Backward Trace Slicing for Rewriting Logic Theories*

— Technical Report —

M. Alpuente[1], D. Ballis[2], J. Espert[1], and D. Romero[1]

[1] DSIC-ELP, Universidad Politécnica de Valencia
Camino de Vera s/n, Apdo 22012, 46071 Valencia, Spain
{alpuente,jespert,dromero}@dsic.upv.es
[2] Dipartimento di Matematica e Informatica
Via delle Scienze 206, 33100 Udine, Italy demis.ballis@uniud.it

Abstract. Trace slicing is a widely used technique for execution trace analysis that is effectively used in program debugging, analysis and comprehension. In this paper, we present a backward trace slicing technique that can be used for the analysis of Rewriting Logic theories. Our trace slicing technique allows us to systematically trace back rewrite sequences modulo equational axioms (such as associativity and commutativity) by means of an algorithm that dynamically simplifies the traces by detecting control and data dependencies, and dropping useless data that do not influence the final result. Our methodology is particularly suitable for analyzing complex, textually-large system computations such as those delivered as counter-example traces by Maude model-checkers.

1 Introduction

The analysis of execution traces plays a fundamental role in many program manipulation techniques. Trace slicing is a technique for reducing the size of traces by focusing on selected aspects of program execution, which makes it suitable for trace analysis and monitoring [7].

Rewriting Logic (RWL) is a very general *logical* and *semantic framework*, which is particularly suitable for formalizing highly concurrent, complex systems (e.g., biological systems [5, 21] and Web systems [2, 3]). RWL is efficiently implemented in the high-performance system Maude [9]. Roughly speaking, a *rewriting logic theory* seamlessly combines a *term rewriting system* (TRS) together with an *equational theory* that may include sorts, functions, and algebraic laws (such as commutativity and associativity) so that rewrite steps are applied *modulo* the equations. Within this framework, the system states are typically

* This work has been partially supported by the EU (FEDER) and the Spanish MEC TIN2010-21062-C02-02 project, by Generalitat Valenciana PROMETEO2011/052, and by the Italian MUR under grant RBIN04M8S8, FIRB project, Internationalization 2004. Daniel Romero is also supported by FPI–MEC grant BES–2008–004860.

represented as elements of an algebraic data type that is specified by the equational theory, while the system computations are modeled via the rewrite rules, which describe transitions between states.

Due to the many important applications of RWL, in recent years, the debugging and optimization of RWL theories have received growing attention [1, 15, 18, 19]. However, the existing tools provide hardly support for execution trace analysis. The original motivation for our work was to reduce the size of the counterexample traces delivered by Web-TLR, which is a RWL-based model-checking tool for Web applications proposed in [2, 3]. As a matter of fact, the analysis (or even the simple inspection) of such traces may be unfeasible because of the size and complexity of the traces under examination. Typical counterexample traces in Web-TLR are 75 Kb long for a model size of 1.5 Kb, that is, the trace is in a ratio of 5.000% w.r.t. the model.

To the best of our knowledge, this paper presents the first trace slicing technique for RWL theories. The basic idea is to take a trace produced by the RWL engine and traverse and analyze it backwards to filter out events that are irrelevant for the rewritten task. The trace slicing technique that we propose is fully general and can be applied to optimizing any RWL-based tool that manipulates rewrite logic traces. Our technique relies on a suitable mechanism of backward tracing that is formalized by means of a procedure that labels the calls (terms) involved in the rewrite steps. The backward traversal is preferred to a forward one because a causal relation is computed. This allows us to infer, from a term t and positions of interest on it, positions of interest of the term that was rewritten to t. Our labeling procedure extends the technique in [6], which allows descendants and origins to be traced in orthogonal (i.e., left-linear and overlap-free) term rewriting systems in order to deal with rewrite theories that may contain commutativity/associativity axioms, as well as nonleft-linear, collapsing equations and rules.

Plan of the paper. Section 2 summarizes some preliminary definitions and notations about term rewriting systems. In Section 3, we recall the essential notions concerning rewriting modulo equational theories. In Section 4, we formalize our backward trace slicing technique for rewriting logic theories, which computes the reverse dependence among the symbols involved in a rewrite step and removes all data that are irrelevant with respect to a given slicing criterion. Section 5 extends the trace slicing technique of Section 4 by considering extended rewrite theories, i.e., rewrite theories that may include collapsing, nonleft-linear rules, associative/commutative equational axioms, and built-in operators. Section 6 describes a software tool that implements the proposed backward slicing technique and presents an experimental evaluation of the tool that allows us to assess the practical advantages of the trace slicing technique. In Section 7, we discuss some related work, and Section 8 concludes. Proofs of the main technical results can be found in Appendix A. This appendix is only meant to help the referees evaluate the manuscript and is not part of the paper.

2 Preliminaries

A many-sorted signature (Σ, S) consists of a set of sorts S and a $S^* \times S$-indexed family of sets $\Sigma = \{\Sigma_{\bar{s}\times s}\}_{(\bar{s},s)\in S^*\times S}$, which are sets of *function symbols* (or operators) with a given string of argument sorts and result sort. Given an S-sorted set $\mathcal{V} = \{\mathcal{V}_s \mid s \in S\}$ of disjoint sets of variables, $T_\Sigma(\mathcal{V})_s$ and $T_{\Sigma s}$ are the sets of terms and ground terms of sorts s, respectively. We write $T_\Sigma(\mathcal{V})$ and T_Σ for the corresponding term algebras. An *equation* is a pair of terms of the form $s = t$, with $s, t \in T_\Sigma(\mathcal{V})_s$. In order to simplify the presentation, we often disregard sorts when no confusion can arise.

Terms are viewed as labelled trees in the usual way. Positions are represented by sequences of natural numbers denoting an access path in a term. The empty sequence Λ denotes the root position. By $root(t)$, we denote the symbol that occurs at the root position of t. We let $\mathcal{P}os(t)$ denote the set of positions of t. By notation $w_1.w_2$, we denote the concatenation of positions (sequences) w_1 and w_2. Positions are ordered by the prefix ordering, that is, given the positions w_1, w_2, $w_1 \leq w_2$ if there exists a position x such that $w_1.x = w_2$. $t_{|u}$ is the subterm at the position u of t. $t[r]_u$ is the term t with the subterm rooted at the position u replaced by r. Given a term t, we say that t is *ground* if no variables occur in t. A substitution σ is a mapping from variables to terms $\{x_1/t_1, \ldots, x_n/t_n\}$ such that $x_i\sigma = t_i$ for $i = 1, \ldots, n$ (with $x_i \neq x_j$ if $i \neq j$), and $x\sigma = x$ for any other variable x. By ε, we denote the *empty* substitution. Given a substitution σ, the *domain* of σ is the set $Dom(\sigma) = \{x | x\sigma \neq x\}$. By $Var(t)$ (resp. $FSymbols(t)$), we denote the set of variables (resp. function symbols) occurring in the term t.

A *context* is a term $\gamma \in T_{\Sigma\cup\{\Box\}}(\mathcal{V})$ with zero or more holes $\Box$, and $\Box \notin \Sigma$. We write $\gamma[\,]_u$ to denote that there is a hole at position u of γ. By notation $\gamma[\,]$, we define an arbitrary context (where the number and the positions of the holes are clarified *in situ*), while we write $\gamma[t_1, \ldots t_n]$ to denote the term obtained by filling the holes appearing in $\gamma[\,]$ with terms $t_1, \ldots, t_n$. By notation $t^\Box$, we denote the context obtained by applying the substitution $\sigma = \{x_1/\Box, \ldots, x_n/\Box\}$ to t, where $Var(t) = \{x_1 \ldots, x_n\}$ (i.e., $t^\Box = t\sigma$).

A *term rewriting system* (TRS for short) is a pair (Σ, R), where Σ is a signature and R is a finite set of reduction (or rewrite) rules of the form $\lambda \to \rho$, $\lambda, \rho \in T_\Sigma(\mathcal{V})$, $\lambda \notin \mathcal{V}$ and $Var(\rho) \subseteq Var(\lambda)$. We often write just R instead of (Σ, R). A rewrite step is the application of a rewrite rule to an expression. A term s *rewrites* to a term t via $r \in R$, $s \xrightarrow{r}_R t$ (or $s \xrightarrow{r,\sigma}_R t$), if there exists a position q in s such that λ *matches* $s_{|q}$ via a substitution σ (in symbols, $s_{|q} = \lambda\sigma$), and t is obtained from s by replacing the subterm $s_{|q} = \lambda\sigma$ with the term $\rho\sigma$, in symbols $t = s[\rho\sigma]_q$. The rule $\lambda \to \rho$ (or equation $\lambda = \rho$) is *collapsing* if $\rho \in \mathcal{V}$; it is *left-linear* if no variable occurs in λ more than once. We denote the transitive and reflexive closure of $\to$ by $\to^*$.

3 Rewriting Modulo Equational Theories

An *equational theory* is a pair (Σ, E), where Σ is a signature and $E = \Delta \cup B$ consists of a set of (oriented) equations Δ together with a collection B of equational axioms (e.g., associativity and commutativity axioms) that are associated with some operator of Σ. The equational theory E induces a least congruence relation on the term algebra $T_\Sigma(\mathcal{V})$, which is usually denoted by $=_E$.

A *rewrite theory* is a triple $\mathcal{R} = (\Sigma, E, R)$, where (Σ, E) is an equational theory, and R is a TRS. Examples of rewrite theories can be found in [9].

Rewriting modulo equational theories [15] can be defined by lifting the standard rewrite relation $\rightarrow_R$ on terms to the E-congruence classes induced by $=_E$. More precisely, the rewrite relation $\rightarrow_{R/E}$ for rewriting modulo E is defined as $=_E \circ \rightarrow_R \circ =_E$. A computation in $\mathcal{R}$ using $\rightarrow_{R\cup\Delta,B}$ is a *rewriting logic deduction*, in which the *equational simplification* with Δ (i.e., applying the oriented equations in Δ to a term t until a canonical form $t{\downarrow_E}$ is reached where no further equations can be applied) is intermixed with the rewriting computation with the rules of R, using an *algorithm of matching modulo*[3] B in both cases.

Formally, given a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, where $E = \Delta \cup B$, a *rewrite step modulo* E on a term s_0 by means of the rule $r : \lambda \rightarrow \rho \in R$ (in symbols, $s_0 \overset{r}{\rightarrow}_{R\cup\Delta,B} s_1$) can be implemented as follows: (i) apply (modulo B) the equations of Δ on s_0 to reach a canonical form $(s_0 \downarrow_E)$; (ii) rewrite (modulo B) $(s_0 \downarrow_E)$ to term v by using $r \in R$; and (iii), apply (modulo B) the equations of Δ on v again to reach a canonical form for v, $s_1 = v \downarrow_E$.

Since the equations of Δ are implicitly oriented (from left to right), the equational simplification can be seen as a sequence of (equational) rewrite steps $(\rightarrow_{\Delta/B})$. Therefore, a *rewrite step modulo* E $s_0 \overset{r}{\rightarrow}_{R\cup\Delta,B} s_1$ can be expanded into a sequence of rewrite steps as follows:

$$s_0 \overbrace{\rightarrow_{\Delta/B} .. \rightarrow_{\Delta/B} s_0{\downarrow_E}}^{\text{equational simplification}} \overbrace{=_B u \overset{r}{\rightarrow}_R v}^{\text{rewrite step}/_B} \overbrace{\rightarrow_{\Delta/B} .. \rightarrow_{\Delta/B} v{\downarrow_E}}^{\text{equational simplification}} = s_1$$

Given a finite rewrite sequence $\mathcal{S} = s_0 \rightarrow_{R\cup\Delta,B} s_1 \rightarrow_{R\cup\Delta,B} \ldots \rightarrow s_n$ in the rewrite theory $\mathcal{R}$, the *execution trace* of $\mathcal{S}$ is the rewrite sequence $\mathcal{T}$ obtained by expanding all the rewrite steps $s_i \rightarrow_{R\cup\Delta,B} s_{i+1}$ of $\mathcal{S}$ as is described above.

The computability of $\rightarrow_{R\cup\Delta,B}$ as well as its equivalence w.r.t. $\rightarrow_{R/E}$ are assured by enforcing some conditions on the considered rewrite theories [15, 23], specifically, *coherence* between the rules and the equations as well as the assumption of *Church-Rosser* and *termination* properties of Δ modulo the equational axioms B[4].

[3] A subterm of t matches l (*modulo* B) via the substitution σ if $t =_B u$ and $u_{|q} = l\sigma$ for a position q of u.

[4] These conditions are quite natural in practical rewriting logic specifications, and can generally be checked by using the Maude Church-Rosser, Termination, and Coherence tools [9].

A rewrite theory $\mathcal{R} = (\Sigma, B \cup \Delta, R)$ is called *elementary* if $\mathcal{R}$ does not contain equational axioms ($B = \emptyset$) and both rules and equations are left-linear and not collapsing.

4 Backward Trace Slicing for Elementary Rewrite Theories

In this section, we formalize a backward trace slicing technique for *elementary rewrite theories* that is based on a term labeling procedure that is inspired by [6]. Since equations in Δ are treated as rewrite rules that are used to simplify terms, our formulation for the trace slicing technique is purely based on standard rewriting. In Section 5, we will drop all these restrictions in order to consider more expressive rewrite theories.

4.1 Labeling procedure for rewrite theories

Let us define a labeling procedure for rules similar to [6] that allows us to trace symbols involved in a rewrite step. First, we provide the notion of labeling for terms, and then we show how it can be naturally lifted to rules and rewrite steps.

Consider a set $\mathcal{A}$ of *atomic labels*, which are denoted by Greek letters $\alpha, \beta, \ldots$. *Composite labels* (or simply *labels*) are defined as finite sets of elements of $\mathcal{A}$. By abuse, we write the label $\alpha\beta\gamma$ as a compact denotation for the set $\{\alpha, \beta, \gamma\}$.

A *labeling* for a term $t \in T_{\Sigma \cup \{\Box\}}(\mathcal{V})$ is a map L that assigns a label to (the symbol occurring at) each position w of t, provided that $root(t_{|w}) \neq \Box$. If t is a term, then t^L denotes the labeled version of t. Note that, in the case when t is a context, occurrences of symbol $\Box$ appearing in the labeled version of t are not labeled. The *codomain* of a labeling L is denoted by $Cod(L) = \{l \mid (w \mapsto l) \in L\}$.

An *initial labeling* for the term t is a labeling for t which assigns distinct fresh atomic labels to each position of the term. For example, given $t = f(g(a, a), \Box)$, then $t^L = f^{\alpha}(g^{\beta}(a^{\gamma}, a^{\delta}), \Box)$ is the labeled version of t via the initial labeling $L = \{\Lambda \mapsto \alpha, 1 \mapsto \beta, 1.1 \mapsto \gamma, 1.2 \mapsto \delta\}$. This notion extends to rules and rewrite steps in a natural way as shown below.

Labeling of Rules. Let us introduce the notions of *redex pattern* and *contractum pattern* of a rule. Let $r : \lambda \to \rho$ be a rule. We call the context $\lambda^{\Box}$ (resp. $\rho^{\Box}$) *redex pattern* (resp. *contractum pattern*) of r.

Example 1. Given the rule $r : f(g(x, y), a)) \to d(s(y), y)$, where a is a constant symbol, the redex pattern of r is the context $f(g(\Box, \Box), a)$, while the contractum pattern of r is the context $d(s(\Box), \Box)$.

Definition 1. *(rule labeling) [6] Given a rule $r : \lambda \to \rho$, a labeling L_r for r is defined by means of the following procedure.*

r_1. *The redex pattern $\lambda^{\Box}$ is labeled by means of an initial labeling L.*

r_2. *A new label l is formed by joining all the labels that occur in the labeled redex pattern $\lambda^{\square}$ (say in alphabetical order) of the rule r. Label l is then associated with each position w of the contractum pattern $\rho^{\square}$, provided that $root(\rho^{\square}_{|w}) \neq \square$.*

Example 2. Consider the rule r of Example 1. The labeled version of rule r using the initial labeling $L = \{(\Lambda \mapsto \alpha, 1 \mapsto \beta, 2 \mapsto \gamma\}$ is as follows: $f^{\alpha}(g^{\beta}(x, y), a^{\gamma}) \rightarrow d^{\alpha\beta\gamma}(s^{\alpha\beta\gamma}(y), y)$.

The labeled version of r w.r.t. L_r is denoted by r^{L_r}. Note that the labeling procedure shown in Definition 1 does not assign labels to variables but only to the function symbols occurring in the rule.

Labeling of Rewrite Steps. Before giving the definition of labeling for a rewrite step, we need to formalize the auxiliary notion of substitution labeling.

Definition 2. *(substitution labeling) Let $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$ be a substitution. A labeling L_σ for the substitution σ is defined by a set of initial labelings $L_\sigma = \{L_{x_1/t_1}, \ldots, L_{x_n/t_n}\}$ such that (i) for each binding (x_i/t_i) in the substitution σ, t_i is labeled using the corresponding initial labeling L_{x_i/t_i}, and (ii) the sets $Cod(L_{x_1/t_1}), \ldots, Cod(L_{x_n/t_n})$ are pairwise disjoint.*

By using Definition 2, we can formulate a labeling procedure for rewrite steps as follows.

Definition 3. *(rewrite step labeling) Let $r : \lambda \rightarrow \rho$ be a rule, and $\mu : t \stackrel{r,\sigma}{\rightarrow} s$ be a rewrite step using r such that $t = C[\lambda\sigma]_q$ and $s = C[\rho\sigma]_q$, for a context C and position q. Let $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$. Let L_r be a labeling for the rule r, L_C be an initial labeling for the context C, and $L_\sigma = \{L_{x_1/t_1}, \ldots, L_{x_n/t_n}\}$ be a labeling for the substitution σ such that the sets $Cod(L_C)$, $Cod(L_r)$, and $Cod(\sigma)$ are pairwise disjoint, where $Cod(\sigma) = \bigcup_{i=1}^{n} Cod(L_{x_i/t_i})$.*

The rewrite step *labeling L_μ for μ is defined by successively applying the following steps:*

s_1. *First, positions of t or s that belong to the context C are labeled by using the initial labeling L_C.*

s_2. *Then positions of $t_{|q}$ (resp. $s_{|q}$) that correspond to the redex pattern (resp. contractum pattern) of the rule r rooted at the position q are labeled according to the labeling L_r.*

s_3. *Finally, for each term t_j, $j = \{1, \ldots, n\}$, which has been introduced in t or s via the binding $x_j/t_j \in \sigma$, with $x_j \in Var(\lambda)$, t_j is labeled using the corresponding labeling $L_{x_j/t_j} \in L_\sigma$*

The labeled version of a rewrite step μ w.r.t. L_μ is denoted by μ^{L_μ}. Let us illustrate it by means of a rather intuitive example.

Example 3. Consider again the rule $r : f(g(x,y),a)) \to d(s(y),y)$ of Example 1, and let $\mu : C[\lambda\sigma] \xrightarrow{r} C[\rho\sigma]$ be a rewrite step using r, where $C[\lambda\sigma] = d(f(g(a,h(b)),a),a)$, $C[\rho\sigma] = d(d(s(h(b)),h(b)),a)$, and $\sigma = \{x/a, y/h(b)\}$.

Assume that r is labeled by means of the rule labeling of Example 2, that is

$$r^L : f^{\alpha}(g^{\beta}(x,y),a^{\gamma}) \to d^{\alpha\beta\gamma}(s^{\alpha\beta\gamma}(y),y)$$

Let $L_C = \{\Lambda \mapsto \delta,\ 2 \mapsto \epsilon\}$, $L_{x/a} = \{\Lambda \mapsto \zeta\}$, and $L_{y/h(b)} = \{\Lambda \mapsto \eta, 1 \mapsto \theta\}$ be the labelings for C and the bindings in σ, respectively. Then, the corresponding labeled rewrite step μ^L is as follows

$$\mu^{\mathtt{L}} : \mathtt{d}^{\delta}(\mathtt{f}^{\alpha}(\mathtt{g}^{\beta}(\mathtt{a}^{\zeta},\mathtt{h}^{\eta}(\mathtt{b}^{\theta})),\mathtt{a}^{\gamma}),\mathtt{a}^{\epsilon}) \to \mathtt{d}^{\delta}(\mathtt{d}^{\alpha\beta\gamma}(\mathtt{s}^{\alpha\beta\gamma}(\mathtt{h}^{\eta}(\mathtt{b}^{\theta})),\mathtt{h}^{\eta}(\mathtt{b}^{\theta})),\mathtt{a}^{\epsilon})$$

Now, we are ready to define our labeling-based, *backward tracing relation* on rewrite steps.

Definition 4. *(origin positions) Let $\mu : t \xrightarrow{r} s$ be a rewrite step and L be a labeling for μ where L_t (resp. L_s) is the labeling of t (resp. s). Given a position w of s, the set of origin positions of w in t w.r.t. μ and L (in symbols, $\lhd^L_\mu w$) is defined as follows:*

$$\lhd^L_\mu w = \{v \in \mathcal{P}os(t) \mid \exists p \in \mathcal{P}os(s), (v \mapsto l_v) \in L_t, (p \mapsto l_p) \in L_s \text{ s.t. } p \leq w \text{ and } l_v \subseteq l_p\}$$

Roughly speaking, a position v in t is an origin of w, if the label of the symbol occurring in t^L at position v is contained in the label of a symbol occurring in s^L in the path from its root to the position w.

Example 4. Consider again the rewrite step $\mu^L : t^L \to s^L$ of Example 3, and let w be the position 1.2 of s^L. The set of labeled symbols occurring in s^L in the path from its root to position w is the set $\mathtt{z} = \{\mathtt{h}^{\eta}, \mathtt{d}^{\alpha\beta\gamma}, \mathtt{d}^{\delta}\}$. Now, the labeled symbols occurring in t^L whose label is contained in the label of one element of $\mathtt{z}$ is the set $\{\mathtt{h}^{\eta}, \mathtt{f}^{\alpha}, \mathtt{g}^{\beta}, \mathtt{a}^{\gamma}, \mathtt{d}^{\delta}\}$. By Definition 4, the set of origin positions of w in μ^L is $\lhd^{\mathtt{L}}_{\mu}\mathtt{w} = \{\mathtt{1.1.2},\ \mathtt{1},\ \mathtt{1.1},\ \mathtt{1.2},\ \mathtt{\Lambda}\}$.

Note that the origin positions of w in the rewrite step $\mu : t \xrightarrow{r} s$ are not the antecedent positions of w in μ [17]; one main difference is the fact that we consider all positions of s in the path from its root to w for computing the origins, and we use the labeling to trace back every relevant piece of information involved in the step μ.

4.2 The Backward Trace Slicing Algorithm

First, let us formalize the slicing criterion, which basically represents the information we want to trace back across the execution trace in order to find out the "origins" of the data we observe. Given a term t, we denote by $\mathcal{O}_t$ the set of *observed* positions of t, which point to the symbols of t that we want to trace/observe.

Definition 5. *(slicing criterion) Given a rewrite theory $\mathcal{R} = (\Sigma, \Delta, R)$ and an execution trace $\mathcal{T} : s \rightarrow^* t$ in $\mathcal{R}$, a slicing criterion for $\mathcal{T}$ is any set $\mathcal{O}_t$ of positions of the term t.*

In the following, we show how backward trace slicing can be performed by exploiting the backward tracing relation $\lhd_\mu^L$ that was introduced in Definition 4. Informally, given a slicing criterion $\mathcal{O}_{t_n}$ for $\mathcal{T} : t_0 \rightarrow t_2 \rightarrow \ldots \rightarrow t_n$, at each rewrite step $t_{i-1} \rightarrow t_i$, $i = 1, \ldots, n$, our technique inductively computes the backward tracing relation between the relevant positions of t_i and those in t_{i-1}. The algorithm proceeds backwards, from the final term t_n to the initial term t_0, and recursively generates at step i the corresponding set of relevant positions, $P_{t_{n-i}}$. Finally, by means of a removal function, a simplified trace is obtained where each t_j is replaced by the corresponding *term slice* that contains only the relevant information w.r.t. P_{t_j}.

Definition 6. *(sequence of relevant position sets) Let $\mathcal{R} = (\Sigma, \Delta, R)$ be a rewrite theory, and $\mathcal{T} : t_0 \stackrel{r_1}{\rightarrow} t_1 \ldots \stackrel{r_n}{\rightarrow} t_n$ be an execution trace in $\mathcal{R}$. Let L_i be the labeling for the rewrite step $t_i \rightarrow t_{i+1}$ with $0 \leq i < n$. The sequence of relevant position sets in $\mathcal{T}$ w.r.t. the slicing criterion $\mathcal{O}_{t_n}$ is defined as follows:*

$$relevant\_positions(\mathcal{T}, \mathcal{O}_{t_n}) = [P_0, \ldots, P_n]$$
$$where \begin{cases} P_n = \mathcal{O}_{t_n} \\ P_j = \bigcup_{p \in P_{j+1}} \lhd^{L_j}_{(t_j \rightarrow\ t_{j+1})} p, \ with\ 0 \leq j < n \end{cases}$$

Now, it is straightforward to formalize a procedure that obtains a term slice from each term t in $\mathcal{T}$ and the corresponding set of relevant positions of t. We introduce the fresh symbol $\bullet \notin \Sigma$ to replace any information in the term that is not relevant (i.e., those symbols that occur at any position of t that is not above a relevant position of the term), hence does not affect the observed criterion.

Definition 7. *(term slice) Let $t \in T_\Sigma$ be a term and P be a set of positions of t. A* term slice *of t with respect to P is defined as follows:*

$$slice(t, P) = sl\_rec(t, P, \Lambda), \ where$$

$$sl\_rec(t, P, p) = \begin{cases} f(sl\_rec(t_1, P, p.1), \ldots, sl\_rec(t_n, P, p.n)) \\ \qquad if\ t = f(t_1, \ldots, t_n)\ and\ there\ exists\ w\ s.t.\ (p.w) \in P \\ \bullet \qquad otherwise \end{cases}$$

In the following, we use the notation $t^\bullet$ to denote a term slice of the term t. Roughly speaking, the symbol $\bullet$ can be thought of as a variable, so that any term $t' \in \tau(\Sigma)$ can be considered as a possible concretization of $t^\bullet$ if it is an "instance" of $[t^\bullet]$, where $[t^\bullet]$ is the term that is obtained by replacing all occurrences of $\bullet$ in $t^\bullet$ with fresh variables.

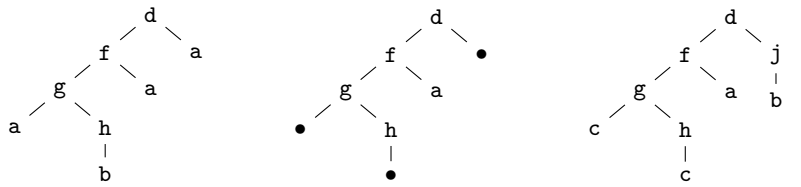

Fig. 1. A term slice and a possible concretization.

Definition 8. *(term slice concretization) Given $t' \in T_\Sigma$ and a term slice $t^\bullet$, we define $t^\bullet \propto t'$ if $[t^\bullet]$ is (syntactically) more general than t' (i.e. $[t^\bullet]\sigma = t'$, for some substitution σ). We also say that t' is a concretization of $t^\bullet$.*

Figure 1 illustrates the notions of term slice and term slice concretization for a given term t w.r.t. the set of positions $\{1.1.2, 1.2\}$.

Let us define a *sliced rewrite step* between two term slices as follows.

Definition 9. *(sliced rewrite step) Let $\mathcal{R} = (\Sigma, \Delta, R)$ be a rewrite theory and r a rule of $\mathcal{R}$. The term slice $s^\bullet$ rewrites to the term slice $t^\bullet$ via r (in symbols, $s^\bullet \xrightarrow{r} t^\bullet$) if there exist two terms s and t such that $s^\bullet$ is a term slice of s, $t^\bullet$ is a term slice of t, and $s \xrightarrow{r} t$.*

Using Definition 9, backward trace slicing is formalized as follows.

Definition 10. *(backward trace slicing) Let $\mathcal{R} = (\Sigma, \Delta, R)$ be a rewrite theory, and $\mathcal{T} : t_0 \xrightarrow{r_1} t_1 \ldots \xrightarrow{r_n} t_n$ be an execution trace in $\mathcal{R}$. Let $\mathcal{O}_{t_n}$ be a slicing criterion for $\mathcal{T}$, and let $[P_0, \ldots, P_n]$ be the sequence of the relevant position sets of $\mathcal{T}$ w.r.t. $\mathcal{O}_{t_n}$. A trace slice $\mathcal{T}^\bullet$ of $\mathcal{T}$ w.r.t. $\mathcal{O}_{t_n}$ is defined as the sliced rewrite sequence of term slices $t_i^\bullet = slice(t_i, P_i)$ which is obtained by glueing together the sliced rewrite steps in the set*

$$\mathcal{K}^\bullet = \{t_{k-1}^\bullet \xrightarrow{r_k} t_k^\bullet \mid 0 < k \leq n \ \wedge \ t_{k-1}^\bullet \neq t_k^\bullet\}.$$

Note that in Definition 10, the sliced rewrite steps that do not affect the relevant positions (i.e., $t_{k-1}^\bullet \xrightarrow{r_k} t_k^\bullet$ with $t_{k-1}^\bullet = t_k^\bullet$) are discarded, which further reduces the size of the trace.

A desirable property of a slicing technique is to ensure that, for any concretization of the term slice $t_0^\bullet$, the trace slice $\mathcal{T}^\bullet$ can be reproduced. This property ensures that the rules involved in $\mathcal{T}^\bullet$ can be applied again to every concrete trace $\mathcal{T}'$ that we can derive by instantiating all the variables in $[t_0^\bullet]$ with arbitrary terms.

Theorem 1. *(soundness) Let $\mathcal{R}$ be an elementary rewrite theory. Let $\mathcal{T}$ be an execution trace in the rewrite theory $\mathcal{R}$, and let $\mathcal{O}$ be a slicing criterion for $\mathcal{T}$. Let $\mathcal{T}^\bullet : t_0^\bullet \xrightarrow{r_1} t_1^\bullet \ldots \xrightarrow{r_n} t_n^\bullet$ be the corresponding trace slice w.r.t. $\mathcal{O}$. Then, for any concretization t'_0 of $t_0^\bullet$, it holds that $\mathcal{T}' : t'_0 \xrightarrow{r_1} t'_1 \ldots \xrightarrow{r_n} t'_n$ is an execution trace in $\mathcal{R}$, and $t_i^\bullet \propto t'_i$, for $i = 1, \ldots, n$.*

The proof of Theorem 1 relies on the fact that redex patterns are preserved by backward trace slicing. Therefore, for $i = 1, \ldots, n$, the rule r_i can be applied to any concretization t'_{i-1} of term $t_{i-1}^\bullet$ since the redex pattern of r_i does appear in $t_{i-1}^\bullet$, and hence in t'_{i-1}. A detailed proof of Theorem 1 is included in Appendix A.

5 Backward Trace Slicing for Extended Rewrite Theories

In this section, we consider an extension of our basic slicing methodology that allows us to deal with extended rewrite theories. An extended rewrite theory $\mathcal{R} = (\Sigma, E, R)$ is a rewrite theory where the equational theory (Σ, E) may contain associativity and commutativity axioms, and R may contain collapsing as well as nonleft-linear rules. Moreover, we provide a further extension to deal with the built-in operators existing in Maude, that is, operators that are not equipped with an explicit functional definition (e.g., Maude arithmetical operators and if-then-else conditional operators).

It is worth noting that all the proposed extensions are restricted to the labeling procedure of Section 4.1, leaving the backbone of our slicing technique unchanged.

5.1 Dealing with collapsing and nonleft-linear rules

Collapsing Rules. The main difficulty with collapsing rules is that they have a trivial contractum pattern, which consists in the empty context $\Box$; hence, it is not possible to propagate labels from the left-hand side of the rule to its right-hand side. This makes the rule labeling procedure of Definition 1 completely unproductive for trace slicing.

In order to overcome this problem, we keep track of the labels in the left-hand side of the collapsing rule r, whenever a rewrite step involving r takes place. This amounts to extending the labeling procedure of Definition 3 as follows.

Definition 11. *(rewrite step labeling for collapsing rules) Let $\mu : t \xrightarrow{r,\sigma} s$ be a rewrite step s.t. $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$. Let L_r be a labeling for the rule r. For the case of a rewrite step given by using a collapsing rule $r : \lambda \to x_i$, the labeling procedure formalized in Definition 3 is extended as follows:*

s_4. *Let t_i be the term introduced in s via the binding $x_i/t_i \in \sigma$, for some $i \in \{1, \ldots, n\}$. Then, the label l_i of the root symbol of t_i in s is replaced by a new composite label $l_c l_i$, where l_c is formed by joining all the labels appearing in the redex pattern of r^{L_r}.*

Example 5. Consider again the labeled collapsing rule $f^{\alpha}(a^{\beta}, x) \to x$, together with the rewrite step $\mu : f(a, h(b)) \to h(b)$ and matching substitution $\sigma = \{x/h(b)\}$. Let $L_{\sigma} = \{\{\Lambda \mapsto \gamma,\ 1 \mapsto \delta\}\}$ be the labeling for σ. Then, by applying Definition 11, the labeling of μ is

$$f^{\alpha}(a^{\beta}, h^{\gamma}(b^{\delta})) \to h^{\alpha\beta\gamma}(b^{\delta})$$

and the trace slice for $f(a, h(b)) \to h(b)$ w.r.t. the slicing criterion $\{\Lambda\}$ is $f(a, h(\bullet)) \to h(\bullet)$.

Note that if we had merely applied Definition 3 instead of Definition 11, we would have got the following labeling for μ: $f^{\alpha}(a^{\beta}, h^{\gamma}(b^{\delta})) \to h^{\gamma}(b^{\delta})$, which is undesirable since it does not correctly record the redex pattern information that we need for backward trace slicing: e.g. if we slice the rewriting step μ w.r.t. $\{\Lambda\}$ using this wrong labeling, we would get $f(\bullet, h(\bullet)) \to h(\bullet)$.

Nonleft-linear Rules. The trace slicing technique we described in Section 4 does not work for nonleft-linear TRS. Consider the rule: $r : f(x, y, x) \to g(x, y)$ and the one-step trace $\mathcal{T} : f(a, b, a) \to g(a, b)$. If we are interested in tracing back the symbol g that occurs in the final state $g(a, b)$, we would get the following trace slice $\mathcal{T}^{\bullet} : f(\bullet, \bullet, \bullet) \to g(\bullet, \bullet)$. However, $f(a, b, b)$ is a concretization of $f(\bullet, \bullet, \bullet)$ that cannot be rewritten by using r. In the following, we augment Definition 11 in order to also deal with nonleft-linear rules.

Definition 12. *(rewrite step labeling procedure for nonleft-linear rules) Let* $\mu : t \xrightarrow{r,\sigma} s$ *be a rewrite step s.t.* $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$. *Let* $L_{\sigma} = \{x_1/t_1, \ldots, x_n/t_n\}$ *be a labeling for the substitution* σ. *For the case of a rewrite step given by using a nonleft-linear rule* r, *the labeling procedure formalized in Definition 11 is extended as follows:*

s_5. *For each variable* x_j *that occurs more than once in the left-hand side of the rule* r, *the following steps should be performed:*
- *we form a new label* l_{x_j} *by joining all the labels in* $Cod(L_{x_j/t})$ *where* $L_{x_j/t} \in L_{\sigma}$;
- *let* l_s *be the label of the root symbol of* s. *Then,* l_s *is replaced by a new composite label* $l_{x_j} l_s$.

Example 6. Consider the nonleft-linear (labeled) rule $f^{\alpha}(x, y, x) \to g^{\alpha}(x, y)$ together with the rewrite step $\mu : f(g(a), b, g(a)) \to g(g(a), b)$, and matching substitution $\sigma = \{x/g(a), y/b\}$. Then, for the labeling $L_{\sigma} = \{L_{x/g(a)},\ L_{y/b}\}$, with $L_{x/g(a)} = \{\Lambda \mapsto \beta,\ 1 \mapsto \gamma\}$ and $L_{y/b} = \{\Lambda \mapsto \delta\}$, the labeled version of μ is

$$f^{\alpha}(g^{\beta}(a^{\gamma}), b^{\delta}, g^{\beta}(a^{\gamma})) \to g^{\alpha\beta\gamma}(g^{\beta}(a^{\gamma}), b^{\delta}).$$

Finally, by considering the criterion $\{1\}$, we can safely trace back the symbol g at the position 1 of the term $g(g(a), b)$ and obtain the following trace slice

$$f(g(a), \bullet, g(a)) \to g(g(\bullet), \bullet).$$

5.2 Built-in Operators

In practical implementations of RWL (e.g., Maude [9]), several commonly used operators are pre-defined (e.g., arithmetic and boolean operators, if-then-else constructs). Obviously, backward trace slicing of function calls involving built-in operators is not supported by our basic technique. This would require an explicit (rule-based or equational) specification of every single operator involved in the execution trace. To overcome this limitation, we further extend the labeling procedure of Definition 12 in order to deal with built-in operators.

Definition 13. *(rewrite step labeling procedure for built-in operators) For the case of a rewrite step* $\mu : C[op(t_1, \ldots, t_n)] \rightarrow C[t']$ *involving a call to a built-in, n-ary operator op, we extend Definition 12 by introducing the following additional case:*

s_6. *Given an initial labeling* L_{op} *for the term* $op(t_1, \ldots, t_n)$,
- *each symbol occurrence in* t' *is labeled with a new label that is formed by joining the labels of all the (labeled) arguments* $t_1, \ldots, t_n$ *of op;*
- *the remaining symbol occurrences of* $C[t']$ *that are not considered in the previous step inherit all the labels appearing in* $C[op(t_1, \ldots, t_n)]$.

For example, by applying Definition 13, the addition of two natural numbers implemented through the built-in operator + might be labeled as $+^{\alpha}(7^{\beta}, 8^{\gamma}) \rightarrow 15^{\beta\gamma}$.

5.3 Associative-Commutative Axioms

Let us finally consider an extended rewrite theory $\mathcal{R} = (\Sigma, \Delta \cup B, R)$, where B is a set of associativity (A) and commutativity (C) axioms that hold for some function symbols in Σ. As described in Section 3, an execution trace in $\mathcal{R}$ may contain rewrite steps modulo B that have the form $t =_B t' \rightarrow t''$, where $=_B$ is the congruence relation induced by the set of axioms B. Now, since B only contains associativity/commutativity (AC) axioms, terms can be represented by means of a single representative of their AC congruence class, called *AC canonical form* [11]. This representative is obtained by replacing nested occurrences of the same AC operator by a flattened argument list under a variadic symbol, whose elements are sorted by means of some linear ordering. In other words, if a function symbol f is declared to be associative, then the subterms rooted by f of any term t are flattened; and if f is also commutative, the subterms are sorted with respect to a fixed (internal) ordering [5].

The inverse process to flat transformation is unflat transformation, which is nondeterministic in the sense that it generates all the unflattended terms that are equivalent (modulo AC) to the flattened term.

For example, consider a binary AC operator f together with the standard lexicographic ordering over symbols. Given the B-equivalence $f(b, f(f(b, a), c)) =_B$

[5] Specifically, Maude uses the lexicographic order of symbols.

$f(f(b,c),f(a,b))$, we can represent it by using the "internal sequence" $f(b,f(f(b,a),c)) \rightarrow^*_{flat_B} f(a,b,b,c) \rightarrow^*_{unflat_B} f(f(b,c),f(a,b))$, where the first one corresponds to the *flattening* transformation sequence that obtains the AC canonical form, while the second one corresponds to the inverse, unflattening one.

These two processes are typically hidden inside the B-matching algorithms[6] that are used to implement rewriting modulo B.

The key idea for extending our labeling procedure in order to cope with B-equivalence $=_B$ is to exploit the flat transformation ($\rightarrow^*_{flat_B}$) and unflat transformation ($\rightarrow^*_{unflat_B}$) mentioned above. Without loss of generality, we assume that flat/unflat transformations are stable w.r.t. the lexicographic ordering over positions $\sqsubseteq$[7] (i.e., the relative ordering among the positions of multiple occurrences of a term is preserved).

This assumption allows us to trace back arguments of commutative operators, since multiple occurrences of the same symbol can be precisely identified.

Definition 14. *(AC Labeling.) Let f be an associative-commutative operator and B be the AC axioms for f. Consider the B-equivalence $t_1 =_B t_2$ and the corresponding (internal) flat/unflat transformation $\mathcal{T} : t_1 \rightarrow^*_{flat_B} s \rightarrow^*_{unflat_B} t_2$. Let L be an initial labeling for t_1. The labeling procedure for $t_1 =_B t_2$ is as follows.*

1. *(flattening) For each flattening transformation step $t_{|v} \rightarrow_{flat_B} t'_{|v}$ in $\mathcal{T}$ for the symbol f, a new label l_f is formed by joining all the labels attached to the symbol f in any position w of t^L such that $w = v$ or $w \geq v$, and every symbol on the path from v to w is f; then, label l_f is attached to the root symbol of $t'_{|v}$.*
2. *(unflattening) For each unflattening transformation step $t_{|v} \rightarrow_{unflat_B} t'_{|v}$ in $\mathcal{T}$ for the symbol f, the label of the symbol f in the position v of t^L is attached to the symbol f in any position w of t' such that $w = v$ or $w \geq v$, and every symbol on the path from v to w is f.*
3. *The remaining symbol occurrences in t' that are not considered in cases 1 or 2 above inherit the label of the corresponding symbol occurrence in t.*

Example 7. Consider the transformation sequence

$$f(b,f(b,f(a,c))) \rightarrow^*_{flat_B} f(a,b,b,c) \rightarrow^*_{unflat_B} f(f(b,c),f(a,b))$$

[6] See [8] (Section 4.8) for an in-depth discussion on matching and simplification modulo AC in Maude.

[7] The lexicographic ordering $\sqsubseteq$ is defined as follows: $\Lambda \sqsubseteq w$ for every position w, and given the positions $w_1 = i.w'_1$ and $w_2 = j.w'_2$, $w_1 \sqsubseteq w_2$ iff $i < j$ or ($i = j$ and $w'_1 \sqsubseteq w'_2$). Obviously, in a practical implementation of our technique, the considered ordering among the terms should be chosen to agree with the ordering considered by flat/unflat transformations in the RWL infrastructure.

by using Definition 14, the associated transformation sequence can be labeled as follows:

$$f^{\alpha}(b^{\beta}, f^{\gamma}(b^{\delta}, f^{\epsilon}(a^{\zeta}, c^{\eta}))) \rightarrow^{*}_{flat_B} f^{\alpha\gamma\epsilon}(a^{\zeta}, b^{\beta}, b^{\delta}, c^{\eta}) \rightarrow^{*}_{unflat_B} f^{\alpha\gamma\epsilon}(f^{\alpha\gamma\epsilon}(b^{\beta}, c^{\eta}), f^{\alpha\gamma\epsilon}(a^{\zeta}, b^{\delta}))$$

Note that the original order between the two occurrences of the constant b is not changed by the flat/unflat transformations. For example, in the first term, b^{β} is in position 1 and b^{δ} is in position 2.1 with $1 \sqsubseteq 2.1$, whereas, in the last term, b^{β} is in position 1.1 and b^{δ} is in position 2.2 with $1.1 \sqsubseteq 2.2$.

Finally, observe that the methodology described in this section can be easily extended to deal with other equational attributes such as identity (U) by explicitly encoding the internal transformations performed by Maude via suitable rewrite rules.

5.4 Extended Soundness

Soundness of the backward trace slicing algorithm for the extended rewrite theories is established by the following theorem which properly extends Theorem 1. The proof of such an extension can be found in Appendix A.

Theorem 2. *(extended soundness) Let $\mathcal{R} = (\Sigma, E, R)$ be an extended rewrite theory. Let $\mathcal{T}$ be an execution trace in the rewrite theory $\mathcal{R}$, and let $\mathcal{O}$ be a slicing criterion for $\mathcal{T}$. Let $\mathcal{T}^{\bullet} : t_0^{\bullet} \xrightarrow{r_1} t_1^{\bullet} \ldots \xrightarrow{r_n} t_n^{\bullet}$ be the corresponding trace slice w.r.t. $\mathcal{O}$. Then, for any concretization t'_0 of $t_0^{\bullet}$, it holds that $\mathcal{T}' : t'_0 \xrightarrow{r_1} t'_1 \ldots \xrightarrow{r_n} t'_n$ is an execution trace in $\mathcal{R}$, and $t_i^{\bullet} \propto t'_i$, for $i = 1, \ldots, n$.*

6 Experimental Evaluation

We have developed a prototype implementation of our slicing methodology which is publicly available at `http://www.dsic.upv.es/~dromero/slicing.html`. The implementation is written in Maude and consists of approximately 800 lines of code. Maude is a high-performance, reflective language that supports both equational and rewriting logic programming, which is particularly suitable for developing domain-specific applications [12, 13]. The reflection capabilities of Maude allow metalevel computations in RWL to be handled at the object-level. This facility allows us to easily manipulate computation traces of Maude itself and eliminate the irrelevant contents by implementing the backward slicing procedures that we have defined in this paper. Using reflection to implement the slicing tool has one important additional advantage, namely, the ease to rapidly integrate the tool within the Maude formal tool environment [10], which is also developed using reflection.

The prototype takes a Maude execution trace and a slicing criterion as input, and delivers a trace slice together with some quantitative information regarding the reduction achieved. The outcome is formatted in HTML, so it can be easily inspected by means of a Web browser.

In order to evaluate the usefulness of our approach, we benchmarked our prototype with several examples of Maude applications:

War of Souls (WoS). WoS is a nontrivial producer/consumer example that is modeled as a game in which an angel and a daemon fight to conquer the souls of human beings. Basically, when a human being passes away, his/her soul is sent to heaven or to hell depending on his/her faith as well as the strength of the angel and the daemon in play.

Fault-Tolerant Communication Protocol (FTCP). FTCP is a Maude specification borrowed from [16] that models a fault-tolerant, client-server communication protocol. There can be many clients and many servers, where a server can serve many clients; however, each client communicates with a single server. Also, the communication environment might be faulty —that is, messages can arrive out of order, can be duplicated, or can be lost.

Web-TLR: the Web application verifier. Web-TLR [3, 2] is a software tool designed for model-checking real-size Web applications (Web-mailers, Electronic forums, *etc.*) which is based on rewriting logic. Web applications are expressed as rewrite theories which can be formally verified by using the Maude built-in LTL(R) model checker [4].

A detailed description of these Maude applications and the Maude code are available at the URL mentioned above.

We have tested our tool on some execution traces which were generated by the Maude applications described above by imposing different slicing criteria. For each application, we considered two execution traces that were sliced using two different criteria. Table 1 summarizes the results we achieved.

As for the `WoS` example, we have chosen criteria that allow us to backtrace both the values produced and the entities in play — e.g., the criterion $\texttt{WoS}.\mathcal{T}_1.O_2$ isolates the angel and daemon behaviours along the trace $\mathcal{T}_1$.

Execution traces in the `FTCP` example represent client-server interactions. In this case, the chosen criteria aim at (i) isolating a server and/or a client in a scenario which involves multiple servers and clients ($\texttt{FTCP}.\mathcal{T}_2.O_1$), and (ii) tracking the response generated by a server according to a given client request ($\texttt{FTCP}.\mathcal{T}_1.O_1$).

In the last example, we have used Web-TLR to verify two LTL(R) properties of a Webmail application. The considered execution traces are much bigger for this program, and correspond to the counterexamples produced as outcome by the built-in model-checker of Web-TLR. In this case, the chosen criteria allow us to monitor the messages exchanged by the Web browsers and the Webmail server, as well as to focus our attention on the data structures of the interacting entities (e.g., browser/server sessions, server database).

For each criterion, Table 1 shows the size of the original trace and that of the computed trace slice, both measured as the length of the corresponding string. The *%reduction* column shows the percentage of reduction achieved. These results are very encouraging, and show an impressive reduction rate (up to $\sim 95\%$). Actually, sometimes the trace slices are small enough to be easily inspected by

Example	Example trace	Original trace size	Slicing criterion	Sliced trace size	% *reduction*
WoS	WoS.$\mathcal{T}_1$	776	WoS.$\mathcal{T}_1$.O_1	201	74.10%
			WoS.$\mathcal{T}_1$.O_2	138	82.22%
	WoS.$\mathcal{T}_2$	997	WoS.$\mathcal{T}_2$.O_1	404	58.48%
			WoS.$\mathcal{T}_2$.O_2	174	82.55%
FTCP	FTCP.$\mathcal{T}_1$	2445	FTCP.$\mathcal{T}_1$.O_1	895	63.39%
			FTCP.$\mathcal{T}_1$.O_2	698	71.45%
	FTCP.$\mathcal{T}_2$	2369	FTCP.$\mathcal{T}_2$.O_1	364	84.63%
			FTCP.$\mathcal{T}_2$.O_2	707	70.16%
Web-TLR	Web-TLR.$\mathcal{T}_1$	31829	Web-TLR.$\mathcal{T}_1$.O_1	1949	93.88%
			Web-TLR.$\mathcal{T}_1$.O_2	1598	94.97%
	Web-TLR.$\mathcal{T}_2$	72098	Web-TLR.$\mathcal{T}_2$.O_1	9090	87.39%
			Web-TLR.$\mathcal{T}_2$.O_2	7119	90.13%

Table 1. Summary of the reductions achieved.

the user, who can restrict her attention to the part of the computation she wants to observe getting rid of those data which are useless or even noisy w.r.t. the considered slicing criterion.

7 Related Work

Our backward tracing relation (Definition 4) extends a previous tracing relation that was formalized in [6] for orthogonal TRSs. In [6], a label is formed from atomic labels by using the operations of sequence concatenation and underlining, which are used to record every rule application (e.g., a, b, ab, $\underline{abcd}$, are labels). Collapsing rules are simply avoided by coding them away. This is done by replacing each collapsing rule $\lambda \to x$ with the rule $\lambda \to \varepsilon(x)$, where ε is a unary dummy symbol. Then, in order to lift the rewrite relation to terms containing ϵ occurrences, infinitely many new extra-rules are added that are built by saturating all left-hand sides with $\varepsilon(x)$. In contrast to [6], we use a more sophisticated notion of labeling, where composite labels are interpreted as sets of atomic labels, which allows us to deal with collapsing as well as nonleft-linear rules in an effective way.

The work that is most closely related to ours is [14]. [14] formalizes a notion of dynamic dependence among symbols by means of contexts and studies its application to program slicing of TRSs that may include collapsing as well as nonleft-linear rules. Both the *creating* and the *created* contexts associated with a reduction (i.e., the minimal subcontext that is needed to match the left-hand side of a rule and the minimal context that is "constructed" by the right-hand side of the rule, respectively) are tracked. Intuitively, these concepts are similar to our notions of redex and contractum patterns. The main differences with respect to our work are as follows. First, in [14] the slicing is given as a context, while we

consider term slices. Second, the slice is obtained only on the first term of the sequence by the transitive and reflexive closure of the dependence relation, while we slice the whole execution trace, step by step. Obviously, their notion of slice is smaller, but we think that our approach can be more useful for trace analysis and program debugging.

An extension of the method is described in [22], which provides a generic definition of labeling that works not only for orthogonal TRSs as is the case of [14] but for the wider class of all left-linear TRSs. Specifically, [22] describes a methodology of static and dynamic tracing which is mainly based on the notion of *sample of a traced proof term* —i.e., a pair (μ, P) that records a rewrite step $\mu = s \to t$, and a set P of reachable positions in t from a set of observed positions in s. The tracing proceeds forward, while ours employs a backward strategy which is particularly convenient for trace analysis.

Finally, [14] and [22] apply to TRSs whereas we deal with the richer framework of RWL that considers equations and equational axioms, namely rewriting modulo equational theories.

8 Conclusions

Trace slicing has been widely studied in imperative languages, where the dependence among program statements is generally determined by a program dependency graph (e.g., see [20] and the references therein). However, the notion of "dependence" in term rewriting languages, and particularly in RWL, is much more involved due to the combination of equations and rules. To the best of our knowledge, no trace slicing methodology for rewriting logic theories has yet been proposed.

The key idea behind our backward trace slicing technique consists in tracing back —through the rewrite sequence— all the relevant symbols of the final state that we are interested in. Given a slicing criterion $\mathcal{O}$ for a trace $\mathcal{T}$, our algorithm computes a trace slice $\mathcal{T}^\bullet$ that contains only the relevant information of $\mathcal{T}$ with respect to $\mathcal{O}$. The trace slicing technique can be applied to execution trace analysis of sophisticated rewrite theories, which can include equations and equational axioms as well as nonleft-linear and collapsing rules.

The proposed slicing technique has been implemented in a prototype system. Preliminary experiments demonstrate that the system works very satisfactorily on our benchmarks; e.g., we obtained trace slices that achieved a reduction of up to almost 95% in reasonable time (max. 0.5s on a Linux box equipped with an Intel Core 2 Duo 2.26GHz and 4Gb of RAM memory). Naturally, there is still much room for improvement, and we are currently working on increasing the efficiency of the tool. Also, as future work, we plan to deal with the execution traces of more sophisticated theories that may include membership and conditional equations.

References

1. Alpuente, M., Ballis, D., Baggi, M., Falaschi, M.: A Fold/Unfold Transformation Framework for Rewrite Theories extended to CCT. In: Proc. 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2010. pp. 43–52. ACM (2010)
2. Alpuente, M., Ballis, D., Espert, J., Romero, D.: Model-checking Web Applications with Web-TLR. In: 8th Int'l Symp. on Automated Technology for Verification and Analysis (ATVA 2010). Lecture Notes in Computer Science, vol. 6252, pp. 341–346. Springer (2010)
3. Alpuente, M., Ballis, D., Romero, D.: Specification and Verification of Web Applications in Rewriting Logic. In: Formal Methods, Second World Congress FM 2009. Lecture Notes in Computer Science, vol. 5850, pp. 790–805. Springer (2009)
4. Bae, K., Meseguer, J.: A Rewriting-Based Model Checker for the Linear Temporal Logic of Rewriting. In: Proc. of the 9th International Workshop on Rule-Based Programming (RULE'08). ENTCS, Elsevier (2008)
5. Baggi, M., Ballis, D., Falaschi, M.: Quantitative Pathway Logic for Computational Biology. In: Proc. of 7th International Conference on Computational Methods in Systems Biology (CMSB '09). Lecture Notes in Computer Science, vol. 5688, pp. 68–82. Springer (2009)
6. Bethke, I., Klop, J.W., de Vrijer, R.: Descendants and origins in term rewriting. Inf. Comput. 159(1-2), 59–124 (2000)
7. Chen, F., Rosu, G.: Parametric trace slicing and monitoring. In: TACAS. Lecture Notes in Computer Science, vol. 5505, pp. 246–261. Springer (2009)
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talco, C.: Maude Manual (Version 2.4). Tech. rep., SRI International, Computer Science Laboratory (2009), available at `http://maude.cs.uiuc.edu/maude2-manual/`
9. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude: A High-Performance Logical Framework, LNCS, vol. 4350. Springer-Verlag (2007)
10. Clavel, M., Durán, F., Hendrix, J., Lucas, S., Meseguer, J., Ölveczky, P.C.: The Maude Formal Tool Environment. In: CALCO. Lecture Notes in Computer Science, vol. 4624, pp. 173–178. Springer (2007)
11. Eker, S.: Associative-Commutative Rewriting on Large Terms. In: Proc. of 14th International Conference,Rewriting Techniques and Applications (RTA '03). Lecture Notes in Computer Science, vol. 2706, pp. 14–29. Springer (2003)
12. Eker, S., Meseguer, J., Sridharanarayanan, A.: The Maude LTL model checker and its implementation. In: Model Checking Software: Proc. 10 th Intl. SPIN Workshop. LNCS, vol. 2648, pp. 230–234. Springer (2003)
13. Escobar, S., Meadows, C., Meseguer, J.: A Rewriting-Based Inference System for the NRL Protocol Analyzer and its Meta-Logical Properties. Theoretical Computer Science 367(1-2), 162–202 (2006)
14. Field, J., Tip, F.: Dynamic dependence in term rewriting systems and its application to program slicing. In: Proc. of the 6th Int'l Symposium on Programming Language Implementation and Logic Programming, PLILP '94. pp. 415–431. Springer-Verlag, London, UK (1994)
15. Martí-Oliet, N., Meseguer, J.: Rewriting Logic: Roadmap and Bibliography. Theoretical Computer Science 285(2), 121–154 (2002)
16. Meseguer, J.: The Temporal Logic of Rewriting: A Gentle Introduction. In: Concurrency, Graphs and Models: Essays Dedicated to Ugo Montanari on the Occasion

of his 65th Birthday. vol. 5065, pp. 354–382. Springer-Verlag, Berlin, Heidelberg (2008)
17. Réty, P.: Improving basic narrowing techniques. In: Proc. the Conf. on Rewriting Techniques and Applications. pp. 228–241. Springer LNCS 256 (1987)
18. Riesco, A., Verdejo, A., Caballero, R., Martí-Oliet, N.: Declarative Debugging of Rewriting Logic Specifications. In: Recent Trends in Algebraic Development Techniques, 19th International Workshop, WADT 2008. Lecture Notes in Computer Science, vol. 5486, pp. 308–325. Springer (2009)
19. Riesco, A., Verdejo, A., Martí-Oliet, N.: Declarative Debugging of Missing Answers for Maude Specifications. In: Proc. RTA 2010. LIPIcs, vol. 6, pp. 277–294. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010)
20. Rosu, G., Havelund, K.: Rewriting-Based Techniques for Runtime Verification. Autom. Softw. Eng. 12(2), 151–197 (2005)
21. Talcott, C.: Pathway logic. Formal Methods for Computational Systems Biology 5016, 21–53 (2008)
22. TeReSe (ed.): Term Rewriting Systems. Cambridge University Press, Cambridge, UK (2003)
23. Viry, P.: Rewriting: An effective model of concurrency. In: Proceedings of the 6th Int'l PARLE Conference on Parallel Architectures and Languages Europe. pp. 648–660. Springer-Verlag, London, UK (1994)

A Proofs of Theorems 1 and 2

Proof of Theorem 1

We first demonstrate some auxiliary results which facilitate the proof of Theorem 1. The following auxiliary result is straightforward.

Lemma 1. *Let $t^\bullet$ be a term slice, and let t' be a term such that $t^\bullet \propto t'$. For every position $w \in \mathcal{P}os(t')$, it holds that, either $root(t'_{|w}) = root(t^\bullet_{|w})$, or there exists a position u of $t^\bullet$ such that $u \leq w$ and $root(t^\bullet_{|u}) = \bullet$.*

Proof. Immediate by Definition 8. □

The following definitions are auxiliary. Let C be a context. We define the set of positions of C as the set $\mathcal{P}os(C) = \{v \mid root(C_{|v}) \neq \Box\}$. Given a term t, by $path_w(t)$, we denote the set of symbols in t that occur in the path from its root to the position w of t, e.g., $path_{(2.1)}(f(a, g(b), c)) = \{f, g, b\}$.

Definition 15. *Let $r : \lambda \to \rho$ be a rule of $\mathcal{R}$. Let $\mu : s \overset{r,\sigma}{\to} t$ be a rewrite step such that $s = C[\lambda\sigma]$ and $t = C[\rho\sigma]$. Given a position w, we say that w* is involved in *μ, if there exist w' and w'' such that $w = w'.w''$, $C_{|w'} = \Box$ and $w'' \in \mathcal{P}os(\rho\sigma)$.*

The following lemma establishes that, if a relevant position is involved in a rewrite step, then the origin position relation preserves the redex pattern of the rule.

Lemma 2. *Let $r : \lambda \to \rho$ be a rule of an elementary rewrite theory $\mathcal{R}$. Let $\mu : s \overset{r,\sigma}{\to} t$ be a rewrite step such that $s = C[\lambda\sigma]$ and $t = C[\rho\sigma]$, where σ is a substitution and C is a context. Let L be a labeling for the rewrite step μ, and $w \in \mathcal{P}os(t)$.*

1. *if $w \in \mathcal{P}os(C)$, then $\lhd^L_\mu w = \{v \in \mathcal{P}os(C) \mid w = v.v'\}$*
2. *if $w = w'.w''$, $C_{|w'} = \Box$, and $w'' \in \mathcal{P}os(\rho\sigma)$, then $\lhd^L_\mu w \supseteq \{w'.v' \in \mathcal{P}os(s) \mid v' \in \mathcal{P}os(\lambda)\}$*

Proof. Given the rule $r : \lambda \to \rho$ and the labeling L for the rewrite step $\mu : s \overset{r,\sigma}{\to} t$, let us consider the labeled rewrite step $\mu^L : s^L \overset{r^L_{\mathrm{i}},\sigma^L}{\to} t^L$. By Definition 3, we can decompose the labeling L into three labelings L_C, L_r, and L_σ that respectively label the context C, the redex and the contractum patterns appearing in μ, and the terms in μ introduced by the substitution σ. In other words, we have $s^L = C^{L_C}[\lambda^{L_r}\sigma^{L_\sigma}]$ and $t^L = C^{L_C}[\rho^{L_r}\sigma^{L_\sigma}]$.

Let us prove the two claims independently.

Claim 1. We assume that $w \in \mathcal{P}os(t)$ and $w \in \mathcal{P}os(C)$. Since the context C has the same initial labeling C^{L_C} in both s and t, and the sets $Cod(L_C)$, $Cod(L_r)$, and $Cod(L_\sigma)$ are pairwise disjoint, the set of origin positions $\lhd^L_{s\to t} w$ in s is the set of positions lying on the path from the root position of s to w. Hence, $\lhd^L_\mu w = \{v \in \mathcal{P}os(C) \mid w = v.v'\}$.

Claim 2. We assume that $w = w'.w''$, $C_{|w'} = \Box$, and $w'' \in \mathcal{P}os(\rho\sigma)$. Then, since r belongs to an elementary rewrite theory $\mathcal{R}$, r is non-collapsing. This implies that there exists a labeled symbol $f^{l'} \in path_w(t^L)$ belonging to the contractum pattern of the rule r. By Definition 1, for each labeled symbol g^l in the redex pattern of r, we have that $l \subseteq l'$. Now, since the redex pattern of r is embedded into s and the contractum pattern of r is embedded into t, the inclusion $\lhd^L_\mu w \supseteq \{v.v' \in \mathcal{P}os(s) \mid v' \in \mathcal{P}os(\lambda)\}$ trivially holds by Definition 4. □

The following lemma establishes that, given the rewrite step $\mu : t_0 \xrightarrow{r} t_1$ and a term slice $t_0^\bullet$ of t_0, any concretization of $t_0^\bullet$ is reduced by the rule r to the corresponding term slice concretization of t_1.

Lemma 3. *Let $r : \lambda \to \rho$ be a rule of an elementary rewrite theory $\mathcal{R}$. Let $\mu : t_0 \xrightarrow{r,\sigma} t_1$ be a rewrite step such that $t_0 = C[\lambda\sigma]$ and $t_1 = C[\rho\sigma]$, where σ is a substitution and C is a context. Let L be a labeling for the rewrite step μ, and $[P_0, P_1]$ be the sequence of the relevant position sets for $\mu : t_0 \xrightarrow{r,\sigma} t_1$ w.r.t. the slicing criterion $\mathcal{O}$. Let $t_0^\bullet = slice(t_0, P_0)$, and $t_1^\bullet = slice(t_1, P_1)$.*

1. *if $P_1 \subseteq \mathcal{P}os(C)$ then $t_0^\bullet = t_1^\bullet$.*
2. *if $P_1 \cap \{w | w = v.v',\ C_{|v} = \Box,\ and\ v' \in \mathcal{P}os(\rho\sigma)\} \neq \emptyset$, then for any concretization t'_0 of $t_0^\bullet$, we have that $t'_0 \xrightarrow{r,\sigma'} t'_1$ where $t_1^\bullet \propto t'_1$.*

Proof. We proof the two claims separately.
Claim 1. Let $P_1 \subseteq \mathcal{P}os(C)$. Then, by Lemma 2 (Claim 1), for any $w \in P_1$, $\lhd^L_\mu w = \{v \in \mathcal{P}os(C) \mid w = v.v'\}$. Additionally, by Definition 6, $P_0 = \bigcup_{w \in P_1}(\lhd^L_\mu w)$, and hence $P_0 = \bigcup_{w \in P_1}\{v \in \mathcal{P}os(C) \mid w = v.v'\}$. Therefore, it holds that (i) $P_1 \subseteq P_0 \subseteq \mathcal{P}os(C)$, and for any $v \in P_0 \setminus P_1$, there exists a position v' such that $w = v.v'$ for some $w \in P_1$; (ii) by Definition 7, the function $slice(t, P)$ delivers a term slice $t^\bullet$ where all the symbols of t that do not occur in the path connecting the root position of t with some position $w \in P$ are abstracted by the $\bullet$ symbol. Now, since $t_0^\bullet = slice(t_0, P_0)$ and $t_1^\bullet = slice(t_1, P_1)$, by (i) and (ii), we can conclude that $\lambda\sigma$ and $\rho\sigma$ are abstracted by $\bullet$, and the context C is abstracted by the term slice $C^\bullet$ in both t_0 and t_1. Hence, $t_0^\bullet = C^\bullet[\bullet] = t_1^\bullet$.
Claim 2. We assume $P_1 \cap \{w | w = v.v',\ C_{|v} = \Box,$ and $v' \in \mathcal{P}os(\rho\sigma)\} \neq \emptyset$. Then, there exists a position $w \in P_1$ such that $w \in \{w | w = v.v',\ C_{|v} = \Box,$ and $v' \in \mathcal{P}os(\rho)\}$. By Lemma 2 (Claim 2), it follows that $\lhd^L_\mu w \supseteq \{v.v' \in \mathcal{P}os(t_0) \mid v' \in \mathcal{P}os(\lambda)\}$. By Definition 6, $P_0 = \bigcup_{w \in P_1}(\lhd^L_\mu w)$, and hence $P_0 \supseteq \{v.v' \in \mathcal{P}os(t_0) \mid v' \in \mathcal{P}os(\lambda)\}$. Now, by Definition 7 and the fact that $P_0 \supseteq \{v.v' \in \mathcal{P}os(t_0) \mid v' \in \mathcal{P}os(\lambda)\}$, the redex pattern of the rule r is embedded into $t_0^\bullet = slice(t_0, P_0)$. In other words, $t_0^\bullet = C^\bullet[\lambda\sigma^\bullet]$, where $C^\bullet$ is a term slice for the context C, and $\sigma^\bullet$ represents the term slices for the terms introduced by the substitution σ. Thus, by Lemma 1, any concretization t'_0 of $t_0^\bullet$ has the form $t'_0 = C'[\lambda\sigma']$, where $C^\bullet \propto C'$ and for each $x/t \in \sigma'$, there exists $x/t^\bullet \in \sigma^\bullet$ such that $t^\bullet \propto t$. Note also that $t_0^\bullet$ embeds the redex pattern $\lambda^\Box$ of r. Furthermore, since r belongs to the elementary rewrite theory $\mathcal{R}$, r is left-linear. Thus, the

following rewrite step $t'_0 \stackrel{r,\sigma'}{\rightarrow} t'_1$ can be executed for any substitution σ'. The rewrite step $t'_0 \stackrel{r,\sigma'}{\rightarrow} t'_1$ can be decomposed as follows: $t'_0 = C'[\lambda\sigma'] \stackrel{r,\sigma'}{\rightarrow} C'[\rho\sigma']$, for some context C' and substitution σ'. Moreover, by definition of rewrite step, t'_1 embeds the contractum pattern of r. Finally, $t_1^\bullet = C^\bullet[\rho^\bullet\sigma^\bullet]$, and thus t'_1 is a concretization of $t_1^\bullet$. □

The following proposition allows the soundness of our methodology to be proved for one-step traces on an elementary rewrite theory.

Proposition 1. *Let $\mathcal{R}$ be an elementary rewrite theory. Let $\mathcal{T}$ be an execution trace in $\mathcal{R}$, and let $\mathcal{O}$ be a slicing criterion for $\mathcal{T}$. Let $\mathcal{T}^\bullet : t_0^\bullet \stackrel{r_1}{\rightarrow} t_1^\bullet$ be the trace slice w.r.t. $\mathcal{O}$ of $\mathcal{T}$. Then, for any concretization t'_0 of $t_0^\bullet$, it holds that $\mathcal{T}' : t'_0 \stackrel{r_1}{\rightarrow} t'_1$ is an execution trace in $\mathcal{R}$ such that $t_1^\bullet \propto t'_1$.*

Proof. Given the trace slice $\mathcal{T}^\bullet : t_0^\bullet \stackrel{r_1}{\rightarrow} t_1^\bullet$ w.r.t. $\mathcal{O}$ of $\mathcal{T}$, let $[P_0, P_1]$ be the sequence of the relevant position sets of $\mathcal{T}$ w.r.t. $\mathcal{O}$. We have (i) $t_0^\bullet = \mathit{slice}(s_0, P_0)$ and $t_1^\bullet = \mathit{slice}(s_1, P_1)$, where $s_0 \stackrel{r_1}{\rightarrow} s_1$ is a rewrite step occurring in $\mathcal{T}$; (ii) $t_0^\bullet \neq t_1^\bullet$. Let r_1 be the rule $\lambda \rightarrow \rho$. The rewrite step $s_0 \stackrel{r_1}{\rightarrow} s_1$ can be decomposed as follows: $s_0 = C[\lambda\sigma] \stackrel{r_1}{\rightarrow} C[\rho\sigma] = s_1$, for some context C and substitution σ.

Since $\mathcal{R}$ is elementary and $t_0^\bullet \neq t_1^\bullet$, by Claim 1 of Lemma 3, $P_1 \not\subseteq \mathcal{P}os(C)$. Hence, there exists a position $w \in P_1$ such that $w = v.v'$ and $v' \in \mathcal{P}os(\rho\sigma)$. Also, because $\mathcal{R}$ is elementary, we can apply Claim 2 of Lemma 3, and for any concretization t'_0 of $t_0^\bullet$, we get $t'_0 \stackrel{r_1}{\rightarrow} t'_1$ such that t'_1 is a concretization of $t_1^\bullet$. □

Theorem 1. *(soundness) Let $\mathcal{R}$ be an elementary rewrite theory. Let $\mathcal{T}$ be an execution trace in $\mathcal{R}$ and let $\mathcal{O}$ be a slicing criterion for $\mathcal{T}$. Let $\mathcal{T}^\bullet : t_0^\bullet \stackrel{r_1}{\rightarrow} t_1^\bullet \ldots \stackrel{r_n}{\rightarrow} t_n^\bullet$ be the corresponding trace slice w.r.t. $\mathcal{O}$. Then, for any concretization t'_0 of $t_0^\bullet$, it holds that $\mathcal{T}' : t'_0 \stackrel{r_1}{\rightarrow} t'_1 \ldots \stackrel{r_n}{\rightarrow} t'_n$ is an execution trace in $\mathcal{R}$, and $t_i^\bullet \propto t'_i$, for $i = 1, \ldots, n$.*

Proof. The proof proceeds by induction on the length of the trace slice $\mathcal{T}^\bullet$ and exploits Proposition 1 to prove the inductive case. Routine. □

Proof of Theorem 2

In oder to prove Theorem 2, we use the same proof scheme as for elementary rewrite theories, since the extended technique described in Section 5 is only concerned with suitable extensions of the labeling procedure given in Definition 3, which do not affect the overall backward trace slicing methodology.

Let us start by proving an extension of Lemma 2 (Claim 2), which holds for nonleft-linear as well as collapsing rules.

Lemma 4. *Let $r : \lambda \rightarrow \rho$ be a rule that is either nonleft-linear or collapsing. Let $\mu : s \stackrel{r,\sigma}{\rightarrow} t$ be a rewrite step such that $s = C[\lambda\sigma]$ and $t = C[\rho\sigma]$, where σ is a substitution and C is a context. Let L be a labeling for the rewrite step μ, and $w \in \mathcal{P}os(t)$. Then,*

1. *if* $w \in \mathcal{P}os(C)$, *then* $\lhd^L_\mu w = \{v \in \mathcal{P}os(C) \mid w = v.v'\}$
2. *if* $w = w'.w''$, $C_{|w'} = \Box$, *and* $w'' \in \mathcal{P}os(\rho\sigma)$, *then* $\lhd^L_\mu w \supseteq \{w'.v' \in \mathcal{P}os(s) \mid v' \in \mathcal{P}os(\lambda)\}$

Proof. We prove the two claims separately.
Claim 1. The proof is identical to the proof of Claim 1 of Lemma 2.
Claim 2. To prove the lemma, we distinguish three cases.

Case 1: Rule r is collapsing. Given the collapsing rule $r = \lambda \to \rho$ where $\rho = x$ with $x \in Var(\lambda)$, let us consider the term t_i introduced by the substitution σ via the binding x/t_i, and we have $\mu = C[\lambda\sigma] \xrightarrow{r} C[t_i]$. Let us also consider the labeled rewrite step $\mu^L : s^L \xrightarrow{r^{L_r},\sigma^{L_\sigma}} t^L$ via the labeling L. By Definition 3, we have $s^L = C^{L_C}[\lambda^{L_r}\sigma^{L_\sigma}]$ and $t^L = C^{L_C}[t_i^{L_\sigma}]$.
Let $f^{l'}$ be the labeled root symbol of $t_i^{L_\sigma}$. By Definition 11 (Step s_4), we have that $l' = l_\lambda l_i$, where l_λ is formed by joining all the labels appearing in the redex pattern λ^{L_r} and l_i is the label of the root of the labeled term $t_i^{L_\sigma}$. This implies that, for each labeled symbol g^l in the redex pattern of r, we have that $l \subseteq l'$. Furthermore, by hypothesis, we have that $w \in C[t_i]$ and $w'' \in Pos(t_i)$. Hence, by Definition 4, the inclusion $\lhd^L_\mu w \supseteq \{v.v' \in \mathcal{P}os(s) \mid v' \in \mathcal{P}os(\lambda)\}$ trivially holds.

Case 2: rule r is nonleft-linear. Given the nonleft-linear rule r, the proof is perfectly analogous to the proof of Lemma 2 since, by Definition 12 (Step s_5), the label of each symbol in the contractum pattern of the rule r includes all the labels appearing in the redex pattern of r.

Case 3: rule r is collapsing and nonleft-linear. Since r is both collapsing and nonleft-linear, μ is labelled according to Definition 11 (Step s_4) and Definition 12 (Step s_5). Therefore, we can prove the claim by simply combining the arguments used to prove Case 1 ad Case 2.

□

The following Lemma extends Lemma 3 to deal with collapsing and nonleft-linear rules.

Lemma 5. *Let* $r : \lambda \to \rho$ *be a rule which is either left-linear or collapsing. Let* $\mu : t_0 \xrightarrow{r,\sigma} t_1$ *be a rewrite step such that* $t_0 = C[\lambda\sigma]$ *and* $t_1 = C[\rho\sigma]$, *where* σ *is a substitution and* C *is a context. Let* L *be a labeling for the rewrite step* μ, *and* $[P_0, P_1]$ *be the sequence of the relevant position sets for* $\mu : t_0 \xrightarrow{r,\sigma} t_1$ *w.r.t. the slicing criterion* $\mathcal{O}$. *Let* $t_0^\bullet = slice(t_0, P_0)$, *and* $t_1^\bullet = slice(t_1, P_1)$. *Then,*

1. *if* $P_1 \subseteq \mathcal{P}os(C)$ *then* $t_0^\bullet = t_1^\bullet$.
2. *if* $P_1 \cap \{w | w = v.v',\ C_{|v} = \Box,$ *and* $v' \in \mathcal{P}os(\rho\sigma)\} \neq \emptyset$, *then for any concretization* t_0' *of* $t_0^\bullet$, *we have that* $t_0' \xrightarrow{r,\sigma'} t_1'$ *where* $t_1^\bullet \propto t_1'$.

Proof. We proof the two claims separately.

Claim 1. The proof is identical to the proof of Claim 1 of Lemma 3.
Claim 2. To prove the lemma, we distinguish three cases.

Case 1: rule r is collapsing. Given the collapsing rule r, the proof is perfectly analogous to the one of Lemma 3 Claim 2. By using Lemma 4 instead of Lemma 2, we are still able to prove that the redex pattern of r embedded in t_0 is also embedded in $t_0^\bullet$, and hence for any concretization t_0' of $t_0^\bullet$, the rewrite step $t_0' \overset{r,\sigma'}{\to} t_1'$ can be proved. Finally, by using the same argument of Lemma 3 Claim 2, we conclude that $t_1^\bullet \propto t_1'$.

Case 2: rule r is nonleft-linear. Given the nonleft-linear rule r, the proof is similar to the one of Lemma 3. By exploiting Lemma 4 and Definition 12 (Step s_5), we can show that (i) the redex pattern of r embedded in t_0 is also embedded in $t_0^\bullet$, and (ii) for each term t introduced in t_0 by a binding $x/t \in \sigma$ such that x occurs multiple times in λ, t is preserved in $t_0^\bullet$ (i.e., t is not abstracted by $\bullet$ in $t_0^\bullet$). By (i) and (ii), it is immediate to prove that, for any concretization t_0' of $t_0^\bullet$, the rewrite step $t_0' \overset{r,\sigma'}{\to} t_1'$ can be proved. Finally, by using the same argument of Lemma 3 Claim 2, we can show that $t_1^\bullet \propto t_1'$.

Case 3: rule r is collapsing and nonleft-linear. Firstly we observe that, as the rule r is collapsing, by Lemma 4 the redex pattern of r embedded in t_0 is also embedded in $t_0^\bullet$, and hence for any concretization t_0' of $t_0^\bullet$, the redex pattern of r is embedded in t_0' as well. Secondly, since r is nonleft-linear, by Lemma 4 and Definition 12 (Step s_5), for each term t introduced in t_0 by a binding $x/t \in \sigma$ such that x occurs multiple times in λ, t is preserved in $t_0^\bullet$. Hence, t is also embedded in t_0', for any concretization t_0' of $t_0^\bullet$. From the two facts above, it directly follows that for any t_0' such that $t_0^\bullet \propto t_0'$, the rewrite step $t_0' \overset{r,\sigma'}{\to} t_1'$ can be proved. Finally, by using the same argument of Lemma 3 Claim 2, we can show that $t_1^\bullet \propto t_1'$.

□

The following proposition allows us to prove the soundness of our methodology for one-step traces on an extended rewrite theory.

Proposition 2. *Let $\mathcal{R}$ be an extended rewrite theory. Let $\mathcal{T} : t_0 \overset{r_1}{\to} t_1$ be an execution trace in $\mathcal{R}$, and let $\mathcal{O}$ be a slicing criterion for $\mathcal{T}$. Let $\mathcal{T}^\bullet : t_0^\bullet \overset{r_1}{\to} t_1^\bullet$ be the trace slice w.r.t. $\mathcal{O}$ of $\mathcal{T}$. Then, for any concretization t_0' of $t_0^\bullet$, it holds that $\mathcal{T}' : t_0' \overset{r_1}{\to} t_1'$ is an execution trace in $\mathcal{R}$ such that $t_1^\bullet \propto t_1'$.*

Proof. Consider the rewrite step $\mu : t_0 \overset{r_1}{\to} t_1$. In the case when r_1 is left-linear and non-collapsing (i.e., a rule belonging to an elementary rewrite theory), the proof is identical to the proof of Proposition 2. Hence w.l.o.g. we assume that r corresponds to a collapsing or nonleft-linear rule, built-in operator evaluation, or AC axiom.

Nonleft-linear/collapsing rules. In this case, the proof of Proposition 2 is analogous to the proof of Proposition 1, by using Lemma 5 in the place of Lemma 3.

Built-in Operators. Let $t_0 = C[op(t_1, \ldots, t_m)]$ and $t_1 = C[t']$. Hence, $\mu : C[op(t_1, \ldots, t_m)] \to C[t']$ is a rewrite step mimicking the evaluation of the built-in operator call $op(t_1, \ldots, t_m)$. By Definition 13 and Definition 4, it is

immediate to show that $op(t_1, \ldots, t_m)$ is embedded in $t_0^\bullet$, and thus for any concretization $t_0^\bullet \propto t'_0$, $t'_0 \xrightarrow{r_1} t'_1$ and $t_1^\bullet \propto t'_1$.

Associative-Commutative Axioms. Flat/unflat transformations are interpreted as rewrite steps that reduce AC symbols. Let us first consider the flat transformation $t \rightarrow_{flat_B} t'$ that reduces the AC symbol f. By Definition 14, the label of the occurrence of f in t' contains all the labels of the different occurrences of f appearing in t that have been reduced by the transformation. In other words, the label of f in t' keeps track of all the occurrences of f that have been reduced in t, and therefore the claim holds directly. The claim for unflat transformations can be proved in a similar way.

□

Finally, we exploit Proposition 2 in order to prove the extended soundness of our methodology on extended rewrite theories.

Theorem 2. *(extended soundness) Let $\mathcal{R} = (\Sigma, E, R)$ be an extended rewrite theory. Let $\mathcal{T}$ be an execution trace in the rewrite theory $\mathcal{R}$, and let $\mathcal{O}$ be a slicing criterion for $\mathcal{T}$. Let $\mathcal{T}^\bullet : t_0^\bullet \xrightarrow{r_1} t_1^\bullet \ldots \xrightarrow{r_n} t_n^\bullet$ be the corresponding trace slice w.r.t. $\mathcal{O}$. Then, for any concretization t'_0 of $t_0^\bullet$, it holds that $\mathcal{T}' : t'_0 \xrightarrow{r_1} t'_1 \ldots \xrightarrow{r_n} t'_n$ is an execution trace in $\mathcal{R}$ and $t_i^\bullet \propto t'_i$, for $i = 1, \ldots, n$.*

Proof. The proof proceeds by induction on the length of the trace slice $\mathcal{T}^\bullet$ and exploits Proposition 2 in order to prove the inductive case. Routine. □