MU-TERM GTRS: A Tool for Proving Termination of Generalized Term Rewriting Systems

Raúl Gutiérrez
DSIC & VRAIN, Universitat Politècnica de València, Spain

Salvador Lucas
DSIC & VRAIN, Universitat Politècnica de València, Spain

Abstract

We present MU-TERM GTRS, a tool for proving termination of Generalized Term Rewriting Systems (GTRSs), where rewriting computations are defined by rules that may include not only conditions involving reachability, joinability, and conversion, but also atoms defined by Horn clauses—particularly conditions expressing relations among terms (e.g., sort information, subterm relations, etc.). Termination control is enhanced through the use of a *replacement map*, which explicitly specifies the argument positions of function symbols where rewriting is allowed. GTRSs offer a powerful framework for modeling computations in advanced reduction-based languages such as Maude.

2012 ACM Subject Classification Theory of computation Equational logic and rewriting, Logic and verification, Automated reasoning

Keywords and phrases Generalized Term Rewriting Systems, Termination

Acknowledgements Supported by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe" (PID2021-122830OB-C42 and PID2021-122830OB-C44) and by the grant CIPROM/2022/6 funded by Generalitat Valenciana

1 Introduction

A *Generalized Term Rewriting System (GTRS [10])* is a tuple $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$, where $\mathcal{F}$ is a signature of *function symbols*; Π is a signature of *predicate symbols*, including $\to$ and $\to^*$; μ is a *replacement map* i.e., for all k-ary function symbols $f \in \mathcal{F}$, $\mu(f) \subseteq \{1, \ldots, k\}$ is the set of *active* arguments on which rewriting steps are allowed [9]; H is a set of definite Horn clauses $A \Leftarrow c$, where the predicate symbol of A is not $\to$ or $\to^*$; and R is a set of rewrite rules $\ell \to r \Leftarrow c$ [10, Definition 51]. In both cases, c is a sequence of atoms. Programs of reduction-based systems like Maude [2] can often be encoded as GTRSs [10, 11, Section 8].

▶ **Example 1.** The Maude program `OvConsOS` in Figure 1 (left) exemplifies the use of functions on both finite lists (predicate `NatList`) and infinite lists (predicate `NatIList`) of natural numbers 0, s(0),. . . , built using the list constructor cons, which is made "lazy" for the evaluation of the second argument by means of a `strat` annotation. Symbol zeros represents an infinite list of 0. Although take can be used to obtain a finite prefix of an infinite list, length should be applied to *finite* lists only. Thus, explicit *sorts* are given to inputs and output of functions (e.g., `length : NatList -> Nat`). The GTRS $\mathcal{R}$ in Figure 1 (right), given in the extended COPS format for GTRSs of [6], permits a proof of termination of `OvConsOS`, which depends on the appropriate sorting of function symbols (`length` can only be applied to finite lists) and the use of $\mu(\texttt{cons}) = \{1\}$.

Termination of GTRSs $\mathcal{R}$ in the usual sense, i.e., as the absence of infinite rewrite sequences $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$, has been studied in [11] using appropriate notions of *dependency pairs*. This paper presents MU-TERM GTRS[1], a tool for automatically (dis)proving termination of

[1] `http://zenon.dsic.upv.es/muterm-gtrs/`

```
fmod OvConsOS is
 sorts Nat NatList NatIList .
 subsort NatList < NatIList .
 op 0 : -> Nat .
 op s : Nat -> Nat .
 op zeros : -> NatIList .
 op nil : -> NatList .
 op cons : Nat NatIList -> NatIList [strat (1 0)] .
 op cons : Nat NatList -> NatList [strat (1 0)] .
 op take : Nat NatIList -> NatList .
 op length : NatList -> Nat .
 vars M N : Nat .
 var IL : NatIList .
 var L : NatList .
 eq zeros = cons(0,zeros) .
 eq take(0,IL) = nil .
 eq take(s(M),cons(N,IL)) = cons(N,take(M,IL)) .
 eq length(nil) = 0 .
 eq length(cons(N,L)) = s(length(L)) .
endfm
```

```
(VAR M N L IL)
(REPLACEMENT-MAP
 (cons 1)
)
(HORN-CLAUSES
 NatIList(L) | NatList(L)
 Nat(0)
 Nat(s(N)) | Nat(N)
 NatList(nil)
 NatList(cons(N,L)) | Nat(N), NatList(L)
 NatIList(zeros)
 NatIList(cons(N,IL)) | Nat(N), NatIList(IL)
 NatList(take(N,IL)) | Nat(N), NatIList(IL)
 Nat(length(L)) | NatList(L)
)
(RULES
 zeros -> cons(0,zeros)
 take(0,IL) -> nil | NatIList(IL)
 take(s(M),cons(N,IL)) -> cons(N,take(M,IL))
   | Nat(M), Nat(N), NatIList(IL)
 length(nil) -> 0
 length(cons(N,L)) -> s(length(L))
   | Nat(N), NatList(L)
)
```

Figure 1 The Maude module OvConsOS in [3, Figure 1] and MU-TERM GTRS module

Table 1 Generic sentences of the first-order theory of rewriting

Label	Sentence
(Rf)	$(\forall x)\ x \to^* x$
(Co)	$(\forall x, y, z)\ x \to y \wedge y \to^* z \Rightarrow x \to^* z$
$(\mathrm{Pr})_{f,i}$	$(\forall x_1, \ldots, x_k, y_i)\ x_i \to y_i \Rightarrow f(x_1, \ldots, x_i, \ldots, x_k) \to f(x_1, \ldots, y_i, \ldots, x_k)$
$(\mathrm{HC})_{A \Leftarrow A_1, \ldots, A_n}$	$(\forall x_1, \ldots, x_p)\ A_1 \wedge \cdots \wedge A_n \Rightarrow A$
	where $x_1, \ldots, x_p$ are the variables occurring in $A_1, \ldots, A_n$ and A

GTRSs based on the Dependency Pair Framework introduced in [11].

2 Termination of GTRSs using Dependency Pairs

A GTRS $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ determines a first-order theory $\overline{\mathcal{R}} = \{(\mathrm{Rf}), (\mathrm{Co})\} \cup \{(\mathrm{Pr})_{f,i} \mid f \in \mathcal{F}, i \in \mu(f)\} \cup \{(\mathrm{HC})_\alpha \mid \alpha \in H \cup R\}$ (see Table 1). Note that rules in R are Horn clauses which are often given a label α.

▶ **Example 2.** The GTRS in Figure 1 (right) is $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ in Figure 2, where $\mathcal{F} = \{\mathsf{0}, \mathsf{s}, \mathsf{nil}, \mathsf{cons}, \mathsf{zeros}, \mathsf{take}, \mathsf{length}\}$, $\Pi = \{\to, \to^*, \mathsf{Nat}, \mathsf{NatList}, \mathsf{NatIList}\}$, $\mu(\mathsf{cons}) = \{1\}$ and $\mu(f) = \{1, \ldots, k\}$ for all k-ary $f \in \mathcal{F} - \{\mathsf{cons}\}$, $H = \{(1), \ldots, (9)\}$, and $R = \{(10), \ldots, (14)\}$.

For all terms s and t, we write $s \to_{\mathcal{R}} t$ (resp. $s \to^*_{\mathcal{R}} t$) iff $\overline{\mathcal{R}} \vdash s \to t$ (resp. $\overline{\mathcal{R}} \vdash s \to^* t$); $\mathcal{R}$ is terminating iff there is no infinite sequence $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$.

Symbols $f \in \mathcal{F}$ are called *defined* if $\mathit{root}(\ell) = f$ for some $\ell \to r \Leftarrow c \in R$. A subterm t of s is *active* (regarding μ, written $s \trianglerighteq_\mu t$) if $s = t$ or $s = f(s_1, \ldots, s_i, \ldots, s_k)$ and $s_i \trianglerighteq_\mu t$ for some $i \in \mu(f)$. We say that t is *frozen* in s (written $s \vartriangleright_{\not\mu} t$) if $s \trianglerighteq t$ and $s \ntrianglerighteq_\mu t$. As for the DP approach for TRSs [1, 8], 'classical' dependency pairs for GTRSs $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ are:

$$H_{DP}(\mathcal{R}) = \{\ell^\sharp \xrightarrow{\mathsf{t}} v^\sharp \Leftarrow c \mid \ell \to r \Leftarrow c \in R, r \trianglerighteq_\mu v,\ \ \mathit{root}(v) \text{ is defined, and } \ell \ntrianglerighteq_\mu v\}$$

with $\xrightarrow{\mathsf{t}}$ a new predicate symbol. For $t = f(t_1, \ldots, t_k)$, let $t^\sharp = f^\sharp(t_1, \ldots, t_k)$ for a *new* symbol $f^\sharp$, often *capitalized* as F. For $\mathcal{R}$ in Ex. 2, $H_{DP}(\mathcal{R}) = \{(15)\}$ with

$$\mathsf{LENGTH}(\mathsf{cons}(N, L)) \xrightarrow{\mathsf{t}} \mathsf{LENGTH}(L) \Leftarrow \mathsf{Nat}(N), \mathsf{NatList}(L) \qquad (15)$$

$$
\begin{array}{llll}
\mathsf{NatIList}(L) \Leftarrow \mathsf{NatList}(L) & (1) & \mathsf{NatIList}(\mathsf{cons}(N, IL)) \Leftarrow \mathsf{Nat}(N), \mathsf{NatIList}(IL) & (6)\\
\mathsf{Nat}(0) & (2) & \mathsf{NatList}(\mathsf{cons}(N, L)) \Leftarrow \mathsf{Nat}(N), \mathsf{NatList}(L) & (7)\\
\mathsf{Nat}(\mathsf{s}(N)) \Leftarrow \mathsf{Nat}(N) & (3) & \mathsf{NatList}(\mathsf{take}(N, IL)) \Leftarrow \mathsf{Nat}(N), \mathsf{NatIList}(IL) & (8)\\
\mathsf{NatList}(\mathsf{nil}) & (4) & \mathsf{Nat}(\mathsf{length}(L)) \Leftarrow \mathsf{NatList}(L) & (9)\\
\mathsf{NatIList}(\mathsf{zeros}) & (5) & &
\end{array}
$$

$$
\begin{array}{lr}
\mathsf{zeros} \to \mathsf{cons}(0, \mathsf{zeros}) & (10)\\
\mathsf{take}(0, IL) \to \mathsf{nil} \Leftarrow \mathsf{NatIList}(IL) & (11)\\
\mathsf{take}(\mathsf{s}(M), \mathsf{cons}(N, IL)) \to \mathsf{cons}(N, \mathsf{take}(M, IL)) \Leftarrow \mathsf{Nat}(M), \mathsf{Nat}(N), \mathsf{NatIList}(IL) & (12)\\
\mathsf{length}(\mathsf{nil}) \to 0 & (13)\\
\mathsf{length}(\mathsf{cons}(N, L)) \to \mathsf{s}(\mathsf{length}(L)) \Leftarrow \mathsf{Nat}(N), \mathsf{NatList}(L) & (14)
\end{array}
$$

Figure 2 GTRS for the Maude module OvConsOS in [3, Figure 1]

Dependency pairs for TRSs are useful to encode (possibly) infinite rewrite sequences (see [8]). However, in order to properly capture termination of GTRSs we also need *collapsing* dependency pairs [11, Sect. 5] to handle computations involving conditions and context-sensitive replacement restrictions. By expressing special features of conditional and context-sensitive dependency pairs as Horn clauses, we obtain a new valid GTRS as a result. While [11, Def. 16] defines them for GTRSs, we refine this using the richer framework from [4], based on hidden terms and hiding contexts.

A term t such that $root(t)$ is a *defined symbol* is *hidden* if there is a rule $\alpha : \ell \to r \Leftarrow c$ such that t is a frozen subterm of r: $r \rhd_{\not\mu} t$; $\mathcal{HT}(\mathcal{R})$ is the set of hidden terms in $\mathcal{R}$.

▶ **Example 3.** For $\mathcal{R}$ in Example 2, $\mathcal{HT}(\mathcal{R}) = \{\mathsf{zeros}, \mathsf{take}(M, IL)\}$ (in red on the right-hand sides of (10) and (12)).

Instances of hidden terms are the only ones which may *become active* when matched by a left-hand side. More precisely, a function symbol f *hides the active argument position* $i \in \mu(f)$ *in the right hand side* r *of* α if $r \rhd_{\not\mu} f(r_1, \ldots, r_k)$ for some terms $r_1, \ldots, r_k$, and r_i contains (a) an *active defined symbol* or (b) an active variable which is (b.1) frozen both in ℓ and r, and (b.2) not active in ℓ or r.

When dealing with *conditional* rules $\ell \to r \Leftarrow c$, collapsing dependency pairs are also necessary to capture the continuation of infinite rewrite sequences on subterms s of instances $\sigma(x)$ of variables x occurring in r but *not occurring* in ℓ and possibly occurring in the conditional part c, see [15, Sect. 4.3]. Let $\varpi_{\unrhd_{\mathsf{unh}}}$, $\varpi_{\unrhd_\mu}$ and Mk be new (binary) predicate symbols (denoting the *hiding* and *active subterm* relation on terms, and the *marking* of terms), respectively defined by sets of clauses $Unh(\mathcal{F}) = \{x \,\varpi_{\unrhd_{\mathsf{unh}}}\, x\} \cup \{f(x_1, \ldots, x_i, \ldots, x_k) \,\varpi_{\unrhd_{\mathsf{unh}}} x'_i \Leftarrow x_i \,\varpi_{\unrhd_\mu}\, x'_i \mid f \in \mathcal{F}, f \text{ hides } i\}$, $\mathcal{S}ubt(\mathcal{F}, \mu) = \{x \,\varpi_{\unrhd_\mu}\, x\} \cup \{f(x_1, \ldots, x_i, \ldots, x_k) \,\varpi_{\unrhd_\mu}\, x'_i \Leftarrow x_i \,\varpi_{\unrhd_\mu}\, x'_i \mid f \in \mathcal{F}, k = ar(f), i \in \mu(f)\}$. and $\mathcal{M}ark(\mathcal{F}) = \{\mathsf{Mk}(f(x_1, \ldots, x_k), f^\sharp(x_1, \ldots, x_k)) \mid f \in \mathcal{F}, k = ar(f)\}$, respectively. Then, we let

$$
\begin{array}{rcl}
H_{DPC}(\mathcal{R}) & = & \{\ell^\sharp \xrightarrow{\mathsf{t}} t^\sharp \Leftarrow c, x \;\varpi_{\unrhd_{\mathsf{unh}}}\; t \mid \ell \to r \Leftarrow c \in R, t \in \mathcal{HT}(\mathcal{R}), \text{ and} \\
 & & \quad x \in \mathcal{V}ar(\ell) \cap [\mathcal{V}ar^\mu(r) - \mathcal{V}ar^\mu(\ell)]\} \\
 & \cup & \{\ell^\sharp \xrightarrow{\mathsf{t}} x'' \Leftarrow c, x \;\varpi_{\unrhd_\mu}\; x', \mathsf{Mk}(x', x'') \mid \ell \to r \Leftarrow c \in R, \text{ and} \\
 & & \quad x \in \mathcal{V}ar^\mu(r) - \mathcal{V}ar(\ell) \text{ and } x' \text{ and } x'' \text{ are fresh variables}\} \qquad (16)
\end{array}
$$

where, given a term t, $\mathcal{V}ar^\mu(t)$ is the set of variables which occur active in t.

▶ Remark 4. In contrast to (16), in [11, Def. 16] $H_{DPC}(\mathcal{R})$ only considers the second group of *collapsing* pairs *with* $x \in \mathcal{V}ar^{\mu}(r) - \mathcal{V}ar^{\mu}(\ell)$. Then, for $\mathcal{R}$ in Example 2, $H_{DPC}(\mathcal{R})$ would consist of a single (collapsing) pair

$$\mathsf{LENGTH}(\mathsf{cons}(N, L)) \xrightarrow{\mathsf{t}} L'' \Leftarrow \mathsf{Nat}(N), \mathsf{NatList}(L), L \mathrel{\varpi_{\trianglerighteq_\mu}} L', \mathsf{Mk}(L', L'') \tag{17}$$

▶ **Example 5.** For $\mathcal{R}$ in Example 2, using (16), $H_{DPC}(\mathcal{R}) = \{(18), (19)\}$ with

$$\mathsf{LENGTH}(\mathsf{cons}(N, L)) \xrightarrow{\mathsf{t}} \mathsf{ZEROS} \Leftarrow \mathsf{Nat}(N), \mathsf{NatList}(L), L \mathrel{\varpi_{\trianglerighteq_{\mathsf{unh}}}} \mathsf{zeros} \tag{18}$$

$$\mathsf{LENGTH}(\mathsf{cons}(N, L)) \xrightarrow{\mathsf{t}} \mathsf{TAKE}(M, IL) \Leftarrow \mathsf{Nat}(N), \mathsf{NatList}(L), L \mathrel{\varpi_{\trianglerighteq_{\mathsf{unh}}}} \mathsf{take}(M, IL) \tag{19}$$

▶ **Definition 6.** *Let* $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ *be a GTRS whose set of defined symbols is* $\mathcal{D}$. *The GTRS* $\mathsf{DP}_{HC}(\mathcal{R}) = (\mathcal{F}_{HC}, \Pi_{HC}, \mu^\sharp, H_{HC}, R)$, *where:* $\mathcal{F}_{HC} = \mathcal{F} \cup \mathcal{D}^\sharp$; $\Pi_{HC} = \Pi \cup \{\varpi_{\trianglerighteq_{\mathsf{unh}}}, \varpi_{\trianglerighteq_\mu}, \mathsf{Mk}\}$; *for all* $f \in \mathcal{F}_{HC}$, $\mu^\sharp(f) = \mu(f)$ *if* $f \in \mathcal{F}$ *and* $\mu(g)$ *if* $f = g^\sharp$ *for some* $g \in \mathcal{D}$; *and* $H_{HC} = H \cup \mathit{Unh}(\mathcal{F}) \cup \mathit{Subt}(\mathcal{F} \cup \mathcal{D}^\sharp, \mu^\sharp) \cup \mathit{Mark}(\mathcal{D}) \cup H_{DP}(\mathcal{R}) \cup H_{DPC}(\mathcal{R})$.

▶ **Theorem 7.** *A GTRS* $\mathcal{R}$ *is* terminating *if there is no infinite minimal* $\mathsf{DP}_{HC}(\mathcal{R})$*-chain*[2]. *If there is an infinite* $\mathsf{DP}_{HC}(\mathcal{R})$*-chain, then* $\mathcal{R}$ *is not terminating.*

In the following, given a GTRS $\mathcal{R}$, we write $P_{\mathcal{R}}$ to denote the subset of Horn clauses in $\mathcal{R}$ of the form $u \xrightarrow{\mathsf{t}} v \Leftarrow c$. Note that $P_{\mathsf{DP}_{HC}(\mathcal{R})} = H_{DP}(\mathcal{R}) \cup H_{DPC}(\mathcal{R})$.

3 Implementation of the DP Framework for GTRSs.

MU-TERM GTRS is implemented in 27 Haskell modules including around 4000 *loc*. The tool has been developed independently from MU-TERM [5] only using semantic processors, so they are independent tools. A specific parser has been created to accept the TPDB notation[3] enriched with a new block `HORN-CLAUSES` to specify them (see Figure 1)[4]. To implement the framework, the followign data type is used:

```haskell
data GTRS c
   = GTRS { gName :: Map Int String, gArity :: Map Int Int, gMu :: Map Int [Int]
          , gSymbols :: Set Int, gVariables :: Set Int , gPredicates :: Set Int
          , gEquations :: Set (CEquation c), gHornClauses :: Set (HornClause c)
          , gRules :: Set (CRule c), gLabel :: String} deriving (Eq,Show)
```

We use the `muterm-framework` library to define instances of problems, processors, and also strategy combinators to easily define our strategy (see below).

```haskell
data Problem typ p = Problem typ p
-- | GRewriting problem
data GRewriting = GRewriting PScheme deriving (Eq, Ord, Show)
-- | Problems contains a rewrite system
class IsProblem typ problem | typ -> problem where
    getProblemType :: Problem typ problem -> typ
    getR :: Problem typ problem -> problem
-- | GRewriting problem is a Problem
instance IsProblem GRewriting where
  data Problem GRewriting a = GRew PScheme a deriving (Eq, Ord, Show)
  getProblemType (GRew m _) = GRewriting m t
  getR (GRew _ r) = r
```

[2] We use the natural extension of chain, see [11, Definition 14]
[3] `https://www.lri.fr/~marche/tpdb/format.html`
[4] As part of future work, we aim to extend the ARI format as well.

where `PScheme` is a pair that records whether the current problem can be proved terminating or non-terminating. Initially, `PScheme` has the value (m, a); this means that to test termination, we start with minimal sequences, but to test non-termination, we start with arbitrary sequences. A processor modifies the pair values depending on whether the processor is sound or complete. These flags can only have the following values: m for minimal, a for arbitrary, or $\bullet$ to indicate that the processor returns an incompatible type. The computation of the initial GTRS problem is treated as a special *dependency pair processor*:

```
data DPProcessor = DPProcessor
-- | The information obtained is the new GTRS
data DPProcInfo problem
    = DPProcInfo { outProblem :: problem }
-- DPProcessor
instance (...  functional  dependencies  ...  )
  => Processor info DPProcessor (Problem GRewriting trs)
                                (Problem GRewriting trs)  where
    apply  DPProcessor inP = singleP (DPProcInfo outP) inP outP  where ...
```

Each processor has its own name. The strategy combines the different processors in order to find a proof tree. We use the following strategy:

```
grewStrat
 = idProc .&. (mace4irProc  .|.  return) .&. dpProc .&. (infProc  .|.  simpProc  .|.  return)
    .&. sccProc .&.  fixSolver ((subProc  .|.  mace4rpProc .|.  agesrpProc) .&. sccProc)
```

where `mace4irProc` removes infeasible rules of the input system, and the other names refer to the processors described in [7, Section 5.1] and in [11, Section 7]. `simpProc` applies simplification processors, `infProc` applies non-termination techniques and we use Mace4 and AGES for the semantic RP processor. Finally, `.&.` and `.|.` are the ‘and’ and ‘or’ strategy combinators, and `fixSolver` is a fix-point application strategy.

4 Conclusions and Future Work

We have presented MU-TERM GTRS, a new tool for proving termination of GTRSs. The tool implements the Dependency Pair Framework for proving termination of GTRSs [11], although we have introduced some relevant improvements in the treatment of collapsing dependency pairs induced by the replacement restrictions (see (16) and Remark 4), advantageously adapting the developments in [4]. All processors described in [11] have been implemented. Besides, some processors developed for the Confluence Framework for proving confluence of GTRSs [7] have been adapted and implemented as well. Since GTRSs can be used to model Maude programs (see Example 1), the tool can be used to prove termination of Maude programs. The stronger property of *operational termination* [13], which implies termination (but not vice versa) has been defined for GTRSs in [12]. Operational termination of GTRSs is also a subject for future work as it is important for a practical use of GTRSs, as discussed in [14] in the context of rewrite theories.

References

1 Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theor. Comput. Sci.*, 236(1-2):133–178, 2000. doi:10.1016/S0304-3975(99)00207-8.

2 Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. *All About Maude - A High-Performance Logical Framework,*

How to Specify, Program and Verify Systems in Rewriting Logic, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007. doi:10.1007/978-3-540-71999-1.

3 Francisco Durán, Salvador Lucas, José Meseguer, Claude Marché, and Xavier Urbain. Proving termination of membership equational programs. In Nevin Heintze and Peter Sestoft, editors, *Proceedings of the 2004 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, 2004*, pages 147–158. ACM, 2004. doi:10.1145/1014007.1014022.

4 Raúl Gutiérrez and Salvador Lucas. Proving termination in the context-sensitive dependency pair framework. In Peter Csaba Ölveczky, editor, *Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Held as a Satellite Event of ETAPS 2010, Revised Selected Papers*, volume 6381 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2010. doi:10.1007/978-3-642-16310-4_3.

5 Raúl Gutiérrez and Salvador Lucas. MU-TERM: Verify Termination Properties Automatically (System Description). In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 436–447. Springer, 2020. doi:10.1007/978-3-030-51054-1_28.

6 Raúl Gutiérrez and Salvador Lucas. Proving and disproving feasibility with infChecker. In Raúl Gutérrez and Naoki Nishida, editors, *14th International Workshop on Confluence, IWC 2025*, page to appear, 2025.

7 Raúl Gutiérrez, Salvador Lucas, and Miguel Vítores. Proving Confluence in the Confluence Framework with CONFident. *Fundamenta Informaticae*, 192(2):167–217, 2024. doi:10.3233/FI-242192.

8 Nao Hirokawa and Aart Middeldorp. Dependency pairs revisited. In Vincent van Oostrom, editor, *Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004, Proceedings*, volume 3091 of *Lecture Notes in Computer Science*, pages 249–268. Springer, 2004. doi:10.1007/978-3-540-25979-4_18.

9 Salvador Lucas. Context-sensitive Rewriting. *ACM Comput. Surv.*, 53(4):78:1–78:36, 2020. doi:10.1145/3397677.

10 Salvador Lucas. Local confluence of conditional and generalized term rewriting systems. *Journal of Logical and Algebraic Methods in Programming*, 136:paper 100926, pages 1–23, 2024. doi:10.1016/j.jlamp.2023.100926.

11 Salvador Lucas. Termination of Generalized Term Rewriting Systems. In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024)*, volume 299 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2024.32.

12 Salvador Lucas. Semantic Properties of Computations Defined by Elementary Inference Systems. In Emmanuel D'Angelis and Florian Frohn, editors, *12th International Workshop on Horn Clauses in Verification and Synthesis, HCVS 2025*, 2025. to appear.

13 Salvador Lucas, Claude Marché, and José Meseguer. Operational termination of conditional term rewriting systems. *Inf. Process. Lett.*, 95(4):446–453, 2005. doi:10.1016/j.ipl.2005.05.002.

14 Salvador Lucas and José Meseguer. Normal forms and normal theories in conditional rewriting. *J. Log. Algebr. Meth. Program.*, 85(1):67–97, 2016. doi:10.1016/j.jlamp.2015.06.001.

15 Salvador Lucas and José Meseguer. Dependency pairs for proving termination properties of conditional term rewriting systems. *J. Log. Algebr. Meth. Program.*, 86(1):236–268, 2017. doi:10.1016/j.jlamp.2016.03.003.