overline{(\_ \vdash \_)}$ for encoding pairs consisting of a finitely presentable rewrite theory T and a sentence in it as sentences in U,

$$\overline{(\_ \vdash \_)} : \bigcup_{T \in \mathcal{C}} \{T\} \times \mathit{sen}(T) \longrightarrow \mathit{sen}(U)$$

The definition of $\overline{(\_ \vdash \_)}$ is given in a top-down fashion. This will make clear the intended meaning of most of the operation symbols of Σ_U. Note that, to ease readability, we recursively define the representation of theories, rules, terms, etc. using an overloaded function symbol $\overline{(\_)}$.

- For $T \in \mathcal{C}$ a finitely presentable rewrite theory, and $t \longrightarrow t'$ a sentence in T, $\overline{(T \vdash t \longrightarrow t')} = (\overline{T} @ * [\rightarrow \overline{t}]) \longrightarrow (\overline{T} @ * [\rightarrow \overline{t'}])$.
- For T a rewrite theory $(\Sigma, R) \in \mathcal{C}$ such that V is the finite set of variables appearing in the rules R, $\overline{T} = \langle \overline{V}, \overline{R} \rangle$.
- For V a set of variables $\{v_1, \ldots, v_n\}$, $\overline{V} = \overline{v_1}; \ldots; \overline{v_n}$; for V an empty set, $\overline{V} = \cdot$.
- For R a set of rewrite rules $\{r_1, \ldots, r_n\}$, $\overline{R} = \overline{r_1}; \ldots; \overline{r_n}$; for R an empty set, $\overline{R} = \cdot$.
- For r a rewrite rule $(t \longrightarrow t')$, $\overline{r} = (\overline{t} \Rightarrow \overline{t'})$.
- For A a set of assignments $(w_1/x_1, \ldots, w_n/x_n)$, $\overline{A} = \overline{w_1} \mapsto \overline{x_1}; \ldots; \overline{w_n} \mapsto \overline{x_n}$; for A an empty set, $\overline{A} = \cdot$.
- For t a term $f(t_1, \ldots, t_n)$, $f \in \Sigma_n$, $n > 0$, $\overline{t} = \mathrm{op}\{\hat{f}\}[\overline{t_1}; \ldots; \overline{t_n}]$; for t a term c, $c \in \Sigma_0$, $\overline{t} = \mathrm{op}\{\hat{c}\}[\cdot]$; and for t a term v, $v \in \mathit{Var}$, $\overline{t} = \mathrm{var}\{\hat{v}\}[\cdot]$. Note that we have assumed that all operators and variables are strings of ASCII characters.
- For a string of ASCII characters $l = a_1\, a_2 \ldots a_n$, $\hat{l} = a_1.a_2.\ldots.a_n$.

We next introduce the set E_U of equations,

$$(x; \cdot) = x = (\cdot; x)$$
$$x; (y; z) = (x; y); z$$

Therefore, the operation symbol '$\_;\_$' is declared associative with '$\cdot$' as its identity element.

We next introduce by groups the set R_U of rules, and explain their intended meaning. As we shall prove later, U itself belongs to the class $\mathcal{C}$ of finitely presentable rewrite theories and with the representation function $\overline{(\_ \vdash \_)}$ makes the entailment system of rewriting logic reflective. This means that U reifies the entailment relation, and that the operation symbols, equations, and rules of U can be seen as a specification in rewriting logic of its own rules of deduction, including the congruence and replacement rules. The reflexivity and transitivity rules of deduction for a theory $T \in \mathcal{C}$ are directly mirrored by the reflexivity and transitivity rules of deduction for U.

Note that rewriting logic has also a proof calculus [23]. By extending the definition of U along the lines of [16], so as to make explicit and reify the proofs built up by the deduction process, one can similarly exhibit a finitely presentable universal theory U' making the proof calculus of rewriting logic $\mathcal{C}$-reflective, as defined in [6].

To ease readability, variables in the rules appear in *italics*; this is shorthand notation for the convention that all variables are character strings beginning with a quote and having length at least two, so that no ambiguity may ever arise with the ASCII characters themselves, that are constants of Σ_U. We also introduce the notation $t \longleftrightarrow t'$ to indicate a bidirectional rule, that is, a pair of rules $t \longrightarrow t'$ and $t' \longrightarrow t$.

To reify the rule of congruence we will use contexts and (potential) redexes. In fact, our idea is to use contexts and redexes to combine the rules of congruence and replacement. As a result, a step of reified replacement will be taken in any subterm of a reified subject term. In particular, we use the ASCII character '$*$' and the operation symbol '$\_[\rightarrow \_]$' in Σ_U. In rule 1 below we use these operation symbols to decompose a term $\overline{t}$ to be rewritten into a context and a potential redex. The intended meaning of rules 1 is to indicate the subterm $\overline{t_1}$ of $\overline{t}$ in which a step of rewriting will be attempted.

$$t@u[\rightarrow p[l; q[l_1]; l']] \overset{1}{\longleftrightarrow} t@u[\rightarrow p[l; *; l']][\rightarrow q[l_1]]$$

To reify the rule of replacement we have first to reify the condition for its application. Given a rule $t_1(x_1, \ldots, x_n) \longrightarrow t_2(x_1, \ldots, x_n)$, a replacement can be made in a term t iff $t = t_1(\vec{w}/\vec{x})$. We use the rule 2 below to *set aside* a matching problem between the lefthand side of a rule and a potential redex. For that we have introduced in Σ_U the operation symbol '$\_ :: \langle \_, \_, \_, \_ \rangle$'. The first argument of '$\_ :: \langle \_, \_, \_, \_ \rangle$' is a pair formed by a rewrite theory and a term decomposed into a context and a potential redex, to which the matching problem is related; the second and third arguments are the lefthand side of a rule and the potential redex respectively. We will use the operation symbols '$\triangleright$' and '$\_[\hookrightarrow \_]$' in Σ_U during the matching process of any two terms to simultaneously decompose both, in a similar way that '$*$' and '$\_[\rightarrow \_]$' are used in rule 1. But in addition, '$\triangleright$' will divide the lists of subterms of any terms being matched into a list of subterms already matched and a list of subterms that have to be matched. Note that rule 2 sets to empty the lists of subterms already matched of the lefthand side of the rule and the potential redex. The fourth argument consists of a pair built up with the operation symbol '_/_'. The first element of this pair represents a set of assignments, and the second a set of variables. As we will see below, this argument is needed to handle the case of nonlinear lefthand sides in which a variable can have several occurrences. Note that rule 2 sets to empty the initial set of assignments, and sets the initial set of variables to the set of variables of the theory. Finally, the fifth argument is the righthand side of the selected rule. As we shall see below, this allows us to continue the reified replacement process without having to keep track of the rule that we have selected.

$$\begin{array}{l}\langle\, v \;,\; r; (p_1[l_1] \Rightarrow q); r' \rangle @ u[\rightarrow p[l]] \xrightarrow{2} \\ \langle\, v \;,\; r; (p_1[l_1] \Rightarrow q); r' \rangle \ddot{@} u[\rightarrow p[l]] :: \langle \rhd[\hookrightarrow p_1[\rhd; l_1]] \;,\; \rhd[\hookrightarrow p[\rhd; l]] \;,\; \cdot / v \;,\; q \rangle\end{array}$$

The matching problem between any terms $\overline{t}$ and $\overline{t_1}$ will be handled by rules 3–7 below. As expected, they will try to come out with a set of assignments A, such that $\overline{t}$ substituted by A is equal to $\overline{t_1}$. The reified matching process can be seen as a recursive process trying to identify $\overline{t}$ and $\overline{t_1}$ while keeping track of their differences in A. In particular, the rules 3 and 6 indicate that any terms with the same top operator will match only if all their subterms match, and the matching of these subterms will be attempted from left to right. To handle the case of non-linear lefthand sides, we use the pair A/V. The idea is to keep the set A of variables already assigned and the set V of variables not yet assigned disjoint from each other. Note that the initial set V is the set of variables in the theory. When in the reified matching process we reach the base case of matching a variable $\overline{x_i}$ and a term $\overline{t}$, we consider two cases: rule 4 when $\overline{x_i}$ is in V, and rule 5 when $\overline{x_i}$ is in the set of variables in A, because $\overline{x}$ has already been encountered in another occurrence during the matching. We use the operation symbol '$\_[\hookleftarrow \_]$' in Σ_U to indicate that a subterm has been succesfully matched. Then, rule 7 simultaneously inserts the subterms successfully matched at the end of the corresponding list.

$$\begin{array}{l} z :: \langle u[\hookrightarrow \mathrm{op}\{s\}[l; \rhd; p_1[l_1]; l']] \;,\; u[\hookrightarrow \mathrm{op}\{s\}[l; \rhd; p[l]; l'']] \;,\; a/v \;,\; q \rangle \xrightarrow{3} \\ z :: \langle u[\hookrightarrow \mathrm{op}\{s\}[l; \rhd; l']][\hookrightarrow p_1[\rhd; l_1]] \;,\; u[\hookrightarrow \mathrm{op}\{s\}[l; \rhd; l'']][\hookrightarrow p[\rhd; l]] \;,\; a/v \;,\; q \rangle \end{array}$$

$$\begin{array}{l} z :: \langle u[\hookrightarrow \mathrm{var}\{x\}[\rhd; \cdot]] \;,\; u[\hookrightarrow p[\rhd; l]] \;,\; a/(v; (\mathrm{var}\{x\}[\cdot]); v') \;,\; q \rangle \xrightarrow{4} \\ z :: \langle u[\hookleftarrow p[l]] \;,\; u[\hookleftarrow p[l]] \;,\; ((p[l] \mapsto \mathrm{var}\{x\}[\cdot]); a)/(v; v') \;,\; q \rangle \end{array}$$

$$\begin{array}{l} z :: \langle u[\hookrightarrow \mathrm{var}\{x\}[\rhd; \cdot]] \;,\; u[\hookrightarrow p[\rhd; l]] \;,\; (a; (p[l] \mapsto \mathrm{var}\{x\}[\cdot]); a')/v \;,\; q \rangle \xrightarrow{5} \\ z :: \langle u[\hookleftarrow p[l]] \;,\; u[\hookleftarrow p[l]] \;,\; (a; (p[l] \mapsto \mathrm{var}\{x\}[\cdot]); a')/v \;,\; q \rangle \end{array}$$

$$\begin{array}{l} z :: \langle u[\hookrightarrow \mathrm{op}\{s\}[l; \rhd]] \;,\; u[\hookrightarrow \mathrm{op}\{s\}[l; \rhd]] \;,\; a/v \;,\; q \rangle \xrightarrow{6} \\ z :: \langle u[\hookleftarrow \mathrm{op}\{s\}[l]] \;,\; u[\hookleftarrow \mathrm{op}\{s\}[l]] a/v \;,\; q \rangle \end{array}$$

$$z :: \langle u[\hookrightarrow \mathrm{op}\{s\}[l; \triangleright; l']][\hookleftarrow p] \ , \ u[\hookrightarrow \mathrm{op}\{s\}[l; \triangleright; l'']][\hookleftarrow p] \ , \ a/v \ , \ q\rangle \xrightarrow{7}$$
$$z :: \langle u[\hookrightarrow \mathrm{op}\{s\}[l; p; \triangleright; l']] \ , \ u[\hookrightarrow \mathrm{op}\{s\}[l; p; \triangleright; l'']] \ , \ a/v \ , \ q\rangle$$

A matching process is terminated using rule 8 below. Note that this rule can only be applied when the matching process has been succesful, that is, when, after a number of applications of the rules 3–7 a potential redex $\overline{t}$ has been made equal to the lefthand side of a rule $\overline{t_1}$, and A has become the set of assignments such that $\overline{t_1}$ substituted by A is equal to $\overline{t}$.

$$z :: \langle \triangleright[\hookleftarrow p] \ , \triangleright[\hookleftarrow p] \ , \ a/v \ , \ p_2[l_2]\rangle \xrightarrow{8} z :: \langle \triangleright[\hookrightarrow p_2[\triangleright; l_2]] \ , \ a\rangle$$

An application of rule 8 changes the matching task into a replacement task. For that we have introduced in Σ_U the operation symbol '$\_ :: \langle \_ \ , \ \_\rangle$'. The first argument of '$\_ :: \langle \_ \ , \ \_\rangle$' is a pair formed by a rewrite theory an a term decomposed into a context an a redex; the second argument is the righthand side $\overline{t_2}$ of a rule whose lefthand side has been succesfully matched with the redex; the third argument is the set A of assignments resulting from this matching process. A replacement task is carried out as a two-phase process: the first phase consists in applying the substitution A to $\overline{t_2}$, and the second consists in the actual replacement using the result of this substitution.

The application of the substitution A to the term $\overline{t_2}$ is performed by rules 9–12 below and follows the same recursive style as the matching process.

$$z :: \langle u[\hookrightarrow \mathrm{op}\{s\}[l; \triangleright; q[l_1]; l']] \ , \ a\rangle \xrightarrow{9} z :: \langle u[\hookrightarrow \mathrm{op}\{s\}[l; \triangleright; l']][\hookrightarrow q[\triangleright; l_1]] \ , \ a\rangle$$

$$z :: \langle u[\hookrightarrow \mathrm{var}\{x\}[\triangleright; \cdot]] \ , \ (a; (p \mapsto \mathrm{var}\{x\}[\cdot]); a')\rangle \xrightarrow{10}$$
$$z : \langle u[\hookleftarrow p] \ , \ (a; (p \mapsto \mathrm{var}\{x\}[\cdot]); a')\rangle$$

$$z :: \langle u[\hookrightarrow \mathrm{op}\{s\}[l; \triangleright]] \ , \ a\rangle \xrightarrow{11} z :: \langle u[\hookleftarrow \mathrm{op}\{s\}[l]] \ , \ a\rangle$$

$$z :: \langle u[\hookrightarrow \mathrm{op}\{s\}[l; \triangleright; l'][\hookleftarrow p]] \ , \ a\rangle \xrightarrow{12} z :: \langle u[\hookrightarrow \mathrm{op}\{s\}[l; p; \triangleright; l']] \ , \ a\rangle$$

Finally, the actual replacement is handled in the expected way by rule 14 below.

$$t\ddot{@}u[\rightarrow p] :: \langle \triangleright[\hookleftarrow p'] \ , \ a\rangle \xrightarrow{13} t@u[\rightarrow p']$$

3.3 Correctness of the universal theory

In this section we prove that the representation function $\overline{(\_ \vdash \_)}$ makes the theory U a $\mathcal{C}$-universal theory, for $\mathcal{C}$ the class of unconditional and unsorted finitely presentable rewrite theories. We also assume that for any rule $t \longrightarrow t'$ in $T \in \mathcal{C}$, $\mathrm{var}(t') \subseteq \mathrm{var}(t)$. Thus, the main result will be the following,

Theorem 3.2 *For any theory $T = (\Sigma, \emptyset, R) \in \mathcal{C}$, $t, t' \in T_\Sigma$,*

$$T \vdash t \longrightarrow t' \iff U \vdash \overline{T \vdash t \longrightarrow t'}.$$

This theorem will be proved by structural induction on rewriting logic proofs. Consider now the following rules of deduction:

2′. **S-Congruence** For each $f \in \Sigma_n$, $n \in \mathbb{N}$,

$$\frac{t_i \longrightarrow t'_i}{f(t_1, \ldots, t_{i-1}, t_i, t_{i+1}, \ldots, t_n) \longrightarrow f(t_1, \ldots, t_{i-1}, t'_i, t_{i+1}, \ldots, t_n)}$$

3′. **S-Replacement** For each rewrite rule $t(x_1, \ldots, x_n) \longrightarrow t'(x_1, \ldots, x_n)$ in T,

$$\frac{}{t(w_1/x_1, \ldots, w_n/x_n) \longrightarrow t'(w_1/x_1, \ldots, w_n/x_n)}$$

Lemma 3.3 *Any sequent derivable by the rules* 1–4*, can be derived by the rules* $\{1, 2', 3', 4\}$.

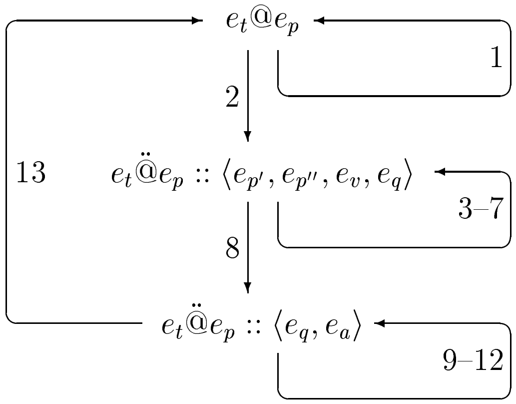
Given a theory $T \in \mathcal{C}$, a rewrite $t \longrightarrow t'$ is called a *one-step rewrite* if and only if it can be derived from T by finite application of the rules 1–2′–3′, with at least one application of the rule 3′. We say that a term is *irreducible* if and only if there does not exit any one-step rewrite $t \longrightarrow t'$. Finally, we say that a term $f(t_1, \ldots, t_n)$ is *only reducible at the top* if and only if t_i is irreducible for $1 \leq i \leq n$.

Lemma 3.4 *For any rule* $t(x_1, \ldots, x_n) \longrightarrow t'(x_1, \ldots, x_n)$ *in* U*, and any substitution* $w_1/x_1, \ldots, w_n/x_n$*, such that* w_i *is irreducible for* $1 \leq i \leq n$*, then* $t'(w_1/x_1, \ldots, w_n/x_n)$ *is only reducible at the top.*

As a corollary we can state the following lemma,

Lemma 3.5 *Any rewrite* $t \longrightarrow t'$ *derivable in* U *by the rules* 1–4*, and such that* t *is only reducible at the top, can be derived in* U *by the rules* $\{1, 3', 4\}$.

In particular, any rewrite $t \longrightarrow t'$ derivable in U by the rules $\{1, 3', 4\}$ will be an instance of a path in the following automaton, where a label l in an arc indicates a s-replacement rewrite with rule l. From now on, the metavariable e (possibly primed and/or subindexed) ranks over the set of irreducible terms in U_Σ.



In order to prove the direction ($\Rightarrow$) in Theorem 3.2 we prove first several technical lemmas.

For any terms e_u, e_v, we write $e_u \bigtriangledown e_v$ if and only if either $e_u = *$, or $e_u = e_{u'}[\rightarrow e\,]$, $e_v = e_{v'}[\rightarrow e\,]$, and $e_{u'} \bigtriangledown e_{v'}$. Also, for any terms e_u, e_v such that

$e_u \bigtriangledown e_v$,

$$e_v - e_u = \begin{cases} e_v & \text{if } e_u = * \\ e_{v'} - e_{u'} & \text{if } e_v = e_{v'}[\leftarrow e\,] \text{ and } e_u = e_{u'}[\leftarrow e\,]. \end{cases}$$

Lemma 3.6 *Given a rewrite* $\overline{T}@e_u[\to \overline{t}\,] \longrightarrow \overline{T}@e_{u'}[\to \overline{t'}\,]$*, then, for any term* e_v *such that* $e_u \bigtriangledown e_v$*, there is a rewrite* $\overline{T}@e_v[\to \overline{t}\,] \longrightarrow \overline{T}@e_{v'}[\to \overline{t'}\,]$ *such that* $e_{u'} \bigtriangledown e_{v'}$ *and* $e_v - e_u = e_{v'} - e_{u'}$*.*

Proof. By Lemma 3.5, $\overline{T}@e_u[\to \overline{t}\,] \longrightarrow \overline{T}@e_{u'}[\to \overline{t'}\,]$, is a composition of n rewrites of any of these forms,

(1) $\overline{T}@e_u[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};\overline{t_i};\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,] \xrightarrow{+1}$
$\overline{T}@e_u[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};*;\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,][\to \overline{t_i}\,]$,

(2) $\overline{T}@e_u[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};*;\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,][\to \overline{t_i}\,] \xrightarrow{-1}$
$\overline{T}@e_u[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};\overline{t_i};\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,]$, or

(3) $\overline{T}@e_u[\to \overline{t}\,] \xrightarrow{2;(3-7)^*;8;(9-12)^*;13} \overline{T}@e_u[\to \overline{t'}\,]$,

where, from now on, $+1$ and -1, indicate a s-replacement rewrite with rule 1 from left to right and from right to left, respectively; also, ";" indicates sequential composition of rewrites. It is easy to prove that given a rewrite $\overline{T}@e_u[\to \overline{t}\,] \longrightarrow \overline{T}@e_{u'}[\to \overline{t'}\,]$ of the form (1), (2) or (3), for any term e_v such that $e_u \bigtriangledown e_v$, there is a rewrite $\overline{T}@e_v[\to \overline{t}\,] \longrightarrow \overline{T}@e_{v'}[\to \overline{t'}\,]$ such that $e_{u'} \bigtriangledown e_{v'}$ and $e_v - e_u = e_{v'} - e_{u'}$. Then, applying transitivity $n-1$ times we obtain the desired result. □

Lemma 3.7 *For any theory* $T = (\Sigma, \emptyset, R)$ *in* $\mathcal{C}$*,* $f \in \Sigma_n$*,* $t_1, \ldots, t_n \in T_\Sigma$*, for any* $1 \le i \le n$*, if there is a rewrite* $\overline{T}@*[\to \overline{t_i}\,] \longrightarrow *[\to \overline{t'_i}\,]$*, then there is a rewrite*

$$\overline{T}\,@*[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};\overline{t_i};\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,] \longrightarrow$$
$$\overline{T}\,@*[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};\overline{t'_i};\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,].$$

Proof. The following is a rewrite in U,

$$\overline{T}\,@*[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};\overline{t_i};\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,] \xrightarrow{+1}$$
$$\overline{T}\,@*[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};*;\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,][\to \overline{t_i}\,] \xrightarrow{\delta}$$
$$\overline{T}\,@*[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};*;\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,][\to \overline{t'_i}\,] \xrightarrow{-1}$$
$$\overline{T}\,@*[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};\overline{t'_i};\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,],$$

where the rewrite δ exists by Lemma 3.6. □

From now on, we assume that for any set of assignments $A = (\vec{w}/\vec{x})$, $w_i \in T_\Sigma$, for $1 \le i \le n$. Also, given a set of assignments A, $\mathrm{var}(A)$ is the set of variables assigned in A. Finally, for any sets of assignments A, A', we write $A \diamondsuit A'$ if and only if for any terms $t \in T_\Sigma(X)$, $t' \in T_\Sigma$, if $t' = t(A')$, then $t' = t(A)(A')$. In particular, if $A \diamondsuit A'$, then for any assignment $a \in A'$, $A \cup \{a\} \diamondsuit A'$.

Lemma 3.8 *For any set of assignments A, and any set of variables V, given a rewrite*

$$e_z :: \langle e_p, e_{p_1}, \overline{A}/\overline{V}, e_q\rangle \xrightarrow{(3-7)^*} e_z :: \langle e_{p'}, e_{p_2}, \overline{A'}/\overline{V'}, e_q\rangle,$$

then $A \subseteq A'$.

Lemma 3.9 *Given a theory $T = (\Sigma, \emptyset, R)$ in $\mathcal{C}$, $t \in T_\Sigma(X)$, $t' \in T_\Sigma$, and $t' = t(\vec{w}/\vec{x})$, then, for any set of assignments A, and any set of variables V, such that* $\mathrm{var}(A) \cap V = \emptyset$, $(\vec{x}) \subseteq \mathrm{var}(A) \cup V$, *and $A \diamondsuit (\vec{w}/\vec{x})$, there is a rewrite*

$$\begin{aligned}&e_z :: \langle e_u[\hookrightarrow \triangleright(\overline{t})\,]\ ,\ e_u[\hookrightarrow \triangleright(\overline{t'})\,]\ ,\ \overline{A}/\overline{V}\ ,\ e_q\rangle \longrightarrow\\ &e_z :: \langle e_u[\hookleftarrow \overline{t'}\,]\ ,\ e_u[\hookleftarrow \overline{t'}\,]\ ,\ \overline{A'}/\overline{V'}\ ,\ e_q\rangle\end{aligned}$$

such that $t' = t(A')$, $\mathrm{var}(A') \cap V' = \emptyset$, $(\vec{x}) \subseteq \mathrm{var}(A') \cup V'$, *and $A' \diamondsuit (\vec{w}/\vec{x})$, where from now on, for any $t \in T_\Sigma(X)$, such that $\overline{t} = e_p[e_l\,]$, $\triangleright(\overline{t}) = e_p[\triangleright, e_l\,]$.*

Proof. By structural induction on t.

(1) $t = c$, for $c \in \Sigma_0$, $t' = c(\vec{w}/\vec{x})$. Then, the following is a rewrite in U

$$\begin{aligned}&e_z :: \langle e_u[\hookrightarrow \triangleright(\overline{c})\,]\ ,\ e_u[\hookrightarrow \triangleright(\overline{c})\,]\ ,\ \overline{A}/\overline{V}\ ,\ e_q\rangle \xrightarrow{6}\\ &e_z :: \langle e_u[\hookleftarrow \overline{c}\,]\ ,\ e_u[\hookleftarrow \overline{c}\,]\ ,\ \overline{A}/\overline{V}\ ,\ e_q\rangle,\end{aligned}$$

such that $c = c(A)$.

(2) $t = x_i$, $w_i = x_i(\vec{w}/\vec{x}) = w_i$. We have to consider two subcases:

(a) $x_i \in V$. Then, the following is a rewrite in U

$$\begin{aligned}&e_z :: \langle e_u[\hookrightarrow \triangleright(\overline{x_i})\,]\ ,\ e_u[\hookrightarrow \triangleright(\overline{w_i})\,]\ ,\ \overline{A}/\overline{V}\ ,\ e_q\rangle \xrightarrow{4}\\ &e_z :: \langle e_u[\hookleftarrow \overline{w_i}\,]\ ,\ e_u[\hookleftarrow \overline{w_i}\,]\ ,\ (\overline{A \cup \{w_i/x_i\}})/(\overline{V - \{x_i\}})\ ,\ e_q\rangle,\end{aligned}$$

such that $w_i = x_i(A \cup \{w_i/x_i\})$, $\mathrm{var}(A \cup \{w_i/x_i\}) \cap (V - \{x_i\}) = \emptyset$, $(\vec{x}) \subseteq \mathrm{var}(A \cup \{w_i/x_i\}) \cup (V - \{x_i\})$, and $A \cup \{w_i/x_i\} \diamondsuit (\vec{w}/\vec{x})$.

(b) $x_i \in \mathrm{var}(A)$. By assumption $A \diamondsuit (\vec{w}/\vec{x})$. Then, the following is a rewrite in U

$$\begin{aligned}&e_z :: \langle e_u[\hookrightarrow \triangleright(\overline{x_i})\,]\ ,\ e_u[\hookrightarrow \triangleright(\overline{w_i})\,]\ ,\ \overline{A}/\overline{V}\ ,\ e_q\rangle \xrightarrow{5}\\ &e_z :: \langle e_u[\hookleftarrow \overline{w_i}\,]\ ,\ e_u[\hookleftarrow \overline{w_i}\,]\ ,\ \overline{A}/\overline{V}\ ,\ e_q\rangle,\end{aligned}$$

with $w_i = x_i(A)$.

(3) $t = f(t_1, \ldots, t_n)$, $f \in \Sigma_n$, $n > 0$, $t' = f(t'_1, \ldots, t'_n)$, $t'_i = t_i(\vec{w}/\vec{x})$, for $1 \leq i \leq n$. Then, the following is a rewrite in U,

$$\begin{aligned}&e_z :: \langle e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\triangleright; \overline{t_1}; \ldots; \overline{t_n}\,]]\ ,\ e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\triangleright, \overline{t'_1}; \ldots; \overline{t'_n}\,]]\ ,\ \overline{A}/\overline{V}\ ,\ e_q\rangle \xrightarrow{\beta}\\ &e_z :: \langle e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\,\overline{t'_1}; \ldots; \overline{t'_n}; \triangleright]]\ ,\ e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\,\overline{t'_1}; \ldots; \overline{t'_n}; \triangleright]]\ ,\ \overline{A'}/\overline{V'}\ ,\ e_q\rangle \xrightarrow{6}\\ &e_z :: \langle e_u[\hookleftarrow \mathrm{op}\{\hat{f}\}[\,\overline{t'_1}; \ldots; \overline{t'_n}\,]]\ ,\ e_u[\hookleftarrow \mathrm{op}\{\hat{f}\}[\,\overline{t'_1}; \ldots; \overline{t'_n}\,]]\ ,\ \overline{A'}/\overline{V'}\ ,\ e_q\rangle,\end{aligned}$$

where the rewrite β is a composition of n rewrites of the form

$$\begin{aligned}&e_z :: \langle\, e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\,\overline{t'_1}; \ldots; \overline{t'_{i-1}}; \triangleright; \overline{t_i}; \ldots; \overline{t_n}\,]]\ ,\\ &\qquad e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\,\overline{t'_1}; \ldots; \overline{t'_{i-1}}; \triangleright; \overline{t'_i}; \ldots; \overline{t'_n}\,]]\ ,\ \overline{A_i}/\overline{V_i}\ ,\ e_q\rangle \xrightarrow{3}\\ &e_z :: \langle\, e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\,\overline{t'_1}; \ldots; \overline{t'_{i-1}}; \triangleright; \ldots; \overline{t_n}\,]][\hookrightarrow \triangleright(\overline{t_i})\,]\ ,\\ &\qquad e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\,\overline{t'_1}; \ldots; \overline{t'_{i-1}}; \triangleright; \ldots; \overline{t'_n}\,]][\hookrightarrow \triangleright(\overline{t'_i})\,]\ ,\ \overline{A_i}/\overline{V_i}\ ,\ e_q\rangle \xrightarrow{\beta_i}\end{aligned}$$

$$\begin{aligned}
&e_z :: \langle\, e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\,\overline{t'_1};\ldots;\overline{t'_{i-1}};\triangleright;\ldots;\overline{t_n}\,]\,][\hookleftarrow \overline{t'_i}\,]\ ,\\
&\qquad e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\,\overline{t'_1};\ldots;\overline{t'_{i-1}};\triangleright;\ldots;\overline{t'_n}\,]\,][\hookleftarrow \overline{t'_i}\,]\ ,\ \overline{A'_i/V'_i}\ ,\ e_q\rangle \xrightarrow{7}\\
&e_z :: \langle\, e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\,\overline{t'_1};\ldots;\overline{t'_{i-1}};\overline{t'_i};\triangleright;\ldots;\overline{t_n}\,]\,]\ ,\\
&\qquad e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\,\overline{t'_1};\ldots;\overline{t'_{i-1}};\overline{t'_i};\triangleright;\ldots;\overline{t'_n}\,]\,]\ ,\ \overline{A'_i/V'_i}\ ,\ e_q\rangle,
\end{aligned}$$

where, by induction hypothesis, the rewrite β_i exists, for $1 \leq i \leq n$, and is such that $t'_i = t_i(A'_i)$, $\mathrm{var}(A'_i) \cap V'_i = \emptyset$, $(\vec{x}) \subseteq \mathrm{var}(A'_i) \cup V'_i$, $A'_i \diamondsuit (\vec{w}/\vec{x})$. Note also that by Lemma 3.8,

$$A = A_1 \subseteq A'_1 = A_2 \subseteq \ldots \subseteq A'_n = A'.$$

Therefore, $f(t'_1,\ldots,t'_n) = f(t_1,\ldots,t_n)(A')$, $\mathrm{var}(A') \cap V' = \emptyset$, $(\vec{x}) \subseteq \mathrm{var}(A') \cup V'$, and $A' \diamondsuit (\vec{w}/\vec{x})$.

□

Lemma 3.10 *Given a theory $T = (\Sigma, \emptyset, R)$ in $\mathcal{C}$, $t \in T_\Sigma(X)$, $t' \in T_\Sigma$, and $t' = t(\vec{w}/\vec{x})$, there is a rewrite*

$$e_z :: \langle e_u[\hookrightarrow \triangleright(\overline{t})\,]\ ,\ \overline{\vec{w}/\vec{x}}\rangle \longrightarrow e_z :: \langle e_u[\hookleftarrow \overline{t'}\,]\ ,\ \overline{\vec{w}/\vec{x}}\rangle.$$

Proof. By structural induction on t analogously to the proof of Lemma 3.9.□

Theorem 3.11 *For any theory $T = (\Sigma, \emptyset, R)$ in $\mathcal{C}$, $t, t' \in T_\Sigma$,*

$$T \vdash t \longrightarrow t' \quad\Longrightarrow\quad U \vdash \overline{T}@*[\rightarrow \overline{t}\,] \longrightarrow \overline{T}@*[\rightarrow \overline{t'}\,]$$

Proof. By structural induction on rewriting logic proofs. For the reflexivity and transitivity rules, the result is obvious.

(1) S-Congruence. Given a rewrite in T

$$f(t_1,\ldots,t_{i-1},t_i,t_{i+1},\ldots,t_n) \longrightarrow f(t_1,\ldots,t_{i-1},t'_i,t_{i+1},\ldots,t_n),$$

with $t_i \longrightarrow t'_i$ as premise, by induction hypothesis there is a rewrite in U

$$\overline{T}@*[\rightarrow \overline{t_i}\,] \longrightarrow \overline{T}@*[\rightarrow \overline{t'_i}\,],$$

and, by Lemma 3.7, the following is also a rewrite in U

$$\begin{aligned}
&\overline{T}@*[\rightarrow \overline{f(t_1,\ldots,t_{i-1},t_i,t_{i+1},\ldots,t_n)}\,] \longrightarrow\\
&\overline{T}@*[\rightarrow \overline{f(t_1,\ldots,t_{i-1},t'_i,t_{i+1},\ldots,t_n)}\,]
\end{aligned}$$

(2) S-Replacement. Given a rewrite $t_1 \longrightarrow t_2$ in T, such that $t(\vec{x}) \longrightarrow t'(\vec{x})$ is a rule in T, and $t_1 = t(\vec{w}/\vec{x})$, $t_2 = t'(\vec{w}/\vec{x})$, the following is a rewrite in U

$$\begin{aligned}
&\langle\,\overline{V}\ ,\ (\ldots;\overline{t}\Rightarrow\overline{t'};\ldots)\rangle@*[\rightarrow \overline{t_1}\,] \xrightarrow{2}\\
&\langle\,\overline{V}\ ,\ (\ldots;\overline{t}\Rightarrow\overline{t'};\ldots)\rangle\ddot{@}*[\rightarrow \overline{t_1}\,] :: \langle \triangleright[\hookrightarrow \triangleright(\overline{t})\,]\ ,\ \triangleright[\hookrightarrow \triangleright(\overline{t_1})\,]\ ,\ \cdot/\overline{V}\ ,\ \overline{t'}\rangle \xrightarrow{\alpha}\\
&\langle\,\overline{V}\ ,\ (\ldots;\overline{t}\Rightarrow\overline{t'};\ldots)\rangle\ddot{@}*[\rightarrow \overline{t_1}\,] :: \langle \triangleright[\hookleftarrow \overline{t_1}\,]\ ,\ \triangleright[\hookleftarrow \overline{t_1}\,]\ ,\ \overline{A'}/\overline{V'}\ ,\ \overline{t'}\rangle \xrightarrow{8}\\
&\langle\,\overline{V}\ ,\ (\ldots;\overline{t}\Rightarrow\overline{t'};\ldots)\rangle\ddot{@}*[\rightarrow \overline{t_1}\,] :: \langle \triangleright[\hookrightarrow \triangleright\overline{t'}\,]\ ,\ \overline{A'}\rangle \xrightarrow{\beta}\\
&\langle\,\overline{V}\ ,\ (\ldots;\overline{t}\Rightarrow\overline{t'};\ldots)\rangle\ddot{@}*[\rightarrow \overline{t_1}\,] :: \langle \triangleright[\hookleftarrow \overline{t_2}\,]\ ,\ \overline{A'}\rangle \xrightarrow{13}\\
&\langle\,\overline{V}\ ,\ (\ldots;\overline{t}\Rightarrow\overline{t'};\ldots)\rangle@*[\rightarrow \overline{t_2}\,]
\end{aligned}$$

where by Lemma 3.9 the rewrite α exists, and is such that $t(A) = t_1 = t(\vec{w}/\vec{x})$, and by Lemma 3.10 the rewrite β exists as well, since by assumption on the class of theories we are considering, $\mathrm{var}(t') \subseteq \mathrm{var}(t)$, and therefore, $t(A) = t_1 = t(\vec{w}/\vec{x})$ implies $t'(A) = t_2 = t'(\vec{w}/\vec{x})$.

$\square$

In order to prove the direction ($\Leftarrow$) in Theorem 3.2 we prove first several technical lemmas, regarding the correctness of the rules for selecting a subterm (Lemma 3.12), solving a matching problem with the lefthand side of a rule (Lemma 3.17), and performing the consequent substitution on the righthand side of the rule (Lemma 3.18). We define now a partial function ($\underline{e}$) that intuitively is the inverse function of ($\overline{e}$), but also gives a term from decomposed representation of it, obtained by s-replacement rewrites with rule 1. In particular,

$$\underline{e} = \begin{cases} t & \text{if } e = *[\to \overline{t}\,] \\ \underline{e_{u'}[\to \mathrm{op}\{\hat{f}\}[\ldots;\overline{t};\ldots]\,]} & \text{if } e = e_{u'}[\to \mathrm{op}\{\hat{f}\}[\ldots;*;\ldots]\,][\to \overline{t}\,] \end{cases}$$

Lemma 3.12 *For any theory $T = (\Sigma, \emptyset, R) \in \mathcal{C}$, $t, t' \in T_\Sigma$, given a rewrite*

$$\overline{T}@e_u[\to \overline{t'}\,] \xrightarrow{1^*} \overline{T}@e_{u'}[\to e\,],$$

if $\underline{e_u[\to \overline{t'}\,]} = t$, then $\underline{e_{u'}[\to e\,]} = t$.

Proof. By induction on the number of applications of rules $+1$ and -1. $\square$

Lemma 3.13 *For any theory $T = (\Sigma, \emptyset, R) \in \mathcal{C}$, $t, t' \in T_\Sigma$, given a rewrite*

$$\overline{T}@e_u[\to \overline{t'}\,] \xrightarrow{1^*} \overline{T}@e'_u[\to e\,],$$

if $\underline{e_u[\to \overline{t'}\,]} = t$, then there is a rewrite

$$\overline{T}@e_u[\to \overline{t'}\,] \xrightarrow{-1^*} \overline{T}@*[\to \overline{t}\,] \xrightarrow{+1^*} \overline{T}@e'_u[\to e\,].$$

Lemma 3.14 *For any theory $T = (\Sigma, \emptyset, R) \in \mathcal{C}$, $t, t', t_1, t_2 \in T_\Sigma$, given a rewrite*

$$\overline{T}@e_u[\to \overline{t_1}\,] \longrightarrow \overline{T}@e_u[\to \overline{t_2}\,],$$

such that, $t = \underline{e_u[\to \overline{t_1}\,]}$, $t' = \underline{e_u[\to \overline{t_2}\,]}$, if $T \vdash t_1 \longrightarrow t_2$, then $T \vdash t \longrightarrow t'$.

Proof. By induction on the size of e_u, where $\mathrm{size}(*) = 1$, and $\mathrm{size}(e_{u'}[\to e\,]) = 1 + \mathrm{size}(e_{u'})$. For $\mathrm{size}(e_u) = 1$, the result is obvious. For $\mathrm{size}(e_u) = n+1$, assume that the lemma holds for any $e_{u'}$ such that $\mathrm{size}(e_{u'}) = n$. Then, given a rewrite

$$\begin{array}{l} \overline{T} @ e_{u'}[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};*;\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,][\to \overline{t_i}\,] \longrightarrow \\ \overline{T} @ e_{u'}[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};*;\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,][\to \overline{t'_i}\,], \end{array}$$

note that

$$\begin{array}{l} \underline{e_{u'}[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};*;\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,][\to \overline{t_i}\,]} = \\ \underline{e_{u'}[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};\overline{t_i};\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,]}, \text{ and} \end{array}$$

$$\begin{array}{l} \underline{e_{u'}[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};*;\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,][\to \overline{t'_i}\,]} = \\ \underline{e_{u'}[\to \mathrm{op}\{\hat{f}\}[\,\overline{t_1};\ldots;\overline{t_{i-1}};\overline{t'_i};\overline{t_{i+1}};\ldots;\overline{t_n}\,]\,]}. \end{array}$$

Now, by assumption, $T \vdash t_i \longrightarrow t'_i$. Thus, by s-congruence,

$$T \vdash f(t_1, \ldots, t_{i-1}, t_i, t_{i+1}, \ldots, t_n) \longrightarrow f(t_1, \ldots, t_{i-1}, t'_i, t_{i+1}, \ldots, t_n),$$

and, finally, by induction hypothesis, we obtain the desired result

$$T \vdash \underline{e_{u'}[\rightarrow \mathrm{op}\{\hat{f}\}[\,\overline{t_1}; \ldots; \overline{t_{i-1}}; \overline{t_i}; \overline{t_{i+1}}; \ldots; \overline{t_n}\,]\,]} \longrightarrow$$
$$\underline{e_{u'}[\rightarrow \mathrm{op}\{\hat{f}\}[\,\overline{t_1}; \ldots; \overline{t_{i-1}}; \overline{t'_i}; \overline{t_{i+1}}; \ldots; \overline{t_n}\,]\,]}.$$

□

Next, we define a partial function $\mathrm{snm}(e)$ that intuitively gives the cardinality of the set of subterms not yet matched of a term that is being matched:

$$\mathrm{snm}(e) = \begin{cases} 0 & \text{if } e = \triangleright \\ \mid \mathrm{sbt}(e_{l'}) \mid + \,\mathrm{snm}(e_u) & \text{if } e = e_u[\hookrightarrow e_p[e_l; \triangleright; e_{l'}\,]\,] \\ \mathrm{snm}(e_u) & \text{if } e = e_u[\hookleftarrow e_p[e_l\,]\,], \end{cases}$$

where $\mathrm{sbt}(e)$ is a function that gives the set of subterms of a list of terms.

Remark 3.15 *For any rewrite* $\alpha = e_z :: \langle e_{p_1}, e_p, e_v, e_q\rangle \stackrel{(3-7)^*}{\longrightarrow} e_z :: \langle e_{p_2}, e_{p'}, e_{v'}, e_q\rangle$, *note that* $\mathrm{snm}(e_{p_1}) \geq \mathrm{snm}(e_{p_2})$. *In particular, if the rewrite* α *contains at least one application of the rule* 3, *then* $\mathrm{snm}(e_{p_1}) > \mathrm{snm}(e_{p_2})$.

Lemma 3.16 *For any rewrite* $\alpha =$

$$e_z :: \langle e_u[\hookrightarrow e_{p_1}\,], e_u[\hookrightarrow e_p\,], e_v, e_q\rangle \stackrel{(3-7)^*}{\longrightarrow} e_z :: \langle e_u[\hookleftarrow e_{p'}\,], e_u[\hookleftarrow e_{p'}\,], e_{v'}\ ,\ e_q\rangle$$

there does not exist any rewrite $\alpha' =$

$$e_z :: \langle e_u[\hookleftarrow e_{p'}\,], e_u[\hookleftarrow e_{p'}\,], e_{v'}\ ,\ e_q\rangle \stackrel{(3-7)^*}{\longrightarrow} e_z :: \langle e_{u'}[\hookleftarrow e_{p''}\,], e_{u'}[\hookleftarrow e_{p''}\,], e_{v''}, e_q\rangle,$$

such that $e_{p'} \neq \mathrm{e}_{p''}$, *and* $e_u = e_{u'}$.

Proof. Assume a rewrite α' such that $e_{p'} \neq \mathrm{e}_{p''}$. By definition of the rules 3–7, we have to consider two cases,

(1) α' is a composition of s-replacement rewrites with rules 7 and 6 only. It is clear in this case that $e_u \neq e_{u'}$.

(2) α' is a composition of s-replacement rewrites 3–7 which contains at least one application of the rule 3. Then, $\mathrm{snm}(e_u[\hookleftarrow e_{p'}\,]) > \mathrm{snm}(e_{u'}[\hookleftarrow e_{p''}\,])$ by Remark 3.15. Therefore, $e_u \neq e_{u'}$.

□

Lemma 3.17 *Given a theory* $T = (\Sigma, \emptyset, R)$ *in* $\mathcal{C}$, $t \in T_\Sigma(X)$, $t' \in T_\Sigma$, *then, for any set of assignments* A, *and any set of variables* V, *such that* $\mathrm{var}(A) \cap V = \emptyset$, *if there is a rewrite* $\alpha =$

$$e_z :: \langle e_u[\hookrightarrow \triangleright(\overline{t})\,]\ ,\ e_u[\hookrightarrow \triangleright(\overline{t'})\,]\ ,\ \overline{A}/\overline{V}\ ,\ e_q\rangle \longrightarrow$$
$$e_z :: \langle e_u[\hookleftarrow e\,]\ ,\ e_u[\hookleftarrow e\,]\ ,\ \overline{A'}/\overline{V'}\ ,\ e_q\rangle,$$

then $\underline{e} = t' = t(A')$ *and* $\mathrm{var}(A') \cap V' = \emptyset$.

Proof. By structural induction on t.

(1) $t = c$, for $c \in \Sigma_0$, $t' = c(\vec{w}/\vec{x})$. Then, by Lemma 3.16, and the definition of rules 3–7, any rewrite α will be a s-replacement rewrite with rule 6,

$$e_z :: \langle e_u[\hookrightarrow \triangleright(\overline{c})]\ ,\ e_u[\hookrightarrow \triangleright(\overline{c})]\ ,\ \overline{A}/\overline{V}\ ,\ e_q\rangle \xrightarrow{6}$$
$$e_z :: \langle e_u[\hookleftarrow \overline{c}]\ ,\ e_u[\hookleftarrow \overline{c}]\ ,\ \overline{A}/\overline{V}\ ,\ e_q\rangle,$$

with $c = c(A)$.

(2) $t = x_i$. We have to consider two subcases:

(a) $x_i \in V$. By Lemma 3.16, and the definition of rules 3–7, any rewrite α will be a s-replacement rewrite with rule 4,

$$e_z :: \langle e_u[\hookrightarrow \triangleright(\overline{x_i})]\ ,\ e_u[\hookrightarrow \triangleright(\overline{w_i})]\ ,\ \overline{A}/\overline{V}\ ,\ e_q\rangle \xrightarrow{4}$$
$$e_z :: \langle e_u[\hookleftarrow \overline{w_i}]\ ,\ e_u[\hookleftarrow \overline{w_i}]\ ,\ (\overline{A \cup \{w_i/x_i\}})/(\overline{V - \{x_i\}})\ ,\ e_q\rangle,$$

with $w_i = x_i(A \cup \{w_i/x_i\})$ and $\mathrm{var}(A \cup \{w_i/x_i\}) \cap (V - \{x_i\}) = \emptyset$.

(b) $x_i \in \mathrm{var}(A)$. By Lemma 3.16, and the definition of rules 3–7, any rewrite will be a s-replacement rewrite with rule 5,

$$e_z :: \langle e_u[\hookrightarrow \triangleright(\overline{x_i})]\ ,\ e_u[\hookrightarrow \triangleright(\overline{w_i})]\ ,\ (\ldots;\overline{w_i}/\overline{x_i};\ldots)/\overline{V}\ ,\ e_q\rangle \xrightarrow{5}$$
$$e_z :: \langle e_u[\hookleftarrow \overline{w_i}]\ ,\ e_u[\hookleftarrow \overline{w_i}]\ ,\ (\ldots;\overline{w_i}/\overline{x_i};\ldots)/\overline{V}\ ,\ e_q\rangle,$$

with $w_i = x_i(\ldots, w_i/x_i, \ldots)$.

(3) $t = f(t_1, \ldots, t_n)$, $f \in \Sigma_n$, $n > 0$, $t' = f(t'_1, \ldots, t'_n)$. By Lemma 3.16, and the definition of rules 3–7, any rewrite α will be a composition of rewrites whose last step is a s-replacement rewrite with rule 6,

$$e_z :: \langle e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\triangleright;\overline{t_1};\ldots;\overline{t_n}]]\ ,\ e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[\triangleright,\overline{t'_1};\ldots;\overline{t'_n}]]\ ,\ \overline{A}/\overline{V}\ ,\ e_q\rangle \xrightarrow{\beta}$$
$$e_z :: \langle e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[e_1;\ldots;e_n;\triangleright]]\ ,\ e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[e_1;\ldots;e_n;\triangleright]]\ ,\ \overline{A'}/\overline{V'}\ ,\ e_q\rangle \xrightarrow{6}$$
$$e_z :: \langle e_u[\hookleftarrow \mathrm{op}\{\hat{f}\}[e_1;\ldots;e_n]]\ ,\ e_u[\hookleftarrow \mathrm{op}\{\hat{f}\}[e_1;\ldots;e_n]]\ ,\ \overline{A'}/\overline{V'}\ ,\ e_q\rangle.$$

Moreover, the rewrite β will be a composition of n rewrites of the following form,

$$\begin{aligned}
e_z :: \langle\ & e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[e_1;\ldots;e_{i-1};\triangleright;\overline{t_i};\ldots;\overline{t_n}]]\ ,\\
& e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[e_1;\ldots;e_{i-1};\triangleright;\overline{t'_i};\ldots;\overline{t'_n}]]\ ,\ \overline{A_i}/\overline{V_i}\ ,\ e_q\rangle \xrightarrow{3}\\
e_z :: \langle\ & e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[e_1;\ldots;e_{i-1};\triangleright;\ldots;\overline{t_n}]][\hookrightarrow \triangleright(\overline{t_i})]\ ,\\
& e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[e_1;\ldots;e_{i-1};\triangleright;\ldots;\overline{t'_n}]][\hookrightarrow \triangleright(\overline{t'_i})]\ ,\ \overline{A_i}/\overline{V_i}\ ,\ e_q\rangle \xrightarrow{\beta_i}\\
e_z :: \langle\ & e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[e_1;\ldots;e_{i-1};\triangleright;\ldots;\overline{t_n}]][\hookleftarrow e_i]\ ,\\
& e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[e_1;\ldots;e_{i-1};\triangleright;\ldots;\overline{t'_n}]][\hookleftarrow e_i]\ ,\ \overline{A'_i}/\overline{V'_i}\ ,\ e_q\rangle \xrightarrow{7}\\
e_z :: \langle\ & e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[e_1;\ldots;e_{i-1};e_i;\triangleright;\ldots;\overline{t_n}]]\ ,\\
& e_u[\hookrightarrow \mathrm{op}\{\hat{f}\}[e_1;\ldots;e_{i-1};e_i;\triangleright;\ldots;\overline{t'_n}]]\ ,\ \overline{A'_i}/\overline{V'_i}\ ,\ e_q\rangle.
\end{aligned}$$

Note that, by induction hypothesis, for any rewrite β_i, $1 \leq i \leq n$, $\underline{e_i} = t'_i = t_i(A'_i)$ and $\mathrm{var}(A') \cap V' = \emptyset$. Also, by Lemma 3.8,

$$A = A_1 \subseteq A'_1 = A_2 \subseteq \ldots \subseteq A'_n = A'.$$

Therefore, $\underline{\mathrm{op}\{\hat{f}\}[e_1;\ldots;e_n]} = f(t'_1, \ldots, t'_n) = f(t_1, \ldots, t_n)(A')$, and $\mathrm{var}(A') \cap V' = \emptyset$.

□

Lemma 3.18 *Given a theory $T = (\Sigma, \emptyset, R)$ in $\mathcal{C}$, $t \in T_\Sigma(X)$, $t' \in T_\Sigma$, if there is a rewrite $\alpha =$*

$$e_z :: \langle e_u[\hookrightarrow \triangleright(\overline{t'})\,]\ ,\ \overline{A}\rangle \stackrel{(9-12)^*}{\longrightarrow} e_z :: \langle e_u[\hookleftarrow e\,]\ ,\ \overline{A}\rangle,$$

then $\underline{e'} = t'(A)$.

Proof. This proof is analogous to the proof of Lemma 3.17. $\square$

Theorem 3.19 *For any theory* $T \in \mathcal{C}$, $t, t' \in T_\Sigma$,

$$U \vdash \overline{T}@*[\to \overline{t}\,] \longrightarrow \overline{T}@*[\to \overline{t'}\,] \quad \Longrightarrow \quad T \vdash t \longrightarrow t'.$$

Proof. For $t = t'$ the result is obvious. For $t \neq t'$, by Lemma 3.5 and Lemma 3.13, $U \vdash \overline{T}@*[\to \overline{t}\,] \longrightarrow \overline{T}@*[\to \overline{t'}\,]$ can be obtained by composition of n paths of the following form

$$\begin{array}{l}
\langle \overline{V}\ ,\ e_r; (\overline{l} \Rightarrow \overline{r}); e_{r'}\rangle @ *[\to e_1\,] \stackrel{1^*}{\longrightarrow} \\
\langle \overline{V}\ ,\ e_r; (\overline{l} \Rightarrow \overline{r}); e_{r'}\rangle @ e_{u'}[\to e'\,] \stackrel{2}{\longrightarrow} \\
z :: \langle \triangleright[\hookrightarrow \triangleright(\overline{l})\,]\ ,\ \triangleright[\hookrightarrow \triangleright(e')\,]\ ,\ \cdot/\overline{V}\ ,\ \overline{r}\rangle \stackrel{(3-7)^*}{\longrightarrow} \\
z :: \langle \triangleright[\hookleftarrow e_m\,]\ ,\ \triangleright[\hookleftarrow e_m\,]\ ,\ \overline{A}/\overline{V'}\ ,\ \overline{r}\rangle \stackrel{8}{\longrightarrow} \\
z :: \langle \triangleright[\hookrightarrow \triangleright\overline{r}\,]\ ,\ \overline{A}\rangle \stackrel{(9-12)^*}{\longrightarrow} \\
z :: \langle \triangleright[\hookleftarrow e''\,]\ ,\ \overline{A}\rangle \stackrel{13}{\longrightarrow} \\
\langle \overline{V}\ ,\ e_r; (\overline{l} \Rightarrow \overline{r}); e_{r'}\rangle @ e_{u'}[\to e''\,] \stackrel{1^*}{\longrightarrow} \\
\langle \overline{V}\ ,\ e_r; (\overline{l} \Rightarrow \overline{r}); e_{r'}\rangle \ddot{@} *[\to e_2\,]
\end{array}$$

where $z = \langle \overline{V}\ ,\ e_r; (\overline{l} \Rightarrow \overline{r}); e_{r'}\rangle \ddot{@} e_{u'}[\to e'\,]$.

Note that for each one of these paths we have the following: assuming that $\underline{e_1} = t_i \in T_\Sigma$, then by Lemma 3.12 $\underline{e'} = t_{i_1} \in T_\Sigma$. Thus, by Lemma 3.17, $t_{i_1} = l(A')$, and by Lemma 3.18, $\underline{e''} = r(A) = t_{i_2} \in T_\Sigma$. Note then that $T \vdash t_{i_1} \longrightarrow t_{i_2}$ is a s-replacement rewrite in T with rule $l \longrightarrow r$, and therefore, by Lemma 3.14, $T \vdash \underline{e_{u'}[\to \overline{t_{i_1}}\,]} \longrightarrow \underline{e_{u'}[\to \overline{t_{i_2}}\,]}$ also holds. Finally, note that by Lemma 3.12, $\underline{e_{u'}[\to \overline{t_{i_1}}\,]} = \underline{*[\to \overline{t_i}\,]}$ and $\underline{e_{u'}[\to \overline{t_{i_2}}\,]} = \underline{*[\to \overline{t'_i}\,]}$, for $t'_i \in T_\Sigma$. Therefore, applying transitivity $n-1$ times in T we obtain the desired rewrite. $\square$

Therefore, Theorem 3.2 is proved by Theorem 3.11 and Theorem 3.19. $\square$

4 Strategies in General Logics

A metacircular interpreter may have a fixed strategy, and such a strategy may remain at the metalevel. This will make such an interpreter less flexible, and will complicate formal reasoning about its correctness. Even if quite flexible strategies can be defined, say in a theorem-prover's tactic language, such a language may remain a programming language external to the logic that it is controlling. In such a situation, "control" becomes an extralogical feature of the system.

If strategies can be defined *inside* the logic that they control, we are in a much better situation, since formal reasoning within the system can be applied to the strategies themselves to prove important properties about them. Reflective logics offer good opportunities for defining internal strategy languages of this kind, because the metalevel being controlled can be expressed within the logic.

We give here general axioms for a strategy language, and for one internal to a logic. Again, we cover the case of entailment systems. Similar axioms for strategy languages at the proof calculus level can be found in [6].

Given a logical theory T, a strategy is a computational way of looking for certain theorems of T. This may be done by having a strategy language $\mathcal{S}(T)$ associated to T. Generally, we can think abstractly of the strategy language $\mathcal{S}(T)$ as a computational system in which strategy expressions can be further and further evaluated—in some cases perhaps forever, and sometimes in highly nondeterministic ways—so that the more we evaluate one such expression the more theorems will be *exhibited* in further stages. We can naturally represent such evaluations from a strategy expression E to another E' by labelled transitions $\alpha : E \longrightarrow E'$. If we denote by $\theta(E)$ the set of theorems exhibited by E, then our requirement is that if there is a computation $\alpha : E \longrightarrow E'$, then $\theta(E) \subseteq \theta(E')$. Of course, transitions $\alpha : E \longrightarrow E'$ and $\beta : E' \longrightarrow E''$ should be composable, to yield $\alpha;\beta : E \longrightarrow E''$, and it is always possible to add idle transitions $E : E \longrightarrow E$. Therefore, we can axiomatize the computations of the strategy language as a category $\mathcal{S}(T)$.

Definition 4.1 Given an entailment system $\mathcal{E}$, an *external strategy language* for it is a functor $\mathcal{S} : \mathbf{Th}_1 \longrightarrow \mathbf{Cat}$, where $\mathbf{Th}_1$ is a subcategory of $\mathbf{Th}_0$ together with a natural transformation $\theta : \mathcal{S} \Rightarrow \mathcal{P} \circ \mathit{thm}$, where $\mathcal{P} : \mathbf{Set} \longrightarrow \mathbf{Cat}$ is the functor sending each set X to the poset $\mathcal{P}(X)$ of subsets of X, viewed as a category with arrows the subset inclusions. In addition, the strategy language is required to be *complete* in the sense that for each theorem $\varphi \in \mathit{thm}(T)$ there is an $E \in \mathcal{S}(T)$ such that $p \in \theta(E)$.

An *internal strategy language* for $\mathcal{E}$ consists of:

- an endofunctor $\mathcal{S} : \mathbf{Th}_1 \longrightarrow \mathbf{Th}_1$—that should be thought of as associating to each theory T a "local metatheory" $\mathcal{S}(T)$, axiomatizing strategies for T inside the logic;
- a functor $D : \mathbf{Th}_1 \longrightarrow \mathbf{Cat}$ assigning to each theory T a "category of deductions" $D(T)$;
- a natural transformation $\theta : D \circ \mathcal{S} \Rightarrow \mathcal{P} \circ \mathit{thm}$ such that $(D \circ \mathcal{S}, \theta)$ is an external strategy language for $\mathcal{E}$. $\Box$

$$\begin{tikzcd} \mathbf{Th_1} \arrow[r, "\mathcal{S}"] \arrow[d, "\mathit{thm}"'] \arrow[dr, phantom, "\Downarrow\theta"] & \mathbf{Th_1} \arrow[d, "D"] \\ \mathbf{Set} \arrow[r, "\mathcal{P}"'] & \mathbf{Cat} \end{tikzcd}$$

We say that a strategy expression $E \in \mathcal{S}(T)$ is *Church-Rosser* iff whenever we have transitions $\alpha : E \longrightarrow E'$ and $\beta : E \longrightarrow E''$ there is always an E''' and transitions $\alpha' : E' \longrightarrow E'''$ and $\beta' : E'' \longrightarrow E'''$. We say that a strategy expression E is *sequential* iff there is a (finite or countable) sequence of nonidle transitions $\alpha_n : E_n \longrightarrow E_{n+1}$ with $E_0 = E$, such that for each nonidle transition $\alpha : E \longrightarrow E'$ there exists a unique k such that $\alpha = \alpha_0; \ldots ; \alpha_k$. Each sequential strategy expression is obviously Church-Rosser, but the contrary is not true in general. Strategy expressions need not be Church-Rosser. For example, a strategy language can have a nondeterministic choice operator $\_ \oplus \_$ so that an expression $E \oplus E'$ has transitions $\alpha : E \oplus E' \longrightarrow E$, and $\beta : E \oplus E' \longrightarrow E'$. Such a nondeterministic choice operator should then be "opaque," so that no proofs are exhibited until it has disappeared, that is, $\theta(E \oplus E') = \emptyset$.

If a reflective entailment system has an internal strategy language, then the strategies $\mathcal{S}(U)$ for the universal theory are particularly important, since they represent at the object level strategies for computing in the universal theory. A *metacircular interpreter* for such a language can then be regarded as the implementation of a particular strategy in $\mathcal{S}(U)$. In general, it is easier to define internal strategy languages when a logic is reflective; and such languages can then be very expressive and powerful, not only for U, but also for all other theories. This is indeed the case for rewriting logic, as we discuss in the next section.

The usefulness of internal strategies is of course very general. For example, in the context of typed lambda calculi, the important advantages of having an internal strategy language has been stressed by several authors. Thus, using reflective capabilities both tactics and decision pocedures can be specified, reasoned about, and executed inside the Nuprl constructive type theory [1,7]. Similarly, Rueß[20] discusses in detail an elegant approach for endowing the calculus of constructions with internal strategies, as part of his treatment of reflection for such a calculus.

5 Strategies in Rewriting Logic

Since theories in rewriting logic need not to be Church-Rosser and do not have to terminate, the issue of strategies for controlling the rewriting process is of outmost importance. This has been addressed in ELAN [14], where what in our terminology amounts to an external strategy language is given for this purpose. More recently, the definition of strategies by rewriting in the context of ELAN has also been proposed in [2]. In this section we apply to rewriting logic the general semantic axioms for internal strategy language presented in Section 4. We give a general method for defining and proving correct a strategy language, and show how the correctness of a universal theory greatly simplifies the proof of correctness of a given strategy language. Since we have proved rewriting logic reflective for the class $\mathcal{C}$ of unconditional and unsorted finitely presentable rewrite theories, we will define an internal strategy language for controlling the rewriting inference process of theories in $\mathcal{C}$.

As defined in Section 4 an internal strategy language is given by an endofunctor $\mathcal{S} : \mathbf{Th_1} \longrightarrow \mathbf{Th_1}$, satisfying additional semantic requirement; for our purposes $\mathbf{Th_1}$ will be the category of finitely presentable theories in rewriting logic but with morphisms restricted to identities. This endofuctor $\mathcal{S}$ associates to each rewrite theory T a "local metatheory" $\mathcal{S}(\mathcal{T})$, axiomatizing strategies for T in the logic.

Our idea is to use the reflective capabilities of rewriting logic, and in particular, the existence of a universal theory U, to define a subfunctor $\mathcal{K} \hookrightarrow \mathcal{S}$, what we call the *kernel* of the internal strategy language, whose correctness is based on the correctness of U itself as a universal theory. Then, a wide variety of endofunctors $\mathcal{S}$ can extend such a kernel axiomatizing additional strategies, but their correctness can be reduced to that of the kernel.

For example a typical kernel $\mathcal{K}$ can be defined as a function $\mathcal{K}$ which takes a rewrite theory $T = (\Sigma, \emptyset, L, R)$ [3] and returns a theory $\mathcal{K}(\mathcal{T})$, with

$\diamond\Sigma_{\mathcal{K}(\mathcal{T})} = \{\Sigma_U \cup \{\mathrm{metapply}(\_,\_), \mathrm{rew\_} \Rightarrow \mathrm{\_with\_}, \mathrm{apply}(\_), \mathrm{failure}, \mathrm{idle}\}\}$,

$\diamond E_{\mathcal{K}(\mathcal{T})} = \{E_U \cup \{\mathrm{def_l}\}_{\mathrm{l} \in \mathrm{L}}\}\}$,
where if $l : t(\vec{x}) \longrightarrow t'(\vec{x})$ in R, then $\mathrm{def_l}$ is the equation,
$\mathrm{metapply}(\overline{\mathrm{t_1}}, \overline{\mathrm{l}}) = \langle \cdot\ ,\ \cdot \rangle \ddot{@} * [\rightarrow \overline{\mathrm{t_1}}\ ] :: \langle \triangleright[\hookrightarrow \triangleright(\overline{\mathrm{t}})]\ ,\ \triangleright[\hookrightarrow \triangleright(\overline{\mathrm{t_1}})]\ ,\ \cdot/\overline{\vec{\mathrm{x}}}\ ,\ \overline{\mathrm{t'}}\rangle$,

$\diamond R_{\mathcal{K}(\mathcal{T})} =$
$\{\mathrm{rew}\ \overline{\mathrm{t}} \Rightarrow \overline{\mathrm{t'}}\ \mathrm{with\ apply(l)} \longrightarrow \mathrm{rew}\ \overline{\mathrm{t}} \Rightarrow \overline{\mathrm{t''}}\ \mathrm{with\ idle}$
$\qquad \mathrm{if\ metapply}(\overline{\mathrm{t'}}, \mathrm{l}) \longrightarrow \langle \cdot\ ,\ \cdot \rangle \ddot{@} * [\rightarrow \overline{\mathrm{t''}}\ ]$,
$\mathrm{rew}\ \overline{\mathrm{t}} \Rightarrow \overline{\mathrm{t'}}\ \mathrm{with\ apply(l)} \longrightarrow \mathrm{failure}$
$\mathrm{if\ metapply}(\overline{\mathrm{t'}}, \mathrm{l}) \longrightarrow \langle \cdot\ ,\ \cdot \rangle \ddot{@} :: \langle \mathrm{u}[\hookrightarrow \mathrm{p[l]}]\ ,\ \mathrm{u'}[\hookrightarrow \mathrm{q[l']}]\ ,\ \mathrm{a/v}\ ,\ \overline{\mathrm{t'}}\rangle \mathrm{\ and\ p} \neq \mathrm{q}\}$.

In this case it is trivial to define the natural transformation θ that extracts from a rewriting strategy expression, the rewrites that it was supposed to describe

$$\theta(\mathrm{rew}\ \overline{\mathrm{t}} \Rightarrow \overline{\mathrm{t'}}\ \mathrm{with\ idle}) = \{\overline{\mathrm{t}} \Rightarrow \overline{\mathrm{t'}}\}.$$

Note that the correctness of this kernel is based on the correctness of U as a universal theory. In particular, the correctness of the apply(_) strategy depends on the correctness of metapply(_, _) in representing the replacement rule in rewriting logic. But this is a result that can be easily obtained as a corollary of Theorem 3.19.

Now we can define $\mathcal{S}$ as extending $\mathcal{K}$, in the sense that it axiomatizes additional strategies, but always based on the kernel defined by $\mathcal{K}$. Consider for example the function $\mathcal{S}$ which takes a rewrite theory $T = (\Sigma, \emptyset, L, R)$ and returns a new theory $\mathcal{S}(\mathcal{T})$, with

$\diamond\Sigma_{\mathcal{S}(\mathcal{T})} = \{\Sigma_{\mathcal{K}(\mathcal{T})} \cup \{\_;\_, \mathrm{\_andthen\_}\ \}\}$,

$\diamond E_{\mathcal{S}(\mathcal{T})} = E_{\mathcal{K}(\mathcal{T})}$,

$\diamond R_{\mathcal{S}(\mathcal{T})} =$
$\{R_{\mathcal{K}(\mathcal{T})} \cup$
$\{\mathrm{rew}\ \overline{\mathrm{t}} \Rightarrow \overline{\mathrm{t'}}\ \mathrm{with\ (S\ ;S')} \longrightarrow (\mathrm{rew}\ \overline{\mathrm{t}} \Rightarrow \overline{\mathrm{t'}}\ \mathrm{with\ S})\ \mathrm{andthen\ S'}$,
$\mathrm{failure\ andthen\ S} \longrightarrow \mathrm{failure}$,

[3] As in Section 3.3—to simplify the exposition— we asume that the equations have transformed into bidirectional rules. Without loss of generality we also assume that labels are string of ASCII characters.

$(\mathrm{rew}\ \overline{\mathrm{t}} \ \Rightarrow\ \overline{\mathrm{t}'}\ \mathrm{with\ idle})\ \mathrm{andthen\ S} \longrightarrow \mathrm{rew}\ \overline{\mathrm{t}} \ \Rightarrow\ \overline{\mathrm{t}'}\ \mathrm{with\ S}\ \}\}$

Again, we can define the natural transformation θ as

$$\theta(\mathrm{rew}\ \overline{\mathrm{t}} \ \Rightarrow\ \overline{\mathrm{t}'}\ \mathrm{with\ idle}) = \{\overline{\mathrm{t}} \ \Rightarrow\ \overline{\mathrm{t}'}\}.$$

Note that $\mathcal{S}$ extends $\mathcal{K}$ axiomatizing concatenation of strategies and its correctness is based on the correctness of $\mathcal{K}$. A more developed example of an internal strategy language based on reflection can be found in [5].

6 Concluding Remarks

We have specified a universal theory for rewriting logic and have proved its correctness. We have also introduced the notion of internal strategy language and have given a general method for defining such languages in rewriting logic. In joint work with Steven Eker and Patrick Lincoln we are applying these ideas and techniques in the context of the Maude language. Future developments and applications that we think particularly important include:

- Definition of a universal theory for the variant of rewriting logic whose underlying equational logic is membership equational logic [21,3].
- Applications of rewriting logic to give semantics to other reflective sytems and languages.
- Uses of reflection in logical framework applications of rewriting logic (see [15] for a discussion of this topic).
- Metaprogramming uses of reflection in rewriting logic, including general module composition and transformation operations [5], and special topics such as supercompilation [30].
- Further development of, and experimentation with, internal strategy languages.
- Development of formal environments extending rewriting logic languages.

Acknowledgement

We cordially thank Carolyn Talcott for her very helpful suggestions and her encouragement of this work along many discussions, and Narciso Martí-Oliet who read thorough many drafts of this paper and gave us most valuable comments and suggestions for improving the exposition. Our warm thanks also to our fellow members in the Maude team, Steven Eker and Patrick Lincoln, with whom we have exchanged many ideas about these topics, and with whom we have worked to realize them in the Maude language.

References

[1] William E. Aitken, Robert L. Constable, and Judith L. Underwood. Metalogical frameworks II: Using reflected decision procedures. Technical Report,

Computer Sci. Dept., Cornell University, 1993; also, lecture at the Max Planck Institut für Informatik, Saarbrücken, Germany, July 1993.

[2] P. Borovanský, C. Kirchner, and H. Kirchner. Controlling rewriting by rewriting. To appear in *Proc.* 1^{st} *Intl. Workshop on Rewriting Logic and its Applications*, ENTCS, North Holland, 1996.

[3] Adel Bouhoula, Jean-Pierre Jouannaud, and José Meseguer. Specification and proof in membership equational logic. Manuscript, SRI International, August 1996.

[4] R. S. Boyer and J Strother Moore. Metafunctions: proving them correct and using them efficiently as new proof procedures. In Robert Boyer and J Moore, editors, *The Correctness Problem in Computer Science*, pages 103–185. Academic Press, 1981.

[5] Manuel Clavel, Steven Eker, Patrick Lincoln, and José Meseguer. Principles of Maude. In José Meseguer, editor, *Proceedings of the first international workshop on rewriting logic*, volume 4, Asilomar (California), September 1996. Electronic Notes in Theoretical Computer Science.

[6] Manuel Clavel and José Meseguer. Axiomatizing reflective logics and languages. In Gregor Kiczales, editor, *Proceedings of Reflection'96, San Francisco, California, April 1996*, pages 263–288. Xerox PARC, 1996.

[7] Robert L. Constable. Using reflection to explain and enhance type theory. In Helmut Schwichtenberg, editor, *Proof and Computation*, volume 139 of *Computer and System Sciences*, pages 109–144. Springer, 1995.

[8] Fausto Giunchiglia, Paolo Traverso, Alessandro Cimatti, and Paolo Pecchiari. A system for multi-level reasoning. In *IMSA'92*, pages 190–195. Information-Technology Promotion Agency, Japan, 1992.

[9] John Harrison. Metatheory and reflection in theorem proving: a survey and critique. University of Cambridge Computer Laboratory, 1995.

[10] Patricia Hill and John Lloyd. *The Gödel Programming Language.* MIT Press, 1994.

[11] Douglas J. Howe. Reflecting the semantics of reflected proof. In Peter Aczel, Harold Simmons, and Stanley S. Wainer, editors, *Proof Theory*, pages 229–250. Cambridge University Press, 1990.

[12] Gregor Kiczales, editor. *Proceedings of Reflection'96, San Francisco, California, April 1996.* Xerox PARC, 1996.

[13] Gregor Kiczales, Jim des Rivieres, and Daniel G. Bobrow. *The Art of the Metaobject Protocol.* MIT Press, 1991.

[14] C. Kirchner, H. Kirchner, and M. Vittek. Designing constraint logic programming languages using computational systems. In V. Saraswat and P. van Hentryck, editors, *Principles and Practice of Constraint Programming: The Newport Papers*, pages 133–160. MIT Press, 1995.

[15] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. This volume.

[16] Narciso Martí-Oliet and José Meseguer. General logics and logical frameworks. In D. Gabbay, editor, *What is a Logical System?*, pages 355–392. Oxford University Press, 1994.

[17] Satoshi Matsuoka, Takuo Watanabe, Yuuji Ichisugi, and Akinori Yonezawa. Object-oriented concurrent reflective architectures. In M. Tokoro, O. Nierstrasz, and P. Wegner, editors, *Object-Based Concurrent Computing*, pages 211–226. Springer LNCS 612, 1992.

[18] Seán Matthews. Reflection in logical systems. In *IMSA'92*, pages 178–183. Information-Technology Promotion Agency, Japan, 1992.

[19] François-Nicola Demers and Jacques Malenfant. Reflection in logic, functional and object-oriented programming: a short comparative study. In *IJCAI '95 Workshop on Reflection and Metalevel Architectures and their Applications in AI*, pages 29–38, August 1995.

[20] Harald Rueß. Reflection of formal tactics in a deductive reflection framework. Manuscript, Universität Ulm, Abt. Künstliche Intelligenz, January 96.

[21] José Meseguer. Membership algebra. Lecture at the Dagstuhl Seminar on "Specification and Semantics," July 9, 1996. Extended version in preparation.

[22] José Meseguer. General logics. In H.-D. Ebbinghaus et al., editor, *Logic Colloquium'87*, pages 275–329. North-Holland, 1989.

[23] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[24] Hideaki Okamura, Yutaka Ishikawa, and Mario Tokoro. AL-1/D: A distributed programming system with multi-model reflection framework. In *IMSA'92*, pages 36–47. Information-Technology Promotion Agency, Japan, 1992.

[25] Luis H. Rodriguez, Jr. A study on the viability of a production-quality metaobject protocol-based statically parallelizing compiler. In *IMSA'92*, pages 107–112. Information-Technology Promotion Agency, Japan, 1992.

[26] N. Shankar. *Metamathematics, Machines, and Gödel's Proof.* Cambridge University Press, 1994.

[27] Brian Smith and Akinori Yonezawa, editors. *Proc. of the IMSA'92 International Workshop on Reflection and Meta-Level Architecture, Tokyo, November 1992.* Research Institute of Software Engineering, 1992.

[28] Brian C. Smith. Reflection and Semantics in Lisp. In *Proc. POPL'84*, pages 23–35. ACM, 1984.

[29] G. L. Steele, Jr. and G. J. Sussman. The art of the interpreter or, the modularity complex. Technical Report AIM-453, MIT AI-Lab, May 1978.

[30] Valentin F. Turchin. The concept of a supercompiler. *ACM Transactions on Programming Languages and Systems*, 8(3):292–325, 1986.

[31] M. Wand and D.P. Friedman. The mystery of the tower revealed. *Lisp and Symbolic Computation*, 1(1):11–38, 1988.

[32] Richard W. Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, 13:133–170, 1980.