Bernhard Gramlich
Salvador Lucas (Eds.)

Reduction Strategies in Rewriting and Programming

3rd International Workshop, WRS 2003
Valencia, Spain, June 8, 2003
Proceedings

Volume Editors

Bernhard Gramlich
Technische Universität Wien, Austria
Email: gramlich@logic.at

Salvador Lucas
Universidad Politécnica de Valencia, Spain
Email: slucas@dsic.upv.es

Proceedings of the 3rd International Workshop on Reduction Strategies in Rewriting and Programming, WRS'03
Valencia, Spain, June 8, 2003

ISBN:

Depósito Legal:

Impreso en España.

Technical Report DSIC-II/14/03,
http://www.dsic.upv.es
Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia, 2003.

Preface

This volume contains the preliminary proceedings of the **Third International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2003**. The final proceedings of WRS 2003 will appear in ENTCS (Electronic Notes in Theoretical Computer Science). The workshop was held as part of the Federated Conference on Rewriting, Deduction and Programming (RDP 2003) in Valencia (Spain) on June 8, 2003, and has been (co-)organized by TU Valencia and TU Wien.

Reduction strategies in rewriting and programming have attracted an increasing attention within the last years. New types of reduction strategies have been invented and investigated, and new results on rewriting / computation under particular strategies have been obtained. Research in this field ranges from primarily theoretical questions about reduction strategies to very practical application and implementation issues. The need for a deeper understanding of reduction strategies in rewriting and programming, both in theory and practice, is obvious, since they bridge the gap between unrestricted general rewriting (computation) and (more deterministic) rewriting with particular strategies (programming). Moreover, reduction strategies provide a natural way to go from operational principles (e.g., graph and term rewriting, narrowing, lambda-calculus) and semantics (e.g., normalization, computation of values, infinitary normalization, head-normalization) to implementations of programming languages. Therefore any progress in this area is likely to be of interest not only to the rewriting community, but also to neighbouring fields like functional programming, functional-logic programming, and termination proofs of algorithms.

WRS 2003 is the third edition in a series of workshops intended to stimulate and promote research and progress in this important field. The workshop wants to provide a forum for the presentation and discussion of new ideas and results, recent developments, new research directions, as well as of surveys on existing knowledge in this area. Furthermore we aim at fostering interaction and exchange between researchers and students actively working on such topics. WRS 2002 took place in Copenhagen, Denmark, on July 21, 2002 and WRS 2001 was held in Utrecht, The Netherlands, on May 26, 2001.

Topics of interest include, but are not restricted to:

- theoretical foundations for the definition and semantic description of reduction strategies,
- strategies in different frameworks (term rewriting, graph rewriting, infinitary rewriting, lambda calculi, higher order rewriting and explicit substitutions, conditional rewriting, rewriting with built-ins, narrowing, constraint solving, etc.) and their application in (equational, functional, functional-logic) programming (languages),
- properties of reduction strategies / computations under strategies (e.g., completeness, computability, decidability, complexity, optimality, (hyper-)normalization, cofinality, fairness, perpetuality, context-freeness, neededness, laziness, eagerness, strictness),
- interrelations, combinations and applications of reduction under different strategies (e.g., equivalence conditions for fundamental properties like termination and confluence, applications in modularity analysis, connections between strategies of different frameworks, etc.),
- program analysis and other semantics-based optimization techniques dealing with reduction strategies,
- rewrite systems / tools / implementations with flexible / programmable strategies as essential concept / ingredient,
- specification of reduction strategies in (real) languages, and
- data structures and implementation techniques for reduction strategies.

The technical program for the workshop includes seven accepted papers and two invited talks by Manuel Clavel (Universidad Complutense de Madrid) and Claude Kirchner (LORIA & INRIA Lorraine). We thank the invited speakers for accepting our invitation.

Regarding the refereeing process we are very grateful to the program committee and to the additional external referees: Thomas Genet, Sebastien Limet, Sylvain Lippi, Sarah Mocas, Monica Nesi, and Vincent van Oostrom. Finally, we would like to extend our sincere thanks to all authors who submitted papers and to all workshop participants.

Bernhard Gramlich and Salvador Lucas
WRS 2003 Program Commitee Co-Chairs

Program Committee

Sergio Antoy	Portland State University (USA)
Zena M. Ariola	University of Oregon (USA)
Roberto Di Cosmo	Université de Paris VII (France)
Jürgen Giesl	RWTH Aachen (Germany)
Bernhard Gramlich (co-chair)	Technische Universität Wien (Austria)
Salvador Lucas (co-chair)	Universidad Politécnica de Valencia (Spain)
Aart Middeldorp	University of Tsukuba (Japan)
Ricardo Peña	Universidad Complutense de Madrid (Spain)
Pierre Réty	Université d'Orléans (France)
Eelco Visser	Universiteit Utrecht (The Netherlands)

Organizing Committee

Bernhard Gramlich	Technische Universität Wien (Austria)
Salvador Lucas	Universidad Politécnica de Valencia (Spain)

Table of Contents

Rewrite Strategies in the Rewriting Calculus

Horatiu Cirstea, Claude Kirchner, Luigi Liquori,
Benjamin Wack

INRIA & LORIA 54506 Vandoeuvre-lès-Nancy, BP 239 Cedex France

{First.Last}@loria.fr

Abstract

This paper presents an overview on the use of the rewriting calculus to express rewrite strategies. We motivate first the use of rewrite strategies by examples in the ELAN language. We then show how this has be modelled in the initial version of the rewriting calculus and how the matching power of this framework facilitates the writing of powerful strategies.

1 Introduction

The notion of strategy appears in all human activities under many different names: strategies, tactics, plans, road-maps, ... to mention just a few. This always means that one wants to describe in a concise manner how to reach a given goal in an environment where many different possibilities can arise and where consequently one needs to search rather than just compute.

In computer science the strategy concept spread-off everywhere, from AI to logic or semantics. When giving a model to a certain situation like planning for the moves of a robot on Mars, the inferences to be made by a theorem prover or the evaluation rules of a programming language, the notion of rewrite rule naturally arises.

Indeed a so called rewrite rule consists of a pattern that describes a schematic situation and the transformation that should be applied to it. Again, a natural model for the pattern is to use term objects but other objects like matrices or graphs could also be considered. So in this paper, we are considering rewrite rules consisting of a pair of terms that model the class of objects to be transformed and the objects in which they should be actually transformed.

What makes the rewrite rule concept so attractive is twofold. First it is schematic: one needs only to describe a schema and not all its instances. Consequently, to decide if a rewrite rule applies requires the use of a procedure to check that the schema is applicable, *i.e.* a matching algorithm. Second it is local: the application of a rewrite rule does not depend on the context and thus can be decided locally.

Therefore, rewrite rules are very convenient to describe schematically and locally the transformations one wants to operate. This is used in many situations like dealing with equational logic, performing inferences in theorem proving, or when performing computations. But because of the local aspect of a rewrite rule, one main ingredient is missing to make the concept operational and allow to answer the questions: which rule should be applied and where? This fundamental ingredient is a *rewrite strategy*.

As such, the notion of rewrite strategy is in everyday use: from normalization strategies to optimal strategies, from leftmost-innermost to lazy ones. Typically, programming languages use call by value or lazy strategies. Automated theorem provers use depth-first or breadth-first proof search strategies. In many cases these strategies are built-in and the users of the programming language or of the prover have no way to adapt them to their specific use.

Since strategies allow us to control the schematically and locally described transformations, it is conceptually and practically fundamental to permit the users to define their own ones. This is what proof assistants like LCF, Coq, ELF permit under the name of tactics and tacticals. In programming languages such a control can be reached using reflexivity like in LISP and Maude or explicitly using a dedicated language like in ELAN or Stratego.

In this last kind of languages, strategy combinators are provided by the language to allow for user-defined strategy definition and execution. How to design such a language and its semantics is the topic of this paper.

Historically, we started in designing the ELAN language whose aim was to easily prototype the combination of deduction and computation mechanisms as needed in constraint solving and automated theorem proving. This leaded us to a first semantics of the language using rewriting logic [22,4] and then to the design of a very general calculus able to take into account all the aspects of the language; we call it the rewriting calculus [7,5]. As we shortly recall later in this paper, the rewriting calculus generalizes the lambda-calculus as abstractions can be made not only on variables but also on terms.

An important feature of the rewriting calculus is its ability to model rewriting strategies. What was needed for this purpose was: (i) to have rewrite rules as primal strategies, (ii) to return explicitly sets of terms to model the fact that equational rewriting can produce sets of results. The explicit handling of result sets also allowed us to detect that a rewrite step can not be applied at some occurrence, since in this case the result set is just empty. Then we had to iterate rewritings in a term and therefore, we naturally added a (iii) fix-point iterator, and since the rewriting calculus embeds the lambda one, we first used standard fix-points like Turing's one. One important feature that escaped from the scope of this initial design was the capability to stop a computation as soon as a first rewrite is successfully applied. We therefore (iv) enriched the rewriting calculus with a strategy called *first*, which returns the results of the first non-failing computation. In this initial setting we were able to describe useful strategies like the innermost or outermost ones. The fact

that a specific failure operator allowed for the definition of the first combinator was showed later and all this is described in the second section of this paper.

From this initial design, we started to simplify the syntax and, on one hand to better understand the potential applications of the calculus (*e.g.* for encoding object calculi) and on the other hand to introduce sophisticated type systems. This leaded us to the discovery of a quite powerful feature of the simply typed version of the rewrite calculus: because of its matching power, the calculus allows indeed some well-typed terms to be non-normalizing. One nice application of this is the definition of fix-points based on the rewriting capability of the calculus that are well typed and very concise. As a consequence this allows us to describe strategies in a very powerful language based on these matching based fix-points. This is described in the last part of this paper together with applications to the description of rewrite strategies and to the combination of such strategies.

The new simplified syntax, together with a deterministic call-by-value operational semantics for the calculus, allows us to detect matching failures more easily. The encoding of the *first* by clever use of an exception catching mechanism is then available. Various options are available for exception handling, depending on the new constructs we introduce and on the semantics we put on them. The specific mechanisms related to exceptions and the behavior of the "modern" *first* are detailed too in the last part of the paper.

2 ELAN: A Language which Deals with Strategies

The ELAN system [20] provides an environment for specifying and prototyping deduction systems in a language based on rules controlled by strategies. Its purpose is to support the design of theorem provers, logic programming languages, constraints solvers and decision procedures and to offer a modular framework for studying their combination.

ELAN takes from functional programming the concept of abstract data types and the function evaluation principle based on rewriting. But rewriting is inherently non-deterministic since several rules can be applied at different positions in a same term, and in ELAN, a computation may have several results. This aspect is taken into account through choice operations and a backtracking capability. One of the main originality of the language is to provide strategy constructors to specify whether a function call returns several, at-least one or only one result. This declarative handling of non-determinism is part of a strategy language allowing the programmer to specify the control on rules application. This is in contrast to many existing rewriting-based languages where the term reduction strategy is hard-wired and not accessible to the designer of an application. The strategy language offers primitives for sequential composition, iteration, deterministic and non-deterministic choices of elementary strategies that are labelled rules. From these primitives, more complex strategies can be expressed. In addition the user can introduce new strategy operators and define them by rewrite rules.

2.1 The Specification Language

The specification formalism provided in the ELAN system is close to the algebraic specification formalism. Signatures introduce sorts of data and operations applied of them. Operators can be defined using a mixfix syntax and can be declared as associative and commutative; the associativity and commutativity axioms are called structural axioms and their application is embedded into the matching process.

```
module boolean
sort Bool; end
operators global
  true      :             Bool;
  false     :             Bool;
  @ and @   : (Bool Bool) Bool;
  @ or  @   : (Bool Bool) Bool;
    not @   : (Bool )     Bool;
end
```

In the algebraic style, the semantics of operations on data is described by a set of first-order formulas. In ELAN, the formulas are a very general form of rewrite rules with conditions and local evaluations. For instance, simple rewrite rules for booleans are given as follows:

```
rules for Bool
  P : Bool;
global
  [] true  or P  => true      end
  [] false or P  => P         end
  [] true  and P => P         end
  [] false and P => false     end
  [] not true    => false     end
  [] not false   => true      end
end
```

The two values `true` and `false` are said irreducible or in normal form. This set of rules is terminating and confluent, which ensures that any boolean formula has a unique normal form. For such systems, it is not needed to specify in which order the rules are applied, nor at which position in the term. The ELAN system adopts in such a case a strategy by default which selects the leftmost and innermost redex at each step. However in many situations, and especially to deal with non-confluent or non-terminating rewrite systems, it is suitable to express which rule to apply.

For specifying this kind of control, ELAN introduces the possibility to name rules, using brackets in front of a rule to enclose its name. In the previous boolean example, the names are unspecified and such rules are said unlabelled. The labelled rules and strategies are applied at the top of a term. In order to apply strategies on subterms, the syntax of rewrite rules has been enriched by local evaluations, used to call strategies and to specify conditions of application. We concentrate in this paper on the application of (basic) rules

and strategies and therefore we do not present here the syntax of the local evaluation constructions.

The capability of specifying control is a main originality of ELAN compared to other specification languages. Let us explain in more details how to build strategies that compute one or several results, specify the order of applied rules, or iterate as much as possible the application of a strategy or a rule on a term.

2.2 Strategies Specification

A labelled rule is the most elementary strategy and is called a primal strategy. The result of applying a rule labelled lab on a term t is a set of terms. Note that there may be several rules with the same label. If no rule labelled lab applies on the term t, the set of results is empty and we say that the rule lab fails. To understand why applying one rule at the top of a term can yield several results, one has to know that local assignments in a rewrite rule can call strategies on subterms. If the strategy in a local assignment has several results, so has the rewrite rule. A labelled rule lab can be considered as the simplest form of a strategy which returns all results of the rule application. As any strategy, lab can also be encapsulated by an operator dc one that returns a non-deterministically chosen result. In that case, dc one(lab) returns at most one result. In addition ELAN provides a few built-in strategy operators that take possibly several strategies as arguments and can be used to build new strategies:

- the concatenation operator denoted ; builds the sequential composition of two strategies S_1 and S_2. The strategy $S_1; S_2$ fails if S_1 fails, otherwise it returns all results (maybe none) of S_2 applied to the results of S_1;
- the dk operator, with a variable arity, is an abbreviation of *dont know choose*. dk$(S_1, \ldots, S_n)$ takes all strategies given as arguments, and returns, for each of them the set of all its results. dk$(S_1, \ldots, S_n)$ fails if all strategies $S_1, \ldots, S_n$ fail;
- the dc operator, with a variable arity, is an abbreviation of *dont care choose*. dc$(S_1, \ldots, S_n)$ selects only one strategy that does not fail among its arguments, say S_i, and returns all its results. dc$(S_1, \ldots, S_n)$ fails if all strategies $S_1, \ldots, S_n$ fail. How to choose S_i is not specified;
- a specific way to choose an S_i is provided by the first operator that selects the first strategy that does not fail among its arguments, and returns all its results. So if S_i is selected, this means that all strategies $S_1, \ldots, S_{i-1}$ have failed. Again first$(S_1, \ldots, S_n)$ fails if all strategies $S_1, \ldots, S_n$ fail;
- if only one result is wanted, one can use the operators first one or dc one that select a non-failing strategy among their arguments (either the first or anyone respectively), and return a non-deterministically chosen result of the selected strategy;
- id is the identity strategy that does nothing, and never fails;

- fail always fails and returns an empty set of results;
- repeat*(S) iterates the strategy S until it fails and then returns the last obtained result. repeat*(S) never fails and terminates only when S fails;
- iterate*(S) is similar to repeat*(S), except that it returns all intermediate results of successive applications of S.

In addition to these primitive strategy operators, the user can define new strategy operators and strategy rules for their evaluation [3].

Example 2.1 If the strategy `dk(x=>x+1,x=>x+2)` is applied to the term a, ELAN provides two results: $a+1$ and $a+2$. When `first(x=>x+1,x=>x+2)` is applied to the same term only the $a+1$ result is obtained. The strategy `first(b=>b+1,a=>a+2)` applied to the term a yields the result $a+2$.

Using non-deterministic strategies, we can explore exhaustively the search space of a given problem and find paths described by some specific properties.

3 The Initial Design of the Rewriting Calculus

We briefly present here the Rho Calculus as introduced in [6,7] and we define several operators allowing us to define different classical rewriting strategies like, for example, innermost or outermost.

3.1 First Version of the Rho Calculus

We consider $\mathcal{X}$ a set of variables and $\mathcal{F} = \bigcup_m \mathcal{F}_m$ a set of ranked function symbols, where for all m, $\mathcal{F}_m$ is the subset of function symbols of arity m. We assume that each symbol has a unique arity *i.e.* that the $\mathcal{F}_m$ are disjoint. We denote by $\mathcal{T}(\mathcal{F},\mathcal{X})$ the set of first-order terms built on $\mathcal{F}$ using the variables in $\mathcal{X}$. The set of basic ρ-terms is inductively defined by the following grammar:

$$\mathcal{T} ::= \mathcal{X} \mid f(\mathcal{T},\ldots,\mathcal{T}) \mid \{\mathcal{T},\ldots,\mathcal{T}\} \mid [\mathcal{T}](\mathcal{T}) \mid \mathcal{T} \twoheadrightarrow \mathcal{T}$$

In what follows we suppose that $s,t,u,v,\ldots \in \mathcal{T}$, and $x,y,\ldots \in \mathcal{X}$, and $f,g,\ldots \in \mathcal{F}$. These symbols can be also indexed.

We adopt a very general discipline for the rewrite rule formation, and we do not enforce any of the standard restrictions often used in the term rewriting community like non-variable left-hand sides or occurrence of the right-hand side variables in the left-hand side. We also consider rewrite rules containing rewrite rules as well as rewrite rule application. For convenience, we consider that the symbols $\{\}$ and $\emptyset$ both represent the empty set. We usually use the notation f instead of $f()$ for a function symbol of arity 0 (*i.e.* a constant). For the terms of the form $\{t_1,\ldots,t_n\}$ we assume, as usually, that the comma is an associative, commutative and idempotent function symbol.

The main intuition behind this syntax is that a rewrite rule is an abstraction, the left-hand side of which determines the bound variables and some contextual information. Having new variables in the right-hand side is just

the ability to have free variables in the calculus. One can notice that the λ-terms [2] and standard first-order rewrite rules [12,1] are clearly objects of this calculus. For example, the λ-term $\lambda x.(y\ x)$ corresponds to the ρ-term $x \twoheadrightarrow [y](x)$ and a rewrite rule in first-order rewriting corresponds to the same rewrite rule in the rewriting-calculus.

We have chosen sets as the data structure for handling the potential non-determinism. A set of terms can be seen as the set of distinct results obtained by applying a rewrite rule to a term. Other choices could be made depending on the intended use of the calculus. For example, if we want to provide all the results of an application, including the identical ones, a multi-set could be used. When the order of the computation of the results is important, lists could be employed. Since in this presentation of the calculus we focus on the possible results of a computation and not on their number or order, sets are used.

Example 3.1 If we consider $\mathcal{F}_0$=$\{a, b, c\}$, $\mathcal{F}_1$=$\{f\}$, $\mathcal{F}_2$=$\{g\}$, $\mathcal{F}$=$\mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{F}_2$ and x, y variables in $\mathcal{X}$, some ρ-terms are:

- $[g(x, y) \twoheadrightarrow f(x)](g(a, b))$; a classical rewrite rule application.
- $[y \twoheadrightarrow [x \twoheadrightarrow x + y](b)]([x \twoheadrightarrow x](a))$; a ρ-term that corresponds to the λ-term $(\lambda y.((\lambda x.x + y)\ b))\ ((\lambda x.x)\ a)$. In the rewrite rule $x \twoheadrightarrow x + y$ the variable y is free but in the rewrite rule $y \twoheadrightarrow [x \twoheadrightarrow x + y](b)$ this variable is bound.
- $[x \twoheadrightarrow [x](x)](x \twoheadrightarrow [x](x))$; the well-known ($\Omega$) λ-term. We will see that the evaluation of this term is not terminating.

Computing the matching substitutions from a ρ-term t to a ρ-term t' is an important parameter of the Rho Calculus.

For a given theory $\mathbb{T}$ over ρ-terms, a $\mathbb{T}$-*match-equation* is a formula of the form $t \ll_{\mathbb{T}} t'$, where t and t' are ρ-terms. A substitution σ is a solution of the $\mathbb{T}$-match-equation $t \ll_{\mathbb{T}} t'$ if $\mathbb{T} \models \sigma(t) \equiv t'$. A $\mathbb{T}$-*matching system* is a conjunction of $\mathbb{T}$-match-equations. A substitution is a solution of a $\mathbb{T}$-matching system P if it is a solution of all the $\mathbb{T}$-match-equations in P. We denote by $\mathcal{F}$ a $\mathbb{T}$-matching system without solution. A $\mathbb{T}$-matching system is called *trivial* when all substitutions are solution of it. We define the function $\mathcal{S}olution$ on a $\mathbb{T}$-matching system $\mathcal{S}$ as returning the set of all $\mathbb{T}$-matches of $\mathcal{S}$ when $\mathcal{S}$ is not trivial and $\{\sigma_{\mathsf{id}}\}$, where σ_{id} is the identity substitution, when $\mathcal{S}$ is trivial. Notice that when the matching system has no solution the function $\mathcal{S}olution$ returns the empty set. By abuse of notation we denote by $\sigma_{(t_1 \ll t_2)}$ the solution of the matching equation $t_1 \ll t_2$.

Since in general we could consider arbitrary theories over ρ-terms, $\mathbb{T}$-matching is in general undecidable, even when restricted to first-order equational theories [17]. But we are interested here in the decidable cases. For example when $\mathbb{T}$ is empty, the syntactic matching substitution from t to t', when it exists, is unique and can be computed by a simple recursive algorithm given for example by G. Huet [16,19].

Since we are dealing with "$\twoheadrightarrow$" as a binder, like for any calculus involving

$$
\begin{array}{lrcl}
(Fire) & [l \twoheadrightarrow r](t) & \rightarrow & \{\sigma_1 r, \ldots, \sigma_n r, \ldots\} \\
 & & & \text{where } \{\sigma_1, \ldots, \sigma_n, \ldots\} = \mathcal{S}olution(l \ll_{\mathbb{T}} t) \\
(Cong) & [f(u_1, \ldots, u_n)](f(v_1, \ldots, v_n)) & \rightarrow & \{f([u_1](v_1), \ldots, [u_n](v_n))\} \\
(Cong\_fail) & [f(u_1, \ldots, u_n)](g(v_1, \ldots, v_m)) & \rightarrow & \emptyset \\
(Distrib) & [\{u_1, \ldots, u_n\}](v) & \rightarrow & \{[u_1](v), \ldots, [u_n](v)\} \\
(Batch) & [v](\{u_1, \ldots, u_n\}) & \rightarrow & \{[v](u_1), \ldots, [v](u_n)\} \\
(Switch_L) & \{u_1, \ldots, u_n\} \twoheadrightarrow v & \rightarrow & \{u_1 \twoheadrightarrow v, \ldots, u_n \twoheadrightarrow v\} \\
(Switch_R) & u \twoheadrightarrow \{v_1, \ldots, v_n\} & \rightarrow & \{u \twoheadrightarrow v_1, \ldots, u \twoheadrightarrow v_n\} \\
(OpOnSet) & f(v_1, \ldots, \{u_1, \ldots, u_m\}, \ldots, v_n) & \rightarrow & \\
 & & & \{f(v_1, \ldots, u_1, \ldots, v_n), \ldots, f(v_1, \ldots, u_m, \ldots, v_n)\} \\
(Flat) & \{u_1, \ldots, \{v_1, \ldots, v_n\}, \ldots, u_m\} & \rightarrow & \{u_1, \ldots, v_1, \ldots, v_n, \ldots, u_m\}
\end{array}
$$

Fig. 1. The evaluation rules of the Rho Calculus

binders (as the λ-calculus), α-conversion should be used to obtain a correct substitution calculus and the first-order substitution (called here *grafting*) is not directly suitable for the Rho Calculus. We consider the usual notions of α-conversion and higher-order substitution as defined, for example, in [13].

The set of evaluation rules of the Rho Calculus is presented in Figure 1, where we assume we are given a theory $\mathbb{T}$ over ρ-terms having a decidable matching problem.

As defined by the evaluation rule $(Fire)$, the application of a rewrite rule at the root position of a term is accomplished by matching the left-hand side of the rewrite rule on the term and returning the appropriately instantiated right-hand side. When the matching yields a failure represented by an empty set of substitutions, the result of the application of the rule $(Fire)$ is the empty set. This rule, like all the evaluation rules of the calculus, can be applied at any position of a ρ-term.

We should point out that, as in λ-calculus, an application can always be evaluated. But, unlike in λ-calculus, the set of results can be empty. More generally, when matching modulo a theory $\mathbb{T}$, the set of resulting matches may be empty, a singleton (as in the empty theory), a finite set (as for associativity-commutativity) or infinite (see [14]). We have thus chosen to represent the result of a rewrite rule application to a term as a set. An empty set means that the rewrite rule $l \twoheadrightarrow r$ fails to apply to t in the sense of a matching failure between l and t.

In order to push rewrite rule application deeper into terms, we introduce the two $(Congruence)$ evaluation rules. When we have the same head symbol for the two terms of the application $[u](v)$ the arguments of the term u are applied on those of the term v argument-wise. If the head symbols are not the same, an empty set is obtained.

The rules ($Distrib$) and ($Batch$) describe the interaction between the application and the set operators, the rules ($Switch_L$) and ($Switch_R$) describe the interaction between the abstraction and the set operators, the rule ($OpOnSet$) describe the interaction between the symbols of the signature and the set operators.

We usually care about the set of results obtained by reducing the redexes and not about the exact trace of the reduction leading to these results. We use the evaluation rule ($Flat$) that flattens the sets and eliminates the (nested) set symbols. Notice that this implies that failure (the empty set) is *not* strictly propagated on sets.

The strategy guiding the application of the evaluation rules, is crucial for obtaining good properties for the Rho Calculus such as the confluence and it has been shown [7] that if the rule ($Fire$) is applied under no conditions at any position of a ρ-term, confluence does not hold.

The main reason for the confluence failure comes from the undesirable matching failures due to terms that are not completely evaluated or not instantiated. On the other hand, we can have sets with more than one element that can lead to undesirable results in a non-linear context or empty sets that are not strictly propagated. The reasons for the non-confluence of the calculus are explained in [7] and a solution is proposed for obtaining a confluent calculus. The confluent strategy can be given explicitly or as a condition on the application of the rule ($Fire$).

Since the sets (empty or having more than one element) are the main cause of the non-confluence of the calculus, a natural strategy consists in reducing the application of a rewrite rule by respecting the following steps: instantiate and reduce the argument of the application, push out the resulting set braces by distributing them in the terms and only when none of the previous reductions is possible, use the evaluation rule ($Fire$). We can easily express this strategy by imposing a simple condition for the application of the evaluation rule ($Fire$): the evaluation rule ($Fire$) is applied to a redex $[l \twoheadrightarrow r](t)$ only if the terms l, t are first order ground terms. Less restrictive strategies guaranteeing the confluence can be defined [7].

3.2 Defining Strategy Operators in Rho Calculus

We have shown [7] that for any reduction in a rewrite theory there exists a corresponding reduction in the Rho Calculus: if the term u reduces to the term v in a rewrite theory $\mathcal{R}$ we can build a ρ-term $\xi_{\mathcal{R}}(u)$ that reduces to the term $\{v\}$. The method used for constructing the term $\xi_{\mathcal{R}}(u)$ depends on all the reduction steps from u to v in the theory $\mathcal{R}$: $\xi_{\mathcal{R}}(u)$ is a representation in the Rho Calculus of the derivation trace. We can go further on and to give a method for constructing a term $\xi_{\mathcal{R}}(u)$ without knowing a priori the derivation from u to v. Hence we want to answer to the following question: *"Given a rewrite theory $\mathcal{R}$ does there exist a ρ-term $\xi_{\mathcal{R}}$ such that for any term u if u normalizes to the term v in the rewrite theory $\mathcal{R}$ then $[\xi_{\mathcal{R}}](u)$ ρ-reduces to a*

set containing the term v?" The definition of normalization strategies is in general done at the *meta-level* while the Rho Calculus allows us to represent such derivations at the *object level*.

When computing the normal form of a term u *w.r.t.* a rewrite system $\mathcal{R}$, the rewrite rules are applied *repeatedly* at *any position* of a term u until no rule from $\mathcal{R}$ is *applicable*. Hence, the ingredients needed for defining such a strategy are:

- an iteration operator applying *repeatedly* a set of rewrite rules;
- a term traversal operator applying a rewrite rule at *any position* of a term;
- an operator testing if a set of rewrite rules is *applicable* to a term.

In what follows we describe how the operators with the above functionalities can be defined in the Rho Calculus. We start with some auxiliary operators and afterwards, we introduce the ρ-operators that correspond to the functionalities listed above.

3.2.1 Some Auxiliary Operators

First, we define three auxiliary operators that will be used in the next sections. These operators are just aliases used to define more complex ρ-terms and are used for giving more compact and clear definitions for the recursion operators. On the other hand, these operators correspond to the homonymous ones defined in ELAN.

The first of these operators is the *identity* (denoted id) that applied to any ρ-term t evaluates to the singleton containing this term,*i.e.* $[id](t) \longrightarrow_\rho \{t\}$:

$$id \triangleq x \twoheadrightarrow x.$$

In a similar way we can define the strategy $fail$ which always fails, *i.e.* applied to any term, leads to $\emptyset$:

$$fail \triangleq x \twoheadrightarrow \emptyset.$$

The third one is the binary operator ";" that represents the sequential application of two ρ-terms. A ρ-term of the form $[u; v](t)$ represents the application of the term v to the result of the application of u to t:

$$u; v \triangleq x \twoheadrightarrow [v]([u](x)).$$

3.2.2 The "first" Operator

We introduce now a new operator whose role is to select between its arguments the first one that applied to a given ρ-term does not evaluate to $\emptyset$. If all the arguments evaluate to $\emptyset$ then the final result of the evaluation is $\emptyset$. The evaluation rules describing the *first* operator and the auxiliary operator $\langle \_, \ldots, \_ \rangle$ are presented in Figure 2. We will show later that this operator can be expressed in more recent versions of the Rho Calculus. The application of a ρ-term $first(s_1, \ldots, s_n)$ to a term t returns the result of the first "successful" application of one of its arguments to the term t. Hence, if $[s_i](t)$ evaluates to $\emptyset$

$$
\begin{array}{lrcl}
(First) & [\mathit{first}(s_1,\ldots,s_n)](t) & \to & \langle [s_1](t),\ldots,[s_n](t)\rangle \\
(FirstFail) & \langle \emptyset, t_1,\ldots,t_n\rangle & \to & \langle t_1,\ldots,t_n\rangle \\
(FirstSuccess) & \langle t, t_1,\ldots,t_n\rangle & \to & \{t\} \\
 & t \text{ contains no redexes, no free variables and is not } \emptyset & & \\
(FirstSingle) & \langle\rangle & \to & \emptyset
\end{array}
$$

Fig. 2. The *first* operator

for $i = 1,\ldots,k-1$, and $[s_k](t)$ does not evaluate to $\emptyset$, then $[\mathit{first}(s_1,\ldots,s_n)](t)$ evaluates to the same term as the term $[s_k](t)$. If the evaluation of the terms $[s_i](t)$, $i = 1,\ldots,k-1$, leads to $\emptyset$ and the evaluation of $[s_k](t)$ does not terminate then the evaluation of the term $[\mathit{first}(s_1,\ldots,s_n)](t)$ does not terminate.

Example 3.2 The non-deterministic application of one of the rules $a \twoheadrightarrow b$, $a \twoheadrightarrow c$, $a \twoheadrightarrow d$ to the term a is represented in the Rho Calculus by the application $[\{a \twoheadrightarrow b, a \twoheadrightarrow c, a \twoheadrightarrow d\}](a)$. This last ρ-term is reduced to the term $\{b,c,d\}$ which represents a non-deterministic choice among the three terms. If we want to apply the above rules in a deterministic way and in the specified order, we use the ρ-term $[\mathit{first}(a \twoheadrightarrow b, a \twoheadrightarrow c, a \twoheadrightarrow d)](a)$ with, for example, the reduction:

$$
\begin{array}{ll}
 & [\mathit{first}(a \twoheadrightarrow b, a \twoheadrightarrow c, a \twoheadrightarrow d)](a) \\
\longrightarrow_{First} & \langle [a \twoheadrightarrow b](a), [a \twoheadrightarrow c](a), [a \twoheadrightarrow d](a)\rangle \\
\longrightarrow_{Fire} & \langle \{b\}, [a \twoheadrightarrow c](a), [a \twoheadrightarrow d](a)\rangle \\
\longrightarrow_{FirstSuccess} & \{\{b\}\} \\
\longrightarrow_{Flat} & \{b\}
\end{array}
$$

We can notice that even if all the rewrite rules can be applied successfully (*i.e.* no empty set) to the term a, the final result is given by the first tried rewrite rule.

3.2.3 Term Traversal Operators

Let us now define operators that apply a ρ-term at some position of another ρ-term. The first step is the definition of two operators that push the application of a ρ-term one level deeper on another ρ-term. This is already possible in the Rho Calculus due to the rule *Cong* but we want to define a generic operator that applies a ρ-term r to the sub-terms u_i, $i = 1\ldots n$, of a term of the form $F(u_1,\ldots,u_n)$ independently on the head symbol F.

To this end, we define two term traversal operators, $\Phi(r)$ and $\Psi(r)$, whose behavior is described by the rules in Figure 3. These operators are inspired by the operators of the *System S* described in [24]. The application of the ρ-term $\Phi(r)$ to a term $t = f(u_1,\ldots,u_n)$ results in the successful application of the term r to one of the terms u_i. More precisely, r is applied to the first u_i, $i = 1,\ldots,n$ such that $[r](u_i)$ does not evaluate to the empty set. If there *exists no* such u_i and in particular, if t is a function with no argu-

$$(TraverseSeq)\ \ [\Phi(r)](f(u_1,\ldots,u_n)) \quad \rightarrow \quad \langle\{f([r](u_1),\ldots,u_n)\},\ldots,\{f(u_1,\ldots,[r](u_n))\}\rangle$$

$$(TraversePar)\ \ [\Psi(r)](f(u_1,\ldots,u_n)) \quad \rightarrow \quad \{f([r](u_1),\ldots,[r](u_n))\}$$

Fig. 3. Two traversal operators

ments (t is a constant), then the term $[\Phi(r)](t)$ reduces to the empty set: $[\Phi(r)](c) \longrightarrow_{TraverseSeq} \langle\{\}\rangle \longrightarrow_{FirstFail} \langle\rangle \longrightarrow_{FirstSingle} \emptyset$. When the ρ-term $\Psi(r)$ is applied to a term $t = f(u_1,\ldots,u_n)$ the term r is applied to all the arguments u_i, $i = 1,\ldots,n$ if *for all* i, $[r](u_i)$ does not evaluate to $\emptyset$. If there exists an u_i such that $[r](u_i)$ reduces to $\emptyset$, then the result is the empty set. If we apply $\Psi(r)$ to a constant c, since there are no sub-terms the term $[\Psi(r)](c)$ reduces to $\{c\}$: $[\Psi(r)](c) \longrightarrow_{TraversePar} \{c\}$. If we consider a Rho Calculus with a finite signature $\mathcal{F}$ and if we denote by $\mathcal{F}_0$=$\{c_1,\ldots,c_n\}$ the set of constant function symbols and by $\mathcal{F}_+$=$\{f_1,\ldots,f_m\}$ the set of function symbols with arity at least one, the two term traversal operators can be expressed in the Rho Calculus by some appropriate ρ-terms. If the following two definitions are considered

$$\Phi'(r) \triangleq \mathit{first}(f_1(r,id,\ldots,id),\ldots,f_1(id,\ldots,id,r),\ldots, f_m(r,id,\ldots,id),\ldots,f_m(id,\ldots,id,r))$$

$$\Psi(r) \triangleq \{c_1,\ldots,c_n,f_1(r,\ldots,r),\ldots,f_m(r,\ldots,r)\}$$

with $c_i \in \mathcal{F}_0$, $i = 1,\ldots,n$, and $f_j \in \mathcal{F}_+$, $j = 1,\ldots,m$, we obtain:

$$[\Phi'(r)](f_k(u_1,\ldots,u_p)) \xrightarrow{*} \langle\{f_k([r](u_1),\ldots,u_p)\},\ldots, \{f_k(u_1,\ldots,[r](u_p))\},\emptyset,\ldots,\emptyset\rangle$$

and

$$[\Psi(r)](f_k(u_1,\ldots,u_p)) \xrightarrow{*} \{f_k([r](u_1),\ldots,[r](u_p))\}$$

The operator Φ' does not correspond exactly to the definition from the Figure 3 but a similar result is obtained when applying the terms $\Phi(r)$ and $\Phi'(r)$ to a term $f_k(u_1,\ldots,u_p)$. We can thus state that the term traversal operators Φ and Ψ can be expressed in the Rho Calculus.

3.2.4 Iterators

The definition of the evaluation (normalization) strategies as, for example, *top-down* or *bottom-up*, is based on the application of one term to the top position or to the deepest positions of another term.

For the moment, we have the possibility of applying a ρ-term r either to one or all the arguments u_i of a ρ-term $t = f(u_1,\ldots,u_n)$, or to the sub-terms of t at an explicitly specified depth. But the depth of a term is not known *a priori* and thus, we cannot apply a term r to the deepest positions of a term t. If we want to apply the term r to the sub-terms at the maximum depth of a term t we must define a recursive operator which reiterates the application of the $\Phi(r)$ and $\Psi(r)$ terms and thus, pushes the application deeper into terms.

We start by presenting the ρ-term used for describing recursive applications

in the Rho Calculus. Starting from the Turing fixed-point combinator ([23]) we define the ρ-term $\Theta = [A](A)$ with

$$A = x \twoheadrightarrow (y \twoheadrightarrow [y]([[x](x)](y))).$$

and for a given term G we obtain the reduction:

$$[\Theta](G) \xrightarrow{*}_{\rho} \{[G]([\Theta](G))\} \qquad (FixPoint)$$

Since the ρ-term $[\Theta](G)$ can obviously lead to infinite reductions, a strategy should be used in order to obtain termination and thus the desired behavior. If Θ is considered as an independent ρ-term with the behavior described by an evaluation rule corresponding to the reduction $(FixPoint)$, the strategy suggested previously could be easily implemented. Alternatively, an *outermost* strategy can be used. It is clear that such a strategy prevents only the infinite reductions due to the operator Θ, but it cannot ensure the termination of the untyped Rho Calculus.

As we mentioned previously, the main goal of this section is the representation of normalization strategies by ρ-terms and thus, we want to describe the application of a term r to all the positions of another term t. Therefore, we must define the appropriate term G that propagates the application of a ρ-term in the sub-terms of another ρ-term.

3.2.5 Top-down and Bottom-up Applications

Using the term traversal operator Φ we can define ρ-terms that apply a specific term only at one position of a ρ-term in a *bottom-up* or *top-down* way. We will see that the operators built using the Φ operator are convenient for the construction of normalization operators.

The ρ-term used in the *bottom-up* case is

$$H_{bu}(r) \triangleq f \twoheadrightarrow (x \twoheadrightarrow [first(\Phi(f), r)](x))$$

and we define an operator that applies only once a ρ-term in a *bottom-up* way,

$$Once_{bu}(r) \triangleq [\Theta](H_{bu}(r)).$$

The term $[Once_{bu}(r)](t) \triangleq [[\Theta](H_{bu}(r))](t)$ can lead to an infinite reduction if an appropriate strategy is not employed. Once again, we can use an outermost strategy in order to obtain the desired behavior.

Example 3.3 The application $[Once_{bu}(a \twoheadrightarrow b)](a)$ is reduced to the term $\{\langle[(a \twoheadrightarrow b)](a)\rangle\}$ and thus, to $\{b\}$. The application of the rule $a \twoheadrightarrow b$ to the leftmost-innermost position of a term $g(a, f(a))$ is represented by the term $[Once_{bu}(a \twoheadrightarrow b)](g(a, f(a)))$ and the corresponding evaluation is presented below:

$$\begin{aligned} & [Once_{bu}(a \twoheadrightarrow b)](g(a, f(a))) \\ \xrightarrow{*}_{\rho} & \{\langle\langle g([Once_{bu}(a \twoheadrightarrow b)](a), f(a)), g(a, [Once_{bu}(a \twoheadrightarrow b)](f(a)))\rangle, \\ & \qquad\qquad [a \twoheadrightarrow b](g(a, f(a)))\rangle\} \end{aligned}$$

$$
\begin{array}{llll}
(Repeat_{success}) & [repeat(r)](t) & \rightarrow & [repeat(r)]([r](t)) \\
 & & & \mathsf{if}\ [r](t) \text{ is not reduced to } \emptyset \\
(Repeat_{fail}) & [repeat(r)](t) & \rightarrow & t \\
 & & & \mathsf{if}\ [r](t) \text{ is reduced to } \emptyset
\end{array}
$$

Fig. 4. The operator *repeat*

$\xrightarrow{*}_\rho \{\langle\langle g(\{b\}, f(a)), g(a, [Once_{bu}(a \twoheadrightarrow b)](f(a)))\rangle, [a \twoheadrightarrow b](g(a, f(a)))\rangle\}$
$\xrightarrow{*}_\rho \{\langle\{g(b, f(a))\}, [a \twoheadrightarrow b](g(a, f(a)))\rangle\}$
$\xrightarrow{*}_\rho \{g(b, f(a))\}$

If we want to define an operator that applies a specific term only at one position of a ρ-term in a *top-down* way we should use the ρ-term

$$H_{td}(r) \triangleq f \twoheadrightarrow (x \twoheadrightarrow [first(r, \Phi(f))](x))$$

and we obtain immediately the operator $Once_{td}$,

$$Once_{td}(r) \triangleq [\Theta](H_{td}(r)).$$

In the case of an application $[Once_{td}(r)](t)$, the application of the term r is first tried at the top position of t and in the case of a failure, r is applied deeper in the term t.

3.2.6 Repetition and Normalization Operators

In the previous sections we have defined operators that describe the application of a term at some position of another term (*e.g.* $Once_{bu}$) and operators that allow us to recover from failing evaluations ($first$).

Now we want to define an operator that applies repeatedly a given strategy r to a ρ-term t. We call it *repeat* and its behavior can be described by the evaluation rules presented in Figure 4. Hence, we need an operator similar to the *repeat* one, that stores the last non-failing result and when no further application is possible returns this result. We use once again the fixed-point operator presented in the previous section and we define the ρ-term

$$J(r) \triangleq f \twoheadrightarrow (x \twoheadrightarrow [first(r; f,\ id)](x))$$

that is used for describing the *repeat* operator

$$repeat(r) \triangleq [\Theta](J(r)).$$

We should not forget that we assume here that an application $[u](v)$ is reduced by applying the evaluation rules at the top position, then to its argument v and only afterwards to the term u.

Example 3.4 The repeated application of the rewrite rules $a \twoheadrightarrow b$ and $b \twoheadrightarrow c$ on the term a is represented by the term $[repeat(\{a \twoheadrightarrow b, b \twoheadrightarrow c\})](a)$ that

evaluates as follows:

$$
\begin{array}{ll}
 & [repeat(\{a \twoheadrightarrow b, b \twoheadrightarrow c\})](a) \\
\xrightarrow{*} & \{\langle [repeat(\{a \twoheadrightarrow b, b \twoheadrightarrow c\})]([\{a \twoheadrightarrow b, b \twoheadrightarrow c\}](a)), [id](a)\rangle\} \\
\xrightarrow{*} & \{\langle [repeat(\{a \twoheadrightarrow b, b \twoheadrightarrow c\})](\{b\}), [id](a)\rangle\} \\
\xrightarrow{*} & \{\langle\{\langle [repeat(\{a \twoheadrightarrow b, b \twoheadrightarrow c\})]([\{a \twoheadrightarrow b, b \twoheadrightarrow c\}](b)), [id](b)\rangle\}, [id](a)\rangle\} \\
\xrightarrow{*} & \{\langle\{\langle [repeat(\{a \twoheadrightarrow b, b \twoheadrightarrow c\})](\{c\}), [id](b)\rangle\}, [id](a)\rangle\} \\
\xrightarrow{*} & \{\langle\{\langle\{\langle [repeat(\{a \twoheadrightarrow b, b \twoheadrightarrow c\})]([\{a \twoheadrightarrow b, b \twoheadrightarrow c\}](c)), [id](c)\rangle\}, [id](b)\rangle\}, [id](a)\rangle\} \\
\xrightarrow{*} & \{\langle\{\langle\{\langle [repeat(\{a \twoheadrightarrow b, b \twoheadrightarrow c\})](\emptyset), \{c\}\rangle\}, [id](b)\rangle\}, [id](a)\rangle\} \\
\xrightarrow{*} & \{\langle\{\langle\{\langle \emptyset, \{c\}\rangle\}, [id](b)\rangle\}, [id](a)\rangle\} \\
\xrightarrow{*} & \{\langle\{\langle\{c\}, [id](b)\rangle\}, [id](a)\rangle\} \\
\xrightarrow{*} & \{\langle\{\{c\}\}, [id](a)\rangle\} \\
\xrightarrow{*} & \{c\}
\end{array}
$$

Using the above operators it is easy to define some specific normalization strategies. For example, the *innermost* strategy is defined by

$$im(r) \triangleq repeat(Once_{bu}(r))$$

and an *outermost* strategy is defined by

$$om(r) \triangleq repeat(Once_{td}(r)).$$

We have now all the ingredients needed for describing the normalization of a term t in a rewrite theory $\mathcal{R}$. The term $\xi_{\mathcal{R}}(u)$ described at the beginning of this section can be defined using the $im(\mathcal{R})$ or $om(\mathcal{R})$ operators and thus, we can represent the normalization of a term u *w.r.t.* a rewriting theory $\mathcal{R}$ by the ρ-terms

$$\xi_{\mathcal{R}}(u) \triangleq [im(\mathcal{R})](u)$$

or

$$\xi_{\mathcal{R}}(u) \triangleq [om(\mathcal{R})](u).$$

Example 3.5 If we denote by $\mathcal{R}$ the set of rules $\{a \twoheadrightarrow b, g(x, f(x)) \twoheadrightarrow x\}$, we represent by $[im(\mathcal{R})](g(a, f(a)))$ the leftmost-innermost normalization of the term $g(a, f(a))$ according to the set of rules $\mathcal{R}$ and the following derivation is obtained:

$$
\begin{array}{ll}
 & [im(\mathcal{R})](g(a, f(a))) \\
\triangleq & [repeat(Once_{bu}(\mathcal{R}))](g(a, f(a))) \\
\xrightarrow{*} & \{\langle [repeat(Once_{bu}(\mathcal{R}))]([Once_{bu}(\mathcal{R})](g(a, f(a)))), [id](g(a, f(a)))\rangle\} \\
\xrightarrow{*} & \{\langle [repeat(Once_{bu}(\mathcal{R}))](\{g(b, f(a))\}), [id](g(a, f(a)))\rangle\} \\
\xrightarrow{*} & \{\langle\{[repeat(Once_{bu}(\mathcal{R}))](g(b, f(a)))\}, [id](g(a, f(a)))\rangle\} \\
\xrightarrow{*} & \{\langle\{\{\langle [repeat(Once_{bu}(\mathcal{R}))]([Once_{bu}(\mathcal{R})](g(b, f(a)))), \\
 & \quad [id](g(b, f(a)))\rangle\}\}, [id](g(a, f(a)))\rangle\} \\
\xrightarrow{*} & \{\langle\{\langle [repeat(Once_{bu}(\mathcal{R}))](\{g(b, f(b))\}), \\
 & \quad [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\} \\
\xrightarrow{*} & \{\langle\{\langle\{\langle\{[repeat(Once_{bu}(\mathcal{R}))]([Once_{bu}(\mathcal{R})](g(b, f(b))))\},
\end{array}
$$

$$
\begin{aligned}
&\quad [id](g(b, f(b)))\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\} \\
&\xrightarrow{*} \{\langle\{\langle\{\langle\{[repeat(Once_{bu}(\mathcal{R}))](\{b\})\}, \\
&\quad [id](g(b, f(b)))\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\} \\
&\xrightarrow{*} \{\langle\{\langle\{\langle\{\langle[repeat(Once_{bu}(\mathcal{R}))]([Once_{bu}(\mathcal{R})](b)), [id](b)\rangle, \\
&\quad [id](g(b, f(b)))\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\} \\
&\xrightarrow{*} \{\langle\{\langle\{\langle\{\langle[repeat(Once_{bu}(\mathcal{R}))](\emptyset), [id](b)\rangle, \\
&\quad [id](g(b, f(b)))\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\} \\
&\xrightarrow{*} \{\langle\{\langle\{\langle\{\langle\emptyset, [id](b)\rangle, \\
&\quad [id](g(b, f(b)))\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\} \\
&\xrightarrow{*} \{\langle\{\langle\{\langle\{\{\{b\}\}\}, [id](g(b, f(b)))\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\} \\
&\xrightarrow{*} \{\langle\{\langle\{\{\{b\}\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\} \\
&\xrightarrow{*} \{\langle\{\{\{b\}\}, [id](g(a, f(a)))\rangle\} \\
&\xrightarrow{*} \{\langle\{\{b\}, [id](g(a, f(a)))\rangle\} \\
&\xrightarrow{*} \{b\}
\end{aligned}
$$

Given a term u, if the rewriting theory $\mathcal{R}$ is not confluent then, the result of the reduction of the term $[im(\mathcal{R})](u)$ is a set representing all the possible results of the reduction of the term u in the rewriting theory $\mathcal{R}$. Each of the elements of the result set represents the result of a reduction in the rewriting theory $\mathcal{R}$ for a given application order of the rewrite rules in $\mathcal{R}$.

Example 3.6 Let us consider the set $\mathcal{R} = \{a \twoheadrightarrow b, a \twoheadrightarrow c, g(x, x) \twoheadrightarrow x\}$ of non-confluent rewrite rules. The term $[im(\mathcal{R})](g(a, a))$ representing the *innermost* normalization of the term $g(a, a)$ according to the set of rewrite rules $\mathcal{R}$ is reduced to $\{b, g(c, b), g(b, c), c\}$. The term $[om(\mathcal{R})](g(a, a))$ representing the *outermost* normalization is reduced to $\{b, c\}$.

We have now all the ingredients necessary to describe in a concise way the normalization process induced by a rewrite theory. Of course, the standard properties of termination and confluence of the rewrite system will allow us to get uniqueness of the result. Our approach differs from this and we define this normalization even in the case where there is no unique normal form or where termination is not warranted. This is why in general we do not get termination or uniqueness of the normal form.

3.3 Encoding "first" in Small Step

As we have seen in the previous section, the *first* operator plays a crucial role in the definition of the various strategies. One can wonder thus if this operator can be expressed using the basic operators of the Rho Calculus.

When trying to do this, the main difficulty is the ambivalent use of the empty set. For example, when applying on the term b the rewrite rule $a \twoheadrightarrow \emptyset$ that rewrites the constant a into the empty set, the evaluation rule of the calculus returns $\emptyset$ because *the matching against b fails*: $[a \twoheadrightarrow \emptyset](b) \quad \longrightarrow_\rho \quad \emptyset$. But it is also possible to explicitly rewrite an object into the empty set like in $[a \twoheadrightarrow \emptyset](a) \quad \xrightarrow{*}_\rho \quad \emptyset$, and in this case the result is also the empty set because

$$
\begin{array}{lrcl}
(Fire) & [l \twoheadrightarrow r](t) & \to & \{\sigma r\} \\
 & & & \text{where } \{\sigma\} = \mathcal{S}olution(l \ll t) \\
(Congr) & [f(t_1,\ldots,t_n)](f(u_1,\ldots,u_n)) & \to & \{f([t_1](u_1),\ldots,[t_n](u_n))\} \\
(CongrFail) & [f(t_1,\ldots,t_n)](g(u_1,\ldots,u_n)) & \to & \{\bot\} \\
(Distrib) & [\{u_1,\ldots,u_n\}](v) & \to & \text{if } n>0,\ \{[u_1](v),\ldots,[u_n](v)\} \\
 & & & \text{if } n=0,\ \{\bot\} \\
(Batch) & [v](\{u_1,\ldots,u_n\}) & \to & \text{if } n>0,\ \{[v](u_1),\ldots,[v](u_n)\} \\
 & & & \text{if } n=0,\ \{\bot\} \\
(SwitchR) & u \twoheadrightarrow \{v_1,\ldots,v_n\} & \to & \{u \twoheadrightarrow v_1,\ldots,u \twoheadrightarrow v_n\} \\
 & & & \text{if } n>0 \\
(OpSet) & h(v_1,\ldots,\{u_1,\ldots,u_n\},\ldots,v_m) & \to & \{h(v_1,\ldots,u_i,\ldots,v_m)\}_{1\leqslant i\leqslant n} \\
 & & & \text{if } n>0 \text{ and } h\in\mathcal{F}_\varepsilon \\
(Flat) & \{u_1,\ldots,\{v_1,\ldots,v_m\},\ldots,u_n\} & \to & \{u_1,\ldots,v_1,\ldots,v_m,\ldots,u_n\} \\
(AppBotR) & [v](\bot) & \to & \{\bot\} \\
(AppBotL) & [\bot](v) & \to & \{\bot\} \\
(AbsBotR) & v \twoheadrightarrow \bot & \to & \{\bot\} \\
(OpBot) & f(t_1,\ldots,t_k,\bot,t_{k+1},\ldots,t_n) & \to & \{\bot\} \\
(FlatBot) & \{t_1,\ldots,\bot,\ldots,t_n\} & \to & \{t_1,\ldots,t_n\} \\
 & & & \text{if } n>0 \\
(Exn) & exn(t) & \to & \{t\} \\
 & & & \{t\}\downarrow\ \neq\{\bot\} \text{ and } \mathcal{FV}(t\downarrow)=\emptyset
\end{array}
$$

Fig. 5. The evaluation rules of the ρ_ε-calculus

the rewrite rule *explicitly introduces* it.

It becomes then clear that one should avoid the ambivalent use of the empty set and introduce an *explicit distinction between failure and the empty set of results*. In fact, by making this distinction, we add to the matching power of the rewriting calculus a *catching power*, which allows to describe, in a rewriting style, exception mechanisms, and therefore to express easily convenient and elaborated strategies needed when computing or proving. An extended version of the Rho Calculus, denoted ρ_ε-calculus, was proposed in [15] where a single meaning is given to the empty set: to represent *only* the empty set of terms. Consequently, the application of a term to the empty set should lead to failure (*i.e.* $[v](\emptyset) \xrightarrow{*}_\rho \{\bot\}$) and the application of the empty set to a term should lead too to failure (*i.e.* $[\emptyset](v) \xrightarrow{*}_\rho \{\bot\}$). On the other hand, provided all the t_i are in normal form, a term like $f(t_1,\ldots,\emptyset,\ldots,t_n)$ will be

considered as a normal form and not rewritable to $\{\bot\}$. This is particularly useful to keep the first class status of the empty set. Moreover, we would like a ρ-term of the form $u \twoheadrightarrow \emptyset$ to be in normal form since it allows us to express a void function. The terms of the ρ_ε-calculus are defined by:

$$\mathcal{T} ::= \mathcal{X} \mid f(\mathcal{T}, \ldots, \mathcal{T}) \mid \{\mathcal{T}, \ldots, \mathcal{T}\} \mid [\mathcal{T}](\mathcal{T}) \mid \mathcal{T} \twoheadrightarrow \mathcal{T} \mid \bot \mid exn(\mathcal{T})$$

In ρ_ε-calculus, we generally consider only first-order terms in the left hand side of an abstraction. The patterns are thus built on variables and using symbols from $\mathcal{F}_\varepsilon \triangleq \mathcal{F} \cup \{exn, \bot\}$. The evaluation rules of ρ_ε-calculus are described in Figure 5. For simplicity, in what follows we restrict to syntactic matching.

The ρ_ε-calculus is neither confluent nor strict. To obtain a confluent calculus, the evaluation mechanism must be tamed by a suitable strategy. This can be described in two ways: either by an independent strategy, or by technical conditions directly added to the $(Fire)$ and (Exn) rules as detailed in [15]. Once a confluent strategy is defined, a *shallow* encoding of the *first* operator in the ρ_ε-calculus can be given, *i.e.* we can define a term to express it.

The *first* is an operator whose role is to select between its arguments the first one that, applied to a given ρ-term, does not evaluate to $\{\bot\}$. This is something quite difficult to express in the Rho Calculus. We propose to represent the term $[\mathit{first}(t_1, \ldots, t_n)](r)$ by a set of n terms, denoted $\{u_1, \ldots, u_n\}$, with the property that every term u_i can be reduced to $\{\bot\}$ except perhaps one (say u_l). In other words, the initial set is reduced to a singleton, containing the normal from of u_l, by successively reducing each u_i $(i \neq l)$ to $\{\bot\}$ and then using the $(FlatBot)$ rule. To express the *first* in the ρ_ε-calculus, we can use its two main trumps: the matching and the failure catching. Each u_i is expressed using the $i-1$ first terms: if all u_j $(j < i)$ are reduced to $\{\bot\}$, then u_i leads to $[t_i](r)$ else u_i leads to $\{\bot\}$ by a rule application failure. So, we define each u_i by

$$\begin{aligned}
u_1 &\triangleq [t_1](x)\\
u_2 &\triangleq \big[exn(\bot) \twoheadrightarrow [t_2](x)\big]\big(exn(u_1)\big)\\
u_3 &\triangleq \big[exn(\bot) \twoheadrightarrow [exn(\bot) \twoheadrightarrow [t_3](x)](exn(u_2))\big]\big(exn(u_1)\big)\\
u_4 &\triangleq \big[exn(\bot) \twoheadrightarrow [exn(\bot) \twoheadrightarrow [exn(\bot) \twoheadrightarrow [t_4](x)](exn(u_3))](exn(u_2))\big]\big(exn(u_1)\big)\\
&\vdots\\
u_n &\triangleq [exn(\bot) \twoheadrightarrow [exn(\bot) \twoheadrightarrow [\ldots \twoheadrightarrow [t_n](x)](exn(u_{n-1}))\ldots](exn(u_2))](exn(u_1))
\end{aligned}$$

and we define *first* by:

$$\mathit{first}(t_1, \ldots, t_n) \triangleq x \twoheadrightarrow \{u_1, \ldots, u_n\}.$$

Example 3.7 Using this *first* operator and the failure catching mechanism,

we can express the evaluation scheme:

if M is evaluated to the failure term
then evaluate the term P_{fail}
else evaluate the term P_{nor}.

thanks, for example, to the term $[\mathit{first}(exn(\bot) \twoheadrightarrow P_{fail}, x \twoheadrightarrow P_{nor})](exn(M))$.

(i) if $M \xrightarrow{*}_{\rho_\varepsilon} \{\bot\}$, we have the following reduction:

$$\begin{array}{ll} & \big[\mathit{first}\big(exn(\bot) \twoheadrightarrow P_{fail}, x \twoheadrightarrow P_{nor}\big)\big]\big(exn(M)\big) \\ \xrightarrow{*}_{\rho_\varepsilon} & \big[\mathit{first}\big(exn(\bot) \twoheadrightarrow P_{fail}, x \twoheadrightarrow P_{nor}\big)\big]\big(exn(\{\bot\})\big) \\ \longrightarrow_{OpSet} & \big[\mathit{first}\big(exn(\bot) \twoheadrightarrow P_{fail}, x \twoheadrightarrow P_{nor}\big)\big]\big(\{exn(\bot)\}\big) \\ \xrightarrow{*}_{\rho_\varepsilon} & \{P_{fail}\} \end{array}$$

and thus, the initial term leads to P_{fail}, which is the wanted behavior.

(ii) if $M \xrightarrow{*}_{\rho_\varepsilon} \{M' \downarrow\}$ where $\{M' \downarrow\} \neq \{\bot\}$, we obtain:

$$\begin{array}{ll} & \big[\mathit{first}\big(exn(\bot) \twoheadrightarrow P_{fail}, x \twoheadrightarrow P_{nor}\big)\big]\big(exn(M)\big) \\ \xrightarrow{*}_{\rho_\varepsilon} & \big[\mathit{first}\big(exn(\bot) \twoheadrightarrow P_{fail}, x \twoheadrightarrow P_{nor}\big)\big]\big(exn(M')\big) \\ \xrightarrow{*}_{\rho_\varepsilon} & \{P_{nor}\} \end{array}$$

and in this case also, we have the wanted result: P_{nor}.

The rules of the system ELAN can be expressed using the Rho Calculus. A rule with no conditions and no local assignments $l \Rightarrow r$ is represented by $l \twoheadrightarrow r$. In fact, the ELAN evaluation mechanism distinguishes between labeled rewrite rules and unlabeled rewrite rules. The unlabeled rewrite rules are used to normalize the result of all the applications of a labeled rewrite rule to a term and thus, each time a labeled rewrite rule is applied to a term, the ELAN evaluation mechanism normalizes the result of its application with respect to the set of unlabeled rewrite rules. Hence, the labeled rewrite rule $[lab]\ \ l \Rightarrow r$ is represented by $l \twoheadrightarrow [im(\mathcal{R})](r)$ where $\mathcal{R}$ is the representation of the set of unlabeled ELAN rules.

The elementary ELAN strategies have, in most of the cases, a direct representation in the Rho Calculus. The identity (id) and the failure (fail) as well as the concatenation (;) are directly represented in the Rho Calculus by the ρ-operators id, $fail$ and ";" respectively, defined in Section 3.2.1. The strategy $\mathsf{dk}(S_1, \ldots, S_n)$ is represented in the Rho Calculus by the set $\{S_1, \ldots, S_n\}$ and the strategy $\mathsf{first}(S_1, \ldots, S_n)$ by the ρ-term $\mathit{first}(S_1, \ldots, S_n)$. The iteration strategy operator repeat* is easily represented by using the ρ-operator *repeat*.

The ELAN strategies are expressed using rewrite rules and herefore, can be represented by ρ-terms in the same way as the ELAN rewrite rules.

We have briefly described here on the representation of simple ELAN rules and the corresponding implicit and user defined strategies of ELAN. This representation can be extended to more general rules with conditions and local evaluations as detailed in [7].

4 Enhancing the Rewriting Calculus Design

In this section we present the untyped (*i.e.* à la Curry) syntax of the Rho Calculus as it was presented in [9] and that enhances (*i.e.* simpler but as expressive as before) the one presented in the previous section. In what follows, we will quickly introduce the new syntax, two operational semantics (small-step and big-step) and we will play with various strategies. In particular we show how a small enhancement in the calculus (more precisely in the big-step) semantics could help us to encode the *first* operator.

4.1 Enhanced Version of the Rho Calculus

The syntax of the enhanced ρCal is defined as follows:

$$\mathcal{T} ::= \mathcal{X} \mid \mathcal{K} \mid \mathcal{T} \twoheadrightarrow \mathcal{T} \mid [\mathcal{T} \ll \mathcal{T}].\mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

where $\mathcal{X}$ represents a denumerable set of variables and $\mathcal{K}$ a set of constants. By abuse of notation, we denote members of these sets by the same letters possibly indexed. The characteristics of this new syntax are the following:

(i) $\mathcal{T}_1 \twoheadrightarrow \mathcal{T}_2$ denotes a *rule abstraction* with pattern $\mathcal{T}_1$ and body $\mathcal{T}_2$; this is unchanged with respect to the initial version.

(ii) $[\mathcal{T}_1 \ll \mathcal{T}_2].\mathcal{T}_3$ denotes a *delayed matching constraint* with pattern $\mathcal{T}_1 \ll \mathcal{T}_2$ and body $\mathcal{T}_3$; this is a major evolution in the syntax and capability of the calculus. The application of an abstraction $\mathcal{T}_1 \twoheadrightarrow \mathcal{T}_3$ to a term $\mathcal{T}_2$ always "fires" and produces the term $[\mathcal{T}_1 \ll \mathcal{T}_2].\mathcal{T}_3$ which represents a constrained term where the matching equation is "put on the stack". The body of the constrained term may be evaluated or delayed. If a solution exists, the delayed matching constraint can be evaluated to $\sigma(\mathcal{T}_3)$, where σ is the solution of the matching between $\mathcal{T}_1$ and $\mathcal{T}_2$.

(iii) The application operator, previously denoted $[\mathcal{T}_1](\mathcal{T}_2)$, is now simply denoted as in the lambda calculus $(\mathcal{T}_1\ \mathcal{T}_2)$.

(iv) Finally, the use of the set brackets in the initial syntax can be usefully decomposed into two quite different parts. The first is a structure constructor denoted ",". The second is the semantics given to the structure constructor, which is described by an appropriate theory that depends on the kind of result one wants to formalize. Typically we recover the initial semantics of sets of result by giving an associative-commutative and idempotent semantics to ",". If one prefers lists or multisets results, then the corresponding formalization of "," should be specified.

As in the first version of the calculus, the substitutions are higher-order substitutions but grafting can be used when working modulo α-*convention*. Similarly, the same notion of matching as presented in Section 3.1 is used but we focus here on *syntactic matching*. Using these notions, we can are now ready to present the enhanced reduction semantics.

$$(\rho)\ (\mathcal{T}_1 \twoheadrightarrow \mathcal{T}_2)\mathcal{T}_3 \quad \rightarrow_\rho \quad [\mathcal{T}_1 \ll \mathcal{T}_3].\mathcal{T}_2$$

$$(\sigma)\ [\mathcal{T}_1 \ll \mathcal{T}_3].\mathcal{T}_2 \quad \rightarrow_\sigma \quad \sigma_{(\mathcal{T}_1 \ll \mathcal{T}_3)}(\mathcal{T}_2)$$

$$(\delta) \quad (\mathcal{T}_1, \mathcal{T}_2)\,\mathcal{T}_3 \quad \rightarrow_\delta \quad \mathcal{T}_1\,\mathcal{T}_3, \mathcal{T}_2\,\mathcal{T}_3$$

Fig. 6. The small-step semantics of the Rho Calculus

The small-step reduction semantics is defined by the reduction rules presented below: The central idea of the (ρ) rule of the calculus is that the application of a term $\mathcal{T}_1 \twoheadrightarrow \mathcal{T}_2$ to a term $\mathcal{T}_3$ reduces to the delayed matching constraint $[\mathcal{T}_1 \ll \mathcal{T}_3].\mathcal{T}_2$, while (the application of) the (σ) rule consists in solving the matching equation $\mathcal{T}_1 \lessdot \mathcal{T}_3$, and applying the obtained result to the the term $\mathcal{T}_2$. The rule (δ) deals with the distributivity of the application on the structures built with the "," constructor. As usual, we can introduce the classical notions of one-step, many-steps ($\mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta}$), and congruence ($=_{\rho\sigma\delta}$) relation of $\rightarrow_{\rho\sigma\delta}$.

4.2 Type Inference for the Rho Calculus

Various typed versions of the ρCal have been or are being developed, for different purposes: normalization, automated deduction, typed programming disciplines...

Here we present a simple (first-order) type inference system *à la* Curry which affects types to some terms of the ρCal. The rules are adapted from the polymorphic type system for the ρCal [9], and from a first-order type system *à la* Church for the ρCal [11].

In a nutshell, monomorphic and polymorphic type systems for ρCal have the "nice" properties that they do not guarantee termination since it allows the encoding of many fixpoints. Therefore, they are suitable to be taken as a foundational basis of realistic rewriting- and functional-based programming languages. The type system is adapted from the simply typed λ-calculus by generalizing the abstraction rule for patterns:

$$\frac{\Gamma, \Delta \vdash \mathcal{P} : \phi \qquad \Gamma, \Delta \vdash \mathcal{T}_2 : \psi \quad \mathsf{Dom}(\Delta) = \mathsf{Fv}(\mathcal{P})}{\Gamma \vdash \mathcal{P} \twoheadrightarrow \mathcal{T}_2 : \phi \twoheadrightarrow \psi}\,(Abs) \qquad \frac{\Gamma \vdash \mathcal{T}_1 : \phi \twoheadrightarrow \psi \quad \Gamma \vdash \mathcal{T}_2 : \phi}{\Gamma \vdash \mathcal{T}_1\ \mathcal{T}_2 : \psi}\,(Appl)$$

The full type system can be found in [11] with all its properties and proofs. As a simple example, we present here a term inspired by the famous $\omega\,\omega$ term of the untyped λ-calculus. Using the constant f whose type is $(\alpha \twoheadrightarrow \alpha) \twoheadrightarrow \alpha$, we define:

$$\omega_f \triangleq f(X) \twoheadrightarrow X\ f(X)$$

and we can typecheck it the following way (we suppose the type of constant

is given implicitily in a suitable signature (here omitted)):

$$\dfrac{\dfrac{(1)}{\vdash \omega_f : \alpha \twoheadrightarrow \alpha} \qquad \dfrac{\vdash f : (\alpha \twoheadrightarrow \alpha) \twoheadrightarrow \alpha \quad \vdash \omega_f : \alpha \twoheadrightarrow \alpha}{\vdash f(\omega_f) : \alpha}}{\vdash \omega_f \ f(\omega_f) : \alpha}$$

where (1) is

$$\dfrac{\dfrac{X{:}\alpha \twoheadrightarrow \alpha \vdash f : (\alpha \twoheadrightarrow \alpha) \twoheadrightarrow \alpha \qquad X{:}\alpha \twoheadrightarrow \alpha \vdash X : \alpha \twoheadrightarrow \alpha}{X{:}\alpha \twoheadrightarrow \alpha \vdash f(X) : \alpha} \qquad \dfrac{X{:}\alpha \twoheadrightarrow \alpha \vdash X : \alpha \twoheadrightarrow \alpha \qquad X{:}\alpha \twoheadrightarrow \alpha \vdash f(X) : \alpha}{X{:}\alpha \twoheadrightarrow \alpha \vdash X \ f(X) : \alpha}}{\vdash \omega_f : \alpha \twoheadrightarrow \alpha}$$

Finally, the only reduction path from $\omega_f \ f(\omega_f)$ can not end:

$$\begin{aligned} \omega_f \ f(\omega_f) \ &\equiv \ (f(X) \twoheadrightarrow X \ f(X)) \ f(\omega_f) \\ &\mapsto_\rho \ [f(X) \ll f(\omega_f)].(X \ f(X)) \\ &\mapsto_\sigma \ (X \ f(X))[\omega_f/X] \equiv \omega_f \ f(\omega_f) \\ &\mapsto_\rho \ \ldots \end{aligned}$$

This kind of typed fixpoint is made possible because any well-formed type can be chosen for the constants of the calculus. Here, f has type $(\alpha \twoheadrightarrow \alpha) \twoheadrightarrow \alpha$, so at the type level it makes a function over the set α look like an object in α.

A similar method can be used in ML in order to build a fixpoint without using `let rec`. Instead of defining a constant with an arbitrary type, we just have to define a peculiar inductive type whose unique constructor behaves like f:

```
type t = F of (t -> t);;
let omega x = match x with (F y) -> y (F y);;
```

Then the evaluation of `omega (F omega)` does not terminate.

In the ρCal, we have the possibility to use explicitly this kind of constants every time we want to express recursion. By checking the type of the constants and the context they are used in, this gives us the possibility to have an accurate control over non-termination.

4.3 Encoding Fix-points in Small-Step

Along the lines of the previous subsection, we show how to build some elaborated terms dedicated to applying a rewrite system on a term according to a specific strategy. We do not give the type derivations, but all the terms given here remain typable with the same system as before.

Since we will have to "try" all the rules of the rewrite system on a given subterm, the reductions in our terms will generate a lot of "junk" terms corresponding to the various matching failures. We will deal with them thanks to an enhanced matching theory for eliminating definitively-stuck-values, obtained by non recoverable matching failures. We call this theory $\mathbb{T}_{\mathsf{nostuck}}$ and use it for defining equivalence classes over the structure operator ",", in order to properly define a "result" term:

$$\frac{\nexists \sigma.\ \sigma(\mathcal{P}) \equiv \mathcal{T}_1 \vee \sigma(\mathcal{T}_1) \equiv \mathcal{P}}{[\mathcal{P} \ll \mathcal{T}_1].\mathcal{T}_2, \mathcal{T}_3 =^{\mathsf{no}}_{\mathsf{stuck}} \mathcal{T}_3}\,(\mathbb{T}_{\mathsf{nostuck}})$$

Intuitively, this theory drops, in a structure, all the matching failures which can not be made solvable by a further instantiation. For instance, $f(X) \ll g(3)$ will be dropped, but not $f(3) \ll f(X)$ (this last one becoming solvable if X is instantiated to 3). Note that, since the patterns contain no structures, the matching remains syntactic.

Moreover, to simulate the *global* behavior of a TRS $\mathcal{R}$, we need to write a suitable recursion operator, based on a fixpoint similar to the one presented in the subsection before. We give here some examples taken from [11], and encodings for some other strategies.

Let us begin with the example of the addition, using two constants rec and add.

$$plus \triangleq \begin{pmatrix} rec(S) \twoheadrightarrow add(0, Y) \twoheadrightarrow Y, \\ rec(S) \twoheadrightarrow add(suc(X), Y) \twoheadrightarrow suc\,(S.rec\ \ add(X, Y)) \end{pmatrix}$$

Then, this term computes indeed the addition over Peano integers, as illustrated in the following example, where the expressions "$\overline{m}$", and "$\overline{m+n}$", and "$\overline{m-n}$" are just aliases for the Peano representations of these numbers as sequences of $suc(\ldots suc(0)\ldots)$). To ease the reading, within the failed matching constraints, we keep only the subterms which do lead to a failure (for instance,

in $[add(0,Y) \ll add(\overline{n},\overline{m})]$, we only keep $[0 \ll \overline{n}]$).

$$
\begin{array}{ll}
& plus.rec \;\; add(\overline{n},\overline{m}) \\
\mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta} & (add(0,Y) \twoheadrightarrow Y) \;\; add(\overline{n},\overline{m}), \\
& (add(suc(X),Y) \twoheadrightarrow suc\,(plus.rec \;\; add(X,Y))) \;\; add(\overline{n},\overline{m}) \\
& \cdots \\
\mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta} & [0 \ll \overline{n}].\overline{m} \;, \quad [0 \ll \overline{n-1}].(\overline{m+1}), \cdots \\
& [0 \ll 0].(\overline{m+n}), \\
& [suc(X) \ll 0].(suc\,(plus.rec \;\; add(X,Y))) \\
=^{\mathsf{no}}_{\mathsf{stuck}} & [0 \ll 0].(\overline{m+n}) \\
\mapsto_{\sigma} & \overline{m+n}
\end{array}
$$

Worthy noticing is that all the stuck results are dropped by $=^{\mathsf{no}}_{\mathsf{stuck}}$; the only interesting member of the structure is $[0 \ll 0].(\overline{m+n})$. On the left of this term, all the terms get stuck because we try to match 0 against $suc(\overline{n})$; on the right too because we try to match $suc(X)$ against 0. Notice that if we erase all the "administrative" subterms which encode the recursive machinery, we get back a TRS computing addition:

$$
\begin{array}{ll}
add(0,Y) & \to Y \\
add(suc(X),Y) & \to suc\,(add(X,Y))
\end{array}
$$

This mechanical encoding works for many TRS as well: one just has to put the subterm "$S.rec$ " before all the defined constants, so that the whole rewrite system can be re-applied to any of the corresponding subterms. A full theoretical treatment and generalisation of the object-fixpoint engine can be found in [11].

Let us see a slightly more elaborated example:

$$
fibonacci \triangleq \left(\begin{array}{lll}
rec(S) \twoheadrightarrow fibo(0) & \twoheadrightarrow 1 \;\; , \\
rec(S) \twoheadrightarrow fibo(suc(0)) & \twoheadrightarrow 1 \;\; , \\
rec(S) \twoheadrightarrow fibo(suc(suc(N))) & \twoheadrightarrow plus.rec\; add(S.rec\; fibo(N), \\
 & \qquad S.rec\; fibo(suc(N)))
\end{array} \right)
$$

The engine we have shown has a *lazy innermost* policy: because of the presence of $S.rec$ before all the defined symbols, a position which is not innermost in the term can be reduced if and only if the corresponding rewrite rule discards all the subterms which contain a defined constant.

Other versions of this engine can be designed. For example, the ρ-term

given below computes the length of a list with an *outermost* evaluation strategy. The first call should have shape $length.rec\ \ len(0, L)$.

$$length \triangleq \begin{pmatrix} rec(S) \twoheadrightarrow len(N, nil) \twoheadrightarrow N, \\ rec(S) \twoheadrightarrow len(N, cons(X, L)) \twoheadrightarrow S.rec\ \ len(suc(N), L) \end{pmatrix}$$

This kind of encoding works well for rewrite systems written in a Prolog style, since there is always a defined symbol (len) on the head of the term, and the result is accumulated in an argument (N).

Notice that the notion of primitive recursion is well-preserved in this encoding. Let us suppose that there exists a decreasing argument (here, the list). The first computations are dropped because L has not been reduced to a base value nil, and the successful computations yield the required results. Finally, the other computations are dropped because they try to match a base value nil with an algebraic term $cons(X, L)$. If we want to "optimize" our programs, we have to ensure that there is no unnecessary recursive call. This is done by checking as soon as possible that a matching constraint does not fail; in other words, we would need a *call-by-value* evaluation for our calculus.

To end up with traversal, let us have a glance at how one can define a "customized" evaluation strategy. As in ELAN, a user of the ρCal may want to combine different strategies in the same rewrite system. For instance, in a function which selects a part of a list, the traversal of the list can be done straightforwardly (to generate less "junk" terms) and a lazy innermost strategy is better for the evaluation of each member of the list (because it is more efficient).

The easiest way to obtain this kind of user-defined evaluation is to have one term for each specific strategy and call them mutually. Let us take the example of the term which selects all the numbers greater than two in a list. The traversal acts like a recursive function on lists, by propagating the function to the tail of the list. The members of the list can be written with $plus$, so their evaluation is innermost. This encoding will be efficient because $plus$ "pushes" the constants suc directly out of the term, so the comparison to two does not need the full evaluation of the numbers:

$$select \triangleq \begin{pmatrix} rec2(T) \twoheadrightarrow nil \twoheadrightarrow nil, \\ rec2(T) \twoheadrightarrow cons(0, L) \twoheadrightarrow T.rec2\ L, \\ rec2(T) \twoheadrightarrow cons(suc(0), L) \twoheadrightarrow T.rec2\ L, \\ rec2(T) \twoheadrightarrow cons(suc(suc(N)), L) \twoheadrightarrow cons(suc(suc(N)), T.rec2\ L) \end{pmatrix}$$

The TRS defining $plus$ is encoded as before. Thus, when we call $select.rec2\ L$, the comparison to two is "distributed" over all the members of the list, and then the numbers must be at least partially evaluated so that the TRS $select$ can check whether there are two "suc" or not at the beginning of each member. Notice that we enforce an evaluation mechanism similar to the one in ELAN,

$$\frac{}{\mathcal{V} \Downarrow \mathcal{V}}\,(Red{-}Val)$$

$$\frac{\mathcal{T}_1 \Downarrow \mathcal{T}_3 \twoheadrightarrow \mathcal{T}_4 \quad [\mathcal{T}_3 \ll \mathcal{T}_2].\mathcal{T}_4 \Downarrow \mathcal{O}}{\mathcal{T}_1\,\mathcal{T}_2 \Downarrow \mathcal{O}}\,(Red{-}\rho)$$

$$\frac{\mathcal{T}_1 \Downarrow \mathcal{T}_3, \mathcal{T}_4 \quad \mathcal{T}_3\,\mathcal{T}_2 \Downarrow \mathcal{V}_1 \quad \mathcal{T}_4\,\mathcal{T}_2 \Downarrow \mathcal{V}_2}{\mathcal{T}_1\,\mathcal{T}_2 \Downarrow \mathcal{V}_1, \mathcal{V}_2}\,(Red{-}\delta)$$

$$\frac{\exists\sigma.\ \sigma(\mathcal{T}_1) \equiv \mathcal{T}_2 \quad \sigma(\mathcal{T}_3) \Downarrow \mathcal{O}}{[\mathcal{T}_1 \ll \mathcal{T}_2].\mathcal{T}_3 \Downarrow \mathcal{O}}\,(Red{-}\sigma_1)$$

$$\frac{\nexists\sigma.\ \sigma(\mathcal{T}_1) \equiv \mathcal{T}_2}{[\mathcal{T}_1 \ll \mathcal{T}_2].\mathcal{T}_3 \Downarrow \mathsf{wrong}}\,(Red{-}\sigma_2)$$

$$\frac{\mathcal{T}_1 \Downarrow \mathsf{wrong}}{\mathcal{T}_1\,\mathcal{T}_2 \Downarrow \mathsf{wrong}}\,(Red{-}Prop_1)$$

$$\frac{\mathcal{T}_1 \Downarrow \mathcal{T}_3, \mathcal{T}_4 \quad \mathcal{T}_3\,\mathcal{T}_2 \Downarrow \mathsf{wrong}}{\mathcal{T}_1\,\mathcal{T}_2 \Downarrow \mathsf{wrong}}\,(Red{-}Prop_2)$$

$$\frac{\mathcal{T}_1 \Downarrow \mathcal{T}_3, \mathcal{T}_4 \quad \mathcal{T}_3\,\mathcal{T}_2 \Downarrow \mathcal{V} \quad \mathcal{T}_4\,\mathcal{T}_2 \Downarrow \mathsf{wrong}}{\mathcal{T}_1\,\mathcal{T}_2 \Downarrow \mathsf{wrong}}\,(Red{-}Prop_3)$$

Fig. 7. The natural semantics of the Rho Calculus

because *plus* carries in fact the whole TRS defining addition (which can be considered as unlabeled), together with its associated innermost strategy.

4.4 Encoding "first" Using Natural Semantics

We define an operational semantics via a natural proof deduction system *à la* Kahn [18]. The purpose of the deduction system is to map every closed expression into a normal form, *i.e.* an irreducible term in weak head normal form. The presented strategy is *lazy call-by-name* since it does not work under plain abstractions (*i.e.* $\mathcal{T} \twoheadrightarrow \mathcal{T}$), structures (*i.e.* $\mathcal{T}, \mathcal{T}$), and algebraic terms (*i.e.* $\mathcal{K}\,\mathcal{T}$). We define the set of values $\mathcal{V}$ and output values $\mathcal{O}$ as follows:

$$\mathcal{V} ::= \mathcal{K} \mid \mathcal{T} \twoheadrightarrow \mathcal{T} \mid \mathcal{K}\,\mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

$$\mathcal{O} ::= \mathcal{V} \mid \mathsf{wrong}$$

The special output wrong represents the result obtained by a computation involving a "matching equation failure" (represented in [8] by null). The semantics is defined via a judgment of the shape $\mathcal{T} \Downarrow \mathcal{O}$, and its rules (almost self explaining) are presented in Figure 7. The big-step operational semantics

is deterministic, and immediately suggests how to build an interpreter for the calculus.

Example 4.1 [A call-by-value derivation] Let $\omega\,\omega \triangleq (\mathcal{X} \twoheadrightarrow \mathcal{X}\,\mathcal{X})\,(\mathcal{X} \twoheadrightarrow \mathcal{X}\,\mathcal{X})$, and take the term $(\mathcal{X} \twoheadrightarrow 3)\,(\omega\,\omega)$. One can check that this term diverges using a call-by-value strategy, *i.e.*,

$$\frac{\begin{array}{c}\infty\\ \therefore\end{array}}{(\mathcal{X} \twoheadrightarrow 3)\,(\omega\,\omega) \Downarrow \mathsf{stack\ overflow}}$$

while it would converge to 3 using call-by-name.

It is worth noticing that the output value wrong is used in order to denote "bad" computations where matching failure occurs at run-time. As such, the presented semantics does abort computations, as for example in

$$\frac{\vdots}{(3 \twoheadrightarrow 3, 4 \twoheadrightarrow 4)\,4 \Downarrow \mathsf{wrong}}$$

keeping in the final result the fact that one computation goes wrong and throwing away all non-wrong results. This corresponds to "killing" the computation once a wrong value is produced; as such, our machine is "pessimistic" (the machine stops if at least one wrong occurs).

In the following, we present a small extension of ρCal and its big-step semantics which takes into account a comfortable encoding of the *first* operator. To do this we need first to add an "exception" handling constructor to the ρCal syntax, *i.e.*

$\mathcal{T} ::= \mathsf{try}\ \mathcal{T}\ \mathsf{with}\ \mathcal{T} \mid$ ... as before ...

Executing a $\mathsf{try}\ \mathcal{T}_1\ \mathsf{with}\ \mathcal{T}_2$ means that if a matching failure occurs when evaluating the term $\mathcal{T}_1$, the handler $\mathcal{T}_2$ will be executed. Otherwise, the result of the execution of $\mathcal{T}_1$ will be propagated outside the scope of the try_with.

More precisely: first, we evaluate $\mathcal{T}_1$ (the protected term). If $\mathcal{T}_1$ evaluates to an output without matching failures (*i.e.* an output value), then this value will be the result of the whole try_with expression. This roughly corresponds to executing $\mathcal{T}_1$ without matching failure. Counterwise, if a matching failure occurs, then we execute the handler $\mathcal{T}_2$. Note that in the case of a multiple nesting of try_with, our operational semantics will handle the feature by executing the innermost "handler". This simple extension allows us to easily encode the *first* operator.

Therefore, we keep all the previous rules and we add the two deduction

rules below:

$$\frac{\mathcal{T}_1 \Downarrow \mathsf{wrong} \quad \mathcal{T}_2 \Downarrow \mathcal{O}}{\mathsf{try}\ \mathcal{T}_1\ \mathsf{with}\ \mathcal{T}_2 \Downarrow \mathcal{O}}\,(Red{-}Fail_1)$$

$$\frac{\mathcal{T}_1 \Downarrow \mathcal{V}}{\mathsf{try}\ \mathcal{T}_1\ \mathsf{with}\ \mathcal{T}_2 \Downarrow \mathcal{V}}\,(Red{-}Fail_2)$$

Intuitively, the first deduction rule evaluates the handler $\mathcal{T}_2$ if and only if the protected body $\mathcal{T}_1$ goes to wrong; otherwise the protected body $\mathcal{T}_1$ succeeds without matching failures, and the second rule applies.

As before, the presented interpreter implements a "pessimistic" machine that strictly propagates the matching failure signal. Thus, one should remember that, according to the propagation rules, a matching failure signal is propagated independently of the other (possibly successful) computations.

We are now ready to provide a canonical encoding of the *first* operator using the newly introduced try_with construct as follows:

$$\mathit{first}(\mathcal{T}_1 \cdots \mathcal{T}_n) \triangleq \mathcal{X} \twoheadrightarrow (\mathsf{try}\ \mathcal{T}_1\,\mathcal{X}\ \mathsf{with}\ \mathsf{try}\ \mathcal{T}_2\,\mathcal{X}\ \mathsf{with}\ \ldots\ \mathsf{try}\ \mathcal{T}_{n-1}\,\mathcal{X}\ \mathsf{with}\ \mathcal{T}_n\,\mathcal{X})$$

Example 4.2 [One run of *first*] Take the term $\mathit{first}(1 \twoheadrightarrow 1, 2 \twoheadrightarrow 2, 3 \twoheadrightarrow 6)\,2$. A derivation for this term is:

$$\dfrac{\dfrac{\dfrac{}{1 \twoheadrightarrow 1 \Downarrow 1 \twoheadrightarrow 1} \quad \dfrac{\nexists\sigma.\sigma(1) \equiv 2}{[1 \ll 2].1 \Downarrow \mathsf{wrong}}}{(1 \twoheadrightarrow 1)\,2 \Downarrow \mathsf{wrong}} \quad \dfrac{\dfrac{}{2 \twoheadrightarrow 2 \Downarrow 2 \twoheadrightarrow 2} \quad \dfrac{\exists\sigma_{\mathsf{ID}}.\sigma_{\mathsf{ID}}(2) \equiv 2 \quad 2 \Downarrow 2}{[2 \ll 2].2 \Downarrow 2}}{(2 \twoheadrightarrow 2)\,2 \Downarrow 2}}{\mathsf{try}\ (2 \twoheadrightarrow 2)\,2\ \mathsf{with}\ (3 \twoheadrightarrow 6)\,2 \Downarrow 2}}{\mathsf{try}\ (1 \twoheadrightarrow 1)\,2\ \mathsf{with\ try}\ (2 \twoheadrightarrow 2)\,2\ \mathsf{with}\ (3 \twoheadrightarrow 6)\,2 \Downarrow 2}}{\mathit{first}(1 \twoheadrightarrow 1, 2 \twoheadrightarrow 2, 3 \twoheadrightarrow 6)\,2 \Downarrow 2}$$

4.5 Encoding First-class Exceptions

In the previous section, we saw that the wrong output value aborts computations, and the construct try_with was introduced in order to encode the *first* operator. Nevertheless, modern programming languages are interested in trying some chunks of code in a protected environment, and whenever a run-time matching error occurs, catch it first, and then execute some exception handler which can be declared many "miles" away from where the exception was raised. This is the case of the `try{...}catch{...}` mechanism of Java, or similar constructs in ML, C#, ... In ρCal an exception can be represented by the fact that some matching failure occurred at run-time, such as matching the constant 3 against another constant 4.

Among the possible extensions of ρCal taking into account first-class exceptions it is worth to mention:

(i) In [9], we presented an extension of ρCal which takes into account matching exceptions and their handling. This extension was also inspired from [21]. We extended the syntax as follows:

$$\mathcal{T} ::= \mathsf{try}\ \mathcal{T}\ \mathsf{catch}\ [\mathcal{T} \ll \mathcal{T}]\ \mathsf{with}\ \mathcal{T} \mid \ldots \text{ as before } \ldots$$

Intuitively, $[\mathcal{T} \ll \mathcal{T}]$ denotes a matching equation without solutions, like *e.g.* $[3 \ll 4]$. Executing a $\mathsf{try}\ \mathcal{T}_1\ \mathsf{catch}\ [\mathcal{T}_2 \ll \mathcal{T}_3]\ \mathsf{with}\ \mathcal{T}_4$ means that the scope for catching the matching failure $[\mathcal{T}_2 \ll \mathcal{T}_3]$ is the term $\mathcal{T}_1$, and that if that exception occurs, the handler $\mathcal{T}_4$ will be executed. Otherwise the raised exception will be propagated outside the scope of the try_catch_with.

More precisely, we first evaluate $\mathcal{T}_1$ (the protected term) and if $\mathcal{T}_1$ evaluates to an output without matching failures (*i.e.* a value), then this value will be the result of the whole try_catch_with expression (this roughly corresponds to executing $\mathcal{T}_1$ without raising exceptions). Counterwise, if a matching failure occurs, then we must check whether the failure is the one declared in the try_catch_with expression (*i.e.* $[\mathcal{T}_2 \ll \mathcal{T}_3]$) or not; in the first case we execute the handler $\mathcal{T}_4$, while in the latter case we propagate the matching failure outside the scope of the try_catch_with expression. Note that the scope of the exception catching ranges over $\mathcal{T}_1$ but not $\mathcal{T}_4$.

(ii) In [10], we are currently studying another possible choice for handling exceptions where the syntax is extended as follows:

$$\mathcal{T} ::= \mathsf{try}\ \mathcal{T}\ \mathsf{with}\ \mathcal{T} \twoheadrightarrow \mathcal{T} \mid \mathsf{raise}\ \mathcal{T} \mid \ldots \text{ as before } \ldots$$

The term $\mathsf{try}\ \mathcal{T}_1\ \mathsf{with}\ \mathcal{T}_2 \twoheadrightarrow \mathcal{T}_3$ represents the evaluation of the term $\mathcal{T}_1$ in a protected environment. If an exception (due to a matching failure) is produced during the evaluation of $\mathcal{T}_1$, the exception handler $\mathcal{T}_2 \twoheadrightarrow \mathcal{T}_3$ is used for "interpreting" this exception. Using the matching possibilities of the calculus the exceptions can be discriminated and actions taken in consequence. If the handler $\mathcal{T}_2 \twoheadrightarrow \mathcal{T}_3$ cannot be applied (because of a matching failure) to the resulting exception, this new failure is propagated outside the scope of the try_with.

As in the previous approach, we first evaluate $\mathcal{T}_1$ and if $\mathcal{T}_1$ evaluates to an output without matching failures, then this value will be the result of the whole try_with expression. Counterwise, if a matching failure occurs, then a $\mathsf{raise}\ \mathcal{T}_4$ expression is produced and the value $\mathcal{T}_4$ is propagated to the first declared try_with expression; at this point, if the handler $\mathcal{T}_2 \twoheadrightarrow \mathcal{T}_3$ is applicable to $\mathcal{T}_4$, then the exception is "captured", otherwise another $\mathsf{raise}\ \mathcal{T}_4$ will propagate the exception further towards the next try_with expression. As an example, evaluating the term below in a suitably modified natural semantics (here omitted) will yield the following judgment:

$$\mathsf{try}\ (\mathsf{try}\ (f(4,3) \twoheadrightarrow 5)\, f(3,4)\ \mathsf{with}\ f(4,X) \twoheadrightarrow X)\ \mathsf{with}\ f(3,X) \twoheadrightarrow X \Downarrow 4.$$

5 Conclusion

We have summarized in this paper the evolution of the rewriting calculus from its first version to the recent enhanced versions and discuss different extensions of the calculus.

We have shown how different classical strategies like, for example, innermost and outermost, can be defined using the different versions of the Rho Calculus. This has been done first by defining fix-point operators and adding a new operator, first, that selects a "successful" reduction. Extending the calculus with an exception handling mechanism allows us to define this latter operator in the calculus itself.

The definition of these operators allows us to give an operational semantics to the execution of rewrite rules and strategies of the language ELAN.

A very nice feature of the simply typed rewriting calculus is to allow to type strategies, *i.e.* terms of the rewriting calculus, that could be not normalizing. This is quite powerful and promising but this should be controled and we are now working on elaborating type systems giving such a control.

References

[1] F. Baader and T. Nipkow. *Term Rewriting and all That*. Cambridge University Press, 1998.

[2] H. Barendregt. *The Lambda-Calculus, its syntax and semantics*. Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. Second edition.

[3] P. Borovanský. *Le contrôle de la réécriture: étude et implantation d'un formalisme de stratégies*. Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, France, October 1998. also TR LORIA 98-T-326.

[4] P. Borovansky, C. Kirchner, H. Kirchner, and P.-E. Moreau. ELAN from a rewriting logic point of view. *Theoretical Computer Science*, (285):155–185, July 2002.

[5] H. Cirstea. *Calcul de réécriture : fondements et applications*. Thèse de Doctorat d'Université, Université Henri Poincaré - Nancy I, 2000.

[6] H. Cirstea and C. Kirchner. Combining higher-order and first-order computation using ρ-calculus: Towards a semantics of ELAN. In D. Gabbay and M. de Rijke, editors, *Frontiers of Combining Systems 2*, Research Studies, ISBN 0863802524, pages 95–120. Wiley, 1999.

[7] H. Cirstea and C. Kirchner. The rewriting calculus — Part I *and* II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–498, May 2001.

[8] H. Cirstea, C. Kirchner, and L. Liquori. Matching Power. In A. Middeldorp, editor, *Rewriting Techniques and Applications*, volume 2051 of *Lecture Notes in Computer Science*, Utrecht, The Netherlands, May 2001. Springer-Verlag.

[9] H. Cirstea, C. Kirchner, and L. Liquori. Rewriting Calculus with(out) Types. In *Workshop on Rewriting Logic and its Applications - WRLA'2002, Pisa, Italy*. ENTCS, September 2002.

[10] H. Cirstea, L. Liquori, and *alt.* An exceptional rewriting calculus. Manuscript, 2003.

[11] H. Cirstea, L. Liquori, and B. Wack. Typed recursive rewriting calculus. Manuscript. Available at `http://www.loria.fr/~wack/papers/full-TypedRecursive.ps.gz`, 2003.

[12] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990.

[13] G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions, extended abstract. In D. Kozen, editor, *Proceedings of LICS'95*, pages 366–374, San Diego, June 1995.

[14] F. Fages and G. Huet. Unification and matching in equational theories. In *Proceedings Fifth Colloquium on Automata, Algebra and Programming, L'Aquila (Italy)*, volume 159 of *Lecture Notes in Computer Science*, pages 205–220. Springer-Verlag, 1983.

[15] G. Faure and C. Kirchner. Exceptions in the rewriting calculus. In S. Tison, editor, *Proceedings of the RTA conference*, volume 2368 of *Lecture Notes in Computer Science*, pages 66–82, Copenhagen, July 2002. Springer-Verlag.

[16] G. Huet. *Résolution d'equations dans les langages d'ordre 1,2, ...,ω*. Thèse de Doctorat d'Etat, Université de Paris 7 (France), 1976.

[17] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. The MIT press, Cambridge (MA, USA), 1991.

[18] G. Kahn. Natural Semantics. In *Proc. of STACS*, volume 247 of *LNCS*, pages 22–39. Springer-Verlag, 1987.

[19] C. Kirchner and H. Kirchner. Rewriting, solving, proving. A preliminary version of a book available at `www.loria.fr/~ckirchne/rsp.ps.gz`, 1999.

[20] C. Kirchner, H. Kirchner, and M. Vittek. Designing constraint logic programming languages using computational systems. In P. Van Hentenryck and V. Saraswat, editors, *Principles and Practice of Constraint Programming. The Newport Papers.*, chapter 8, pages 131–158. The MIT press, 1995.

[21] L. Liquori. Semantica e Pragmatica di un Linguaggio Funzionale con le Continuazioni Esplicite. Laurea in Science dell'Informazione, University of Udine, 1990. In Italian, 74 pp.

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

[23] A. M. Turing. The $\wp$-functions in λ-K-conversion. *The Journal of Symbolic Logic*, 2:164, 1937.

[24] E. Visser and Z. el Abidine Benaissa. A core language for rewriting. In C. Kirchner and H. Kirchner, editors, *Proceedings of the second International Workshop on Rewriting Logic and Applications*, volume 15, `http://www.elsevier.nl/locate/entcs/volume16.html`, Pont-à-Mousson (France), September 1998. Electronic Notes in Theoretical Computer Science.

Call-by-Value, Call-by-Name, and Strong Normalization for the Classical Sequent Calculus

Stéphane Lengrand

École Normale Supérieure de Lyon
46, Allée d'Italie, 69364 Lyon cedex 07, France
Stephane.Lengrand@ENS-Lyon.fr
and
PPS - Université Denis Diderot
2, Place Jussieu, 75251 Paris cedex 05, France

Abstract

We present a typed calculus $\lambda\xi$ isomorphic to the implicational fragment of the classical sequent calculus LK. Reductions in LK eliminate the cut-rule by local rewriting steps, which correspond to the evaluation of explicit substitutions in the calculus. This bridges the gap between Curien and Herbelin's $\overline{\lambda}\mu\widetilde{\mu}$-calculus and Urban's rewriting system for proofs. Encodings of one into the other are defined, and from one of them we derive the strong normalization of $\overline{\lambda}\mu\widetilde{\mu}$. Identifying two reduction strategies CBV and CBN in Urban's rewriting system enables us to derive two corresponding semantics of continuations from those of $\overline{\lambda}\mu\widetilde{\mu}$, via the other encoding.

1 Introduction

The sequent calculus, introduced by Gentzen in [Gen35], is a logical system in which the rules only introduce connectives (but on both sides of a sequent), on the contrary to natural deduction which uses introduction and elimination rules. The only way to eliminate a connective is to eliminate the whole formula in which it appears, with an application of the cut-rule. Gentzen calculus for classical logic LK allows sequents of the form $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$, where $A_1, \ldots, A_n$ is to be understood as $A_1 \wedge \ldots \wedge A_n$ and $B_1, \ldots, B_m$ is to be understood as $B_1 \vee .... \vee B_m$. Thus, LK appears as a very symmetrical system.

The cut-rule does not increase the expressive power of the system since a *cut-elimination procedure* has been defined to eliminate all applications of the cut-rule from the proof of a sequent, generating a proof in normal form of the same sequent, that is, with no cut. It proceeds with local rewriting steps of reductions of the proof-tree, and that has the flavour of the evaluation of explicit substitutions, now a wide area of interest in programming theory.

Indeed, the typing rule of an explicit substitution, say in λx [BR95], is nothing else but a cut, and a lot of work has been done to better understand the connection between explicit substitutions and local cut-reduction procedures.

We present here a correspondence *à la* Curry-Howard for LK bringing together the various features of two different approaches that we compare: that of Urban [Urb00] and that of Curien and Herbelin [CH00].

On the logical side, in [Urb00], Urban analyzes thoroughly, among other things, Gentzen-like cut-elimination procedures, and defines a very general reduction system for the proofs in LK which is strongly normalizing, and in which proofs are represented by a syntax of terms. We call the implicational fragment $\lambda\xi$.

Its typing system being isomorphic to LK, the symmetry of the latter naturally leads to the idea of a symmetrical explicit substitution of the form $(M_1\widehat{\alpha} \dagger \widehat{x}M_2)$ typed by the symmetrical cut-rule of LK. The only redexes are the explicit substitutions and the process of cut-elimination corresponds to their evaluation. Normal forms are the terms built from the restricted syntax without explicit substitutions, just as simplified proofs are the proofs built without the cut-rule. The intuitive meaning of the symmetrical substitution would be like "the input x of M_2 is replaced by the output α of M_1" (or is it the output α of M_1 that is replaced by the input x of M_2?).

Such a question tackles the problem of duality in computation between input and output, program and context, or more relevantly between call-by-name and call-by-value evaluation. Indeed, the non-determinism (and non-confluence) of the cut-elimination in LK comes from the presence of many critical pairs, the reductions of which come within the problem of evaluation strategies, which is to be connected to the duality aforementioned. The latter has been shown several times those last few years, in particular the symmetry between CBV and CBN, which Selinger displayed by defining the semantics of Parigot's $\lambda\mu$-calculus ([Par92]) in control and co-control categories [Sel99]. The symmetry is inherent to classical logic, even in natural deduction.

Filinsky had already defined in [Fil89] a symmetrical λ-calculus, based on an approach with continuations, with a nice syntactic conversion of functions as values to deal with higher-order programming. Eventually, Curien and Herbelin manage to connect the duality of computation to the symmetry of classical sequent calculus in a nice way: they define in [CH00] the $\overline{\lambda}\mu\widetilde{\mu}$-calculus to interpret computationally the implicational fragment of LK. Selinger's symmetry gets syntactically exhibited in the $\overline{\lambda}\mu\widetilde{\mu}$-calculus by giving priority to a reduction rule or its dual one.

On the one hand, there is classical logic, and in Parigot's $\lambda\mu$-calculus there is one main conclusion that is being manipulated and possibly several alternative ones. On the other hand, there is sequent calculus and the necessity for the left-introduction rules to manipulate hypotheses, and there the concept of *stoup* that Herbelin has thoroughly studied in [Her95] seems to be much relevant.

One of the key points of $\overline{\lambda}\mu\widetilde{\mu}$-calculus is to notice that the stoup and the

main conclusion of $\lambda\mu$ were the dual notions of each other, and to express this duality in a very symmetrical syntax. But the duality goes far beyond: for instance, the reduction rules display syntactically very easily the duality between the CBV and CBN evaluations.

In this paper, we study the relationship between the $\overline{\lambda}\mu\widetilde{\mu}$-calculus and the implicational fragment $\lambda\xi$ of Urban's rewriting system. We encode them into each other. Thus, the former inherits the strong normalization of the latter, and the latter inherits the semantics of the former. In particular, we could not define a denotational semantics of $\lambda\xi$ before we first identified two reduction strategies CBV and CBN in the calculus.

Section 2 presents the classical sequent calculus, section 3 briefly presents the $\overline{\lambda}\mu\widetilde{\mu}$-calculus and its semantics (of continuations). Section 4 presents the implicational fragment of Urban's term rewriting system for LK, as well as the identification of the CBV and CBN evaluation strategies. Section 5 encodes $\overline{\lambda}\mu\widetilde{\mu}$ into Urban's calculus and proves its strong normalization. Section 6 presents the derived semantics for Urban's calculus.

2 Logical setting

We consider the following implicational fragment of Gentzen's system LK for classical logic. The hypotheses and the conclusions of a logical sequent are multi-sets, and the comma will henceforth mean the union of multi-sets.

$$\frac{}{A \vdash A}\mathsf{Ax} \qquad \frac{\Gamma, A \vdash \Delta \quad \Gamma' \vdash A, \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \mathsf{cut}$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', (A \Rightarrow B) \vdash \Delta, \Delta'} \Rightarrow_L \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash (A \Rightarrow B), \Delta} \Rightarrow_R$$

The *structural* rules (contraction and weakening) deal with the multi-sets:

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\mathsf{Contr}_L \quad \frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\mathsf{Contr}_R \qquad \frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\mathsf{Weak}_L \ \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\mathsf{Weak}_R$$

Gentzen's cut-elimination procedure eliminates applications of the cut-rule in a proof by moving them upwards, towards the applications of the axiom rule, where they eventually disappear. When we generalize the procedure by allowing the rewriting rules to be applied in any context without restriction, it gives a reduction system which is not deterministic and highly not confluent.

The cut-rule is necessary to simulate the $\Rightarrow$-elimination of natural deduction without having to modify deeply the proofs of the premisses. When λ-calculus (natural deduction) is thus encoded into sequent calculus, if we want to simulate β-reduction by cut-elimination, then some sort of cut-permutation must be allowed. Usually, strong normalization of the cut-elimination is lost when this is allowed. The power of Urban's calculus is its ability to simulate β-reduction without loosing the strong normalization.

3 The $\overline{\lambda}\mu\widetilde{\mu}$-calculus and its semantics

There are two sets of variables: $x, y, z, \ldots$ label the types of the hypotheses and $\alpha, \beta, \gamma, \ldots$ label the types of the conclusions. Moreover the syntax of $\overline{\lambda}\mu\widetilde{\mu}$ has three different categories: commands, terms, and contexts. Correspondingly, they are typed by three kinds of sequents: the usual sequents $\Gamma \vdash \Delta$ type commands, while the sequents typing terms (contexts) are of the form $\Gamma \vdash A| \Delta$ ($\Gamma| A \vdash \Delta$), making the conclusion (hypothesis) A *active*.

Definition 3.1 [Commands, Terms and Contexts]

$$\begin{array}{ll} c ::= \langle v|e\rangle & \text{(commands)} \\ v ::= x|\mu\beta.c|\lambda x.v & \text{(terms)} \\ e ::= \alpha|\widetilde{\mu}x.c|v \cdot e & \text{(contexts)} \end{array}$$

$\mu\alpha.c$, $\widetilde{\mu}x.c'$ and $\lambda y.v$ respectively bind α in c, x in c' and y in v, and we always consider terms, contexts and commands up to α-conversion, writing them in a way satisfying Barendregt's convention.

A context can be a variable but can be more complex, so as to have a typing rule introducing $\rightarrow$ on the left-hand side of a sequent, and commands fill the hole of a context with a term. The typing system $\mathsf{LK}_{\mu\widetilde{\mu}}$ is very close to LK:

Definition 3.2 [$\mathsf{LK}_{\mu\widetilde{\mu}}$]

$$\frac{\Gamma \vdash v : A|\Delta \quad \Gamma|e : A \vdash \Delta}{\langle v|e\rangle : (\Gamma \vdash \Delta)} \qquad \text{which corresponds to the logical } \mathsf{cut}\text{-rule}$$

$$\frac{}{\Gamma|\alpha : A \vdash \alpha : A, \Delta} \quad \frac{}{\Gamma, x : A \vdash x : A|\Delta}$$

which are two forms corresponding to the logical axiom rule (but with weakening).

$$\frac{\Gamma, x : A \vdash v : B|\Delta}{\Gamma \vdash \lambda x.v : A \rightarrow B|\Delta} \qquad \frac{\Gamma \vdash v : A|\Delta \quad \Gamma|e : B \vdash \Delta}{\Gamma|v \cdot e : A \rightarrow B \vdash \Delta}$$

which correspond respectively to the right and left introduction of the arrow.

$$\frac{c : (\Gamma \vdash \beta : A, \Delta)}{\Gamma \vdash \mu\beta.c : A|\Delta} \qquad \frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma|\widetilde{\mu}x.c : A \vdash \Delta}$$

which are two rules specific to the calculus that select the conclusion or the hypothesis that will be manipulated in the next typing rule.

It appears that the contraction and the weakening rules of LK are subtly integrated into the above rules, as it will be discussed in the next section. Note that the type of a context is the type that a term is expected to have in order to fill the hole. With conventional notations about contexts, $v \cdot e$ is to be thought as $e[[\ ]\ v]$. We see here how a term (context) is built either

by introducing $\to$ on the right-hand side (left-hand side) of a sequent, or just by activating one conclusion (hypothesis) from a sequent typing a command: $\mu\alpha.c$ is inherited from Parigot's $\lambda\mu$ [Par92], and $\widetilde{\mu}x.c$ is to be thought as $\mathsf{let}\ x = [\,]\ \mathsf{in}\ c$. As an example, here is a proof witness for Peirce's Law: $\vdash \lambda z.\mu\alpha.\langle z|(\lambda x.\mu\beta.\langle x|\alpha\rangle)\cdot\alpha\rangle : ((A\to B)\to A)\to A$

Proofs can sometimes be simplified, that is, commands can be computed:

Definition 3.3 [Reductions in $\overline{\lambda}\mu\widetilde{\mu}$]

$$
\begin{array}{lll}
(\to) & \langle \lambda x.v_1 | v_2 \cdot e\rangle & \to \langle v_2 | \widetilde{\mu}x.\langle v_1|e\rangle\rangle \\
(\mu) & \langle \mu\beta.c | e\rangle & \to c[\beta \leftarrow e] \\
(\widetilde{\mu}) & \langle v | \widetilde{\mu}x.c\rangle & \to c[x \leftarrow v]
\end{array}
$$

The system has a critical pair $\langle \mu\alpha.c_1|\ \widetilde{\mu}x.c_2\rangle$ and applying in this case the rule μ gives a call-by-value evaluation, whereas applying the rule $\widetilde{\mu}$ gives a call-by-name evaluation. It is interesting to see that the system with both rules is not confluent, which is connected to the fact that neither is the cut-elimination of the classical sequent calculus.

In [CH00], Curien and Herbelin say that it should be interesting to interpret the typed $\overline{\lambda}\mu\widetilde{\mu}$ in Selinger's control categories. The interpretation of Parigot's $\lambda\mu$ in control categories being based on a semantics of continuations, they define such a semantics for $\overline{\lambda}\mu\widetilde{\mu}$ by translation into CPS. Although the CBN case involves the interpretation of types that Hofmann and Streicher gave in [HS97], we give here as an example an interpretation *à la* Plotkin for its simplicity, as it does not require an extension with products of the lambda-calculus.

Definition 3.4 We define the semantics of terms as follows:

	CBV-translation		CBN-translation
$\overline{x}$	$:= \lambda k.k\ \widetilde{x}$	$\underline{x}$	$:= \widetilde{x}$
$\overline{\lambda x.v}$	$:= \lambda k.(k\ (\lambda u.\lambda\widetilde{x}.(\overline{v}\ u)))$	$\underline{\lambda x.v}$	$:= \lambda k.(k\ \lambda\widetilde{x}.\underline{v})$
$\overline{\mu\alpha.c}$	$:= \lambda\widetilde{\alpha}.\overline{c}$	$\underline{\mu\alpha.c}$	$:= \lambda\widetilde{\alpha}.\underline{c}$
$\overline{\alpha}$	$:= \widetilde{\alpha}$	$\underline{\alpha}$	$:= \lambda u.(u\ \widetilde{\alpha})$
$\overline{v\cdot e}$	$:= \lambda u.(\overline{v}\ (u\ \overline{e}))$	$\underline{v\cdot e}$	$:= \lambda u.(c\ (\underline{e}\ \underline{v}))$
$\overline{\widetilde{\mu}x.c}$	$:= \lambda\widetilde{x}.\overline{c}$	$\underline{\widetilde{\mu}x.c}$	$:= \lambda\widetilde{x}.\underline{c}$
$\overline{\langle v\|\ e\rangle}$	$:= \overline{v}\ \overline{e}$	$\underline{\langle v\|\ e\rangle}$	$:= \underline{e}\ \underline{v}$

As we always want to see the connexion with logic, we are concerned with the translation of types. σ denotes a base type (of $\overline{\lambda}\mu\widetilde{\mu}$), R is the type of responses, V_A denotes a type of values, K_A denotes a type of continuations, and C_A denotes a type of computations, all defined as follows:

Definition 3.5

$$\begin{array}{ll} \mathsf{CBV} & \mathsf{CBN} \\ V_\sigma^{\mathsf{CBV}} := \sigma & V_\sigma^{\mathsf{CBN}} := \sigma \\ V_{A\to B}^{\mathsf{CBV}} := K_B^{\mathsf{CBV}} \to K_A^{\mathsf{CBV}} & V_{A\to B}^{\mathsf{CBN}} := C_A^{\mathsf{CBN}} \to C_B^{\mathsf{CBN}} \\ K_A^{\mathsf{CBV}} := V_A^{\mathsf{CBV}} \to R & K_A^{\mathsf{CBN}} := V_A^{\mathsf{CBN}} \to R \\ C_A^{\mathsf{CBV}} := K_A^{\mathsf{CBV}} \to R & C_A^{\mathsf{CBN}} := K_A^{\mathsf{CBN}} \to R \end{array}$$

Such an interpretation of types enables us to prove the following:

Theorem 3.6 (Preservation of types)

$$\begin{cases} \Gamma \vdash v : A | \Delta \Rightarrow & V_\Gamma^{CBV}, K_\Delta^{CBV} \vdash \overline{v} : C_A^{CBV} \\ & C_\Gamma^{CBN}, K_\Delta^{CBN} \vdash \underline{v} : C_A^{CBN} \\ \Gamma | \ e : B \vdash \Delta \Rightarrow & V_\Gamma^{CBV}, K_\Delta^{CBV} \vdash \overline{e} : K_B^{CBV} \\ & C_\Gamma^{CBN}, K_\Delta^{CBN} \vdash \underline{e} : K_B^{CBN} \\ c : (\Gamma \vdash \Delta) \Rightarrow & V_\Gamma^{CBV}, K_\Delta^{CBV} \vdash \overline{c} : R \\ & C_\Gamma^{CBN}, K_\Delta^{CBN} \vdash \underline{c} : R \end{cases}$$

The preservation of the types is very interesting from a logical point of view. The typing tree of the semantics is the intuitionistic proof of a sequent in natural deduction, which is the typing system of standard λ-calculus. By modifying the types, we have actually defined two transformations of the logical judgements by adding double negations, making classically provable sequents provable in intuitionistic logic, an idea introduced by Gödel.

Fortunately, it is the case that a CBV-reduction preserves the CBV-semantics and that a CBN-reduction preserves the CBN-semantics (up to α, β-conversion).

Theorem 3.7 (Preservation of semantics)

- *If* $M \to_{CBV} N$ *then* $\overline{M} =_{\alpha,\beta} \overline{N}$
- *If* $M \to_{CBN} N$ *then* $\underline{M} =_{\alpha,\beta} \underline{N}$

Of course, the λ-terms can now be interpreted in a cartesian closed category, and if we compose those interpretations for a command c, we get the morphisms:

$$(V_{A_1}^{\mathsf{CBV}} \times \ldots \times V_{A_n}^{\mathsf{CBV}} \times K_{B_1}^{\mathsf{CBV}} \times \ldots \times K_{B_m}^{\mathsf{CBV}}) \xrightarrow{[\![\overline{c}]\!]} R$$

$$(C_{A_1}^{\mathsf{CBN}} \times \ldots \times C_{A_n}^{\mathsf{CBN}} \times K_{B_1}^{\mathsf{CBN}} \times \ldots \times K_{B_m}^{\mathsf{CBN}}) \xrightarrow{[\![\underline{c}]\!]} R$$

Of course, since we have a translation into CPS, we do not even need a cartesian closed category, we need a response category, which need not have exponential object B^A for every pair A, B, but only for $B = R$, the response object

of the category. Besides, we require a response category to have distributive finite products and coproducts and we also require that the morphisms $A \longrightarrow R^{R^A}$ are monic for all A. Note that the sub-category consisting of the objects of the form R^A, called a continuation category, is cartesian closed with: $1 \simeq R^0$, $R^A \times R^B \simeq R^{A+B}$, and $(R^B)^{R^A} \simeq R^{B \times R^A}$, where 1 (resp. 0) is a terminal (resp. initial) object. So we can curryfy the interpretations which become morphisms of that sub-category of continuations:

$$(K^{\mathsf{CBV}}_{B_1} \times \ldots \times K^{\mathsf{CBV}}_{B_m}) \xrightarrow{\Lambda[\![\overline{c}]\!]} R^{V^{\mathsf{CBV}}_{A_1} \times \ldots \times V^{\mathsf{CBV}}_{A_n}}$$

$$(C^{\mathsf{CBN}}_{A_1} \times \ldots \times C^{\mathsf{CBN}}_{A_n}) \xrightarrow{\Lambda[\![c]\!]} R^{K^{\mathsf{CBN}}_{B_1} \times \ldots \times K^{\mathsf{CBN}}_{B_m}}$$

That defines exactly what the semantics in a (co-)control category should be. Indeed, in [Sel99], Selinger proposed two categorical semantics of $\lambda\mu$-calculus: a call-by-name semantics in a control category, and a call-by-value semantics in a co-control category, expressing the duality between the two disciplines. An interesting point of control categories, the definition of which is too long to fit in here, is that every control category is equivalent to a category of continuations. Hence, we have the interpretation of $\overline{\lambda}\mu\widetilde{\mu}$ into a control category for free.

The vantage point of expressing the semantics in a (co-)control category is that the syntax is in a purely logical style (with the disjunction, the conjunction,...) and forgets about the continuations. The computational interpretation of $\overline{\lambda}\mu\widetilde{\mu}$ with continuations where the semantics comes from is completely transparent and the latter appears in a purely algebraic way. As Selinger suggested, we see that in the CBV-semantics should rather be interpreted in a co-control category, since the arrow goes *"the wrong way"*.

$$K^{\mathsf{CBV}}_{A_1} \otimes \ldots \otimes K^{\mathsf{CBV}}_{A_n} \xrightarrow{\Lambda[\![\overline{c}]\!]} K^{\mathsf{CBV}}_{B_1} + \ldots + K^{\mathsf{CBV}}_{B_m}$$

$$C^{\mathsf{CBN}}_{A_1} \times \ldots \times C^{\mathsf{CBN}}_{A_n} \xrightarrow{\Lambda[\![c]\!]} C^{\mathsf{CBN}}_{B_1} \wp \ldots \wp C^{\mathsf{CBN}}_{B_m}$$

where $\wp$ and $\otimes$ are respectively the binoidal functors of the control category and the co-control category.

4 Curry-Howard correspondence for a step-by-step cut-elimination

In the remark 4.1 of [CH00], Curien and Herbelin give a hint on a way to connect $\mathsf{LK}_{\mu\widetilde{\mu}}$ and LK. The proofs of LK embed in $\mathsf{LK}_{\mu\widetilde{\mu}}$ by considering the following sub-syntax of $\overline{\lambda}\mu\widetilde{\mu}$:

$$c ::= \langle x|\ \alpha\rangle |\ \langle \lambda x.\mu\alpha.c|\ \beta\rangle |\ \langle y|\ \mu\alpha.c \cdot \widetilde{\mu}x.c\rangle |\ \langle \mu\alpha.c|\ \widetilde{\mu}x.c\rangle$$

which can be formalized in a simplified syntax, that we shall call (the pure terms of) $\lambda\xi$:

$$M ::= \langle x \cdot \alpha\rangle |\ \widehat{x}M\widehat{\alpha} \cdot \beta |\ M\widehat{\alpha}\,[y]\,\widehat{x}M |\ M\widehat{\alpha} \dagger \widehat{x}M$$

where $\widehat{x}M\widehat{\alpha}\cdot\beta$ binds x and α in M, $M\widehat{\alpha}\,[y]\,\widehat{x}M'$ and $M\widehat{\alpha}\dagger\widehat{x}M'$ bind α in M and x in M'. Again we consider terms up to α-conversion and when we write a term, we always suppose that it satisfies Barendregt's convention. We shall call $^{\mu}$ the obvious embedding of (the pure terms of) $\lambda\xi$ into $\overline{\lambda}\mu\widetilde{\mu}$.

The typing rules are inherited from $\overline{\lambda}\mu\widetilde{\mu}$, and are hence isomorphic to the implicational version of LK with weakening moved in axiom rules and contraction associated to introduction rules. We can always suppose that the constructed formula was already amongst the formulae of the premisses, provided we introduced it at every application of the axiom rule thanks to the weakening.

Definition 4.1 [Typing system of $\lambda\xi$]

$$\frac{}{\langle x\cdot\alpha\rangle : (\Gamma, x : A \vdash \alpha : A, \Delta)}$$

$$\frac{M : (\Gamma, x : A \vdash \alpha : B, \alpha' : A \to B, \Delta)}{(\widehat{x}M\widehat{\alpha}\cdot\alpha') : (\Gamma \vdash \alpha' : A \to B, \Delta)}$$

$$\frac{M : (\Gamma, x' : A \to B \vdash \Delta, \alpha : A) \quad M' : (\Gamma, x' : A \to B, x : B \vdash \Delta)}{M\widehat{\alpha}\,[x']\,\widehat{x}M' : (\Gamma, x' : A \to B \vdash \Delta)}$$

$$\frac{M : (\Gamma \vdash \Delta, \alpha : A) \quad M' : (\Gamma, x : A \vdash \Delta)}{M\widehat{\alpha}\dagger\widehat{x}M' : (\Gamma \vdash \Delta)}$$

The chosen syntax is meant to express an idea that we would like to develop in the future, namely that we like to see the terms as processes and the reductions as communications. Intuitively, the axiom is denoted by $\langle x\cdot\alpha\rangle$, connecting directly an *input channel* x to an *output channel* α. The right introduction is denoted by $\widehat{y}M\widehat{\beta}\cdot\alpha$, meaning that when M receives something (say of type A) on its (private) channel y, it returns something on its (private) channel β (say of type B), thus generating on a specific channel α a transformer from A to B. The left introduction is denoted by $M_1\widehat{\alpha}\,[y]\,\widehat{x}M_2$, meaning that M_1 tries to communicate with M_2, but since α (say of type A) and x (say of type B) do not necessarily fit in their type, a transformer from A to B is expected on channel y to enable the communication. The cut is denoted by $M_1\widehat{\alpha}\dagger\widehat{x}M_2$ and connects the output channel α of M_1 with the input channel x of M_2. In what follows we shall see that it also behaves as an explicit substitution operator.

Unfortunately, the reductions in $\overline{\lambda}\mu\widetilde{\mu}$ do not eliminate every applications of the cut-rule, and that is because the command operator, of which it is the typing rule, has a double role: it is either used to give a name to a formula without creating a redex, or to create a redex typed by a cut that can be eliminated.

The sub-syntax $\lambda\xi$ constrains the system $\overline{\lambda}\mu\widetilde{\mu}$ by naming every subformula. It is interesting to see how it splits the use of the command. Indeed, the terms of $\lambda\xi$ that are normal forms in $\overline{\lambda}\mu\widetilde{\mu}$ are exactly:

$$M ::= \langle x \cdot \alpha \rangle | \ \widehat{x}M\widehat{\alpha} \cdot \beta | \ M\widehat{\alpha}\,[y]\,\widehat{x}M$$

where the command is only used to give a name to the formula that has just been manipulated. And $M\widehat{\alpha} \dagger \widehat{x}M'$ is always a redex in $\overline{\lambda}\mu\widetilde{\mu}$, and more than a redex, it is a critical pair.

Now this syntax $\lambda\xi$ also appears as a sub-syntax of Urban's proof-describing syntax; so using his terminology, it gives

$$\begin{aligned}
\langle x \cdot \alpha \rangle &= \mathsf{Ax}(x, \alpha) \\
\widehat{x}M\widehat{\alpha} \cdot \beta &= \mathsf{Imp}_R((x)\langle\alpha\rangle M, \beta) \\
M\widehat{\alpha}\,[y]\,\widehat{x}M' &= \mathsf{Imp}_L(\langle\alpha\rangle M, (x)M', y) \\
M\widehat{\alpha} \dagger \widehat{x}M' &= \mathsf{Cut}(\langle\alpha\rangle M, (x)M')
\end{aligned}$$

which defines the so-called *pure* terms, to which he adds
$\overleftarrow{\mathsf{Cut}}(\langle\alpha\rangle M, (x)M')$ (now $M\widehat{\alpha}\dagger_{\mathsf{V}}\widehat{x}M'$) and $\overrightarrow{\mathsf{Cut}}(\langle\alpha\rangle M, (x)M')$ (now $M\widehat{\alpha}\dagger_{\mathsf{N}}\widehat{x}M'$). They are the *active* versions of the (*inactive*) explicit substitution $M\widehat{\alpha} \dagger \widehat{x}M'$, in the sense that they are being computed in a CBV or CBN way, respectively, as we shall see. They are also typed by the cut-rule.
Note that the translation $^{\mu}$ is only defined on pure terms.

Definition 4.2 [Reduction system of $\lambda\xi$]
Now the reduction relation is the closure under any context of the following rules:

Logical rules:

$$\begin{aligned}
\langle y \cdot \alpha\rangle\widehat{\alpha} \dagger \widehat{x}\langle x \cdot \beta\rangle &\to \langle y \cdot \beta\rangle \\
(\widehat{y}M\widehat{\beta} \cdot \alpha)\widehat{\alpha} \dagger \widehat{x}\langle x \cdot \beta'\rangle &\to \widehat{y}M\widehat{\beta} \cdot \beta' && \text{if } \alpha \notin FV(M) \\
\langle z \cdot \alpha\rangle\widehat{\alpha} \dagger \widehat{x}(M_1\widehat{\beta}\,[x]\,\widehat{y}M_2) &\to M_1\widehat{\beta}\,[z]\,\widehat{y}M_2 && \text{if } x \notin FV(M_1) \cup FV(M_2) \\
(\widehat{y}M_1\widehat{\alpha} \cdot \gamma)\widehat{\gamma} \dagger \widehat{z}(M_2\widehat{\beta}\,[z]\,\widehat{x}M_3) &\begin{cases} \to\ M_2\widehat{\beta} \dagger \widehat{y}(M_1\widehat{\alpha} \dagger \widehat{x}M_3) \\ \to\ (M_2\widehat{\beta} \dagger \widehat{y}M_1)\widehat{\alpha} \dagger \widehat{x}M_3 \end{cases} \\
& \text{if } \gamma \notin FV(M_1) \text{ and } z \notin FV(M_2) \cup FV(M_3)
\end{aligned}$$

CBV propagation:

$$\begin{aligned}
\langle y\cdot\alpha\rangle\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M &\to \langle y\cdot\alpha\rangle\widehat{\alpha}\,\dagger\,\widehat{x}M && \\
\langle y\cdot\beta\rangle\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M &\to \langle y\cdot\beta\rangle && \text{if } \beta\neq\alpha\\
(\widehat{y}M'\widehat{\beta}\cdot\alpha)\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M &\to (\widehat{y}(M'\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M)\widehat{\beta}\cdot\alpha)\widehat{\alpha}\,\dagger\,\widehat{x}M && \\
(\widehat{y}M'\widehat{\beta}\cdot\gamma)\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M &\to \widehat{y}(M'\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M)\widehat{\beta}\cdot\gamma && \text{if } \gamma\neq\alpha\\
(M_1\widehat{\beta}\,[z]\,\widehat{y}M_2)\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M &\to (M_1\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M)\widehat{\beta}\,[z]\,\widehat{y}(M_2\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M) && \\
(M_1\widehat{\beta}\,\dagger\,\widehat{y}\langle y\cdot\alpha\rangle)\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M &\to (M_1\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M)\widehat{\beta}\,\dagger\,\widehat{x}M && \\
(M_1\widehat{\beta}\,\dagger\,\widehat{y}M_2)\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M &\to (M_1\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M)\widehat{\beta}\,\dagger\,\widehat{y}(M_2\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M) && \textit{otherwise}
\end{aligned}$$

CBN propagation:

$$\begin{aligned}
M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}\langle x\cdot\beta\rangle &\to \langle y\cdot\alpha\rangle\widehat{\alpha}\,\dagger\,\widehat{x}M && \\
M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}\langle y\cdot\beta\rangle &\to \langle y\cdot\beta\rangle && \text{if } y\neq x\\
M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}(\widehat{y}M'\widehat{\beta}\cdot\gamma) &\to \widehat{y}(M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}M')\widehat{\beta}\cdot\gamma && \\
M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}(M_1\widehat{\beta}\,[x]\,\widehat{y}M_2) &\to M\widehat{\alpha}\,\dagger\,\widehat{x}((M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}M_1)\widehat{\beta}\,[x]\,\widehat{y}(M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}M_2)) && \\
M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}(M_1\widehat{\beta}\,[z]\,\widehat{y}M_2) &\to (M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}M_1)\widehat{\beta}\,[z]\,\widehat{y}(M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}M_2) && \text{if } z\neq x\\
M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}(\langle x\cdot\beta\rangle\widehat{\beta}\,\dagger\,\widehat{y}M_2) &\to M\widehat{\alpha}\,\dagger\,\widehat{y}(M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}M_2) && \\
M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}(M_1\widehat{\beta}\,\dagger\,\widehat{y}M_2) &\to (M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}M_1)\widehat{\beta}\,\dagger\,\widehat{y}(M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}M_2) && \textit{otherwise}
\end{aligned}$$

Activating the cuts and choosing the strategy:

$$M\widehat{\alpha}\,\dagger\,\widehat{x}M' \to M\widehat{\alpha}\,\dagger_{\mathsf{V}}\,\widehat{x}M' \text{ if } M \text{ does not freshly introduce } \alpha$$
$$M\widehat{\alpha}\,\dagger\,\widehat{x}M' \to M\widehat{\alpha}\,\dagger_{\mathsf{N}}\,\widehat{x}M' \text{ if } M' \text{ does not freshly introduce } x$$

where M freshly introduces x only if $M\equiv M'\widehat{\beta}\,[x]\,\widehat{y}M''$ with $x\notin FV(M')\cup FV(M'')$ or $M\equiv\langle x\cdot\beta\rangle$ and M freshly introduces α only if $M\equiv\widehat{y}M'\widehat{\beta}\cdot\alpha$ with $\alpha\notin FV(M')$ or $M\equiv\langle y\cdot\alpha\rangle$.

This reduction system which we close under context is (a subsystem of) Urban's $(\mathcal{T}^{\leftrightarrow},\xrightarrow{loc})$ [Urb00]. The CBV (resp. CBN) evaluation consists in forbidding the CBN-activation (resp. CBV) of an inactive cut when the latter could be CBV-activated (resp. CBN-activated).

Now we see that the reduction rules move the explicit substitution towards the variables of a term step-by-step, and correspondingly, the cut-rule is pushed towards the axiom rules. In fact, this is a restriction of Urban's system, but still, a term is a normal form if and only if it does not contain an explicit substitution (which represents, through Curry-Howard correspondence, a proof without cut). Note that M is a normal form in $\lambda\xi$ if and only

if M^μ is a normal form in $\overline{\lambda}\mu\widetilde{\mu}$.

For instance, in this system, if M is pure (i.e. if it contains no active cut), then

$$M\widehat{\alpha} \dagger \widehat{x}\langle x \cdot \beta\rangle \longrightarrow^* M[\alpha := \beta]$$
$$\langle y \cdot \alpha\rangle\widehat{\alpha} \dagger \widehat{x}M \longrightarrow^* M[x := y]$$

Urban allows those reductions as a one-step rule for any M.

The basic property that this reduction system has is the Subject Reduction [Urb00], which is required if we want to see it as proof transformations: the reductions preserve the typing. The proof describes in detail how to move the cut-rule upwards and exhibits explicitly the connection between reductions and cut-elimination.

Urban proved in [Urb00] the strong normalization of $(\mathcal{T}^{\leftrightarrow}, \xrightarrow{loc})$: if a term M is typable, then there is no infinite sequence of reductions starting from it. He first proves the strong normalization of a system with implicit substitutions, adapting the technique of the symmetric reducibility candidates. Using this lemma, he encodes some terms of $\lambda\xi$ into a first-order syntax with a well-founded rpo-order and concludes that any reduction sequence starting with a pure term is finite. Then he generalizes to all the terms by showing that each of them is a reduced form of a suitable pure term (where the cuts have been subtly deactivated). This gives a proof of the cut-elimination in sequent calculus.

5 Strong Normalization of $\overline{\lambda}\mu\widetilde{\mu}$

Although $\lambda\xi$ is a sub-syntax of $\overline{\lambda}\mu\widetilde{\mu}$, $\overline{\lambda}\mu\widetilde{\mu}$ can be encoded into $\lambda\xi$ as follows:

Definition 5.1 [Translation of $\overline{\lambda}\mu\widetilde{\mu}$ into $\lambda\xi$]

$$\begin{array}{llll}
(x)^\xi_\alpha & := \langle x \cdot \alpha\rangle & (\alpha)^\xi_x & := \langle x \cdot \alpha\rangle \\
(\lambda x.v)^\xi_\alpha & := \widehat{x}(v)^\xi_\beta\widehat{\beta} \cdot \alpha & (v \cdot e)^\xi_x & := (v)^\xi_\alpha\widehat{\alpha}\,[x]\,\widehat{y}(e)^\xi_y \\
(\mu\beta.c)^\xi_\alpha & := (c[\beta := \alpha])^\xi & (\widetilde{\mu}y.c)^\xi_x & := (c[y := x])^\xi
\end{array}$$

$$\begin{array}{ll}
\langle x|\ e\rangle^\xi & := (e)^\xi_x \\
\langle v|\ \alpha\rangle^\xi & := (v)^\xi_\alpha \\
\langle v|\ e\rangle^\xi & := (v^\xi_\alpha)\widehat{\alpha} \dagger \widehat{x}(e^\xi_x) \\
\multicolumn{2}{l}{\text{in every other case}}
\end{array}$$

The translation is well-defined, since on the one hand $\langle x|\ \alpha\rangle^\xi = (\alpha)^\xi_x = \langle x \cdot \alpha\rangle$ and on the other hand $\langle x|\ \alpha\rangle^\xi = (x)^\xi_\alpha = \langle x \cdot \alpha\rangle$ as well. Note that with this translation, we only get pure terms. Of course, when the translation introduces a new bound variable, we choose a fresh one (ie, not free in the term that we translate).

$^\xi$ clearly preserves the type (if $c : (\Gamma \vdash \Delta)$, then $c^\xi : (\Gamma \vdash \Delta)$), and although $^\xi$ is clearly not injective, it is surjective on pure terms, since $(M^\mu)^\xi =_\alpha M$ if M is pure. Now we can prove the following by simultaneous structural induction:

Lemma 5.2

- $(c)^\xi\widehat{\alpha} \dagger_{\mathsf{V}} \widehat{x}(e)^\xi_x \longrightarrow^* (c[\alpha := e])^\xi$ *where* x *is fresh.*
  $(v)^\xi_\alpha\widehat{\alpha} \dagger_{\mathsf{N}} \widehat{x}(c)^\xi \longrightarrow^* (c[x := v])^\xi$ *where* α *is fresh.*
- $(v')^\xi_\beta\widehat{\alpha} \dagger_{\mathsf{V}} \widehat{x}(e)^\xi_x \longrightarrow^* (v'[\alpha := e])^\xi_\beta$ *where* x *is fresh and* $\alpha \neq \beta$.
  $(v)^\xi_\alpha\widehat{\alpha} \dagger_{\mathsf{N}} \widehat{x}(v')^\xi_\beta \longrightarrow^* (v'[x := v])^\xi_\beta$ *where* α *is fresh.*
- $(e')^\xi_y\widehat{\alpha} \dagger_{\mathsf{V}} \widehat{x}(e)^\xi_x \longrightarrow^* (e'[\alpha := e])^\xi_y$ *where* x *is fresh.*
  $(v)^\xi_\alpha\widehat{\alpha} \dagger_{\mathsf{N}} \widehat{x}(e')^\xi_y \longrightarrow^* (e'[x := v])^\xi_y$ *where* α *is fresh and* $x \neq y$.

Using this lemma, we can now prove the simulation of the reductions in $\overline{\lambda}\mu\widetilde{\mu}$.

Theorem 5.3 $c \longrightarrow c'$ *implies* $c^\xi \longrightarrow^* c'^\xi$. *Furthermore, the case where* $c^\xi \equiv c'^\xi$ *only happens when the size of* c *is strictly greater than that of* c'.

Proof For instance, the μ-rule: $\langle\mu\alpha.c|e\rangle \to c[\alpha := e]$.

- If e is not a variable and c^ξ does not freshly introduce α, then $\langle\mu\alpha.c|e\rangle^\xi \equiv (c^\xi)\widehat{\alpha} \dagger \widehat{x}(e)^\xi_x \longrightarrow (c^\xi)\widehat{\alpha} \dagger_{\mathsf{V}} \widehat{x}(e)^\xi_x$ and we can apply the previous lemma.
- If e is not a variable and $c^\xi = \langle y \cdot \alpha\rangle$, then $\langle\mu\alpha.c|e\rangle^\xi \equiv \langle y \cdot \alpha\rangle\widehat{\alpha} \dagger \widehat{x}(e)^\xi_x \longrightarrow^* (e)^\xi_x[x := y] \equiv (e)^\xi_y \equiv \langle y|e\rangle^\xi$, because $(e)^\xi_x$ is pure.
- In every other case $\langle\mu\alpha.c|e\rangle^\xi \equiv (c[\alpha := e])^\xi$ and the size of $\langle\mu\alpha.c|e\rangle$ is strictly greater than that of $c[\alpha := e]$.

□

Finally, that is all we need to prove the strong normalization of $\overline{\lambda}\mu\widetilde{\mu}$.

Theorem 5.4 *Typed* $\overline{\lambda}\mu\widetilde{\mu}$*-calculus is strongly normalizing.*

Proof Suppose an infinite reduction chain: $c_1 \longrightarrow \cdots \longrightarrow c_i \longrightarrow \cdots$. Using the previous lemma, $c_1^\xi \longrightarrow^* \cdots \longrightarrow^* c_i^\xi \longrightarrow^* \cdots$. Since $^\xi$ preserves the typing and Urban has proved the strong normalization of his system, for a certain j, all the c^ξ_{j+n} are equal. But then, the size of c_{j+n} strictly decreases when n increases, which is impossible. □

6 Semantics of the step-by-step cut-elimination

Since the encoding $^\mu$ is only defined on pure terms, we need to define the inactivation $\mathcal{I}(M)$ of a term M, which is the pure term M in which every active cut (either CBV or CBN) has been replaced by an inactive cut.

Then we define three sets of terms: $\mathcal{P}$ is the set of pure typed terms, $\mathcal{P}_{\mathsf{CBV}}$ and $\mathcal{P}_{\mathsf{CBN}}$ are the closures of $\mathcal{P}$ under $\longrightarrow_{\mathsf{CBV}}$ and $\longrightarrow_{\mathsf{CBN}}$ respectively, both containing the pure terms. Those terms are intuitively the well-formed terms according to a strategy. Indeed, those are the terms for which the inactivation is reversible: only using the activation rules, we have
$\mathcal{I}(M) \longrightarrow^*_{\mathsf{CBV}} M$ if $M \in \mathcal{P}_{\mathsf{CBV}}$ and $\mathcal{I}(M) \longrightarrow^*_{\mathsf{CBN}} M$ if $M \in \mathcal{P}_{\mathsf{CBN}}$.

We define the CBV-translation (resp. CBN-translation) of $\mathcal{P}_{\mathsf{CBV}}$ (resp. $\mathcal{P}_{\mathsf{CBN}}$) into CPS as follows:

the interpretation of types is just as for the semantics of $\overline{\lambda}\mu\widetilde{\mu}$,
and if $M \in \mathcal{P}_{\mathsf{CBV}}$ (resp. $M \in \mathcal{P}_{\mathsf{CBN}}$), then $\overline{M} := \overline{\mathcal{I}(M)^\mu}$ (resp. $\underline{M} := \underline{\mathcal{I}(M)^\mu}$).

Since $\mathcal{I}$ and $^\mu$ preserve the typing, we have again the preservation of types by the translations:

$$M : (\Gamma \vdash \Delta) \Rightarrow \begin{cases} V_\Gamma^{\mathsf{CBV}}, K_\Delta^{\mathsf{CBV}} \vdash \overline{M} : R \\ C_\Gamma^{\mathsf{CBN}}, K_\Delta^{\mathsf{CBN}} \vdash \underline{M} : R \end{cases}$$

Now the important point is the following:

Theorem 6.1 (Validity with respect to $\overline{\lambda}\mu\widetilde{\mu}$)
If $M \in \mathcal{P}_{\mathsf{CBV}}$ and $M \longrightarrow_{\mathsf{CBV}} N$ then $\mathcal{I}(M)^\mu$ and $\mathcal{I}(N)^\mu$ are CBV-joinable.
If $M \in \mathcal{P}_{\mathsf{CBN}}$ and $M \longrightarrow_{\mathsf{CBN}} N$ then $\mathcal{I}(M)^\mu$ and $\mathcal{I}(N)^\mu$ are CBN-joinable.

This can be proven by giving, for each reduction rule in $\lambda\xi$, the command in $\overline{\lambda}\mu\widetilde{\mu}$ to which $\mathcal{I}(M^\mu)$ and $\mathcal{I}(N^\mu)$ both reduce. Hence we can derive from the preservation of the semantics in $\overline{\lambda}\mu\widetilde{\mu}$ the preservation of the semantics of $\lambda\xi$, namely:

Corollary 6.2 (Preservation of the semantics in $\lambda\xi$)
if $M \to_{\mathsf{CBV}} N$ then $\overline{M} =_{\alpha,\beta} \overline{N}$
if $M \to_{\mathsf{CBN}} N$ then $\underline{M} =_{\alpha,\beta} \underline{N}$

As another corollary, we also get for free the validity with respect to the big-step operational semantics:
Supposing the confluence of the CBV and CBN evaluations in $\overline{\lambda}\mu\widetilde{\mu}$,
for all pure M and all normal form N,
if $M \longrightarrow^*_{\mathsf{CBV}} N$ then $M^\mu \longrightarrow^*_{\mathsf{CBV}} N^\mu$ and N^μ is a normal form
if $M \longrightarrow^*_{\mathsf{CBN}} N$ then $M^\mu \longrightarrow^*_{\mathsf{CBV}} N^\mu$ and N^μ is a normal form.

This means that the reductions in $\lambda\xi$ simulate the reductions in $\overline{\lambda}\mu\widetilde{\mu}$, as we expected. Note that the confluence of the CBV and CBN evaluations in $\overline{\lambda}\mu\widetilde{\mu}$ is only conjectured, but now that we have the strong normalization of the typed $\overline{\lambda}\mu\widetilde{\mu}$, only the local confluence remains to be proven.

Finally, we can interpret the $\lambda\xi$-calculus categorically just as we did for $\overline{\lambda}\mu\widetilde{\mu}$, by interpreting CPS into a continuation category. We give here as a result what it gives. For CBV:

$$\begin{aligned}
[\![\langle x \cdot \alpha \rangle : (\Gamma, A \vdash A, \Delta)]\!]^{\mathsf{K}}_{\mathsf{V}} &:= V_\Gamma \times V_A \times K_A \times K_\Delta \xrightarrow{\langle \pi_{K_A}, \pi_{V_A} \rangle} K_A \times V_A \xrightarrow{\epsilon} R \\
[\![\widehat{x} M \widehat{\alpha} \cdot \alpha' : (\Gamma \vdash A \to B, \Delta)]\!]^{\mathsf{K}}_{\mathsf{V}} &:= V_\Gamma \times K_{A\to B} \times K_\Delta \\
&\qquad \xrightarrow{\mathsf{Id}_{K_{A\to B}} \times \Lambda[\![M]\!]^{\mathsf{K}}_{\mathsf{V}}} K_{A\to B} \times R^{V_A \times K_B} \xrightarrow{\epsilon} R \\
[\![M \widehat{\alpha}\, [x']\, \widehat{x} M' : (\Gamma, A \to B \vdash \Delta)]\!]^{\mathsf{K}}_{\mathsf{V}} &:= V_\Gamma \times V_{A\to B} \times K_\Delta \\
&\qquad \xrightarrow{\mathsf{Id}_{V_{A\to B}} \times \langle \mathsf{Id}_{V_\Gamma \times K_\Delta},\, \Lambda[\![M']\!]^{\mathsf{K}}_{\mathsf{V}} \rangle} V_{A\to B} \times V_\Gamma \times K_\Delta \times R^{V_B} \\
&\qquad \xrightarrow{\mathsf{Id}_{V_\Gamma \times K_\Delta} \times \epsilon} V_\Gamma \times K_\Delta \times K_A \xrightarrow{[\![M]\!]^{\mathsf{K}}_{\mathsf{V}}} R \\
[\![M \widehat{\alpha} \dagger \widehat{x} M' : (\Gamma \vdash \Delta)]\!]^{\mathsf{K}}_{\mathsf{V}} &:= V_\Gamma \times K_\Delta \xrightarrow{\langle \mathsf{Id},\, \Lambda[\![M']\!]^{\mathsf{K}}_{\mathsf{V}} \rangle} V_\Gamma \times K_\Delta \times R^{V_A} \xrightarrow{[\![M]\!]^{\mathsf{K}}_{\mathsf{V}}} R
\end{aligned}$$

and for CBN:

$$\begin{aligned}
[\![\langle x\cdot\alpha\rangle : (\Gamma, A\vdash A,\Delta)]\!]^{\mathsf{K}}_{\mathsf{V}} &:= C_\Gamma\times C_A\times K_A\times K_\Delta \xrightarrow{\langle\pi_{C_A},\pi_{K_A}\rangle} C_A\times K_A \xrightarrow{\epsilon} R\\
[\![\widehat{x}M\widehat{\alpha}\cdot\alpha' : (\Gamma\vdash A\to B,\Delta)]\!]^{\mathsf{K}}_{\mathsf{V}} &:= C_\Gamma\times K_{A\to B}\times K_\Delta\\
&\qquad \xrightarrow{\langle\pi_{K_{A\to B}},\Lambda[\![M]\!]^{\mathsf{K}}_{\mathsf{N}}\rangle} K_{A\to B}\times R^{K_B\times C_A} \xrightarrow{\epsilon} R\\
[\![M\widehat{\alpha}\,[x']\,\widehat{x}M' : (\Gamma, A\to B\vdash\Delta)]\!]^{\mathsf{K}}_{\mathsf{V}} &:= C_\Gamma\times C_{A\to B}\times K_\Delta \xrightarrow{\langle\pi_{C_{A\to B}},\,\Lambda f\rangle} C_{A\to B}\times R^{V_{A\to B}}\\
&\qquad \xrightarrow{\epsilon} R\\
[\![M\widehat{\alpha}\,\dagger\,\widehat{x}M' : (\Gamma\vdash\Delta)]\!]^{\mathsf{K}}_{\mathsf{V}} &:= C_\Gamma\times K_\Delta \xrightarrow{\langle \mathsf{Id},\,\Lambda[\![M]\!]^{\mathsf{K}}_{\mathsf{N}}\rangle} C_\Gamma\times K_\Delta\times R^{K_A} \xrightarrow{[\![M']\!]^{\mathsf{K}}_{\mathsf{N}}} R
\end{aligned}$$

where f is:

$$\begin{aligned}
C_\Gamma\times C_{A\to B}\times K_\Delta\times V_{A\to B} &\xrightarrow{\langle \mathsf{Id},\,\Lambda[\![M]\!]^{\mathsf{K}}_{\mathsf{N}}\rangle\times\mathsf{Id}_{V_{A\to B}}} C_\Gamma\times C_{A\to B}\times K_\Delta\times R^{K_A}\times V_{A\to B}\\
&\xrightarrow{\mathsf{Id}\times\epsilon} C_\Gamma\times C_{A\to B}\times K_\Delta\times C_B \xrightarrow{[\![M']\!]^{\mathsf{K}}_{\mathsf{N}}} R
\end{aligned}$$

And again, we get the semantics in a control category and in a co-control category by curryfying the above semantics as follows:

$$[\![M : (\Gamma\vdash\Delta)]\!]^{\mathsf{C}}_{\mathsf{V}} := \Lambda\left(V_\Gamma\times K_\Delta \xrightarrow{[\![M:(\Gamma\vdash\Delta)]\!]^{\mathsf{K}}_{\mathsf{V}}} R\right) = K_\Delta \xrightarrow{\Lambda[\![M:(\Gamma\vdash\Delta)]\!]^{\mathsf{K}}_{\mathsf{V}}} K_\Gamma$$

$$[\![M : (\Gamma\vdash\Delta)]\!]^{\mathsf{C}}_{\mathsf{N}} := \Lambda\left(C_\Gamma\times K_\Delta \xrightarrow{[\![M:(\Gamma\vdash\Delta)]\!]^{\mathsf{K}}_{\mathsf{N}}} R\right) = C_\Gamma \xrightarrow{\Lambda[\![M:(\Gamma\vdash\Delta)]\!]^{\mathsf{K}}_{\mathsf{N}}} C_\Delta$$

Conclusion

We have managed to encode $\lambda\xi$ into $\overline{\lambda}\mu\widetilde{\mu}$ and vice-versa. We have used one translation to prove the strong normalization of $\overline{\lambda}\mu\widetilde{\mu}$ from that of $\lambda\xi$, we have identified two strategies in $\lambda\xi$ corresponding to the call-by-name and call-by-value evaluations, and having done that, we have been able to give them a denotational semantics of continuations by translating them into CPS and then interpreting CPS into categories. This completes the correspondence *à la* Curry-Howard by connecting the logical features of LK to the computational features of a programming language.

It is likely that the CBV and CBN evaluations are each confluent. This is true when the permutation of cuts is not allowed, by we can only conjecture the property when it is.

The CBN-semantics could also be given using Hofmann and Streicher's interpretation of types, as it seems that a better symmetry with the CBV-semantics should be thus expected. It should also be interesting to extend the semantics to other connectives. It is not difficult to add the constant *false*, or $\bot$, but we would like to investigate what happens with the quantifiers.

More generally, in the long term, we would like to work out a system with dependent types, in the aim of reaching a system in the spirit of coq, which,

being based on classical sequent calculus and explicit substitutions, could have interesting and efficient features, for instance in proof search.

We would also like to push forward the idea of seeing proof as processes and see to what extent we can thus express the main features of concurrency and mobility.

Acknowledgement

The author is very grateful to Samson Abramsky, Delia Kesner, Pierre Lescanne, Steffen Van Bakel, Dan Dougherty and Norman Danner for their help and advice.

References

[BR95] R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN '95—Computing Science in the Netherlands*, pages 62–72, Koninklijke Jaarbeurs, Utrecht, November 1995.

[CH00] P.-L. Curien and H. Herbelin. The duality of computation. In *Proceedings of the* 5^{th} *ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, pages 233–243. ACM Press, 2000.

[Fil89] Andrzej Filinski. Declarative continuations and categorical duality. Master's thesis, DIKU, Computer Science Department, University of Copenhagen, August 1989. DIKU Rapport 89/11.

[Gen35] G. Gentzen. Investigations into logical deduction. In *Gentzen collected works*. Ed M. E. Szabo, North Holland, 68ff (1969), 1935.

[Her95] H. Herbelin. *Séquents qu'on calcule*. PhD thesis, Université Paris 7, 1995.

[HS97] M. Hofmann and T. Streicher. Continuation models are universal for $\lambda\mu$-calculus. In *Proc. of Logic in Computer Science (LICS)*, pages 387–397, 1997.

[Par92] M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *Proc. of the International Conference on Logic Programming and Automated Reasoning (LPAR)*. Lecture Notes in Computer Science 1, 1992.

[Sel99] P. Selinger. Control categories and duality: on the categorical semantics of the $\lambda\mu$-calculus. *Mathematical Structures in Computer Science*, 1999.

[Urb00] C. Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, 2000.

Simulating Liveness by Reduction Strategies

Jürgen Giesl [1]

LuFG Informatik II, RWTH Aachen, Ahornstr. 55, 52074 Aachen, Germany

Hans Zantema [2]

Department of Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

We define a general framework to handle liveness and related properties by reduction strategies in abstract reduction and term rewriting. Classically, reduction strategies in rewriting are used to simulate the evaluation process in programming languages. The aim of our work is to use reduction strategies to also study liveness questions which are of high importance in practice (e.g., in protocol verification for distributed processes). In particular, we show how the problem of verifying liveness is related to termination of term rewrite systems (TRSs). Using our results, techniques for proving termination of TRSs can be used to verify liveness properties.

1 Introduction

In this paper, we give a formal definition of *safety* and *liveness* using the framework of abstract reduction (Sect. 2). In particular, liveness is formalized by imposing a suitable reduction strategy. In Sect. 3 we show how the reduction-based definitions of safety and liveness correspond to standard definitions from the literature [1]. Then in Sect. 4 the notion of liveness is specialized to the framework of term rewriting. In Sect. 5 we investigate the connection between liveness and termination. More precisely, we show how termination of ordinary rewriting is related to termination under the reduction strategy required for liveness properties. To this end, we present a transformation such that termination of the transformed TRS is equivalent to the liveness property of the original TRS. In [11], a similar transformation was presented for liveness properties of a certain form (global liveness), but we show that such transformations can also be given for other liveness properties (local liveness). With these results, (existing) termination techniques for TRSs can be used to infer

[1] Email: giesl@informatik.rwth-aachen.de
[2] Email: h.zantema@tue.nl

liveness. So our approach differs from most previous applications of rewriting techniques in process verification which were mainly concerned with the verification of other properties (e.g., reachability [4,7] or equivalence [5,13]).

2 Formalizing Safety and Liveness

We define safety and liveness using the framework of abstract reduction. Let S be a set of states and let $\rightarrow \subseteq S \times S$ where "$t \rightarrow u$" means that a computation step from t to u is possible. A *computation sequence* or *reduction* is a finite sequence $t_0, t_1, \ldots, t_n$ or an infinite sequence $t_0, t_1, t_2, \ldots$ with $t_i \rightarrow t_{i+1}$. As usual, $\rightarrow^*$ is the reflexive transitive closure of $\rightarrow$. To define safety and liveness we assume a set $G \subseteq S$ of *goal states* and a set $I \subseteq S$ of *initial states*.

2.1 Formalizing Safety

To formalize safety, the goal states G are considered to be "good" and safety means that something bad will not happen. In other words, in every reduction starting from an initial state all states are good.

Definition 2.1 (Safety) $\mathsf{Safe}(I, \rightarrow, G)$ iff $\forall t \in I, u \in S : (t \rightarrow^* u) \Rightarrow u \in G$.

Usually safety properties are proved by choosing an invariant. Then one proves that the invariant is true initially and that the invariant holds for every state u where $t \rightarrow u$ for some state t satisfying the invariant. In this way safety is proved in a purely local way: only one-step reductions are considered in the proof. In other words, safety is proved by induction on the length of the reduction, and a claim about arbitrary reductions can be made by analyzing only one-step reductions. It is a natural question whether this approach covers all safety properties. The next theorem answers this question positively. Here, the set G' represents the set of all states satisfying the invariant.

Theorem 2.2 (Proving Safety) $\mathsf{Safe}(I, \rightarrow, G)$ *iff there is a set* $G' \subseteq S$ *with*

- $I \subseteq G' \subseteq G$, *and*
- $\forall t \in G', u \in S : (t \rightarrow u) \Rightarrow u \in G'$.

Proof. For the "if"-part, let G' satisfy the properties above and $t = t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_n = u$ for $t \in I$. By induction on i, one can show that $t_i \in G'$ for every $i = 0, \ldots, n$. This implies $t_n = u \in G' \subseteq G$, which we had to prove.

For the "only if"-direction we assume $\mathsf{Safe}(I, \rightarrow, G)$ and let $G' = \{u \in S \mid \exists t \in I : t \rightarrow^* u\}$. Now we show that G' satisfies the properties above.

- $I \subseteq G'$ holds since $t \rightarrow^* t$,
- $G' \subseteq G$ is implied by $\mathsf{Safe}(I, \rightarrow, G)$, and
- $\forall t \in G', u \in S : (t \rightarrow u) \Rightarrow u \in G'$ follows from the definition of G'. □

Note that the "only if"-direction cannot be proved simply by defining $G' = G$, since in general, G is not invariant. For instance, if $S = \{1, 2, 3\}$, $\rightarrow = \{(1, 1), (2, 3)\}$, $I = \{1\}$ and $G = \{1, 2\}$, then $\mathsf{Safe}(I, \rightarrow, G)$, but $u \in G$ does

not always hold if $t \in G$ and $t \to u$. Essentially this corresponds to the difference between "always true" and "invariant" as pointed out in [14].

2.2 Formalizing Liveness

Next we define liveness properties which state that some goal will eventually be reached. In order to formalize "eventuality" we need to consider maximal reductions that continue as long as possible. More precisely, a reduction is called *maximal* if it is either infinite or if its last element is in the set of *normal forms* $\mathsf{NF} = \{t \in S \mid \neg\exists u : t \to u\}$. The liveness property $\mathsf{Live}(I, \to, G)$ holds if every maximal reduction starting in I contains an element of G.

Definition 2.3 (Liveness) $\mathsf{Live}(I, \to, G)$ holds iff

(i) $\forall t_0, t_1, t_2, \ldots : (t_0 \in I \wedge \forall i \in \mathbb{N} : t_i \to t_{i+1}) \Rightarrow \exists i \in \mathbb{N} : t_i \in G$, and

(ii) $\forall t_0, t_1, \ldots, t_n : (t_0 \in I \wedge t_n \in \mathsf{NF} \wedge \forall i \in \{0, \ldots, n-1\} : t_i \to t_{i+1}) \Rightarrow \exists i \in \{0, \ldots, n\} : t_i \in G.$

For example, *termination* (or *strong normalization*) is a special liveness property describing the non-existence of infinite reductions, i.e.,

$$\mathsf{SN}(I, \to) \;=\; \neg(\exists t_0, t_1, t_2, \ldots : \; t_0 \in I \wedge \forall i \in \mathbb{N} : t_i \to t_{i+1}).$$

Theorem 2.4 (SN is a Liveness Prop. [11]) $\mathsf{SN}(I, \to)$ *iff* $\mathsf{Live}(I, \to, \mathsf{NF})$.

The next theorem states a kind of converse. Here, we impose a reduction strategy such that "$\to$" may only proceed if the current state is not in G.

Definition 2.5 ($\to_G$) Let $\to_G \subseteq S \times S$ where $t \to_G u$ iff $t \to u$ and $t \notin G$.

Now one can show that $\mathsf{Live}(I, \to, G)$ is equivalent to $\mathsf{SN}(I, \to_G)$. The "only if"-part holds without any further conditions. For the "if"-part, G must contain all normal forms $\mathsf{NF}(I)$ reachable from I, where $\mathsf{NF}(I) = \{u \in \mathsf{NF} \mid \exists t \in I : t \to^* u\}$. Otherwise, if there is a terminating sequence $t_0 \to \ldots \to t_n$ with all $t_i \notin G$, we might have $\mathsf{SN}(I, \to_G)$ but not $\mathsf{Live}(I, \to, G)$.

Theorem 2.6 (Equivalence of Liveness and Termination [11])
Let $\mathsf{NF}(I) \subseteq G$. *Then* $\mathsf{Live}(I, \to, G)$ *holds iff* $\mathsf{SN}(I, \to_G)$ *holds.*

By Thm. 2.6 we can verify actual liveness properties: if $\mathsf{NF}(I) \subseteq G$, then one can instead verify termination of $\to_G$. If $\mathsf{NF}(I) \not\subseteq G$, then $\mathsf{SN}(I, \to_G)$ still implies liveness for all infinite computations. In Sect. 5 we discuss how techniques to prove full termination of TRSs can be used for termination of $\to_G$.

3 Connection to Previous Formalization

In this section we show that our notions of *safety* and *liveness* specialize the "standard" definitions of Alpern and Schneider [1]. In their framework, a property P is a set of infinite sequences of states. Terminating executions of a program are represented by repeating the final state infinitely often. According to [1], a property P is a *safety property* iff the following condition holds:

(1) If $\langle t_0, t_1, \ldots\rangle \notin P$, then there is an $i \in \mathbb{N}$ such that
for all $\langle t_{i+1}, t_{i+2}, \ldots\rangle$ we have $\langle t_0, \ldots, t_i, t_{i+1}, t_{i+2}, \ldots\rangle \notin P$.

In other words, if an infinite sequence $\langle t_0, t_1, \ldots\rangle$ does not satisfy a safety property P, then there is a finite prefix $\langle t_0, \ldots, t_i\rangle$ of the sequence which already violates it. So irrespective of how this finite prefix is extended to an infinite sequence $\langle t_0, \ldots, t_i, t_{i+1}, t_{i+2}, \ldots\rangle$, the property P is not fulfilled.

A property P is a *liveness property* according to [1] iff every finite prefix of states can be extended into an infinite sequence satisfying P. Formally, the requirement for liveness properties is the following:

(2) For all $\langle t_0, \ldots, t_i\rangle$ there exist $\langle t_{i+1}, t_{i+2}, \ldots\rangle$ such that
$\langle t_0, \ldots, t_i, t_{i+1}, t_{i+2}, \ldots\rangle \in P$.

In contrast to safety, this liveness definition does not require that there is a discrete point in the infinite sequence from which on the liveness condition is always fulfilled. To ease the checking of a liveness property (in the rewriting framework), it turns out to be useful to add this demand. We call P a *discrete* property iff it satisfies the following discreteness condition:

(3) If $\langle t_0, t_1, \ldots\rangle \in P$, then there is an $i \in \mathbb{N}$ such that
for all $\langle t_{i+1}, t_{i+2}, \ldots\rangle$ we have $\langle t_0, \ldots, t_i, t_{i+1}, t_{i+2}, \ldots\rangle \in P$.

In other words, if an infinite sequence satisfies the property P, then this is due to a finite prefix $\langle t_0, \ldots, t_i\rangle$. No matter how this prefix is extended, the property will always be preserved. A *discrete liveness* property is a property satisfying both (2) and (3). An example for a non-discrete liveness property is starvation freedom which states that a process makes progress infinitely often.

There is a natural correspondence between safety and discreteness: P is a safety property iff P's complement is a discreteness property. This indicates that it is often more intuitive to use discreteness as the definition of "liveness". Indeed, one may argue that non-discrete liveness properties like starvation freedom should rather be called *fairness* instead of *liveness properties*.

Now we compare the definitions of [1] to our formalizations in Sect. 2. We defined safety and liveness not as properties of arbitrary sequences of states, but of sequences representing computation. So if P_2 is a safety or liveness property of sequences of states and P_1 is the property that a sequence of states corresponds to a particular computation, then we formulate statements like "$P_1 \subseteq P_2$" expressing that P_1 implies P_2. Moreover, in our approach the "*goals*" to be reached are made explicit as properties of states. We demonstrate that our notions of safety and liveness nevertheless correspond to the notions of [1] provided that their liveness definition is restricted to *discrete* liveness.

First, we show that our safety and liveness concepts can be simulated by the concepts of Alpern and Schneider. In other words, our definitions of safety and liveness are also safety and (discrete) liveness properties according to [1]. Here, $P_{I\rightarrow}$ is the property that a sequence of states corresponds to a computation starting in I. Then $\mathsf{Safe}(I, \rightarrow, G)$ can be formulated as "$P_{I\rightarrow} \subseteq P_G^{\mathsf{safe}}$" and $\mathsf{Live}(I, \rightarrow, G)$ can be formulated as "$P_{I\rightarrow} \subseteq P_G^{\mathsf{live}}$" for suitable safety and

liveness properties P_G^{safe} and P_G^{live} in the framework of Alpern and Schneider.

Theorem 3.1 (Simulating Def. 2.1 and 2.3 in [1]) *Let $G \neq \varnothing$. We define the following properties (i.e., sets of infinite sequences of states):*

- $P_{I\rightarrow} = \{\langle t_0, t_1, \ldots\rangle \mid t_0 \in I,\ \forall i \in \mathbb{N} : t_i \rightarrow t_{i+1}$ *or* $t_i \in \mathsf{NF}$ *and* $t_i = t_{i+1}\}$
- $P_G^{\mathsf{safe}} = \{\langle t_0, t_1, \ldots\rangle \mid t_i \in G$ *for all* $i \in \mathbb{N}\}$
- $P_G^{\mathsf{live}} = \{\langle t_0, t_1, \ldots\rangle \mid t_i \in G$ *for some* $i \in \mathbb{N}\}$

Then we have:

(a) $P_{I\rightarrow}$ *and* P_G^{safe} *are safety properties,* P_G^{live} *is a discrete liveness property.*

(b) $\mathsf{Safe}(I, \rightarrow, G)$ *iff* $P_{I\rightarrow} \subseteq P_G^{\mathsf{safe}}$

(c) $\mathsf{Live}(I, \rightarrow, G)$ *iff* $P_{I\rightarrow} \subseteq P_G^{\mathsf{live}}$

Proof.

(a) If $\langle t_0, t_1, \ldots\rangle \notin P_{I\rightarrow}$, then we have $t_0 \notin I$ or $t_i \notin \mathsf{NF}$ and $t_i \not\rightarrow t_{i+1}$ or $t_i \in \mathsf{NF}$ and $t_i \neq t_{i+1}$ for some i. In the first case, every infinite sequence starting with the prefix t_0 will not be in $P_{I\rightarrow}$. In the second and third case, every infinite sequence starting with $\langle t_0, \ldots, t_i, t_{i+1}\rangle$ will not be in $P_{I\rightarrow}$. Hence, $P_{I\rightarrow}$ is indeed a safety property (i.e., it satisfies (1)).

If $\langle t_0, t_1, \ldots\rangle \notin P_G^{\mathsf{safe}}$, then there is a $t_i \notin G$. So every infinite sequence starting with $\langle t_0, \ldots, t_i\rangle$ is not in P_G^{safe} and thus, P_G^{safe} is a safety property.

Let $\langle t_0, \ldots, t_i\rangle$ be arbitrary states. Since $G \neq \varnothing$, there is a $t_{i+1} \in G$. So every finite prefix $\langle t_0, \ldots, t_i\rangle$ can be extended to an infinite sequence $\langle t_0, \ldots, t_i, t_{i+1}, \ldots\rangle$ satisfying P_G^{live}. Hence, P_G^{live} is a liveness property.

Let $\langle t_0, t_1, \ldots\rangle \in P_G^{\mathsf{live}}$. Then there is a $t_i \in G$. Hence, if one extends $\langle t_0, \ldots, t_i\rangle$ by an arbitrary sequence $\langle t_{i+1}, t_{i+2}, \ldots\rangle$ we again have $\langle t_0, \ldots, t_i, t_{i+1}, t_{i+2}, \ldots\rangle \in P_G^{\mathsf{live}}$. Thus, P_G^{live} is a discrete liveness property.

(b) For "only if", let $\langle t_0, t_1, \ldots\rangle \in P_{I\rightarrow}$. So $t_0 \in I$ and for all i we have $t_0 \rightarrow^* t_i$. By $\mathsf{Safe}(I, \rightarrow, G)$ this implies $t_i \in G$. So we obtain $\langle t_0, t_1, \ldots\rangle \in P_G^{\mathsf{safe}}$.

For the "if"-direction, let $t_0 \in I$ and $t_0 \rightarrow^* t_i$. Then there exists an infinite sequence $\langle t_0, \ldots, t_i, \ldots\rangle \in P_{I\rightarrow}$. So $P_{I\rightarrow} \subseteq P_G^{\mathsf{safe}}$ implies $t_i \in G$.

(c) For the "only if"-direction, let $\langle t_0, t_1, \ldots\rangle \in P_{I\rightarrow}$ and therefore $t_0 \in I$. If $t_i \rightarrow t_{i+1}$ for all i, then by $\mathsf{Live}(I, \rightarrow, G)$ there is a $t_i \in G$. If there is a $t_n \in \mathsf{NF}$ and $t_n = t_m$ for all $m > n$, then by $\mathsf{Live}(I, \rightarrow, G)$ there is a $t_i \in G$ for $i \leq n$. Therefore, in both cases we obtain $\langle t_0, t_1, \ldots\rangle \in P_G^{\mathsf{live}}$.

For "if', let $t_0 \in I$. If $t_0 \rightarrow t_1 \rightarrow ...$ is infinite, then $\langle t_0, t_1, ...\rangle \in P_{I\rightarrow} \subseteq P_G^{\mathsf{live}}$ and $t_i \in G$ for some i. Similarly, if $t_0 \rightarrow ... \rightarrow t_n$ for $t_n \in \mathsf{NF}$, then $\langle t_0, ..., t_n, t_n, t_n, \ldots\rangle \in P_{I\rightarrow} \subseteq P_G^{\mathsf{live}}$ and therefore $t_i \in G$ for some i. □

Now we prove the converse, i.e., all safety and (discrete) liveness properties of [1] can also be expressed in our framework. In fact, one can even simulate every discrete property by a liveness property in our framework. So if P^{safe} is a safety property and P^{live} is a discrete property in the framework of [1] (i.e., P^{safe} satisfies (1) and P^{live} satisfies (3)), then for any property P, "$P \subseteq P^{\mathsf{safe}}$" is a safety property and "$P \subseteq P^{\mathsf{live}}$" is a liveness property in our framework.

Thus, these two frameworks are equally powerful for examining properties "$P_1 \subseteq P_2$" of computations (provided that the liveness definition of [1] is restricted to *discrete* liveness). In [1], whether the *goal* has been achieved does not only depend on a state itself but it may also depend on the way that state is reached. In contrast to that, in our approach being a *goal* is just a property of the state alone. However, Alpern and Schneider's concept of *goals* can be simulated in our framework as well. For that purpose, the notion of state can be extended until it covers all relevant information for being a goal. So instead of the original states we now consider finite tuples of states $(t_0, \ldots, t_i)$ which stand for "state t_i, if it has been reached by the sequence $\langle t_0, \ldots, t_i \rangle$". The initial "tuple-state" is the empty tuple $(\,)$. To encode questions like "$P_1 \subseteq P_2$" in our framework, we simulate the first property P_1 by a relation $\rightarrow_{P_1}$ on tuple-states. Here, $\rightarrow_{P_1}$ builds up elements of P_1 step by step.

Theorem 3.2 (Simulating [1] in Def. 2.1 and 2.3) *Let $P \neq \varnothing$ be an arbitrary property, let P^{safe} satisfy (1), and let P^{live} satisfy (3). Let $S' = \{(t_0, \ldots, t_n) \mid t_i \in S, n \geq 0\}$ and let $G^{\mathsf{safe}}, G^{\mathsf{live}} \subseteq S'$ with*

- $G^{\mathsf{safe}} = \{(t_0, \ldots, t_i) \mid \langle t_0, \ldots, t_i, t_{i+1}, \ldots \rangle \in P^{\mathsf{safe}} \text{ for some } \langle t_{i+1}, \ldots \rangle\}$
- $G^{\mathsf{live}} = \{(t_0, \ldots, t_i) \mid \langle t_0, \ldots, t_i, t_{i+1}, \ldots \rangle \in P^{\mathsf{live}} \text{ for all } \langle t_{i+1}, \ldots \rangle\}$

We define the relation $\rightarrow_P \subseteq S' \times S'$ as $(t_0, \ldots, t_i) \rightarrow_P (t_0, \ldots, t_i, t_{i+1})$ iff there exist $\langle t_{i+2}, \ldots \rangle$ such that $\langle t_0, \ldots, t_i, t_{i+1}, t_{i+2}, \ldots \rangle \in P$. Then we have:

(a) $P \subseteq P^{\mathsf{safe}}$ *iff* $\mathsf{Safe}((\,), \rightarrow_P, G^{\mathsf{safe}})$

(b) $P \subseteq P^{\mathsf{live}}$ *iff* $\mathsf{Live}((\,), \rightarrow_P, G^{\mathsf{live}})$

Proof.

(a) For the "if"-direction, let $\langle t_0, t_1, \ldots \rangle \in P$. So we have $(\,) \rightarrow_P (t_0) \rightarrow_P (t_0, t_1) \rightarrow_P \ldots$ By $\mathsf{Safe}((\,), \rightarrow_P, G^{\mathsf{safe}})$ this implies $(t_0, \ldots, t_i) \in G^{\mathsf{safe}}$ for all i. Assume that $\langle t_0, t_1, \ldots \rangle \notin P^{\mathsf{safe}}$. Since P^{safe} is a safety property, there is an i such that for all $\langle t_{i+1}, t_{i+2}, \ldots \rangle$ we have $\langle t_0, \ldots, t_i, t_{i+1}, t_{i+2}, \ldots \rangle \notin P^{\mathsf{safe}}$. This implies $(t_0, \ldots, t_i) \notin G^{\mathsf{safe}}$ which is a contradiction.

For "only if", let $(\,) \rightarrow_P^* (t_0, \ldots, t_i)$. By definition of $\rightarrow_P$, there is $\langle t_{i+1}, t_{i+2}, \ldots \rangle$ with $\langle t_0, \ldots, t_i, t_{i+1}, \ldots \rangle \in P \subseteq P^{\mathsf{safe}}$. Hence, $(t_0, \ldots, t_i) \in G^{\mathsf{safe}}$.

(b) For "if", let $\langle t_0, t_1, \ldots \rangle \in P$, i.e., $(\,) \rightarrow_P (t_0) \rightarrow_P (t_0, t_1) \rightarrow_P \ldots$ By definition, $\rightarrow_P$ has no normal forms. By $\mathsf{Live}((\,), \rightarrow_P, G^{\mathsf{live}})$ there is an i with $(t_0, \ldots, t_i) \in G^{\mathsf{live}}$. So for all infinite extensions of $\langle t_0, \ldots, t_i \rangle$, the resulting sequence is in P^{live}. In particular, we obtain $\langle t_0, t_1, \ldots \rangle \in P^{\mathsf{live}}$.

For "only if", let $(\,) \rightarrow_P (t_0) \rightarrow_P (t_0, t_1) \rightarrow_P \ldots$ By definition of $\rightarrow_P$ we have $\langle t_0, \ldots \rangle \in P \subseteq P^{\mathsf{live}}$. Since P^{live} satisfies (3), there is an i such that *for all* $\langle t_{i+1}, t_{i+2}, \ldots \rangle$ we have $\langle t_0, \ldots, t_i, t_{i+1}, t_{i+2}, \ldots \rangle \in P^{\mathsf{live}}$. In other words, $(t_0, \ldots, t_i) \in G^{\mathsf{live}}$ for some i and thus $\mathsf{Live}((\,), \rightarrow_P, G^{\mathsf{live}})$. □

4 Liveness in Term Rewriting

Now we focus on liveness in rewriting. More precisely, we study the property

$\mathsf{Live}(I, \to_R, G)$ where $\to_R$ is the rewrite relation corresponding to a TRS R. For an introduction to term rewriting, the reader is referred to [3], for example.

Let Σ be a signature with at least one constant and let $\mathcal{V}$ be a set of variables. $\mathcal{T}(\Sigma, \mathcal{V})$ is the set of terms over Σ and $\mathcal{V}$ and $\mathcal{T}(\Sigma)$ is the set of ground terms. Now $\mathcal{T}(\Sigma, \mathcal{V})$ represents computation states and $G \subseteq \mathcal{T}(\Sigma, \mathcal{V})$.

By Thm. 2.6, $\mathsf{Live}(I, \to, G)$ is equivalent to $\mathsf{SN}(I, \to_G)$, if $\mathsf{NF}(I) \subseteq G$. To verify liveness, we want to prove termination of $\to_G$ by approaches for termination proofs of ordinary TRSs. But due to the reduction strategy in the definition of $\to_G$, classical termination techniques are not applicable directly. In Sect. 5 we present a transformation from a TRS R and a set G of terms to a TRS R' such that $\to_G$ terminates iff the rewrite relation $\to_{R'}$ terminates. A transformation where "if" holds is called *sound* and if the "only if"-direction holds, it is called *complete*. The existence of a sound and complete transformation means that liveness and termination are essentially equivalent.

Depending on the form of G, different transformations have to be developed. We concentrate on two kinds of liveness properties: *local liveness* where G is closed under contexts and substitutions and *global liveness* where the complement of G is closed under contexts and substitutions. In global liveness, the property of the term to be reached eventually is that a certain pattern does not occur *anywhere* in the term, which is a global property of the term. In local liveness, the desired property is that a certain pattern occurs *somewhere* in the term, being a local property. Clearly, there exist liveness properties which do not belong to our classes of local or global liveness. However, in [11] and in the following sections, we demonstrate that local and global liveness indeed capture many interesting liveness properties.

4.1 Global Liveness

A liveness property $\mathsf{Live}(I, \to_R, G)$ is called *global* if G has the form [3]

$$G = \{t \mid t \text{ does not contain an instance of } p\} \qquad \text{for some term } p.$$

In other words, G consists of all terms which cannot be written as $C[p\sigma]$ for any context C and substitution σ. As before, $t \to_G u$ holds iff $t \to_R u$ and $t \notin G$. So a term t may be reduced whenever it contains an instance of the term p. Note that for sets G as above, the relation $\to_G$ is a rewrite relation (i.e., it is closed under substitutions and contexts). This also makes clear that ground termination of $\to_G$ is equivalent to full termination of $\to_G$.

A typical global liveness property is that eventually all processes requesting a resource are granted access, cf. [11, Sect. 5.3]. Here, the network of processes is described by a term t and processes that have not yet gained access to the resource are represented as an instance of the subterm $\mathsf{old}(x)$. The aim is to prove that eventually, t reduces to a term without old. This form of liveness is called "global" since the goal situation is stated as a condition on *all* processes.

[3] The approach can easily be extended to sets G of the form $\{t \mid t$ does not contain instances of p_1, p_2, …, or $p_n\}$ for terms $p_1, \ldots, p_n$.

For arbitrary terms and TRSs, $\to_G$ is not useful: if there is a symbol f of arity > 1 or if p contains a variable x (i.e., if $p = C[x]$ for some context C), then termination of $\to_G$ implies termination of the full rewrite relation $\to_R$. The reason is that any infinite reduction $t_0 \to_R t_1 \to_R \cdots$ gives rise to an infinite reduction $f(t_0, p, \ldots) \to_R f(t_1, p, \ldots) \to_R \cdots$ or $C[t_0] \to_R C[t_1] \to_R \cdots$ where in both cases none of the terms is in G. Therefore we concentrate on the particular case of *top rewrite systems* in which there is a designated unary symbol tp which may only occur on the root position of terms. Moreover, if the root of one side of a rule is tp, then the other side must also start with tp.

Top rewrite systems typically suffice to model networks of processes, since the whole network is represented by a top term [8]. Clearly, in top rewrite systems, top terms can only be reduced to top terms again. In such systems we consider properties $\mathsf{Live}(\mathcal{T}_{\mathsf{top}}, \to_R, G)$, where $\mathcal{T}_{\mathsf{top}}$ is the set of ground terms with tp on root position. So the goal is to prove that every maximal reduction of ground top terms contains a term without an instance of p. Transformations by which this can be treated are elaborated extensively in [11].

4.2 Local Liveness

In the remainder we concentrate on *local liveness*, where G has the form [4]

$$G \;=\; \{t \mid t \text{ contains an instance of } p\} \quad \text{for some term } p.$$

So in local liveness, G is closed under substitutions and contexts, whereas in global liveness the complement of G is closed under substitutions and contexts. Now $t \to_G u$ holds if $t \to_R u$ and t contains no instance of p. A typical local liveness property is that eventually at least *one* process requesting a resource is granted access, rather than requiring this for all processes as in global liveness.

Example 4.1 (Waiting Lines) This TRS describes the behavior of two waiting lines of processes. The combination of the lines has a bounded size, i.e., a new process can only enter a waiting line if some process was "served". The processes in the lines are served on a "first in - first out" basis. So at the front end of a waiting line, a process may be served, where serving is denoted by a constant serve. If a process is served, its place in the line is replaced by a free place, denoted by free. If the place in front of some process is free, this process may take the free place, creating a free place on its original position. If a line has a free place at its back end, a new process new may enter *any* waiting line and the free place is deleted. Apart from new processes represented by new we also consider old processes represented by old, which were already in the line initially. Introducing the binary symbol tp having the representations of the waiting lines as its arguments, this network is described by the following top rewrite system R. Here, the leftmost symbol of a term represents the "back end" of the waiting line and the rightmost symbol is the "front end".

[4] Again, the approach can easily be extended to sets G of the form $\{t \mid t \text{ contains an instance of } p_1, p_2, \text{ or } p_n\}$ for terms $p_1, \ldots, p_n$.

$$\mathsf{tp}(\mathsf{free}(x), y) \to \mathsf{tp}(\mathsf{new}(x), y) \qquad \mathsf{new}(\mathsf{free}(x)) \to \mathsf{free}(\mathsf{new}(x))$$
$$\mathsf{tp}(\mathsf{free}(x), y) \to \mathsf{tp}(x, \mathsf{new}(y)) \qquad \mathsf{old}(\mathsf{free}(x)) \to \mathsf{free}(\mathsf{old}(x))$$
$$\mathsf{tp}(x, \mathsf{free}(y)) \to \mathsf{tp}(\mathsf{new}(x), y) \qquad \mathsf{new}(\mathsf{serve}) \to \mathsf{free}(\mathsf{serve})$$
$$\mathsf{tp}(x, \mathsf{free}(y)) \to \mathsf{tp}(x, \mathsf{new}(y)) \qquad \mathsf{old}(\mathsf{serve}) \to \mathsf{free}(\mathsf{serve})$$

For various variations of this system we proved global liveness with respect to $p = \mathsf{old}(x)$ in [11], stating that eventually *all* old clients will be served. However, for this version this global liveness property does not hold: we have an infinite reduction of top terms all containing the symbol old:

$$\mathsf{tp}(\mathsf{new}(\mathsf{serve}), \mathsf{old}(\mathsf{serve})) \to_R \mathsf{tp}(\mathsf{free}(\mathsf{serve}), \mathsf{old}(\mathsf{serve})) \to_R$$
$$\mathsf{tp}(\mathsf{new}(\mathsf{serve}), \mathsf{old}(\mathsf{serve})) \to_R \ldots$$

But we can prove the weaker local liveness property that eventually *some* client will be served. In our formalism this is done by choosing $G = \{t \mid t$ contains an instance of $\mathsf{free}(\mathsf{serve})\}$, since $\mathsf{free}(\mathsf{serve})$ always occurs after serving a client.

Now $\to_G$ is not a rewrite relation since it is neither closed under contexts nor under substitutions. Moreover, ground termination of $\to_G$ does not imply termination of $\to_G$. To permit the restriction to ground terms, when regarding local liveness, we always assume that our signature contains at least an extra constant c and an extra unary function symbol h which do not appear in p. In this case, ground termination and full termination of $\to_G$ are equivalent.

5 Transformations for Local Liveness

Now we investigate the correspondence between liveness and termination in the framework of term rewriting. Then all existing techniques for termination proofs of TRSs (including future developments) can be used for liveness properties. A first step into this direction was taken in [8], where the termination proof technique of *dependency pairs* was used to verify certain liveness properties of telecommunication processes. However, we now develop an approach to connect liveness and termination in general. While in [11] we considered global liveness, we now show that a similar approach is possible for local liveness, although $\to_G$ is no longer a rewrite relation. Hence, let $G = \{t|$ t contains an instance of $p\}$. Our goal is to present a transformation L such that $\mathsf{SN}(\to_G)$ iff $\mathsf{SN}(L(R,p))$, i.e., iff all $L(R,p)$-reductions are terminating. [5]

As a first approach we choose a sound transformation L_s defined by

$$L_s(R,p) = \{\ l \to r \in R \mid p \text{ does not occur in } r\ \}.$$

Clearly, rules creating instances of p may not be applied in infinite $\to_G$-reductions. Hence, $\mathsf{SN}(L_s(R,p))$ implies $\mathsf{SN}(\to_G)$, i.e., L_s is sound. To prove liveness in Ex. 4.1, for $p = \mathsf{free}(\mathsf{serve})$, $L_s(R,p)$ consists of R's first 6 rules and termination is easily proved by dependency pairs [2].

[5] To ease the presentation, we only present transformations without regarding initial states I. However, our transformations can easily be extended to take I into account.

However, L_s is not complete: $\mathsf{SN}(\to_G)$ does not imply $\mathsf{SN}(L_s(R,p))$. As an example let $R = \{\mathsf{f}(x) \to \mathsf{f}(\mathsf{f}(x))\}$ and let $p = \mathsf{f}^{10}(x)$. In the sequel we will see that $\mathsf{SN}(\to_G)$ holds. However, $L_s(R,p) = R$ is clearly not terminating. The rest of this section is devoted to a sound and complete transformation L.

The construction of $L(R,p)$ is motivated by an existing transformation [9,10] which was developed for a completely different purpose (termination of context-sensitive rewriting). To this end we introduce a fresh binary symbol mat, fresh unary symbols tp, chk, active, mark, no, and for every variable in p we introduce one fresh constant. We write $\overline{p}$ for the ground term obtained by replacing every variable in p by its corresponding fresh constant. Let Σ_G denote the resulting extended signature. As in [9,10], the symbol active is used to specify potential next redexes and mark means that the its argument must be inspected in order to identify those subterms which may be reduced next. But in [9,10], this depends on the position of a subterm (only subterms in "active" positions of all function symbols above them are marked with active). In contrast, now all subterms that do not contain an instance of p may be marked with active.

To simplify the rules for mat, we restrict ourselves to linear terms p, i.e., terms in which every variable occurs at most once. Here, $\mathsf{mat}(\overline{p},t)$ checks whether p does not match t. A corresponding transformation is also possible for non-linear terms p, but then mat would have to take the instantiation of p's variables into account. Now there would be more cases where matching of p with t fails, i.e., it also fails if a matcher would have to instantiate two occurrences of a variable by different terms.

We define the TRS $L(R,p)$ to consist of the following rules.

$$
\begin{aligned}
\mathsf{active}(l) &\to \mathsf{mark}(r) \quad \text{for all rules } l \to r \text{ in } R\\
\mathsf{tp}(\mathsf{mark}(x)) &\to \mathsf{tp}(\mathsf{chk}(\mathsf{mat}(\overline{p},x)))\\
\mathsf{mat}(f(x_1,\ldots,x_n),f(y_1,\ldots,y_n)) &\to f(y_1,\ldots,\mathsf{mat}(x_i,y_i),\ldots,y_n)\\
&\quad \text{for } f\in\Sigma \text{ of arity } n>0 \text{ in } p,\ 1\le i\le n\\
\mathsf{mat}(f(x_1,\ldots,x_n),g(y_1,\ldots,y_m)) &\to \mathsf{no}(g(y_1,\ldots,y_m))\\
&\quad \text{for } f,g\in\Sigma,\ f \text{ occurs in } p,\ f\neq g\\
f(x_1,\ldots,\mathsf{no}(x_i),\ldots,x_n) &\to \mathsf{no}(f(x_1,\ldots,x_n))\\
&\quad \text{for } f\in\Sigma \text{ of arity } n>0,\ 1\le i\le n\\
\mathsf{chk}(\mathsf{no}(f(x_1,\ldots,x_n))) &\to f(\mathsf{chk}(\mathsf{mat}(\overline{p},x_1)),\ldots,\mathsf{chk}(\mathsf{mat}(\overline{p},x_n)))\\
&\quad \text{for } f\in\Sigma \text{ of arity } n>0\\
\mathsf{chk}(\mathsf{no}(c)) &\to \mathsf{active}(c) \quad \text{for } c\in\Sigma \text{ of arity } 0\\
f(\mathsf{active}(x_1),\ldots,\mathsf{active}(x_n)) &\to \mathsf{active}(f(x_1,..,x_n)) \text{ for } f\in\Sigma, \text{ arity } n>0\\
f(\mathsf{active}(x_1),..,\mathsf{mark}(x_i),..,\mathsf{active}(x_n)) &\to \mathsf{mark}(f(x_1,\ldots,x_n))\\
&\quad \text{for } f\in\Sigma \text{ of arity } n>0,\ 1\le i\le n
\end{aligned}
$$

Theorem 5.1 *Let $p \in \mathcal{T}(\Sigma, \mathcal{V})$ be linear. The relation $\to_G$ is terminating if and only if the TRS $L(R, p)$ is terminating.*

The proof of Thm. 5.1 is given in the appendix. As an example, let R again consist of the rule $\mathsf{f}(x) \to \mathsf{f}(\mathsf{f}(x))$ and let $p = \mathsf{f}^{10}(x)$. We required Σ to contain at least one extra constant c and one extra unary symbol h. To ease the presentation, here we omit h and let $\Sigma = \{\mathsf{c}, \mathsf{f}\}$. We obtain $\overline{p} = \mathsf{f}^{10}(X)$ where X is a fresh constant. $L(R, p)$ consists of the rules

$$\begin{aligned}
\mathsf{active}(\mathsf{f}(x)) &\to \mathsf{mark}(\mathsf{f}(\mathsf{f}(x))) & \mathsf{chk}(\mathsf{no}(\mathsf{f}(x))) &\to \mathsf{f}(\mathsf{chk}(\mathsf{mat}(\mathsf{f}^{10}(X), x)))\\
\mathsf{mat}(\mathsf{f}(x), \mathsf{f}(y)) &\to \mathsf{f}(\mathsf{mat}(x, y)) & \mathsf{chk}(\mathsf{no}(\mathsf{c})) &\to \mathsf{active}(\mathsf{c})\\
\mathsf{mat}(\mathsf{f}(x), \mathsf{c}) &\to \mathsf{no}(\mathsf{c}) & \mathsf{f}(\mathsf{active}(x)) &\to \mathsf{active}(\mathsf{f}(x))\\
\mathsf{f}(\mathsf{no}(x)) &\to \mathsf{no}(\mathsf{f}(x)) & \mathsf{f}(\mathsf{mark}(x)) &\to \mathsf{mark}(\mathsf{f}(x))\\
\mathsf{tp}(\mathsf{mark}(x)) &\to \mathsf{tp}(\mathsf{chk}(\mathsf{mat}(\mathsf{f}^{10}(X), x)))
\end{aligned}$$

Now the step $\mathsf{f}(\mathsf{c}) \to_G \mathsf{f}(\mathsf{f}(\mathsf{c}))$ transforms to the following reduction in $L(R, p)$:

$$\begin{aligned}
&\mathsf{tp}(\mathsf{mark}(\mathsf{f}(\mathsf{c}))) \to \mathsf{tp}(\mathsf{chk}(\mathsf{mat}(\mathsf{f}^{10}(X), \mathsf{f}(\mathsf{c})))) \to\\
&\mathsf{tp}(\mathsf{chk}(\mathsf{f}(\mathsf{mat}(\mathsf{f}^{9}(X), \mathsf{c})))) \to \mathsf{tp}(\mathsf{chk}(\mathsf{f}(\mathsf{no}(\mathsf{c})))) \to\\
&\mathsf{tp}(\mathsf{chk}(\mathsf{no}(\mathsf{f}(\mathsf{c})))) \to \mathsf{tp}(\mathsf{f}(\mathsf{chk}(\mathsf{mat}(\mathsf{f}^{10}(X), \mathsf{c})))) \to\\
&\mathsf{tp}(\mathsf{f}(\mathsf{chk}(\mathsf{no}(\mathsf{c})))) \to \mathsf{tp}(\mathsf{f}(\mathsf{active}(\mathsf{c}))) \to\\
&\mathsf{tp}(\mathsf{active}(\mathsf{f}(\mathsf{c}))) \to \mathsf{tp}(\mathsf{mark}(\mathsf{f}(\mathsf{f}(\mathsf{c}))))
\end{aligned}$$

In general, for a ground term $t \in \mathcal{T}(\Sigma)$, an $L(R, p)$-reduction of $\mathsf{tp}(\mathsf{mark}(t))$ starts with checking whether p matches t. If not, then on every position below the root, a chk is created, after which a similar matching check can be done on the direct subterms. This goes on until leaves are reached. Hence, active is created only if none of the subterms on the path to the leaf is matched by p. Then these active-symbols may move back to the root, while during this process exactly one R-step may be applied, changing active into mark. (Several R-steps are impossible, since mark can only be propagated upwards in terms $f(\mathsf{active}(x_1), ..., \mathsf{mark}(x_i), ..., \mathsf{active}(x_n))$ where only one argument of f has the root mark.) The TRS $L(R, p)$ is designed in such a way that infinite reductions are only possible if this process is repeated infinitely often. In our example one can prove termination of $L(R, p)$ by the dependency pair method [2].

6 Conclusion

In this paper, we showed how to define safety and liveness in terms of abstract reduction and rewriting. In particular, liveness is defined by imposing a suitable reduction strategy. We have shown that our definitions are comparable with existing definitions of safety and liveness in the literature [1]. While safety properties can be proved in a local way (Thm. 2.2), for proving liveness properties, it is useful to investigate the connection between liveness and ter-

mination: It turns out that liveness and termination are essentially equivalent. Depending on the form of the liveness condition, different transformations can be given such that termination of the transformed TRS is equivalent to the desired liveness property to be proved. This means that techniques for termination analysis of ordinary TRSs can be used to verify liveness properties.

While the formalization of liveness using abstract reduction was already presented in [11], we extended the results of [11] by regarding safety (Sect. 2), by comparing our notions with the ones of Alpern and Schneider [1] (Sect. 3), and by introducing local liveness (Sect. 4 and 5).

To increase the practical applicability of this approach, instead of the sound and complete transformation given in the paper, one may use simpler sound (but incomplete) transformations like L_s, since proving termination of $L(R,p)$ can be very hard in general. Then termination of the transformed TRS still implies the desired local liveness property, but not vice versa. In [11] we already presented such techniques for global liveness and we were able to perform automated proofs of liveness properties for interesting process protocols (e.g., networks with shared resources and a token ring protocol). In contrast to model checking and related methods, this approach does not require finiteness of the state space. We plan to refine and to develop such approaches further in future work.

A Appendix: The Proof of Theorem 5.1

We first present auxiliary lemmata required in the proof of Thm. 5.1. Lemma A.1 describes the behavior of mat and it is the only lemma where the linearity restriction on p is used. As explained in Sect. 5, a similar transformation (with more complicated mat-rules) is also possible for non-linear terms p.

Lemma A.1 (Reductions with mat) *For $p \in \mathcal{T}(\Sigma,\mathcal{V})$, p linear, $t \in \mathcal{T}(\Sigma)$, we have $\mathsf{mat}(\overline{p},t) \to^+_{L(R,p)} \mathsf{no}(u)$ iff $t = u$ and no substitution σ satisfies $p\sigma = t$.*

Proof. We apply induction on the structure of p. The cases where p is a constant or a variable, or p and t have distinct root symbols follow directly from the analysis of the shape of the rules. For the remaining case $p = f(p_1,\ldots,p_n)$, $t = f(t_1,\ldots,t_n)$ we need the induction hypothesis and the property that for a linear term $p = f(p_1,\ldots,p_n)$ we have $\exists\sigma\forall i : p_i\sigma = t_i \iff \forall i\exists\sigma : p_i\sigma = t_i$. □

The next lemma shows that the rule $\mathsf{tp}(\mathsf{mark}(x)) \to \mathsf{tp}(\mathsf{chk}(\mathsf{mat}(\overline{p},x)))$ is crucial for the termination behavior of $L(R,p)$.

Lemma A.2 (Termination Behavior of $L(R,p)$) *Let $L'(R,p) = L(R,p) \setminus \{\mathsf{tp}(\mathsf{mark}(x)) \to \mathsf{tp}(\mathsf{chk}(\mathsf{mat}(\overline{p},x)))\}$. Then $L'(R,p)$ is terminating.*

Proof. Let R_1 consist of the mat-rules and the rules of the form $f(x_1,\ldots,\mathsf{no}(x_i),\ldots,x_n) \to \mathsf{no}(f(x_1,\ldots,x_n))$ from $L(R,p)$. Termination of R_1 can be shown by the RPO [6] with the precedence $\mathsf{mat} > f > \mathsf{no}$ for all $f \in \Sigma$.

We define a filtering fil where $\mathsf{fil}(t)$ is the normal form with respect to the two rules $\mathsf{no}(x) \to x$ and $\mathsf{mat}(x,y) \to y$. Moreover, let R_2 consist of

$$\begin{aligned}
\mathsf{active}(l) &\to \mathsf{mark}(r)\\
\mathsf{chk}(f(x_1,\ldots,x_n)) &\to f(\mathsf{chk}(x_1),\ldots,\mathsf{chk}(x_n))\\
\mathsf{chk}(c) &\to \mathsf{active}(c)\\
f(\mathsf{active}(x_1),\ldots,\mathsf{active}(x_n)) &\to \mathsf{active}(f(x_1,\ldots,x_n))\\
f(\mathsf{active}(x_1),..,\mathsf{mark}(x_i),..,\mathsf{active}(x_n)) &\to \mathsf{mark}(f(x_1,\ldots,x_n))
\end{aligned}$$

for all $l \to r$ in R, all $f \in \Sigma$ with arity $n > 0$, and all constants $c \in \Sigma$.

If $t \to_{L'(R,p)} u$ then either $\mathsf{fil}(t) \to_{R_2} \mathsf{fil}(u)$ or both $\mathsf{fil}(t) = \mathsf{fil}(u)$ and $t \to_{R_1} u$. Hence, for termination of $L'(R,p)$ it suffices to prove termination of R_2.

For any term t we define the multiset $M(t)$ over the natural numbers as follows: for every path from the root to a leaf of t we count the total number of active and chk symbols on this path, and put the resulting number in $M(t)$. For example, if $t = \mathsf{chk}(\mathsf{active}(c), \mathsf{chk}(\mathsf{active}(c)))$ then $M(t) = \{2,3\}$. Now:

$M(t) > M(u)$ if the step $t \to_{R_2} u$ was done by applying a rule of the form $\mathsf{active}(l) \to \mathsf{mark}(r)$ (for a duplicating rule $l \to r$ it is possible that $M(u)$ is obtained from $M(t)$ by replacing one element of $M(t)$ by a number of smaller elements)

$M(t) \geq M(u)$ if the step $t \to_{R_2} u$ was done by applying another rule of R_2.

By the well-foundedness of the multiset order we conclude that in an infinite R_2-reduction, rules of the form $\mathsf{active}(l) \to \mathsf{mark}(r)$ are applied only finitely many times. Hence it remains to prove termination of the rest of R_2:

$$\begin{aligned}
\mathsf{chk}(f(x_1,\ldots,x_n)) &\to f(\mathsf{chk}(x_1),\ldots,\mathsf{chk}(x_n))\\
\mathsf{chk}(c) &\to \mathsf{active}(c)\\
f(\mathsf{active}(x_1),\ldots,\mathsf{active}(x_n)) &\to \mathsf{active}(f(x_1,\ldots,x_n))\\
f(\mathsf{active}(x_1),..,\mathsf{mark}(x_i),..,\mathsf{active}(x_n)) &\to \mathsf{mark}(f(x_1,\ldots,x_n))
\end{aligned}$$

We use RPO with precedence $\mathsf{chk} > f > \mathsf{active} > \mathsf{mark}$ for all $f \in \Sigma$. □

Now we can formulate a lemma which relates $L(R,p)$ to $\to_R$.

Lemma A.3 (Relationship between $L(R,p)$ and $\to_R$) *If $t,u \in \mathcal{T}(\Sigma_G \setminus \{\mathsf{tp}\})$ and $\mathsf{chk}(\mathsf{mat}(\overline{p},t)) \to^+_{L(R,p)} \mathsf{mark}(u)$ then $t,u \in \mathcal{T}(\Sigma)$ and $t \to_R u$. Moreover, if $\mathsf{chk}(\mathsf{mat}(\overline{p},t)) \to^+_{L(R,p)} \mathsf{active}(u)$, then $t,u \in \mathcal{T}(\Sigma)$ and $t = u$.*

Proof. First note that since t does not contain the symbol tp, in the reduction one can only use rewrite rules from $L'(R,p)$ (where $L'(R,p)$ is defined as in Lemma A.2). Let $t \succsim u$ hold iff $\mathsf{fil}(t)\ (\to_{R_2} \cup \rhd_\Sigma)^*\ \mathsf{fil}(u)$. Here, fil and R_2 are defined as in the proof of Lemma A.2 and $\rhd_\Sigma$ is the subterm relation for symbols from Σ, i.e., it is the rewrite relation of the TRS with the rules $f(x_1,\ldots,x_i,\ldots,x_n) \to x_i$ for all $f \in \Sigma$. Let $\succ$ be the strict part of $\succsim$, i.e., $\succ = \succsim \setminus \precsim$. The well-foundedness of $\succ$ follows from the well-foundedness of R_2, which was shown in the proof of Lemma A.2.

We use induction on $\succ$. When reducing $\mathsf{chk}(\mathsf{mat}(\overline{p},t))$ to $\mathsf{mark}(u)$ or $\mathsf{active}(u)$, first some reductions may take place inside t. Hence, t is reduced to t' with $t \to^*_{L'(R,p)} t'$. As shown in the proof of Lemma A.2, this implies $\mathsf{fil}(t) \to^*_{R_2} \mathsf{fil}(t')$ and hence, $t \succsim t'$. Then, the redex $\mathsf{mat}(\overline{p},t')$ is reduced.

Hence, $t' = f(t_1, \ldots, t_n)$ and $f \in \Sigma$ (otherwise no mat-rule is applicable).

If $f \neq \mathrm{root}(\overline{p})$ and f is a constant (i.e., $n = 0$), then we have

$$\begin{aligned}\mathsf{chk}(\mathsf{mat}(\overline{p}, t)) &\to^*_{L'(R,p)} \mathsf{chk}(\mathsf{mat}(\overline{p}, t')) \\ &= \quad\ \mathsf{chk}(\mathsf{mat}(\overline{p}, f)) \\ &\to_{L'(R,p)} \mathsf{chk}(\mathsf{no}(f)) \\ &\to_{L'(R,p)} \mathsf{active}(f)\end{aligned}$$

and $\mathsf{active}(f) \to_{L'(R,p)} \mathsf{mark}(r)$ if $f \to r \in R$, which proves the lemma.

Now we regard the case where $f \neq \mathrm{root}(\overline{p})$ and $n > 0$. Here we have

$$\begin{aligned}\mathsf{chk}(\mathsf{mat}(\overline{p}, t)) &\to^*_{L'(R,p)} \mathsf{chk}(\mathsf{mat}(\overline{p}, t')) \\ &= \quad\ \mathsf{chk}(\mathsf{mat}(\overline{p}, f(t_1, \ldots, t_n))) \\ &\to_{L'(R,p)} \mathsf{chk}(\mathsf{no}(f(t_1, \ldots, t_n))) \\ &\to^+_{L'(R,p)} f(\mathsf{chk}(\mathsf{mat}(\overline{p}, t'_1)), \ldots, \mathsf{chk}(\mathsf{mat}(\overline{p}, t'_n)))\end{aligned}$$

with $t_i \to^*_{L'(R,p)} t'_i$. This term can only rewrite to $\mathsf{mark}(u)$ if there is a j such that $\mathsf{chk}(\mathsf{mat}(\overline{p}, t'_j))$ rewrites to some $\mathsf{mark}(u_j)$ and $\mathsf{chk}(\mathsf{mat}(\overline{p}, t'_i))$ rewrites to a term of the form $\mathsf{active}(u_i)$ for all $i \neq j$. Similarly, $f(\mathsf{chk}(\mathsf{mat}(\overline{p}, t'_1)), \ldots, \mathsf{chk}(\mathsf{mat}(\overline{p}, t'_n)))$ can only rewrite to $\mathsf{active}(u)$ if all $\mathsf{chk}(\mathsf{mat}(\overline{p}, t'_i))$ rewrite to terms of the form $\mathsf{active}(u_i)$. Since $t \gtrsim t' \succ t_i \gtrsim t'_i$ for all i, we can apply the induction hypothesis which implies the lemma.

Finally, if $f = \mathrm{root}(\overline{p})$ then we have

$$\begin{aligned}\mathsf{chk}(\mathsf{mat}(\overline{p}, t')) &= \quad\ \mathsf{chk}(\mathsf{mat}(\overline{p}, f(t_1, \ldots, t_n))) \\ &\to_{L'(R,p)} \mathsf{chk}(f(t_1, \ldots, \mathsf{mat}(\overline{p}, t_i), \ldots, t_n)).\end{aligned}$$

Reducing a t_j with $j \neq i$ to a term $\mathsf{no}(u_j)$ and then lifting this no on top of f would lead to a term $\mathsf{chk}(\mathsf{no}(f(\ldots, t'_i, \ldots))) \to_{L'(R,p)} f(\ldots, \mathsf{chk}(\mathsf{mat}(\overline{p}, t'_i)), \ldots)$ with $t'_i \notin \mathcal{T}(\Sigma)$. Note that $t \gtrsim t' \succ t_i \gtrsim t'_i$. By the induction hypothesis, the term $\mathsf{chk}(\mathsf{mat}(\overline{p}, t'_i))$ cannot reduce to a term of the form $\mathsf{mark}(u_i)$ or $\mathsf{active}(u_i)$ and thus, the whole term cannot reduce to $\mathsf{mark}(u)$ or $\mathsf{active}(u)$.

Hence, there must be a term u_i with $\mathsf{mat}(\overline{p}, t_i) \to^+_{L'(R,p)} \mathsf{no}(u_i)$ and the reduction continues as follows:

$$\begin{aligned}&\mathsf{chk}(f(t_1, \ldots, \mathsf{mat}(\overline{p}, t_i), \ldots, t_n)) \to^+_{L'(R,p)} \\ &f(\mathsf{chk}(\mathsf{mat}(\overline{p}, t'_1)), \ldots, \mathsf{chk}(\mathsf{mat}(\overline{p}, u'_i)), \ldots \mathsf{chk}(\mathsf{mat}(\overline{p}, t'_n)))\end{aligned}$$

where $t_j \to^*_{L'(R,p)} t'_j$ and $u_i \to^*_{L'(R,p)} u'_i$. Note that $t \gtrsim t' \succ t_j \gtrsim t'_j$ for $j \neq i$ and $t \gtrsim t' \succ t_i \gtrsim u'_i$. The reason for $t_i \gtrsim u'_i$ is that $\mathsf{mat}(\overline{p}, t_i) \to^*_{L'(R,p)} \mathsf{no}(u_i)$ implies $\mathsf{fil}(t_i) = \mathsf{fil}(\mathsf{mat}(\overline{p}, t_i)) \to^*_{R_2} \mathsf{fil}(\mathsf{no}(u_i)) = \mathsf{fil}(u_i)$. So similar to the case where $f \neq \mathrm{root}(\overline{p})$, now the conjecture follows from the induction hypothesis.□

Now we can formulate a lemma on the relationship between $L(R, p)$ and $\to_G$.

Lemma A.4 (Relationship between $L(R, p)$ and $\to_G$) *Let $p \in \mathcal{T}(\Sigma, \mathcal{V})$*

be linear. For ground terms t over Σ we have $t \to_G^+ u$ if and only if $\mathsf{tp}(\mathsf{mark}(t)) \to_{L(R,p)}^+ \mathsf{tp}(\mathsf{mark}(u))$.

Proof. For "only if", we must show that $\mathsf{tp}(\mathsf{mark}(t)) \to_{L(R,p)}^+ \mathsf{tp}(\mathsf{mark}(u))$ if $t \to_G u$. By the assumption $t \to_G u$ we know $t \notin G$, hence $\mathsf{mat}(\overline{p}, t') \to_{L(R,p)}^+ \mathsf{no}(t')$ for subterms t' of t (Lemma A.1). Now the reduction $\mathsf{tp}(\mathsf{mark}(t)) \to_{L(R,p)}^+ \mathsf{tp}(\mathsf{mark}(u))$ can easily be constructed by propagating the chk-symbols to the leaves until all leaves c are replaced by $\mathsf{active}(c)$. Then these active-symbols can be propagated back to the root. During this propagation the redex from the reduction $t \to_G u$ is reduced and the corresponding active-symbol is replaced by a mark-symbol. Hence, we end up in $\mathsf{tp}(\mathsf{mark}(u))$.

For the converse, since $t \in \mathcal{T}(\Sigma)$, we must have $\mathsf{tp}(\mathsf{mark}(t)) \to_{L(R,p)} \mathsf{tp}(\mathsf{chk}(\mathsf{mat}(\overline{p}, t))) \to_{L(R,p)}^+ \mathsf{tp}(\mathsf{mark}(u))$. Hence, Lemma A.3 implies $t \to_R u$.

We still need to prove that $t \notin G$. Note that $\mathsf{chk}(\mathsf{mat}(\overline{p}, t)) \to_{L(R,p)}^+ \mathsf{mark}(u)$ implies that chk is propagated downwards to the leaves of t. By Lemma A.1 this implies that no subterm of t is matched by p. Thus, the $\mathsf{active}(\ldots) \to \mathsf{mark}(\ldots)$ step done in the reduction $\mathsf{chk}(\mathsf{mat}(\overline{p}, t)) \to_{L(R,p)}^+ \mathsf{mark}(u)$ can also be done with $\to_G$. Note that in the reduction $\mathsf{chk}(\mathsf{mat}(\overline{p}, t)) \to_{L(R,p)}^+ \mathsf{mark}(u)$ we perform exactly one such step (if one would perform another $\mathsf{active}(\ldots) \to \mathsf{mark}(\ldots)$ step, then mark could no longer be propagated to the top since the rules

$$f(\mathsf{active}(x_1), \ldots, \mathsf{mark}(x_i), \ldots, \mathsf{active}(x_n)) \to \mathsf{mark}(f(x_1, \ldots, x_n))$$

can only be applied if there is exactly one occurrence of mark). □

Now we prove Thm. 5.1: $\to_G$ is terminating iff the TRS $L(R, p)$ is terminating.

Proof. We first show the "if"-direction. If $\to_G$ were not terminating, then there would be an infinite reduction

$$t_0 \to_G t_1 \to_G \ldots$$

of ground terms over Σ. (We extended the signature such that ground termination already implies termination of $\to_G$.) By Lemma A.4 this would imply

$$\mathsf{tp}(\mathsf{mark}(t_0)) \to_{L(R,p)}^+ \mathsf{tp}(\mathsf{mark}(t_1)) \to_{L(R,p)}^+ \ldots$$

in contradiction to the termination of $L(R, p)$.

For "only if", assume that $L(R, p)$ is not terminating. Then by type introduction [12,15] one can show that there is an infinite $L(R, p)$-reduction where every term has tp as root, whereas tp does not occur below the root. Due to Lemma A.2 the reduction contains infinitely many applications of the rule $\mathsf{tp}(\mathsf{mark}(x)) \to \mathsf{tp}(\mathsf{chk}(\ldots))$. Hence, the reduction has the form

$$\begin{aligned} &\mathsf{tp}(\mathsf{mark}(t_0)) \to_{L(R,p)} \mathsf{tp}(\mathsf{chk}(\mathsf{mat}(\overline{p}, t_0))) \to_{L(R,p)}^+ \\ &\mathsf{tp}(\mathsf{mark}(t_1)) \to_{L(R,p)} \mathsf{tp}(\mathsf{chk}(\mathsf{mat}(\overline{p}, t_1))) \to_{L(R,p)}^+ \ldots \end{aligned}$$

By Lemma A.3 this implies $t_0, t_1, \ldots \in \mathcal{T}(\Sigma)$. Thus, Lemma A.4 implies

$$t_0 \to_G^+ t_1 \to_G^+ \ldots$$

in contradiction to the termination of $\rightarrow_G$. □

References

[1] Alpern, B. and F. B. Schneider, *Defining liveness*, Information Processing Letters **21** (1985), pp. 181–185.

[2] Arts, T. and J. Giesl, *Termination of term rewriting using dependency pairs*, Theoretical Computer Science **236** (2000), pp. 133–178.

[3] Baader, F. and T. Nipkow, "Term Rewriting and All That," Cambridge University Press, 1998.

[4] Bouajjani, A., *Languages, rewriting systems, and verification of infinite-state systems*, in: *Proc. ICALP '01*, LNCS 2076, 2001, pp. 24–39.

[5] Caucal, D., *On the regular structure of prefix rewriting*, Theoretical Computer Science **106** (1992), pp. 61–86.

[6] Dershowitz, N., *Termination of rewriting*, Journal of Symbolic Computation **3** (1987), pp. 69–116.

[7] Fribourg, L. and H. Olsén, *Reachability sets of parametrized rings as regular languages*, in: *Proc. INFINITY '97*, ENTCS 9, 1997.

[8] Giesl, J. and T. Arts, *Verification of Erlang processes by dependency pairs*, Applicable Algebra in Engineering, Communication and Computing **12** (2001), pp. 39–72.

[9] Giesl, J. and A. Middeldorp, *Transforming context-sensitive rewrite systems*, in: *Proc. 10th RTA*, LNCS 1631, 1999, pp. 271–285.

[10] Giesl, J. and A. Middeldorp, *Transformation techniques for context-sensitive rewrite systems*, Journal of Functional Programming (2003), to appear. Preliminary extended version appeared as Technical Report [6] AIB-2002-02, RWTH Aachen, Germany.

[11] Giesl, J. and H. Zantema, *Liveness in rewriting*, in: *Proc. 14th RTA*, LNCS, 2003, to appear. Extended version appeared as Technical Report [6] AIB-2002-11, RWTH Aachen, Germany.

[12] Middeldorp, A. and H. Ohsaki, *Type introduction for equational rewriting*, Acta Informatica **36** (2000), pp. 1007–1029.

[13] Moller, F., *Infinite results*, in: *Proc. CONCUR '96*, LNCS 1119, 1996, pp. 195–216.

[14] van Gasteren, A. and G. Tel, *Comments on "On the proof of a distributed algorithm": always-true is not invariant*, Information Processing Letters **35** (1990), pp. 277–279.

[15] Zantema, H., *Termination of term rewriting: Interpretation and type elimination*, Journal of Symbolic Computation **17** (1994), pp. 23–50.

[6] Available from `http://aib.informatik.rwth-aachen.de`.

A Rewriting Strategy for Protocol Verification (Extended Abstract) [1]

Monica Nesi [2], Giuseppina Rucci and Massimo Verdesca

Dipartimento di Informatica
Università degli Studi di L'Aquila
via Vetoio, 67010 L'Aquila, Italy

Abstract

This paper presents a rewriting strategy for the analysis and verification of communication protocols. In a way similar to the approximation technique defined by Genet and Klay, a rewrite system $\mathcal{R}$ specifies the protocol and a tree automaton $\mathcal{A}$ describes the initial set of communication requests. Given a term t that represents a property to be proved, a rewriting strategy is defined that suitably expands and reduces t using the rules in $\mathcal{R}$ and the transitions in $\mathcal{A}$ to derive whether or not t is recognized by the intruder. This is done by simulating a completion process in a bottom-up manner starting from t and trying to derive a transition $t \rightarrow q_f$ from critical pairs, where q_f is a final state of the tree automaton. The rewriting strategy is defined by means of a set of inference rules and used for reasoning about the authentication and secrecy properties of the Needham-Schroeder public-key protocol.

1 Introduction

The analysis and verification of communication and security protocols have recently become an important subject of research, mostly due to the development of Internet and the related electronic commerce services. Several approaches have been applied to protocol specifications in order to formally prove various properties of interest, such as authentication, secrecy or confidentiality, freshness of nonces, etc. These approaches range from model checking [15,17] to theorem proving [16,23,24,26] through rewriting techniques and strategies [2,4,12] combined with tree automata and abstract interpretation [7,9,18]. There is also ongoing work on combining different approaches, such as the combination of Genet and Klay's approximation technique with Paulson's inductive method in Isabelle [20,21].

[1] Work partially supported by MURST 40% "Architetture Software per Infrastrutture di Rete ad Accesso Eterogeneo (SAHARA)".

[2] Email: monica@di.univaq.it

We are interested in the approach based on the approximation technique developed in [6,7,9] that uses rewrite systems and tree automata to specify and verify properties of security protocols. This technique aims at finding that there are no attacks on a protocol, rather than at discovering attacks. The protocol is operationally specified through a rewrite system $\mathcal{R}$, while the initial set E of communication requests is described through a tree automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) \supseteq E$. Starting from $\mathcal{R}$ and $\mathcal{A}$, the approximation technique by Genet and Klay builds a tree automaton which over-approximates the set of the messages exchanged among the protocol agents. In other words, the language recognized by the resulting approximation automaton $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ includes all $\mathcal{R}$-descendants of E. In this way, in order to prove whether a property p is satisfied, it is sufficient to consider the intersection between the language of $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ and the language of a tree automaton $\mathcal{A}_{\overline{p}}$ which models the negation of p and thus contains the "prohibited" terms. If such intersection is empty, then p is satisfied.

Like in the approach based on the approximation technique, we consider a rewrite system $\mathcal{R}$ to specify the protocol and a tree automaton $\mathcal{A}$ to represent the initial set E of communication requests. Given a term t that describes a property to be proved, a rewriting strategy is defined that suitably expands and reduces t using the rules in $\mathcal{R}$ and the transitions in $\mathcal{A}$ to derive whether or not t is recognized by the intruder. This is done by simulating a completion process in a bottom-up manner starting from t and trying to derive if a transition $t \rightarrow q_f$ can be generated from a critical pair, where q_f is a final state of $\mathcal{A}$. If the transition $t \rightarrow q_f$ is derived by the strategy, this means that the term t is recognized by the intruder and thus the property represented by t is not satisfied by the protocol. If $t \rightarrow q_f$ is not derived, then the property given by t is true.

This strategy is inspired by a previously defined rewriting strategy to deal with the problem of divergence of the completion process [10,11]. The automaton $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ resulting from the approximation technique is characterized by a finite set of transitions, but their number can be very high. We think that also in this case it can be worth to apply the bottom-up strategy, specially when the language intersection is not empty and there may be an attack.

The protocol specification under consideration is the Needham-Schroeder Public-Key protocol [19] (NSPK from now on), which is the typical first case study of protocol verification. This simple protocol, developed in 1978, was proved insecure only many years later by Lowe [14] and Meadows [16].

The paper is organized as follows. Section 2 briefly describes the NSPK and its formalization as a rewrite system. In Section 3 the approximation technique is briefly recalled and discussed. The rewriting strategy is then introduced and defined by means of inference rules in Section 4 and shown to be correct, terminating and complete. The application of the strategy to the specification of the NSPK for proving the properties of secrecy and authentication is finally outlined. The paper ends with some concluding remarks and directions for future work.

2 The NSPK Protocol

The NSPK [19] aims at the mutual authentication of two agents communicating through an insecure network. Every agent A has a public key K_A, that is known to any agent and can be obtained from a key server, and a secret key K_A^{-1}, the inverse of K_A, that is supposed to be known only to A. A message m crypted with a key K is denoted $\{m\}_K$. Thus, any agent can encrypt a message m with the public key of any other agent A by producing $\{m\}_{K_A}$, but only A can decrypt such a message using its private key K_A^{-1} (if the secrecy of the private key is guaranteed). The protocol is based on the exchange of messages made of *nonces* $N_A, N_B, \ldots$ (random numbers freshly generated) where the index denotes the agent that has created the nonce. Given two agents A and B, the NSPK protocol (without key server) can be described as follows:

$$
\begin{aligned}
&1.\ A \longrightarrow B\ :\ \{N_A, A\}_{K_B} \\
&2.\ B \longrightarrow A\ :\ \{N_A, N_B\}_{K_A} \\
&3.\ A \longrightarrow B\ :\ \{N_B\}_{K_B}
\end{aligned}
$$

The agent A (initiator of the communication) sends a message (crypted with the public key of B) to the agent B. The message of A contains a nonce N_A and its identity. The responder B can decrypt the message and sends back to A a message encrypted with the public key of A, containing the nonce N_A that B has received and a nonce N_B. In the last step of the protocol, A returns the nonce N_B to B.

This version of the NSPK has been proved insecure by Lowe [14] and Meadows [16]. In particular, the property of authentication fails. As shown by Lowe in [14], the attack involves two parallel executions of the protocol, where in the first session the agent A establishes a communication with the intruder I, and in the second session I impersonates A and establishes a valid communication with the agent B. At the end of the executions of the protocol, B thinks he has established a communication with A, while he is communicating with I that has impersonated A. The corrected version of the NSPK proposed by Lowe [15] introduces the identity of the responder B in the message exchanged in the second step of the protocol, and the initiator A checks this identity against that of the agent with whom A has started the communication. The corrected version of the NSPK is the following:

$$
\begin{aligned}
&1.\ A \longrightarrow B\ :\ \{N_A, A\}_{K_B} \\
&2.\ B \longrightarrow A\ :\ \{N_A, N_B, B\}_{K_A} \\
&3.\ A \longrightarrow B\ :\ \{N_B\}_{K_B}
\end{aligned}
$$

In this way an intruder I cannot send along a message from B to A as if it was his own.

The NSPK can be operationally described by a rewrite system $\mathcal{R}$ [7]. [3] Let $\mathcal{F}$ denote the signature of $\mathcal{R}$. Let L_{agt} be an infinite set of agent labels. We are interested in the behaviour of two agents A and B, and all other agents are coded through natural numbers built using the constructors 0 and s. Thus, $L_{agt} = \{A, B\} \cup \mathbb{N}$. $agt(l)$ denotes an agent whose label is $l \in L_{agt}$. $mesg(x, y, c)$ denotes a message from x to y with contents c. $pubkey(a)$ represents the public key of an agent a and $encr(k, a, c)$ denotes the result of encrypting c with the key k (a is a flag that stores the agent that did the encryption). A term $N(x, y)$ denotes a nonce generated by agent x for communicating with agent y. $goal(x, y)$ denotes that x tries to establish a communication with y. $c\_init(x, y, z)$ represents the fact that x thinks of communicating with y but in fact x has established a communication with z. $c\_resp(x, y, z)$ denotes that x thinks he has replied to a communication requested by y but in fact he has replied to a communication requested by z. A list made of x and y is represented by $cons(x, y)$.

Every protocol step is formalized by means of a rewrite rule whose left hand side expresses a condition on the current state of the protocol (received messages and communication requests), and the right hand side is the message to be sent if the condition is satisfied. The rewrite system $\mathcal{R}$ we have been using for experimenting our strategy on the insecure NSPK is given below. We have $\mathcal{R} = \mathcal{R}_P \cup \mathcal{R}_I$, where the rules in $\mathcal{R}_P = (1) \div (5)$ describe the steps of the protocol, and the rules in $\mathcal{R}_I = (6) \div (10)$ define the intruder's ability of decomposing and decrypting messages.

$$goal(agt(a), agt(b)) \tag{1}$$
$$\rightarrow mesg(agt(a), agt(b), encr(pubkey(agt(b)), agt(a), cons(N(agt(a), agt(b)), agt(a))))$$

$$mesg(agt(a_4), agt(b), encr(pubkey(agt(b)), agt(a_3), cons(a_1, agt(a)))) \tag{2}$$
$$\rightarrow mesg(agt(b), agt(a), encr(pubkey(agt(a)), agt(b), cons(a_1, N(agt(b), agt(a)))))$$

$$mesg(agt(a_6), agt(a), encr(pubkey(agt(a)), agt(a_5), cons(N(agt(a), agt(b)), a_2))) \tag{3}$$
$$\rightarrow mesg(agt(a), agt(b), encr(pubkey(agt(b)), agt(a), a_2))$$

$$mesg(agt(a_6), agt(a), encr(pubkey(agt(a)), agt(a_5), cons(N(agt(a), agt(b)), a_2))) \tag{4}$$
$$\rightarrow c\_init(agt(a), agt(b), agt(a_5))$$

$$mesg(agt(a_8), agt(b), encr(pubkey(agt(b)), agt(a_7), N(agt(b), agt(a)))) \tag{5}$$
$$\rightarrow c\_resp(agt(b), agt(a), agt(a_7))$$

$$\cup(cons(x, y), z) \rightarrow x \tag{6}$$

$$\cup(cons(x, y), z) \rightarrow y \tag{7}$$

$$encr(pubkey(agt(0)), y, z) \rightarrow z \tag{8}$$

$$encr(pubkey(agt(s(x))), y, z) \rightarrow z \tag{9}$$

$$mesg(x, y, z) \rightarrow z \tag{10}$$

[3] We refer to [1,5] for more details about rewrite systems and to [3] for the theory of tree automata.

Note that $\mathcal{R}$ is slightly different from the system given in [7,20,21]. In particular, the LHS operator, that keeps track of the steps applied to yield a certain term, and the *add* operator are omitted, and the $\cup$ operator is removed from some rewrite rules, except for rules (6) and (7).

3 The Approximation Technique

Many approaches to the verification of security protocols have proved the authentication property (besides other properties) for the corrected version of the NSPK. The approach based on the approximation technique [6,7,9] is one of them. This technique allows one to automatically compute an over-approximation of the set $\mathcal{R}^*(E)$ of the $\mathcal{R}$-descendants of a set of terms E, where E is described through a tree automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) \supseteq E$. The approximation automaton $\mathcal{T}_{\mathcal{R}}\!\uparrow\!(\mathcal{A})$ is such that $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow\!(\mathcal{A})) \supseteq \mathcal{R}^*(E)$. The quality of the approximation depends on an approximation function γ which defines the so-called folding positions, i.e. subterms that can be approximated.

We recall the idea of the approximation technique very briefly. Starting from $\mathcal{A}_0 = \mathcal{A}$, the technique builds a finite number of tree automata $\mathcal{A}_i$ with transitions Δ_i such that for all $i \geq 0$ we have $\mathcal{L}(\mathcal{A}_i) \subset \mathcal{L}(\mathcal{A}_{i+1})$ until an automaton $\mathcal{A}_k$ is obtained such that $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$, i.e. $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(E)$. The approximation technique can be seen as a particular completion process between the rules in $\mathcal{R}$ and the transitions Δ_i of $\mathcal{A}_i$. In fact, in order to build $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$, a critical pair is computed between a rewrite rule in $\mathcal{R}$ and the transitions in Δ_i. The rule derived from the critical pair is a new transition that is normalized using the approximation function γ and then added to Δ_i, thus yielding Δ_{i+1}.

The approximation technique, initially given for left linear rewrite systems [6], has then been extended to AC non-left linear rewrite systems in [7]. Given $\mathcal{T}_{\mathcal{R}}\!\uparrow\!(\mathcal{A})$, reachability properties on $\mathcal{R}$ and E can be proved by checking whether the intersection between $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow\!(\mathcal{A}))$ and a set of terms, that must not be reached from E through $\mathcal{R}$, is empty. In the case of protocol verification, the negation of the property p to be proved is described through a tree automaton $\mathcal{A}_{\overline{p}}$, whose language contains those terms that must not be recognized by the intruder. If the intersection $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow\!(\mathcal{A})) \cap \mathcal{L}(\mathcal{A}_{\overline{p}})$ is empty, then the property p is satisfied. On the contrary, whenever such intersection is not empty, we cannot say that there is an attack. Actually, this approach has been designed for proving that there are no attacks on a protocol, rather than for detecting them. The fact is that the quality of the approximation automaton depends on the approximation function γ and, in spite of further studies on γ and its automation [22], whenever the intersection is not empty, it is not possible to distinguish whether there is an attack on the protocol or the function γ is too "large" and $\mathcal{T}_{\mathcal{R}}\!\uparrow\!(\mathcal{A})$ defines a language that also includes terms of $\mathcal{L}(\mathcal{A}_{\overline{p}})$. Hence, the approximation technique is not able to discover the attack on the insecure version of the NSPK, while it demonstrates the properties of authentication and secrecy on the corrected version [7,9].

4 The Rewriting Strategy

The implementation of the approximation technique we have been using for our experimentation is the one in Timbuk [8,9], a library of Objective Caml [13] for manipulating tree automata. Oehl has kindly provided us with his Timbuk files for both the insecure and the corrected versions of the NSPK, together with his approximation functions γ. We were primarily interested in running the approximation technique on the insecure version of the NSPK and understanding why the language intersection is not empty for both secrecy (nonces must be kept secret) and authentication of agents.

The rewrite system $\mathcal{R}$ specifying the insecure NSPK has been given in Section 2. The tree automaton $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ describing the initial set E of communication requests is taken from [7], where $\mathcal{Q}_f = \{q_f\}$ and the transitions in Δ are as follows:

$0 \to q_{int}$		
$s(q_{int}) \to q_{int}$	$agt(q_{int}) \to q_{agtI}$	
$A \to q_A$	$agt(q_A) \to q_{agtA}$	
$B \to q_B$	$agt(q_B) \to q_{agtB}$	
$goal(q_{agtA}, q_{agtB}) \to q_f$	$goal(q_{agtA}, q_{agtA}) \to q_f$	*communication requests*
$goal(q_{agtB}, q_{agtA}) \to q_f$	$goal(q_{agtB}, q_{agtB}) \to q_f$	
$goal(q_{agtA}, q_{agtI}) \to q_f$	$goal(q_{agtI}, q_{agtA}) \to q_f$	
$goal(q_{agtB}, q_{agtI}) \to q_f$	$goal(q_{agtI}, q_{agtB}) \to q_f$	
$goal(q_{agtI}, q_{agtI}) \to q_f$	$\cup(q_f, q_f) \to q_f$	
$agt(q_{int}) \to q_f$	$pubkey(q_{agtI}) \to q_f$	*intruder's initial knowledge*
$agt(q_A) \to q_f$	$pubkey(q_{agtA}) \to q_f$	
$agt(q_B) \to q_f$	$pubkey(q_{agtB}) \to q_f$	
$mesg(q_f, q_f, q_f) \to q_f$	$N(q_{agtI}, q_{agtI}) \to q_f$	
$cons(q_f, q_f) \to q_f$	$N(q_{agtI}, q_{agtA}) \to q_f$	
$encr(q_f, q_{agtI}, q_f) \to q_f$	$N(q_{agtI}, q_{agtB}) \to q_f$	

Let t be a term that describes a property to be proved or disproved. We consider the problem of checking whether t can be recognized by the approximation automaton $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$. This means checking whether a transition $t \to q_f$ can be generated from critical pairs, where q_f is a final state of $\mathcal{A}$. We do not build $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$, but starting from t we simulate a completion process in a bottom-up manner guided by the critical pairs, by reconstructing the rewriting path that has led to the intruder's knowledge of t, if any. If $t \to q_f$ can be generated during this process, the property represented by t is not satisfied. Moreover, by going up along the critical pairs we get to know which terms have been previously (in the completion process) recognized by the intruder, thus getting some feedback on the error location. This strategy is similar to a rewriting strategy defined in [10,11] to deal with the problem of divergence of the completion process, where the bottom-up strategy allows one to compute the normal form of a term with respect to the infinite canonical rewrite

system.

The critical pairs between the rules in $\mathcal{R}$ and the transitions in Δ computed by the approximation technique are generated by the strategy in a bottom-up manner, by applying an expansion process on terms with respect to $\mathcal{R}$ and then checking the (instances of the) resulting terms for recognizability by the intruder using only its initial knowledge Δ. Moreover, a notion of well-formedness of terms is used for ensuring the termination of the expansion process.

A term t is expanded with $\mathcal{R}$ if t unifies with the right hand side of a rule of $\mathcal{R}$: $\mathit{expansion}(t, \mathcal{R}) = \{s = \sigma(t[l]_p) \mid \exists\, l \to r \in \mathcal{R},\ p \in Pos'(t)$ and $\sigma = \mathrm{mgu}(t|_p, r)\}$. Thus, an expansion step is a narrowing step with a reversed rule of $\mathcal{R}$. The expansion process may introduce occurrences of "new" variables in s. These variables are considered as implicitly universally quantified and will be then instantiated by means of a finite set of ground terms $\mathit{Inst} = \{c_1, \ldots, c_k\}$, thus getting the *instance set* $\mathcal{I}(t, \mathit{Inst}) = \{\sigma(t) \mid \sigma : \mathit{Var}(t) \to \mathit{Inst}\}$. Typically, *Inst* can be chosen as a finite set of agent labels needed for instantiating the agent variables in $\mathcal{R}$. For the NSPK it will be sufficient to take $\mathit{Inst} = \{A, B, 0\}$.

A term t is recognizable by the intruder if q_f can be derived from t using the transitions in Δ. As the strategy simulates a completion process that would produce the intruder's incremental knowledge, whenever t is not directly recognizable using Δ, the strategy is applied to those subterms of t that are not recognizable in Δ. Based on the following proof system $\vdash_{\mathcal{A}}$, we define a function $\mathit{recognizable}(t) = \{q_f\}$ if $t \vdash_{\mathcal{A}} q_f$, otherwise $\mathit{recognizable}(t) = \{t_i \mid t = \mathcal{C}[t_i]$ for some context $\mathcal{C}$ and $t_i \not\vdash_{\mathcal{A}} q_f\}$, thus yielding those subterms of t labelling the leaves of the proof tree of t in $\vdash_{\mathcal{A}}$ that remain unsolved.

$$\frac{t \xrightarrow{*}_{\Delta} q \quad q \in \{q_f, q_{agtI}\}}{t \vdash_{\mathcal{A}} q} \qquad \frac{t_1 \vdash_{\mathcal{A}} q_f \quad t_2 \vdash_{\mathcal{A}} q_f}{\cup(t_1,t_2) \vdash_{\mathcal{A}} q_f} \qquad \frac{t_1 \vdash_{\mathcal{A}} q_f \quad t_2 \vdash_{\mathcal{A}} q_f \quad t_3 \vdash_{\mathcal{A}} q_f}{\mathit{mesg}(t_1,t_2,t_3) \vdash_{\mathcal{A}} q_f}$$

$$\frac{t_1 \vdash_{\mathcal{A}} q_f \quad t_2 \vdash_{\mathcal{A}} q_f}{\mathit{cons}(t_1,t_2) \vdash_{\mathcal{A}} q_f} \qquad \frac{t_1 \vdash_{\mathcal{A}} q_{agtI} \quad t_2 \vdash_{\mathcal{A}} q_f}{N(t_1,t_2) \vdash_{\mathcal{A}} q_f} \qquad \frac{t_1 \vdash_{\mathcal{A}} q_f \quad t_2 \vdash_{\mathcal{A}} q_{agtI} \quad t_3 \vdash_{\mathcal{A}} q_f}{\mathit{encr}(t_1,t_2,t_3) \vdash_{\mathcal{A}} q_f}$$

The notion of well-formedness of terms is based on the following intuition. A term t is well-formed if it "agrees" with the syntactic structure of $\mathcal{R}$. For example, $t_1 = N(agt(a_1), agt(a_2))$ is well-formed for any variables or agent labels a_1, a_2, while the term $t_2 = N(agt(a_1), \mathit{pubkey}(agt(a_2)))$ is not, as there is no term t in $\mathcal{R}$ such that $\mathit{root}(t) = N$ and the second argument of t starts with *pubkey*. One could also talk of "well-typed" terms, in the sense that when considering the function associated to, for example, the symbol N, this function is expected to take as input two terms of type agent, thus the type-checking of t_2 will fail.

The terms describing the authentication and secrecy properties are well-formed in the sense above. Moreover, during the expansion phase of the strategy, only well-formed terms will be considered and the non-well-formed ones will be cut out of the search space. This is not enough for ensuring the

termination of the expansion process. In fact, rule (3) of $\mathcal{R}$ can be applied as a reversed rule infinitely many times to produce well-typed terms that contain subterms of the form $t = cons(x_1, cons(y_1, cons(z_1, \ldots)))$. Due to the structure of the contents of messages in the NSPK and of the terms in the language describing the properties to be checked, it is not needed to derive such terms by expansion as they cannot be the vertex of any interesting critical peak. Hence, expansion by rule (3) is stopped by defining that terms containing instances of t above are not well-formed. Whenever other protocols and/or different properties are considered, the definition of the well-formedness of terms might have to be changed accordingly.

A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is *well-formed*, written $wf(t)$, if (i) $t \in \mathcal{X} \cup \mathcal{F}^0$ or (ii) $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{F}^n$ $(n > 0)$ and either $t_i \in \mathcal{X}$ or t_i satisfies the following conditions based on the value of f $(i = 1, \ldots, n)$:

- $f = agt$ and $t_1 \in L_{agt}$;
- $f = goal$, $root(t_i) = agt$ and $wf(t_i)$ for $i = 1, 2$;
- $f = mesg$, $root(t_i) = agt$ and $wf(t_i)$ for $i = 1, 2$, $root(t_3) = encr$ and $wf(t_3)$;
- $f = encr$, $root(t_1) = pubkey$, $root(t_2) = agt$, $root(t_3) \in \{cons, N, agt\}$ and $wf(t_i)$ for $i = 1, 2, 3$;
- $f = pubkey$, $root(t_1) = agt$ and $wf(t_1)$;
- $f = cons$, $root(t_i) \in \{N, agt\}$ and $wf(t_i)$ for $i = 1, 2$;
- $f = N$, $root(t_i) = agt$ and $wf(t_i)$ for $i = 1, 2$;
- $f \in \{c\_init, c\_resp\}$, $root(t_i) = agt$ and $wf(t_i)$ for $i = 1, 2, 3$;
- $f = \cup$ and $wf(t_i)$ for $i = 1, 2$.

The input to the strategy is given by the rewrite system $\mathcal{R} = \mathcal{R}_P \cup \mathcal{R}_I$, the predicate *wf* and the instantiation set *Inst* (as defined above), the intruder's initial knowledge Δ in $\mathcal{A}$, and the well-formed term t_{in} describing the property under consideration. The strategy is defined through a set of inference rules (Figure 1) over configurations, which simply consist of the set E of well-formed terms resulting from the expansion process and that can be possibly expanded further. The initial configuration is $E = \{t_{in}\}$. The predicate $subterm(t, t')$ is true if t' is a subterm of t.

The rewriting strategy for protocol verification is then defined as

$$((\textit{Well-formed Expansion.Cut})^*.(\textit{Failure} + \textit{Success1} + \textit{Success2} + \textit{Split}))^*$$

where r^* means that the inference rule r is repeated as long as its applicability conditions are satisfied, $r.r'$ means sequencing of r and r', and $r + r'$ means nondeterministic choice between r and r'.

Given the rewrite system $\mathcal{R}$, the predicate *wf*, the instantiation set *Inst*, the tree automaton $\mathcal{A}$ with transitions Δ and the negation automaton $\mathcal{A}_{\overline{p}}$ for the property to be checked, the correctness, termination and completeness of the strategy are formalized as follows.

Proposition 4.1 *(correctness) Let* $t_{in} \in \mathcal{L}(\mathcal{A}_{\overline{p}})$.

Well-formed Expansion:

$$\frac{t \in E \quad expansion(t, \mathcal{R}) = E'}{E \vdash E \setminus \{t\} \cup \{t' \in E' \mid wf(t')\}}$$

Failure:

$$\frac{E = \emptyset}{failure}$$

Cut:

$$\frac{t \in E \quad expansion(t, \mathcal{R}_P) = \emptyset \quad root(t) = mesg \quad subterm(t, t_{in})}{E \vdash E \setminus \{t\}}$$

Success1:

$$\frac{t \in E \quad expansion(t, \mathcal{R}_P) = \emptyset \quad root(t) = goal}{success}$$

Success2:

$$\frac{\begin{array}{c} t \in E \quad expansion(t, \mathcal{R}_P) = \emptyset \quad root(t) = mesg \quad not(subterm(t, t_{in})) \\ instance(t, Inst) = E_1 \quad \exists t_1 \in E_1.\, recognizable(t_1) = \{q_f\} \end{array}}{success}$$

Split:

$$\frac{\begin{array}{c} t \in E \quad expansion(t, \mathcal{R}_P) = \emptyset \quad root(t) = mesg \quad not(subterm(t, t_{in})) \\ instance(t, Inst) = E_1 \quad \exists t_1 \in E_1.\, recognizable(t_1) = \{t_1, \ldots, t_k\} \end{array}}{E \vdash E \setminus \{t\} \cup \{t_1, \ldots, t_k\}}$$

Fig. 1. The inference rules of the strategy.

(i) If $\{t_{in}\} \vdash$ success*, then there exists a derivation path that generates the transition* $t_{in} \to q_f$.
(ii) If $\{t_{in}\} \vdash$ failure*, then the transition* $t_{in} \to q_f$ *cannot be generated from critical pairs.*

Proof. (Sketch)
(i) Let $\{t_{in}\} \vdash$ *success* be derived by means of a derivation $\{t_{in}\} \vdash E_1 \vdash \ldots \vdash E_n \vdash$ *success*. If *success* is obtained by applying the inference rule Success1, this means that t_{in} has been expanded using $\mathcal{R}_I$ and $\mathcal{R}_P$ until a term $t' \in E_n$ has been obtained such that $root(t') = goal$. As all communication requests $goal(x, y)$ are in the intruder's initial knowledge, the critical peak $t_{in} \overset{*}{\leftarrow}_{\mathcal{R}} t' \to_{\Delta} q_f$ produces the transition $t_{in} \to q_f$.
If *success* is obtained by applying the inference rule Success2, then the situation is as follows. The term t_{in} has been expanded using $\mathcal{R}_I$ and $\mathcal{R}_P$ until a term $t' \in E_k$ for some $k \leq n$ is derived such that either Success2 or Split can be applied. The application of Success2 means that t' is a vertex of a critical peak as t' can be reduced to t_{in} in $\mathcal{R}$ and is recognizable by the intruder, i.e. $t' \overset{*}{\to}_{\Delta} q_f$. Hence, the transition $t_{in} \to q_f$ is generated. The application of the rule Split means that there exist subterms in t' that cannot be recognized based on the intruder's initial knowledge, thus further critical pairs must be searched in a bottom-up manner by iterating the application of the strategy.
(ii) Let $\{t_{in}\} \vdash$ *failure* be derived by means of a derivation $\{t_{in}\} \vdash E_1 \vdash \ldots \vdash E_n \vdash$ *failure*, where $E_n = \emptyset$ (rule Failure). Terms have been expanded until

terms have been derived that are not well-formed (no terms are added to the expansion set E and the expanded terms are removed from E by the rule Well-formed Expansion) or contain the input term t_{in} (they can be deleted using the rule Cut without losing information about the intruder's knowledge). Any such term cannot be the vertex of a critical peak, thus the transition $t_{in} \rightarrow q_f$ cannot be generated from critical pairs. □

Proposition 4.2 *(termination) The rewriting strategy terminates on any input term* $t_{in} \in \mathcal{L}(\mathcal{A}_{\overline{p}})$.

Proof. (Sketch) The set E only contains well-formed terms. Given any well-formed $t_{in} \in \mathcal{L}(\mathcal{A}_{\overline{p}})$, the well-formed expansion process terminates because the repeated application of the rules of $\mathcal{R}$ as expansion rules will eventually produce only terms that do not satisfy the well-formed predicate. Thus, the expansion process along the several different derivation paths terminates because the terms on each path are all well-formed and each path ends with a term that cannot be further expanded in a well-formed way. If the strategy does not derive *success*, then the set E decreases till it becomes empty (rule Failure), because the rule Cut is repeatedly applied along the different paths or the expanded terms are removed from E and not replaced by other well-formed terms (rule Well-formed Expansion). □

Corollary 4.3 *(completeness) Let* $t_{in} \in \mathcal{L}(\mathcal{A}_{\overline{p}})$.
(i) If the transition $t_{in} \rightarrow q_f$ *can be generated from critical pairs, then* $\{t_{in}\} \vdash$ success.
(ii) If the transition $t_{in} \rightarrow q_f$ *cannot be generated from critical pairs, then* $\{t_{in}\} \vdash$ failure.

Proof. It follows from the termination (Proposition 4.2) and the correcteness (Proposition 4.1) of the strategy. □

Example 4.4 Let us consider the property of secrecy for the NSPK. Given the system $\mathcal{R}$ (Section 2), let $t = N(agt(B), agt(A)) \in \mathcal{L}(\mathcal{A}_{\overline{s}})$, where $\mathcal{L}(\mathcal{A}_{\overline{s}})$ denotes the negation automaton for the property of secrecy of nonces. [4] We want to prove whether the nonce t is recognizable by the intruder, thus attacking the secrecy of the NSPK. By abstracting from some details, we get the following derivation steps:

$\{N(agt(B), agt(A))\}$
$\vdash \{\cup(cons(N(agt(B), agt(A)), y_1), z_1), \cup(cons(x_1, N(agt(B), agt(A))), z_1), encr(pubkey(agt(0)), y_1,$
$N(agt(B), agt(A))), encr(pubkey(agt(s(x))), y_1, N(agt(B), agt(A))), mesg(x_1, y_1, N(agt(B), agt(A)))\}.$

By expanding the third term in the current configuration, we obtain:

$encr(pubkey(agt(0)), y_1, N(agt(B), agt(A)))$
$\vdash mesg(x_2, y_2, encr(pubkey(agt(0)), y_1, N(agt(B), agt(A))))$
$\vdash mesg(agt(a_6), agt(a), encr(pubkey(agt(a)), agt(a_5), cons(N(agt(a), agt(0)), N(agt(B), agt(A)))))$
$\vdash mesg(agt(a_4), agt(B), encr(pubkey(agt(B)), agt(a_3), cons(N(agt(A), agt(0)), agt(A)))).$

[4] Due to the definition of tree automaton, t is actually the term $N(q_{agtB}, q_{agtA})$ that gets expanded into $N(agt(B), agt(A))$. In the example we will abstract from these details.

This term cannot be further expanded and does not have the initial term as subterm. Thus, its variables are instantiated through *Inst*. Among the various (finite) possible instances, let us consider the substitution $\sigma = \{A/a_4, 0/a_3\}$, thus yielding the term

$$mesg(agt(A), agt(B), encr(pubkey(agt(B)), agt(0), cons(N(agt(A), agt(0)), agt(A)))).$$

Using the function *recognizable* on t, we have that $N(agt(A), agt(0))$ is a non-recognizable subterm of t, i.e. $N(agt(A), agt(0))$ does not belong to the intruder's basic knowledge. Thus, the rule Split adds $N(agt(A), agt(0))$ to the expansion set E and the following derivation is obtained:

$$N(agt(A), agt(0)) \vdash encr(pubkey(agt(0)), y'_1, N(agt(A), agt(0))))$$
$$\vdash mesg(x'_2, y'_2, encr(pubkey(agt(0)), y'_1, N(agt(A), agt(0))))$$
$$\vdash mesg(agt(a'_6), agt(a'), encr(pubkey(agt(a')), agt(a'_5), cons(N(agt(A), agt(0)), N(agt(A), agt(0)))))$$
$$\vdash mesg(agt(a'_4), agt(A), encr(pubkey(agt(A)), agt(a'_3), cons(N(agt(0), agt(0)), agt(0)))).$$

The last term cannot be further expanded. Using the function *recognizable* on an instance of such a term (for example, $\sigma' = \{0/a'_4, 0/a'_3\}$), q_f is derived and the strategy terminates with *success*. Hence, we have $N(agt(A), agt(0)) \to q_f$ and the transition $N(agt(B), agt(A)) \to q_f$ can also be generated from critical pairs. Thus, t is recognizable by the intruder and the property of secrecy of nonces is not true for the NSPK. It is worth noting that the transition $N(agt(B), agt(A)) \to q_f$ derives from a critical pair involving the transition $N(agt(A), agt(0)) \to q_f$ which is, in turn, generated by a critical pair with the basic transition $N(agt(0), agt(0)) \to q_f$.

If we run the strategy on the corrected version of the NSPK and the term $t = N(agt(B), agt(A))$, the strategy terminates with *failure*. In fact, the expansion set E becomes empty without deriving any transition $t \to q_f$. In order to prove that the secrecy of nonces is true for the protocol, the strategy must be applied to all terms in the (finite) language of $\mathcal{A}_{\overline{s}}$.

In a similar way it can be proved that the property of authentication is true for the corrected version of the NSPK, while it fails on the insecure version. In fact, given $t = c\_init(agt(B), agt(A), agt(0))$, the strategy derives *success*.

5 Concluding Remarks and Further Research

This paper has presented a rewriting strategy for the analysis and the verification of protocol specifications. Our experimentation with the authentication and secrecy properties on both the insecure and the corrected versions of the NSPK has shown that the strategy is able to detect the attacks in the insecure version, and fails when giving the same input term with the corrected version of the protocol. The correctness, termination and completeness of the strategy have been given for this case study by defining a specific notion of well-formedness on terms, which allows one to reduce the search space of the strategy. However, more study is needed on the relationship between the rules describing a protocol and the languages characterizing the properties under

consideration, in order to guarantee the termination and the completeness of the strategy.

With respect to the approach based on the approximation technique, the strategy carries out a bottom-up completion driven by the term representing the property to be proved, without generating the whole approximation automaton. Moreover, the strategy is able to say whether a property is satisfied or not, as it does not depend on any approximation function, and feedback on error location can be obtained in case of attacks by going back along the critical pairs in the bottom-up search.

The approximation technique provides an automatic manner for checking whether there are no attacks on a protocol specification, but has limitations on the set of properties that can be checked and is not able to assert that a property is not satisfied by a protocol due to the approximation function. On the contrary, using induction in Isabelle, Paulson has proved (or disproved) many properties of several kinds of security protocols using both a certain degree of automation and some interaction with the system. The knowledge and experience that a user needs to have in order to carry out some verification tasks on protocol specifications is much more relevant when using the inductive approach in Isabelle, than using the approximation technique in Timbuk. This evidence is the starting point for the combination of the two approaches presented in [20,21]. This simply consists in first applying the approximation technique to prove a property. If the property is satisfied, then this result is used as an axiom in Paulson's inductive method. Whenever the intersection between the languages of the tree automata is not empty, then the logical theories developed in Isabelle are used to prove or disprove the property. To make easier the link between the two systems, a translation is given from the Isabelle files to the input files for Timbuk. The strategy illustrated in this paper can be seen as a compromise between the full efficiency of the approximation technique and the full expressive power of Paulson's approach based on theorem proving.

As most protocols are formalized in a way similar to the NSPK, we think that the strategy can be used for verifying the authentication and secrecy properties of several kinds of protocols [25]. Future work includes (i) the application of the approach to other classes of protocols, (ii) the extension of the strategy to checking other properties of interest and (iii) the implementation of the strategy in a theorem proving environment.

Acknowledgements

We would like to thank Thomas Genet for kindly and promptly replying to all our questions and doubts on the approximation technique, Frédéric Oehl for sharing his Timbuk files with us, and the anonymous referees for many helpful comments on the preliminary version of the paper.

References

[1] F. Baader and T. Nipkow, *Term Rewriting and All That*, (Cambridge University Press, Cambridge, 1998).

[2] H. Cirstea, Specifying Authentication Protocols Using Rewriting and Strategies, in: *Proceedings PADL 2001*, Lecture Notes in Computer Science, Vol. 1990,Springer-Verlag, 138–152.

[3] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi, *Tree Automata Techniques and Applications*, `http://www.grappa.univ-lille3.fr/tata/`, 1997.

[4] G. Denker, J. Meseguer and C. Talcott, Protocol Specification and Analysis in Maude, in: *Proceedings 2nd WRLA Workshop*, (Pont-à-Mousson, France, 1998).

[5] N. Dershowitz and J.-P. Jouannaud, Rewrite Systems, in: J. van Leeuwen ed., *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, (North-Holland, Amsterdam, 1990), 243-320.

[6] T. Genet, Decidable Approximations of Sets of Descendants and Sets of Normal Forms, in: *Proceedings RTA 1998*, Lecture Notes in Computer Science, Vol. 1379, Springer-Verlag, 151–165.

[7] T. Genet and F. Klay, Rewriting for Cryptographic Protocol Verification, in: *Proceedings CADE 2000*, Lecture Notes in Artificial Intelligence, Vol. 1831, Springer-Verlag, 271–290.

[8] T. Genet and V. Viet Triem Tong, Timbuk - A tree automata library, `http://www.irisa.fr/lande/genet/timbuk`.

[9] T. Genet and V. Viet Triem Tong, Reachability Analysis of Term Rewriting Systems with Timbuk, in: *Proceedings LPAR 2001*, Lecture Notes in Artificial Intelligence, Vol. 2250, Springer-Verlag, 695–706.

[10] P. Inverardi and M. Nesi, A Strategy to Deal with Divergent Rewrite Systems, in: *Proceedings CTRS 1992*, Lecture Notes in Computer Science, Vol. 656, Springer-Verlag, 458–467.

[11] P. Inverardi and M. Nesi, Semi-equational Rewriting for Divergent Rewrite Systems, Technical Report No. 113, Department of Pure and Applied Mathematics, University of L'Aquila, July 1996.

[12] F. Jacquemard, M. Rusinowitch and L. Vigneron, Compiling and Verifying Security Protocols, in: *Proceedings LPAR 2000*, Lecture Notes in Artificial Intelligence, Vol. 1955, Springer-Verlag, 131–160.

[13] X. Leroy, D. Doligez, J. Garrigue, D. Rémy and J. Vouillon, The Objective Caml system, `http://caml.inria.fr/ocaml/`.

[14] G. Lowe, An Attack on the Needham-Schroeder Public-Key Authentication Protocol, *Information Processing Letters* **56** (1995) 131–133.

[15] G. Lowe, Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR, in: *Proceedings TACAS 1996*, Lecture Notes in Computer Science, Vol. 1055, Springer-Verlag, 147-166.

[16] C. A. Meadows, Analyzing the Needham-Schroeder Public-Key Protocol: A Comparison of Two Approaches, in: *Proceedings ESORICS 1996*, Lecture Notes in Computer Science, Vol. 1146, Springer-Verlag, 351-364.

[17] J. C. Mitchell, M. Mitchell and U. Stern, Automated analysis of cryptographic protocols using Murphi, in: *Proceedings IEEE Symposium on Security and Privacy 1997*, 141–153.

[18] D. Monniaux, Abstracting Cryptographic Protocols with Tree Automata, in: *Proceedings SAS 1999*, Lecture Notes in Computer Science, Vol. 1694, Springer-Verlag, 149–163.

[19] R. M. Needham and M. D. Schroeder, Using Encryption for Authentication in Large Networks of Computers, *Communications of the ACM* **21(12)** (1978) 993–999.

[20] F. Oehl and D. Sinclair, Combining two approaches for the verification of cryptographic protocols, in: *Proceedings Workshop on Specification, Analysis and Validation for Emerging Technologies in Computational Logic (SAVE 2001)*, (Paphos, Cyprus, December 2001).

[21] F. Oehl and D. Sinclair, Combining Isabelle and Timbuk for Cryptographic Protocol Verification, in: *Proceedings Workshop on Securité de la Communication sur Internet (SECI 2002)*, (Tunis, September 2002), 57–67.

[22] F. Oehl, G. Cécé, O. Kouchnarenko and D. Sinclair, Automatic Approximation for the Verification of Cryptographic Protocols, INRIA Technical Report RR-4599 (October 2002).

[23] L. Paulson, Proving Properties of Security Protocols by Induction, in: *Proceedings 10th Computer Security Foundations Workshop*, (IEEE Computer Society Press, 1997), 70–83.

[24] L. C. Paulson, The Inductive Approach to Verifying Cryptographic Protocols, *Journal on Computer Security* **6** (1998) 85–128.

[25] P. Syverson and I. Cervesato, The Logic of Authentication Protocols, in: *Proceedings Foundations of Security Analysis and Design 2000*, Lecture Notes in Computer Science, Vol. 2171, Springer-Verlag, 63–136.

[26] C. Weidenbach, Towards an Automatic Analysis of Security Protocols in First-Order Logic, in: *Proceedings CADE 1999*, Lecture Notes in Artificial Intelligence, Vol. 1632, Springer-Verlag, 314–328.

Innermost Reductions Find All Normal Forms on Right-Linear Terminating Overlay TRSs

Masahiko Sakai [1], Kouji Okamoto [2], Toshiki Sakabe [3]

Graduate School of Information Science,
Nagoya University,
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan

Abstract

A strategy of TRSs is said to be complete if all normal forms of a given term are reachable from the term. We show the innermost strategy is complete for terminating, right-linear and overlay TRSs. This strategy is fairly efficient to calculate all normal forms of a given term by searching reduction trees. We also discuss the possibilities for weakening the conditions.

1 Introduction

Most of properties of non-CR (Church Rosser) or non-UN (uniquely normalizing) TRSs are still hidden in mist. Concerning normalizing strategies that guarantee a safe evaluation in order to obtain a normal form, a lot of studies are effective on orthogonal TRSs[7,9,13,14,5,6]. There are several studies on normalizing strategies for non-orthogonal TRSs[11,18,3], which are effective on UN TRSs.

On the other hand, the authors have been studying on a transformational approach of inverse computation of functions given by a TRS[15,16]. The conversion itself can be done for constructor TRSs, which contain no defined symbol in any proper subterms of the left-hand sides. For instance, consider the following TRS:

$$R_1 = \begin{cases} d(0) \to 0, \\ d(s(0)) \to 0, \\ d(s(s(x)) \to s(d(x)) \end{cases}$$

[1] Email:sakai@is.nagoya-u.ac.jp
[2] Email:okamo@sakabe.nuie.nagoya-u.ac.jp
[3] Email:sakabe@is.nagoya-u.ac.jp

where the function d is for division by two. The transformation[15,16] gives the following TRS from R_1:

$$R_2 = \begin{cases} D(0) \to 0, \\ D(0) \to s(0), \\ D(s(y)) \to U(D(y)), \\ U(x) \to s(s(x)) \end{cases}.$$

This TRS R_2 has a function D that calculates inverse image with respect to d; since $d(s(s(s(0)))) \xrightarrow{*}_{R_1} s(0)$, we have $D(s(0)) \xrightarrow{*}_{R_2} s(s(s(0)))$. On the other hand, $D(s(0))$ is also reachable to $s(s(0))$, because $d(s(s(0))) \xrightarrow{*}_{R_1} s(0)$. As this example shows, the output TRSs are non-UN in general. Thus, strategies that can find all normal forms are important.

In this paper, we first give the notion of the complete strategy, which can find all normal forms of a given term. Then, we show that innermost reductions is a complete strategy of terminating, right-linear and overlay TRSs. On UN TRSs, we know that any strategy is complete, of course. However, it is not true in non-UN setting.

As a result, it appears that the innermost reductions contribute in efficiency to find all normal forms shown as follows.

Example 1.1 Consider the following TRS:

$$R_3 = \begin{cases} f(x) \to x, \\ g(x) \to i(x), \\ h(x, y) \to y, \\ h(x, i(y)) \to x \end{cases}$$

R_3 is terminating but not UN (thus not CR). Hence, we have to search all spaces in order to obtain all normal forms 0 and $i(0)$ of a term $h(f(0), g(0))$. We have 14 reductions required to calculate its all normal forms without memorizing terms encountered through the search. The search space is reduced to only 4 reductions by the leftmost innermost reduction as shown in Figure 1. Of course, we can use dag search which takes 11 reductions. However constructing the dag that shares the same terms is considerably heavy.

Consider the transformation[15,16] again. In general, the output TRSs are non-terminating and overlay, and contain extra variables, where extra variables are variables that appear in the right-hand side of a rule and not in the left-hand side of the rule. However, we know that the output TRSs have no extra variables if the inputs are non-erasing, and that they are right-linear if the inputs are left-linear. Therefore, innermost reductions contribute to an efficient computation of the inverse image of functions, if the input constructor TRSs are left-linear and non-erasing and the output TRSs are terminating.

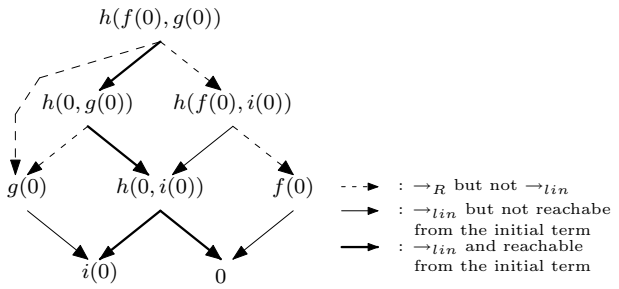


Fig. 1. The search space from $h(f(0), g(0))$ on R_3

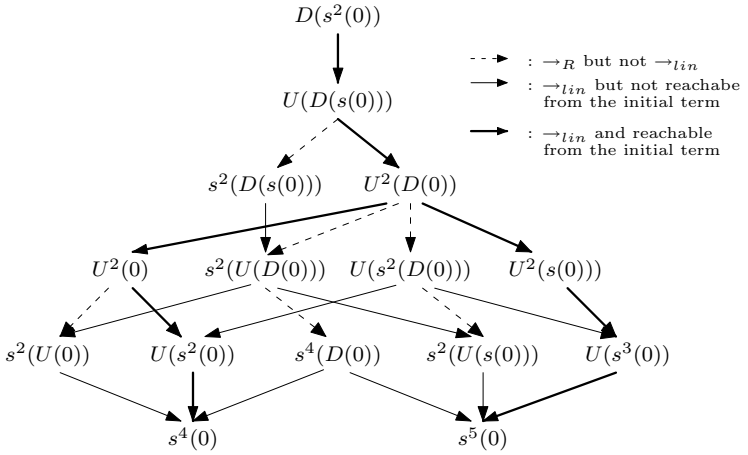


Fig. 2. The search space from $D(s(s(0)))$ on R_2

Example 1.2 Consider the TRS R_2 above. While 34 reductions are required to calculate all normal forms of $D(s(s(0)))$ by the depth-first tree search on $\rightarrow_{R_2}$, the search space is reduced to only 8 reductions by the leftmost innermost reduction as shown in Figure 2.

2 Preliminary Concepts

In this paper, we mainly follow the notation of [4,12]. An *abstraction reduction system* is a structure $A = (S, \rightarrow)$ where S is a set and $\rightarrow$ is a binary relation over S, which is called a *reduction relation*. A *reduction sequence* is a finite sequence $a_0 \rightarrow a_1 \rightarrow \cdots \rightarrow a_n$ $(n \geq 0)$ or an infinite sequence $a_0 \rightarrow a_1 \rightarrow \cdots$ of reductions. The identity of elements a and b in S is denoted by $a \equiv b$. The reflexive and transitive closure of $\rightarrow$ is denoted by $\stackrel{*}{\rightarrow}$. The transitive closure of $\rightarrow$ is denoted by $\stackrel{+}{\rightarrow}$. If there is no element b such that $a \rightarrow b$, then we say a is a *normal form* (with respect to A). We use NF_A or $NF_{\rightarrow}$ for the set of all normal forms. If $a \stackrel{*}{\rightarrow} b \in NF_A$, we say b is a normal form of a or a has a normal form b. We say b is *reachable* from a (by $\rightarrow$) if $a \stackrel{*}{\rightarrow} b$. We say A is *confluent* (or equivalently Church-Rosser, CR) if $a \stackrel{*}{\rightarrow} b \wedge a \stackrel{*}{\rightarrow} b'$ implies $\exists c \in S$, $b \stackrel{*}{\rightarrow} c \wedge b' \stackrel{*}{\rightarrow} c$ for all a, b and b' in S. We say A is *uniquely normalizing* (UN) if every element a has at most one normal forms. We say

A is *terminating* if there exists no infinite reduction sequence.

Let F be a set of function symbols accompanied with a mapping *arity* from F to the set of natural numbers, which is called a *signature*. Let X be a set of variables. The set of all terms over F and X is denoted by $T(F, X)$ (or simply T). We often use f and g for function symbols, x, y and z for variables, and s, t and u for terms. The set of variables which appear in a term t is denoted by $Var(t)$. For any term t and function symbol f with $arity(f) = 1$, we use $f^n(t)$ to represent $\overbrace{f(\cdots f(}^{n} t)\cdots)$. A term t is called *linear* if every variable in t occurs only once.

We use an extra constant $\Box$ as *hole* in terms. A term C in $T(F \cup \{\Box\}, X)$ is called a *context*. For a context C with n $\Box$s and for $t_1, \ldots, t_n \in T(F, X)$, $C[t_1, \ldots, t_n]$ denotes the term obtained by replacing $\Box$s with $t_1, \ldots, t_n$ from left to right order. In the sequel, we use contexts that possess exactly one $\Box$. We say a reduction $\rightarrow$ is monotonic if $t \rightarrow s$ implies $C[t] \rightarrow C[s]$ for any context C. If $t \equiv C[s]$, we say that s is a *subterm* of t written as $t \trianglerighteq s$. Moreover, if $C \not\equiv \Box$, s is a *proper subterm* of t, denoted by $t \triangleright s$. The reduction $(\rightarrow \cup \triangleright)$ is terminating if $\rightarrow$ is monotonic and terminating.

The notation $\{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$ denotes a *substitution* σ such that $x_i\sigma \equiv u_i$.

A *rewrite rule* is a pair (l, r), denoted by $l \rightarrow r$, where the left-hand side l $(\notin X)$ and the right-hand side r are terms such that $Var(l) \supseteq Var(r)$. Letting R be a set of rewrite rules, $(T(F, X), \rightarrow_R)$ is called a *term rewriting system* (*TRS*), where $s \rightarrow_R t$ if and only if $s \equiv C[l\sigma]$ and $t \equiv C[r\sigma]$ for some $l \rightarrow r$ in R, context C and substitution σ. We say $l\sigma$ a redex. If $C \equiv \Box$ we say it is a top reduction, denoted by $s \xrightarrow[\varepsilon]{} t$; Otherwise, $s \xrightarrow[\varepsilon<]{} t$. In the sequel, we identify R with $(T(F, X), \rightarrow_R)$ if that causes no confusion. An another definition of $\rightarrow_R$ is as follows:

(a) $l\sigma \rightarrow_R r\sigma$ for a substitution σ and a rewrite rule $l \rightarrow r$.

(b) $f(s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots, s_n) \rightarrow_R f(s_1, \ldots, s_{i-1}, \mathrm{t}, s_{i+1}, \ldots, s_n)$, if $s \rightarrow_R t$.

We say that a rewrite rule $l \rightarrow r$ is *right-linear* if r is linear, and say that a rewrite rule $l \rightarrow r$ is *left-linear* if l is linear. A TRS is called *right-linear* (*left-linear*) if every rule is right-linear (left-linear). The rules $l \rightarrow r$ and $l' \rightarrow r'$ *overlap* if there exist a subterm s of l' and substitutions σ and σ' such that $s \notin X$ and $l\sigma \equiv s\sigma'$. Especially, we say that they overlap at non-root position if $s \not\equiv l'$. A TRS is *overlay* if it has no overlapping rules at non-root position. A TRS is *non-erasing* if $Var(l) = Var(r)$ for every rule $l \rightarrow r$.

3 Complete strategy of TRSs

Firstly, we give a notion of complete strategy.

Definition 3.1 Let R be a TRS. A relation $\rightarrow_c$ is called a *strategy* of R, if

(a) $\rightarrow_c \subseteq \rightarrow_R$, and

(b) $NF_R = NF_{\to_c}$.

A strategy $\to_c$ of R is *complete*, if

(c) $t \xrightarrow{*}_R u \in NF_R$ implies $t \xrightarrow{*}_c u$.

Obviously $\to_R$ itself is a trivial complete strategy of R, although it is nonsense.

We can regard the sequence $t \xrightarrow{*}_c u$ as a standard sequence of $t \xrightarrow{*}_R u$, if the sequence $t \xrightarrow{*}_c u$ is unique. Hence, the notion of complete strategies are related to the notion of standardizations.

If we can find a terminating complete strategy $\to_c$, it can find all normal forms of a given term with respect to R. Even if R is terminating, non-trivial complete strategies are worthful, because they contribute to the efficiency. Note that the condition (b) is not necessary for finding all normal forms. Assume we want to find all normal forms of t by using a terminating strategy $\xrightarrow{*}_c$ that satisfies (a) and (c). Since $NF_R \subseteq NF_{\to_c}$ by (a), the only difference is that it may find a normal form of t in $NF_{\to_c}$ but not in NF_R. However, we can simply dispose the term since (c) ensures that all normal forms of t with respect R are reachable from t by $\to_c$.

Next, we give a formal definition of innermost reductions.

Definition 3.2 Let R be a TRS. The *innermost* reduction relation of R, denoted by $\to_{in}$, is defined as follows:

(a) $l\sigma \to_{in} r\sigma$ for any substitution σ and rewrite rule $l \to r$ if all proper subterms of $l\sigma$ are normal forms.

(b) $f(s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots, s_n) \to_{in} f(s_1, \ldots, s_{i-1}, t, s_{i+1}, \ldots, s_n)$ if $s \to_{in} t$.

The *leftmost innermost* reduction $\to_{lin}$ (*rightmost innermost* reduction $\to_{rin}$) is defined in similar to above by adding to (b) an extra condition that $s_1, \ldots, s_{i-1}$ ($s_{i+1}, \ldots, s_n$) are normal forms.

Obviously, $\to_{lin} \subseteq \to_{in} \subseteq \to_R$ and $\to_{rin} \subseteq \to_{in} \subseteq \to_R$.

The following two lemmas show general properties of innermost reductions.

Lemma 3.3 *Let R be a TRS. Let $\to_c$ be either $\to_{lin}$, $\to_{rin}$, or $\to_{in}$. If $t \xrightarrow{*}_c s \in NF_R$, then there exists a term t' such that $t \xrightarrow[\varepsilon<]{*}_c t' \xrightarrow{*}_c s$ and every proper subterm of t' is a normal form.*

Proof. If the reduction sequence contains no top reduction, it is trivial by taking t as t'. Otherwise, we can represent the reduction sequence as $t \xrightarrow[\varepsilon<]{*}_c t' \xrightarrow[\varepsilon]{}_c t'' \xrightarrow{*}_c s$. Since $t' \xrightarrow[\varepsilon]{}_c t''$ is innermost, all proper subterms of t' are normal forms. □

Lemma 3.4 *Let R be a TRS, t be a linear term, s be a normal form, and σ be a substitution. Let $\to_c$ be either $\to_{lin}$, $\to_{rin}$, or $\to_{in}$. If $t\sigma \xrightarrow{*}_c s$, then there exists a normalized substitution σ' such that $t\sigma \xrightarrow{*}_{in} t\sigma' \xrightarrow{*}_c s$.*

Proof. We prove by structural induction on t. In the case that t is a variable x, it is enough by taking $\{x \mapsto s\}$ as σ'. In the case that $t \equiv f(t_1, \ldots, t_n)$, we

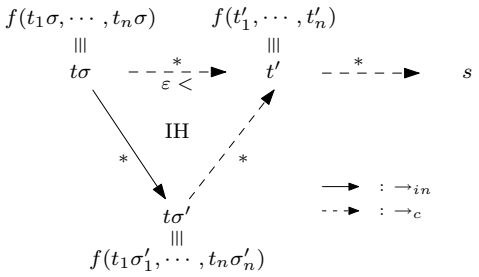


Fig. 3. The proof of Lemma 3.4

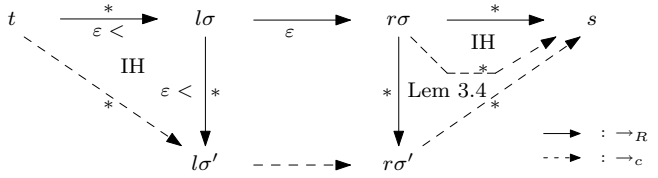


Fig. 4. The proof of Theorem 3.5 (the former case)

have $t\sigma \equiv f(t_1\sigma,\ldots,t_n\sigma)$. From Lemma 3.3, there exists a term t' such that $t\sigma \xrightarrow[\varepsilon<]{*}{}_c t' \xrightarrow{*}_c s$ and every proper subterm of t' is a normal form (See Figure 3). Since there is no top reduction in $t\sigma \xrightarrow[\varepsilon<]{*}{}_c t'$, the term t' can be represented as $f(t'_1,\ldots,t'_n)$. Since we have $t_i\sigma \xrightarrow{*}_R t'_i$, it follows from induction hypothesis that there exists normalized substitution σ'_i such that $t_i\sigma \xrightarrow{*}_{in} t_i\sigma'_i \xrightarrow{*}_c t'_i$ for each i. Hence, $t\sigma \equiv f(t_1\sigma,\ldots,t_n\sigma) \xrightarrow{*}_{in} f(t_1\sigma'_1,\ldots,t_n\sigma'_n) \xrightarrow{*}_c f(t'_i,\ldots,t'_n) \equiv t' \xrightarrow{*}_c s$. Here, we can compose σ'_is into one substitution σ' from the linearity of t. Therefore, we conclude this case since $f(t_1\sigma'_1,\ldots,t_n\sigma'_n) \equiv t\sigma'$. □

Theorem 3.5 *Let R be a right-linear terminating overlay TRS. Then, the innermost strategy, the leftmost innermost strategy and the rightmost innermost strategy are complete strategies of R.*

Proof. Let $\to_c$ be either $\to_{lin}$, $\to_{rin}$, or $\to_{in}$. We prove that $t \xrightarrow{*}_R s \in NF_R$ implies $t \xrightarrow{*}_c s$ by Noetherian Induction on t with respect to $(\to_R \cup \rhd)$. We have two cases whether $t \xrightarrow{*}_R s$ contains top reductions or not.

Firstly, we consider the latter case that is easier. Let $t \equiv f(t_1,\ldots,t_n)$. Then, s can be represented as $f(s_1,\ldots,s_n)$. Since $t_i \xrightarrow{*}_R s_i \in NF_R$ for all i, we have $t_i \xrightarrow{*}_c s_i$ by induction hypothesis. Thus, $t \xrightarrow{*}_c s$ follows.

Secondly, we consider the former case. By focusing the first top reduction, it can be represented as $t \xrightarrow[\varepsilon<]{*}{}_R l\sigma \underset{\varepsilon}{\to}{}_R r\sigma \xrightarrow{*}_R s \in NF_R$ (See Figure 4). Since $t \xrightarrow{+}_R r\sigma$ we can apply the induction hypothesis to $r\sigma \xrightarrow{*}_R s$, which results $r\sigma \xrightarrow{*}_c s$. It follows from the right-linearity of r and Lemma 3.4 that $r\sigma \xrightarrow{*}_R r\sigma' \xrightarrow{*}_c s$ for some normalized substitution σ'. Then, we have $l\sigma' \to_c r\sigma'$ since σ' is normalized and R is overlay. We also have $t \xrightarrow[\varepsilon<]{*} l\sigma \xrightarrow[\varepsilon<]{*} l\sigma'$ since l is not a variable. Let $t \equiv f(t_1,\ldots,t_n)$. Then, $l\sigma'$ can be represented as $f(s_1,\ldots,s_n)$ and $t_i \xrightarrow{*}_R s_i \in NF_R$. we have $t_i \xrightarrow{*}_c s_i$ by induction hypothesis. Whatever $\to_c$ is either of $\to_{lin}$, $\to_{rin}$ or $\to_{in}$, we can show $t \xrightarrow{*}_c l\sigma'$. Therefore, $t \xrightarrow{*}_c s$.□

Followings are counter examples showing that all of the conditions in Theorem 3.5 are essential.

Example 3.6 Consider the following TRS that is overlay and right linear but not terminating:

$$R_4 = \begin{cases} f(x) \to b, \\ a \to a \end{cases}.$$

We have $f(a) \to_{R_4} b \in NF_{R_4}$ but not $f(a) \xrightarrow{*}_{in} b$.

Example 3.7 Consider the following TRS that is terminating and right linear but not overlay:

$$R_5 = \begin{cases} f(a) \to b, \\ a \to c \end{cases}.$$

We have $f(a) \to_{R_5} b \in NF_{R_5}$ but not $f(a) \xrightarrow{*}_{in} b$.

Example 3.8 Consider the following TRS that is terminating and overlay but not right linear:

$$R_6 = \begin{cases} f(x) \to g(x, x), \\ a \to b, \\ a \to c \end{cases}.$$

We have $f(a) \xrightarrow{*}_{R_6} g(b, c) \in NF_{R_6}$ but not $f(a) \xrightarrow{*}_{in} g(b, c)$.

4 Discussion

We discuss the possibilities for weakening the conditions of Theorem 3.5.

Let's consider the condition "terminating". Even if we replace it by the condition "innermost terminating", the well-founded ordering used in Theorem 3.5 works no more. Nevertheless, we think the conjecture obtained from the theorem by completely removing the condition may hold if we add extra conditions "left-linear" and "non-erasing", because the number of reductions may work as a measurement for the proof. Moreover, if so, the condition "left-linear" will be removed by using the parallel reduction[8]. Hence, we give the following conjecture.

Conjecture 4.1 *Let R be a right-linear non-erasing overlay TRS. Then, the innermost strategy, the leftmost innermost strategy and the rightmost innermost strategy are complete strategies of R.*

Let's consider the following overlay TRS that is not innermost terminating.

$$R_7 = \begin{cases} f(b,c) \to f(a,a), \\ a \to b, \\ a \to c \end{cases} .$$

We have leftmost innermost reduction sequences from $f(b,c)$ to normal forms $f(b,b)$, $f(c,b)$ and $f(c,c)$ of a term $f(b,c)$. For instance, $f(b,c) \to_{R_7} f(a,a) \to_{R_7} f(c,a) \to_{R_7} f(c,c)$. Hence, the leftmost innermost strategy is complete for R_7.

Even if Conjecture 4.1 holds, we require "innermost terminating" in order to find all normal forms as long as we use innermost strategies. The dependency pair method[1,2] is usable to show the innermost terminating property of TRSs.

If we remove the condition "non-erasing" from Conjecture 4.1, it does not hold any more as shown by Example 3.6. In this case, the authors think that the basic strategy used in basic narrowing[10] is a candidate for a replacement of innermost strategies. Intuitively, a reduction sequence is basic, if every redex, ever substituted into a variable of the rule in a previous reduction, is not reduced. Now, Example 3.6 is not a counter example, because $f(a) \to_{R_4} b$ is a basic reduction sequence. Thus, we have another conjecture.

Conjecture 4.2 *Let R be a right-linear overlay TRS. Then, the basic strategy is a complete strategy of R.*

How about the condition "overlay"? In case that we have non-overlay critical pairs like Example 3.7, complete strategies cannot ignore either reducts of the critical pairs. Hence, we need to introduce weakly innermost strategy, where a redex is weakly innermost if it is innermost or it overlaps some innermost redex. The conjecture is as follows:

Conjecture 4.3 *Let R be a right-linear terminating TRS. Then, the weakly innermost strategy is a complete strategy of R.*

On the other hand, removing the condition "right-linear" seems to be impossible as long as we use strategies based on innermost reduction.

As for outermost based reductions, the authors think that there is a possibility to work as complete strategies of left-linear overlay TRSs. For example, consider the following TRS:

$$R_8 = \begin{cases} f(a,x) \to c, \\ f(x,a) \to d, \\ b \to a \end{cases} .$$

For a sequence $f(b,b) \xrightarrow{*}_{R_8} d \in NF_{R_8}$, we have no leftmost outermost re-

duction sequence, which shows that the leftmost outermost strategy is not complete for R_8. However, the outermost strategy is complete, although it is not efficient. The authors prospect that the left-linearity is needed at least, since we have the following counter example:

$$R_9 = \begin{cases} f(x,x) \to c, \\ f(x,y) \to d, \\ a \to b \end{cases} .$$

Although $f(a,b) \xrightarrow{*}_{R_9} c$, the only outermost sequence is $f(a,b) \to_{R_9} d$.

Developing normalizing strategies, we mean terminating complete strategy of TRSs, supposed to be difficult but should be explored. As for TRSs with extra variables, which we call EV-TRS, the authors have proposed a simulation method[17]. Hence, complete strategies for the simulation are also to be explored.

Acknowledgement

We thank anonymous referees, Dr. Mizuhito Ogawa, and members of TRS Meeting Japan for suggestions. This work is partially supported by Grants from Ministry of Education, Science and Culture of Japan #15500007, by the Intelligent Media Integration Project of Nagoya University as one of the 21st Century COE of JAPAN and by the Nitto foundation.

References

[1] T. Arts and J. Gisel, Proving Innermost Termination Automatically, Proc. of the 8th Conf. on Rewriting Techniques and Applications (RTA'97), LNCS, *1232*, 1997, pp.157–171.

[2] T. Arts, Automatically Proving Termination and Innermost Normalization of Term Rewriting Systems, PhD thesis, Universiteit Utrecht, 1997.

[3] S. Antoy and A. Middeldorp, A Sequential Reduction Strategy, Theoretical Computer Science, *165(1)*, 1996, pp.75–95.

[4] F. Baader and T. Nipkow, Term Rewriting and All That, Cambridge University Press, 1998.

[5] H. Comon, Sequentiality, Monadic Second Order Logic and Tree Automata, Information and Computation, *157*, 2000, pp.25–51.

[6] I. Durand and A. Middeldorp, Decidable Call by Need Computations in Term Rewriting (Extended Abstract), Proc. of the 14th Int. Conf. on Automated Deduction (CADE'97), LNAI, *1249*, 1997, pp.4–18.

[7] G. Huet and J.-J. Lévy, Call-by-Need Computations in Non-Ambiguous Linear Term Rewriting Systems, Rapport INRIA, *359*, 1979.

[8] G. Huet, Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems, Journal of the ACM, *27*, 1980, pp.797–821.

[9] G. Huet and J.-J. Lévy, Computations in Orthogonal Rewriting Systems, I and II, in Computational Logic, Essays in Honor of Alan Robinson, The MIT Press, 1991, pp.396–443.

[10] J.M. Hullot, Canonical Forms and Unification, Proc. of the 5th Conf. on Automated Deduction (CADE'80), LNCS, *87*, 1980, pp.318–334.

[11] J.R. Kennaway, Sequential Evaluation Strategies for Parallel-Or and Related Reduction Systems, Annals of Pure and Applied Logic, *43*, 1989, pp.31–56.

[12] J. W. Klop, Term Rewriting Systems, in *Handbook of Logic in Computer Science*, ed.Abramsky et al., Oxford University Press, 1992.

[13] M. Oyamaguchi, NV-Sequentiality: A Decidable Condition for Call-by-Need Computations in Term Rewriting Systems, SIAM Journal on Computing, *22(1)*, 1993, pp.114-135.

[14] T. Nagaya, M. Sakai and Y. Toyama, NVNF-Sequentiality of Left-Linear Term Rewriting Systems, RIMS Technical Report, *918*, 1995, pp.109–117.

[15] N. Nishida, M. Sakai and T. Sakabe, Generation of Inverse Term Rewriting Systems for Pure Treeless Functions, The International Workshop on Rewriting in Proof and Computation (RPC'01), Sendai, Japan, October 25-27, 2001, pp.188–198.

[16] N. Nishida, M. Sakai and T. Sakabe, Generation of a TRS Implementing the Inverses of the Functions with Specified Arguments Fixed, Technical Report of IEICE, *COMP2001-67*, 2001, pp.33–40.

[17] N. Nishida, M. Sakai and T. Sakabe, Narrowing-based Simulation of Term Rewriting Systems with Extra variables and its Termination Proof, Proc. of 12th Int'l Workshop on Functional and (Constraint) Logic Programming (WFLP'03), Valencia, Spain, June 12–13, 2003(to appear).

[18] Y. Toyama, Strong Sequentiality of Left-Linear Overlapping Term Rewriting Systems, The 7th annual IEEE Symposium on Logic in Computer Science, 1992, pp.274–284,

Strategies and User Interfaces in Maude (Extended Abstract)

Manuel Clavel [1,2]

Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid
Madrid, Spain

Abstract

Maude is a high-level language and a high-performance system supporting executable specification and declarative programming in rewriting logic. Rewriting logic is reflective, in the sense of being able to express its own metalevel at the object level. Reflection is systematically exploited in Maude endowing the language with powerful metaprogramming capabilities. This paper explains and illustrates with examples two of these reflective features: user-definable declarative strategies and user-definable declarative interfaces. We conclude with an application, namely, an inductive theorem prover, whose implementation effectively combines both reflective features.

The Maude System

Maude [4] is a high-level language and a high-performance system supporting executable specification and declarative programming in rewriting logic [7]. Since rewriting logic contains equational logic, Maude also supports equational specification and programming in its sublanguage of functional modules. The underlying equational logic chosen for Maude is membership equational logic [8], that has sorts, subsorts, operator overloading, and partiality definable by membership and equality conditions. Rewriting logic is reflective [6,5], in the sense of being able to express its own metalevel at the object level. Reflection is systematically exploited in Maude endowing the language with powerful metaprogramming capabilities [3], including both user-definable strategies to guide the deduction process and user-definable interfaces to interact with systems which are specified using rewrite rules.

[1] Research supported by Spanish CICYT Project MELODIAS TIC2002-01167.
[2] Email: clavel@sip.ucm.es

Reflection in Maude

In Maude, reflection is supported through its predefined `META-LEVEL` module. In this module:

- Maude terms are metarepresented as elements of a data type `Term` of terms;
- Maude modules are metarepresented as terms in a data type `Module` of modules;
- the process of reducing a term to normal form using Maude's `reduce` command is metarepresented by a built-in function `metaReduce`;
- the process of applying (without extension) a rule in a module at the top of a term is metarepresented by a built-in function `metaApply`;
- the process of applying (with extension) a rule in a module at any position of a term is metarepresented by a built-in function `metaXapply`;
- the processes of rewriting a term in a module using Maude's `rewrite`, and `frewrite` commands are metarepresented by built-in functions `metaRewrite`, and `metaFrewrite`;
- the process of matching (without extension) two terms at the top is metarepresented by a built-in function `metaMatch`;
- the process of matching (with extension) a pattern to any subterm of a term is metarepresented by a built-in function `metaXmatch`; and
- parsing and pretty printing of a term in a module, as well as key sort operations such as comparing sorts in the subsort ordering of a signature, are also metarepresented by corresponding built-in functions.

Strategies in Maude

The most general Maude modules are system modules. These are rewrite theories that do not need to be Church-Rosser and terminating. Therefore, we need to have good ways of controlling the rewriting inference process —which in principle could not terminate or could go in many undesired directions— by means of adequate *strategies*. This need has been addressed in other languages; for example, the ELAN language provides a strategy language to guide the rewrites and allows user extensions for such a language [1]. In Maude, thanks to its reflective capabilities, strategies are made *internal* to the logic, that is, they are defined by rewrite rules in a normal module in Maude, and can be reasoned about as with rules in any other module.

In fact, there is great freedom for defining many different types of strategies, or even many different strategy languages inside Maude. This can be done in a completely user-definable way, so that users are not limited by a fixed and closed particular strategy language. A general methodology for defining internal strategy languages for reflective logic is introduced in [2]. In general, strategies are defined in extensions of the `META-LEVEL` module by using `metaReduce`, `metaRewrite` `metaApply`, and `metaXapply` as building blocks.

User interfaces in Maude

In Maude, a general input/output facility is supported through the `LOOP-MODE` module. In this module, there is a special operator `[_,_,_]` with three arguments that provides a generic read-eval-print loop. Using object oriented concepts, this operator can be seen as a persistent object —that we call the *loop object*— with an input channel (the first argument) that accepts lists of quoted identifiers, an output channel (the third argument) that yields lists of quoted identifiers, and a state (given by its second argument). Currently, the way to distinguish the inputs passed to the loop object from the inputs passed to the Maude system is by enclosing them in parentheses. When something is written in the Maude prompt enclosed in parentheses it is converted into a list of quoted identifiers, which is then placed in the first slot of the loop object. The output is handled in the reverse way, that is, the list of quoted identifiers placed in the third slot of the loop object is printed on the terminal after "unquoting" each of the identifiers in the list. We can think of the input and output events as *implicit rewrites* that transfer —in a slightly modified, quoted or unquoted form— the input and output data between two objects, namely the loop object and the "user" or "terminal" object.

Besides providing an input and output channels, the operator `[_,_,_]` gives us the possibility of maintaining a persistent state in its second component. There is complete flexibility for defining the terms we want to have for representing the persistent state of the loop in each particular application.

The ITP Tool

Using the reflective features of Maude, we have built an inductive theorem prover [2] for equational specifications that can be used to prove inductive properties of functional modules in Maude. The inference system for inductive proofs is specified at the metalevel, since it uses the functional module about which we want to prove inductive theorems as data. The user interacts with the tool by typing proof commands in an interface. This interface converts the commands typed by the user into strategies controlling the application of the inference rules. Both the interface and the strategies are implemented in Maude in modules extending, respectively, the `LOOP-MODE` and the `META-LEVEL` modules.

References

[1] P. Borovanský, C. Kirchner, H. Kirchner, and P.-E. Moreau. ELAN from a rewriting logic point of view. *Theoretical Computer Science*, 285(2):155–185, August 2002.

[2] M. Clavel. *Reflection in Rewriting Logic: Metalogical Foundations and Metaprogramming Applications*. CSLI Publications, 2000.

[3] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: Specification and programming in rewriting logic.

Manual distributed as documentation of the Maude system, Computer Science Laboratory, SRI International. `http://maude.csl.sri.com/manual`, Jan. 1999.

[4] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.

[5] M. Clavel and J. Meseguer. Reflection in conditional rewriting logic. *Theoretical Computer Science*, 285(2):245–288, August 2002.

[6] M. Clavel, J. Meseguer, and M. Palomino. Reflection in membership equational logic, many-sorted equational logic, Horn logic with equality, and rewriting logic. In F. Gadducci and U. Montanari, editors, *Proc. Fourth International Workshop on Rewriting Logic and its Applications*, volume 71 of *Electronic Notes in Theoretical Computer Science*, pages 63–78. Elsevier, 2002.

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

[8] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proc. WADT'97*, volume 1376 of *LNCS*, pages 18–61. Springer-Verlag, 1998.

Weak Reduction and Garbage Collection in Interaction Nets

Jorge Sousa Pinto [1]

Departamento de Informática
Universidade do Minho
4710-057 Braga, Portugal

Abstract

This paper presents an implementation device for the *weak reduction of interaction nets to interface normal form*. The results produced by running several benchmarks are given, suggesting that weak reduction greatly improves the performance of the interaction combinators-based implementation of the λ-calculus.

1 Introduction

The main purpose of this paper is to present an implementation mechanism for weak interaction net reduction. This strategy never triggers reductions in disconnected parts of the net, and in fact it may stop before all the redexes in the connected component (with respect to the interface) are reduced. This mechanism is particularly useful for the evaluation of functional programs.

An interaction net [6] is an undirected graph in which edges are connected to specific positions (called *ports*) in the nodes (with at most one edge connected to each port). Each node is labeled with a symbol and has a distinguished *principal port*. Redexes (here called *active pairs*) are pairs of adjacent nodes connected by their principal ports. The remaining ports of a node are called *auxiliary*. The *interface* of the net is the set of its free ports (i.e. ports with no edge connected).

An *interaction rule* is a graph-rewriting rule which replaces an active pair by a net such that all the connections (i.e. the interface of the active pair) are preserved. Figure 1(a) contains an example of such a rule. An *interaction system* is a set of rules such that there is at most one applicable rule for each pair of symbols. The main property of this rewriting framework is *strong local confluence* (diamond property), a consequence of the fact that each node is involved in no more than one active pair. The number of steps required for fully reducing a net is thus fixed.

[1] Email:jsp@di.uminho.pt

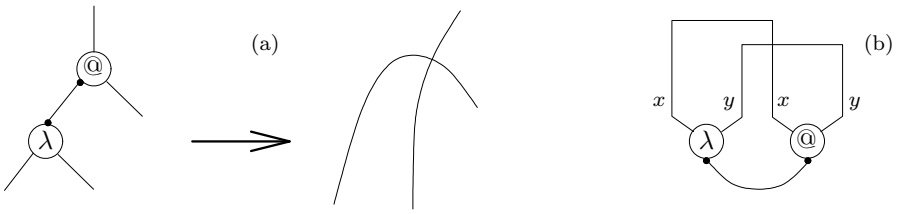


Fig. 1. (a) Example interaction rule, (b) redrawn for conversion to text format

INs and Functional Programming.

Interaction nets have been used as a technique for the implementation of functional programs, characterized by a high degree of *sharing* of computations – interaction nets are notably used when *optimal β-reduction* is sought [1,5,8].

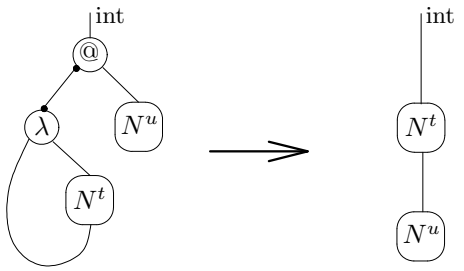


Fig. 2. A net representing the β-redex $(\lambda x.t)u$ – 'int' is its interface; principal ports are marked with a dot; N^t and N^u are the nets representing terms t and u, and the curved wire in the left-hand side represents the bound variable x. A step of reduction (using the rule of figure 1) results in the net on the right

β-redexes are in general encoded as active pairs consisting of a binary @ agent (standing for application) and a binary λ agent (for abstractions), as shown in figure 2. To understand why interaction nets are good at sharing, observe that in the λ-calculus computations are duplicated when a bound variable is substituted in a term where it occurs more than once. These multiple occurrences would typically be encoded with duplicating agents (denoted by δ) which copy the substituted term to obtain the required number of copies.

It is then easy to understand that *copying forces evaluation*, in the sense that, since the principal port of the @ agent is facing the λ agent, it is not possible to duplicate a redex by connecting a δ agent to it: the redex must first be reduced (see figure 3). We remark that this discussion is too simplistic, since the real problem has to do with the duplication of virtual, rather than explicit, redexes.

We remark that it is not obvious from the above discussion how to encode λ-terms as interaction nets so that net reduction soundly corresponds to β-reduction – when variables occur non-linearly, some mechanism must be used to correctly handle the duplication of terms. For instance, each of the encodings (of expressions into nets) reviewed in section 5 handles this problem differently. Studying how such an encoding behaves with respect to the sharing of computations is an extremely complex issue.

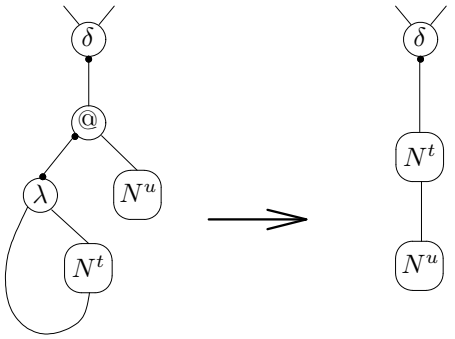


Fig. 3. An attempt to copy a β-redex, forcing evaluation

Garbage collection with Interaction Nets.

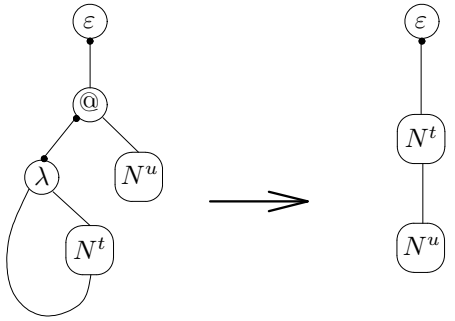


Fig. 4. Erasing a β-redex

In the λ-calculus garbage collection occurs when a bound variable is substituted in a term where it does not occur; this non-occurrence is encoded by an erasing agent (ε). Similarly to duplication, erasing a redex implies first reducing it (see figure 4).

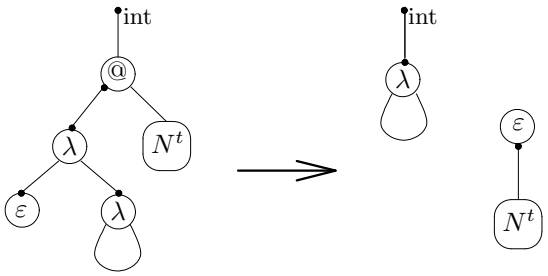


Fig. 5. Reduction of $(\lambda xy.y)t$. The term $\lambda y.y$ can be read at the interface after one step of reduction

To take a concrete example, consider $(\lambda xy.y)t \longrightarrow \lambda y.y$ where t is a big term. Figure 5 shows the corresponding net after one step of reduction, where the net for t becomes disconnected (unreachable from the interface). Reducing this net implies reducing (the net representing) t unnecessarily. If t is non-terminating, this will produce a non-terminating interaction net.

Weak Reduction and Interface Normal Forms.

The above examples show that in some situations full reduction is too strong since it forces the reduction of disconnected nets which may not terminate. Fernández and Mackie [2] have proposed *weak reduction to interface normal form* (WRINF) as an alternative strategy. As in functional programming, where reduction usually stops with weak head normal forms, WRINF stops as soon as a principal port reaches the interface of the net, doing in fact the minimum amount of work necessary for that. In particular, no reductions are performed in disconnected components. If the net encodes a λ-term, reduction stops with the principal port of a λ agent at the interface.

This Paper.

The author has proposed an abstract machine for full interaction net reduction [12]. This machine addressed the implementation issues left open by the calculus (notably those concerning concurrency), by decomposing interaction steps into finer-grained operations. The present paper continues that work by proposing a mechanism which implements the WRINF strategy, in the same spirit of implementation mechanisms for the λ-calculus such as the SECD machine or Krivine's machine, that also force specific reduction strategies.

An important issue to consider when λ-terms are encoded as interaction nets is that the nets that become disconnected may be generated by the encoding, rather than intrinsic to the term (as in the previous examples). Moreover, these disconnected nets are potentially dangerous in the sense that their reduction may generate bigger intermediate nets to be garbage-collected. Section 5 contains a major application of weak reduction: a set of experimental results allow to study the nature of the garbage generated by different encodings of the λ-calculus, leading to a better understanding of the encodings (and in particular their time and space behaviour). The results also offer convincing evidence that weak reduction may dramatically improve the behaviour of interaction net-based program evaluators.

2 Background

In this paper we use a simpler form of nets for which the interface consists of a single free port. These are sufficient for representing closed functional expressions (programs), and moreover all the work here can be easily carried over to the general case. Rather than elaborating on the graphical formalism we now turn to the calculus for interaction nets of Fernández and Mackie [2], simplified for the class of nets we consider.

The Calculus for Interaction Nets.

Nets are reduced in the context of an interaction system $\langle \Sigma, \mathcal{R} \rangle$, where Σ is a set of symbols ranged over by $\alpha, \beta, \ldots$, each symbol α with an arity $\mathsf{ar}(\alpha) \in \mathbf{N}$; and $\mathcal{R}$ is a set of interaction rules. Additionally we use a set of

names or variables x, y, z, etc. Terms (ranged over by $t, u, \ldots$) in our language are either names or of the form $\alpha(\boldsymbol{t})$ where $\boldsymbol{t}$ is a sequence of terms $t_1, \ldots, t_n$ and $\mathsf{ar}(\alpha) = n$.

The occurrence of α in $\alpha(\boldsymbol{t})$ is called an *agent* (a node in the graph) and $\mathsf{ar}(\alpha)$ corresponds to the number of its auxiliary ports. A net is represented as a pair $\langle t \mid \Delta \rangle$ consisting of a multiset Δ of unordered pairs of terms (equations) together with a distinguished term t corresponding to the interface of the net. Each variable occurs exactly twice in a net.

Representing Nets in the Calculus.

The following observations explain the relation between the language of the calculus and (graphical) nets:

- A term $t = \alpha(t_1, \ldots, t_n)$ corresponds to a *tree*, with the principal port of α at the root, and each t_i of the form $\beta(u_1, \ldots, u_m)$ a sub-tree (with the principal port of β connected to the ith auxiliary port of α). Variables in t correspond to auxiliary ports which are leaves in the tree.
- The pair of leaves represented by the two occurrences of the same variable (either in the same or in different trees) is connected by an edge.
- Equations of the form $\alpha(\boldsymbol{t}) = \beta(\boldsymbol{u})$ correspond to active pairs.
- The remaining equations (such as $x = t$) are simply edges.

As an example, consider again the net of figure 5. This will be written as $\langle z \mid @(z, u^t) = \lambda(\lambda(y, y), \varepsilon), \Delta^t \rangle$, with $\Sigma = \{@, \lambda\}$. Δ^t is the multiset of equations corresponding to the term t, and u^t its interface. Observe that alternative representations of this net exist, such as the more bureaucratic $\langle z \mid b = \lambda(\lambda(a, a), \varepsilon), @(z, u^t) = b, \Delta^t \rangle$.

We use the notation Δ_1, Δ_2 for $\Delta_1 \cup \Delta_2$ and e, Δ for $\{e\} \cup \Delta$ with e an equation. The set of names $\mathcal{N}(t)$ occurring in a term t is defined in the obvious way and extended to equations and multisets. Substitution is also defined as usual; it will be clear from rules **Indirection** and **Collect** below that substitutions are performed to remove pairs of variables from the net, and as such they can be implemented as assignment (no terms can be erased or duplicated); $t[x := u]$ denotes the term t where x is substituted by u; even though no explicit binding operator is present, we assume α-equivalence of nets for names (denoted by $\equiv$).

An interaction rule $r \in \mathcal{R}$ is written as an equation $t \bowtie u$, where names occur exactly twice. Rules are converted to this format by connecting together corresponding free ports in the left- and right-hand sides; one obtains a net with an active pair that can be written as an equation as explained above. For instance, the rule in figure 1(a) can be written as $@(x, y) \bowtie \lambda(x, y)$ after being redrawn as in figure 1(b).

We now give the three rules of the calculus, allowing to reduce nets:

Interaction $\langle w \mid \alpha(\boldsymbol{t}) = \beta(\boldsymbol{u}), \Delta\rangle \longrightarrow \langle w \mid T, U, \Delta\rangle$, where

$$\begin{aligned}
&\boldsymbol{t} = t_1, \ldots, t_j \text{ and } \boldsymbol{u} = u_1, \ldots, u_k \\
&\alpha(t'_1, \ldots, t'_j) \bowtie \beta(u'_1, \ldots, u'_k) \in \mathcal{R} \\
&T = \{t_1 = t'_1, \ldots, t_j = t'_j\} \text{ and } U = \{u_1 = u'_1, \ldots, u_k = u'_k\}
\end{aligned}$$

Indirection $\langle w \mid x = t, u = v, \Delta\rangle \longrightarrow \langle w \mid u[x := t] = v, \Delta\rangle$ if $x \in \mathcal{N}(u)$

Collect $\langle w \mid x = t, \Delta\rangle \longrightarrow \langle w[x := t] \mid \Delta\rangle$ if $x \in \mathcal{N}(w)$

The first rule of the calculus applies the graphical interaction rules, and the remaining rules handle the bureaucracy introduced by the textual notation.

Remark 2.1 [Variable Convention] In the interaction rule, α-equivalence is used to produce a copy of the rule $\alpha(t'_1, \ldots, t'_j) \bowtie \beta(u'_1, \ldots, u'_k)$ where all variables are fresh.

Properties.

The calculus (as well as the graphical formalism) enjoys strong confluence and uniqueness of normal forms. The latter are either of the form $\langle w \mid \emptyset\rangle$ (called *reduced nets*) or $\langle w \mid x = u\rangle$ where $x \in \mathcal{N}(u)$, corresponding to a *cycle* – a tree together with an edge connecting its root to one of its leaves. Cycles represent *deadlocked* computations; they do not arise in the context of functional programming (or in general whenever typed nets are used).

Example.

Assuming Δ^t empty in our example net, the following reduction sequence would be possible in the interaction system $\langle\{@, \lambda\}, \{@(x, y) \bowtie \lambda(x, y)\}\rangle$:

$$\begin{aligned}
\langle z \mid @(z, u^t) = \lambda(\lambda(y, y), \varepsilon)\rangle &\equiv \langle z \mid @(z, u^t) = \lambda(\lambda(a, a), \varepsilon)\rangle \\
&\longrightarrow \langle z \mid z = x, u^t = y, \lambda(a, a) = x, \varepsilon = y\rangle \\
&\longrightarrow \langle z \mid z = x, \lambda(a, a) = x, \varepsilon = u^t\rangle \\
&\longrightarrow \langle z \mid \lambda(a, a) = z, \varepsilon = u^t\rangle \\
&\longrightarrow \langle \lambda(a, a) \mid \varepsilon = u^t\rangle
\end{aligned}$$

where the last net corresponds to the right-hand side of figure 5. The above reduction sequence corresponds to an application of the **Interaction** rule, followed by two applications of **Indirection** and finally **Collect**. Reduction would then proceed to garbage collect u^t.

3 Interface Normal Forms and Weak Reduction

A net is in *interface normal form* [2] when one of the following is connected to the interface:

- the principal port of an agent, corresponding to a net of the form $\langle \alpha(\boldsymbol{t}) \mid \Delta \rangle$;
- an auxiliary port of an agent which is part of some cycle, corresponding to a net like $\langle z \mid x = \alpha(\boldsymbol{t}), \Delta \rangle$ where $x, z \in \mathcal{N}(\boldsymbol{t})$.

Observe that reduced nets are interface normal forms: if the equation multiset is empty, the interface cannot be a variable (otherwise its other occurrence would have to be in some equation). We shall take interface normal forms as *canonical* (i.e., no further reductions are performed). *Weak reduction* (which we denote by $\longrightarrow_w$) is obtained by restricting reduction as follows:

> *A net $\langle z \mid \Delta \rangle$ can only be rewritten by applying rules which involve the unique equation $(t = u) \in \Delta$ such that $z \in \mathcal{N}(t) \cup \mathcal{N}(u)$.*

The following examples show that weak reduction is not deterministic:

(i) Let $N = \langle z \mid y = t, z = \alpha(y) \rangle$. Then $N \longrightarrow_w \langle \alpha(y) \mid y = t \rangle$ and $N \longrightarrow_w \langle z \mid z = \alpha(t) \rangle$.

(ii) Let $N' = \langle z \mid y = t, \alpha(z) = \beta(y) \rangle$. Then $N' \longrightarrow_w \langle z \mid \alpha(z) = \beta(t) \rangle$ and a second reduction of N' is possible using the **Interaction** rule.

(iii) Let $N'' = \langle z \mid y = t, x = \alpha(z, y), u = \beta(x) \rangle$. Then $N'' \longrightarrow_w \langle z \mid x = \alpha(z, t), u = \beta(x) \rangle$ and $N'' \longrightarrow_w \langle z \mid y = t, u = \beta(\alpha(z, y)) \rangle$.

Let $\Downarrow$ denote weak reduction to canonical form. The first example above shows that weak reduction does not yield unique canonical forms, for we have $N \Downarrow \langle \alpha(y) \mid y = t \rangle$ and $N \Downarrow \langle \alpha(t) \mid \emptyset \rangle$. Although non-determinism is caused by the indirection rule (a bureaucratic artifact of the calculus), it cannot be eliminated by giving lower priority to this rule (see example iii. above).

Remark 3.1 Canonical forms are unique in *graphical* interaction nets, since weak reduction corresponds to always reducing the active pair which is closest to the interface (a deterministic strategy).

Deterministic Calculus for WRINF.

The calculus stands closer to the implementation level than the graphical formalism (nets in the calculus can easily be seen as data-structures). The first step towards the design of an abstract machine is then to force determinism for weak reduction. We do this by introducing the following modified calculus:

Interaction $\langle z \mid \alpha(\boldsymbol{t}) = \beta(\boldsymbol{u}), \Delta \rangle \longrightarrow_w \langle z \mid T, U, \Delta \rangle$ if $z \in \mathcal{N}(\boldsymbol{t}) \cup \mathcal{N}(\boldsymbol{u})$, where

$$\boldsymbol{t} = t_1, \ldots, t_j \text{ and } \boldsymbol{u} = u_1, \ldots, u_k$$

$$\alpha(t'_1, \ldots, t'_j) \bowtie \beta(u'_1, \ldots, u'_k) \in \mathcal{R}$$

$$T = \{t_1 = t'_1, \ldots, t_j = t'_j\} \text{ and } U = \{u_1 = u'_1, \ldots, u_k = u'_k\}$$

Indirection $\langle z \mid x = \alpha(\boldsymbol{t}), u = v, \Delta\rangle \longrightarrow_w \langle z \mid u[x := \alpha(\boldsymbol{t})] = v, \Delta\rangle$ if $z \in \mathcal{N}(\boldsymbol{t})$ and $x \in \mathcal{N}(u)$

Collect $\langle z \mid z = t, \Delta\rangle \longrightarrow_w \langle t \mid \Delta\rangle$

These rules apply only to configurations whose interface is a variable; moreover there are now additional conditions for the first two rules. The key point is that reduction is directed by where the interface variable occurs.

It is immediate to see that there is a unique reduction for each of the previous examples using this calculus. Henceforth the symbol $\longrightarrow_w$ will refer to weak reduction as defined by the calculus.

Properties.

The calculus performs weak reductions only, and clearly a single rule applies to each net: if the variable in the interface occurs directly as a member in some equation, rule **Collect** applies; otherwise it must occur in a term of the form $\alpha(\boldsymbol{t})$ and only one of the rules **Interaction** or **Indirection** will apply. This immediately yields uniqueness of normal forms.

The calculus stops exactly with interface normal forms: if the interface is not a variable or the interface variable occurs in a cycle, no rule applies, and some rule always applies to a net that is not in interface normal form.

Example.

The following is an example reduction in this calculus:

$$\begin{aligned}\langle z \mid b = \lambda(\lambda(a,a),\varepsilon), @(z,u^t) = b\rangle &\longrightarrow_w \langle z \mid @(z,u^t) = \lambda(\lambda(a,a),\varepsilon)\rangle\\ &\longrightarrow_w \langle z \mid z = x, u^t = y, \lambda(a,a) = x, \varepsilon = y\rangle\\ &\longrightarrow_w \langle x \mid u^t = y, \lambda(a,a) = x, \varepsilon = y\rangle\\ &\longrightarrow_w \langle \lambda(a,a) \mid u^t = y, \varepsilon = y\rangle\end{aligned}$$

This reduction sequence is unique and ends with an interface normal form.

4 The Abstract Machine

An efficient algorithm for implementing WRINF can be read from the deterministic calculus. We will express the algorithm as an abstract machine acting on configurations – the data-structures used to represent interaction nets. For any net $\langle z \mid \Delta\rangle$ the multiset of equations Δ will be represented by a sequence; we will enforce the following invariant:

> *The unique equation where the interface variable z occurs always occupies the first position in the sequence L representing Δ.*

In fact L will contain *annotated equations* – pairs (e, n), which we will write simply as $e^{\mathbf{n}}$, where e is an equation and $\mathbf{n}$ is a positive integer, the *address*

of the equation in the sequence. The algorithm (specifically the **Indirection** rule) requires access to equations other than that where the interface variable occurs; the notion of *address* allows for access in constant time. We remark that an address is simply an integer uniquely assigned to an equation, which does not necessarily correspond to the position of the equation in the sequence.

Definition 4.1 A *configuration* is a tuple $\langle t \mid L \mid \phi \mid \mathcal{P} \rangle$, where t is a term and L a finite sequence of annotated equations. Let $\mathsf{Names} = \mathcal{N}(t) \cup \mathcal{N}(L)$; then $\phi : \mathsf{Names} \longrightarrow \mathsf{Names}$ is an involutive fixpoint-free function (i.e, $\phi(x) = y$ implies $y \neq x$ and $\phi(y) = x$), and the partial map $\mathcal{P} : \mathsf{Names} \longrightarrow \mathbf{N}$ assigns addresses to names. Each $x \in \mathsf{Names}$ occurs once in the configuration (either in t or in L).

The two occurrences of each name x in a net are replaced by a pair of names x_1, x_2 such that $\phi(x_1) = x_2$, following the abstract machine for full reduction [12]. This allows for nets to be represented as sets of *trees*, together with the additional linking information in ϕ. This splitting of names also applies to interaction rules.

The invariant on a configuration $\langle z \mid L \mid \phi \mid \mathcal{P} \rangle$ can be restated as follows: the variable $\phi(z)$ always occurs in the head equation in L (see lemma 4.5). $\mathcal{P}$ associates to each variable the address of the equation where it occurs, which will be essential for preserving the invariant.

Notation and Conventions.

Interaction rules are written as $t \overset{\Phi}{\bowtie} u$ where Φ is the involution collecting the linking information for the split variables. We write $\mathcal{P}[x \mapsto m]$ for $\mathcal{P} \cup \{(x, m)\}$ and $\phi[x \leftrightarrow y]$ for $\phi \cup \{(x, y), (y, x)\}$. The symbol ',' is used as a separator in sequences; the same symbol is overloaded to represent sequence concatenation. $\mathcal{P}_{\setminus [x_1, \ldots, x_n]}$ denotes the function obtained by excluding $x_1, \ldots, x_n$ from the domain of $\mathcal{P}$. The operator $\mathrm{tl}(\cdot)$ returns the tail of a sequence. In the presentation of the abstract machine, we assume that pairs of terms are unordered, i.e., if necessary members are swapped before a rule is applied. The variable convention applies as in section 2.

The abstract machine rules used for rewriting configurations are given in table 1. We make here some key observations. First of all, we observe that the rule to apply to a configuration is given by the form of the head equation in the list (where $\phi(z)$ occurs).

- The **AgAg** rule performs an interaction, since the equation where $\phi(z)$ occurs is an active pair. This rule adds to the involution in the current configuration the Φ information in the interaction rule.
- The **VarVar** rule handles the case where the head equation has the form $x = y$, where $\phi(y)$ is the interface variable z and $\phi(x)$ occurs in some equation whose address (given by $\mathcal{P}$) is m. The rule manipulates the involution changing $\phi(z)$; consequently the equation with address m must be moved

AgAg $\langle z \mid \alpha(t_1,\ldots,t_j) = \beta(u_1,\ldots,u_k)^{\mathbf{n}}, S \mid \phi \mid \mathcal{P}\rangle$
$\longrightarrow \langle z \mid \hat{L}, S \mid \phi \cup \Phi \mid \mathcal{P} \setminus \mathcal{P}^0 \cup \hat{\mathcal{P}}\rangle$
where $\alpha(t'_1,\ldots,t'_j) \overset{\Phi}{\bowtie} \beta(u'_1,\ldots,u'_k) \in \mathcal{R}$, $\phi(z)$ occurs in t_i $(1 \le i \le j)$, and

$$\hat{L} = t_i = t_i'^{\mathbf{n+k+j-1}}, t_j = t_j'^{\mathbf{n+k+j-2}}, \ldots, t_{i+1} = t_{i+1}'^{\mathbf{n+k+i-1}}, t_{i-1} = t_{i-1}'^{\mathbf{n+k+i-2}},$$
$$\ldots, t_1 = t_1'^{\mathbf{n+k}}, u_k = u_k'^{\mathbf{n+k-1}}, \ldots, u_1 = u_1'^{\mathbf{n}}$$
$$\mathcal{P}^0 = \{x \mapsto n \mid x \in \mathcal{N}(t_1) \cup \cdots \cup \mathcal{N}(t_j) \cup \mathcal{N}(u_1) \cup \cdots \cup \mathcal{N}(u_k)\}$$
$$\hat{\mathcal{P}} = \{x \mapsto n \mid x \in \mathcal{N}(e), e^{\mathbf{n}} \in \mathrm{tl}(\hat{L})\}$$

VarVar $\langle z \mid x = y^{\mathbf{n}}, S_1, e^{\mathbf{m}}, S_2 \mid \phi[x \leftrightarrow x', y \leftrightarrow z] \mid \mathcal{P}[x' \mapsto m]\rangle$
$\longrightarrow \langle z \mid e^{\mathbf{n}}, S_1, S_2 \mid \phi[x' \leftrightarrow z] \mid \mathcal{P}_{\setminus[x,y]}\rangle$

VarAg $\langle z \mid x = \alpha(\boldsymbol{t})^{\mathbf{n}}, S_1, e^{\mathbf{m}}, S_2 \mid \phi[x \leftrightarrow x'] \mid \mathcal{P}[x' \mapsto m]\rangle$
$\longrightarrow \langle z \mid e[x' := \alpha(\boldsymbol{t})]^{\mathbf{n}}, S_1, S_2 \mid \phi \mid \mathcal{P}_{\setminus[x,x']}\rangle$ if $x' \neq z$

Interface $\langle z \mid x = \alpha(\boldsymbol{t})^{\mathbf{n}}, S \mid \phi[x \leftrightarrow z] \mid \mathcal{P}\rangle \longrightarrow \langle \alpha(\boldsymbol{t}) \mid S \mid \phi \mid \mathcal{P}_{\setminus[x,z]}\rangle$

Table 1
Abstract Machine Rules

to the head of the list (to preserve the invariant).

- **VarAg** corresponds to the remaining case of the Indirection rule of the calculus. Observe that $\phi(z)$ occurs in $\alpha(\boldsymbol{t})$, which will substitute some variable in another equation, that must thus be moved to the head of the list.
- **Interface** is a restricted version of the Collect rule of the calculus.

With respect to how the $\mathcal{P}$ component is updated and used, we remark that it is not necessary for $\mathcal{P}$ to store the addresses of variables occurring in the first equation in the list. $\mathcal{P}$ will not be consulted for variables in this equation (since we assume cycles do not arise) and it may even contain incorrect addresses for these variables. For this reason, rules **VarVar** and **VarAg** need not update in $\mathcal{P}$ the addresses of variables that are moved to the head of the list. The **AgAg** rule on the other hand must update information in $\mathcal{P}$ corresponding to all the equations (but the first) added to the configuration.

We now show how an initial configuration is constructed:

Definition 4.2 From any (non-canonical) interaction net $N = \langle z \mid \Delta\rangle$ one obtains a corresponding *initial configuration* $\mathcal{C}_{(N)} = \langle z \mid L \mid \phi \mid \mathcal{P}\rangle$ by replacing (specializing) the two occurrences of each variable x by x_1 and x_2 in z and Δ; then $\phi \doteq \{(x_1,x_2),(x_2,x_1) \mid x \in \mathcal{N}(\Delta)\}$; let $e_1, e_2, e_3, \ldots e_n$ be any enumeration of the elements of Δ, then $L \doteq e_i^{\mathbf{n}}, e_n^{\mathbf{n-1}}, \ldots, e_{i+1}^{\mathbf{i}}, e_{i-1}^{\mathbf{i-1}}, \ldots, e_1^{\mathbf{1}}$ where $\phi(z) \in \mathcal{N}(e_i)$; $\mathcal{P} \doteq \{(x,n) \mid x \in \mathcal{N}(e), e^{\mathbf{n}} \in \mathrm{tl}\, L\}$.

Definition 4.3 The *interpretation* of a configuration $\mathcal{C} = \langle t \mid L \mid \phi \mid \mathcal{P} \rangle$ is an interaction net $[\![\mathcal{C}]\!] \doteq \langle t' \mid \Delta \rangle$ where t', Δ are obtained by replacing in t and L all pairs of variables x, y such that $\phi(x) = y$ by a single (fresh) name $v_{x,y}$.

The interpretation of a configuration allows to read back a net from it. The following lemma is straightforward to prove:

Lemma 4.4 *Let N be a net; then $[\![\mathcal{C}_{(N)}]\!] \equiv N$.*

We now give a series of results concerning properties of the abstract machine. The reader is referred to the full version of this paper for the proofs of these results.

Lemma 4.5 *Let $\mathcal{C}_0$ be an initial configuration and $\mathcal{C}_0 \longrightarrow^* \mathcal{C} = \langle z \mid e_0^{\mathbf{n}}, L \mid \phi \mid \mathcal{P} \rangle$; then*

(i) *For $m \neq n$, $\mathcal{P}(x) = m$ iff $x \in \mathcal{N}(e)$ with $e^{\mathbf{m}} \in L$; moreover if $e'^{\mathbf{m}} \in L$ then $e' = e$.*

(ii) *$\phi(z) \in \mathcal{N}(e_0)$;*

Lemma 4.6 *Let $\mathcal{C}_0$, $\mathcal{C}$, $\mathcal{C}'$ be configurations such that $\mathcal{C}_0$ is initial, $\mathcal{C}_0 \longrightarrow^* \mathcal{C}$, and $\mathcal{C} \longrightarrow \mathcal{C}'$; then $[\![\mathcal{C}]\!] \longrightarrow_w N$ and $N \equiv [\![\mathcal{C}']\!]$.*

Lemma 4.7 *Let N, N' be nets such that $N \longrightarrow_w N'$, and $\mathcal{C}$ a configuration such that $[\![\mathcal{C}]\!] \equiv N$ and $\mathcal{C}_0 \longrightarrow^* \mathcal{C}$ for some initial configuration $\mathcal{C}_0$; then $\mathcal{C} \longrightarrow \mathcal{C}'$ and $[\![\mathcal{C}']\!] \equiv N'$.*

The following correctness result follows in a straightforward way from lemmas 4.4, 4.6, and 4.7.

Proposition 4.8 (Correctness) *Let N_0 be an interaction net; then*

$$\mathcal{C}_{(N_0)} \longrightarrow^* \mathcal{C} \quad \textit{iff} \quad N_0 \longrightarrow_w^* N$$

and $N \equiv [\![\mathcal{C}]\!]$. Here $\longrightarrow^*$ denotes the reflexive and transitive closure of a reduction relation (modulo equivalence of nets in the case of net reduction).

Example.

We take the example interaction net used in section 3:

$$N = \langle z \mid b = \lambda(\lambda(a, a), \varepsilon), @(z, u^t) = b \rangle$$

An initial configuration for this net is:

$$\mathcal{C}_{(N)} = \langle z_1 \mid @(z_2, u^t) = b1^{\mathbf{2}}, b_2 = \lambda(\lambda(a_1, a_2), \varepsilon)^{\mathbf{1}} \mid a_1 \leftrightarrow a_2, b_1 \leftrightarrow b_2, z_1 \leftrightarrow z_2 \mid a_1 \mapsto 1, a_2 \mapsto 1, b_2 \mapsto 1 \rangle$$

and the following is the interaction rule for $@ \bowtie \lambda$, after variables have been split:

$$@(x_1, y_1) \overset{\Phi}{\bowtie} \lambda(x_2, y_2), \text{ with } \Phi = \{x_1 \leftrightarrow x_2, y_1 \leftrightarrow y_2\}$$

The unique reduction sequence for this initial configuration is:

$$\begin{aligned}
\mathcal{C}_{(N)} \longrightarrow\ & \langle z_1 \mid @(z_2, u^t) = \lambda(\lambda(a_1, a_2), \varepsilon)^{\mathbf{2}} \mid \\
& a_1 \leftrightarrow a_2, z_1 \leftrightarrow z_2 \mid \\
& a_1 \mapsto 1, a_2 \mapsto 1\rangle \\
\longrightarrow\ & \langle z_1 \mid z_2 = x_1^{\mathbf{5}}, u^t = y_1^{\mathbf{4}}, \lambda(a_1, a_2) = x_2^{\mathbf{3}}, \varepsilon = y_2^{\mathbf{2}} \mid \\
& a_1 \leftrightarrow a_2, z_1 \leftrightarrow z_2, x_1 \leftrightarrow x_2, y_1 \leftrightarrow y_2 \mid \\
& a_1 \mapsto 3, a_2 \mapsto 3, x_2 \mapsto 3, y_1 \mapsto 4, y_2 \mapsto 2\rangle \\
\longrightarrow\ & \langle z_1 \mid \lambda(a_1, a_2) = x_2^{\mathbf{5}}, u^t = y_1^{\mathbf{4}}, \varepsilon = y_2^{\mathbf{2}} \mid \\
& a_1 \leftrightarrow a_2, z_1 \leftrightarrow x_2, y_1 \leftrightarrow y_2 \mid \\
& a_1 \mapsto 3, a_2 \mapsto 3, x_2 \mapsto 3, y_1 \mapsto 4, y_2 \mapsto 2\rangle \\
\longrightarrow\ & \langle \lambda(a_1, a_2) \mid u^t = y_1^{\mathbf{4}}, \varepsilon = y_2^{\mathbf{2}} \mid \\
& a_1 \leftrightarrow a_2, y_1 \leftrightarrow y_2 \mid \\
& a_1 \mapsto 3, a_2 \mapsto 3, y_1 \mapsto 4, y_2 \mapsto 2\rangle
\end{aligned}$$

Where rules **VarAg**, **AgAg**, **VarVar**, and **Interface** were used, in this order. The last net is a normal form of the abstract machine and has as interpretation the following net:

$$\overline{N} = \langle \lambda(a, a) \mid u^t = y, \varepsilon = y\rangle$$

We recall that throughout reduction, the map $\mathcal{P}$ may contain "wrong" addresses for variables occurring in the head of the list.

Implementation Issues.

Consider a configuration $\langle t \mid L \mid \phi \mid \mathcal{P}\rangle$. The sequence of equations L has an obvious linked implementation, where each node is a pair of trees corresponding to terms with uniquely occurring variables (the leaves in the trees). t is yet one such term. The involution ϕ is kept locally as pointers between pairs of variables (leaves in trees). The $\mathcal{P}$ component is also implemented locally by pointers (stored in variables) to nodes in the linked list.

As an example we explain how the **VarAg** rule is implemented: the ϕ pointer is stored as a field in the structure representing variable x, which contains the address of the structure representing x'; the $\mathcal{P}$ pointer is stored as a field in x', and contains the address $\mathbf{m}$ of a node in the linked list of equations; simple pointer operations allow to (i) move the node with address $\mathbf{m}$ to the head of the list and (ii) substitute the term $\alpha(\boldsymbol{t})$ for x' in the equation stored in this node. Since no searching is required, all these operations are done in constant time.

The **AgAg** rule is computationally heavier than its equivalent in the full-reduction version of the abstract machine. In particular, the work involved

in building the $\hat{\mathcal{P}}$ mapping represents the overhead of implementing weak reduction. This can be done in linear time in the number of variables occurring in the head equation in the configuration.

5 Application to Encodings of the λ-calculus

In the optimal reduction systems [5,8] the number of graphical rewrite steps which actually correspond to β-reductions is optimal in the sense of Lévy [9]; however many other steps are needed to support that optimality – the overhead is quite significant. Asperti and colleagues have addressed this problem by introducing rewrite rules which do not fit in the strict definition of interaction but can dramatically improve performance.

Other systems have been proposed that give up optimality, opting instead to reduce the total number of interactions performed – since interaction steps are performed in constant time, this is a measure of the actual cost of evaluating an expression. The following two encodings follow the latter approach; both result from work on translations of Linear Logic [4] proofs into INs:

Yale Encoding [10] This encoding forces *closed reduction* of the encoded terms; it captures locally the notion of an exponential box (from Linear Logic proof nets) by means of boundary agents which allow substitutions to be carried over inside abstractions. The encoding requires garbage collection of the boundary agents when a box is opened.

Combinators Encoding [11] This was motivated by the work of Lafont [7], who introduced a set of agents capable of simulating every interaction net system. This is the simplest encoding (the combinators interaction system consists of 3 agents and 6 rules) but surprisingly it results in a high level of sharing – the number of β-reductions is lower than for Yale. The drawback is that the total number of interactions is in general very high, and for this reason the combinators system has not been considered to be useful in practice. One of the three agents in this system is precisely an eraser agent, which can easily generate disconnected nets.

This section contains a direct application of weak reduction to these two encodings. We saw in section 1 that garbage collection is triggered in λ-terms containing non-occurring bound variables. We now turn to a different sort of garbage: our goal here is to study the nature of the disconnected nets generated by the *dynamics of the encoding*. The process of garbage collecting these nets may be harmful in the sense that it may generate bigger intermediate nets – a situation which would clearly benefit from weak reduction.

This issue has been studied in BOHM, where the encoding of terms generates garbage which should be collected as soon as possible [1]. This not only prevents a space explosion but it also affects efficiency. The data given below allows for a better understanding of how garbage collection should be handled in the Yale and Combinators encodings, and of whether the encodings behave significantly better when WRINF is used rather than full reduction.

We give in table 2 experimental results concerning the reduction of two sequences of λ-terms: **N2II** and **N22II**, for **N** between **1** and **10**, where the numbers correspond to Church numerals. These sequences generate computations which greatly benefit from the sharing allowed by interaction nets.

The results are given for full reduction and for WRINF implemented by the abstract machine of section 4. We remark that for these terms both strategies result in full normal forms (the term $\lambda x.x$ can be read from the interface). The differences between the two strategies concern reduction in the disconnected parts of the nets only.

For each term, we give the total number of interactions obtained with each strategy; the number of interactions which correspond to β-reductions steps; and finally (under the heading "Space"), the number of cells contained in the disconnected components of the nets when WRINF is used. Observe that the machine used here does not perform garbage collection during reduction; the blocked disconnected nets are simply erased once an interface normal form has been reached. Their size is thus a measure of the work involved in this post-reduction garbage collection.

	Yale				Comb			
	Full Red.	WRINF	Space	β-steps	Full Red.	WRINF	Space	β-steps
22II	43	27	21	9	253	42	65	9
32II	67	44	28	12	559	70	90	12
42II	91	61	35	15	1121	98	115	15
52II	115	78	42	18	2195	126	140	18
'10'2II	235	163	77	33	65933	266	265	33
222II	127	88	46	20	1269	149	145	18
322II	383	280	110	51	17031	319	270	29
422II	4395	3292	1110	550	4195705	625	507	48
522II	1051207	788464	262750	131369		1203	968	83
'10'22II						35101	28809	2082

Table 2
Experimental Results

We make the following observations:

(i) The two sequences differ in that (for both encodings) the number of β steps is linear in **N** for **N2II**, and exponential in **N** for **N22II**.

(ii) In the latter case, the exponential growth in the number of β steps is much faster for the Yale encoding than for the Combinators encoding.

(iii) For the Yale encoding the total number of interactions performed is linear (for both sequences) in the number of β steps; the effect of weak reduction is to lower the linearity constant.

(iv) For the Combinators encoding the total number of interactions using full reduction is exponential in **N** for both sequences; the effect of weak reduction is to restore its linearity.

With respect to the size of the disconnected nets when WRINF is used, we remark that for Yale the size of the disconnected nets is equal to the difference between the number of steps obtained using full reduction and weak reduction (minus a constant corresponding to the number of eraser agents in the disconnected net). This means that reduction of the disconnected nets does not generate more garbage: exactly the same work must be done cleaning up the disconnected nets (after WRINF) as if full reduction is used. We conclude that weak reduction is not useful with respect to the garbage generated by Yale (although it *would* still be useful for terms generating intrinsic garbage).

In the combinators system weak reduction brings a major revelation: most of the work involved in (fully) reducing a term is in fact useless for the purpose of evaluation. The size of the disconnected nets is linear in the number of β-reductions, while the work saved is exponential; the disconnected nets contain many potential interactions which are blocked by weak reduction.

Finally, we remark that when weak reduction is used, the Combinators encoding behaves asymptotically better than Yale, since both the number of interactions and the size of the disconnected net are linear in the number of β-steps, which is asymptotically lower than for Yale. In fact the Combinators encoding seems to behave even better than BOHM for the second sequence (even though the optimal number of β steps is linear in $\mathbf{N}$, the total work needed to support that optimality is asymptotically higher than for the Combinators encoding).

6 Conclusions and Further Work

We have proposed a simple device for implementing weak reduction of interaction nets, thus correcting an operational problem of interaction net-based implementations of functional languages, which arises when disconnected nets are generated. In this context, the present work shows that the interaction combinators are an excellent choice for encoding λ-terms as nets, with respect to both the number of β-reduction steps and the total number of interactions.

Weak reduction certainly reveals something about the dynamics of the Yale and Combinators encodings; however, no theoretical results exist allowing to compare these with the optimal reducer (specifically, concerning the global work required to support the underlying strategies).

Our experimental results indicate that the size of the disconnected nets may be quite significant; in particular for the sequence **N22II** this size grows exponentially with $\mathbf{N}$, which may cause a space explosion of the evaluator program. The solution is of course to somehow erase the disconnected nets as they are generated, without waiting for weak reduction to finish. The examples in section 1 show that this cannot be done without leaving the interaction framework: if an eraser agent ε is to erase an active pair, then it must be capable of interacting with the application agent through one of its auxiliary ports. The full version of this paper includes some preliminary work on how the abstract machine can handle non-interaction rules (inspired

by the BOHM garbage collection rules, which are always executed as soon as possible).

References

[1] A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages*, volume 45 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

[2] M. Fernández and I. Mackie. A calculus for interaction nets. In G. Nadathur, editor, *Proceedings of the International Conference on Principles and Practice of Declarative Programming (PPDP'99)*, number 1702 in Lecture Notes in Computer Science, pages 170–187. Springer-Verlag, September 1999.

[3] M. Fernández, I. Mackie, and J. S. Pinto. A higher-order calculus for graph transformation. In *Proceedings of the International Workshop on Term Graph Rewriting, TERMGRAPH 2002*, volume 72 of *Electronic Notes in Theoretical Computer Science*.

[4] J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[5] G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Proceedings of the 19th ACM Symposium on Principles of Programming Languages (POPL'92)*, pages 15–26. ACM Press, Jan. 1992.

[6] Y. Lafont. Interaction nets. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, Jan. 1990.

[7] Y. Lafont. Interaction combinators. *Information and Computation*, 137(1):69–101, 1997.

[8] J. Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 16–30. ACM Press, Jan. 1990.

[9] J.-J. Lévy. Optimal reductions in the lambda calculus. In J. P. Hindley and J. R. Seldin, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 159–191. Academic Press, 1980.

[10] I. Mackie. YALE: Yet another lambda evaluator based on interaction nets. In *Proceedings of the 3rd ACM SIGPLAN International Conference on Functional Programming (ICFP'98)*, pages 117–128. ACM Press, September 1998.

[11] I. Mackie and J. S. Pinto. Encoding linear logic with interaction combinators. *Information and Computation*, 176(2):153–186, 2002.

[12] J. S. Pinto. Sequential and Concurrent Abstract Machines for Interaction Nets. In J. Tiuryn, editor, *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS)*, number 1784 in Lecture Notes in Computer Science, pages 267–282. Springer-Verlag, 2000.

An Abstract Concept of Optimal Implementation

Zurab Khasidashvili [1]

Logic and Validation Technology
Intel, IDC, Haifa, Israel

John Glauert [2]

School of Computing Sciences, UEA
Norwich NR4 7TJ, UK

Abstract

In previous works, we introduced *Stable Deterministic Residual Structures* (SDRSs), Abstract Reduction Systems with an axiomatized residual relation which model orthogonal term and graph rewriting systems, and *Deterministic Family Structures* (DFSs), which add an axiomatized notion of *redex-family* to capture known *sharing* concepts in the λ-calculus and other orthogonal rewrite systems. In this paper, we introduce and study a concept of *implementation* of DFSs. We show that for any DFS $\mathcal{F}$, its implementation $\mathcal{F}_I$ is a non-duplicating DFSs with zig-zag as the family relation, where zig-zag is simply the symmetric and transitive closure of the residual relation on redexes with histories. Further, we show that sharing is compositional: the sharing in a DFS $\mathcal{F}$ can be decomposed into a weaker sharing $\mathcal{F}'$ (such as zig-zag) and a sharing in the implementation $\mathcal{F}'_I$ of $\mathcal{F}'$ stronger than zig-zag. These results require study of the family relation in non-duplicating SDRSs. We show that zig-zag forms a family-relation in every non-duplicating SDRS, and that it is the only *separable* family relation in such SDRSs.

1 Introduction

In order to achieve optimal evaluation of λ-terms, Lévy introduced a notion of *redex-family* to capture the concept of redexes of the 'same origin'. The hope was that it would be possible to mimic multi-step reductions which contract whole families in a term by reduction of some graph representation, in which every multi-step would be represented by contraction of a single graph redex [Lév78,Lév80]. Such an implementation has indeed been achieved by

[1] Email: Zurab.Khasidashvili@intel.com
[2] Email: J.Glauert@cmp.uea.ac.uk

Lamping and Kathail, reviving interest in optimal graph reduction. Since then, there have been many interesting discoveries around the theory and practice of optimality, and we refer the reader to [AG98] for a detailed treatment of the subject.

Lévy introduced the family concept in three different ways: via a suitable notion of *labelling*, via *extraction*, and by *zig-zag*. He showed that they all yield the same concept, in the λ-calculus. The same holds for all orthogonal Higher-Order Rewriting Systems (HORSs) R if all three family concepts are defined in the refinement of R which decomposes every original R-step into first-order or TRS-step and a number of substitution steps [Oos96]. However, the zig-zag family can be defined directly in R, and this yields a different, weaker, family concept [AL93]. Klop [Klo80], Maranget [Mar91], and others have defined similar yet different labellings for orthogonal (first or higher order) rewrite systems, all yielding a concept of family for these systems.

This variety of family concepts, and development of alternative graph rewriting algorithms for optimal implementation of orthogonal rewriting systems, such as Term Graph Rewriting [KKSV93], Jungle Rewriting [HP91], and many others, inspired by Wadsworth's original work on graph-based implementation of the λ-calculus [Wad71], created the need to develop an abstract notion of family general enough to cover all the existing notions, and refined enough to enable proof of normalization and optimality results. Such structures were introduced by the authors in [GK96] as *Deterministic Family Structures* (DFSs) building on recent developments of abstract reduction systems with an axiomatized residual relation, such as the *Concurrent Transition Systems* (CTSs) of Stark [Sta89] and the *Abstract Reduction Systems* (ARSs) of Gonthier et al [GLM92].

Our DFSs are defined as *Deterministic Residual Structures* (DRSs) with an axiomatized family relation. DRSs, in turn, are Abstract Reduction Systems with an axiomatized residual relation. Despite its highly abstract nature, a counterpart of Berry's *stability* property [Ber79] enabled us to prove the normalization theorem for all DRSs, and not only w.r.t. normal forms, but in general for *regular stable sets* of 'results', such as head-normal forms or Böhm tress [GK96,GK02].

In this paper, we continue the abstract study of optimality theory started in [GK96]. We introduce an abstract concept of *Lévy-implementation*: With a DFS $\mathcal{F}$ we associate a *non-duplicating* DRS $\mathcal{R}_I$, called the implementation of $\mathcal{F}$, whose steps exactly correspond to complete family-reduction steps in $\mathcal{F}$, thus, for example, they model sharing-graph implementation of Lévy's complete family-reductions in the λ-calculus. It is not difficult to show that needed reductions in $\mathcal{R}_I$ (w.r.t. any stable set of results) correspond exactly to needed complete family-reductions in $\mathcal{F}$, implying that the former indeed implement optimal computations in $\mathcal{F}$ in the sense of Lévy [Lév78,Lév80].

Further, we show that the family relation on $\mathcal{R}_I$ induced by that of $\mathcal{F}$ coincides with zig-zag, which is the weakest family sharing. At first sight, one might think that there is no need or possibility of a stronger sharing

in non-duplicating SDRSs. However, this is not the case as the example of Asperti and Laneve [AL93] demonstrating 'inadequacy' of Lévy's extraction algorithm for Interaction Systems is, when considered as an Abstract Reduction System, a non-duplicating SDRS. This example also shows that zig-zag does not coincide with the labelling and their extraction families based on the *shift* operation. These effects are caused by the fact that more than one members of a family can be created by a single step – the reduction step $t = \mu(\lambda x.(xx)) \to (\mu(\lambda x.(xx))\,\mu(\lambda x.(xx))) = s$, according to the μ-rule $\mu(\lambda x.X) \to [\mu(\lambda x.X)/x]X$, simultaneously creates the two μ-redexes in s; these redexes are intuitively in the same family, but cannot be related by zig-zag, nor by an extraction procedure similar to Lévy's. We call such families *non-separable.*

Actually, we show that the sharing concept formalized in DFSs is *compositional*: any sharing can be decomposed into a weaker sharing (such as zig-zag, when the latter forms a family-relation) and a non-separable sharing in the non-duplicating DRS implementing that weaker sharing. To this end, we investigate the family relation in non-duplicating DFSs. We show that zig-zag forms a family-relation in every non-duplicating SDRS, and that it is the only separable family relation in such SDRSs. These results are obtained by defining an abstract extraction procedure for non-duplicating SDRSs, and showing that zig-zag coincides with our extraction-family relation.

The paper is organized as follows. In the next section, we recall DRSs and DFSs, and some necessary results concerning normalization and standardization in these frameworks. In Section 3, we introduce zig-zag and separable affine DFSs. Section 4 gives a characterization of zig-zag relation via extraction, used in Section 5 to prove that zig-zag is a family relation in every non-duplicating stable DRS. In section 6 we show that a family relation in a non-duplicating SDRS is separable iff it is zig-zag. In Section 7 we define and study implementation DRSs, and show the compositionality of sharing. Conclusions appear in Section 8.

2 Deterministic Residual and Family Structures

In this section, for those unfamiliar with the earlier work, we recall *Deterministic Residual Structures* (DRSs) and *Deterministic Family Structures* (DFSs), and give a number of examples. We will also recall some basic theory of standardization in DRSs from [KG96].

2.1 Deterministic Residual Structures

The DRS concept is based on [Lév80,HL91]. DRSs model orthogonal term as well as graph rewrite systems, both first and higher order, and including the λ-calculus and its sharing evaluation models, with the standard Church notion of residual. Closely related models are the CTSs of Stark [Sta89], the ARSs of Gonthier et al. [GLM92], and Axiomatic Rewriting Theory of

Melliès [Mel0X]. See [Ter03] for more information on abstract rewriting (with or without a residual relation). DRSs are a minimal axiomatization of the residual concept enabling one to develop a theory of normalization [GK96] and standardization [KG96] for conflict-free reduction systems in an abstract manner, and to study their semantics [KG02,KG97a].

Definition 2.1 We define an ARS as a triple $A = (Ter, Red, \rightarrow)$ where Ter is a set of *terms*,[3] ranged over by t, s, o, e; Red is a set of *redexes* (or *redex occurrences*), ranged over by u, v, w; and $\rightarrow : Red \rightarrow (Ter \times Ter)$ is a function such that for any $t \in Ter$ there is only a finite set of $u \in Red$ such that $\rightarrow(u) = (t, s)$, written $t \xrightarrow{u} s$. This set will be known as the redexes of term t, where $u \in t$ denotes that u is a member of the redexes of t and $U \subseteq t$ denotes that U is a subset of the redexes. Note that $\rightarrow$ is a *total* function, so one can identify u with the triple $t \xrightarrow{u} s$. A *reduction* is a sequence $t \xrightarrow{u_1} t_2 \xrightarrow{u_2} \ldots$. Reductions are denoted by P, Q, N. We write $P : t \twoheadrightarrow s$ or $t \xrightarrow{P} s$ if P denotes a reduction (sequence) from t to s. $|P|$ denotes the length of P. $P + Q$ denotes the concatenation of P and Q. We use U, V, W to denote sets of redexes of a term.

Definition 2.2 A DRS is a pair $\mathcal{R} = (A, /)$, where A is an ARS and $/$ is a *residual* relation on redexes relating redexes in the source and target term of every reduction $t \xrightarrow{u} s \in A$, such that for $v \in t$, the set v/u of *residuals of* v *under* u is a set of redexes of s; a redex in s may be a residual of only one redex in t under u, and $u/u = \emptyset$. If v has more than one u-residual, then u *duplicates* v. If $v/u = \emptyset$, then u *erases* v. A redex of s which is not a residual of any $v \in t$ under u is said to be u-*new* or *created* by u. The set u/P of residuals of u under any reduction P is defined by transitivity.

A *development* of $U \subseteq t$ is a reduction $P : t \twoheadrightarrow$ that only contracts residuals of redexes from U; it is *complete* if $U/P = \cup_{u \in U} u/P = \emptyset$. Development of $\emptyset$ is identified with the empty reduction. U will also denote a complete development of $U \subseteq t$. The residual relation satisfies the following two axioms:

• [FD] ([GLM92]) All developments are terminating; all complete developments of $U \subseteq t$ end at the same term; and residuals of a redex $v \in t$ under all complete developments of U are the same.

• [weak acyclicity] ([Sta89]) Let $u, v \in t$, let $u \neq v$, and let $u/v = \emptyset$. Then $v/u \neq \emptyset$.

We call a DRS $\mathcal{R}$ *stable* (SDRS) if:

• [stability] If $u, v \in t$ are different redexes, $t \xrightarrow{u} e$, $t \xrightarrow{v} s$, and u creates a redex $w \in e$, then the redexes in $w/(v/u)$ are not u/v-residuals of redexes of s, i.e., they are created along u/v .

We call an SDRS *non-duplicating* or *affine*, ASDRS, if the residual relation is non-duplicating. Note that, since the only observables of DRSs are redexes,

[3] We use the term 'term' rather than say 'object', but note that our terms may model various objects, such as graphs.

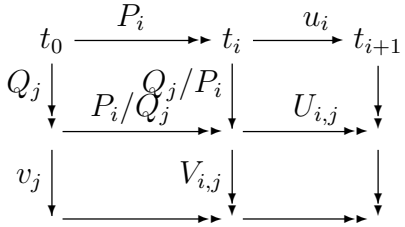
duplicating syntactic rewrite systems may still form affine SDRSs. For example, the DRS corresponding to innermost reductions in an orthogonal TRS is an affine SDRS, although innermost redexes may duplicate their arguments.

Similarly to [HL91,Lév78,Lév80,Sta89], in a DRS $\mathcal{R}$ the residual relation on redexes is extended to all co-initial reductions as follows: $(P_1 + P_2)/Q = P_1/Q + P_2/(Q/P_1)$ and $P/(Q_1+Q_2) = (P/Q_1)/Q_2$, and that *Lévy-equivalence* or *permutation-equivalence* is defined as the smallest relation on co-initial reductions satisfying: $U + V/U \approx_L V + U/V$ and $Q \approx_L Q' \implies P + Q + N \approx_L P + Q' + N$, where U and V are complete developments of redex sets in the same term. Further, one defines $P \trianglelefteq Q$ iff $P/Q = \emptyset$, and can show that $P \approx_L Q$ iff $P \trianglelefteq Q$ and $Q \trianglelefteq P$; and $P \trianglelefteq Q$ iff $Q \approx_L P + N$ for some N. Below, $P \sqcup Q$ will denote $P + Q/P$. Intuitively, $P \approx_L Q$ means that P can be obtained from Q by a number of permutations of adjacent steps, therefore 'Q and P do the same work'; and $P \trianglelefteq Q$ means that P does less work than Q, the difference being Q/P, so $P + Q/P \approx_L Q$. The following *Strong Church-Rosser* property can be proved: for any co-initial finite reductions P, Q, $P \sqcup Q \approx_L Q \sqcup P$.

The stability axiom, and more generally Lemma 2.4 below, states that a redex cannot arise from two 'unrelated' sources. The notion of 'unrelated' is formalized by the notion of *externality*, which expresses the absence of shared (residuals of) redexes.

Definition 2.3 ([GK96]) • Let $u \in U \subseteq t$ and $P : t \twoheadrightarrow o$. We call P *external* to redexes U (resp. u) if P does not contract residuals of redexes in U (resp. residuals of u).

• Let $P : t_0 \overset{P_i}{\twoheadrightarrow} t_i \overset{u_i}{\to} t_{i+1} \twoheadrightarrow t_n$ and $Q : t_0 = s_0 \overset{Q_j}{\twoheadrightarrow} s_j \overset{v_j}{\to} s_{j+1} \twoheadrightarrow s_m$. We call P *external* to reduction Q if for any i, j, $u_i/(Q_j/P_i) \cap v_j/(P_i/Q_j) = \emptyset$ (see the diagram, where $U_{i,j} = u_i/(Q_j/P_i)$ and $V_{i,j} = v_j/(P_i/Q_j)$).



Lemma 2.4 (Stability) *([GK96]) Let $P : t \twoheadrightarrow s$ be external to $Q : t \twoheadrightarrow e$, in a stable DRS, and let P create redexes $W \subseteq s$. Then the residuals $W/(Q/P)$ of redexes in W are created by P/Q, and Q/P is external to W.*

Lemma 2.5 (Weak Acyclicity) *([KG96]) Let P, N be external co-initial finite reductions in a DRS. Then $N \not\approx_L P$.*

2.2 Standardization

In this section, we recall a fragment of results from [KG96], concerning standardization of reductions, relevant to this paper; so we restrict ourselves to affine stable DRSs, ASDRSs, only.

Definition 2.6 • Let $P : t \twoheadrightarrow o$ and $u \in t$, in a DRS. We call u *erased* in P or P-*erased* if $u/P = \emptyset$. We say that P *discards* u if P is external to u and erases it.

• We call u P-*needed* if there is no $Q \approx_L P$ that is external to u, and call it P-*unneeded* otherwise. We call u P-*essential* if there is no $Q \approx_L P$ that discards u, and P-*inessential* otherwise.

We extend these concepts to reductions co-initial with those containing u as a redex of one of its terms.

• Let $Q : t \twoheadrightarrow o$, $P : t \overset{P'}{\twoheadrightarrow} s \twoheadrightarrow e$, and $u \in s$. We say that u is Q-*needed* if u is Q/P'-needed. We call P Q-*needed* if so is every redex contracted in P. We call P *self-needed* if it is P-needed. The other concepts above are extended in the same way.

Note that P-neededness, P-erasure, and P-essentiality do not depend on the choice of a reduction in the class $\langle P \rangle_L$ of reductions Lévy-equivalent to P, since $u/P = u/Q$ when $P \approx_L Q$. The *external* and *discards* concepts however do depend on the particular reduction in the Lévy-equivalence class.

Lemma 2.7 *Let $P : s \twoheadrightarrow t \overset{u}{\to} e \twoheadrightarrow o$ in an ASDRS.*

(1) $w \in t$ is P-needed iff it is P-erased and P-essential.

(2) If $P : t \twoheadrightarrow s' \overset{w}{\to} o$, then $w \in s'$ is P-needed.

(3) If u creates $v \in e$ and u is P-unneeded (resp. P-inessential), then so is v.

(4) If $u \neq v \in t$, then v is P-needed (P-essential) iff v has a P-needed (P-essential) residual in e.

Self-needed reductions play the role of *standard* reductions in SDRSs, since we do not have any nesting relation imposed on redexes, unlike ARSs of [GLM92], and there is no concept of 'left' or 'right' occurrences in DRSs. In the extraction process which we study below (for ASDRSs), self-needed reductions play the same role as outside-in left-to-right standard reductions in the extraction processes of [Lév80,AL93,Oos96].

Definition 2.8 We call a reduction in a DRS *standard* if it is self-needed. We write $P \approx_S Q$ if $P \approx_L Q$ and both P and Q are standard. For any standard P, we define $\langle P \rangle_S = \{Q \mid Q \approx_S P\}$.

The following algorithm is a standardization procedure for reductions in ASDRSs.

Definition 2.9 Let $P : t \twoheadrightarrow s$. The *canonical standard variant of* P, $ST(P)$, is defined as follows: If $P = \emptyset$, then so is $ST(P)$. Otherwise, let $v \in t$ be such that it is P-needed and its residual is contracted in P first among P-needed residuals of P-needed redexes in t (existence of such v follows from Lemma 2.7). Then $ST(P) = v + ST(P/v)$.

Termination of the standardization follows immediately from the fact that $|P/v| \leq |P| - 1$. The following fragment (for ASDRSs) of the standardization theorem from [GK96] implies the correctness of our standardization procedure.

Theorem 2.10 (Standardization) *For any finite reduction P in a stable non-duplicating DRS, $ST(P)$ is a finite standard reduction Lévy-equivalent to P.*

We write $Q \in STV(P)$ if $Q \in STA$ and $Q \approx_L P$, where STA denotes the set of all standard reductions, and call Q a *standard variant* of P. Note that Lemma 2.7 gives an algorithm of construction of a standard variant of any finite reduction $P : t_0 \stackrel{u_0}{\rightarrow} t_1 \stackrel{u_1}{\rightarrow} \ldots \stackrel{u_{n-1}}{\rightarrow} t_n$ in an ASDRS. Indeed, the last step u_{n-1} of P is P-needed by Lemma 2.7.(2). If it is created by u_{n-2}, then the latter is P-needed too, by Lemma 2.7.(3). Otherwise, the ancestor redex of u_{n-1} in t_{n-2} is P-needed by Lemma 2.7.(4). Similarly, we can trace the 'responsible' redex of u_{n-1} in t_0, which is P-needed. Repeated contraction of P-needed redexes terminates, and yields a standard variant of P; this can be shown exactly as the correctness of the standardization algorithm (and is also a corollary of the *Discrete Normalization Theorem* of [KG96]).

In Section 4, we will show that, moreover, all standard variants of a finite reduction P in an ASDRS can be found effectively (there are clearly only a finite number reductions in $STV(P)$ as they all have the same length, which coincides with the number of P-needed steps in P).

2.3 Deterministic Family Structures

We now recall *Deterministic Family structures* (DFSs) which are DRSs where in addition a notion of *redex-family* is axiomatized so that the essence of sharing in the sense of Lévy [Lév78,Lév80] is captured, and all the known family notions (mentioned in the introduction) satisfy these axioms [GK96].

Definition 2.11 A DFS is a triple $\mathcal{F} = (\mathcal{R}, \simeq, \hookrightarrow)$, where $\mathcal{R}$ is a DRS; $\simeq$ is an equivalence relation on redexes with *histories*; and $\hookrightarrow$ is the *contribution* relation on co-initial families, defined as follows:

(1) For any co-initial reductions P and Q, a redex Qv in the final term of Q (read as v *with history* Q) is called a *copy* of a redex Pu, written $Pu \trianglelefteq_z Qv$, if $P \trianglelefteq Q$, i.e., $P{+}Q/P \approx_L Q$, and v is a Q/P-residual of u; the *zig-zag* relation $\simeq_z$ is the symmetric and transitive closure of the copy relation [Lév80]. The *family* relation $\simeq$ is an equivalence relation among redexes with histories containing $\simeq_z$. A *family* is an equivalence class of the family relation; families are ranged over by $\phi, \psi, \ldots$. $Fam(\ )$ denotes the family of its argument.

(2) The relations $\simeq$ and $\hookrightarrow$ satisfy the following axioms:

• [initial] Let $u, v \in t$ and $u \neq v$, in $\mathcal{R}$. Then $Fam(\emptyset_t u) \neq Fam(\emptyset_t v)$, where $\emptyset_t$ is the empty reduction starting from t.

• [contribution] $\phi \hookrightarrow \phi'$ iff for any $Pu \in \phi'$, P contracts at least one redex in ϕ.

• [creation] If $e \stackrel{P}{\twoheadrightarrow} t \stackrel{u}{\rightarrow} s$ and u creates $v \in s$, then $Fam(Pu) \hookrightarrow Fam((P{+}u)v)$.

• [FFD] (*Finite Family Developments*) Any reduction that contracts re-

dexes of a finite number of families is terminating. [4]

Note that the [contribution] can be viewed as a definition of $\hookrightarrow$ rather than as an axiom. Hence sometimes we will consider a DFS as a pair $\mathcal{F} = (\mathcal{R}, \simeq)$.

We will need the following results from [GK96].

Lemma 2.12 *(***Unique Families***) Every family is contracted at most once in a (complete) family-reduction, in a DFS.*

Definition 2.13 ([GK96]) (1) Let $\mathcal{S}$ be a set of terms in a DRS. We call a redex $u \in t$ *$\mathcal{S}$-needed* if at least one residual of it is contracted in any reduction from t to a term in $\mathcal{S}$, and call it *$\mathcal{S}$-unneeded* otherwise. We call a multi-step $U \subseteq t$ $\mathcal{S}$-needed if it contracts at least one $\mathcal{S}$-needed redex.

(2) We call a set $\mathcal{S}$ of terms *stable* if: (a) $\mathcal{S}$ is *closed under reduction*; and (b) $\mathcal{S}$ is *closed under unneeded expansion*: for any $e \xrightarrow{u} o$ such that $e \notin \mathcal{S}$ and $o \in \mathcal{S}$, u is $\mathcal{S}$-needed.

Theorem 2.14 (Relative Optimality [GK96]) *Let $\mathcal{S}$ be a stable set of terms in a DFS $\mathcal{F}$, and let t be an $\mathcal{S}$-normalizable term in $\mathcal{F}$. Then any $\mathcal{S}$-needed (complete) family-reduction Q starting from t is eventually $\mathcal{S}$-normalizing and is $\mathcal{S}$-optimal in the sense that it has a minimal number of family-reduction steps.*

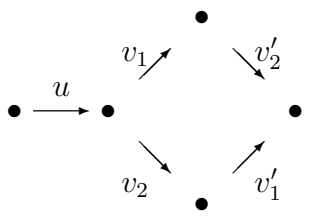
3 Affine DFSs

The following special kinds of DFSs will play an important role in this paper.

Definition 3.1 (1) We call a DFS $\mathcal{F}$ a *zig-zag* DFS, ZDFS, if its family relation is the zig-zag $\simeq_z$.

(2) We call an affine DFS *separable*, SDFS, if, for any redex Pv, v cannot create two different redexes in the same family, that is, if v creates w', w'' and $w' \neq w''$, then $Fam((P+v)w') \neq Fam((P+v)w'')$. In a separable DFS, the family and contribution relations will be denoted by $\simeq_s$ and $\hookrightarrow_s$. The DFS is *non-separable* otherwise.

It is easy to see that not all (even linear, i.e., without duplication and erasure) DFSs are zig-zag DFSs or separable DFSs. For example, in the linear SDRS given by the reduction graph



where u creates v_1 and v_2, $v_1' = v_1/v_2$ and $v_2' = v_2/v_1$, we can set all vs to belong to the same family ϕ, and u to form its own family ψ, and define

[4] This axiom was called [termination] in [GK96].

$\psi \hookrightarrow \phi$. Then we get a DFS which is not a ZDFS as for example $uv_1 \not\simeq_z uv_2$. That DFS is not separable either, as u creates two different members of the family ϕ. We will show below that this is not a coincidence. We could also define $\{v_1, v_1'\}$ and $\{v_2, v_2'\}$ to form separate families ϕ_1 and ϕ_2, and define $\psi \hookrightarrow \phi_1, \phi_2$, and this would yield a ZDFS.

As already mentioned in the introduction, non-separable families (without the name) are studied in [AL93] for Interaction Systems, where it is demonstrated that such a family relation is in general strictly larger than the zig-zag. Another important example where a non-separable sharing is reasonable is the *lazy call-by-value* λ-calculus, λ_{LV} [Plo75]. It is obtained from the λ-calculus by allowing only β-redexes whose arguments are *values* (i.e, variables or abstractions $\lambda x.t$), and that are not in the scope of λ-occurrences (we assume that there are no δ-rules in the calculus). it is easy to see that λ_{LV} is linear: if u, v are redexes in a term t and $u = (\lambda x.e)o$, then $v \notin e$ because of the main λ of u, and $v \notin o$ since o is either a variable or an abstraction; orthogonality of λ_{LV} (i.e., that the residuals of redexes remain admissible) follows from a similar argument. Now consider the reduction

$$t = w = (\lambda x.(xy)(xy))\lambda x.u \xrightarrow{w} o = \overbrace{((\lambda x.u)y)}^{v_1}\overbrace{((\lambda x.u)y)}^{v_2} \twoheadrightarrow uu = s,$$

where u is a λ_{LV}-redex such that x does not occur free in it. It is reasonable to share the two occurrences of u in s, but in order to make the reduction graph of t a DFS (in particular, for the [contribution] axiom to be satisfied), we need to share v_1 and v_2 as well, although the two are not related. This goes beyond Lévy's concept of sharing, and the resulting DFS is not separable as the contraction of w creates two different redexes, v_1 and v_2, of the same family. Note that the occurrences of u in s are *created* by v_1 and v_2, as the occurrences of u in t and o are not λ_{LV}-redexes (they are not admissible).

4 Equivalence of Zig-zag and Extraction

In this section, we introduce an abstract extraction algorithm for ASDRSs and show that zig-zag related redexes (with histories) have the same canonical representatives w.r.t. extraction, up to an equivalence on histories. These canonical representatives are obtained as normal forms of redexes with histories Pv w.r.t. the extraction procedure, which eliminates all steps of histories P that do not 'contribute' to the family of v.

Lévy introduced an extraction procedure for the λ-calculus in [Lév78,Lév80] in order to prove decidability of the family relation. His extraction procedure is effective, and gives canonical representations of families, which are unique, thus implying the decidability of the family relation. For higher order rewrite systems whose reduction steps are more complicated, there are two conceptually different extensions of Lévy's extraction algorithm. The first is due to Asperti and Laneve [AL93], and the second to van Oostrom [Oos96].

The 'problem' arises because a redex can create a number of redexes 'intuitively' in the same family without the help of previous steps, something which cannot happen in the λ-calculus or term rewriting. That these redexes are intuitively in the same family, can be seen after decomposing the rewrite step into two parts – the *TRS part* that only creates new symbols, and the *substitution part*, that performs all (often nested) substitutions. The substitution part can duplicate or erase the redexes created during the TRS part, and all substitution copies of a redex created by the TRS-part are viewed to belong to the same family, as labels of such redexes are the same. Such redexes cannot be related by the zig-zag if one works with the original system [AL93], but can be related if one works in the refinement of the original system [Oos96]. Now, the difference between the two approaches is that Asperti and Laneve decided to accept the inadequacy of the zig-zag, but extended Lévy's extraction algorithm by the *shift* operation which relates all copies of the simultaneously created redexes of the same family to a canonical representative, thus making extraction match the labelling; the resulting family-relation is non-separable. On the contrary, van Oostrom works with the refined rewrite systems and no operation like shift is necessary to ensure coincidence of labelling, extraction and zig-zag families. Since we want to define an extraction procedure adequate for zig-zag, we do not need an operation modelling *shift*. Our results in Section 7 will shed further light on the separability problem.

Definition 4.1 Let $P : t \twoheadrightarrow s$ in an ASDRS, and let $v \in s$. We call Pv *standard* if so is P. We call Pv *canonical* if it is standard and, for any $Q \approx_L P$, the last step in Q creates v.

Note that if $P \approx_S P'$, then Pv is canonical iff so is $P'v$. So canonical forms we speak of are actually objects $\langle P \rangle_S v$, for standard finite reductions P. Our extraction algorithm, defined in Definition 4.4 below, transforms any redex with history into a canonical one, and the main result of this section is that redexes are zig-zag related iff they have the same canonical form. In order for the extraction procedure to be decidable and imply that of the zig-zag relation, we need to establish the decidability of P-neededness for finite reductions P is ASDRSs.

Proposition 4.2 *For any finite reduction P in an ASDRS, P-neededness of redexes in all terms of P is decidable. Consequently, any standard variant of P can be constructed efficiently (in particular, the standardization procedure of P is computable).*

Proof. *Let $P : t_0 \stackrel{u_0}{\to} t_1 \stackrel{u_1}{\to} \ldots \to t_n$. P-(un)neededness of any redex in a term t_i can be established by induction on $n = |P|$, as follows. If $n = 1$, then only the contracted redex u_0 is P-needed in t_0 by Definition 2.6 and Lemma 2.7.(2). Let $n > 1$, and let $P_1 : t_1 \stackrel{u_1}{\to} t_2 \stackrel{u_2}{\to} \ldots \to t_n$. We can assume to have found all the P-needed redexes in all t_i with $i \geq 1$, since P-neededness in these terms coincides with P_1-neededness by Definition 2.6 (and $|P_1| = n - 1$). Then a redex in t_0 different from u_0 is P-needed iff it has such a residual in t_1, by Lemma 2.7.(4). If u_0 does not create u_1, then they can be permuted. If*

$u_1 = u_1'/u_0$ *and* $u_0' = u_0/u_1' = \emptyset$*, then* u_0 *is* P*-unneeded by Definition 2.6; otherwise, if* $u_0' \neq \emptyset$*, by the induction assumption, we can assume to know whether or not* u_0' *is needed w.r.t.* $P' = u_0' + u_2 + \ldots + u_{n-1}$*, and* u_0 *is* P*-needed iff* u_0' *is* P'*-needed. Finally, if* u_0 *creates* u_1*, we can standardize* P_1*, or construct a standard variant* P_1' *of* P_1*; then if* u_0 *still creates the fist step of* P_1' *(which is* P*-needed by Definition 2.6), then* u_0 *is* P*-needed by Lemma 2.7.(3); if not, then we arrive to a previously considered case, and the decidability of* P*-neededness follows. The rest follows from the correctness of the construction of a standard variant of* P *discussed in the previous section.*

□

Lemma 4.3 *Let* $Q : t \xrightarrow{P}\!\!\!\!\rightarrow s \xrightarrow{u} e$*, where* u *does not create* $v \in e$*. Then there is a standard* $Q' \approx_L Q$ *such that* $Q' : t \xrightarrow{P'}\!\!\!\!\rightarrow s' \xrightarrow{u'} e$*, where* $P'u' \trianglelefteq_z Pu$ *and* u' *does not create* v*.*

Proof. *We show that* $ST(Q)$ *can be taken for* Q'*. By Definition 2.9,* $ST(Q)$ *is obtained from* Q *by a sequence of transformations* $Q = Q_1, Q_2, \ldots, Q_n = ST(Q)$ *such that* Q_{i+1} *is obtained from* Q_i *by permuting the first* Q*-needed step that has preceding* Q*-unneeded steps before those* Q*-unneeded steps (all* Q_i *are Lévy-equivalent). Since* u *is the last* Q*-needed step in* Q *by Lemma 2.7.(2), any* Q_i *with* $i < n$ *has the form* $P_i + u$ *such that* $P_i \approx_L P$*, and* P_{n-1} *has the form* $P_{n-1} : t \xrightarrow{P'}\!\!\!\!\rightarrow o \xrightarrow{P''}\!\!\!\!\rightarrow s$ *where* P' *is* Q*-needed and* P'' *is* Q*-unneeded. By Lemma 2.7.(3),* P'' *cannot create* u*, i.e., there is* $u' \in o$ *such that* $u'/P'' = u$*, and* u' *is* Q*-needed by Lemma 2.7.(4). Since* P''/u' *is* Q*-unneeded by Lemma 2.7.(4), and since the last step of* $P' + u' + P''/u'$ *is* Q*-needed by Lemma 2.7.(2),* $P''/u' = \emptyset$*. Since* u' *is* Q*-needed and* P'' *is* Q*-unneeded,* P'' *is external to* u' *by Lemma 2.7.(4). Hence, by the Stability Lemma,* u' *does not create* v*, and the lemma follows since* $ST(Q) = P' + u'$ *is standard by Theorem 2.10, and* $P'u' \trianglelefteq_z Pu$ *since* $u = u'/P''$*.*

$$\begin{array}{ccccccc} t & \xrightarrow{P'}\!\!\!\!\rightarrow & o & \xrightarrow{P''}\!\!\!\!\rightarrow & s & \xrightarrow{u} & e \ni v \\ & & {\scriptstyle u'}\downarrow & & \downarrow{\scriptstyle u} & & \downarrow{\scriptstyle \emptyset} \\ & & v \in e & \xrightarrow[\emptyset]{}\!\!\!\!\rightarrow & e \ni v & \xrightarrow[\emptyset]{}\!\!\!\!\rightarrow & e \ni v \end{array}$$

□

Let $\langle P \rangle_S v$ not be canonical. By Definition 4.1, there is $Q : t \xrightarrow{P'}\!\!\!\!\rightarrow e \xrightarrow{u} s$ such that $Q \approx_L P$ and v is a u-residual of some $v' \in e$. By Lemma 4.3, Q can be chosen standard. In Q, u does not 'contribute' to v, and in the search for a shortest reduction that creates a redex in the zig-zag class of Qv, contraction of u can be omitted – $P'v' \simeq_z Pv$ and $|P'| < |P|$, since all standard Lévy-equivalent reductions have the same (minimal) length [KG96]. Obviously, reductions creating a redex in some family in a quickest way must be standard, since they are the shortest in their Lévy-equivalence classes. The transformation of Pv into $P'v$ is denoted by $Pv \xrightarrow{u}\!\!\!\!\rightharpoonup P'v'$, or just $Pv \rightharpoonup P'v'$. For example, consider the ASDRS corresponding to the TRS $R = \{f(x) \rightarrow$

$g(x)$, $a \to b$, $b \to c\}$, let $P : f(a) \to g(a) \to g(b)$, and let $v = b$ in $g(b)$. Then Pv is not canonical, as $P \approx_S Q : f(a) \to f(b) \to g(b)$ and the last step of Q does not create v – the latter is the residual of $v' = b$ in the final term of $P' : f(a) \to f(b)$. Hence we can perform an extraction step $Pv \rightharpoonup P'v'$. The latter redex is in extraction normal form.

The formal definition of extraction is as follows:

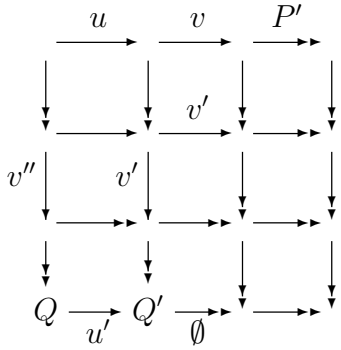
Definition 4.4 (Extraction) Let $Q : t \stackrel{P'}{\twoheadrightarrow} e \stackrel{u}{\to} s$ be a standard variant of P, in an ASDRS, and let $v \in s$ be a u-residual of $v' \in e$. Then we write $Pv \stackrel{u}{\rightharpoonup} P'v'$, and call the transformation an *extraction* step. (Note that, since Q is standard, so is P' by Definition 2.6.) $\rightharpoonup\!\!\!\!\rightharpoonup$ is the transitive and reflexive closure of $\rightharpoonup$.

Since in the above definition $|P'| < |Q| \leq |P|$, the relation $\rightharpoonup$ is trivially strongly normalizing, and in order to prove that it is confluent (modulo $\approx_S$ on histories), it is enough to prove that it is weakly confluent, that is, $Qw'' \stackrel{v}{\leftharpoonup} Nw \stackrel{u}{\rightharpoonup} Pw'$ implies $Qw'' \stackrel{u'}{\rightharpoonup} N^*w^* \stackrel{v'}{\leftharpoonup} Pw'$. We need Lemma 4.9 below to prove this fact. We use the following lemma from [GK96], and three simple new lemmas, in the proof of Lemma 4.9.

Lemma 4.5 *([GK96]) Let $P : t \twoheadrightarrow s$ be external to a set $U \subseteq t$ of redexes, in a DRS, and let $Q : t \twoheadrightarrow o$. Then P/Q is external to the set U/Q.*

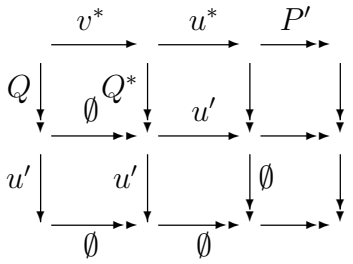
Lemma 4.6 *Let $u + P \approx_S Q + u'$, where $u' = u/Q$ and $P = v + P'$. Then u does not create v, and u can be contracted after v, i.e., $u + v \approx_L v^* + u^*$, where $v = v^*/u$ and $u^* = u/v^*$. Further, $v^*/Q = \emptyset$ and $u' = u^*/(Q/v^*)$.*

Proof. *Let $Q' = Q/u$. Since $u + P$ is standard, so is P by Definition 2.6, so v is P-needed, and since $P \approx_L Q'$, v is Q-needed too, i.e, Q' contracts a residual v' of v. Since Q' contracts residuals of redexes contracted in Q, Q contracts a redex v'' whose residual is v'. So we have the following picture:*



Now, since Q is external to u (since u has a Q-residual u' and the DRS is non-duplicating), we have immediately by the Stability Lemma that both v and v'' are residuals of some redex v^* in the initial term. Hence $u+v \approx_L v^*+u^*$, where $v = v^*/u$ and $u^* = u/v^*$. Since Q contracts a residual v'' of v^*, $v^*/Q = \emptyset$. Since Q is external to u, we have by Lemma 4.5 that $Q^* = Q/v^*$ is external to

u^*. *Since* u *is* $u + P$*-needed, so is* u^* *by Lemma 2.7.(4). Hence* u^* *is* $Q^* + u'$*-needed. Since* Q^* *is external to* u^* *and* $Q^* + u'$ *contracts a residual of* u^**, we have* $u' = u^*/Q^*$.



□

Lemma 4.7 *Let* $P + u \approx_S Q + v$ *and let* $u \neq v$*. Then* $P \not\approx_L Q$.

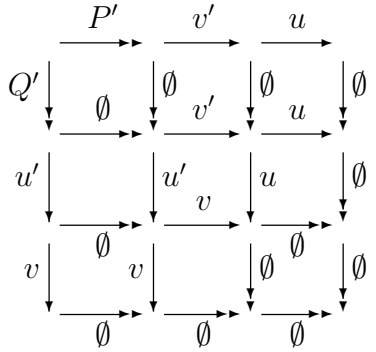
Proof. *Suppose on the contrary that* $P \approx_L Q$*. Then* $P+u \approx_S Q+v$ *iff* $u \approx_L v$*. But, by [weak acyclicity], this is only possible when* $u = v$ *– a contradiction.*□

Lemma 4.8 *Let* $P \approx_S Q$*. Then any non-empty step in Klop's diagram of* P *and* Q *in* P*-needed.*

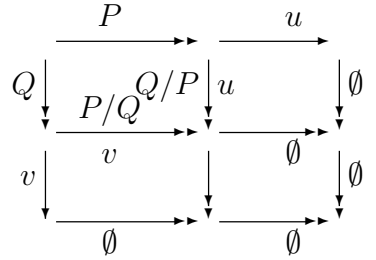
Proof. *Since every step in the diagram is a residual of a redex contracted in* P *or* Q*, the lemma follows immediately from Lemma 2.7.(4).* □

We are now ready to prove Lemma 4.9.

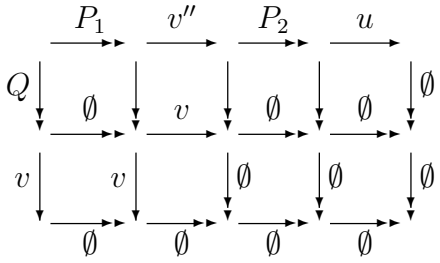
Lemma 4.9 *Let* $P + u \approx_S Q + v$ *and let* $u \neq v$*. Then there are* $P'v'$ *and* $Q'u'$ *such that* $P' + v' \approx_S P$*,* $Q' + u' \approx_S Q$*,* $P' \approx_S Q'$*,* $P'v' \simeq_z Qv$ *and* $Q'u' \simeq_z Pu$*,* $u = u'/v'$ *and* $v = v'/u'$.



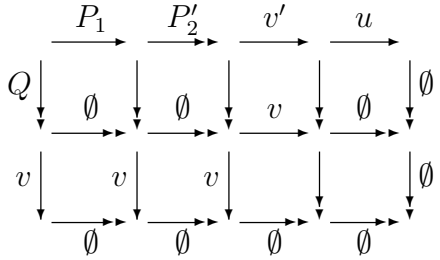
Proof. *Since* $P + u \approx_S Q + v$*, we have* $v \approx_L (P + u)/Q$*. By Lemma 2.5,* $(P + u)/Q$ *contracts a residual of* v*. We show that* $P/Q \neq \emptyset$.



Suppose on the contrary that $P/Q = \emptyset$. Then, by [weak acyclicity], $v = u/(Q/P)$. Further, by Lemma 4.7, $P \not\approx_L Q$, hence $P/Q = \emptyset$ implies $Q/P \neq \emptyset$. But $P + u \approx_L Q + v$ implies $(Q/P)/u = \emptyset$. Since $Q + v$ is standard, the first (and any other) step of Q whose residual, say w, is contracted in Q/P is u-needed by Lemma 4.8. Hence $w/u = \emptyset$ implies $u = w$, and therefore $Q/P = u$ and $u/(Q/P) = \emptyset$, contrary to $v = u/(Q/P)$. So $P/Q \neq \emptyset$. Since P is $P + u$-needed (recall that $P + u$ is standard), so is P/Q, i.e., P/Q is v-needed. Hence, by [weak acyclicity], the first (and the only) step of P/Q coincides with v, i..e., $P/Q = v$. Thus P contracts a redex v'' whose residual is v. So if $P = P_1 + v'' + P_2$, then $(Q + v)/P_1 = Q/P_1 + v$, and we have $v'' + P_2 + u \approx_S Q/P_1 + v$ and $v'' + P_2 \not\approx_L Q/P_1$:

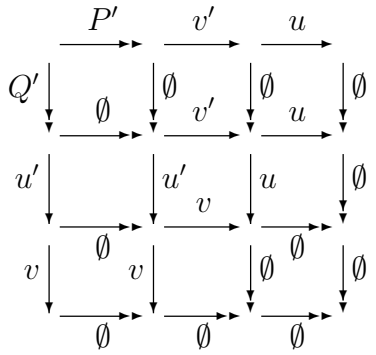


Now, by repeated application of Lemma 4.6, $v'' + P_2 + u$ can be transformed into a reduction $P_2' + v' + u$ such that $v'' + P_2 \approx_S P_2' + v'$, $v' = v''/P_2'$, and $v = v'/(Q/(P_1 + P_2'))$.



Hence, if we take $P' = P_1 + P_2'$, we have that $P' + v' \approx_S P$ and $P'v' \trianglelefteq_z Qv$. Existence of $Q'u'$ such that $Q' + u' \approx_S Q$ and $Q'u' \trianglelefteq_z Pu$ can be shown similarly. Since $P_2/(Q/(P_1 + v'')) = \emptyset$, we have again by repeated application of Lemma 4.6 that $P'/Q \approx_L P'/(Q' + u') \approx_L (P'/Q')/u' = \emptyset$. But P'/Q' is external to u' since u' has a $P/Q' \approx_L P'/Q' + v'/(Q'/P')$-residual (by

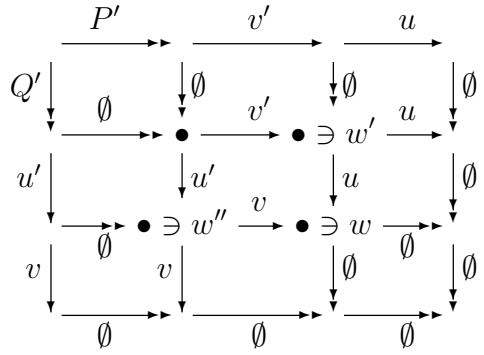
$Q'u' \trianglelefteq_z Pu$).



Hence we have by Lemma 2.5 and Lemma 4.8 that $P'/Q' = \emptyset$. *The converse is proved similarly, so* $P' \approx_L Q'$. *It follows that* $u = u'/v'$ *and* $v = v'/u'$. □

Theorem 4.10 ***(Extraction)*** *Every redex* Pv *in an ASDRS has exactly one extraction normal form* $\langle P^* \rangle_S v^*$ *which is canonical, and* $P^*v^* \trianglelefteq_z Pv$.

Proof. *It is enough to show that the extraction relation* $\rightharpoonup$ *is weakly confluent. Let* $Qw'' \overset{v}{\leftharpoonup} Nw \overset{u}{\rightharpoonup} Pw'$ *with* $u \neq v$ *(since if* $u = v$ *then there is nothing to prove). We will show that* $Qw'' \overset{u'}{\rightharpoonup} N^*w^* \overset{v'}{\leftharpoonup} Pw'$ *for some* N^*w^*, u', *and* v' *such that* $u = u'/v'$ *and* $v = v'/u'$. *By Definition 4.4, we have from* $Qw'' \overset{v}{\leftharpoonup} Nw \overset{u}{\rightharpoonup} Pw'$ *that* $Q + v \approx_S N' \approx_S P + u$, *where* N' *is a standard variant of* N, *and* $w''/v = w'/u = w$. *By Lemma 4.9, we have the following situation, where* $P' + v' \approx_S P$, $Q' + u' \approx_S Q$, $P' \approx_S Q'$, $u = u'/v'$, *and* $v = v'/u'$ *(hence* $P'v' \simeq_z Qv$, $Q'u' \simeq_z Pu$*).*



Now, by [stability], there is a redex w^* *in the final term of* P' *(and* Q'*) such that* $w^*/v' = w'$ *and* $w^*/u' = w''$. *Thus, for* $N^* = P'$, *we have* $Qw'' \overset{u'}{\rightharpoonup} N^*w^* \overset{v'}{\leftharpoonup} Pw'$ *by Definition 4.4.* □

The extraction normal form $\langle P^* \rangle_S v^*$ of Pv is called a *canonical form of* Pv, and so are all $P'v' \in \langle P^* \rangle_S v^*$. Now we can prove the adequacy of our extraction procedure for the zig-zag.

Theorem 4.11 *In an ASDRS,* $Pu \simeq_z Qv$ *iff they have the same unique canonical form* $\langle N \rangle_S w$.

Proof. *By definition of $\simeq_z$, $Pu \simeq_z Qv$ implies existence of $P_0u_0 = Pu$, $P_1u_1, \ldots P_nu_n = Qv$ such that $P_0u_0 \trianglerighteq_z P_1u_1 \trianglelefteq_z P_2u_2 \trianglerighteq_z \ldots P_nu_n$. By the Standardization Theorem, we can take P_i to be standard. Since $P_0u_0 \trianglerighteq_z P_1u_1$, there is Q_1 such that $P_0 \approx_L P_1 + Q_1$ and $u_0 = u_1/Q_1$. Let $P_1'u_1'$ be a canonical form of P_1u_1: $P_1u_1 \twoheadrightarrow P_1'u_1'$. Then there is P_1^* such that $P_1 \approx_S P_1' + P_1^*$. We show that P_1' is $P_1' + P_1^* + Q_1$-needed, i.e., P_0-needed (since $P_1' + P_1^* + Q_1 \approx_L P_0$). Suppose on the contrary that P_1' contracts a P_0-unneeded redex. Let w be the latest P_0-unneeded step in P_1'. By Lemma 2.7.(3), w does not create the next step in P_1' (if w is not the last step in P_1'), therefore can be permuted with its next step. That w-step is again P_1'-unneeded, and can be contracted after its next step, and so on. So we can assume that w is the last step in P_1' (P_1' is chosen up to $\approx_S$). Since u_1' has a residual along $P_1^* + Q_1$, it is P_0-essential by Definition 2.6. Since w is P_0-unneeded, it is P_0-inessential by Lemma 2.7.(1). Hence w does not create u_1' by Lemma 2.7.(3). But this is impossible since $P_1'u_1'$ is canonical and w is the last step of P_1'. So we have proven that P_1' is P_0-needed. This implies that the standardization procedure of Definition 2.9 does not effect P_1' when applied to $P_1' + P_1^* + Q_1$, i.e., we can assume a standard $P_0' \approx_S P_0$ such that $P_0' = P_1' + P_0''$ for some P_0'', and $u_0 = u_1'/P_0''$. Hence $P_0u_0 \twoheadrightarrow P_1'u_1'$ by the definition of $\rightharpoonup$, and $P_1'u_1'$ is a canonical form of both P_0u_0 and P_1u_1. Similarly, since $P_1u_1 \trianglelefteq_z P_2u_2$, we have that $P_1'u_1'$ is a canonical form of P_2u_2, and so on. The theorem now follows from Theorem 4.10.* □

This theorem, together with computability of extraction normal forms, implies decidability of the zig-zag relation in ASDRSs. Note that, at this stage, we do not yet know whether or not zig-zag is a family relation. This is the subject of the next section.

5 Affine Zig-zag Families

In this section we show that, in ASDRSs, the zig-zag relation forms a family relation, that is, it satisfies the family axioms of DFSs.

Below, $FAM_z(P)$ (resp. $SFAM_z(P)$) denotes the set of zig-zag classes whose member (P-needed) redexes are contracted in P, in an ASDRS; and $Fam_z(Qu)$ denotes the zig-zag class of Qu. Further, if ϕ', ϕ are zig-zag classes, we write $\phi' \hookrightarrow_z \phi$ iff for any $Pu \in \phi$, P contracts a redex in ϕ'.

Lemma 5.1 *Let P be Q-needed, in an ASDRS. Then $FAM_z(P) \subseteq FAM_z(Q)$. In particular, if $P \in STV(Q)$, then $FAM_z(P) \subseteq FAM_z(Q)$, and if $P \approx_S Q$, then $FAM_z(P) = FAM_z(Q)$.*

Proof. *Let v be a contracted redex in P, say $P = P' + v + P''$. Then v is Q/P'-needed. Hence $Fam_z(P'v) \in FAM_z(Q/P') \subseteq FAM_z(Q)$, implying the lemma.* □

Lemma 5.2 *If $Pv \simeq_z Qw$, then $v/(Q/P) = w/(P/Q)$. In particular, if $P \approx_S Q$, then $v = w$.*

Proof. *By Theorem 4.11, $Q \approx_L N + Q'$, $P \approx_L N + P'$, $w = u/Q'$ and*

$v = u/P'$, *where* Nu *is a canonical form of* Qw *and* Pv. *Then* $P/Q \approx_L P'/Q'$ *and* $Q/P \approx_L Q'/P'$. *Hence* $w/(Q/P) = w/(Q'/P') = u/(P' \sqcup Q') = u/(Q' \sqcup P') = v/(Q'/P') = v/(Q/P)$. □

Lemma 5.3 *Let* $Q^* : t \xrightarrow{P} s \xrightarrow{u} e$ *and* u *create* $v \in e$. *Then, for any canonical form* $Q'v'$ *of* Q^*v, Q' *contracts a redex zig-zag related to* Pu.

Proof. *We have by Lemma 4.3 that* $Q = ST(Q^*) = P' + u'$, *where* $P \approx_L P' + P''$ *(for some* Q*-unneeded* P''*) and* $u = u'/P''$. *If* Qv *is not a canonical form, by Lemma 4.3 there is an extraction step* $Qv \xrightarrow{w_1} Q_1v_1$ *(i.e.,* $Q \approx_S Q_1 + w_1$ *and* $v = v_1/w_1$*). Since* $Q_1 + w_1 \approx_S Q = P' + u'$, *we have by Lemma 4.9 that* $Q_1 \approx_S P_1 + u_1$ *such that* $P_1u_1 \simeq_z P'u' \simeq_z Pu$. *So we have* $(P' + u')v \xrightarrow{w_1} (P_1 + u_1)v_1$ *such that* $P_1u_1 \simeq_z P'u'$. *Similarly, if* $(P_1 + u_1)v_1$ *is not a canonical form, there is an extraction step* $(P_1 + u_1)v_1 \xrightarrow{w_2} (P_2 + u_2)v_2$ *such that* $P_2u_2 \simeq_z P_1u_1 \simeq_z P'u' \simeq_z Pu$, *and so on. So a canonical form of* Qv *has the form* $(P_m + u_m)v_m$ *such that* $Pu \simeq_z P_mu_m$. *Since, by Theorem 4.10, for any canonical form* $Q'v'$ *of* Qv *(and hence of* Q^*v*),* $Q' \approx_S P_m + u_m$ *and* $v_m = v'$, *it follows by Lemma 5.1 that* Q' *contracts a redex in the family of* Pu. □

Lemma 5.4 *Let* $Pv \xrightarrow{w} P'v'$. *Then* $FAM_z(P') \subseteq FAM_z(P)$.

Proof. *By Definition 4.4,* $Pv \xrightarrow{w} P'v'$ *implies that* $P' + w \in STV(P)$, *and by Lemma 5.1,* $FAM_z(P') \subseteq FAM_z(P' + w) \subseteq FAM_z(P)$. □

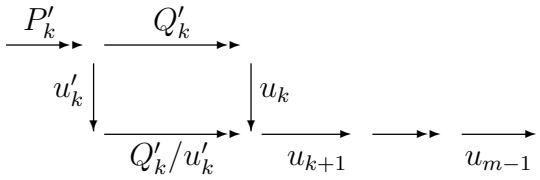
Lemma 5.5 *Let* $Q : e \xrightarrow{P} t \xrightarrow{u} s$ *and let* u *create* $v \in s$. *Then* $Fam_z(Pu) \hookrightarrow_z Fam_z(Qv)$.

Proof. *By Lemma 5.3, if* $Q'v'$ *is a canonical form of* Qv, *then* $Fam_z(Pu) \in FAM_z(Q')$. *Now it follows from Lemmas 5.1 and 5.4 and Theorem 4.11 that for any* $Q^*v^* \simeq_z Qv$, $Fam_z(Pu) \in FAM_z(Q^*)$, *i.e.,* $Fam_z(Pu) \hookrightarrow_z Fam_z(Qv)$. □

Lemma 5.6 *Let* $P : t_0 \xrightarrow{P_i} t_i \xrightarrow{u_i} t_{i+1} \twoheadrightarrow t_n$. *If* $k < m$ *then* $Fam_z(P_ku_k) \neq Fam_z(P_mu_m)$.

Proof. *By induction on the number of zig-zag classes* $\hookrightarrow_z$*-contributing to* $Fam_z(P_ku_k)$. *Suppose on the contrary that* $Fam_z(P_ku_k) = Fam_z(P_mu_m)$. *Let* $P'_ku'_k$ *be a canonical form of* P_ku_k, *which exists by Theorem 4.10, hence there is* Q'_k *such that* $P'_k + Q'_k \approx_L P_k$ *and* $u_k = u'_k/Q'_k$. *So we have that* $P_{k+1} = P_k + u_k \approx_L P'_k + Q'_k + u_k \approx_L P'_k + u'_k + Q'_k/u'_k$. *Since* P'_k *contracts redexes in all contributor zig-zag classes of* $Fam_z(P'_ku'_k) = Fam_z(P_ku_k) = Fam_z(P_mu_m)$, *and since by the induction assumption no redexes in these classes can be contracted again,* u_m *is not created by its preceding step in* $u'_k + Q'_k/u'_k + u_{k+1} + \ldots + u_{m-1}$, *by Lemma 5.5. Similarly, its ancestor redex is not a created redex, and so on. That is,* u_m *is a residual of some redex* u'_m *in the final term of* P'_k, *different from* u'_k. *Hence* $Fam_z(P'_ku'_k) = Fam_z(P_mu_m) = Fam_z(P'_ku'_m)$ *and* $u'_k \neq u'_m$,

which is not possible by Lemma 5.2 – contradiction.



□

Theorem 5.7 *Let $\mathcal{R}$ be a non-duplicating stable DRS. Then $\mathcal{F}_\mathcal{R} = (R, \simeq_z, \hookrightarrow_z)$ is a zig-zag DFS.*

Proof. *We need to show that $\mathcal{F}_\mathcal{R}$ satisfies all family axioms. [contribution] is immediate by the definition of $\hookrightarrow_z$. Since for any $u, v \in t$, $\emptyset_t u$ and $\emptyset_t v$ are canonical forms, $u \neq v$ implies by Theorem 4.11 that $\emptyset_t u \not\simeq_z \emptyset_t v$, i.e., [initial] holds. [creation] is immediate from Lemma 5.5, and [FFD] from Lemma 5.6.*□

6 Affine Separable Families

Based on the results of previous sections, we now show that an affine DFS is a zig-zag DFS iff it is separable. First we establish a characterization of separability of a DFS $\mathcal{F}$ via uniqueness of contracted families in reductions in $\mathcal{F}$. It shows that, in separable DFSs, and only in such DFSs, there is no sharing (in the affine case) – all reductions are in fact *complete family-reductions*. Recall that a complete family-reduction is a multi-step reduction contracting, in each multi-step, all members of a single family in parallel (we will often omit 'complete').

Lemma 6.1 *Let $\mathcal{F} = \mathcal{R}, \simeq, \hookrightarrow)$ be an affine DFS. Them the following are equivalent:*

(1) $\mathcal{F}$ is separable;

(2) Elements of any family are contracted at most once in any reduction in $\mathcal{F}$;

(3) $Pv \simeq Pv'$ implies $v = v'$;

(4) Any reduction in $\mathcal{F}$ is in fact a family-reduction, and vice versa.

Proof. (1) $\Rightarrow$ (2) *Similar to the proof of Lemma 5.6, but replacing $Fam_z(\ )$ by $Fam(\ )$, using [creation] instead of Lemma 5.5, and using separability instead of Lemma 5.2 (see [KG97]).*

(2) $\Rightarrow$ (3) *If there were $Pv \simeq Pv'$ with $v \neq v'$, then at least one of $P+v+v'/v$, $P+v'+v/v'$ would contract two members of $Fam(Pv)$ by [weak acyclicity], contradicting (2).*

(3) $\Rightarrow$ (4) *Immediate.*

(4) $\Rightarrow$ (1) *If $\mathcal{F}$ was not separable, then there would be Pv, w' and w'' such that v creates both w' and w'', $w' \neq w''$, and $Fam((P+v)w') = Fam((P+v)w'')$. By the assumption (4), the reduction $P+v$ is also a family-reduction, implying that $P+v+w' \| w''$, where $w' \| w''$ is the multi-step contracting w'*

and w'' in parallel, is a family-reduction which is not a reduction, contradicting (4). □

Lemma 6.2 *Let Pv be a canonical element of a family ϕ, in an affine DFS $\mathcal{F} = (R, \simeq, \hookrightarrow)$, and let P contract a redex w. Then $Fam(w) \hookrightarrow \phi$.*

Proof. *Suppose on the contrary that $\psi = Fam(w) \not\hookrightarrow \phi$, and assume that Pv and w are such that w is (one of) the latest among steps in canonical elements of ϕ that do not contribute to ϕ. Since Pv is canonical, w cannot be the last step of P as the last step of P creates v by Definition 4.1, and therefore its family contributes to ϕ by [creation]. Further, if v is the next to w step in P, then w cannot create v as this would imply $\psi \hookrightarrow Fam(v)$ by [creation], implying $\psi \hookrightarrow \phi$ (as $Fam(v) \hookrightarrow \phi$ by the choice of w). Hence w can be permuted with v in P, yielding again (by Lemma 2.7) a canonical element of ϕ in which a step whose family does not contribute to ϕ is contracted later than w in P – a contradiction.* □

Theorem 6.3 *An affine DFS $\mathcal{F}$ is separable iff it is a zig-zag DFS.*

Proof. *($\Rightarrow$) Let $\mathcal{F} = (\mathcal{R}, \simeq_s, \hookrightarrow_s)$ be separable, and let $Pu \simeq_s Qv$. (We want to prove that $Pu \simeq_z Qv$.) By the Extraction Theorem, there are canonical forms $P'u' \simeq_z Pu$ and $Q'v' \simeq_z Qv$, and since $\simeq_z \subseteq \simeq_s$ (by the definition of DFSs), $P'u' \simeq_s Q'v'$. Now if $P' \approx_S Q'$, since both $P'u'$ and $Q'v'$ are canonical, the last step of P' creates both u' and v', and $u' = v'$ by separability. Hence, $P'u' \simeq_z Q'v'$, implying $Pu \simeq_z Qv$, in this case. Otherwise, we must have $P' \not\approx_L Q'$ (as $P', Q' \in STA$), and if say $P' \lhd Q'$, $Q' \approx_L P' + P''$ for some $P'' \neq \emptyset$. Again, since $P'u'$ and $Q'v'$ are canonical, the last step of P' creates u', and that of P'', call it w, creates v' (since the last step of P'' coincides with that of $ST(P' + P'')$). By [creation], $Fam(w) \hookrightarrow_s Fam(v') = Fam(u')$, hence by [contribution] P' must also contract a redex in $Fam(w)$, contradicting separability by Lemma 6.1. Hence $P' \not\approx_L Q'$ cannot hold, and we are done. ($\Leftarrow$) Let $\mathcal{F}$ be a ZDFS. By Lemma 5.6, any family is contracted at most once in a reduction in $\mathcal{F}$, implying by Lemma 6.1 that $\mathcal{F}$ is an SDFS.* □

We conclude this section by a useful characterization of histories of canonical elements of zig-zag (hence extraction and separable) families, in ASDRSs.

Theorem 6.4 *Let Pv be a canonical element of a family ϕ, in an AZDFS $\mathcal{F}_{\mathcal{R}} = (R, \simeq_z, \hookrightarrow_z)$. Then P contracts exactly one redex in every contributor family of ϕ.*

Proof. *By Lemma 6.2, P contracts only redexes in contributor families of ϕ. That every such a family is contracted in P follows immediately from [contribution]. The uniqueness of contracted families follows from Lemma 6.1 and Theorem 6.3.* □

7 Implementation DFSs

We now define the *implementation* $\mathcal{F}_I$ of a DFS $\mathcal{F}$, whose reductions correspond to complete family-reductions in $\mathcal{F}$, hence the name. We also show that optimal reductions in $\mathcal{F}$, relative to any stable set $\mathcal{S}$ of results, are implemented in $\mathcal{F}_I$ by the shortest $\mathcal{S}$-normalizing reductions. We will assume that the reduction graph of $\mathcal{F}$ is the reduction graph of an *initial* term, denoted by $\emptyset$, and that families are considered relative to $\emptyset$, i.e, all histories start with $\emptyset$.

Definition 7.1 Let $\mathcal{F} = (\mathcal{R}, \simeq, \hookrightarrow)$ be a DFS. The *implementation* or *Lévy-implementation* of $\mathcal{F}$ is the AZDFS $\mathcal{F}_I = (\mathcal{R}_I, \simeq_I, \hookrightarrow_I)$, where

• the branches of the reduction graph of the underlying ARS A of $\mathcal{R}_I = (A, /)$ are family-reductions starting from $\emptyset$, each edge (i.e., a reduction step) being a multi-step contracting a family of redexes.

• the residual relation / is defined as follows: let U and V be complete sets of redexes in two families, in a term s, and let $U : s \to o$ be the multi-step contracting U. Then V/U is the multi-step $o \to e$ contracting all members of the set V/U.

• the family and contribution relations $\simeq_I, \hookrightarrow_I$ in $\mathcal{F}_I$ are those induced by $\simeq$ and $\hookrightarrow$: let P_I and Q_I be reductions in $\mathcal{R}_I$ corresponding to family-reductions P and Q in $\mathcal{F}$; then $P_I U \simeq_I Q_I V$ iff for any $u \in U, v \in V$, $Pu \simeq Qv$; and $Fam(P_I U) \hookrightarrow_I Fam(Q_I V)$ iff $Fam(Pu) \hookrightarrow Fam(Qv)$.

We need to verify that $\mathcal{F}_I$ in the above definition is indeed an AZDFS.

Lemma 7.2 *Let $P : \emptyset \twoheadrightarrow s$ be a family-reduction in a DFS $\mathcal{F} = (R, \simeq, \hookrightarrow)$, let $U, V \subseteq s$ be complete sets of redexes of families ϕ and ψ in s, respectively, and let $s \overset{V}{\twoheadrightarrow} o$. Then $U' = U/V$ is the complete set of redexes of ϕ in o.*

Proof. *Since $\simeq_z \subseteq \simeq$, U' consists of redexes of ϕ. Suppose on the contrary that there is $w \in o$ such that $(P + V)w \in \phi$ and $w \notin U'$. Again by $\simeq_z \subseteq \simeq$, w must be created along V, and we have by [creation] that $\psi \hookrightarrow \phi$. But this implies by [contribution] that P contracts a member of ψ, and therefore the complete family ψ (since P is a family-reduction), and $P + V$ contracts the family ψ twice, contradicting Lemma 2.12.* □

Thus the residual relation is well defined. Obviously, $V/V = \emptyset$, and every family in o has at most one ancestor family in s. Further, [weak acyclicity] and [stability] for $\mathcal{F}_I$ follow immediately from Acyclicity Lemma and Stability Lemma, respectively (one just needs to take for the reductions P and Q in these lemmas complete developments of disjoint sets of redexes, which are clearly external). The axiom [initial] in $\mathcal{F}_I$ follows immediately from [initial] in $\mathcal{F}$. If $P+U+V$ is a reduction in $\mathcal{F}_I$ such that U creates V, then the redexes in V are created along U (when $P+U+V$ is considered as a reduction in $\mathcal{F}$), i.e., $Fam(U) \hookrightarrow Fam(V)$ in $\mathcal{F}$, hence $Fam(U) \hookrightarrow_I Fam(V)$, implying [creation] in $\mathcal{F}_I$. Since family-reductions can be viewed as reduction in $\mathcal{F}$ (by considering multi-steps as corresponding complete developments), [contribution] for $\hookrightarrow_I$

follows immediately from [contribution] for $\hookrightarrow$. Finally, [FFD] for $\mathcal{F}_I$ follows immediately from Lemma 2.12 for $\mathcal{F}$. Hence $\mathcal{F}_I$ is indeed a DFS as $\simeq_I$ clearly contains the zig-zag relation. Note that $\mathcal{F}_I$ is separable as its steps contract entire $\simeq$-families, hence it is an AZDFS by Theorem 6.3:

Theorem 7.3 *For any DFS $\mathcal{F}$, $\mathcal{F}_I$ is an AZDFS.*

Next we show that any sharing $\simeq$ in an SDRS stronger than zig-zag can be decomposed into any weaker sharing $\simeq'$ and a *non-separable* sharing $\simeq^*$ in the implementation of $\simeq'$.

Definition 7.4 Let $\mathcal{F} = (\mathcal{R}, \simeq)$ and $\mathcal{F}' = (\mathcal{R}, \simeq')$ be DFSs. We say that $\mathcal{F}$ has a *stronger* sharing than $\mathcal{F}'$, written $\mathcal{F} \geq \mathcal{F}'$, if $\simeq' \subseteq \simeq$.

Theorem 7.5 *Let $\mathcal{F} = (\mathcal{R}, \simeq)$ and $\mathcal{F}' = (\mathcal{R}, \simeq')$ be DFSs. Then $\mathcal{F} > \mathcal{F}'$ iff there is a non-separable family relation $\simeq^*$ on the implementation DRS $\mathcal{R}'_I$ of $\mathcal{F}'$ such that $\mathcal{F}^*_I = \mathcal{F}_I$, where $\mathcal{F}^* = (\mathcal{R}'_I, \simeq^*)$.*

Proof. *($\Rightarrow$) Let $\mathcal{F} > \mathcal{F}'$. Define $\simeq^*$ on $\mathcal{R}'_I$ by: $Pv \simeq^* Qu$ iff $P'v' \simeq Q'u'$, where P' and Q' are (any) reductions in $\mathcal{R}$ corresponding to P and Q, and v' and u' are any $\mathcal{R}$-redexes in redex-sets contracted in multi-steps v and u. It is immediate that the definition is correct (since reductions in $\mathcal{F}$ corresponding to a reduction in $\mathcal{R}'_I$ are all sequentializations of a multi-step reduction and are Lévy-equivalent, and since $\mathcal{F} > \mathcal{F}'$). Further, the family axioms for $\simeq^*$ follow from those of $\mathcal{F}$ exactly as they were verified above for $\mathcal{F}_I$ in the place of $\mathcal{F}^*$ (the only difference is that $\mathcal{F}^*$ is a 'partial implementation' of $\mathcal{F}$ while $\mathcal{F}_I$ is the 'complete' or Lévy-implementation). By the definition of $\simeq^*$, family-reductions in $\mathcal{F}^*$ are exactly family-reductions in $\mathcal{F}$, and $\mathcal{F}^*_I = \mathcal{F}_I$ follows since both are AZDFSs by Theorem 7.3. Since $\mathcal{F} > \mathcal{F}'$ and $\mathcal{F}'_I$ is an AZDFS, $\simeq^*$ is strictly larger than zig-zag, hence is non-separable by Theorem 6.3. ($\Leftarrow$) Immediate from Definition 7.4.* □

Thus, in particular, the study of a sharing in an SDRS strictly larger than the zig-zag can be reduced to studying zig-zag (when it is a family-relation) and studying non-separable affine families.

The following lemma relates neededness in a DFSs with neededness in its implementation. Together with Theorem 2.14, it implies that the implementation DRSs $\mathcal{F}_I$ indeed correctly implements family-reductions in DFSs $\mathcal{F}$.

Lemma 7.6 *Let $\mathcal{S}$ be a stable set of terms (see Definition 2.13) in a DFS $\mathcal{F}$ not containing the initial term $t_0 = \emptyset$, let $P : t_0 \overset{U_0}{\twoheadrightarrow} t_1 \overset{U_1}{\twoheadrightarrow} \ldots \twoheadrightarrow t_n$ be an $\mathcal{S}$-normalizing family-reduction in $\mathcal{F}$, and let $P_I : t_0 \overset{u_0}{\to} t_1 \overset{u_1}{\to} \ldots \to t_n$ be its corresponding reduction in $\mathcal{F}_I$. Then P is $\mathcal{S}$-needed iff so is P_I.*

Proof. *($\Rightarrow$) Assume that P is $\mathcal{S}$-needed, and suppose on the contrary that P_I is not $\mathcal{S}$-needed, i.e., u_i is $\mathcal{S}$-unneeded for some i. Then there is an $\mathcal{S}$-normalizing reduction $N : t_i \twoheadrightarrow s$ that is external to u_i. It remains to show that any corresponding family-reduction N' in $\mathcal{F}$ (no matter how the multi-steps are sequentialized) is external to U_i: the latter implies that U_i does not*

contain an $\mathcal{S}$-needed redex, contradicting $\mathcal{S}$-neededness of P. If on the contrary a multi-step W of N' contracts a residual of a redex in U_i, then it follows from Lemma 7.2 that W is the residual of U_i along N' (as both U_i and W are complete sets of redexes of the same family in corresponding terms), implying that the corresponding step of N is a residual of u_i – a contradiction.

($\Leftarrow$) Let P_I be $\mathcal{S}$-needed. Suppose on the contrary that U_i is not $\mathcal{S}$-needed for some i. Let Q be an $\mathcal{S}$-needed $\mathcal{S}$-normalizing family-reduction (which exists by Theorem 2.14). Note that the residual of U_i in any term along Q forms a complete family by Lemma 7.2, and all residuals of redexes in U_i remain $\mathcal{S}$-unneeded by Corollary 3.1 of [GK96] (which states exactly that the residuals of $\mathcal{S}$-unneeded redexes remain $\mathcal{S}$-unneeded). Thus residuals of U_i along Q are redex-sets disjoint from the contracted redex-sets, implying that the corresponding reduction of Q in $\mathcal{F}_I$ is external to u_i, which contradicts $\mathcal{S}$-neededness of u_i. □

Theorem 7.7 ***(Optimal Implementation)*** *Let $\mathcal{S}$ be a stable set of terms in a DFS $\mathcal{F}$. Then optimal (i.e., $\mathcal{S}$-needed) reductions in $\mathcal{F}_I$, w.r.t. $\mathcal{S}$, implement optimal family-reductions in $\mathcal{F}$, w.r.t. $\mathcal{S}$.*

Proof. *$\mathcal{S}$-needed reductions in $\mathcal{F}_I$, which actually are $\mathcal{S}$-needed family-reductions by Lemma 6.1.(4) and Theorem 7.3, and hence are optimal, implement optimal family-reductions in $\mathcal{F}$.* □

8 Conclusions and Future Work

We have introduced and studied an abstract concept of optimal implementation of DFSs, and showed that needed computations (w.r.t. stable sets of results) in implementation DRSs mimic optimal (in the sense of Lévy and beyond) computations in the original DFSs. Further, we have shown that every affine SDRS can be turned into an affine DFS by taking zig-zag as the family relation, and that zig-zag is the only family relation with the separability property – no redex can create simultaneously two different members of the same family. Finally, we have shown that sharing is compositional. In particular, any family relation can be decomposed into the zig-zag (when it is a family relation) and a non-separable affine family-relation, which facilitates the study of complicated (non-separable) concepts of sharing in duplicating systems (such as the one in [AL93]).

The optimality theory is not the only concern in this work. Non-duplicating systems are of great importance for the study of semantics of computation. Recall that say *distributive domains* (also called *dI-domains* or *stable domains*) correspond to *linear* systems, where there is no duplication nor erasure of redexes [Win86/89].

Indeed, the results on ASDRSs established in this work are the basis of our investigation of the semantics of orthogonal systems. To the best of our knowledge, DFSs are the only abstract systems that allow one to project duplicating and/or erasing computation onto non-duplicating computation,

and indeed the results in this paper prove the correctness of such projection. It is shown in [KG02] that ASDRSs can be seen as the refinement of distributive domains (in the conflict-case): they are distributive domains enriched by an axiomatized *erasure* relation. In these systems, distributivity can be restored by considering needed reductions only.

These projection results have also been used in the definition of *independence* of computations, and construction of Euclidian Geometry from the reduction spaces in orthogonal rewrite systems [KG97a]. The projection results allow to prove results decomposition, normalization and optimality results for duplicating erasing systems by performing proofs for ASDRSs, where proofs are of course much simpler.

We also refer to [KG03] for recent results on computational semantics of conflict free reduction systems, where the framework of SDRSs and DFSs is used as the basis. There, new partial orders, more refined than those of distributive domains are introduced and studied, and relevance of results obtained in this paper for building more refined lambda models from these orderings is an important open question.

References

[AL93] Asperti A., Laneve C. Interaction Systems I: the theory of optimal reductions. MSCS, 11:1-48, Cambridge University Press, 1993.

[AG98] Asperti A., Guerrini, S. The Optimal Implementation of Functional Programming Languages, Cambridge Tracts in Theoretical Computer Science 45, Cambridge University Press, 1998.

[Ber79] Berry G. Modéles complétement adéquats et stables des λ-calculs typés. Thèse de l'Université de Paris VII, 1979.

[GK96] Glauert J.R.W., Khasidashvili Z. Relative normalization in deterministic residual structures. CAAP'96, Springer LNCS, vol. 1059, H. Kirchner, ed. 1996, p. 180-195.

[GK02] Glauert J.R.W., Khasidashvili Z. An Abstract Böhm normalization. WRS'02, vol 70 (6), Elsevier Science B.V., 2002.

[GLM92] Gonthier G., Lévy J.-J., Melliès P.-A. An abstract Standardisation theorem. LICS'92, p. 72-81.

[HL91] Huet G., Lévy J.-J. Computations in Orthogonal Rewriting Systems. In: Computational Logic, Essays in Honor of Alan Robinson, J.-L. Lassez and G. Plotkin, eds. MIT Press, 1991, p. 394-443.

[HP91] Hoffmann B., Plump D. Implementing term rewriting by Jungle evaluation. Theoretical Informatics and Applications 25:445-472, 1991.

[KKSV93] Kennaway J. R., Klop J. W., Sleep M. R, de Vries F.-J. Event structures and orthogonal term graph rewriting. In: M. R. Sleep, M. J. Plasmeijer, and

M. C. J. D. van Eekelen, eds. Term Graph Rewriting: Theory and Practice. John Wiley, 1993.

[KG96] Khasidashvili Z., Glauert J. R. W. Discrete normalization in Deterministic Residual Structures. ALP'96, Springer LNCS, 1139, M. Hanus, M. Rodríguez-Artalejo, eds. 1996, p.135-149.

[KG97] Khasidashvili Z., Glauert J. R. W. Zig-zag, extraction and separable families in stable non-duplicating Deterministic Residual Structures. Technical Report IR-420, Free University, Amsterdam, February 1997.

[KG97a] Khasidashvili Z., Glauert J. R. W. The Geometry of orthogonal reduction spaces. ICALP'97, Springer LNCS, vol. 1256, P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, eds. 1997, p. 649-659. Full version to appear in TCS.

[KG02] Khasidashvili Z., Glauert J. R. W. Relating conflict-free transition and event models via redex families. TCS 286:65-95, 2002.

[KG03] Khasidashvili Z., Glauert J. R. W. Stable computational semantics of conflict-free rewrite systems (partial orders with erasure). RTA'03 (to appear).

[Klo80] Klop J. W. Combinatory Reduction Systems. Mathematical Centre Tracts n. 127, CWI, Amsterdam, 1980.

[Lév78] Lévy J.-J. Réductions correctes et optimales dans le lambda-calcul, Thèse de l'Université de Paris VII, 1978.

[Lév80] Lévy J.-J. Optimal reductions in the Lambda-calculus. In: To H. B. Curry: Essays on Combinatory Logic, Lambda-calculus and Formalism, Hindley J. R., Seldin J. P. eds, Academic Press, 1980, p. 159-192.

[Mar91] Maranget L. Optimal derivations in weak λ-calculi and in orthogonal Term Rewriting Systems. POPL'91, p. 255-269.

[Mel0X] Melliès P.-A. Axiomatic Rewriting Theory I: A diagrammatic standardization theorem, (submitted).

[Oos96] Van Oostrom V. Higher order families. RTA'96, Springer LNCS, vol. 1103, Ganzinger, H., ed., 1996, p. 392–407.

[Plo75] Plotkin G. Call-by-name, call-by-value and the λ-calculus. TCS 1(2):125-159, 1975.

[Sta89] Stark E. W. Concurrent transition systems. TCS 64(3):221-270, 1989.

[Ter03] Terese. Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, Volume 55, Cambridge University Press, 2003.

[Wad71] Wadsworth C.P. Semantics and pragmatics of the Lambda-calculus. Ph.D. thesis, University of Oxford, 1971.

[Win86/89] Winskel G. An introduction to Event Structures. In proc. of School/Workshop on 'Linear time, branching time and partial order in logics and models of concurrency'. Springer LNCS, vol. 354, 1989, p. 364-397. Extended version appears as Cambridge University Computer Laboratory Report n. 95, 1986.

Call-by-Need Reductions for Membership Conditional Term Rewriting Systems

Mizuhito Ogawa [1]

Japan Advanced Institute of Science and Technology, and
Japan Science and Technology Corporation, PRESTO
1-1 Asahidai Tatsunokuchi Nomi Ishikawa, Japan 923-1292

Abstract

This paper proves that, for a membership conditional term rewriting system $\mathcal{R}$, (1) a reducible term has a needed redex if $\mathcal{R}$ is nonoverlapping, and (2) whether a redex is nv-needed is decidable.

1 Introduction

A membership conditional term rewriting system (MCTRS) is a term rewriting system (TRS) with conditions that substitutions are taken from some specific sets; typically the set of normal forms. A normal MCTRS, which restricts substitutions to be in normal forms for each nonlinear variable, is one of useful examples; this can specify the positive part of the equality class of functional programming (e.g., Haskell, ML) without type information. (The negative part of the equality class cannot be deduced without algebraic information of type constructions.)

A nonoverlapping normal MCTRS is a natural extension of an orthogonal TRS to nonlinearity, and it succeeds many nice properties, such as Parallel Move Lemma and confluence [14]. Adding to them, this paper investigates call-by-need reductions for a normal MCTRS. In general, a nonlinear TRS does not have needed redexes even if it is nonoverlapping; for instance,

$$\{d(x,x) \to a, f(y,z) \to b, c \to d\}$$

is a nonoverlapping (and also strongly normalizing, right-ground) nonlinear TRS and $d(f(c,z), f(d,z))$ does not have needed redexes. Thus, the membership restriction is essential; there seem no other choices when one explores the existence of needed redexes in nonlinear TRSs. In fact, the membership condition precisely corresponds to the proof techniques in [10].

[1] Email:mizuhito@acm.org

The main results are:

(i) A reducible term has a needed redex for a nonoverlapping normal MCTRS.

(ii) Reachability and normalizability for a right-ground normal MCTRS are decidable.

(iii) Whether a redex is nv-needed is decidable for a normal MCTRS.

where nv-neededness approximates neededness by relaxing rewrite relation such that variables in the right-hand-side of a rule may be instantiated by any terms.

It is worth remarking that, different from left-linear TRSs, *modern* tree automata techniques [2,5,11] fail for decidability results of normal MCTRSs. This is because the set of normal forms is *not* regular; i.e., the set of normal forms of a normal MCTRS is same to that of the underlying TRS, and the set of normal forms of a nonlinear TRS is known to be not regular [8].

Section 2 prepares basic notations and Section 3 introduces the previous results on confluence of a normal MCTRS. Section 4 shows that a reducible term has a needed redex if a normal MCTRS is nonoverlapping, Section 5 shows that reachability and normalizability of a right-ground normal MCTRS is decidable, and Section 6 shows that whether a redex is nv-needed is decidable for a normal MCTRS. Section 7 concludes the paper and discusses on future topics.

2 Preliminaries

I assume that readers are familiar with rewriting terminology. For details, we refer to [7]. This section mostly explains our notations. Throughout the paper, we will consider only finite term rewriting systems (TRSs).

We will denote the set of function symbols by $\mathcal{F}$, the set of n-ary function symbols by $\mathcal{F}_n$, the set of variables by $\mathcal{V}$, and the set of terms over $\mathcal{F}$ and $\mathcal{V}$ by $T(\mathcal{F},\mathcal{V})$. A term without variables is a ground term, and the set of ground terms is denoted by $T(\mathcal{F})$. A term t is *linear* if each variable x appears in t at most once, and a variable x in a term t is *linear* if x appears once in t. The set of variables that appear in a term t is denoted by $\mathcal{V}ar(t)$, and the set of nonlinear variables that appear in a term t is denoted by $\mathcal{V}ar^{nl}(t)$. For a (possible nonlinear) term t, $\bar{t}$ is a linearization of t, i.e., $\bar{t}$ is obtained by replacing all occurrences of nonlinear variables in t by distinct fresh variables (thus, $\bar{t}$ is linear).

We denote the set of all positions in a term t by $Pos(t)$, the subterm occurring at p in t by $t|_p$, and the head symbol of t by $head(t)(\in \mathcal{F}\cup\mathcal{V})$.

For terms t, s and position $p \in Pos(t)$, $t[s]_p$ is the term obtained from t by replacing the subterm at p with s. For positions p, q and a set U of positions, we denote $p < q$ if p is a proper prefix of q, and $p \perp q$ if neither $p < q$, $p = q$, nor $p > q$, $U < q$ if $\exists p \in U\ p < q$, and $U \leq q$ if $\exists p \in U\ p \leq q$. For a term t and a variable x, we denote the set of positions in t by $\mathcal{P}os(t)$, the set of

positions of function symbols in t by $\mathcal{P}os_{\mathcal{F}}(t)$, the set of positions of variables in t by $\mathcal{P}os_{\mathcal{V}}(t)$, the set of positions where x occurs in t by $\mathcal{P}os(t,x)$, and the number of positions in $\mathcal{P}os(t)$ (i.e., size of t) by $|t|$. If s is a (proper) subterm of t, we denote $s \trianglelefteq t$ ($s \triangleleft t$).

We denote $\xrightarrow{p}_{\mathcal{R}}$ if a rewrite $\to_{\mathcal{R}}$ occurs at the position p, $\xrightarrow{>p}_{\mathcal{R}}$ if a rewrite $\to_{\mathcal{R}}$ occurs at the position larger than p, and $\xrightarrow{\geq p}_{\mathcal{R}}$ if a rewrite $\to_{\mathcal{R}}$ occurs at the position larger than or equal to p. For a set P of positions, we denote $\xrightarrow{\geq P}_{\mathcal{R}}$ if a rewrite $\to_{\mathcal{R}}$ occurs at the position larger than or equal to some $p \in P$.

A term without redexes (of $\to_{\mathcal{R}}$) is a *normal form* (more specifically, $\mathcal{R}$-*normal form*), and the set of $\mathcal{R}$-normal forms is denoted by $NF_{\mathcal{R}}$. We will often omit the index $\mathcal{R}$ in $NF_{\mathcal{R}}$ and $\to_{\mathcal{R}}$ if they are apparent from the context.

3 Confluence of membership conditional TRS

A pair of rules $(l \to r, l' \to r')$ is *overlapping* if there exists a position p such that

- $p \in \mathcal{P}os_{\mathcal{F}}(l)$,
- there exist substitutions σ, σ' such that $l|_p \ \sigma = l'\sigma'$, and
- either $p \neq \epsilon$, or $l \to r$ and $l' \to r'$ are different rules.

A TRS is *nonoverlapping* if no pairs of rules in $\mathcal{R}$ are overlapping.

Theorem 3.1 ([6]) *A left-linear nonoverlapping TRS is confluent.*

Without left-linearity, confluence may fail even for nonoverlapping TRSs. For instance, $\mathcal{R}_1$ below is nonoverlapping, but not confluent.

$$\mathcal{R}_1 = \left\{ \begin{array}{ll} d(x,x) & \to 0 \\ d(x,f(x)) & \to 1 \\ 2 & \to f(2) \end{array} \right\}$$

When nonlinear, some restriction is required to recover confluence. A membership conditional TRS is such an example [14].

Definition 3.2 A *membership conditional* TRS (MCTRS) $\mathcal{R}$ is a finite set of conditional rewrite rules

$$l \to r \Leftarrow C$$

where the condition C is $\wedge_{x \in V} x \in T_x (\subseteq T(\mathcal{F}, \mathcal{V}))$ for $V \subseteq \mathcal{V}ar(l)$.

An MCTRS $\mathcal{R}$ is *normal* if C is $\wedge_{x \in V} x \in NF_{\mathcal{R}}$ and and $\mathcal{V}ar^{nl}(l) \subseteq V \subseteq \mathcal{V}ar(l)$.

Remark 3.3 Since the membership condition is non-monotonic wrt the inclusion of rewrite relations, it looks contradictory. But, this is not true; by induction on the size of a term , the rewrite relation is well-defined (refer to Lemma 4.1 in [14]).

Theorem 3.4 [14] *A nonoverlapping normal MCTRS is confluent.*

Example 3.5 The normal MCTRS, $\mathcal{R}_1$ plus additional membership conditions,

$$\left\{\begin{array}{lll} d(x,x) & \to 0 & x \in NF \\ d(x,f(x)) & \to 1 & x \in NF \\ 2 & \to f(2) & \end{array}\right\}$$

is confluent.

4 Needed redex of nonoverlapping normal MCTRS

A term may have several redexes. A *reduction strategy* is a choice of a redex to rewrite (i.e., a function from a term to a redex in a term). Especially, the normalizing strategy, which guarantees to reach a normal form (if exists), and the optimal strategy, which selects a needed redex, are important issues. A redex is *needed* if either itself or its descendant is contracted in every rewrite sequence to a normal form.

In general, needed redexes may not exist. However, a left-linear nonoverlapping TRS has a needed redex in a term not in normal forms (although it may be *not* computable). The idea in [10] is; instead of a needed redex, a root-needed redex is considered. A redex is *root-needed* if either itself or some of its descendants is contracted in every rewrite sequence to a root-stable form (i.e., a term that cannot be reduced to a redex). Since a normal form is root-stable, a root-needed redex is a needed redex. In this section, similar to [10], we show that a reducible term has a needed redex if a normal MCTRS is nonoverlapping.

For a rewrite sequence $A: t_0 \to \cdots \to t_n$ and $0 \le i \le j \le n$, $B: t_i \to \cdots \to t_j$ is a subsequence, $C: t_0 \to \cdots \to t_i$ is a *prefix sequence*, and $D: t_i \to \cdots \to t_n$ is a *suffix sequence*. For rewrite sequences $A: s \to^* t$ and $B: t \to^* u$, the concatenation is denoted by $A; B: s \to^* u$.

Definition 4.1 A term $s(\trianglelefteq t)$ is *root-stable* if for any s' with $s \to^* s'$, s' is not a redex.

Lemma 4.2 (Lemma 2.1 [10]) *If $\to$ is confluent, $\overset{>\epsilon}{\to}$ is also confluent.*

Lemma 4.3 (Lemma 3.2 [10]) *If a term s is root-stable, each term t with $s \to^* t$ is also root-stable.*

Lemma 4.2 and 4.3 hold for any TRSs. The proof of Lemma 3.3 and 4.2 in [10] require the left-linear and nonoverlapping property [2] to guarantee

- confluence, and
- if s is a redex, t with $s \overset{>\epsilon}{\to}{}^* t$ is also a redex.

[2] More precisely, the requirement of Lemma 3.3 in [10] can be relaxed to almost nonoverlapping.

A nonoverlapping normal MCTRS also satisfies these requirements, and the same proofs in [10] give following Lemma 4.4 and 4.5.

Lemma 4.4 *For a nonoverlapping normal MCTRS $\mathcal{R}$, if a term t is root-stable, each term s with $s \overset{>\epsilon}{\to}^* t$ is also root-stable.*

Lemma 4.5 *For a nonoverlapping normal MCTRS $\mathcal{R}$ and a term t, if there exist a rule $l \to r \in \mathcal{R}$ and a substitution σ such that $t \overset{>\epsilon}{\to}^* l\sigma$ then l is uniquely determined regardless of reduction sequences.*

Similar to Theorem 4.3 in [10], which states that for a left-linear nonoverlapping TRS a term not root-stable has a root-needed redex, the next theorem holds.

Theorem 4.6 *For a nonoverlapping normal MCTRS, a term that is not root-stable has a root-needed redex.*

Proof. By induction on the size of a term. Without loss of generality, we can assume that t is neither a root-stable term nor a redex. Assume $t \to^+ t'$ for some root-stable form t'. From lemma 4.4, a reduction sequence $A : t \to^* t'$ contains a reduction $A : t \overset{>\epsilon}{\to}^+ \Delta_A \overset{\epsilon}{\to} t'' \to^* t'$ at the position ϵ. From lemma 4.5, a rule used in Δ_A is uniquely determined. We denote it by $l \to r \Leftarrow C$ where $C = \wedge_{x \in V} x \in NF$ with $\mathcal{V}ar^{nl}(l) \subseteq V \subseteq \mathcal{V}ar(l)$.

Let P be the set of positions of proper non-root-stable subterms in t. There are two cases; (1) $P \cap \mathcal{P}os_{\mathcal{F}}(l) \neq \phi$, and (2) $P \cap \mathcal{P}os_{\mathcal{F}}(l) = \phi$. For (1) [3], let p be a minimal position in $P \cap \mathcal{P}os_{\mathcal{F}}(l)$. Since $t|_p \subset t$, from induction hypothesis, there exists a root-needed redex Δ in $t|_p$. We claim that Δ is contracted in the subsequence $B : t \overset{>\epsilon}{\to}^+ \Delta_A$. From minimality of p, $t|_p$ must be rewritten to $\Delta_A|_p$, which is root-stable (because $\mathcal{R}$ is nonoverlapping). Thus, Δ is root-needed in t.

For (2), if each $p \in P$ is $p \in \mathcal{P}os(l, x)$ for some $x \in \mathcal{V}ar(l) \setminus V$, t is a redex; thus, root-needed because $\mathcal{R}$ is nonoverlapping. Assume there exists $p \in \mathcal{P}os(l, x)$ for $x \in V$. Let B be a subsequence of A such that $B : t|_p \to^+ \Delta_A|_p$. Since $C = \wedge_{x \in V} x \in NF$, $\Delta_A|_p$ is a normal form. From induction hypothesis, $t|_p$ has a root-needed redex, and this is also root-needed in t. □

Corollary 4.7 *For a nonoverlapping normal MCTRS, a reducible term has a needed redex.*

Remark 4.8 Since a nonoverlapping normal MCTRS satisfies Parallel Move Lemma, the same proofs in Section 5 in [10] work for a nonoverlapping normal MCTRS. Thus, the repeated reduction of root-needed redexes is a root-normalizing reduction strategy.

Further, since a nonoverlapping normal MCTRS is confluent, a context-free root-normalizing reduction strategy is a normalizing reduction strategy (refer to Theorem 6.5 in [10]), where a reduction strategy is context-free if the

[3] For (1), the proof is same as in theorem 4.3 in [10].

choice of a redex in a root-stable term is reduced to the choice of a redex in each direct subterm.

5 Decidable results for right-ground normal MCTRS

In [12], Oyamaguchi proved that reachability and joinability are decidable for a (possibly non-left-linear) right-ground TRS. In this section, similar to his method, we show that reachability and normal joinability are decidable for a right-ground normal MCTRS. The main difference is that we use normal joinability $\{t_1, \cdots, t_n\}_N$ instead of joinability $\{t_1, \cdots, t_n\}_J$ in [12]. Except that, the translation of the proof in [12] is quite straight forward.

We say:

- For terms s, t, s is *reachable* to t if $s \to^* t$, and denoted by $(s, t)_R$.
- Terms $t_1, \cdots, t_n$ are *joinable in normal form* if there exists $t \in NF_R$ with $t_i \to^* t$ for each i, and denoted by $\{t_1, \cdots, t_n\}_N$.

We call $s \to^* t$ a witness of $(s, t)_R$, and the existence of $t \in NF_{\mathcal{R}}$ with $t_i \to^* t$ for each i, a witness of $\{t_1, \cdots, t_n\}_N$. Note that normalizability of a term t is expressed as $\{t\}_N$.

We say that a rewrite sequence $s \to^* t$ is *top-invariant* if $s \overset{\geq\epsilon}{\to}{}^* t$. For a right-ground normal MCTRS $\mathcal{R} = \{l_i \to r_i \Leftarrow C_i\}$, we denote the set $\{l_i\}$ (resp.$\{r_i\}$) of the left-hand-sides (resp. right-hand-sides) of rules in $\mathcal{R}$ by $\mathcal{R}^l$ (resp. $\mathcal{R}^r$). From now on, throughout this section, $\mathcal{R}$ is a right-ground normal MCTRS.

Definition 5.1 For a term t, $\delta_{\mathcal{R}}(t) = \{t' \mid t' \trianglelefteq t \ \vee \ t' \trianglelefteq r \in \mathcal{R}^r\}$. A substitution θ is a $\delta_{\mathcal{R}}(t)$-*substitution*, if for each variable x, $x\theta \in \delta_{\mathcal{R}}(t)$.

We start with an explicit construction of the search space, i.e., possible reduction of $(s, t)_R$ and $\{t_1, \cdots, t_n\}_N$ to "*smaller*" problems. During the construction, next Lemma 5.2 is the key.

Lemma 5.2 *If a rewrite sequence $A : \ s \to^* t$ is not top-invariant, there exist $l \to r \Leftarrow C \in \mathcal{R}$, a substitution σ, and a $\delta_{\mathcal{R}}(s)$-substitution θ such that*

$$B : \ s \ \overset{\geq\epsilon}{\to}{}^* \ \bar{l}\theta \ \overset{\geq \mathcal{P}os_{\mathcal{V}}(l)}{\to}{}^* \ l\sigma \ \overset{\epsilon}{\to} \ r \ \to^* t$$

with the same rewrite steps. (Recall the $\bar{l}$ is a linearization of l.)

Proof. Since A is not top-invariant, there exists a rewrite at the root ϵ. Let $l\sigma \overset{\epsilon}{\to} r$ be the first such rewrite. Let A' be the prefix sequence of A from s to $l\sigma$, and let A'' be the suffix sequence of from $l\sigma$ to t. Then, $A' : \ s \overset{\geq\epsilon}{\to}{}^* l\sigma$. Let $\{p_1, \cdots, p_n\} = \mathcal{P}os_{\mathcal{V}}(l)$ and let A_i be the maximum suffix sequence in A' such that all rewrites are below or equal to p_i. Then, by interchanging the order of parallel rewrites, we can decompose A' as $C; A_1; \cdots; A_n$. By construction of $A_1, \cdots, A_n$, there exists a substitution θ such that $C : \ s \to^* \bar{l}\theta$ and, for each p_i,

(i) either all rewrite steps in C are parallel to p_i, or

(ii) the last rewrite step in C that is *not* parallel to p_i occurs above p_i.

Let $x_i \in \mathcal{V}ar(\bar{l})$ with $\{p_i\} = \mathcal{P}os(\bar{l}, x_i)$. For (i), $x_i\theta = s|_{p_i}$, and for (ii), $x_i\theta$ is a subterm of r' for some $r' \in \mathcal{R}^r$ (Recall that $\mathcal{R}$ is right-ground). Thus, θ is a $\delta_{\mathcal{R}}(s)$-substitution. □

Definition 5.3 Let $s, t, t_1, \cdots, t_n$ be terms and let θ be a $\delta_{\mathcal{R}}(s)$-substitution. Define $\Phi_R((s,t)_R) = \Phi_{R,1}((s,t)_R) \cup \Phi_{R,2}((s,t)_R)$ and $\Phi_N(\{t_1, \cdots, t_n\}_N) = \Phi_{N,1}(\{t_1, \cdots, t_n\}_N) \cup \Phi_{N,2}(\{t_1, \cdots, t_n\}_N)$ where:

$$\Phi_{R,1}((s,t)_R) = \{ \{(s_i, t_i)_R \mid s = f(s_1, \cdots, s_n), t = f(t_1, \cdots, t_n)\} \} \quad \text{if } head(s) = head(t)$$

$$\Phi_{R,2}((s,t)_R) = \left\{ \begin{array}{l|l} \{(s, \bar{l}\theta)_R, (r,t)_R\} \cup & l \to r \Leftarrow C \in \mathcal{R} \\ (\cup_{x \in \mathcal{V}ar^{nl}(l),\ p_i \in \mathcal{P}os(l,x)} \{\{\bar{l}|_{p_1}\ \theta, \cdots, \bar{l}|_{p_m}\ \theta\}_N\}) & \forall x. x\theta \in \delta_{\mathcal{R}}(s) \end{array} \right\}$$

$$\Phi_{N,1}(\{t_1, \cdots, t_n\}_N) = \{ \{\{t_1|_j, \cdots, t_n|_j\}_N \mid 1 \le j \le arity(head(t_1))\} \} \quad \text{if } head(t_1) = \cdots = head(t_n)$$

$$\Phi_{N,2}(\{t_1, \cdots, t_n\}_N) = \{ \{(t_i, r)_R,\ \{t_1, \cdots, t_{i-1}, r, t_{i+1}, \cdots, t_n\}_N\} \mid r \in \mathcal{R}^r \}$$

The intuition for $\Phi_R((s,t)_R)$ and $\Phi_N(\{t_1, \cdots, t_n\}_N)$ is the set of candidates of the reduction of the problem. For instance, $\Phi_{R,1}((s,t)_R)$ corresponds to the case that the witness $s \to^* t$ of $(s,t)_R$ is top-invariant, and $\Phi_{R,2}((s,t)_R)$ corresponds to the case that it is not top-invariant. They enumerate all possible reduction based on Lemma 5.2. Similarly, $\Phi_{N,1}(\{t_1, \cdots, t_n\}_N)$ corresponds to the case that the witness, *for some* $t \in NF_{\mathcal{R}}$, $t_i \to^* t$ *for each* i, is top-invariant for each i. $\Phi_{N,2}(\{t_1, \cdots, t_n\}_N)$ corresponds to the case that some $t_i \to^* t$ is not top-invariant. We assume that redundancy in Φ_R and Φ_N is removed as

- to eliminate $(s,s)_R$, and
- to reduce $\{\cdots, t, t, \cdots\}_N$ to $\{\cdots, t, \cdots\}_N$.

Let either $\rho = (s,t)_R$ or $\rho = \{t_1, \cdots, t_n\}_N$. Next, we define the search path $\Psi_\alpha(\rho)$ for the sequence α of pairs of integers.

Definition 5.4 Let

$$\Phi(\rho) = \begin{cases} \Phi_R(\rho) & \text{if } \rho = (s,t)_R, \\ \Phi_N(\rho) & \text{if } \rho = \{t_1, \cdots, t_n\}_N. \end{cases}$$

and let α be a sequence of pairs of integers. Then, a search path $\Psi_\alpha(\rho)$ is

inductively defined as:

$$\begin{aligned}\Psi_\epsilon(\rho) &= \{\rho\}\\ \Psi_{\alpha.(i,j)}(\rho) &= \{\tau_1,\cdots,\tau_{i-1},\tau_{i+1},\cdots,\tau_m\}\ \cup\ \bar{\tau}_j\\ &\qquad \text{where } \{\tau_1,\cdots,\tau_m\}=\Psi_\alpha(\rho) \text{ and } \{\bar{\tau}_1,\cdots,\bar{\tau}_k\}=\Phi(\tau_i)\end{aligned}$$

We will show that to decide ρ, it is enough to check on finitely many $\Psi_\alpha(\rho)$.

Definition 5.5 Let $s,t,t_1,\cdots,t_n$ be terms. Assume that $(s,t)_R$ and $\{t_1,\cdots,t_n\}_N$ have witness. We denote the minimal (sum of) rewrite steps of the witness of $(s,t)_R$ and $\{t_1,\cdots,t_n\}_N$ by $step((s,t)_R)$ and $step(\{t_1,\cdots,t_n\}_N)$, respectively. Define *weight* ω by

$$\begin{aligned}\omega((s,t)_R) &= (step((s,t)_R),|s|)\\ \omega(\{t_1,\cdots,t_n\}_N) &= (step(\{t_1,\cdots,t_n\}_N),|t_1|+\cdots+|t_n|)\end{aligned}$$

and the lexicographical order over weight is

$$(i,j)>(i',j')\ \Leftrightarrow\ i>i'\ \vee\ (i=i'\ \wedge\ j>j').$$

The next lemma is immediate.

Lemma 5.6 *Let either $\rho=(s,t)_R$ or $\rho=\{t_1,\cdots,t_n\}_N$, and let $\bar{\tau}\in\Phi(\rho)$. If each $\tau\in\bar{\tau}$ has a witness and they give ρ a witness, $\omega(\tau)<\omega(\rho)$.*

We denote the maximum multiplicity of (nonlinear) variables in l in $\mathcal{R}^l$ by $a_\mathcal{R}$, and $\{\overline{l\theta}\mid l\in\mathcal{R}^l,\theta \text{ is a } \delta_\mathcal{R}(s)\text{-substitution}\}$ by $\Delta_\mathcal{R}(s)$.

Definition 5.7 Let $s,t,t_1,\cdots,t_n$ be terms.

$$\begin{aligned}S_R((s,t)_R) &= \{(s',t')_R \mid s'\trianglelefteq s\ \vee\ s'\trianglelefteq r\in\mathcal{R}^r \text{ and}\\ &\qquad t'\trianglelefteq u\in\Delta_\mathcal{R}(s)\ \vee\ t'\trianglelefteq t\ \vee\ t'\trianglelefteq r\in\mathcal{R}^r\}\\ S_N((s,t)_R) &= \{\{t_1,\cdots,t_k\}_N \mid 1\le k\le a_\mathcal{R} \text{ and } t_i\trianglelefteq s\ \vee\ t_i\trianglelefteq r\in\mathcal{R}^r\}\end{aligned}$$

$$\begin{aligned}S_R(\{t_1,\cdots,t_n\}_N) &= \{(s',t')_R \mid s'\trianglelefteq t_i\ \vee\ s'\trianglelefteq r\in\mathcal{R}^r \text{ and}\\ &\qquad t'\trianglelefteq u\in\cup_{1\le i\le n}\Delta_\mathcal{R}(t_i)\ \vee\ t'\trianglelefteq r\in\mathcal{R}^r\}\\ S_N(\{t_1,\cdots,t_n\}_N) &= \{\{t'_1,\cdots,t'_k\}_N \mid 1\le k\le max(n,a_\mathcal{R}) \text{ and}\\ &\qquad (\vee_{1\le i\le n}\ t'_j\trianglelefteq t_i)\ \vee\ t'_j\trianglelefteq r\in\mathcal{R}^r\}\end{aligned}$$

Lemma 5.8 *Let either $\rho=(s,t)_R$ or $\rho=\{t_1,\cdots,t_n\}_N$. Then, for each α, $\Psi_\alpha(\rho)\ \subseteq\ S_R(\rho)\cup S_N(\rho)$.*

Proof. Since $s\in\delta_\mathcal{R}(t)$ implies $\delta_\mathcal{R}(s)\subseteq\delta_\mathcal{R}(t)$, by induction on the length of α, $\Phi_\alpha(\rho)\ \subseteq\ S_R(\rho)\cup S_N(\rho)$. □

Now, we show that it is enough to consider a search path $\Phi_\alpha(\rho)$ with the upper bound for the length of α.

Lemma 5.9 *Let either $\rho = (s,t)_R$ or $\rho = \{t_1, \cdots, t_n\}_N$. There exists an upper bound M_L such that if $\Phi_\alpha(\rho) \neq \phi$ with $|\alpha| > M_L$, then, for any β that contains α as a prefix, $\Phi_\beta(\rho)$ does not give a witness of ρ.*

Proof. Since $\Delta_\mathcal{R}(s)$, $\Delta_\mathcal{R}(t_1)$, $\cdots$, $\Delta_\mathcal{R}(t_n)$ are finite, $S_R(\rho)$ and $S_N(\rho)$ are finite by construction. Let $M_L = 2^{|S_R(\rho) \cup S_N(\rho)|}$. Assume that $\Phi_\beta(\rho)$ gives a witness of ρ for some β that contains α as a prefix. Without loss of generality, we assume $\beta = \alpha$. Then, $\Phi_\alpha(\rho)$ also gives $\Phi_{\alpha'}(\rho)$ a witness for each prefix α' of α.

From Lemma 5.6, for each prefix α' and α'' of α, if α'' is a proper prefix of α', $\Psi_{\alpha'}(\rho) \ll \Psi_{\alpha''}(\rho)$, where $\ll$ is a multiset extension of $<$. Thus, $\Phi_{\alpha'}(\rho) \neq \Phi_{\alpha''}(\rho)$. From Lemma 5.8 and $|\alpha| > M_L$, this is contradiction. □

At last, we give an upper bound M_W for branching of a search path.

Lemma 5.10 *Let either $\rho = (s,t)_R$ or $\rho = \{t_1, \cdots, t_n\}_N$. There is an upper bound M_W such that, for each $\tau \in \bar{\tau} \in \Psi_\alpha(\rho)$, $|\Phi(\tau)| \leq M_W$.*

Proof. Let $v_\mathcal{R} = max\,\{|\mathcal{V}ar(l)| \mid l \in \mathcal{R}^l\}$ and $m = max\,\{|\delta_\mathcal{R}(s)|, |\delta_\mathcal{R}(t_1)|, \cdots, |\delta_\mathcal{R}(t_n)|\}$. Define $M_W = m^{v_\mathcal{R}} \cdot |\mathcal{R}| + 1$. Since $s \in \delta_\mathcal{R}(t)$ implies $\delta_\mathcal{R}(s) \subseteq \delta_\mathcal{R}(t)$, if $\tau = (s,t)_R)$, $|\Phi_R((s,t)_R)| \leq m^{v_\mathcal{R}} \cdot |\mathcal{R}| + 1$, and if $\tau = \{t_1, \cdots, t_n\}_N$, $|\Phi_N(\{t_1, \cdots, t_n\}_N)| \leq |\mathcal{R}| + 1$. Thus, $|\Phi(\tau)| \leq M_W$. □

Theorem 5.11 *For a right-ground normal MCTRS, reachability and normal joinability are decidable.*

Proof. Let either $\rho = (s,t)_R$ or $\rho = \{t_1, \cdots, t_n\}_N$. From Lemma 5.9, it is enough to consider $\Phi_\alpha(\rho)$ with $|\alpha| \leq M_L$. The number of candidates for the next $\Phi_{\alpha.(i,j)}(\rho)$ is at most $|S_R(\rho) \cup S_N(\rho)| \times M_W$, because the possible choice of i is at most $|S_R(\rho) \cup S_N(\rho)|$ from Lemma 5.8, and that of i is at most M_W from Lemma 5.10. Thus, the set of search paths to check is finite, and the theorem follows. □

Corollary 5.12 *For a right-ground normal MCTRS, normalizability is decidable.*

Remark 5.13 Since a normal form may not be preserved when adding context, even if $\Phi_\alpha(\rho)$ has a witness, this does not mean ρ has a witness. We further need to check that $\Phi_\alpha(\rho)$ actually gives ρ a witness. For instance, the witness of $\{t_{1,1}, t_{2,1}\}_N$, $\{t_{1,2}, t_{2,2}\}_N$ that *there exist $t_1, t_2 \in NF_\mathcal{R}$ such that $t_{1,1}, t_{2,1} \to^* t_1$ and $t_{1,2}, t_{2,2} \to^* t_2$*, does not mean that $f(t_{1,1}, t_{1,2})$, $f(t_{2,1}, t_{2,2}) \to^* f(t_1, t_2)$ is a witness of $\{f(t_{1,1}, t_{1,2}), f(t_{2,1}, t_{2,2})\}_N$, because it may be $f(t_1, t_2) \notin NF_\mathcal{R}$. However, this search path is produced for a top-invariant case, and $\{f(t_{1,1}, t_{1,2}), f(t_{2,1}, t_{2,2})\}_N$ will be analyzed in another search path for a not top-invariant case (note that $t_1, t_2 \in NF_\mathcal{R}$ implies $f(t_1, t_2)$ is a redex). Thus, we simply judge this search path fails.

6 NV-needed redex of normal MCTRS

In this section, we provide an alternative definition of a *needed* redex as in [5], and show that whether a redex is nv-needed is decidable for a normal MCTRS. The equivalence of two definitions for a nonoverlapping normal MCTRS is obtained similar to Lemma 4.1 in [5].

Definition 6.1 Let $\bullet \notin \mathcal{F}$, $s \in T(\mathcal{F}, \mathcal{V})$. A redex $s|_p$ (in s) is *needed* if $s[\bullet]_p$ does not rewrite to a normal form without $\bullet$.

Definition 6.2 Let $\Omega(\notin \mathcal{F})$ be a fresh constant and let t be a term. t_Ω is a term obtained by replacing each variable in a term t with Ω.

For terms $t, u \in T(\mathcal{F} \cup \{\Omega\}, \mathcal{V})$, we denote $t \trianglelefteq u$ if t is obtained from u by replacing subterms in u by Ω's.

Definition 6.3 Let $\mathcal{R}$ be a normal MCTRS, and s, t be term. Let $p \in \mathcal{P}os(s)$. If $s|_p$ is a redex of $l \to r \Leftarrow C \in \mathcal{R}$,

$$s \to_{nv} t \Leftrightarrow t = s[u]_p \text{ where } r_\Omega \trianglelefteq u \in T(\mathcal{F})$$

Definition 6.4 A redex is *nv-needed* if it is needed under $\to_{nv}$.

From now on, we concentrate on rewrite sequences starting from ground terms, and we assume $\mathcal{F}_0 \neq \phi$. This restriction does not lose generality, because for a rewrite sequence starting from a non-ground term t, we can regard the variables in t as additional constants.

Definition 6.5 Assume $\mathcal{F}_0 \neq \phi$. For a normal MCTRS $\mathcal{R}$, we define right-ground normal MCTRSs $\mathcal{R}_\Omega$, $\mathcal{R}^1_\Omega$, and $\mathcal{R}^2_\Omega$ as follows.

$$\begin{aligned}
\mathcal{R}_\Omega &= \mathcal{R}^1_\Omega \ \cup \ \mathcal{R}^2_\Omega \\
\mathcal{R}^1_\Omega &= \{l \to r_\Omega \Leftarrow C \mid l \to r \Leftarrow C \in \mathcal{R}\} \\
\mathcal{R}^2_\Omega &= \{\Omega \to f(\underbrace{\Omega, \cdots, \Omega}_{n}) \mid f \in \mathcal{F}_n,\ n \geq 0\}
\end{aligned}$$

We denote $\to_{\mathcal{R}_\Omega}$ (resp. $\to_{\mathcal{R}^1_\Omega}$, $\to_{\mathcal{R}^2_\Omega}$) by $\to_\Omega$ (resp. $\to^1_\Omega$, $\to^2_\Omega$).

Lemma 6.6 *Let s, t be ground terms. Assume $\mathcal{F}_0 \neq \phi$. If $s \to^*_\Omega t$ and $\Omega \not\trianglelefteq s$, then, for each term t' with $t \trianglelefteq t'$ and $\Omega \not\trianglelefteq t'$, $s \to^*_{nv} t'$.*

Proof. By induction on the number m of the occurrences of $\to^1_\Omega$'s in $s \to^*_\Omega t$. Since $\mathcal{F}_0 \neq \phi$, it is easy for $m = 1$.

Assume $m > 1$ and let $s \to^*_\Omega u \overset{p}{\to}{}^1_\Omega v\ (\to^2_\Omega)^* t$. Without loss of generality, we can assume that every rewrite in $v\ (\to^2_\Omega)^* t$ occurs at a position larger-than-or-equal-to p.

Since $u|_p$ is a redex of $\mathcal{R}^1_\Omega$, for each u' with $u|_p \trianglelefteq u'$ and $\Omega \not\trianglelefteq u'$, u' is a redex of $\mathcal{R}^1_\Omega$. (Note that $\Omega \notin NF_{\mathcal{R}_\Omega}$; thus Ω does not appear below nonlinear

variable positions in the rewrite $u|_p \to v|_p$.) Thus, we can modify the rewrite sequence as $s \to_\Omega^* u\ (\to_\Omega^2)^*\ u[u']_p \overset{p\ 1}{\to}_\Omega v\ (\to_\Omega^2)^*\ t$.

For each t' with t' with $t \trianglelefteq t'$ and $\Omega \not\trianglelefteq t'$, $u[u']_p \trianglelefteq t'[u']_p$ and $\Omega \not\trianglelefteq t'[u']_p$. Thus, from induction hypothesis, $s \to_{nv}^* t'[u']_p$, and $u' \to_{nv} t'|_p$. This concludes $s \to_{nv}^* t'$. □

Lemma 6.7 *Let s, t be ground terms with $\Omega \not\trianglelefteq s$. If $s \to_{nv}^* t$, then $s \to_\Omega^* t$.*

Proof. By induction on the number m of rewrite steps of $s \to_{nv}^* t$. For $m = 1$, easy. (Note that $s \to_{nv} t$ implicitly implies $\mathcal{F}_0 \neq \phi$.) Assume $m > 1$. Let $s \to_{nv}^* u \to_{nv} t$. Since the reduction of $\to_{nv}$ does not produce Ω, $\Omega \not\trianglelefteq u$. Thus, induction hypothesis implies $s \to_\Omega^* u \to_\Omega^* t$. □

Lemma 6.8 *For a term t with $\Omega \not\trianglelefteq t$, t is normalizable wrt $\to_{nv}$, if, and only if t is normalizable wrt $\to_\Omega$.*

Proof. A term without Ω is a normal form wrt $\to_{nv}$ if, and only if a normal form wrt $\to_\Omega$. Thus, from Lemma 6.6 and 6.7. □

Since $R_\Omega\ \cup \{\bullet \to \bullet\}$ is a right-ground normal MCTRS, Theorem 6.9 is immediate from Corollary 5.12 and Lemma 6.8.

Theorem 6.9 *For a normal MCTRS, whether a redex is nv-needed is decidable.*

Remark 6.10 $\to_{nv}$ approximates $\to$ (i.e., $NF_\to = NF_{\to_{nv}}$ and $\to \subseteq \to_{nv}$), thus from Lemma 4.5 in [5] show that nv-needed redexes are really needed. Thus, from the remark at the end of Section 4, the repeated reduction of nv-needed redexes is a normalizing strategy (if nv-need redexes exist in each reducible term).

Remark 6.11 As pointed out in Example 5.1 [9], repeated reduction of nv-needed redexes is *not* root-normalizing.

7 Conclusion

This paper investigated call-by-need reductions for a normal membership conditional term rewriting system (MCTRS). Main results are:

(i) A reducible term has a needed redex for a nonoverlapping normal MCTRS.

(ii) Reachability and normalizability for a right-ground normal MCTRS are decidable.

(iii) Whether a redex is nv-needed is decidable for a normal MCTRS.

For the first result, there seem no other choices when one explores the existence of needed redexes in nonlinear TRSs; in fact, the membership condition precisely corresponds to the proof techniques in [10].

For the second and the third result, I expect that reachability and normalizability of a shallow [2,3] and right-linear normal MCTRS would be decidable, and nv-neededness could be extended to shallow-neededness.

Note that, different from left-linear TRSs, growing neededness is undecidable for normal MCTRSs, because Post's Correspondence Problem (PCP) is described as a reachability (or normalizability) problem of a growing and right-linear normal MCTRS (either with or without membership conditions). Let $\{(\alpha_i, \beta_i) \mid 1 \leq i \leq n\}$ be the set of n-pairs of finite sequences. Then, PCP is equivalent to whether A is reachable to B where

$$\left\{\begin{array}{lll} a \to d(\alpha_1(c), \beta_1(c)), & d(x,y) \to d(\alpha_1(x), \beta_1(y)), \\ \quad \ldots & \quad \ldots & \\ a \to d(\alpha_n(c), \beta_n(c)), & d(x,y) \to d(\alpha_n(x), \beta_n(y)), \\ & d(x,x) \to b \ \Leftarrow x \in NF. \end{array}\right\}$$

Here, we assume that the symbol a, b, c, d do not appear in α_i and β_i, and we regard a finite sequence α as applications of monadic function symbols (such as $\alpha(x) = f(g(h(x)))$ for $\alpha = fgh$).

This shows the difficulty for proving decidability results of normal MCTRSs. Another difficulty comes from that modern techniques based tree automata [2,5,11] fail for normal MCTRSs. This is because the set of normal forms of a normal MCTRS is *not* regular; i.e., the set of normal forms of a nonlinear TRS is *not* regular [8], and the reducibility of a normal MCTRS is equivalent to that of the underlying TRS (assume a term matches to the left-hand-side of some rule at some position. If all nonlinear variables are instantiated with normal forms, then it is reducible; otherwise, there is a redex below some nonlinear variable position).

There remains another decidability problem: *whether every reducible term of a normal MCTRS has a nv-needed redex is decidable.* A similar result is found in [13] in the context of sequentiality by a classical method. However, the proof of Assertion 1 in Theorem 6.6 in [13] does not work (at least directly), and the gap is not yet fulfilled.

If a normal MCTRS has only shallow nonlinear variables (i.e., each occurrence of a nonlinear variable in the left-hand-side of a rule in a normal TRS is at depth 1), tree automata with brotherhood equality [1] would work similar to [5], which is in principle the same to a classical tree automata. I expect a reduction automata [4] would further help for this direction.

Acknowledgment

I thank to TRS meeting members, especially Aart Middeldorp, Michio Oyamaguchi, Masahiko Sakai, and Yoshihito Toyama for fruitful discussions, and anonymous referees for detailed comments, which improve the draft a lot.

References

[1] B. Bogaert, F. Seynheave, and S. Tison. The recongnizability problem for tree automata with comparisons between brothers. In *FoSSaCS 99*, pages 150–164, 1999. Lecture Notes in Computer Science, Vol. 1578, Springer-Verlag.

[2] H. Comon. Sequentiality, monadic second-order logic and tree automata. *Information and Computation*, 157(1 & 2):25–51, 2000.

[3] H. Comon, M. Haberstrau, and J.-P. Jouannaud. Syntacticness, cycle-syntacticness, and shallow theories. *Information and Computation*, 111(1):154–191, 1994.

[4] M. Dauchet, A.-C. Caron, and J.-L. Coquide. Automata for reduction properties solving. *Journal of Symbolic Computation*, 20:215–233, 1995.

[5] I. Durand and A. Middeldorp. Decidable call by need computations in term rewriting (extended abstract). In *Proc. 14th International Conference on Automated Deduction, CADE-14*, pages 4–18, 1997. Springer LNAI 1249.

[6] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27:797–821, 1980.

[7] J.W. Klop. Term rewriting systems. In *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford Univ. Press, 1992.

[8] G. Kucherov and M. Tajine. Decidability of regularity and related properties of ground normal form languages. *Information and Computation*, 118:91–100, 1995.

[9] S. Lucas. Root-neededness and approximations of neededness. *Information Processing Letters*, 67(5):245–254, 1998.

[10] A. Middeldorp. Call by need computations to root-stable form. In *Proc. 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'97*, pages 94–105, 1997.

[11] T. Nagaya and Y. Toyama. Decidability for left-linear growing term rewriting systems. *Information and Computation*, 178(2):499–514, 2002.

[12] M. Oyamaguchi. The reachability and joinability problems for right-ground term-rewriting systems. *Journal of Information Processing*, 13(3):347–354, 1990.

[13] M. Oyamaguchi. NV-sequentiality: a decidable condition for call-by-need computations in term rewriting systems. *SIAM Journal on Computing*, 22(1):114–135, 1993.

[14] Y. Toyama. Membership conditional term rewriting systems. *Trans. IEICE*, E72(11):1224–1229, 1989. previously presented in *Proc. 1st International Workshop on Conditional Term Rewriting Systems, CTRS 1987*, pp.228-241, Springer LNCS 308, 1987.

Author Index