EPTCS 110

Proceedings of the

7th International Workshop on

Computing with Terms and Graphs

Rome, 23th March 2013

Edited by: Rachid Echahed and Detlef Plump

Published: 25th February 2013
DOI: 10.4204/EPTCS.110
ISSN: 2075-2180
Open Publishing Association

Preface

This volume contains the proceedings of the Seventh International Workshop on Computing with Terms and Graphs (TERMGRAPH 2013). The workshop took place in Rome, Italy, on March 23rd, 2013, as part of the sixteenth edition of the European Joint Conferences on Theory and Practice of Software (ETAPS 2013).

Research in term and graph rewriting ranges from theoretical questions to practical issues. Computing with graphs handles the sharing of common subexpressions in a natural and seamless way, and improves the efficiency of computations in space and time. Sharing is ubiquitous in several research areas, as witnessed by the modelling of first- and higher-order term rewriting by (acyclic or cyclic) graph rewriting, the modelling of biological or chemical abstract machines, and the implementation techniques of programming languages: many implementations of functional, logic, object-oriented, concurrent and mobile calculi are based on term graphs. Term graphs are also used in automated theorem proving and symbolic computation systems working on shared structures. The aim of this workshop is to bring together researchers working in different domains on term and graph transformation and to foster their interaction, to provide a forum for presenting new ideas and work in progress, and to enable newcomers to learn about current activities in term graph rewriting.

Previous editions of TERMGRAPH series were held in Barcelona (2002), in Rome (2004), in Vienna (2006), in Braga (2007), in York (2009) and in Saarbrücken (2011).

These proceedings contain six accepted papers and the abstracts of two invited talks. All submissions were subject to careful refereeing. The topics of accepted papers range over a wide spectrum, including theoretical aspects of term graph rewriting, concurrency, semantics as well as application issues of term graph transformation.

We would like to thank all who contributed to the success of TERMGRAPH 2013, especially the Program Committee for their valuable contributions to the selection process as well as the contributing authors. We would like also to express our gratitude to all members of the ETAPS 2013 organizing committee for their help in organizing TERMGRAPH 2013.

February, 2013

Rachid Echahed and Detlef Plump

Program chairs of TERMGRAPH 2013

R. Echahed and D. Plump (Eds.): 7th International Workshop on Computing with Terms and Graphs
EPTCS 110, 2013, pp. i–ii, doi:10.4204/EPTCS.110.0

Program committee of TERMGRAPH 2013

Patrick Bahr	University of Copenhagen, Denmark
Paolo Baldan	University of Padova, Italy
Frank Drewes	Umea University, Sweden
Rachid Echahed	CNRS, University of Grenoble, France (co-chair)
Maribel Fernandez	King's College London, UK
Clemens Grabmayer	Utrecht University, the Netherlands
Wolfram Kahl	McMaster University, Canada
Ian Mackie	Ecole Polytechnique, France
Detlef Plump	University of York, UK (co-chair)

Additional Reviewers

Thibaut Balabonski
Simon Gay
Dimitri Hendriks
Yuhang Zhao

Table of Contents

On the Concurrent Semantics of Transformation Systems with Negative Application Conditions

Joint GT-VMT and TERMGRAPH Invited Talk

Andrea Corradini
Dipartimento di Informatica
Università di Pisa, Italy

A rich concurrent semantics has been developed along the years for graph transformation systems, often generalizing in non-trivial ways concepts and results fist introduced for Petri nets. Besides the theoretical elegance, the concurrent semantic has potential applications in verification, e.g. in partial order reduction or in the use of finite prefixes of the unfolding for model checking. In practice (graph) transformation systems are often equipped with Negative Application Conditions, that describe forbidden contexts for the application of a rule. The talk will summarize some recent results showing that if the NACs are sufficiently simple (”incremental”) the concurrent semantics lifts smoothly to systems with NACs, but the general case requires original definitions and intuitions.

This is a joint work with Reiko Heckel, Frank Hermann, Susann Gottmann and Nico Nachtigall

R. Echahed and D. Plump (Eds.): 7th International Workshop on
Computing with Terms and Graphs
EPTCS 110, 2013, pp. 1–1, doi:10.4204/EPTCS.110.1

Programming Language Semantics using $\mathbb{K}$ - true concurrency through term graph rewriting -

Traian Florin Serbanuta
University Alexandru Ioan Cuza of Iasi *

Developed as a rewriting formalism for describing the operational semantics of programming languages, the $\mathbb{K}$ framework [8, 9, 10] proposes a new notation for (term) rewrite rules which identifies a read-only context (or interface) which is not changed by a rewrite rule. This allows for enhancing the transition system to model one-step concurrency with sharing of resources such as concurrent reads of the same memory location and concurrent writes of distinct memory locations.

Given that graph transformations offer theoretical support for achieving parallel rewriting with sharing of resources [2, 5, 4], and that the term-graph rewriting approaches were developed as sound and complete means of representing and implementing rewriting [1, 7, 3, 6], it seems natural to give semantics to $\mathbb{K}$ through term-graph rewriting. However, the existing term-graph rewriting approaches aim at efficiency: rewrite common subterms only once, without attempting to use context-sharing information for enhancing the potential for concurrency. Consequently, the concurrency achieved by current term-graph rewriting approaches is no better than that of standard rewriting. Moreover, the efficiency gained by sharing subterms can inhibit behaviors in non-deterministic (e.g., concurrent) systems.

This talk summarizes the efforts of endowing $\mathbb{K}$ with a (novel) term-graph rewriting semantics developed with the aim of capturing the intended concurrency the $\mathbb{K}$ framework [11]. Challenges encountered during this process and ideas for future development are presented and proposed for discussion.

References

[1] Hendrik Pieter Barendregt, Marko C. J. D. van Eekelen, John R. W. Glauert, Richard Kennaway, Marinus J. Plasmeijer & M. Ronan Sleep (1987): *Term Graph Rewriting*. In: *PARLE*, pp. 141–158.

[2] Hartmut Ehrig & Hans-Jörg Kreowski (1976): *Parallelism of Manipulations in Multidimensional Information Structures*. In: *MFCS*, *LNCS* 45, pp. 284–293.

[3] Annegret Habel, Hans-Jörg Kreowski & Detlef Plump (1987): *Jungle Evaluation*. In: *ADT*, *LNCS* 332, pp. 92–112.

[4] Annegret Habel, Jürgen Müller & Detlef Plump (2001): *Double-pushout graph transformation revisited*. *Mathematical Structures in Computer Science* 11(5), pp. 637–688. Available at `http://dx.doi.org/10.1017/S0960129501003425`.

[5] Hans-Jörg Kreowski (1977): *Transformations of Derivation Sequences in Graph Grammars*. In: *FCT'77*, pp. 275–286.

[6] Masahito Kurihara & Azuma Ohuchi (1995): *Modularity in Noncopying Term Rewriting*. *J. of TCS* 152(1), pp. 139–169. Available at `http://dx.doi.org/10.1016/0304-3975(94)00248-3`.

[7] Detlef Plump (1993): *Hypergraph rewriting: critical pairs and undecidability of confluence*. In: *Term graph rewriting: theory and practice*, John Wiley and Sons Ltd., pp. 201–213.

*This work was supported in part by the Contract 161/15.06.2010, SMISCSNR 602-12516 (DAK)

R. Echahed and D. Plump (Eds.): 7th International Workshop on Computing with Terms and Graphs
EPTCS 110, 2013, pp. 2–3, doi:10.4204/EPTCS.110.2

[8] Grigore Rosu (2003): *CS322, Fall 2003 - Programming Language Design: Lecture Notes*. Technical Report UIUCDCS-R-2003-2897, University of Illinos at Urbana Champaign. Lecture notes of a course taught at UIUC.

[9] Grigore Rosu & Traian Florin Serbanuta (2010): *An Overview of the K Semantic Framework*. *J. of Logic and Algebraic Programming* 79(6), pp. 397–434. Available at `http://dx.doi.org/10.1016/j.jlap.2010.03.012`.

[10] Traian Florin Şerbănuţă (2010): *A Rewriting Approach to Concurrent Programming Language Design and Semantics*. Ph.D. thesis, University of Illinois.

[11] Traian Florin Serbanuta & Grigore Rosu (2012): *A Truly Concurrent Semantics for the K Framework Based on Graph Transformations*. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski & Grzegorz Rozenberg, editors: *ICGT*, *Lecture Notes in Computer Science* 7562, Springer, pp. 294–310. Available at `http://dx.doi.org/10.1007/978-3-642-33654-6_20`.

Non-simplifying Graph Rewriting Termination

Guillaume Bonfante
LORIA
Université de Lorraine

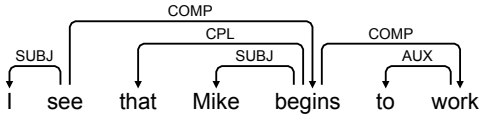
Bruno Guillaume
LORIA
Inria Nancy Grand-Est

So far, a very large amount of work in Natural Language Processing (NLP) rely on trees as the core mathematical structure to represent linguistic informations (e.g. in Chomsky's work). However, some linguistic phenomena do not cope properly with trees. In a former paper, we showed the benefit of encoding linguistic structures by graphs and of using graph rewriting rules to compute on those structures. Justified by some linguistic considerations, graph rewriting is characterized by two features: first, there is no node creation along computations and second, there are non-local edge modifications. Under these hypotheses, we show that uniform termination is undecidable and that non-uniform termination is decidable. We describe two termination techniques based on weights and we give complexity bound on the derivation length for these rewriting systems.

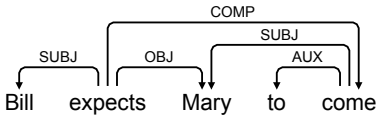
1 Introduction

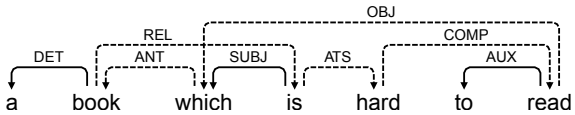
Linguists introduce different levels to describe a natural language sentence. Starting from a sentence given as a sequence of sounds or as a sequence of words; among the linguistic levels, two are deeply considered in literature: the syntactic level (a grammatical analysis of the sentence) and the semantic level (a representation of the meaning of the sentence). These two representations involve mathematical structures such as logical formulae, λ-terms, trees and graphs.

One of the usual ways to describe syntax is to use the notion of dependency [16]. A dependency structure is an ordered sequence of words, together with some relations between these words. For instance, the sentence *"I see that Mike begins to work"* can be represented by the structure on the right.



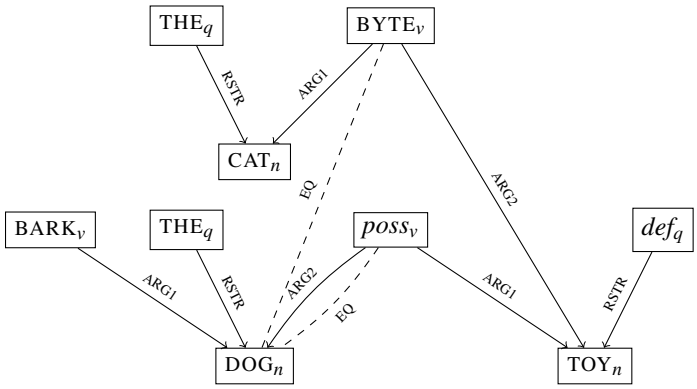
There is a large debate in the literature about the mathematical nature of the structures needed for natural language syntax: do we have to consider trees or graphs? Trees are often considered for their simplicity; however, it is clearly insufficient. Let us illustrate the limitations of tree-representations with some linguistic examples. Consider the sentence *"Bill expects Mary to come"*, the node *"Mary"* is shared, being the subject of *"come"* and the object of *"expects"* (below on the left). The situation can be even worse: cycles may appear such as in the sentence below where edges in the cycle are drawn with dashed line (below on the right).





R. Echahed and D. Plump (Eds.): 7th International Workshop on
Computing with Terms and Graphs
EPTCS 110, 2013, pp. 4–16, doi:10.4204/EPTCS.110.3

For the semantic representation of natural language sentences, first order logic formulae are widely used. To deal with natural language ambiguity, a more compact representation of a set of logic formulae (called underspecified semantic representation) is used. DMRS [4] is one of these compact representation. The DMRS structure for the sentence *"The Dog whose toy the cat bit barked"* is given in the figure on the right.



To describe transformations between syntactic and semantics structures, there are solutions based on many computational models (finite state automata, λ-calculus). It is somewhat surprising that Graph Rewrite Systems (GRS) have been hardly considered so far ([8, 1, 5, 9]). To explain that, GRS implementations are usually considered to be too inefficient to justify their extra-generality. For instance, pattern matching does not take linear time where this is usually seen as an upper limit for fast treatment.

However, if one drops for a while the issue of efficiency, the use of GRS is promising. Indeed, linguistic considerations can be most of the time expressed by some relations between a few words. Thus, they are easily translated into rules. To illustrate this point, in [3, 12], we proposed a syntax to semantics translator based on GRSs: given the syntax of a sentence, it outputs the different meaning associated to this syntax.

In the two earlier mentioned studies, we tried to delineate what are the key features of graph rewriting in the context of NLP. Roughly speaking, node creation are strictly restricted, edges may be shifted from one node to another and there is a need for negative patterns. Based on this analysis, we define here a suitable framework for NLP (see Section 3).

Compared to term rewriting, the semantics of graph rewriting is problematic: different choices can be made in the way the context is glued to the rule application [15]. As far as we see, our notion does not fit properly the DPO approach due to unguarded node deletion nor the SPO approach due to the shift command, as we shall see. Thus we will provide a complete description of our notion. We have chosen to present it in an operational way and we leave for future work a categorial semantics.

In our application, we use several hundreds of rules. To manage such a system, we use a notion of modular graph rewriting system: the full set of rules is divided in smaller subsets (called modules) that are used in turn.

In practice, we need some tools to verify termination and confluence properties of modules. In Section 4, we provide two termination methods based on a weight analysis. First, there is a direct motivation: in our NLP application, any computations should terminate. If it is not the case, it means that the rules where not correctly drawn. Then, termination ensures partly the correctness of the transformation. There is also an indirect reason to consider termination: one way of establishing confluence is through Newman's Lemma [11] which requires termination.

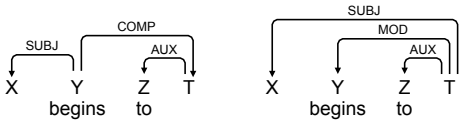
We consider two properties of the above mentioned termination methods. First, we show that they are decidable, that is the existence of weights can be computed statically from the rules, and thus we have a fully automatic tool to verify termination. Obviously, it is not complete. In a second step, we evaluate the strength of the two methods. To do that, we consider what restrictions they impose on the length of computations. We get quadratic time for the first method, polynomial time for the second. This article is an extended abstract of [2].

2 Linguistic motivations

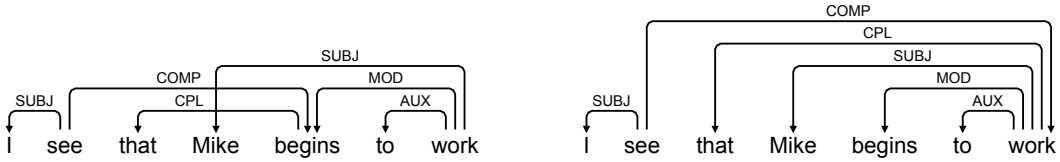
Without any linguistic exhaustivity, we highlight in this section some crucial points of the kind of linguistic transformation we are interested in and hence the relative features of rewriting we have to consider.

Node preservation property. As linguistic examples above suggest, the goal of linguistic analysis is mainly to describe different kinds of relations between elements that are present in the input structure. As a consequence, the set of nodes in the output structure is directly predictable from the input and only a very restrictive notion on node creation is needed. In practice, these node creations can be anticipated in some enriched input structure on which the whole transformation can be described as a non-size increasing process.

Edge shifting. In the first example of the introduction (for the sentence *"I see that Mike begins to work"*), the verb *"begins"* is called a raising verb and we know that *"Mike"* is the *deep subject* of the verb *"work"*; *"begins"* being considered as a modifier of the verb. To recover this deep subject, one may imagine a local transformation of the graph which turns the first graph on the right into the second one.

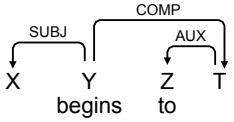


However, in our example above, a direct application of such a transformation leads to the structure below on the left which is not the right structure. Indeed, the transformation should shift what the linguists call the head of the phrase *"Mike begins to work"* from the word *"begins"* to the word *"work"* with all relative edges. In that case, the transformation should produce the structure below on the right:



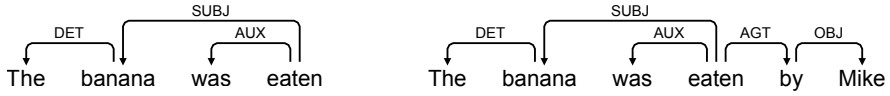
In a more general setting, our transformations may have to specify the fact that all incident edges of some node X must be transported to some other node Y. We call this operation `shift`.

To describe our graph rewriting rules, we introduce a system of commands (like in [6]) which expresses step by step the modifications applied on the input graph. The transformation described above is performed in our setting as follows:



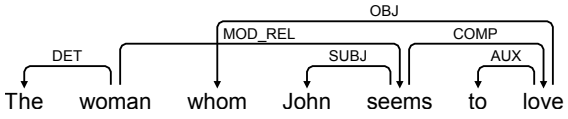
`del_edge` (Y,SUBJ,X); `del_edge` (Y,COMP,T);
`add_edge` (T,SUBJ,X); `add_edge` (T,MOD,Y);
`shift` (Y,T)

Negative conditions. In some situation, rules must be aware of the context of the pattern to avoid unwanted ambiguities. When computing semantics out of syntax, one has to deal with passive sentence; the two sentences below show that the agent is optional.

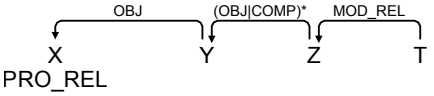


In order to switch to the corresponding active form, two different linguistic transformations have to be defined for these two sentence; but, clearly, the first graph is a subgraph of the second one. We don't want the transformation for the short passive on the left to apply on the long passive on the right. we need to express a negative condition like "there is no out edge labeled by AGT out of the main verb" to prevent the unwanted transformation to occur.

Long distance dependencies. Most of the linguistic transformation can be expressed with successive local transformation like the one above. Nevertheless, there are some cases where more global rewriting is required; consider the sentence *"The women whom John seems to love"*, for which we consider the syntactic structure on the right. One of the steps in the semantic construction of this sentence requires to compute the antecedent of the relative pronoun *"whom"* (the noun *"woman"* in our example).

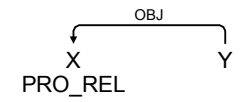


The subgraph we have to search in our graph (which is depicted as a non-local pattern) and the graph modification to perform are given on the right. The number of OBJ or COMP relations to consider (in the relation depicted as (OBJ|COMP)* in the figure) is unbounded (in linguistics, this phenomenon is called long distance dependencies); it is possible to construct grammatical sentences with an arbitrary large number of relations.

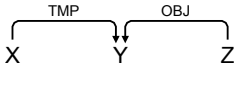


`add_edge` (X,ANT,T)

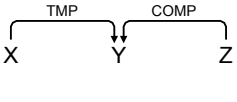
As we want to stay in the well-known framework of local rewriting, we will use several local transformations to implement such a non-local rule.



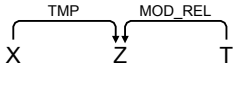
`add_edge` (X,TMP,Y)



`del_edge` (X,TMP,Y)
`add_edge` (X,TMP,Z)



`del_edge` (X,TMP,Y)
`add_edge` (X,TMP,Z)



`del_edge` (X,TMP,Z)
`add_edge` (X,ANT,T)

The second and the third rules above preserve the set of nodes and the number of edges of each kind. Hence, this kind of rule will require special treatment with respect to termination issues.

3 Graph Rewriting for NLP

Before we enter into the technical sections, let us define some useful notations. First, we use the notation $\vec{c}$ to denote sequences. The empty sequence is written $\emptyset$. The length of a sequence is denoted by $|\vec{c}|$. We use the same notation for sets: the empty set is denoted $\emptyset$ and the cardinality of a set S is written $|S|$. The context will make clear whether we are talking about sequences or sets.

Given a function $f : X \to Y$ and some sets $X' \subseteq X$ and $Y' \subseteq Y$, we define $f(X') \triangleq \{f(x) \mid x \in X'\}$ and $f^{-1}(Y') \triangleq \{x \in X \mid f(x) \in Y'\}$; the restriction of the function f to the domain X' is $f|_{X'} : x' \in X' \mapsto f(x')$. The function $\mathbf{c}_X : x \in X \mapsto c \in Y$ is the constant function on X. The identity function is written $\mathbb{1}$. Finally, given a function $f : X \to Y$ and $(x,y) \in X \times Y$, the function $f[x \mapsto y]$ maps $t \neq x$ to $f(t)$ and x to y.

The set of natural numbers is $\mathbb{N}$, integers are denoted by $\mathbb{Z}$. Given two integers a,b, we define $[a,b] = \{x \in \mathbb{Z} \mid a \leq x \leq b\}$.

3.1 Graphs

The graphs we consider are directed graphs with both labels on nodes and labels on edges. We restrict the edge set: given some edge label e, there is at most one edge labeled e between two given nodes α and β. This restriction reflects the fact that, in NLP application, our edges are used to encode linguistic information which are relations. We make no other explicit hypothesis on graphs: in particular, graphs may be disconnected, or have loops.

In this paper, we suppose given a finite set Σ_E of edge labels and another finite set Σ_N of node labels.

Definition 3.1 (Graph)**.** *A* graph *G is defined as a triple $(\mathcal{N},\ell,\mathcal{E})$ where*

- *$\mathcal{N}$ is a finite set of nodes;*
- *ℓ is a labeling function: $\ell : \mathcal{N} \mapsto \Sigma_N$;*
- *$\mathcal{E}$ is a set of edges: $\mathcal{E} \subseteq \mathcal{N} \times \Sigma_E \times \mathcal{N}$.*

Let $n,m \in \mathcal{N}$ and $e \in \Sigma_E$. When there is an edge from n to m labelled e (*i.e.* $(n,e,m) \in \mathcal{E}$), we write $n \xrightarrow{e} m$ or $n \longrightarrow m$ if the edge label is not relevant. If G denotes some graph $(\mathcal{N},\ell,\mathcal{E})$, then $\mathcal{N}_G$, ℓ_G, $\mathcal{E}_G$ denote respectively $\mathcal{N},\ell$ and $\mathcal{E}$.

Definition 3.2 (Graph morphism)**.** *A* graph morphism *μ from the graph $G = (\mathcal{N},\ell,\mathcal{E})$ to the graph $G' = (\mathcal{N}',\ell',\mathcal{E}')$ is a function from $\mathcal{N}$ to $\mathcal{N}'$ such that:*

- *for all $n \in \mathcal{N}$, $\ell'(\mu(n)) = \ell(n)$;*
- *for all $n,m \in \mathcal{N}$ and $e \in \Sigma_E$, if $n \xrightarrow{e} m \in \mathcal{E}$ then $\mu(n) \xrightarrow{e} \mu(m) \in \mathcal{E}'$.*

A graph morphism μ is said to be injective *if $\mu(n) = \mu(m)$ implies $n = m$. We make the following abuse of notation: given some graph morphism $\mu : G \to G'$, and a set $E \subseteq \mathcal{E}_G$, we let $\mu(E) = \{\mu(n) \xrightarrow{e} \mu(m) \mid n \xrightarrow{e} m \in E\}$.*

Definition 3.3 (Basic pattern and basic matching)**.** *A* basic pattern *B is a graph. A* basic matching *μ of the basic pattern B in the graph G is an* injective *graph morphism μ (written $\mu : B \hookrightarrow G$).*

As shown in Section 2, negative conditions on patterns naturally arise in NLP. We classify negative conditions in two categories: the local ones, that is negative conditions on edges within the basic pattern and non-local ones, that is negative conditions concerning edges between a node of the basic pattern and a node of the context (either in-edges or out-edges).

Definition 3.4 (Pattern)**.** *A* pattern *is a quadruple $P = (B,\bar{\mathcal{E}},\bar{\mathcal{I}},\bar{\mathcal{O}})$ of:*

- *a basic pattern $B = (\mathcal{N}_P,\ell_P,\mathcal{E}_P)$;*
- *a set of forbidden edges $\bar{\mathcal{E}} \subset \mathcal{N}_P \times \Sigma_E \times \mathcal{N}_P$ such that $\bar{\mathcal{E}} \cap \mathcal{E}_P = \emptyset$;*
- *a set of forbidden in-edges $\bar{\mathcal{I}} \subset \mathcal{N}_P \times \Sigma_E$*
- *a set of forbidden out-edges $\bar{\mathcal{O}} \subset \mathcal{N}_P \times \Sigma_E$*

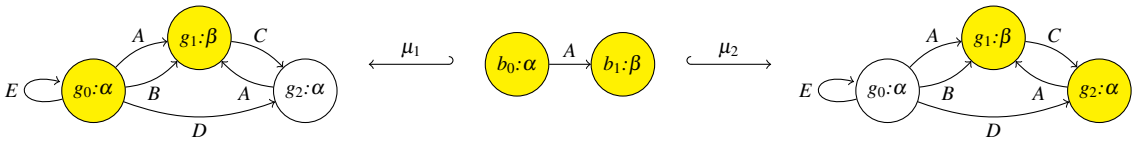
Given a basic pattern B, we shorten $(B,\emptyset,\emptyset,\emptyset)$ to $(B,\vec{\emptyset})$. In the following, given a pattern P, $\mathcal{N}_P$ and $\mathcal{E}_P$ denote respectively the set of nodes of its basic pattern and the set of edges of its basic pattern.

Definition 3.5 (Matching)**.** *Let $P = (B,\bar{\mathcal{E}},\bar{\mathcal{I}},\bar{\mathcal{O}})$ be a pattern, $G = (\mathcal{N},\ell,\mathcal{E})$ be a graph, and $\mu : B \hookrightarrow G$ be a basic matching. We say that μ is a* matching *from P into G (also written $\mu : P \hookrightarrow G$) whenever it satisfies the additional three conditions:*

- $\mu(\bar{\mathcal{E}}) \cap \mathcal{E} = \emptyset$

- *for each* $(n,e) \in \bar{\mathscr{I}}$, $\{p \in \mathscr{N} \setminus \mu(\mathscr{N}_P) \mid p \xrightarrow{e} \mu(n)\} = \emptyset$
- *for each* $(n,e) \in \bar{\mathscr{O}}$, $\{p \in \mathscr{N} \setminus \mu(\mathscr{N}_P) \mid \mu(n) \xrightarrow{e} p\} = \emptyset$

Example 3.1. *Negative conditions are used to remove 'unwanted' matching. To see their effect, consider for instance the basic pattern* B_0 *and its two basic matchings* μ_1 *and* μ_2 *in* G_0*:*



- *First, let* $P_0 = (B_0, \vec{\emptyset})$*. Then,* μ_1 *and* μ_2 *are (the) two matchings* $P_0 \hookrightarrow G_0$*.*
- *Second, let the pattern* $P_1 = (B_0, \{(b_1, C, b_0)\}, \emptyset, \emptyset)$*; then,* μ_1 *is the only matching* $P_1 \hookrightarrow G_0$*.*
- *Third, let the pattern* $P_2 = (B_0, \emptyset, \{(b_0, D)\}, \{(b_0, D)\})$*. Then, there is no matching of* P_2 *into* G_0*.*

In the following, patterns P_1 *and* P_2 *are depicted as:*

3.2 Graph decomposition

The proper description of actions of a rule on some graph G requires first the definition of two partitions: one on nodes and the other on edges. They are both induced by the matching of some pattern P into G.

Definition 3.6 (Nodes decomposition: pattern image, crown and context)**.** *Let* $\mu : P \hookrightarrow G$ *a matching from the pattern* P *into the graph* $G = (\mathscr{N}, \ell, \mathscr{E})$*. Nodes of* G *can be split in a partition of three sets* $\mathscr{N} = \mathscr{P}_\mu \oplus \mathscr{K}_\mu \oplus \mathscr{C}_\mu$*:*

- *the* pattern image *is* $\mathscr{P}_\mu = \mu(\mathscr{N}_P)$*;*
- *the* crown *contains nodes outside the pattern image which are directly connected to the pattern image:* $\mathscr{K}_\mu = \{n \in \mathscr{N} \setminus \mathscr{P}_\mu \mid \exists p \in \mathscr{P}_\mu$ such that $n \longrightarrow p$ or $p \longrightarrow n\}$*;*
- *the* context *contains nodes not linked to the pattern image:* $\mathscr{C}_\mu = \mathscr{N} \setminus (\mathscr{P}_\mu \cup \mathscr{K}_\mu)$*.*

Definition 3.7 (Edges decomposition: pattern edges, crown edges, context edges and pattern-glued edges)**.** *Let* $\mu : P \hookrightarrow G$ *a matching from the pattern* P *into the graph* $G = (\mathscr{N}, \ell, \mathscr{E})$*. Edges of* G *can be split in a partition of four sets* $\mathscr{E} = \mu(\mathscr{E}_P) \oplus \mathscr{K}^\mu \oplus \mathscr{C}^\mu \oplus \mathscr{H}^\mu$*:*

- *the* pattern edges *is* $\mu(\mathscr{E}_P)$*;*
- *the* crown edges *set contains edges which links a pattern image node to a crown node:* $\mathscr{K}^\mu = \{n \longrightarrow m \in \mathscr{E} \mid n \in \mathscr{P}_\mu \wedge m \in \mathscr{K}_\mu\} \cup \{n \longrightarrow m \in \mathscr{E} \mid n \in \mathscr{K}_\mu \wedge m \in \mathscr{P}_\mu\}$*;*
- *the* context edges *set contains edges which connect two nodes that are not in the pattern image:* $\mathscr{C}^\mu = \{n \longrightarrow m \in \mathscr{E} \mid n \notin \mathscr{P}_\mu \wedge m \notin \mathscr{P}_\mu\}$*.*
- *the* pattern-glued edges *set contains edges which are not pattern edges but which connect two nodes that are in the pattern image:* $\mathscr{H}^\mu = \{n \longrightarrow m \in \mathscr{E} \mid n \in \mathscr{P}_\mu \wedge m \in \mathscr{P}_\mu\} \setminus \mu(\mathscr{E}_P)$*.*

3.3 Rules

In our graph rewriting framework, the transformation of the graph is described through some atomic commands (like in [6]). Commands definition refer to some pattern P and pattern nodes $\mathscr{N}_P$ are used as identifiers. Let $a,b \in \mathscr{N}_P$, $\alpha \in \Sigma_N$ and $e \in \Sigma_E$, the five kinds of commands are $\texttt{label}(a,\alpha)$, $\texttt{del\_edge}(a,e,b)$, $\texttt{add\_edge}(a,e,b)$, $\texttt{del\_node}(a)$ and $\texttt{shift}(a,b)$.

Their names speak for themselves, however, we will come back to their precise meaning in the subsection below. Before this, to ensure that commands always refer to valid node identifiers, we restrict command sequences to *consistent* sequences, that is sequences $c_1,\ldots,c_k$ such that for each command c_i, $1 \leq i \leq k$, which is a node deletion command $\texttt{del\_node}(a)$ for some $a \in \mathscr{N}_P$, then the node name a does not occur in any command c_j with $i < j \leq k$.

Definition 3.8 (Rule)**.** *A* rule *R is a pair $R = \langle P, \vec{c} \rangle$ of a pattern P and a sequence of commands $\vec{c}$ consistent with respect to P. A rule $R = \langle P, \vec{c} \rangle$ is said to be* node-preserving *if $\vec{c}$ does not contain any* `del_node` *command.*

3.4 Graph Rewrite System

Let $G = (\mathscr{N}, \ell, \mathscr{E})$ a graph, $R = \langle P, \vec{c} \rangle$ a rule and $\mu : P \hookrightarrow G$ a matching. The application of the sequence $\vec{c}$ on G is a new graph which is written $G \cdot_\mu \vec{c}$ (shortened $G \cdot \vec{c}$ when μ is clear from the context) and is defined by induction on the length k of $\vec{c}$. If $k = 0$, $G \cdot \emptyset = G$. If $k > 0$, let $G' = (\mathscr{N}', \ell', \mathscr{E}')$ be the graph obtained by application of the sequence $c_1,\ldots,c_{k-1}$; then we consider each command in turn:

Label: The command $c_k = \texttt{label}(a,\alpha)$ changes the label of the node $\mu(a)$: $G \cdot \vec{c} = (\mathscr{N}', \ell'', \mathscr{E}')$ with $\ell'' = \ell'[\mu(a) \mapsto \alpha]$.

Delete: The command $c_k = \texttt{del\_edge}(a,e,b)$ deletes the edge from $\mu(a)$ to $\mu(b)$ labelled with $e \in \Sigma_E$; $G \cdot \vec{c} = (\mathscr{N}', \ell', \mathscr{E}'')$ with $\mathscr{E}'' = \mathscr{E}' \setminus \{\mu(a) \xrightarrow{e} \mu(b)\}$.

Add: The command $c_k = \texttt{add\_edge}(a,e,b)$ adds an edge from $\mu(a)$ to $\mu(b)$ labelled with $e \in \Sigma_E$; $G \cdot \vec{c} = (\mathscr{N}', \ell', \mathscr{E}'')$ with $\mathscr{E}'' = \mathscr{E}' \cup \{\mu(a) \xrightarrow{e} \mu(b)\}$.

Delete node: The command $c_k = \texttt{del\_node}(a)$ removes the node $\mu(a)$ of G'; $G \cdot \vec{c} = (\mathscr{N}'', \ell'', \mathscr{E}'')$ with $\mathscr{N}'' = \mathscr{N}' \setminus \{\mu(a)\}$, $\ell'' = \ell'|_{\mathscr{N}''}$ and $\mathscr{E}'' = \mathscr{E}' \cap \{\mathscr{N}'' \times \Sigma_E \times \mathscr{N}''\}$.

Shift edges: The command $c_k = \texttt{shift}(a,b)$ changes in-edges of $\mu(a)$ starting from the crown to in-edges of $\mu(b)$ and all out-edges of $\mu(a)$ going to the crown to out-edges of $\mu(b)$. Formally, $G \cdot \vec{c} = (\mathscr{N}', \ell', \mathscr{E}'')$ with the set $\mathscr{E}''$ defined by, for all $e \in \Sigma_E$:

- for all $p \in \mathscr{K}_\mu$, $\mu(b) \xrightarrow{e} p \in \mathscr{E}''$ iff $\mu(b) \xrightarrow{e} p \in \mathscr{E}'$ or $\mu(a) \xrightarrow{e} p \in \mathscr{E}'$;
- for all $p \in \mathscr{K}_\mu$, $p \xrightarrow{e} \mu(b) \in \mathscr{E}''$ iff $p \xrightarrow{e} \mu(b) \in \mathscr{E}'$ or $p \xrightarrow{e} \mu(a) \in \mathscr{E}'$;
- for all $p,q \in \mathscr{P}_\mu$, $p \xrightarrow{e} q \in \mathscr{E}''$ iff $p \xrightarrow{e} q \in \mathscr{E}'$;
- for all $p,q \in \mathscr{K}_\mu \cup \mathscr{C}_\mu$, $p \xrightarrow{e} q \in \mathscr{E}''$ iff $p \xrightarrow{e} q \in \mathscr{E}'$.

The commands `label`, `del_edge` and `add_edge` are called local commands: they modify only the edges and the nodes described in the pattern. The commands `del_node` and `shift` are non-local: they can modify edges outside the pattern. Note that a rule `add_edge` (resp. `del_edge`) may have no effect if the edge already exists (resp. does not exist). Note also that we can suppose that for a given sequence $\vec{c}$ and a given triple (a,e,b), there is at most one rule $\texttt{del\_edge}(a,e,b)$ or $\texttt{add\_edge}(a,e,b)$ in $\vec{c}$ (if not, only the last one is effective). Hence, we can define uniform rules:

Definition 3.9 (Uniform rule). *For $\vec{c} = c_1, \ldots, c_k$ without node deletion, the rule $\langle P, \vec{c} \rangle$ is* uniform *iff for all $1 \leq i \leq k$, if $c_i = \texttt{add\_edge}(n,e,m)$ then $(n,e,m) \in \bar{\mathscr{E}}_P$ and if $c_i = \texttt{del\_edge}(n,e,m)$ then $(n,e,m) \in \mathscr{E}_P$.*

Definition 3.10 (Rewrite step). *Let $G = (\mathscr{N}, \ell, \mathscr{E})$ a graph, $R = \langle P, \vec{c} \rangle$ a rule and $\mu : P \hookrightarrow G$ a matching. Let $G' = G \cdot \vec{c}$, then we say that G rewrites to G' with respect to the rule R and the matching μ. We write it $G \to_{R,\mu} G'$ or $G \to_R G'$ or even simply $G \to G'$.*

Definition 3.11 (Graph Rewrite System). *A* Graph Rewrite System *$\mathscr{G}$ is a finite set of rules.*

In our application, the translation of the syntax to semantics is split into several independent levels of transformation driven by linguistic consideration (such as translation of passive forms to active ones, computation of the deep subject of infinites). Rules are then grouped in subsets called *modules* and modules apply sequentially; each module being used as a graph rewrite system on the outputs of the previous module.

Lemma 3.1 (Linear modification). *Given a GRS $\mathscr{G}$, there is a constant $C > 0$ such that, for any rewriting step $G \to_{R,\mu} G'$ the two canonical corresponding edge decompositions $\mathscr{E}_G = \mathscr{C}^\mu \oplus \mathscr{Q}^\mu$ and $\mathscr{E}_{G'} = \mathscr{C}^\mu \oplus \mathscr{Q}'^\mu$ satisfy:*

$$|\mathscr{Q}^\mu| \leq C \times (|G|+1) \text{ and } |\mathscr{Q}'^\mu| \leq C \times (|G|+1)$$

Proof. Let $C = \max\{2 \times |P|^2 \times |\Sigma_E|\} \mid \langle P, \vec{c} \rangle \in \mathscr{G}\}$. Both in G and G', edges that are not in the context are either between two pattern nodes or between a pattern node and a crown node. The total number of edges of the first kind (either pattern edges or glued-pattern edges) is bounded by $|P|^2 \times |\Sigma_E|$. For each pattern node, the number of edges which connect this node to some non-pattern node is bounded by $2 \times |G| \times |\Sigma_E|$ and so the total number of edges which link some pattern node to some non-pattern node is bounded by $2 \times |G| \times |\Sigma_E| \times |P|$. Putting everything together, $|\mathscr{Q}^\mu| \leq C \times (|G|+1)$ and $|\mathscr{Q}'^\mu| \leq C \times (|G|+1)$. □

4 Weighted GRS

We recall that a GRS is said to be (strongly) terminating whenever there is no infinite sequence $G_1 \to G_2 \to \cdots$. Given a terminating GRS $\mathscr{G}$ and a graph G, we define the derivation height of G, next denoted $h_{\mathscr{G}}(G)$, to be the length of the longest derivation starting from G if such a derivation exists. If $h_{\mathscr{G}}(G)$ is defined for all G such that $|G| \leq n$, then we define the derivation height of $\mathscr{G}$ by: $h_{\mathscr{G}}(n) = \max\{h_{\mathscr{G}}(G) \mid |G| \leq n\}$.

Actually, for non-size increasing GRS as presented above, we have immediately the decidability of non-uniform termination. That is, given some GRS $\mathscr{G}$ and some graph G, one may decide whether there is an infinite sequence $G_1 \to G_2 \to G_3 \to \cdots$. Indeed, one may observe that for such sequence, for all $i \in \mathbb{N}$, $|G_i| \leq |G|$. Thus, the G_i's range in the finite set $\mathfrak{G}_{\leq |G|}$ of graphs of size less or equal to $|G|$. Consequently, either the system terminates or there is some $j \leq |\mathfrak{G}_{\leq |G|}|$ and some $k \leq j$ such that $G_j = G_k$. To conclude, to decide non-uniform termination, it is sufficient to compute all the (finitely many) possibilities of rewriting G in less than $|\mathfrak{G}_{\leq |G|}|$ steps and to verify the existence of such a j and k above. Finally, since $|\mathfrak{G}_{\leq |G|}| \leq 2^{O(|G|^2)}$, the procedure as described above takes exponential time.

However, uniform termination— given a GRS, is it terminating?— of non-size increasing GRS remains an open problem. Uniform termination was proved undecidable when we drop the property of non-size increasingness (cf. Plump [14]). As a consequence, there is a need to define some termination method pertaining to non-size increasing GRS. Compared to standard work in termination [13, 7], there are two difficulties: first, our graphs may be cyclic, thus forbidding methods developed for DAGs such as term-graphs. Second, using term rewriting terminology, our method should operate for some

non-simplifying GRS, that is GRS for which the output may be "bigger" than the input. Indeed, the NLP programmer sometimes wants to compute some *new* relations, so that the input graph is a strict sub-graph of the resulting graph.

4.1 Termination by weight analysis

In the context of term-rewriting systems, the use of weights is very common to prove termination. There are many examples of such orderings, Knuth-Bendix Ordering [10] to cite one of them. We recall that all graphs we consider are defined relatively to two signatures Σ_E of edge labels and Σ_N of node labels.

Definition 4.1 (Edge weight, node weight)**.** *An* edge weight *is a mapping $w : \Sigma_E \to \mathbb{Z}$. Given some subset E of edges of G, the weight of E is $w(E) = \sum_{n \xrightarrow{e} m \in E} w(e)$. The edge weight of a graph G is $w(G) = w(\mathscr{E}_G)$. A* node weight *is a mapping $\eta : \Sigma_N \to \mathbb{Z}$. For a graph $G = (\mathscr{N}_G, \ell_G, \mathscr{E}_G)$, we define $\eta(G) = \sum_{n \in \mathscr{N}_G} \eta(\ell_G(n))$.*

Let us make some observations. Let $|G|_e$ denote the number of edges in G which have the label e, then $w(G) = \sum_{e \in \Sigma_E} w(e) \times |G|_e$. Second, for a pattern matching $\mu : P \hookrightarrow G$, $w(\mu(P)) = w(P)$.

The weight of a graph may be negative. This is not standard, but it is useful here to cope with non-simplifying rules, that is rules which add new edges. Since a graph G has at most $|\Sigma_E| \times |G|^2$ edges, the following lemma is immediate.

Lemma 4.1. *Given an edge weight w and a node weight η, let $K_w = \max_{e \in \Sigma_E}(|w(e)|)$, $K_E = |\Sigma_E| \times K_w$, $K_\eta = \max_{\alpha \in \Sigma_N}(|\eta(\alpha)|)$, then*

(a) for each subset of edges $E \subset \mathscr{E}_G$ of some graph G, we have $w(E) \leq K_w \times |E|$.

(b) for each graph G, we have $-K_E \times |G|^2 \leq w(G) \leq K_E \times |G|^2$;

(c) for each graph G, we have $|\eta(G)| \leq K_\eta \times |G|$.

Definition 4.2. *Let $R = \langle P, \vec{c} \rangle$ a rule, we define inductively $\Phi_{\vec{c}} : \mathscr{N}_P \to \mathscr{N}_P$ which describes the global effect of the* `shift` *commands in a rule: $\Phi_\emptyset = \mathbb{1}$; $\Phi_{\vec{c},\texttt{shift}(m,n)} = \mathbb{1}[m \mapsto n] \circ \Phi_{\vec{c}}$ and $\Phi_{\vec{c},c} = \Phi_{\vec{c}}$ if c is not a* `shift` *command.*

Definition 4.3 (Compatible weight)**.** *Given a rule $R = \langle (P, \bar{\mathscr{E}}, \bar{\mathscr{I}}, \bar{\mathscr{O}}), \vec{c} \rangle$, an edge weight w is said to be* compatible *with R if:*

1. *either $\vec{c}$ contains a* `del_node` *command*
2. *or R is a node-preserving rule and satisfy the three properties:*
   - *(a) R is uniform,*
   - *(b) $w(P \cdot_{\mathbb{1}} \vec{c}) < w(P)$,*
   - *(c) for all $e \in \Sigma_E$ such that $w(e) < 0$, for all $n \in \Phi(\mathscr{N}_P)$, let $M_n \subset \mathscr{E}_P$ be the set $\Phi_{\vec{c}}^{-1}(n)$; then M_n contains at most one element m such that $(m, e) \notin \bar{\mathscr{I}}$ and M_n contains at most one element $m' \in M_n$ such that $(m', e) \notin \bar{\mathscr{O}}$.*

An edge weight is said to be compatible with a GRS $\mathscr{G}$ if it is compatible with all its rules. A weighted GRS is a pair $(\mathscr{G}, w)$ of a GRS and a compatible weight.

Hypothesis (2.b) will serve to manage edges in the pattern images while Hypothesis (2.c) will serve for the crown edges. One may note that when there is no `shift` commands in the rule, the Hypothesis (2.c) holds whatever w is. Indeed, in that case, Φ is the identity function and all the sets M_n are singletons.

Lemma 4.2. *Let $(\mathscr{G},w)$ a weighted GRS, let $G \to G'$ be a rewrite step of $\mathscr{G}$. Either $|G| > |G'|$ or $|G| = |G'|$ and $w(G) > w(G')$.*

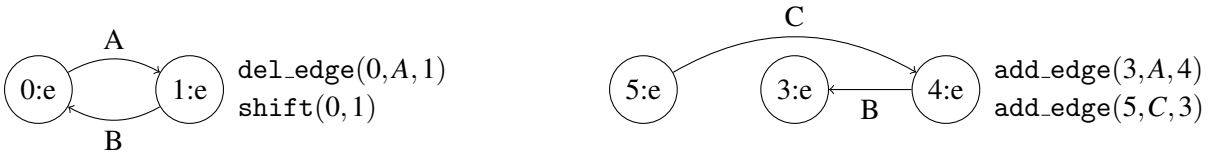
The problem of the synthesis is the following. Given a GRS $\mathscr{G}$, is there a weight w compatible with $\mathscr{G}$? Since the existence of weights can be described in Presburger's arithmetic, we have a positive answer:

Theorem 4.1. *Given a GRS $\mathscr{G}$, one may decide whether or not it has a compatible weight.*

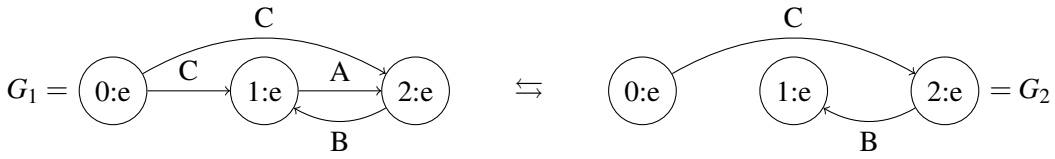
Second point, the existence of weights induce termination:

Theorem 4.2. *Any weighted GRS $(\mathscr{G},w)$ is strongly terminating in quadratic time. Moreover, this quadratic bound is a lower bound: there is a GRS $\mathscr{G}$ with a compatible weight such that $h_{\mathscr{G}}(n) \geq O(n^2)$.*

Condition (2.c) of Definition 4.3 is necessary. Here is a counter-example of a non-terminating system with a compatible weight up to this condition. Consider the two rules $\langle Q_1, \vec{c}_1 \rangle$ and $\langle Q_2, \vec{c}_2 \rangle$:



Set $w(A) = w(B) = 1$ and $w(C) = -2$. Observe that $w(Q_1 \cdot \vec{c}_1) = 1 < 2 = w(Q_1)$ and $w(Q_2 \cdot \vec{c}_2) = -2 < -1 = w(Q_2)$. However, there is an infinite sequence $G_1 \to_{R_1} G_2 \to_{R_2} G_1 \to_{R_1} \cdots$ with G_1 and G_2 being:



Proof sketch of Theorem 4.2. We begin to show the lower bound. Let $\Sigma_E = \{E\}$, $\Sigma_N = \{e\}$. Consider the two rules GRS $\mathscr{G}$ defined by the two basic patterns:

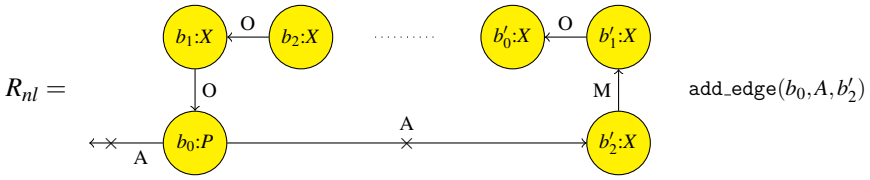
Set $w(E) = 1$. The rules are compatible with w. Each rule deleting exactly one edge, since the clique C_n of size n has n^2 edges, the derivation height $h_{\mathscr{G}}(C_n) = n^2$. The lower bound follows.

For the upper bound, let C be the constant as defined by Lemma 3.1, let $K = \max(1, K_w)$ (we recall that $K_w = \max_{e \in \Sigma_E}(|w(e)|)$). Finally, let $H = \max\{n \mid (P, c_1, \ldots, c_n) \in \mathscr{G}\}$. Let $A = 2 \times K \times C \times (H+1) + 1$. Let Ω be the 'energy function' defined on graphs $\Omega(G) = w(G) + A \times |G|^2$. For each rule application $G \to G'$, one may verify that $\Omega(G) > \Omega(G')$. The last inequality together with Lemma 4.1 leads to the conclusion. □

Full proofs of Theorem 4.1 and of Theorem 4.2 are given in [2].

4.2 Termination by lexicographic weight

In our experiments, in most cases, the weight analysis of the preceding section was sufficient. The main counter-example is however systems composed of rules as given in Section 2. The GRS is strongly terminating but there is no compatible weight. This section provides a conciliable extension of this termination proof method. With a little abstraction, the linguistic example of Section 2 about long distance dependencies is computed by some 'non-local rule' R_{nl}:



Such non-local rules can be implemented by rules:

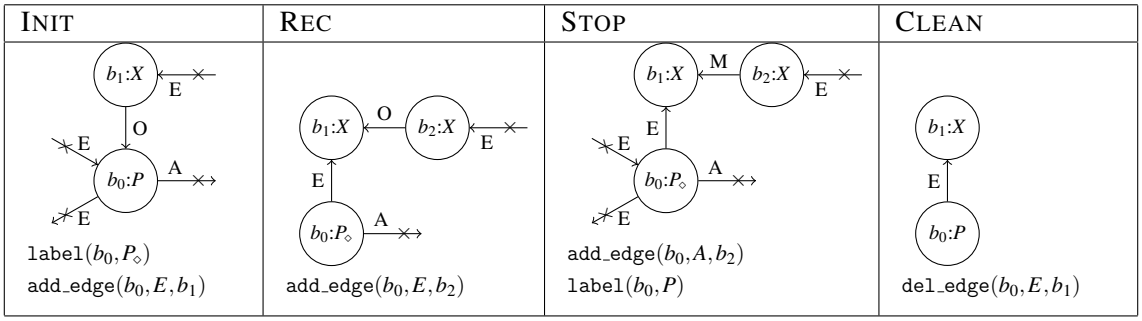


Figure 1: Local implementation of the non-local rule

However, these rules are not compatible with any weight. Actually, as justified in [2], there is no implementation of such a rule by some weighted rules.

Given an order $\prec$ on some set U, its lexicographic extension to sequences in U is defined by $(u_1,\dots,u_k) \prec_{\text{lex}} (v_1,\dots,v_m)$ iff $\exists j \leq \min(m,k) : u_j \prec v_j \wedge \forall i < j : u_i = v_i$. The order $\prec_{\text{lex}}$ is not well-founded in general, but its restriction to sequences of equal length is such as soon as $\prec$ is well-founded.

Definition 4.4 (Contextual weight)**.** *An* edge contextual weight *is a (finite) map* $\omega : \Sigma_N \times \Sigma_E \times \Sigma_N \to \mathbb{Z}$. *As for weights, it extends to any set* $E \subseteq \mathscr{E}_G$ *of some graph* G *by:* $\omega(E) = \sum_{n \xrightarrow{e} m \in E} \omega(\ell(n), e, \ell(m))$. *And the weight of a graph is* $\omega(G) = \omega(\mathscr{E}_G)$.

A contextual weight *is a 4-tuple* $\pi = (a, \omega, b, \eta)$ *with* $a, b \in \mathbb{N}$, ω *an edge contextual weight and* η *a node weight. We define* $\pi(G) = a \times \omega(G) + b \times \eta(G)$.

Let $e \in \Sigma_E$, *if* $a \neq 0$ *and there are* $\alpha, \beta, \alpha', \beta' \in \Sigma_N$ *such that* $\omega(\alpha, e, \beta) \neq \omega(\alpha', e, \beta')$, *then we say that* π *is* e-fragile.

Definition 4.5. *Given an edge weight* $w_0 : \Sigma_E \to \mathbb{Z}$, *given* k *contextual weights* $\pi_1, \dots, \pi_k$ *and a rule* $R = \langle P, \vec{c} \rangle$, *we write* $P' = P \cdot_{\mathbb{1}} \vec{c}$. *We say that* R *is* compatible *with* $(w_0, \pi_1, \dots, \pi_k)$ *iff:*

1. *either* $\vec{c}$ *contains a* `del_node` *command,*
2. *or* R *is an uniform and node-preserving rule such that:*
   - (a) *either the two properties below hold*
     - (i) $w_0(P') < w_0(P)$;
     - (ii) *and for all* $e \in \Sigma_E$ *such that* $w(e) < 0$, *for all* $n \in \Phi(\mathscr{N}_P)$, *let* M_n *be the set* $\Phi^{-1}(n)$; *then* M_n *contains at most one element* m *such that* $(m, e) \notin \bar{\mathscr{F}}$ *and* M_n *contains at most one element* $m' \in M_n$ *such that* $(q, m') \notin \bar{\mathscr{O}}$.
   - (b) *or the four properties below hold*
     - (i) $w_0(P') = w_0(P)$;
     - (ii) $(\pi_1(P'), \dots, \pi_k(P')) <_{lex} (\pi_1(P), \dots, \pi_k(P))$;

(iii) if $\vec{c}$ contains a command `label`(n,α) *and if some π_i is e-fragile, then $(n,e) \in \bar{\mathscr{I}} \cup \bar{\mathscr{O}}$;*
(iv) $\vec{c}$ does not contain any `shift` *commands.*

When a weight w_0 and k contextual weights are compatible with all the rules of some GRS $\mathscr{G}$, we say that $\mathscr{G}$ is lexicographically weighted by $(w_0, \pi_1, \ldots, \pi_k)$.

Example 4.1. *We define $w_0 = \mathbf{0}_{\Sigma_E}[A \mapsto -1]$, and $\omega = \mathbf{0}_{\Sigma_N \times \Sigma_E \times \Sigma_N}[(P,E,X) \mapsto 1, (P_\diamond, E, X) \mapsto -1]$. Consider the lexicographic weight $\pi = (1, \omega, 0, \mathbf{0}_{\Sigma_N})$. For rules in Figure 1, we have: rule* STOP *decreases by (2.a); rules* INIT *and* REC *decrease by (2.b): there is one more edge labeled E starting from $P_\diamond$ and rule* CLEAN *decreases by (2.b): one edge labeled E starting from P disappears.*

Theorem 4.3. *Whenever a program $\mathscr{G}$ is compatible with the lexicographic weight $(w_0, \pi_1, \ldots, \pi_k)$, it is strongly terminating in polynomial time. The bound is tight, that is for all $k > 0$, there is a GRS whose derivation height is $O(n^k)$.*

Proof. Examples for the lower bound are proposed in [2]. For the upper bound, let

$$K_\omega = \max\{|\omega(n,e,m)| \mid (n,e,m) \in \Sigma_N \times \Sigma_E \times \Sigma_N\} \quad \text{and} \quad K_\pi = a \times |\Sigma_E| \times K_\omega + b \times K_\eta$$

Then, adapting Lemma 4.1(b) to the present context, we can state that $|\omega(G)| \leq K_\omega \times |\Sigma_E| \times |G|^2$. With Lemma 4.1(c), we have $|\eta(G)| \leq K_\eta \times |G|$ and finally $|\pi(G)| \leq a \times K_\omega \times |\Sigma_E| \times |G|^2 + b \times K_\eta \times |G| \leq K_\pi \times |G|^2$.

Let $K_0 = \max_{i \in [1,k]}(K_{\pi_i})$. Finally, let K_E be the constant as given by Lemma 4.1 for w_0, we define $K = \max(K_0, K_E)$. Then, for all $i \leq k$, we have: $|\pi_i(G)| \leq K \times |G|^2$ and $|w_0(G)| \leq K \times |G|^2$.

Let $\kappa(G) = (|G|, w_0(G), \pi_1(G), \ldots, \pi_k(G))$. If $G \to G'$, then $\kappa(G) > \kappa(G')$. Consider a sequence $G_1 \to G_2 \to \cdots$. For all graph G_i of the sequence, $|G_i| \leq |G_1|$. Due to previous equations, $\kappa(G_i)$ is ranging in $L = [0, |G_1|] \times [-K \times |G_1|^2, K \times |G_1|^2]^{k+1}$. Thus the result. □

5 Conclusion

The polynomial derivation height that we have proved in the last section can be reconsidered in the following way. The example of a GRS working in $O(n^k)$ can be used as a clock. Then, since each transition of a (non-size increasing) Turing-Machine can be easily simulated by graph rewriting, we can state that any PTIME-predicate can be simulated by a lexicographically weighted GRS (up to a polynomial reduction). Since lexicographically weighted confluent GRS can be computed in polynomial time (each rewriting step can be simulated in linear time), it becomes clear that lexicographically weighted GRSs actually characterize PTIME. This provides a precise description of the computational content of the method.

We have implemented a software —called GREW (`grew.loria.fr`)— based on the Graph Rewriting definition presented in this article. In [12], the software was used to produce a semantically annotated version of the French Treebank; in this experiment, the system contains 34 modules and 571 rules and the corpus is constituted of 12 000 sentences of length up to 100 words. This experiment is a large scale application which shows that the proposed approach can be used in real-size applications.

As said earlier, despite the global non-confluence of the system, we can isolate subsets of rules that are confluent and use our system of modules to benefit from this confluence in implementation. In our last experiment, 26 of our 34 modules are confluent, but confluence proofs are tedious. We leave for further work the study of the local confluence of terminating GRS and the general study of confluence of Graph Rewriting Systems.

References

[1] B. Bohnet & L. Wanner (2001): *On using a parallel graph rewriting formalism in generation*. In: *EWNLG '01: Proceedings of the 8th European workshop on Natural Language Generation*, Association for Computational Linguistics, pp. 1–11, doi:10.3115/1117840.1117847.

[2] G. Bonfante & B. Guillaume (2013): *Non-size increasing Graph Rewriting for Natural Language Processing*. to appear in *Mathematical Structures for Computer Science*.

[3] G. Bonfante, B. Guillaume, M. Morey & G. Perrier (2011): *Modular Graph Rewriting to Compute Semantics*. In: *IWCS 2011*, Oxford, UK, pp. 65–74.

[4] A. Copestake (2009): Invited Talk: *Slacker Semantics: Why Superficiality, Dependency and Avoidance of Commitment can be the Right Way to Go*. In: *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, Association for Computational Linguistics, Athens, Greece, pp. 1–9.

[5] D. Crouch (2005): *Packed Rewriting for Mapping Semantics to KR*. In: *Proceedings of IWCS*.

[6] R. Echahed (2008): *Inductively Sequential Term-Graph Rewrite Systems*. In: *Proceedings of the 4th international conference on Graph Transformations*, ICGT '08, Springer-Verlag, Berlin, Heidelberg, pp. 84–98, doi:10.1007/978-3-540-87405-8_7.

[7] E. Godard, Y. Métivier, M. Mosbah & A. Sellami (2002): *Termination Detection of Distributed Algorithms by Graph Relabelling Systems*. In A. Corradini, H. Ehrig, H.-J. Kreowski & G. Rozenberg, editors: *ICGT*, *Lecture Notes in Computer Science* 2505, Springer, pp. 106–119, doi:10.1007/3-540-45832-8_10.

[8] E. Hyvönen (1984): *Semantic Parsing as Graph Language Transformation - a Multidimensional Approach to Parsing Highly Inflectional Languages*. In: *COLING*, pp. 517–520, doi:10.3115/980491.980601.

[9] V. Jijkoun & M. de Rijke (2007): *Learning to Transform Linguistic Graphs*. In: *Second Workshop on TextGraphs: Graph-Based Algorithms for Natural Language Processing, Rochester, NY, USA*.

[10] D.E. Knuth & P.B. Bendix (1970): *Simple word problems in universal algebras*. In J. Leech, editor: *Computational problems in abstract algebra*, Pergamon, pp. 263–277.

[11] M. Newman (1942): *On Theories With a Combinatorial Definition of "Equivalence"*. *Annals of Math.* 43(2), pp. 223–243, doi:10.2307/1968867.

[12] G. Perrier & B. Guillaume (2012): *Semantic Annotation of the French Treebank with Modular Graph Rewriting*. In Jan Hajic, editor: *META-RESEARCH Workshop on Advanced Treebanking, LREC 2012 Workshop*, META-NET, Istanbul, Turquie. Available at `http://hal.inria.fr/hal-00760577`.

[13] D. Plump (1995): *On Termination of Graph Rewriting*. In: *Proceedings of the 21st International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '95, Springer-Verlag, London, UK, pp. 88–100, doi:10.1007/3-540-60618-1_68.

[14] D. Plump (1998): *Termination of Graph Rewriting is Undecidable*. *Fundamenta Informaticae* 33(2), pp. 201–209, doi:10.3233/FI-1998-33204.

[15] G. Rozenberg, editor (1997): *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific.

[16] L. Tesnière (1959): *Eléments de syntaxe structurale*. Librairie C. Klincksieck, Paris.

Convergence in Infinitary Term Graph Rewriting Systems is Simple (Extended Abstract)*

Patrick Bahr
Department of Computer Science, University of Copenhagen
Universitetsparken 5, 2100 Copenhagen, Denmark
paba@diku.dk

In this extended abstract, we present a simple approach to convergence on term graphs that allows us to unify term graph rewriting and infinitary term rewriting. This approach is based on a partial order and a metric on term graphs. These structures arise as straightforward generalisations of the corresponding structures used in infinitary term rewriting. We compare our simple approach to a more complicated approach that we developed earlier and show that this new approach is superior in many ways. The only unfavourable property that we were able to identify, viz. failure of full correspondence between weak metric and partial order convergence, is rectified by adopting a strong convergence discipline.

1 Introduction

In *infinitary term rewriting* [17] we study infinite terms and infinite rewrite sequences. Typically, this extension to infinite structures is formalised by an ultrametric on terms, which yields infinite terms by metric completion and provides a notion of convergence to give meaning to infinite rewrite sequences. In this paper we extend infinitary term rewriting to term graphs. In addition to the metric approach, we also consider the partial order approach to infinitary term rewriting [4] and generalise it to the setting of term graphs.

One of the motivations for studying infinitary term rewriting is its relation to *non-strict evaluation*, which is used in programming languages such as Haskell [18]. Non-strict evaluation defers the evaluation of an expression until it is "needed" and thereby allows us to deal with conceptually infinite data structures and computations. For example, the function `from` defined below constructs for each number n the infinite list of consecutive numbers starting from n:

```
from(n) = n :: from(s(n))
```

This construction is only conceptual and only results in a terminating computation if it is used in a context where only finitely many elements of the list are "needed". Infinitary term rewriting provides us with an explicit limit construction to witness the outcome of an infinite computation as it is, for example, induced by `from`. After translating the above function definition to a term rewrite rule $\mathit{from}(x) \to x :: \mathit{from}(s(x))$, we may derive an infinite rewrite sequence

$$\mathit{from}(0) \to 0 :: \mathit{from}(s(0)) \to 0 :: s(0) :: \mathit{from}(s(s(0))) \to \dots$$

which converges to the infinite term $0 :: s(0) :: s(s(0)) :: \dots$, which represents the infinite list of numbers $0, 1, 2, \dots$ – as intuitively expected.

*The full version of this paper will appear in Mathematical Structures in Computer Science [7].

R. Echahed and D. Plump (Eds.): 7th International Workshop on Computing with Terms and Graphs
EPTCS 110, 2013, pp. 17–28, doi:10.4204/EPTCS.110.4

Non-strict evaluation is rarely found in isolation, though. Usually, it is implemented as lazy evaluation [14], which complements a non-strict evaluation strategy with *sharing*. The latter avoids duplication of subexpressions by using pointers instead of copying. For example, the function `from` above duplicates its argument `n` – it occurs twice on the right-hand side of the defining equation. A lazy evaluator simulates this duplication by inserting two pointers pointing to the actual argument.

While infinitary term rewriting is used to model the non-strictness of lazy evaluation, term graph rewriting models the sharing part of it. By endowing term graph rewriting with a notion of convergence like in infinitary term rewriting, we aim to unify the two formalisms into one calculus, thus allowing us to model both aspects within the same calculus.

Contributions & Outline At first we recall the basic notions of infinitary term rewriting (Section 2). Afterwards, we construct a metric and a partial order on term graphs and show that both are suitable as a basis for notions of convergence in term graph rewriting (Section 3). Based on these structures we introduce notions of convergence (weak and strong variants) for term graph rewriting and show correspondences between metric-based and partial order-based convergence (Section 4.1 and 4.2). We then present soundness and completeness properties of the resulting infinitary term graph rewriting calculi w.r.t. infinitary term rewriting (Section 4.3). Lastly, we compare our calculi with previous approaches (Section 5).

2 Infinitary Term Rewriting

Before starting with the development of infinitary *term graph* rewriting, we recall the basic notions of infinitary *term* rewriting. Rewrite sequences in infinitary rewriting, also called *reductions*, are sequences of the form $(\phi_\iota)_{\iota<\alpha}$, where each ϕ_ι is a rewrite step from a term t_ι to $t_{\iota+1}$ in a term rewriting system (TRS) $\mathscr{R}$, denoted $\phi_\iota\colon t_\iota \to_{\mathscr{R}} t_{\iota+1}$. The length α of such a sequence can be an arbitrary ordinal. For example, the infinite reduction indicated in Section 1 is the sequence $(\phi_i^{\mathrm{f}}\colon t_i^{\mathrm{f}} \to_{\mathscr{R}^{\mathrm{f}}} t_{i+1}^{\mathrm{f}})_{i<\omega}$, where $t_i^{\mathrm{f}} = 0::\ldots::s^{i-1}(0)::\mathit{from}(s^i(0))$ for all $i<\omega$ and $\mathscr{R}^{\mathrm{f}}$ is the TRS consisting of the single rule $\mathit{from}(x)\to x::\mathit{from}(s(x))$.

2.1 Metric Convergence

The above definition of reductions ensures that consecutive rewrite steps are "compatible", i.e. the result term of the ι-th step, viz. $t_{\iota+1}$, is the start term of the $(\iota+1)$-st step. However, this definition does not relate the start terms of steps at limit ordinal positions to the terms that preceded it. For example, we can extend the abovementioned reduction $(\phi_i^{\mathrm{f}})_{i<\omega}$ of length ω, to a reduction $(\phi_i^{\mathrm{f}})_{i<\omega+1}$ of length $\omega+1$ using any reduction step ϕ_ω^{f}, e.g. $\phi_\omega^{\mathrm{f}}\colon \mathit{from}(0)\to 0::\mathit{from}(s(0))$. In our informal notation this reduction $(\phi_i^{\mathrm{f}})_{i<\omega+1}$ reads as follows:

$$\mathit{from}(0)\to 0::\mathit{from}(s(0))\to 0::s(0)::\mathit{from}(s(s(0)))\to \quad\ldots\quad \mathit{from}(0)\to 0::\mathit{from}(s(0))$$

Intuitively, this does not make sense since the sequence of terms that precedes the last step intuitively converge to the term $0::s(0)::s(s(0))::\ldots$, but not $\mathit{from}(0)$.

In infinitary term rewriting such reductions are ruled out by a notion of convergence and a notion of continuity that follows from it. Typically, this notion of convergence is derived from a metric $\mathbf{d}$ on the set of (finite and infinite) terms $\mathscr{T}^\infty(\Sigma)$: $\mathbf{d}(s,t)=0$ if $s=t$, and $\mathbf{d}(s,t)=2^{-d}$ otherwise, where d is the

minimal depth at which s and t differ. Using this metric, we may also construct the set of (finite and infinite) terms $\mathscr{T}^\infty(\Sigma)$ by *metric completion* of the metric space $(\mathscr{T}(\Sigma),\mathbf{d})$ of finite terms.

The mode of convergence in the metric space $(\mathscr{T}^\infty(\Sigma),\mathbf{d})$ is the basis for the notion of *weak m-convergence* of reductions: a reduction $S=(\phi_\iota\colon t_\iota \to_{\mathscr{R}} t_{\iota+1})_{\iota<\alpha}$ is *weakly m-continuous* if $\lim_{\iota\to\lambda} t_\iota = t_\lambda$ for all limit ordinals $\lambda<\alpha$; it *weakly m-converges* to a term t, denoted $S\colon t_0 \xrightarrow{m}_{\mathscr{R}} t$, if it is weakly m-continuous and $\lim_{\iota\to\widehat{\alpha}} t_\iota = t$, where $\widehat{\alpha}$ is the length of the underlying sequence of terms $(t_\iota)_{\iota<\widehat{\alpha}}$. For example, the reduction $(\phi_i^{\mathrm{f}})_{i<\omega}$ weakly m-converges to the term $0::s(0)::s(s(0))::\ldots$; but the sequence $(\phi_i^{\mathrm{f}})_{i<\omega+1}$ does not weakly m-converge, it is not even weakly m-continuous as $\lim_{\iota\to\omega} t_\iota^{\mathrm{f}}$ is not $\mathit{from}(0)$.

Weak m-convergence is quite a general notion of convergence. For example, given a rewrite rule $a\to a$, we may derive the reduction $a\to a\to\ldots$, which weakly m-converges to a even though the rule $a\to a$ is applied again and again at the same position. This generality causes many desired properties to break, such as unique normal form properties and compression [16]. That is why Kennaway et al. [16] introduced *strong m-convergence*, which in addition requires that the depth at which rewrite steps take place tends to infinity as one approaches a limit ordinal: Let $S=(\phi_\iota\colon t_\iota \to_{\pi_\iota} t_{\iota+1})_{\iota<\alpha}$ be a reduction, where each π_ι indicates the position at which the step ϕ_ι takes place and $|\pi_\iota|$ denotes the length of the position π_ι. The reduction S is said to be *strongly m-continuous* (resp. *strongly m-converge* to t, denoted $S\colon t_0 \xrightarrow{m}\!\!\!\!\!\rightarrow t$) if it is weakly m-continuous (resp. weakly m-converges to t) and if $(|\pi_\iota|)_{\iota<\lambda}$ tends to infinity for all limit ordinals $\lambda<\alpha$ (resp. $\lambda\leq\alpha$). For example, the reduction $(\phi_i^{\mathrm{f}})_{i<\omega}$ also strongly m-converges to the term $0::s(0)::s(s(0))::\ldots$. On the other hand, the reduction $a\to a\to\ldots$ indicated above weakly m-converges to a, but it does not strongly m-converge to a.

2.2 Partial Order Convergence

Alternatively to the metric approach illustrated in Section 2.1, convergence can also be formalised using a partial order $\leq_\bot$ on terms. The idea to use this partial order for infinitary rewriting goes back to Corradini [12]. The signature Σ is extended to the signature $\Sigma_\bot$ by adding a fresh constant symbol $\bot$. When dealing with terms in $\mathscr{T}^\infty(\Sigma_\bot)$, we call terms that do not contain the symbol $\bot$, i.e. terms that are contained in $\mathscr{T}^\infty(\Sigma)$, *total*. We define $s\leq_\bot t$ iff s can be obtained from t by replacing some subterm occurrences in t by $\bot$. Interpreting the term $\bot$ as denoting "undefined", $\leq_\bot$ can be read as "is less defined than". The pair $(\mathscr{T}^\infty(\Sigma_\bot),\leq_\bot)$ is known to form a *complete semilattice* [13], i.e. it has a least element $\bot$, each directed set D in $(\mathscr{T}^\infty(\Sigma_\bot),\leq_\bot)$ has a *least upper bound* (*lub*) $\bigsqcup D$, and every *non-empty* set B in $(\mathscr{T}^\infty(\Sigma_\bot),\leq_\bot)$ has *greatest lower bound* (*glb*) $\bigsqcap B$. In particular, this means that for any sequence $(t_\iota)_{\iota<\alpha}$ in $(\mathscr{T}^\infty(\Sigma_\bot),\leq_\bot)$, its *limit inferior*, defined by $\liminf_{\iota\to\alpha} t_\iota = \bigsqcup_{\beta<\alpha}\left(\bigsqcap_{\beta\leq\iota<\alpha} t_\iota\right)$, exists.

In the same way that the limit in the metric space gives rise to weak m-continuity/-convergence, the limit inferior gives rise to *weak p-continuity* and *weak p-convergence*; simply replace lim by liminf. We write $S\colon t_0 \xrightarrow{p}\!\!\!\!\!\rightarrow t$ if a reduction S starting with term t_0 weakly p-converges to t. The defining difference between the two approaches is that p-continuous reductions always p-converge. The reason for that lies in the complete semilattice structure of $(\mathscr{T}^\infty(\Sigma_\bot),\leq_\bot)$, which guarantees that the limit inferior always exists (in contrast to the limit in a metric space).

The definition of the strong variant of p-convergence is a bit different from the one of m-convergence, but it follows the same idea: a reduction $(\phi_i\colon t_i\to_{\pi_i} t_{i+1})_{i<\omega}$ weakly m-converges iff the minimal depth d_i at which two consecutive terms t_i,t_{i+1} differ tends to infinity. The strong variant of m-convergence is a conservative approximation of this condition; it requires $|\pi_i|$ to tend to infinity. This approximation is conservative since $|\pi_i|\leq d_i$; differences between consecutive terms can only occur below the position at which a rewrite rule was applied.

In the partial order approach we can make this approximation more precise since we have the whole term structure at our disposal instead of only the measure provided by the metric $\mathbf{d}$. In the case of m-convergence, we replaced the actual depth of a minimal difference d_i with its conservative under-approximation $|\pi_i|$. For p-convergence, we replace the glb $t_i \sqcap t_{i+1}$, which intuitively represents the common information shared by t_i and t_{i+1}, with the conservative under-approximation $t_i[\bot]_{\pi_i}$, which replaces the redex at position π_i in t_i with $\bot$. This term $t_i[\bot]_{\pi_i}$ – called the *reduction context* of the step $\phi_i\colon\ t_i \to t_{i+1}$ – is a lower bound of t_i and t_{i+1} w.r.t. $\leq_\bot$ and is, thus, also smaller than $t_i \sqcap t_{i+1}$. The definition of strong p-convergence is obtained from the definition of weak p-convergence by replacing $\liminf_{\iota\to\lambda} t_\iota$ with $\liminf_{\iota\to\lambda} t_\iota[\bot]_{\pi_\iota}$.

A reduction $S = (\phi_\iota\colon\ t_\iota \to_{\pi_\iota} t_{\iota+1})_{\iota<\alpha}$ is called *strongly p-continuous* if $\liminf_{\iota\to\lambda} t_i[\bot]_{\pi_i} = t_\lambda$ for all limit ordinals $\lambda < \alpha$; it *strongly p-converges* to t, denoted $S\colon\ t_0 \xrightarrow{p}\!\!\!\!\rightarrow t$, if it is strongly p-continuous and either $\liminf_{\iota\to\alpha} t_i[\bot]_{\pi_i} = t$ in case α is a limit ordinal, or $t = t_{\alpha+1}$ otherwise.

Example 2.1. The previously mentioned reduction $(\phi_i^{\mathrm{f}})_{i<\omega}$ both strongly and weakly p-converges to the infinite term $0::s(0)::s(s(0))::\ldots$ – like in the metric approach. However, while the reduction $a \to a \to \ldots$ does not strongly m-converge, it strongly p-converges to the term $\bot$.

The partial order approach has some advantages over the metric approach. As explained above, every p-continuous reduction is also p-convergent. Moreover, strong p-convergence has some properties such as infinitary normalisation and infinitary confluence of orthogonal systems [4] that are not enjoyed by strong m-convergence.

Interestingly, however, the partial order-based notions of convergence are merely conservative extensions of the metric-based ones:

Theorem 2.1 ([2, 4])**.** *For every reduction S in a TRS, the following equivalences hold:*

$$\textit{(i)}\ S\colon\ s \xrightarrow{m} t \quad \textit{iff} \quad S\colon\ s \xrightarrow{p} t \ \textit{in}\ \mathscr{T}^\infty(\Sigma). \qquad \textit{(ii)}\ S\colon\ s \xrightarrow{m}\!\!\!\!\rightarrow t \quad \textit{iff} \quad S\colon\ s \xrightarrow{p}\!\!\!\!\rightarrow t \ \textit{in}\ \mathscr{T}^\infty(\Sigma).$$

The phrase "in $\mathscr{T}^\infty(\Sigma)$" means that all terms in S are total (including t). That is, if restricted to total terms, m- and p-convergence coincide.

3 Graphs and Term Graphs

In this section, we present our notion of term graphs and generalise the metric $\mathbf{d}$ and the partial order $\leq_\bot$ from terms to term graphs.

Our notion of graphs and term graphs is largely taken from Barendregt et al. [8].

Definition 3.1 (graphs)**.** A *graph* over signature Σ is a triple $g = (N, \mathsf{lab}, \mathsf{suc})$ consisting of a set N (of *nodes*), a *labelling function* $\mathsf{lab}\colon\ N \to \Sigma$, and a *successor function* $\mathsf{suc}\colon\ N \to N^*$ such that $|\mathsf{suc}(n)| = \mathsf{ar}(\mathsf{lab}(n))$ for each node $n \in N$, i.e. a node labelled with a k-ary symbol has precisely k successors. If $\mathsf{suc}(n) = \langle n_0, \ldots, n_{k-1}\rangle$, then we write $\mathsf{suc}_i(n)$ for n_i.

The successor function suc defines, for each node n, directed edges from n to $\mathsf{suc}_i(n)$. A path from a node m to a node n is a finite sequence $\langle e_0, \ldots, e_l\rangle$ of numbers such that $n = \mathsf{suc}_{e_l}(\ldots \mathsf{suc}_{e_0}(m))$, i.e. n is reached from m by taking the e_0-th edge, then the e_1-th edge etc.

Definition 3.2 (term graphs)**.** A *term graph* g over Σ is a tuple $(N, \mathsf{lab}, \mathsf{suc}, r)$ consisting of an *underlying* graph $(N, \mathsf{lab}, \mathsf{suc})$ over Σ whose nodes are all reachable from the *root node* $r \in N$. The class of all term graphs over Σ is denoted $\mathscr{G}^\infty(\Sigma)$. A *position* of $n \in N$ in g is a path in the underlying graph of g from r to n. The set of all positions of n in g is denoted $\mathscr{P}_g(n)$. The *depth* of n in g, denoted $\mathsf{depth}_g(n)$, is the minimum of the lengths of the positions of n in g, i.e. $\mathsf{depth}_g(n) = \min\{|\pi| \mid \pi \in \mathscr{P}_g(n)\}$. The term

graph g is called a *term tree* if each node in g has exactly one position. We use the notation N^g, lab^g, suc^g and r^g to refer to the respective components N,lab, suc and r of g. Given a graph or a term graph h and a node n in h, we write $h|_n$ to denote the sub-term graph of h rooted in n.

The notion of homomorphisms is crucial for dealing with term graphs. For greater flexibility, we will parametrise this notion by a set of constant symbols Δ for which the homomorphism condition is suspended. This will allow us to deal with variables and partiality appropriately.

Definition 3.3 (Δ-homomorphisms)**.** Let Σ be a signature, $\Delta \subseteq \Sigma^{(0)}$, and $g,h \in \mathscr{G}^\infty(\Sigma)$. A Δ-*homomorphism* ϕ from g to h, denoted $\phi\colon g \to_\Delta h$, is a function $\phi\colon N^g \to N^h$ with $\phi(r^g) = r^h$ that satisfies the following equations for all for all $n \in N^g$ with $\mathsf{lab}^g(n) \notin \Delta$:

$$\mathsf{lab}^g(n) = \mathsf{lab}^h(\phi(n)) \qquad \text{(labelling)}$$

$$\phi(\mathsf{suc}^g_i(n)) = \mathsf{suc}^h_i(\phi(n)) \quad \text{for all } 0 \le i < \mathsf{ar}(\mathsf{lab}^g(n)) \qquad \text{(successor)}$$

Note that, for $\Delta = \emptyset$, we get the usual notion of homomorphisms on term graphs (e.g. Barendsen [9]) and from that the notion of isomorphisms. The nodes labelled with symbols in Δ can be thought of as holes in the term graphs that can be filled with other term graphs.

We do not want to distinguish between isomorphic term graphs. Therefore, we use a well-known trick [19] to obtain canonical representatives of isomorphism classes of term graphs.

Definition 3.4. A term graph g is called *canonical* if $n = \mathscr{P}_g(n)$ holds for each $n \in N^g$. That is, each node is the set of its positions in the term graph. The set of all (finite) canonical term graphs over Σ is denoted $\mathscr{G}^\infty_\mathscr{C}(\Sigma)$ (resp. $\mathscr{G}_\mathscr{C}(\Sigma)$). For each term graph $h \in \mathscr{G}^\infty_\mathscr{C}(\Sigma)$, its *canonical representative* $\mathscr{C}(h)$ is obtained from h by replacing each node n in h by $\mathscr{P}_h(n)$.

This construction indeed yields a canonical representation of isomorphism classes. More precisely: $g \cong \mathscr{C}(g)$ for all $g \in \mathscr{G}^\infty(\Sigma)$, and $g \cong h$ iff $\mathscr{C}(g) = \mathscr{C}(h)$ for all $g,h \in \mathscr{G}^\infty(\Sigma)$.

We consider the set of terms $\mathscr{T}^\infty(\Sigma)$ as the subset of canonical term trees of $\mathscr{G}^\infty_\mathscr{C}(\Sigma)$. With this correspondence in mind, we can define the *unravelling* of a term graph g as the unique term $\mathscr{U}(g)$ such that there is a homomorphism $\phi\colon \mathscr{U}(g) \to g$. For example, g_0 from Figure 1 is the unravelling of g_1, and h_0 and g_ω from Figure 2 both unravel to the infinite term $@(f,@(f,\dots))$. Term graphs that unravel to the same term are called *bisimilar*.

3.1 A Simple Partial Order on Term Graphs

In this section, we want to establish a partial order suitable for formalising convergence of sequences of canonical term graphs similarly to weak p-convergence on terms.

Weak p-convergence on term rewriting systems is based on the partial order $\le_\bot$ on $\mathscr{T}^\infty(\Sigma_\bot)$, which instantiates occurrences of $\bot$ from left to right, i.e. $s \le_\bot t$ iff t is obtained by replacing occurrences of $\bot$ in s by arbitrary terms in $\mathscr{T}^\infty(\Sigma_\bot)$. Analogously, we consider the class of *partial term graphs* simply as term graphs over the signature $\Sigma_\bot = \Sigma \uplus \{\bot\}$. In order to generalise the partial order $\le_\bot$ to term graphs, we need to formalise the instantiation of occurrences of $\bot$ in term graphs. For this purpose, we shall use Δ-homomorphisms with $\Delta = \{\bot\}$, or $\bot$-homomorphisms for short. A $\bot$-homomorphism $\phi\colon g \to_\bot h$ maps each node in g to a node in h while "preserving its structure". Except for nodes labelled $\bot$ this also includes preserving the labelling. This exception to the homomorphism condition allows the $\bot$-homomorphism ϕ to instantiate each $\bot$-node in g with an arbitrary node in h. Using $\bot$-homomorphisms, we arrive at the following definition for our simple partial order $\le^\mathsf{S}_\bot$ on term graphs:

Definition 3.5. For each $g,h \in \mathscr{G}^\infty_\mathscr{C}(\Sigma_\bot)$, define $g \le^\mathsf{S}_\bot h$ iff there is some $\phi\colon g \to_\bot h$.

One can verify that $\leq_\bot^{\mathsf{S}}$ indeed generalises the partial order $\leq_\bot$ on terms. Considering terms as canonical term trees, we obtain the following characterisation of $\leq_\bot$ on terms $s,t \in \mathscr{T}^\infty(\Sigma_\bot)$:

$$s \leq_\bot t \iff \text{there is a } \bot\text{-homomorphism } \phi\colon\ s \to_\bot t.$$

The first important result for $\leq_\bot^{\mathsf{S}}$ is that the semilattice structure that we already had for $\leq_\bot$ is preserved by this generalisation:

Theorem 3.1. *The partially ordered set $(\mathscr{G}_\mathscr{C}^\infty(\Sigma_\bot), \leq_\bot^{\mathsf{S}})$ forms a complete semilattice.*

For terms, we already know that the set of (potentially infinite) terms can be constructed by forming the *ideal completion* of the partially ordered set $(\mathscr{T}(\Sigma_\bot), \leq_\bot)$ of finite terms [11]. More precisely, the ideal completion of $(\mathscr{T}(\Sigma_\bot), \leq_\bot)$ is order isomorphic to $(\mathscr{T}^\infty(\Sigma_\bot), \leq_\bot)$.

An analogous result can be shown for term graphs:

Theorem 3.2. *The ideal completion of $(\mathscr{G}_\mathscr{C}(\Sigma_\bot), \leq_\bot^{\mathsf{S}})$ is order isomorphic to $(\mathscr{G}_\mathscr{C}^\infty(\Sigma_\bot), \leq_\bot^{\mathsf{S}})$.*

3.2 A Simple Metric on Term Graphs

Next, we shall generalise the metric $\mathbf{d}$ from terms to term graphs. To achieve this, we need to formalise what it means for two term graphs to coincide up to a certain depth, so that we can reformulate the definition of the metric $\mathbf{d}$ for term graphs. To this end, we follow the same idea that the original definition of $\mathbf{d}$ on terms from Arnold and Nivat [1] was based on. In particular, we introduce a truncation construction that cuts off nodes below a certain depth:

Definition 3.6. Let $g \in \mathscr{G}^\infty(\Sigma_\bot)$ and $d \leq \omega$. The *simple truncation* $g{\dagger}d$ of g at d is the term graph defined as follows:

$$N^{g\dagger d} = \left\{ n \in N^g \,\middle|\, \mathsf{depth}_g(n) \leq d \right\} \qquad r^{g\dagger d} = r^g$$

$$\mathsf{lab}^{g\dagger d}(n) = \begin{cases} \mathsf{lab}^g(n) & \text{if } \mathsf{depth}_g(n) < d \\ \bot & \text{if } \mathsf{depth}_g(n) = d \end{cases} \qquad \mathsf{suc}^{g\dagger d}(n) = \begin{cases} \mathsf{suc}^g(n) & \text{if } \mathsf{depth}_g(n) < d \\ \langle\rangle & \text{if } \mathsf{depth}_g(n) = d \end{cases}$$

The definition of the simple metric $\mathbf{d}_\dagger$ follows straightforwardly:

Definition 3.7. The *simple distance* $\mathbf{d}_\dagger\colon\ \mathscr{G}_\mathscr{C}^\infty(\Sigma) \times \mathscr{G}_\mathscr{C}^\infty(\Sigma) \to \mathbb{R}_0^+$ is defined as follows:

$$\mathbf{d}_\dagger(g,h) = \begin{cases} 0 & \text{if } g = h \\ 2^{-d} & \text{if } g \neq h \text{ and } d = \max\{e < \omega \mid g{\dagger}e \cong h{\dagger}e\} \end{cases}$$

Again, we can verify that $\mathbf{d}_\dagger$ generalises $\mathbf{d}$. In particular, we can show that our notion of truncation coincides with that of Arnold and Nivat [1] if restricted to terms.

As desired, this generalisation retains the complete ultrametric space structure:

Theorem 3.3. *The pair $(\mathscr{G}_\mathscr{C}^\infty(\Sigma), \mathbf{d}_\dagger)$ forms a complete ultrametric space.*

The metric space analogue to ideal completion is metric completion. On terms, we already know that we can construct the set of (potentially infinite) terms $\mathscr{T}^\infty(\Sigma)$ by metric completion of the metric space $(\mathscr{T}(\Sigma), \mathbf{d})$ of finite terms [10]. More precisely, the metric completion of $(\mathscr{T}(\Sigma), \mathbf{d})$ is the metric space $(\mathscr{T}^\infty(\Sigma), \mathbf{d})$. This property generalises to term graphs as well:

Theorem 3.4. *The metric completion of $(\mathscr{G}_\mathscr{C}(\Sigma), \mathbf{d}_\dagger)$ is the metric space $(\mathscr{G}_\mathscr{C}^\infty(\Sigma), \mathbf{d}_\dagger)$.*

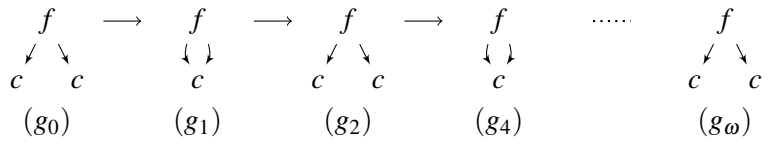
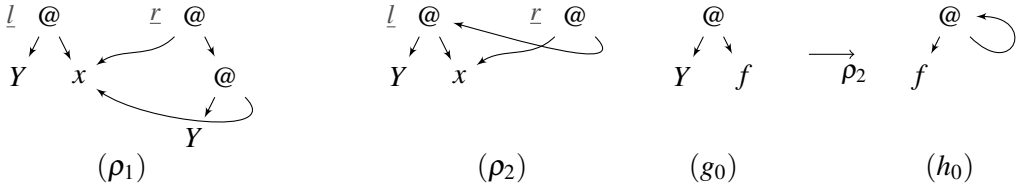
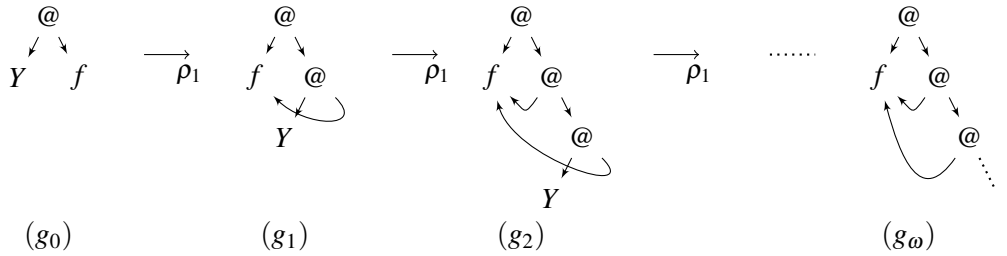


Figure 1: Limit inferior in the presence of acyclic sharing.



(a) Term graph rules that unravel to $Y\,x \to x\,(Y\,x)$.

(b) A single ρ_2-step.



(c) A strongly m-convergent term graph reduction over ρ_1.

Figure 2: Implementation of the fixed point combinator as a term graph rewrite rule.

4 Infinitary Term Graph Rewriting

In this paper, we adopt the term graph rewriting framework of Barendregt et al. [8]. In order to represent placeholders in rewrite rules, we use variables – in a manner much similar to term rewrite rules. To this end, we consider a signature $\Sigma_{\mathscr{V}} = \Sigma \uplus \mathscr{V}$ that extends the signature Σ with a set $\mathscr{V}$ of nullary variable symbols.

Definition 4.1 (term graph rewriting systems)**.** Given a signature Σ, a *term graph rule* ρ over Σ is a triple (g,l,r) where g is a graph over $\Sigma_{\mathscr{V}}$ and $l,r \in N^g$ such that all nodes in g are reachable from l or r. We write ρ_l resp. ρ_r to denote the left- resp. right-hand side of ρ, i.e. the term graph $g|_l$ resp. $g|_r$. Additionally, we require that for each variable $v \in \mathscr{V}$ there is at most one node n in g labelled v, and we have that $n \neq l$ and that n is reachable from l in g. A *term graph rewriting system (GRS)* $\mathscr{R}$ is a pair (Σ,R) with Σ a signature and R a set of term graph rules over Σ.

The notion of unravelling straightforwardly extends to term graph rules: the *unravelling* of a term graph rule ρ, denoted $\mathscr{U}(\rho)$, is the term rule $\mathscr{U}(\rho_l) \to \mathscr{U}(\rho_r)$. The unravelling of a GRS $\mathscr{R} = (\Sigma,R)$, denoted $\mathscr{U}(\mathscr{R})$, is the TRS $(\Sigma, \{\mathscr{U}(\rho) \mid \rho \in R\})$.

Example 4.1. Figure 2a shows two term graph rules which both unravel to the term rule $\rho\colon\ @(Y,x) \to @(x,@(Y,x))$ that defines the fixed point combinator Y. Note that sharing of nodes is used both to refer to variables in the left-hand side from the right-hand side and in order to simulate duplication.

Without going into all details of the construction, we describe the application of a rewrite rule ρ with root nodes l and r to a term graph g in four steps: at first a suitable sub-term graph of g rooted in some node n of g is *matched* against the left-hand side of ρ. This matching amounts to finding a $\mathscr{V}$-homomorphism ϕ from the left-hand side ρ_l to $g|_n$, the *redex*. The $\mathscr{V}$-homomorphism ϕ allows us to instantiate variables in the rule with sub-term graphs of the redex. In the second step, nodes and edges in ρ that are not in ρ_l are copied into g, such that each edge pointing to a node m in ρ_l is redirected to $\phi(m)$. In the next step, all edges pointing to the root n of the redex are redirected to the root n' of the *contractum*, which is either r or $\phi(r)$, depending on whether r has been copied into g or not (because it is reachable from l in ρ). Finally, all nodes not reachable from the root of (the now modified version of) g are removed. With h the result of the above construction, we obtain a *pre-reduction step* $\psi\colon\ g \mapsto_n h$ from g to h.

The definition of term graph rewriting in the form of pre-reduction steps is very operational. While this style is beneficial for implementing a rewriting system, it is problematic for reasoning on term graphs modulo isomorphism, which is necessary for introducing notions of convergence. However, one can easily see that the construction of the result term graph of a pre-reduction step is invariant under isomorphism, which justifies the following definition of reduction steps:

Definition 4.2. Let $\mathscr{R} = (\Sigma, R)$ be GRS, $\rho \in R$ and $g, h \in \mathscr{G}^{\infty}_{\mathscr{C}}(\Sigma)$ with $n \in N^g$ and $m \in N^h$. A tuple $\phi = (g, n, h)$ is called a *reduction step*, written $\phi\colon\ g \to_n h$, if there is a pre-reduction step $\phi'\colon\ g' \mapsto_{n'} h'$ with $\mathscr{C}(g') = g$, $\mathscr{C}(h') = h$, and $n = \mathscr{P}_{g'}(n')$. We also write $\phi\colon\ g \to_{\mathscr{R}} h$ to indicate $\mathscr{R}$.

In other words, a reduction step is a canonicalised pre-reduction step. Figure 2b and Figure 2c illustrate some (pre-)reduction steps induced by the rules ρ_1 respectively ρ_2 shown in Figure 2a.

4.1 Weak Convergence

In analogy to infinitary term rewriting, we employ the partial order $\leq^{\mathsf{S}}_{\perp}$ and the metric $\mathbf{d}_{\dagger}$ for the purpose of defining convergence of transfinite term graph reductions.

Definition 4.3. Let $\mathscr{R} = (\Sigma, R)$ be a GRS.

(i) Let $S = (g_\iota \to_{\mathscr{R}} g_{\iota+1})_{\iota<\alpha}$ be a reduction in $\mathscr{R}$. S is *weakly m-continuous* in $\mathscr{R}$ if $\lim_{\iota\to\lambda} g_\iota = g_\lambda$ for each limit ordinal $\lambda < \alpha$. S *weakly m-converges* to $g \in \mathscr{G}^{\infty}_{\mathscr{C}}(\Sigma)$ in $\mathscr{R}$, written $S\colon\ g_0 \xrightarrow{m}_{\mathscr{R}} g$, if it is weakly m-continuous and $\lim_{\iota\to\widehat{\alpha}} g_\iota = g$.

(ii) Let $\mathscr{R}_\perp$ be the GRS $(\Sigma_\perp, R)$ over the extended signature $\Sigma_\perp$ and $S = (g_\iota \to_{\mathscr{R}_\perp} g_{\iota+1})_{\iota<\alpha}$ a reduction in $\mathscr{R}_\perp$. S is *weakly p-continuous* in $\mathscr{R}$ if $\liminf_{\iota<\lambda} g_i = g_\lambda$ for each limit ordinal $\lambda < \alpha$. S *weakly p-converges* to $g \in \mathscr{G}^{\infty}_{\mathscr{C}}(\Sigma_\perp)$ in $\mathscr{R}$, written $S\colon\ g_0 \xrightarrow{p}_{\mathscr{R}} g$, if it is weakly p-continuous and $\liminf_{\iota<\widehat{\alpha}} g_i = g$.

Example 4.2. Figure 2c illustrates an infinite reduction derived from the rule ρ_1 in Figure 2a. Since $g_i\dagger(i+1) \cong g_\omega\dagger(i+1)$ for all $i < \omega$, we have that $\lim_{i\to\omega} g_i = g_\omega$, which means that the reduction weakly m-converges to the term graph g_ω. Moreover, since each node in g_ω eventually appears in a term graph in $(g_i)_{i<\omega}$ and remains stable afterwards, we have $\liminf_{i\to\omega} g_\iota = g_\omega$. Consequently, the reduction also weakly p-converges to g_ω.

Recall that weak p-convergence for TRSs is a conservative extension of weak m-convergence (cf. Theorem 2.1). The key property that makes this possible is that for each sequence $(t_\iota)_{\iota<\alpha}$ in $\mathscr{T}^{\infty}(\Sigma)$, we have that $\lim_{\iota\to\alpha} t_\iota = \liminf_{\iota\to\alpha} t_\iota$ whenever $(t_\iota)_{\iota<\alpha}$ converges, or $\liminf_{\iota\to\alpha} t_\iota$ is a total term. Sadly, this is not the case for the metric space and the partial order on term graphs: the sequence of term graphs depicted in Figure 1 has a total term graph as its limit inferior, viz. g_ω, although it does not converge in

the metric space. In fact, since the sequence in Figure 1 alternates between two distinct term graphs, it does not converge in any Hausdorff space, i.e. in particular, it does not converge in any metric space.

This example shows that we cannot hope to generalise the compatibility property that we have for terms: even if a sequence of total term graphs has a total term graph as its limit inferior, it might not converge. However, the converse direction of the correspondence does hold true:

Theorem 4.1. *If $(g_\iota)_{\iota<\alpha}$ converges, then $\lim_{\iota\to\alpha} g_\iota = \liminf_{\iota\to\alpha} g_\iota$.*

From this property, we obtain the following relation between weak *m*- and *p*-convergence:

Theorem 4.2. *Let S be a reduction in a GRS $\mathscr{R}$. If $S\colon g \xrightarrow{m}_{\mathscr{R}} h$ then $S\colon g \xrightarrow{p}_{\mathscr{R}} h$.*

As indicated above, weak *m*-convergence is not the total fragment of weak *p*-convergence as it is the case for TRSs, i.e. the converse of the above implication does not hold in general:

Example 4.3. There is a GRS that yields the reduction shown in Figure 1, which weakly *p*-converges to g_ω but is not weakly *m*-convergent. This reduction can be produced by alternately applying the rules ρ_1, ρ_2, where the left hand side of both rules and the right-hand side of ρ_1 is g_0, and the right-hand side of ρ_2 is g_1.

4.2 Strong Convergence

The idea of strong convergence is to conservatively approximate the convergence behaviour somewhat independently from the actual rewrite rules that are applied. Strong *m*-convergence in TRSs requires that the depth of the redexes tends to infinity thereby assuming that anything at the depth of the redex or below is potentially affected by a reduction step. Strong *p*-convergence, on the other hand, uses a better approximation that only assumes that the redex is affected by a reduction step – not however other subterms at the same depth. To this end strong *p*-convergence uses a notion of reduction contexts – essentially the term minus the redex – for the formation of limits. The following definition provides the construction for the notion of reduction contexts that we shall use for term graph rewriting:

Definition 4.4. Let $g \in \mathscr{G}^\infty(\Sigma_\perp)$ and $n \in N^g$. The *local truncation* of g at n, denoted $g\backslash n$, is obtained from g by labelling n with $\perp$ and removing all outgoing edges from n as well as all nodes that thus become unreachable from the root.

Proposition 4.1. *Given a reduction step $g \to_n h$, we have $g\backslash n \leq^S_\perp g, h$.*

This means that the local truncation at the root of the redex is preserved by reduction steps and is therefore an adequate notion of reduction context for strong *p*-convergence [3]. Using this construction we can define strong *p*-convergence on term graphs analogously to strong *p*-convergence on terms. For strong *m*-convergence, we simply take the same notion of depth that we already used for the definition of the simple truncation $g\dagger d$ and thus the simple metric $\mathbf{d}_\dagger$.

Definition 4.5. Let $\mathscr{R} = (\Sigma, R)$ be a GRS.

(i) The *reduction context* c of a graph reduction step $\phi\colon g \to_n h$ is the term graph $\mathscr{C}(g\backslash n)$. We write $\phi\colon g \to_c h$ to indicate the reduction context of a graph reduction step.

(ii) Let $S = (g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\alpha}$ be a reduction in $\mathscr{R}$. S is *strongly m-continuous* in $\mathscr{R}$ if $\lim_{\iota\to\lambda} g_\iota = g_\lambda$ and $(\mathsf{depth}_{g_\iota}(n_\iota))_{\iota<\lambda}$ tends to infinity for each limit ordinal $\lambda < \alpha$. S *strongly m-converges* to g in $\mathscr{R}$, denoted $S\colon g_0 \xrightarrow{m}_{\mathscr{R}} g$, if it is strongly *m*-continuous and either S is closed with $g = g_\alpha$ or S is open with $g = \lim_{\iota\to\alpha} g_\iota$ and $(\mathsf{depth}_{g_\iota}(n_\iota))_{\iota<\alpha}$ tending to infinity.

(iii) Let $S = (g_\iota \to_{c_\iota} g_{\iota+1})_{\iota<\alpha}$ be a reduction in $\mathscr{R}_\bot = (\Sigma_\bot, R)$. S is *strongly p-continuous* in $\mathscr{R}$ if $\liminf_{\iota\to\lambda} c_\iota = g_\lambda$ for each limit ordinal $\lambda < \alpha$. S *strongly p-converges* to g in $\mathscr{R}$, denoted $S\colon\ g_0 \xrightarrow{p}_{\mathscr{R}} g$, if it is strongly p-continuous and either S is closed with $g = g_\alpha$ or S is open with $g = \liminf_{\iota\to\alpha} c_\iota$.

Example 4.4. As explained in Example 4.2, the reduction in Figure 2c both weakly m- and p-converges to g_ω. Because contraction takes place at increasingly large depth, the reduction also strongly m-converges to g_ω. Moreover, since each node in g_ω eventually appears also in the sequence of reduction contexts $(c_i)_{i<\omega}$ of the reduction and remains stable afterwards, we have that $\liminf_{i\to\omega} c_i = g_\omega$. Consequently, the reduction also strongly p-converges to g_ω.

Remarkably, one of the advantages of the strong variant of convergence is that we regain the correspondence between m- and p-convergence that we know from infinitary term rewriting:

Theorem 4.3 ([5])**.** *Let $\mathscr{R}$ be a GRS and S a reduction in $\mathscr{R}_\bot$. We then have that*

$$S\colon\ g \xrightarrow{m}_{\mathscr{R}} h \qquad \textit{iff} \qquad S\colon\ g \xrightarrow{p}_{\mathscr{R}} h \textit{ in } \mathscr{G}^\infty_{\mathscr{C}}(\Sigma).$$

In particular, the GRS given in Example 4.3 that induces the reduction depicted in Figure 1 does not provide a counterexample for the "if" direction of the above equivalence – in contrast to weak convergence. The reduction in Figure 1 does not strongly m-converge but it does strongly p-converge to the term graph $\bot$, which is in accordance with Theorem 4.3 above.

4.3 Soundness and Completeness

In order to assess the value of the modes of convergence on term graphs that we introduced in this paper, we need to compare them to the well-established counterparts on terms. Ideally, we would like to see a strong connection between converging reductions in a GRS $\mathscr{R}$ and converging reductions in the TRS $\mathscr{U}(\mathscr{R})$ in the form of soundness and completeness properties. For example, for m-convergence we want to see that $g \xrightarrow{m}_{\mathscr{R}} h$ implies $\mathscr{U}(g) \xrightarrow{m}_{\mathscr{U}(\mathscr{R})} \mathscr{U}(h)$ – i.e. soundness – and vice versa that $\mathscr{U}(g) \xrightarrow{m}_{\mathscr{U}(\mathscr{R})} t$ implies $g \xrightarrow{m}_{\mathscr{R}} h$ with $\mathscr{U}(h) = t$ – i.e. completeness.

Completeness is already an issue for finitary rewriting [15]: a single term graph redex may correspond to several term redexes due to sharing. Hence, contracting a term graph redex may correspond to several term rewriting steps, which may be performed independently.

In the context of weak convergence, also soundness becomes an issue. The underlying reason for this issue is similar to the phenomenon explained above: a single term graph rewrite step may represent several term rewriting steps, i.e. $g \to_{\mathscr{R}} h$ implies $\mathscr{U}(g) \to^+_{\mathscr{U}(\mathscr{R})} \mathscr{U}(h)$.[1] When we have a converging term graph reduction $(\phi_\iota\colon\ g_\iota \to g_{\iota+1})_{\iota<\alpha}$, we know that the underlying sequence of term graphs $(g_\iota)_{\iota<\widehat{\alpha}}$ converges. However, the corresponding term reduction does not necessarily produce the sequence $(\mathscr{U}(g_\iota))_{\iota<\widehat{\alpha}}$ but may intersperse the sequence $(\mathscr{U}(g_\iota))_{\iota<\widehat{\alpha}}$ with additional intermediate terms, which might change the convergence behaviour.

While we cannot prove soundness for weak convergence due to the abovementioned problems, we can show that the underlying modes of convergence are sound in the sense that convergence is preserved under unravelling.

Theorem 4.4.

(i) $\lim_{\iota\to\alpha} g_\iota = g$ *implies* $\lim_{\iota\to\alpha} \mathscr{U}(g_\iota) = \mathscr{U}(g)$ *for every sequence* $(g_\iota)_{\iota<\alpha}$ *in* $(\mathscr{G}^\infty_{\mathscr{C}}(\Sigma), \mathbf{d}_\dagger)$.

(ii) $\mathscr{U}(\liminf_{\iota\to\alpha} g_\iota) = \liminf_{\iota\to\alpha} \mathscr{U}(g_\iota)$ *for every sequence* $(g_\iota)_{\iota<\alpha}$ *in* $(\mathscr{G}^\infty_{\mathscr{C}}(\Sigma_\bot), \leq^S_\bot)$.

[1] If the term graph g is cyclic, the corresponding term reduction may even be infinite.

Note that the above theorem in fact implies soundness of the modes of convergence on term graphs with the modes of convergence on terms since both $\mathbf{d}_\dagger$ and $\leq_\perp^\mathsf{S}$ specialise to $\mathbf{d}$ respectively $\leq_\perp$ if restricted to term trees.

However, we can observe that strong convergence is more well-behaved than weak convergence. It is possible to prove soundness and completeness properties for strong p-convergence:

Theorem 4.5 ([5]). *Let $\mathscr{R}$ be a left-finite GRS.*

(i) If $\mathscr{R}$ is left-linear and $g \xrightarrow{p}_{\mathscr{R}} h$, then $\mathscr{U}(g) \xrightarrow{p}_{\mathscr{U}(\mathscr{R})} \mathscr{U}(h)$.

(ii) If $\mathscr{R}$ is orthogonal and $\mathscr{U}(g) \xrightarrow{p}_{\mathscr{U}(\mathscr{R})} t$, then there are reductions $g \xrightarrow{p}_{\mathscr{R}} h$ and $t \xrightarrow{p}_{\mathscr{U}(\mathscr{R})} \mathscr{U}(h)$.

Note that the above completeness property is not the one that one would initially expect, namely $\mathscr{U}(g) \xrightarrow{p}_{\mathscr{U}(\mathscr{R})} t$ implies $g \xrightarrow{p}_{\mathscr{R}} h$ with $\mathscr{U}(h) = t$. But this general completeness property is known to already fail for finitary term graph rewriting [15].

The soundness and completeness properties above have an important practical implication: GRSs that only differ in their sharing, i.e. they unravel to the same TRS, will produce the same results, i.e. the same normal forms up to bisimilarity. GRSs with more sharing may, however, reach a result with fewer steps. This can be observed in Figure 2, which depicts two rules ρ_1, ρ_2 that unravel to the same term rule. Rule ρ_1 reaches g_ω in ω steps whereas ρ_2 reaches a term graph h_0, which is bisimilar to g_ω, in one step.

The situation for strong m-convergence is not the same as for strong p-convergence. While we do have soundness under the same preconditions, i.e. $g \xrightarrow{m}_{\mathscr{R}} h$ implies $\mathscr{U}(g) \xrightarrow{m}_{\mathscr{U}(\mathscr{R})} \mathscr{U}(h)$, the completeness property we have seen in Theorem 4.5 fails. This behaviour was already recognised by Kennaway et al. [15]. Nevertheless, we can find a weaker form of completeness that is restricted to normalising reductions:

Theorem 4.6 ([5]). *Given an orthogonal, left-finite GRS $\mathscr{R}$ that is normalising w.r.t. strongly m-converging reductions, we find for each normalising reduction $\mathscr{U}(g) \xrightarrow{m}_{\mathscr{U}(\mathscr{R})} t$ a reduction $g \xrightarrow{m}_{\mathscr{R}} h$ such that $t = \mathscr{U}(h)$.*

5 Concluding Remarks

We have devised two independently defined but closely related infinitary calculi of term graph rewriting. This is not the first proposal for infinitary term graph rewriting calculi; in previous work [6] we presented a so-called *rigid* approach based on a metric and a partial order different from the structures presented here.

There are several arguments why the simple approach presented in this paper is superior to the rigid approach. First of all it is simpler. The rigid metric and partial order have been carefully crafted in order to obtain a correspondence result in the style of Theorem 2.1 for weak convergence on term graphs. This correspondence result of the rigid approach is not fully matched by the simple approach that we presented here, but we do regain the full correspondence by moving to strong convergence.

Secondly, the rigid approach is very restrictive, disallowing many reductions that are intuitively convergent. For example, in the rigid approach the reduction depicted in Figure 2c, would not p-converge (weakly or strongly) to the term graph g_ω as intuitively expected but instead to the term graph obtained from g_ω by replacing f with $\perp$. Moreover, this sequence would not m-converge (weakly or strongly) at all.

Lastly, as a consequence of the restrictive nature of the rigid approach, the completion constructions of the underlying metric and partial order do not yield the full set of term graphs – in contrast to our findings here in Theorem 3.2 and 3.4.

Unfortunately, we do not have solid soundness or completeness results for weak convergence apart from the preservation of convergence under unravelling and the metric/ideal completion construction of the set of term graphs. However, as we have shown, this shortcoming is again addressed by moving to strong convergence.

References

[1] A. Arnold & M. Nivat (1980): *The metric space of infinite trees. Algebraic and topological properties*. *Fundam. Inf.* 3(4), pp. 445–476.

[2] P. Bahr (2009): *Infinitary Rewriting - Theory and Applications*. Master's thesis, Vienna University of Technology, Vienna.

[3] P. Bahr (2010): *Abstract Models of Transfinite Reductions*. In C. Lynch, editor: *RTA'10*, 6, pp. 49–66, doi:10.4230/LIPIcs.RTA.2010.49.

[4] P. Bahr (2010): *Partial Order Infinitary Term Rewriting and Böhm Trees*. In C. Lynch, editor: *RTA'10*, 6, pp. 67–84, doi:10.4230/LIPIcs.RTA.2010.67.

[5] P. Bahr (2012): *Infinitary Term Graph Rewriting is Simple, Sound and Complete*. In A. Tiwari, editor: *RTA'12*, 15, pp. 69–84, doi:10.4230/LIPIcs.RTA.2012.69.

[6] P. Bahr (2012): *Modes of Convergence for Term Graph Rewriting*. *Logical Methods in Computer Science* 8(2):6, doi:10.2168/LMCS-8(2:6)2012.

[7] P. Bahr (2013): *Convergence in Infinitary Term Graph Rewriting Systems is Simple*. Math. Struct. in Comp. Science, to appear.

[8] H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, R. Kennaway, M.J. Plasmeijer & M.R. Sleep (1987): *Term graph rewriting*. In Philip C. Treleaven Jaco de Bakker, A. J. Nijman, editor: *PARLE'87*, *LNCS* 259, Springer, pp. 141–158, doi:10.1007/3-540-17945-3_8.

[9] E. Barendsen (2003): *Term Graph Rewriting*. In Terese, editor: *Term Rewriting Systems*, chapter 13, Cambridge University Press, pp. 712–743.

[10] M. Barr (1993): *Terminal coalgebras in well-founded set theory*. *Theor. Comput. Sci.* 114(2), pp. 299 – 315, doi:10.1016/0304-3975(93)90076-6.

[11] G. Berry & J.-J. Lévy (1977): *Minimal and optimal computations of recursive programs*. In: *POPL'77*, pp. 215–226, doi:10.1145/512950.512971.

[12] A. Corradini (1993): *Term rewriting in CT_Σ*. In M.-C. Gaudel & J.-P. Jouannaud, editors: *TAPSOFT'93*, pp. 468–484, doi:10.1007/3-540-56610-4_83.

[13] J.A. Goguen, J.W. Thatcher, E.G. Wagner & J.B. Wright (1977): *Initial Algebra Semantics and Continuous Algebras*. *J. ACM* 24(1), pp. 68–95, doi:10.1145/321992.321997.

[14] P. Henderson & J.H. Morris, Jr. (1976): *A lazy evaluator*. In: *POPL'76*, pp. 95–103, doi:10.1145/800168.811543.

[15] R. Kennaway, J.W. Klop, M.R. Sleep & F.-J. de Vries (1994): *On the adequacy of graph rewriting for simulating term rewriting*. *ACM Trans. Program. Lang. Syst.* 16(3), pp. 493–523, doi:10.1145/177492.177577.

[16] R. Kennaway, J.W. Klop, M.R. Sleep & F.-J. de Vries (1995): *Transfinite Reductions in Orthogonal Term Rewriting Systems*. *Inf. Comput.* 119(1), pp. 18–38, doi:10.1006/inco.1995.1075.

[17] R. Kennaway & F.-J. de Vries (2003): *Infinitary Rewriting*. In Terese, editor: *Term Rewriting Systems*, 1st edition, chapter 12, Cambridge University Press, pp. 668–711.

[18] S. Marlow (2010): *Haskell 2010 Language Report*.

[19] D. Plump (1999): *Term graph rewriting*. In H. Ehrig, G. Engels, H.-J. Kreowski & G. Rozenberg, editors: *Handbook of Graph Grammars and Computing by Graph Transformation*, 2, World Scientific Publishing Co., Inc., pp. 3–61, doi:10.1142/9789812815149_0001.

Linear Compressed Pattern Matching for Polynomial Rewriting (Extended Abstract)

Manfred Schmidt-Schauss
Institut für Informatik,
Fachbereich Informatik und Mathematik,
Goethe-Universität,
Postfach 11 19 32, D-60054 Frankfurt, Germany
schauss@ki.informatik.uni-frankfurt.de

This paper is an extended abstract of an analysis of term rewriting where the terms in the rewrite rules as well as the term to be rewritten are compressed by a singleton tree grammar (STG). This form of compression is more general than node sharing or representing terms as dags since also partial trees (contexts) can be shared in the compression. In the first part efficient but complex algorithms for detecting applicability of a rewrite rule under STG-compression are constructed and analyzed. The second part applies these results to term rewriting sequences.

The main result for submatching is that finding a redex of a left-linear rule can be performed in polynomial time under STG-compression.

The main implications for rewriting and (single-position or parallel) rewriting steps are: (i) under STG-compression, n rewriting steps can be performed in nondeterministic polynomial time. (ii) under STG-compression and for left-linear rewrite rules a sequence of n rewriting steps can be performed in polynomial time, and (iii) for compressed rewrite rules where the left hand sides are either DAG-compressed or ground and STG-compressed, and an STG-compressed target term, n rewriting steps can be performed in polynomial time.

1 Introduction

An important concept in various areas of computer science like automated deduction, first order logic, term rewriting, type checking, are terms (ranked trees), and also terms containing variables (see e.g. [2]). The basic and widely used algorithms in these areas are matching, unification, term rewriting, equational deduction, asf. For example, a term $f(g(a,b),c)$ may be rewritten into $f(g(b,a),c)$ by the commutativity axiom $g(x,y) = g(y,x)$ for g. Since implemented systems often deal with large terms, perhaps generated ones, it is of high interest to look for compression mechanisms for terms, and consequently, also investigate variants of the known algorithms that also perform efficiently on the compressed terms without prior decompression.

The device of straight line programs (SLP) for compression of strings is a general one and allows analyses of correctness and complexity of algorithms [21, 16]. SLPs are polynomially equivalent to the LZ77-variant of Lempel-Ziv compression [25]. SLPs are non-cyclic context free grammars (CFGs), where every nonterminal has exactly one production in the CFG, such that any nonterminal represents exactly one string. Basic algorithms are the equality check of two compressed strings, which requires polynomial time [19] (see [15] for an efficient version and [11] for a proposal of a further improvement), and the compressed pattern match, i.e., given two SLP-compressed strings s,t, the question whether s is a substring of t can also be solved in polynomial time in the size of the SLPs.

A generalization of SLPs for the compression of terms are singleton tree grammars (STG) [22, 13, 7], a specialization of straight line context free tree grammars [4, 5, 17, 18], where linear SLCF tree grammars

R. Echahed and D. Plump (Eds.): 7th International Workshop on Computing with Terms and Graphs
EPTCS 110, 2013, pp. 29–40, doi:10.4204/EPTCS.110.5

are polynomially equivalent to STGs [17, 18]. Basic notions for tree grammars and tree automata can be found in [6]. Besides using the well-known node sharing, also partial subtrees (contexts) can be shared in the compression. The Plandowski-Lifshits equality test of nonterminals can be generalized to STGs and requires polynomial time [4, 22] in the size of the STG.

A naive generalization of the pattern match is to find a compressed ground term in another compressed ground term, which can be solved by translating this problem into a pattern match of compressed preorder traversals of the terms. A generalization of the pattern match is the following submatching problem (also called encompassment): given two (STG-compressed) terms s,t, where s may contain variables, is there an occurrence of an instance of s in t? A special case is matching, where the question is whether there is a substitution σ, such that $\sigma(s) = t$, which is shown to be in PTIME in [7, 8], including the computation of the (unique) compressed substitution.

In this extended abstract (of [23]) we report informally on progress in finding algorithms operating on STGs for answering the submatching question, and which only operate on the STGs. We show that if s is STG-compressed and linear, then submatching can be solved in polynomial time (Theorem 3.7). If s is ground and compressed or s is DAG-compressed, we describe less complex algorithms that solve the submatching question in polynomial time (Theorem 4.1 and Theorem 4.3). In the general case, we describe a non-deterministic algorithm that runs in polynomial time. The deterministic algorithm runs in time $O(n^{c|FVmult(s)|})$ (Theorem 4.4), where n is the size of the STG and $FVmult(s)$ the set of variables occurring more than once in s. This is an exponential-time algorithm, but in a well-behaved parameter.

As an application and an easy consequence of the submatching algorithms, a (single-position or parallel) deduction step on compressed terms by a compressed left-linear rewriting rule can be performed in polynomial time. We also show that a sequence of n rewrites with a STG-compressed left-linear term rewriting system on an STG-compressed target term can be performed in polynomial time (see Theorem 5.1). Our result confirms results on complexity of rewrite derivations under DAG-compression [1], namely that rewrite systems with a polynomial runtime complexity can be implemented such that the algorithm requires polynomial time.

Example 1.1 *Consider the term rewriting rule $f(x) \to g(x,b)$, and let the term $t_1 = f(f(f(a)))$ be compressed as $C_1 \to f(\cdot)$, $C_2 \to C_1C_1$, $T \to C_2(T'), T' \to f(a)$. A single term rewriting step on the compressed term t_1 by the rule $f(x) \to g(x,b)$ would produce $T' \to g(a,b)$, and hence the reduced and decompressed term is $f(f(g(a,b)))$. Other rewriting steps on the compressed term that do not decompress the term have to analyze the contexts. Let another term be $t_2 = f^{16}(a)$, compressed as $C_1 \to f(\cdot)$, $C_2 \to C_1C_1$, $C_3 \to C_2C_2$, $C_4 \to C_3C_3$, $C_5 \to C_4C_4$, $T \to C_5(a)$. A term rewriting step on T using $f(x) \to g(x,b)$ may rewrite the context $f(\cdot)$ and thus would produce $C_1 \to g(\cdot,b)$, and hence reduces the term in one blow to $g(\ldots,(g(\ldots,b)\ldots),b)$, which is a parallel rewriting step, see Section 5.*

The structure of this extended abstract (of [23]) is as follows. First the basic notions, in particular STGs, are introduced in Section 2. An algorithm for linear submatching is explained in Section 3. In Section 4 we explain submatching for some special cases and also a general non-deterministic algorithm for term submatching of compressed patterns and terms. Finally, in Section 5, we illustrate the application in term rewriting and argue that n rewrites for a left-linear TRS can be performed in polynomial time.

2 Preliminaries

We will use standard notation for signatures, terms, positions, and substitutions (see e.g. [2]). A position is a word over positive integers. For two positions p_1, p_2, we write $p_1 \leq p_2$, if p_1 is a prefix of p_2, and

$p_1 < p_2$, if p_1 is a proper prefix of p_2. We call two strings w_1, w_2 *compatible*, if w_1 is a prefix of w_2, or w_2 is a prefix of w_1. We write $p[i]$ for the i^{th} symbol of p, where 0 is the start index, and $p[i,j]$ for the substring of p starting at i ending at j. The set of free variables in a term t is denoted as $FV(t)$. Let $FVmult(s)$ be the set of variables occurring more than once in s. Terms without occurrences of variables are called *ground*. A term where every variable occurs at most once is called *linear*. A *context* is a term with a single hole, denoted as $[\cdot]$. Sometimes it is convenient to view a linear term containing one variable as a context, where the single variable represents the hole. As a generalization, a *multicontext* is a linear term, where the variable occurrences are also called holes. Let $holep(c)$ be the position (as a string of numbers) of a hole in a context c, and let the *hole depth* be the length of $holep(c)$. If $c = c_1[c_2]$ for contexts c, c_1, c_2, then c_1 is a *prefix context* of c and c_2 is a *suffix context* of c. The notation $c[s]$ means the term constructed from the context c by replacing the hole with s. An n-fold iteration of a context c is denoted as c^n; for example c^3 is $c[c[c]]$. A *substitution* σ is a mapping on variables, extended homomorphically to terms by $\sigma(f(t_1,\ldots,t_n)) = f(\sigma(t_1),\ldots,\sigma(t_n))$.

Definition 2.1 *A term rewriting system (TRS) R is a finite set of pairs $\{(l_i, r_i) \mid i = 1,\ldots,n\}$, called* rewrite rules*, written $\{l_i \to r_i\}$, where we assume that for all $i : l_i$ is not a variable, and $FV(r_i) \subseteq FV(l_i)$. A term rewriting step by R is $t \xrightarrow{R} t'$, if for some i: $t = c[\sigma(l_i)]$ and $t' = c[\sigma(r_i)]$ for some context c and some substitution σ.*

2.1 Tree Grammars for Compression

First we introduce string compression: A *straight line program* (SLP) is a context-free grammar that generates one word, has no cycles, and for every nonterminal A there is exactly one production of the form $A \to A_1A_2$ or $A \to a$.

An application for SLPs is the representation of compressed positions in compressed terms. We will use the well-known (polynomial-time) algorithms, constructions and their complexities on SLPs like equality check of compressed strings, computing prefixes, suffixes, the common prefix (suffix) of two strings (see [21, 9, 19, 20, 12, 15, 14]).

We consider compression of terms using tree grammars:

Definition 2.2 *A* singleton tree grammar (STG) *is a 4-tuple $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, \mathcal{R})$, where $\mathcal{TN}$ are tree/term nonterminals of arity 0, $\mathcal{CN}$ are context nonterminals of arity 1, and Σ is a signature of function symbols (the terminals), such that the sets $\mathcal{TN}$, $\mathcal{CN}$, and Σ are finite and pairwise disjoint. The set of nonterminals $\mathcal{N}$ is defined as $\mathcal{N} = \mathcal{TN} \cup \mathcal{CN}$. The productions in $\mathcal{R}$ must be of the form:*

- $A \to f(A_1,\ldots,A_m)$*, where $A, A_i \in \mathcal{TN}$, and $f \in \Sigma$ is an m-ary terminal symbol.*
- $A \to C_1A_2$ *where $A, A_2 \in \mathcal{TN}$, and $C_1 \in \mathcal{CN}$.*
- $C \to [\cdot]$ *where $C \in \mathcal{CN}$.*
- $C \to C_1C_2$*, where $C, C_1, C_2 \in \mathcal{CN}$.*
- $C \to f(A_1,\ldots,A_{i-1},[\cdot],A_{i+1},\ldots,A_m)$*, where $A_1,\ldots,A_{i-1},A_{i+1},\ldots,A_m \in \mathcal{TN}$, $C \in \mathcal{CN}$, and $f \in \Sigma$ is an m-ary terminal symbol.*
- $A \to A_1$ *(λ-production), where A and A_1 are term nonterminals.*

Let $N_1 >_G N_2$ for two nonterminals N_1, N_2, iff $(N_1 \to t) \in R$, and N_2 occurs in t. The STG must be non-cyclic, i.e. the transitive closure $>_G^+$ must be irreflexive. Furthermore, for every nonterminal N of G there is exactly one production having N as left-hand side. Given a term t with occurrences of nonterminals, the derivation of t by G is an exhaustive iterated replacement of the nonterminals by the corresponding

right-hand sides. The result is denoted as $val_G(t)$. We will write $val(t)$ when G is clear from the context. In the case of a nonterminal N of G, we also say that N (or G) generates *$val_G(N)$ or* compresses *$val_G(N)$. The depth of a nonterminal N is the maximal number of $>_G$-steps starting from N, and the depth of G is the maximal depth of all its nonterminals. The size of an STG is the number of its productions, denoted as $|G|$.*

Definition 2.3 *Let G be an STG and V be a set of variables. Then (G,V) is an* STG with variables*, where additional production forms are permitted:*

- *$A \to x$, where $A \in \mathcal{TN}$ and $x \in V$.*
- *$x \to A$ (λ-production), where $x \in V$ and $A \in \mathcal{TN}$.*

This means that variables may be terminals or nonterminals, depending on the existing productions. The measure $Vdepth(N,V)$ is defined as the maximal number of $>_G$-steps starting from N until an element of V or a terminal is reached, and $Vdepth(G,V)$ the maximum.
In the following we always mean STG with variables if variables are present.
An STG G is called a DAG*, if there are no context nonterminals.* □

The compression rate may be exponential in the best case, but not larger: The size of terms represented with an STG G is at most $O(2^{|G|})$. Note that the term depth of DAG-compressed terms is at most the size of the DAG, whereas the term depth of STG-compressed terms may be exponential in the size of the STG. Note also that every subterm in a DAG-compressed term is represented by a nonterminal, whereas in STG-compressed terms, there may be subterms that are only implicitly represented. It is known that several computations in SLPs and STG, for example length computations, can be done in polynomial time. Several forms of extensions of STGs are well-behaved, such that even a sequence of n such extensions will lead to only polynomial size growth.

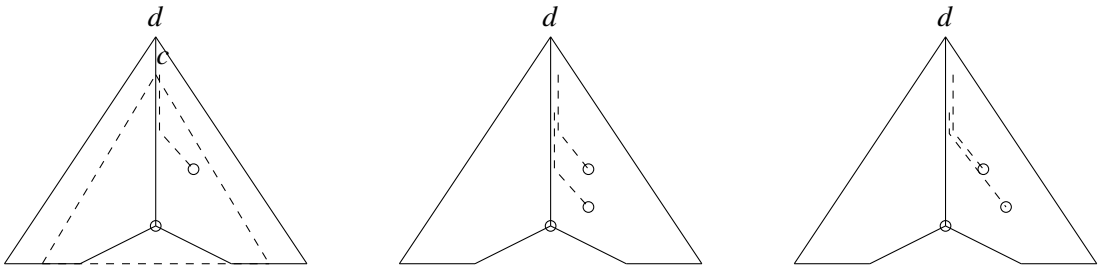
Compressed Matching. The investigation in [7] shows that (exact) term matching, also in the fully compressed version including the computation of a compressed substitution, is polynomial. I.e. given two nonterminals S,T, where S may contain variables, there is a polynomial time algorithm for answering the question whether there is some substitution σ such that $\sigma(val(S)) = val(T)$, and also for computing the substitution, where the representation is a list of variable-nonterminal pairs, and the nonterminals belong to an extension of the input STG.

Compressed Submatching. Given two first-order terms s,t, where s (the pattern) may contain variables, the submatching problem is to identify an instance of s as a subterm of t. Submatching (also called encompassment relation) is a prerequisite for term rewriting.

Definition 2.4 *The* compressed term submatching problem *is:*
*Assume given a term s which may contain variables, and a (ground) term t, both compressed with an STG $G = G_S \cup G_T$, such that $val(T) = t$ and $val(S) = s$ for term nonterminals $S \in G_S$, $T \in G_T$. The task is to compute a (compressed) substitution σ such that $\sigma(s)$ is a subterm of t; also the (compressed) position (all positions) p of the match in t should be computed. Specializations are:*uncompressed *if s is given as a plain term without any compression;* ground *if s is ground;* DAG-compressed*, if s is DAG-compressed; and* linear*, if s is a linear term, i.e. every variable occurs at most once in s.*

Lemma 2.5 *Given an STG G, a term s and a nonterminal T, with $val_G(T) = t$, where t is ground. If there is some substitution σ, such that $\sigma(s)$ is a subterm of t, then there are the following possibilities:*

1. *There is a term nonterminal B of G such that $val_G(B) = \sigma(s)$.*



(a) non-compatible overlap (b) parallel (c) sequential
Subfigures (b) and (c) only show the hole path of two occurrences of the context c.

Figure 1: Non-compatible, parallel and sequential overlap of c with d

2. *There is a production $B \to CB'$ in G, such that $\sigma(s) = c[val_G(B')]$, where c is a nontrivial suffix context of $val_G(C)$. There are subcases for the hole position p of c.*
   (a) *(overlap case) p is a position in s.*
   (b) *$p = p_1p_2$, where p_1 is the maximal prefix of p that is also a position in s. Then $s_{|p_1} = x$ is a variable. The algorithms below have to distinguish the* subterm case *where x occurs more than once in s and the* subcontext case *where x occurs exactly once in s.*

3 Term Submatching with Linear Terms

Overlaps of Linear Terms and Contexts. An important concept and technique used is periodicity of contexts. This is a generalization of periodicity of strings: for example the string "bcabcabc" is periodic with period length 3. A context c is called periodic if $c = d^n d'$ for some contexts d, d' and a positive integer n, where d' is a prefix of d. This is even generalized to multicontexts c (linear terms, where the variables are the holes), and where periodicity means that c can be overlapped with itself at periodic positions without conflicts.
We consider overlapping multicontexts $c, c_1, c_2, \ldots$ and a context d. In particular special variants of overlaps have to be analyzed: Overlaps where the hole of d is not compatible with any hole of c. The overlaps where a hole of c is compatible with a hole of d can be dealt with generalizing results from words (or words with character-holes). If there are non-compatible overlaps of copies of c with d, then only two configurations are possible: parallel and sequential (see Proposition 3.2 and Fig. 1), and there are no mixed configurations. Thus, periodicities in linear terms are not only possible along the hole-path of d but also along other paths, and there are two different kinds of such periodicities: the parallel and the sequential variant. A helpful technical result is a periodicity theorem that tells us that a multi-context c is periodic, if there is a multiple overlap of $h+2$ copies of c where h is the number of holes, and the overlap is sufficiently dense. This will be used in the submatching algorithm for linear terms.

Example 3.1 *Let $d = f(a_1, f([\cdot], a_1))$ and let $c = f(a_1, [\cdot])$. Then c overlaps d at position ε, which is a compatible overlap, since the start as well as the hole position of c is on the hole path of d. The overlap of c with d at position 2 (in d) is a non-compatible overlap, since the hole of c is at 2.2, which is not a prefix or suffix of the hole path of d, which is 2.1.*

Proposition 3.2 *Let c be a multicontext with at least one hole, and let d be a context with exactly one hole, and let $p_1 < p_2$ be two positions of non-compatible overlaps of c in d. Let q_i be the maximal common hole path (mchp) of c at p_i for $i = 1,2$. Then there are the following two cases (see Figure 1):*

1. *$q_1 = q_2$ (the* parallel overlap *case). Then for p' such that $p_1 p' = p_2$ the path $p_1(p')^n$ is compatible with $holep(d)$ for all n. Also, this is a multiple overlap of c' with itself at positions $(p')^i$, where c' is constructed from c with an extra hole at p'', where $p_1 p'' = holep(d)$.*
2. *$q_2 < q_1$ (the* sequential overlap *case). Then $p_2 q_2 = p_1 q_1$. I.e., there is a fixed position on the hole path of d, where the hole paths of occurrences of c deviate.*

Example 3.3 *Let $c' = f(f(a_1,a_2),[\cdot])$ be a context, $c = f(f(x,y),(c')^{100}[.])$, and let $d = (c')^{100}[\cdot]$. Then there is an overlap of c with d at positions $\varepsilon, 2, 2.2, \ldots$. It is an overlap of the first kind, i.e. a parallel overlap. A sequential overlap is the following: Let $c = f(a_1, f(a_1, f(a_1, [\cdot])))$ and let $d = f(a_1, f(a_1, f(a_1, f(a_1, f([\cdot], f(a_1, f(a_1, a_1)))))))$. Then the overlap positions are $\varepsilon, 2, 2.2, 2.2.2$.*

Theorem 3.4 (Periodicity-Theorem) *Let c be a multi-context with $h \geq 1$ holes. Let p be the position of a fixed hole of c, and let $p_i, i = 1, \ldots, n$ be prefixes of p such that $i < j$ implies $p_i < p_j$ with $n \geq h+2$. Assume that there is a (right-cut) overlap of n copies of c starting at position p_i such that p is a prefix of $p_i p$, i.e., the hole position of c starting at p_i is compatible with p for all i, and only positions in c at p_1 are relevant for the overlap. Let p_{max} be $\max\{|p_{i+1}| - |p_i| \mid i = 1, \ldots, n-1\}$. Assume $|p| - |p_n| \geq 2h \cdot p_{max}$; this means there are $2h \cdot p_{max}$ common positions on the path p of all occurrences of c.*
Then the multicontext c is periodic (in the direction p), and a period length is $p_{all} := gcd(|p_2| - |p_1|, |p_3| - |p_2|, \ldots, |p_n| - |p_{n-1}|)$. Moreover, the overlap is consistent with using the same substitution for the variables for every occurrence of c.

Tabling Prefixes of Multicontexts in Contexts.
The core of the algorithm for finding submatches of a linear term s in other terms (under STG-compression) is the construction of a table in dynamic-programming style. The table contains overlaps of s with contexts that are explicitly represented in the STG G by a context nonterminal. In fact the table is split into several tables: There is a table per context nonterminal A of G and per variable (hole) of s for the compatible overlaps. In addition there is an extra table for non-compatible overlaps. This makes $h+1$ tables where h is the number of variables of s.
The entries in the tables are pairs of a position and a substitution necessary for the overlap. Since terms of exponential size and depth may be represented in the STG G, a compact representation of a large number of entries is necessary in order to keep the tables of polynomial size. Indeed this is possible exploiting periodicity. If the number of entries in a table are sufficiently dense, then the periodicity theorem implies that a large subset of the entries enjoys regularities, and a series of periodic overlaps can be represented in one entry, consisting of: a start position, a period (a position, respectively a context nonterminal), and the number of successive entries.
In more detail, the construction of the prefix tables is bottom-up w.r.t. the grammar where the productions $A \to A_1 A_2$ for context nonterminals permit to construct the A-tables from the A_1,A_2-tables, and where the start are the contexts with hole-depth 1. This construction must take into account the compact representation of the entries: single ones and periodic ones, which makes the description of the algorithm rather complex due to lots of cases. The construction of the prefix table in the case $A \to A_1 A_2$ and the periodic cases is depicted in Figure 2 where (a) shows the case where A has a periodic suffix, (b) shows the case where A has an inner part that is periodic, (c) shows a case where the periodicity goes into a direction that is not compatible with the hole of A_2, which leads to the sequential overlap case; and (d) is a case of a sequential overlap already in the table for A_1. The generation of the periodic entries is done in

an extra step: compaction, where the periodic overlaps are detected by searching for sufficiently dense entries. This is the only place where periodic entries are generated.
In addition to the prefix tables there is a result table, which contains the detected submatchings, and which is maintained during construction of the prefix tables.
Since it is necessary to also have submatchings in terms, i.e. for term nonterminals, we keep things simple and assume that every production for a term nonterminal is of the form $A \to CA_1$, where A_1 is a term nonterminal with production $A_1 \to a$, i.e. a constant. This rearrangement of G can be done efficiently, and thus does not restrict generality. For these nonterminals the extraction of the submatchings can be done using the already constructed prefix-tables.
Note that during construction of the tables, the STG G may have to be extended in every step.

Example 3.5 *We describe several small examples for compatible entries in a prefix table. Therefore we slightly extend Example 3.3. Let the STG be* $S \to A; A \to A_1A_1; A_1 \to A_2A_2, A_2 \to f(a_1,[\cdot])$.

1. *Then* (C,A_2,∞) *for* $C \to [\cdot]$ *is a potential entry in a result table for A.*
2. *Let* $A_4 \to g([\cdot]), B \to A_4A, C' \to A_4$. *Then* (C',A_2,∞) *is an entry in the result table for B.*
3. *Let* $B' \to BA_4$, *then* $(A_4,A_2,2)$ *is a potential entry in the result table for* B'.
4. *The tuple* $(A_4,A_2,3)$ *is an entry in the prefix table for B.*
5. *Let* $B'' \to A_6A_4, A_6 \to A_4A_1$. *The context* A_6 *is then a potential entry in the result and prefix tables of* B''.

Note that item 4 cannot be used as a result, since composing B as in $B' \to BA_4$ *in item 3, may render an overlap invalid.*

Example 3.6 *We describe an example for a non-compatible entry in a prefix table. Therefore we slightly modify Example 3.3. Assume there is an STG G. Let* $c = f(a_1,f(a_1,f(a_1,f(a_1,[\cdot]))))$, $d = f(a_1,f(a_1,f(a_1,f([\cdot],f(a_1,f(a_1,a_1))))))$, *and let* P,D,C_0,S *be a nonterminals such that* $val(P) = f(a_1,[\cdot])$, $val(D) = d, val(S) = c$, $val(C_0) = [\cdot]$. *Then an entry in the non-compatible prefix table for D could be* $(C_0,P,3)$.

Theorem 3.7 (Linear Submatching) *Let G be an STG, and* S,T *be two term nonterminals such that* $val(S)$ *is a linear term, and the submatching positions of* $val(S)$ *in* $val(T)$ *are to be determined. Then the algorithm for linear submatchings computes an* $O(|G|^5)$*-sized representation of all submatchings of* $val(S)$ *in* $val(T)$ *in polynomial time dependent on the size of G.*

4 Submatching Algorithms for Other Cases

We consider several specialized situations: ground terms, uncompressed patterns, DAG-compressed terms, and also non-linear terms.

4.1 Ground Term Submatching

If s is ground and compressed by a nonterminal S then submatching can be solved in polynomial time by translating both compressed terms into their compressed preorder traversals (i.e. strings) [4, 5] and then applying string pattern matching [21, 15]. The string matching algorithm in [15, 11] computes a polynomial representation of all occurrences. Note that in our case, the structure of ground terms is

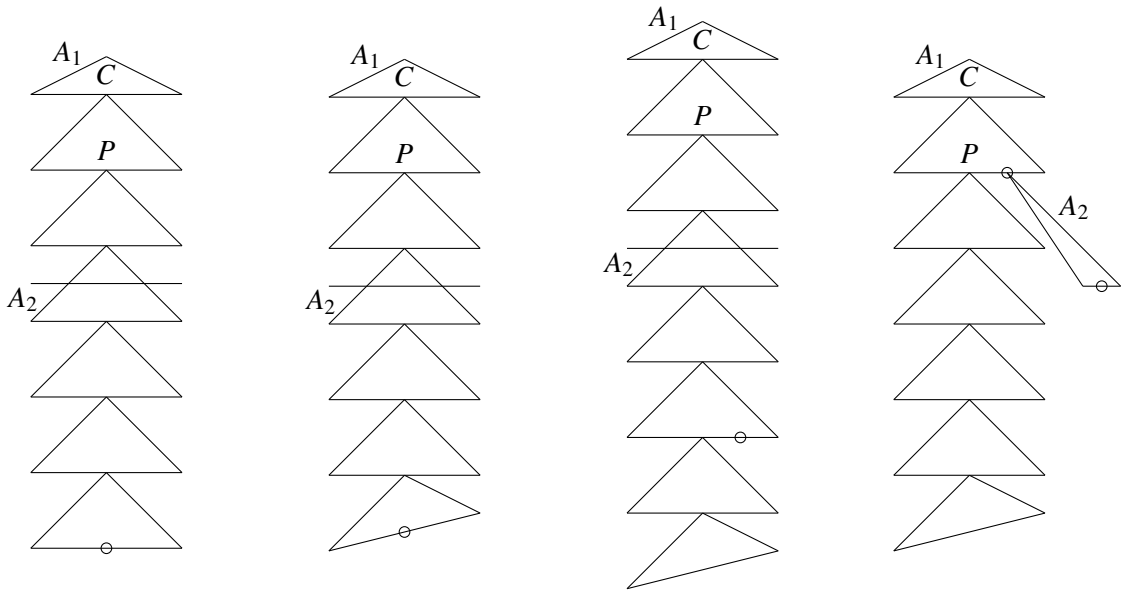


Figure 2: Cases in the construction of the prefix tables for periodic entries

very special as a string matching problem: periodic overlaps of the preorder traversal as strings are not possible. Thus the complete output of the algorithm is as follows: (i) a list of term nonterminals N of the input STG G, where $val(\sigma(S)) = val(N)$, and (ii) a list of pairs (N, p), where the production for N is of the form $N \to CN'$, p is a compressed position, and $val(C)_{|val(p)}[val(N')] = val(S)$. Moreover, every nonterminal N appears at most once in the list.
The required time for string matching is $O(n^2m)$ where n is the size of the SLP of T and m is the size of the SLP of S. Since the preorder traversal can be computed in linear time (see [8]), we have:

Theorem 4.1 *The ground compressed term submatching can be computed in time $O(|G_T|^2|G_S|)$, and the output is a list of linear size.*

4.2 DAG-Compressed Non-Linear Submatching

Now we look for the case of DAG-compressed s, which is slightly more general than the uncompressed case, and where variables may occur several times in s. Also for this case, there is an algorithm for submatching that requires polynomial time. The algorithm outputs enough information to determine all the positions and substitutions of a submatch.

Example 4.2 *The number of possible substitutions for a submatch in a DAG-compressed term may be exponential: Let the productions be $S \to f(x,y)$, and $T \to f(A_1,A_1), A_1 \to f(A_2,A_2), \ldots, A_{n-1} \to f(A_n,A_n), A_n \to a$. Then $val(T)$ is a complete binary tree of depth n and there is a submatch at every non-leaf node. Clearly, it is sufficient to have all A_i as submatchings in the output, which is of linear size.*

In the case of a DAG-compressed or uncompressed pattern-term (not necessarily linear) s and STG-compressed target term t, the algorithm for computing all submatchings is designed in dynamic programming style. It constructs a table of possible submatchings of s in the context nonterminals corresponding

(a) (b)

Figure 3: Cases in the construction of the s-in-C-table for DAG-compression

to t. The key of the table is (C,p), where C is a context nonterminal, and p a position that is a suffix of $val(C)$ as well as a position in s. The number of these positions is linear in $|G_s|+|G_t|$ for every context. The entries are substitutions into the variables of s, i.e. a list of pairs (x_i,A_i), where A_i is a term nonterminal representing a ground term. There is also a result list of found submatchings in contexts C contributing to T, and term nonterminals for ground terms that are instances of s. The construction proceeds again bottom-up in the STG G_t for context nonterminals, and for $A \to A_1A_2$, constructs the table for A from the tables for A_1,A_2, and in case a full submatching is found, inserts a result into the result list.
Finally, from these information, a representation of all submatchings can be constructed by looking at the right hand sides of the productions $A \to CB$ for term nonterminals, and using the table entries for C, and also constructing the occurrences of the ground terms.

Theorem 4.3 *Let G be an STG, and S,T be two term nonterminals such that S is DAG-compressed. Then the submatch computation problem can be solved in polynomial time. Also an explicit polynomial representation of all matching possibilities can be computed in polynomial time.*

4.3 A Non-Deterministic Algorithm for Sub-Matching in the General Case

The submatching problem for STG-compressed pattern terms that may be nonlinear can be solved by a relatively easy search that leads to a non-deterministic polynomial time algorithm: Given S, with non-linear $s = val(S)$, extract and construct a nonterminal B representing a subterm $f(r_1,\ldots,r_n)$ of s such that two terms r_i,r_j contain a common variable. Then non-deterministically choose a right hand side r of a production of G_t of the form $f(\ldots)$, then compute the usual match of B with r using [7] which will produce an instantiation of at least one variable of $val(B)$, and hence of s. Then iterate this until all variables with double occurrences are instantiated. For the resulting linear term we know how to find all matching positions.

Theorem 4.4 (Nondeterministic General Submatch) *Let G be an STG and S,T be two nonterminals of G where $val(S)$ may contain variables. Then the algorithm for fully compressed submatching for compressed terms s,t requires at most searching in $|G|^{|FVmult(s)|}$ alternatives for the substitution and the computation for one alternative can be done in polynomial time. Thus the submatching problem is in NP.*

There remains a gap in the knowledge of the complexity of the fully compressed submatching problem for terms, which for the decision problem is between PTIME and NP.

Remark 4.5 *The non-linear submatching problem can be computed in polynomial time if there are few variable occurrences ($\leq |G|$) in s: First linearize s, then use the linear compressed submatch and then perform a postprocessing checking equality enforced by the variables of s.*

5 Polynomial Compressed Term Rewriting

For our compressed representation the natural approach to rewriting is to use parallel rewriting of the same subterm at several positions and by the same rewriting rule. Note, however, that the set of redexes that are rewritten in parallel will depend on the structure of the STG G_t, and not on the structure of the rewritten term t.
Let R be a compressed TRS, let t be a ground term with $val_G(T) = t$, let R be compressed by the STG G_R as $\{L_i \to R_i \mid i = 1,\ldots,n\}$ where L_i, R_i are term nonterminals.
A (parallel) term rewriting step is performed as follows:
First select $L_i \to R_i$ as the rule. There is an oracle, which is one of our submatching algorithms applied to L_i, for finding the redex for $val(L_i)$ or the set of redexes that provides the following:

1. An extension G' of G, i.e. additional nonterminals and productions.
2. A substitution σ as a list of pairs: $\{x_1 \mapsto A_1,\ldots,x_m \mapsto A_m\}$, where $FV(val(L_i)) = \{x_1,\ldots,x_m\}$, A_i are term nonterminals in G', and $val(A_i)$ is a subterm of t. It is also assumed that the instantiation is integrated in the grammar G' as productions $x_i \to A_i$ for $i = 1,\ldots,m$.
3. A term nonterminal A (corresponding to L_i) in G' which contributes to $val(T)$, and a compressed position p.

Then the rewriting step is performed by modifying the grammar such that somewhere in the part of the grammar contributing to t: L_i is replaced by R_i. This will also generate an extension of G_t on the fly and also a copy of the STG G_R is made.
A single-position rewriting step under STG-compression is performed in a similar way.

Theorem 5.1 *Let R be a TRS compressed with G_R and t be a term compressed with an STG G. Then a sequence of n term rewriting steps where submatching is a non-deterministic oracle that is not counted, can be performed in polynomial time. The size increase by n term rewriting steps is $\mathcal{O}\big(|G_R|^2 n^7\big(|G|^2 + |G|(\log n + 2|G_R|) + (\log n + |G_R|)^2\big)\big)$.*

The complexity bound is $\mathcal{O}(n^7 \log^2(n))$ depending on the number n of rewrites; $\mathcal{O}(|G_0|^2)$ depending on the size of G_T; and $\mathcal{O}(|G_R|^4)$ depending on the size of G_R. Note that the degree of the polynomial for the estimation of the worst case running time is worse than the space bound. The term rewriting sequence has to be constructed (+ 1) and Plandowski equality check has to be used in every construction step, which contributes a factor of 3 in the exponent. But note that there are faster deterministic tests [15, 11] and even faster randomized equality checks [10, 3, 24].
Single-position rewriting requires a partial decompression of the redex position (similar to the parallel), which leads to an extra increase in the size of the STG, but to the same, still polynomial, complexity.
Combining the results on submatching and sequences of rewriting, we obtain the following corollaries:

Corollary 5.2 *Let R be an STG-compressed TRS and t be an STG-compressed term. Then a sequence of n term rewriting steps using the submatching algorithm in Subsection 4.3 can be performed in non-deterministic polynomial time.*

Proof. This follows from Theorems 5.1 and 4.4. □

Corollary 5.3 *Let R be a left-linear STG-compressed TRS and t be an STG-compressed term. Then n term rewriting steps where the submatching algorithms in Subsection 4.3 are used can be performed in polynomial time.*

Proof. This follows from Theorems 5.1 and 3.7. □

Corollary 5.4 *Let R be a TRS with DAG-compressed left-hand sides and STG-compressed right hand sides and let t be an STG-compressed term. Then n term rewriting steps where the submatching algorithm in Subsection 4.2 is used can be performed in polynomial time in n.*

Proof. This follows from Theorems 5.1 and 4.3. □

Corollary 5.5 *Let R be an STG-compressed TRS and t be an STG-compressed term, such that the left hand sides of every rule has at most $|G|$ occurrences of variables. Then n term rewriting steps (see Remark 4.5) can be performed in polynomial time in n.*

6 Conclusion

We have constructed several polynomial algorithms for finding a submatch under STG-compression, or restrictions thereof. It is also shown that n rewrite steps can be performed in polynomial time under STG-compression in several cases: left-linear and STG-compressed TRS, DAG-compressed or ground left hand sides of rules. Also in the general case of non-linear left hand sides n rewrites can be performed non-deterministically in polynomial time, where a search for a redex is required. This is connected to the open problem of the exact complexity of computing submatches also for non-linear terms.
A connection to the results in [1] on polynomial runtime complexity is that our results also imply that for TRSs with polynomial runtime complexity the (single-position and parallel) rewriting can be implemented such that n rewrite steps can be performed in polynomial time.
A remaining open question is whether the general STG-compressed submatching (of nonlinear terms s in t) can be solved in polynomial time or not.

References

[1] Martin Avanzini & Georg Moser (2010): *Closing the Gap Between Runtime Complexity and Polytime Computability*. In Christopher Lynch, editor: *21st RTA*, *LIPIcs* 6, Schloss Dagstuhl, Germany, pp. 33–48, doi:10.4230/LIPIcs.RTA.2010.33.

[2] Franz Baader & Tobias Nipkow (1998): *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA.

[3] Piotr Berman, Marek Karpinski, Lawrence L. Larmore, Wojciech Plandowski & Wojciech Rytter (2002): *On the Complexity of Pattern Matching for Highly Compressed Two-Dimensional Texts*. *J. Comput. Syst. Sci.* 65(2), pp. 332–350, doi:10.1006/jcss.2002.1852.

[4] Giorgio Busatto, Markus Lohrey & Sebastian Maneth (2005): *Efficient Memory Representation of XML Documents*. In: *Proceedings of DBPL 2005*, *LNCS* 3774, pp. 199–216, doi:10.1007/11601524_13.

[5] Giorgio Busatto, Markus Lohrey & Sebastian Maneth (2008): *Efficient Memory Representation of XML Document Trees*. *Information Systems* 33(4–5), pp. 456–474, doi:10.1016/j.is.2008.01.004.

[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison & M. Tommasi (1997): *Tree Automata Techniques and Applications*. Available at http://www.grappa.univ-lille3.fr/tata. Release October 2002.

[7] Adrià Gascón, Guillem Godoy & Manfred Schmidt-Schauß (2008): *Context Matching for Compressed Terms*. In: *23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008)*, IEEE Computer Society, pp. 93–102, doi:10.1109/LICS.2008.17.

[8] Adrià Gascón, Guillem Godoy & Manfred Schmidt-Schauß (2011): *Unification and matching on compressed terms*. *ACM Trans. Comput. Log.* 12(4), pp. 26:1–26:37. Available at http://doi.acm.org/10.1145/1970398.1970402.

[9] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski & Wojciech Rytter (1996): *Efficient Algorithms for Lempel-Ziv Encoding (Extended Abstract)*. In Rolf G. Karlsson & Andrzej Lingas, editors: *SWAT*, *Lecture Notes in Computer Science* 1097, Springer, pp. 392–403, doi:10.1007/3-540-61422-2_148.

[10] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski & Wojciech Rytter (1996): *Randomized Efficient Algorithms for Compressed Strings: The Finger-Print Approach (Extended Abstract)*. In: *7th CPM 96*, *Lecture Notes in Computer Science* 1075, Springer, pp. 39–49, doi:10.1007/3-540-61258-0_3.

[11] Artur Jez (2012): *Faster Fully Compressed Pattern Matching by Recompression*. In: *ICALP (1)*, *Lecture Notes in Computer Science* 7391, Springer, pp. 533–544, doi:10.1007/978-3-642-31594-7_45.

[12] Marek Karpinski, Wojciech Rytter & Ayumi Shinohara (1995): *Pattern-matching for strings with short description*. In: *CPM '95*, *LNCS* 937, Springer-Verlag, pp. 205–214, doi:10.1007/3-540-60044-2_44.

[13] Jordi Levy, Manfred Schmidt-Schauß & Mateu Villaret (2006): *Bounded Second-Order Unification is NP-complete*. In: *Term Rewriting and Applications (RTA-17)*, *LNCS* 4098, Springer, pp. 400–414, doi:10.1007/11805618_30.

[14] Jordi Levy, Manfred Schmidt-Schauß & Mateu Villaret (2008): *The Complexity of Monadic Second-Order Unification*. *SIAM J. of Computing* 38(3), pp. 1113–1140, doi:10.1137/050645403.

[15] Yury Lifshits (2007): *Processing Compressed Texts: A Tractability Border*. In: *CPM 2007*, *LNCS* 4580, Springer, pp. 228–240. Available at http://dx.doi.org/10.1007/978-3-540-73437-6_24.

[16] Markus Lohrey (2012): *Algorithmics on SLP-compressed strings. A survey*. *Groups Complexity Cryptology* 4(2), pp. 241–299, doi:10.1515/gcc-2012-0016.

[17] Markus Lohrey, Sebastian Maneth & Manfred Schmidt-Schauß (2009): *Parameter Reduction in Grammar-Compressed Trees*. In: *12th FoSSaCS*, *LNCS* 5504, Springer, pp. 212–226, doi:10.1007/978-3-642-00596-1_16.

[18] Markus Lohrey, Sebastian Maneth & Manfred Schmidt-Schauß (2012): *Parameter reduction and automata evaluation for grammar-compressed trees*. *J. Comput. Syst. Sci.* 78(5), pp. 1651–1669, doi:10.1016/j.jcss.2012.03.003.

[19] Wojciech Plandowski (1994): *Testing equivalence of morphisms in context-free languages*. In: *ESA 94*, *Lecture Notes in Computer Science* 855, pp. 460–470, doi:10.1007/BFb0049431.

[20] Wojciech Plandowski & Wojciech Rytter (1999): *Complexity of Language Recognition Problems for Compressed Words*. In: *Jewels are Forever*, Springer, pp. 262–272, doi:10.1007/978-3-642-60207-8_23.

[21] Wojciech Rytter (2004): *Grammar Compression, LZ-Encodings, and String Algorithms with Implicit Input*. In J. Diaz et. al., editor: *ICALP 2004*, *LNCS* 3142, Springer-Verlag, pp. 15–27, doi:10.1007/978-3-540-27836-8_5.

[22] Manfred Schmidt-Schauß (2005): *Polynomial Equality Testing for Terms with Shared Substructures*. Frank report 21, Institut für Informatik. FB Informatik und Mathematik. Goethe-Universität Frankfurt.

[23] Manfred Schmidt-Schauss (2013): *Linear Pattern Matching of Compressed Terms and Polynomial Rewriting*. Accepted for publication, 2013.

[24] Manfred Schmidt-Schauss & Georg Schnitger (2012): *Fast Equality Test for Straight-Line Compressed Strings*. *Information processing letters*, doi:10.1016/j.ipl.2012.01.008.

[25] Jacob Ziv & Abraham Lempel (1977): *A Universal Algorithm for Sequential Data Compression*. *IEEE Transactions on Information Theory* 23(3), pp. 337–343, doi:10.1109/TIT.1977.1055714.

Evaluating functions as processes

Beniamino Accattoli
Carnegie Mellon University - Pittsburgh, PA, US

A famous result by Milner is that the λ-calculus can be simulated inside the π-calculus. This simulation, however, holds only modulo strong bisimilarity on processes, i.e. there is a slight mismatch between β-reduction and how it is simulated in the π-calculus. The idea is that evaluating a λ-term in the π-calculus is like running an environment-based abstract machine, rather than applying ordinary β-reduction. In this paper we show that such an abstract-machine evaluation corresponds to linear weak head reduction, a strategy arising from the representation of λ-terms as linear logic proof nets, and that the relation between the two is as tight as it can be. The study is also smoothly rephrased in the call-by-value case, introducing a call-by-value analogous of linear weak head reduction.

Introduction

A key result about the expressiveness of the π-calculus is that it can represent the λ-calculus, as it has been showed by Robin Milner [33]. During the nineties the relationship between the two systems has been explored in-depth, mostly by Davide Sangiorgi [36, 37] and Gérard Boudol [14, 13]. Nowadays, it takes a relevant part in the standard reference for the π-calculus [38], and in any introductory course about it. From the process calculus point of view, it helps in getting deeper insights into its theory, especially because the π-calculus is far less canonical then the λ-calculus. From the λ-calculus point of view, it provides new tools to analyze the behavior of λ-terms and the dynamics of β-reduction.

The idea is that the π-calculus can be considered as a sort of flexible abstract machine to which the λ-calculus can be compiled in various ways. There are in fact various encodings, each one corresponding to a particular evaluation strategy in the λ-calculus. In particular, Milner showed that Plotkin's call-by-name and call-by-value strategies [35] can be both faithfully represented.

The way in which the representation is *faithful*, however, is quite subtle. It is looser than what one might expect, as the diagram in Figure 1.a *does not hold*. It is only possible to get the diagram in Figure 1.b: P_t, the process representing t, does not reduce to P_s, but to a process Q which is strongly bisimilar to P_s. One might think that a better encoding could solve this problem, but this is a naïve expectation: the two systems compute in radically different ways, the mismatch is inherent. In Milner's result P_s and Q are strongly bisimilar, which means that they behave the same *externally*, *i.e.* in their

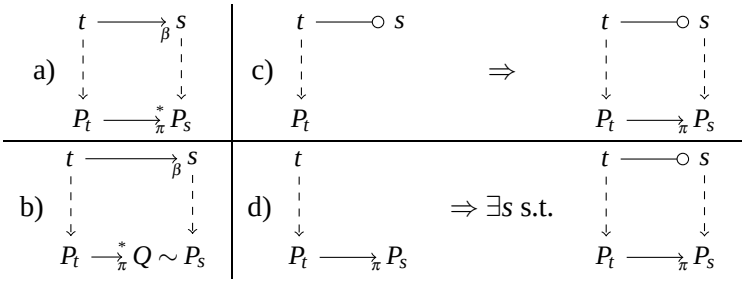


Figure 1: Diagrams describing the relationship between terms and processes.

R. Echahed and D. Plump (Eds.): 7th International Workshop on
Computing with Terms and Graphs
EPTCS 110, 2013, pp. 41–55, doi:10.4204/EPTCS.110.6

interactions with every possible environment. However, the two processes behave in a quite different way *internally*, *i.e.* with respect to reductions. The discrepancy concerns the granularity of evaluation: λ-calculus uses a coarse, big-step substitution rule, while the π-calculus evaluates in small, fine-grained steps, as an abstract machine. Nonetheless, the evaluation of t terminates if and only if the evaluation of the corresponding process P_t terminates. In this sense, the representation is sometimes said to be sound and complete.

This paper refines the relationship between the λ-calculus and the π-calculus by extending the former with explicit substitutions—which may be considered as an alternative to abstract machines—in order to get a closer match of reduction steps. In the call-by-name case we show that the strategy corresponding to the evaluation in the π-calculus is exactly *linear weak head reduction* $\multimap$, the small-step head strategy of linear logic proof nets [29, 3]. This notion of evaluation has connections with Krivine's abstract machine [20], Bohm's separation theorem [29], computational complexity [9], the geometry of interaction [19], game semantics [18, 17], and the differential λ-calculus [24]. The relationship shown here is extremely strong. It is represented in the diagrams in Figure 1.c-d, which hold modulo structural equivalence only. They express the fact that the translation is a strong bisimulation *with respect to reduction* (note that *one* step maps to *one* step, and vice-versa).

The relationship between the π-calculus and linear logic has been analyzed from various points of view [31, 1, 12, 11, 27, 23, 15]. Our study essentially refines the work of Caires, Pfenning, and Toninho in [39], where the encodings of the λ-calculus in the π-calculus are re-understood as the encodings of λ-calculus into linear logic (due to Girard [26], see also [28]). The refinement consists in looking to such encodings via linear logic proof nets, but replacing the explicit use of proof nets with the lighter and equivalent reformulations as calculi of explicit substitutions *at a distance*, developed in [7, 8, 2, 10, 3, 5].

Contributions. In some sense there is not much original content in this paper. Damiano Mazza's master thesis [30] (in French and unpublished) already developed the connection with linear weak head reduction. Similar ideas are sketched by Boudol in the introduction of [13]. Also, Milner's seminal paper already suggested to use some environment device to refine the encodings, an idea that has then been explored by Vasconcelos [40] and recently by Cimini, Sacerdoti Coen, and Sangiorgi [16].

What is original here is the presentation. Our approach provides a remarkably compact development, confirming the relevance of explicit substitutions *at a distance* as a very flexible syntactical tool. Our presentation simplifies in the extreme Mazza's study, by exploiting the simpler and more manageable reformulation of weak linear head reduction in the *linear substitution calculus* [9, 3]. In addition, by clarifying the connection with a crucial concept in the theory of linear logic, we get an important corollary *for free*. In [9] it is proven that linear head reduction is at most quadratically longer than head reduction, and this result holds also with respect to the weak (*i.e.* not under lambdas) variants of these reductions[1]. Plotkin's call-by-name strategy is the same thing as weak head reduction. Consequently, we get a quadratic relation between the call-by-name strategy and the evaluation in the π-calculus, which is a non-trivial quantitative refinement of Milner's result.

However, our contribution is not only about the presentation. The study of call-by-name is complemented by the study of a call-by-value encoding, from which we extract a call-by-value $\multimap_{\mathrm{v}}$ analogous of linear weak head reduction, which has never been considered before. We also show that this new strategy enjoys the analogous of the *subterm property* [9] of linear weak head reduction, which is the basic property for complexity analysis. Last but not least, we give a presentation *at a distance* of the

[1] The upper bound in [9] is exact, and it is based on a trasformation of reductions which applies to arbitrary reduction sequences, in particular even to non-terminating terms. For instance, the quadratic bound is reached by the evaluation of $(\lambda x.xx)\lambda x.xx$, which is weak.

rewriting rules of the π-calculus which is a contribution of independent interest.

Despite the compactness of the presentation, the details turned out to be quite delicate. The use of *distance rules*, which are rewriting rules involving contexts (*i.e.* terms with holes), is crucial. They reflect on terms the local rules of linear logic proof nets, and they are essential in order to get a strong bisimulation of reductions. These contexts can capture variables and names, a fact which requires a very careful analysis of the translations. This is why we present the proofs of the translation in details, almost certifying the result. Moreover, we use colors to ease the reading, so we suggest to read the paper simultaneously on paper and on a computer screen.

The relationship with proof nets. Proof nets do not appear in this paper, we limit ourselves to the equivalent formulations as calculi at a distance. However, for the call-by-value calculus the detailed correspondence between terms and proof nets can be found in [5] (which uses big-step rules, while here we use small-step rules), for call-by-name the interested reader may have a look to [7, 2] (that do employ small-step rules, but in a slightly different way). On proof nets, linear head reduction is the small step strategy which reduces only the cuts at level 0 which do not involve the auxiliary conclusions of !-boxes. The weak variant can be defined in exactly the same way if boxes are also used for $⅋$ (which in this context rather corresponds to the right rule for linear implication in intuitionistic linear logic, and not to the $⅋$ of classical linear logic). Using boxes for linear implication is less *ad-hoc* than it may seem at first sight; a technical discussion of this issue is in Section 6 of [5]. This paper provides another justification for such boxes: they are needed to properly reflect evaluation in the π-calculus.

Plan of the paper. Section 1 introduces the linear substitution calculus, and Section 2 introduces the presentation of the π-calculus that we use. Sections 3 and 4 study the call-by-name and the call-by-value encodings, respectively.

Acknowledgements. To Frank Pfenning, for having encouraged me to work out the details of this work, and to Damiano Mazza, for inspiration and comments on an early draft. This work was partially supported by the Qatar National Research Fund under grant NPRP 09-1107-1-168.

1 The linear substitution calculus

The language of the *linear substitution calculus* λ_{lsub} is given by the following grammar for terms:

$$t,s,u,r \quad ::= \quad x \mid \lambda x.t \mid ts \mid t[x/s]$$

The constructor $t[x/s]$ is called an *explicit substitution* (of s for x in t, the usual (implicit) substitution is instead noted $t\{x/s\}$). Both $\lambda x.t$ and $t[x/s]$ bind x in t. We are not going to define the full calculus (for which we refer to [9, 3]), but only linear weak head reduction. However, let us point out that the linear substitution calculus is a variation over a calculus of explicit substitutions introduced by Robin Milner in [34], to analyze the translation of λ-calculus to Bigraphs.

We shall use contexts extensively, so we define them formally. In particular, we need to specify the set Δ of variables captured by a given context. A weak head context, or simply an **evaluation context**, is a term of the following grammar (to ease the reading on screen all contexts will be in blue):

$$E_\emptyset \quad ::= \quad (\!|\cdot|\!) \mid E_\emptyset t \qquad\qquad E_{\Delta\uplus\{x\}} \quad ::= \quad E_\Delta[x/t] \mid E_{\Delta\uplus\{x\}} t$$

A special case of evaluation context is given by **substitution contexts**, noted L_Δ and defined by:

$$L_\emptyset \quad ::= \quad (\!|\cdot|\!) \qquad\qquad L_{\Delta\uplus\{x\}} \quad ::= \quad L_\Delta[x/t]$$

Definition 1. Linear weak head reduction $\multimap$ is defined as the union of $\multimap_{\mathtt{dB}}$ and $\multimap_{\mathtt{ls}}$, which are given by the closure by evaluation contexts (*i.e.* $\multimap_{\mathtt{dB}} := E_\Delta[\mapsto_{\mathtt{dB}}]$ and $\multimap_{\mathtt{ls}} := E_\Delta[\mapsto_{\mathtt{ls}}]$) of the rules $\mapsto_{\mathtt{dB}}$ and $\mapsto_{\mathtt{ls}}$ defined as:

$$L_\Delta(\!|\lambda x.t|\!)\,s \quad \mapsto_{\mathtt{dB}} \quad L_\Delta(\!|t[x/s]|\!) \qquad\qquad E_\Delta(\!|x|\!)[x/s] \quad \mapsto_{\mathtt{ls}} \quad E_\Delta(\!|s|\!)[x/s] \qquad \text{with } x \notin \Delta$$

The rule $\mapsto_{\mathtt{ls}}$ implicitly assumes the side-condition $\mathtt{fv}(s) \cap \Delta = \emptyset$. The assumption is implicit because it can always be guaranteed by α-conversion: if $u = E_\Delta(\!|x|\!)[x/s]$ and $\mathtt{fv}(s) \cap \Delta \neq \emptyset$ then there exist a set of variables Σ and an evaluation context F_Σ s.t. $u =_\alpha F_\Sigma(\!|x|\!)[x/s]$ and $\mathtt{fv}(s) \cap \Sigma = \emptyset$.

These rule are *at a distance*, because their definition involves contexts, which is how locality on proof nets is reflected on terms. In Milner's calculus the first rule does not use $L_\Delta(\!|\cdot|\!)$. This is not a detail: the results in this paper would not hold with respect to Milner's original presentation.

It is natural to wonder in which sense the linear substitution calculus is *linear*. In contrast to other linear calculi, variables may have multiple occurrences, and arguments are not forced to be used only once. A first superficial linear aspect of the calculus is that variable occurrences are substituted one at the time. A second much deeper aspect is that its head strategy—characterized by a factorization theorem in the same way as head reduction in λ-calculus [3]—is *linear head reduction*, whose main feature is the *subterm property* (namely: any subterm u which is duplicated at any point of a reduction $t \multimap^k s$ is a subterm of t, whose size then does not depend on k) which implies that the implementation cost of *every* step is linear (in the size of t, the parameter for complexity). This is a fundamental property, not enjoyed by any strategy in λ-calculus (for which the cost of one step is not even polynomial in the size of t), and which opens the way to the study of computational complexity [9]. Here we deal with linear *weak* head reduction, which forbids reduction under abstractions. The restriction does not affect the subterm property.

2 The π-calculus

The fragment of the π-calculus we use here is essentially the asynchronous calculus in [21] with both unary and binary inputs and outputs, morally corresponding to the exponential and the multiplicative connectives of linear logic (in the typed case of [21]) and without sums (which correspond to the additives). The only change is that we do not use their forwarding processes[2]. The grammar is:

$$P,Q,R \quad ::= \quad 0 \;\big|\; \overline{x}\langle y\rangle \;\big|\; \overline{x}\langle y,z\rangle \;\big|\; \nu x P \;\big|\; x(y,z).P \;\big|\; !x(y).P \;\big|\; P \mid Q$$

We need a notion of context also for processes. A **non-blocking context** is given by:

$$N_\emptyset \quad ::= \quad (\!|\cdot|\!) \;\big|\; N_\emptyset \mid Q \;\big|\; P \mid N_\emptyset \qquad\qquad N_{\Delta\uplus x} \quad ::= \quad \nu x N_\Delta \;\big|\; N_\emptyset(\!|N_{\Delta\uplus x}|\!)$$

The language is considered modulo **structural congruence**, *i.e.* the minimum equivalence relation generated by the following rules and closed by non-blocking contexts:

$$\frac{}{P \mid 0 \equiv P} \qquad \frac{}{P \mid (Q \mid R) \equiv (P \mid Q) \mid R} \qquad \frac{}{P \mid Q \equiv Q \mid P}$$

$$\frac{}{\nu x 0 \equiv 0} \qquad \frac{x \notin \mathtt{fn}(P)}{P \mid \nu x Q \equiv \nu x.(P \mid Q)} \qquad \frac{}{\nu x \nu y P \equiv \nu y \nu x P}$$

In order to prove the simulation theorems we will use the following three properties of $\equiv$, proved by easy inductions on N_Δ, P, and N_Δ, respectively (the set of free variables of a context is defined as for processes but using $\mathtt{fn}(\!(\cdot)\!) = \emptyset$).

Lemma 2. *Let Δ be a set of variables, N_Δ a non-blocking context, P a process s.t. $\mathtt{fn}(P) \cap \Delta = \emptyset$, and $x,y \notin \Delta$. Then:*

[2] Forwarding processes correspond to axioms in linear logic. In terms of proof nets, avoiding forwarding processes correspond to use an interaction nets presentation, *i.e.* to work modulo cut-elimination on axioms.

1. $N_\Delta(\!|Q|\!) \mid P \equiv N_\Delta(\!|Q \mid P|\!)$.
2. *If* $x \notin \mathtt{fn}(P)$ *then* $\nu x P \equiv P$.
3. *If* $x \notin \mathtt{fn}(N_\Delta)$ *then* $\nu x N_\Delta(\!|P|\!) \equiv N_\Delta(\!|\nu x P|\!)$.

The rewriting rules are the following:

$$\overline{x}\langle y,z\rangle \mid x(y',z').Q \quad \to_\otimes \quad Q\{y'/y\}\{z'/z\} \qquad\qquad \overline{x}\langle y\rangle \mid !x(z).Q \quad \to_! \quad Q\{z/y\} \mid !x(z).Q$$

as usual they are both closed by non-blocking contexts and considered modulo $\equiv$. The second rule puts together replication and unary communication as in [39, 21].

π-calculus, at a distance. In order to simplify the proof of the bisimulation, we are going to use an alternative but equivalent definition of reduction in the π-calculus. Essentially, we have to reformulate the π-calculus *at a distance*. The use of the structural equivalence in the definition of the rewriting relation of the π-calculus induces some annoying complications when one tries to reflect process reductions on terms. We are going to reformulate the reduction rules via non-blocking contexts, and get rid of structural equivalence.

The rewriting rules $\Rightarrow_\otimes$ and $\Rightarrow_!$ are given by the closure by non-blocking contexts (but are not closed by structural congruence) of the following relations: if $x \notin \Delta \cup \Gamma$ then

$$\begin{array}{lcl} N_\Delta(\!|\overline{x}\langle y,z\rangle|\!) \mid M_\Gamma(\!|x(y',z').P|\!) & \mapsto_\otimes & M_\Gamma(\!|N_\Delta(\!|P\{y'/y\}\{z'/z\}|\!)|\!) \\ N_\Delta(\!|\overline{x}\langle y\rangle|\!) \mid M_\Gamma(\!|!x(z).P|\!) & \mapsto_! & M_\Gamma(\!|N_\Delta(\!|P\{z/y\} \mid !x(z).P|\!)|\!) \end{array}$$

Actually, one should ask three futher conditions on variables: 1) $\Delta \cap \Gamma = \emptyset$; 2) $\Delta \cap \mathtt{fv}(P) = \emptyset$; 3) $\mathtt{fv}(N_\Delta) \cap \Gamma = \emptyset$. It is easily seen, however, that these conditions can always be satisfied by choosing an α-equivalent term, as it is the case for the $\mapsto_{\mathtt{ls}}$ rule of λ_{lsub}. Essentially, these rules re-formulate as reduction rules the τ-transitions of the alternative presentation of the π-calculus as a labeled transition system, which is used to study the interaction of a process with its environment. Here, the new rules are more convenient than labeled transitions, because on λ-terms there is no analogous of the transitions whose label is not τ (and τ-transitions are defined using the non-τ transitions). This reformulation is justified by the following lemma, whose proof is along the one of the harmony lemma in [38] (p. 51).

Lemma 3.

1. ***$\equiv$ is a strong bisimulation with respect to $\Rightarrow$:*** *$P \equiv \Rightarrow_\otimes Q$ iff $P \Rightarrow_\otimes \equiv Q$, and $P \equiv \Rightarrow_! Q$ iff $P \Rightarrow_! \equiv Q$.*
2. ***Harmony of $\Rightarrow$ and $\to_\pi$:*** *$P \to_\otimes Q$ iff $P \Rightarrow_\otimes \equiv Q$, and $P \to_! Q$ iff $P \Rightarrow_! \equiv Q$.*

Curiously, the first formulation of the π-calculus was as a labeled transition system; the notions of reduction and structural congruence were introduced by Milner only later on, to study the relationship with the λ-calculus [33]. Our formulation at a distance of the π-calculus—motivated in exactly the same way—is a contribution of independent interest, probably the main one from the π-calculus point of view. It also shows that distance rules are a general syntactic principle whose relevance extends beyond explicit substitutions.

3 The call-by-name encoding

As for the ordinary λ-calculus, the translation from λ_{lsub} to the π-calculus is parametrized by a special channel name a. Actually, we assume that these **special channel names** are taken from a set A which is disjoint from the set of variable names, and whose elements are denoted $a, b, c, d, \ldots$.

The translation is given by (on screen it is in red):

$$\begin{array}{rcl@{\qquad}rcll} [\![x]\!]_a & := & \overline{x}\langle a\rangle & [\![ts]\!]_a & := & \nu b\nu x([\![t]\!]_b \mid \overline{b}\langle x,a\rangle \mid !x(c).[\![s]\!]_c) & x \text{ is fresh} \\ [\![\lambda x.t]\!]_a & := & a(x,b).[\![t]\!]_b & [\![t[x/s]]\!]_a & := & \nu x([\![t]\!]_a \mid !x(b).[\![s]\!]_b) & \end{array}$$

Modulo minor details, this is the original call-by-name encoding given by Milner. With respect to the relation with linear logic developed in [21], special names correspond exactly to multiplicative formulas, while variable names correspond to exponential formulas.

An easy induction on the translation shows:

Lemma 4. *Let t be a term. Then $\mathtt{fn}([\![t]\!]_a) = \mathtt{fv}(t) \uplus \{a\}$.*

To relate terms and processes we need to prove a property of the translation, concerning its action on contexts: it maps evaluation contexts to non-guarding contexts of a special form.

Lemma 5 (Relating E and N via $[\![\cdot]\!]_a$)**.** *Let Δ be a set of variable names, E_Δ an evaluation context, and a a special name. There exist a set of names Γ (possibly containing both variables and special names), a non-blocking context $N_{\Delta\uplus\Gamma}$ and a special name b s.t. $[\![E_\Delta(\!|t|\!)]\!]_a = N_{\Delta\uplus\Gamma}(\!|[\![t]\!]_b|\!)$ and $\Gamma \cap \mathtt{fv}(t) = \emptyset$ for every term t. Moreover, if E_Δ is a substitution context L_Δ then $a = b$, $\Gamma = \emptyset$, and N_Δ does not depend on a.*

Proof. By induction on E_Δ. The base case is given by the empty context $E_\emptyset = (\!|\cdot|\!)$, and it is trivial, just take $\Gamma := \emptyset$, $N_\emptyset := (\!|\cdot|\!)$, and $b = a$. The inductive cases:

- *Left of an application*, $E_\Delta = F_\Delta s$: if x is a fresh variable name:

$$\begin{array}{rcl@{\qquad}rcl} [\![E_\Delta(\!|t|\!)]\!]_a & = & [\![F_\Delta(\!|t|\!)s]\!]_a & = & \nu d\nu x([\![F_\Delta(\!|t|\!)]\!]_d \mid \overline{d}\langle x,a\rangle \mid !x(c).[\![s]\!]_c) & & \\ & & & =_{i.h.} & \nu d\nu x(M_{\Delta\uplus\Sigma}(\!|[\![t]\!]_b|\!) \mid \overline{d}\langle x,a\rangle \mid !x(c).[\![s]\!]_c) & = & N_{\Delta\uplus\Sigma\uplus\{d,x\}}(\!|[\![t]\!]_b|\!) \end{array}$$

  By *i.h.* we get that $\Sigma \cap \mathtt{fv}(t) = \emptyset$. By definition of the translation x is fresh, so $x \notin \mathtt{fv}(t)$. We then conclude by taking $\Gamma := \Sigma \uplus \{d,x\}$.

- *Left of a substitution*, $E_{\Delta\uplus\{x\}} = F_\Delta[x/s]$:

$$\begin{array}{rcl@{\qquad}rcl} [\![E_{\Delta\uplus\{x\}}(\!|t|\!)]\!]_a & = & [\![F_\Delta(\!|t|\!)[x/s]]\!]_a & = & \nu x([\![F_\Delta(\!|t|\!)]\!]_a \mid !x(c).[\![s]\!]_c) & & \\ & & & =_{i.h.} & \nu x(M_{\Delta\uplus\Gamma}(\!|[\![t]\!]_b|\!) \mid !x(c).[\![s]\!]_c) & = & N_{\Delta\uplus\{x\}\uplus\Gamma}(\!|[\![t]\!]_b|\!) \end{array}$$

  and the *i.h.* also gives $\Gamma \cap \mathtt{fv}(t) = \emptyset$.

  Now suppose that $E_{\Delta\uplus\{x\}}$ (and thus F_Δ) is a substitution context L_Δ. Then by *i.h.* we get M_Δ not depending on a s.t.:

$$\begin{array}{rcl@{\qquad}rcl} [\![E_{\Delta\uplus\{x\}}(\!|t|\!)]\!]_a & = & [\![F_\Delta(\!|t|\!)[x/s]]\!]_a & = & \nu x([\![F_\Delta(\!|t|\!)]\!]_a \mid !x(c).[\![s]\!]_c) & & \\ & & & =_{i.h.} & \nu x(M_\Delta(\!|[\![t]\!]_a|\!) \mid !x(c).[\![s]\!]_c) & = & N_{\Delta\uplus\{x\}}(\!|[\![t]\!]_a|\!) \end{array}$$

  Where clearly $N_{\Delta\uplus\{x\}}$ does not depend on a. □

We can now proceed with the simulation.

Theorem 6 ($\to_\pi$ strongly simulates $\multimap$ via $[\![\cdot]\!]_a$)**.**

1. *$t \multimap_{\mathtt{dB}} s$ implies $[\![t]\!]_a \Rightarrow_\otimes \equiv [\![s]\!]_a$.*
2. *$t \multimap_{\mathtt{ls}} s$ implies $[\![t]\!]_a \Rightarrow_! \equiv [\![s]\!]_a$.*

Proof. 1. Two cases:

- *Root rewriting step*: first without $L_\Delta(\!|\cdot|\!)$: $(\lambda x.M)N \mapsto_{\mathtt{dB}} M[x/N]$

$$\begin{array}{rcllcl} [\![(\lambda x.t)s]\!]_a & = & \nu b\nu y([\![\lambda x.t]\!]_b \mid \overline{b}\langle y,a\rangle \mid !y(c).[\![s]\!]_c) & = & \nu b\nu y(b(x,e).[\![t]\!]_e \mid \overline{b}\langle y,a\rangle \mid !y(c).[\![s]\!]_c) \\ & \Rightarrow_{\otimes} & \nu b\nu y([\![t]\!]_a\{x/y\} \mid !y(c).[\![s]\!]_c) & =_{\alpha} & \nu b\nu x([\![t]\!]_a \mid !x(c).[\![s]\!]_c) \\ & = & \nu b[\![t[x/s]]\!]_a & \equiv & [\![t[x/s]]\!]_a \end{array}$$

The $=_\alpha$-step is justified by the fact that y is introduced fresh in the first line. The $\equiv$ step is justified by Lemma 4, for which the only free special name occurring in $[\![t]\!]_a$ is a, and by Lemma 2.2, which allow us to remove the useless νb.

Now, if $L_\Delta\langle\!\langle\lambda x.t\rangle\!\rangle s \mapsto_{\mathtt{dB}} L_\Delta\langle\!\langle t[x/s]\rangle\!\rangle$ we get (some explanations follow):

$$\begin{array}{rcl} [\![L_\Delta\langle\!\langle\lambda x.t\rangle\!\rangle s]\!]_a & = & \nu b\nu y([\![L_\Delta\langle\!\langle\lambda x.t\rangle\!\rangle]\!]_b \mid \overline{b}\langle y,a\rangle \mid !y(c).[\![s]\!]_c) \\ & =_{Lem.5} & \nu b\nu y(N_\Delta\langle\!\langle[\![\lambda x.t]\!]_b\rangle\!\rangle \mid \overline{b}\langle y,a\rangle \mid !y(c).[\![s]\!]_c) \\ & = & \nu b\nu y(N_\Delta\langle\!\langle b(x,e).[\![t]\!]_e\rangle\!\rangle \mid \overline{b}\langle y,a\rangle \mid !y(c).[\![s]\!]_c) \\ & \Rightarrow_{\otimes} & \nu b\nu y(N_\Delta\langle\!\langle[\![t]\!]_a\{x/y\}\{e/a\}\rangle\!\rangle \mid !y(c).[\![s]\!]_c) \\ & =_{\alpha} & \nu b\nu x(N_\Delta\langle\!\langle[\![t]\!]_a\rangle\!\rangle \mid !x(c).[\![s]\!]_c) \\ & \equiv_{Lem.2.1\&Lem.2.3} & \nu bN_\Delta\langle\!\langle\nu x([\![t]\!]_a \mid !x(c).[\![s]\!]_c)\rangle\!\rangle \\ & = & \nu bN_\Delta\langle\!\langle[\![t[x/s]]\!]_a\rangle\!\rangle \\ & =_{Lem.5} & \nu b[\![L_\Delta[t[x/s]]]\!]_a \\ & \equiv_{Lem.4\&Lem.2.2} & [\![L_\Delta[t[x/s]]]\!]_a \end{array}$$

The $=_\alpha$-step and the last step are justified as before. In the first application of $\equiv$ we can apply Lemma 2.1 because by hypothesis $x \notin \Delta$ and $\mathtt{fv}(s)\cap\Delta = \emptyset$, and Lemma 2.3 because $x \notin \mathtt{fn}(N_\Delta)$. The two applications of Lemma 5 are with respect to different special names a and b, but this is sound: the *moreover* part of Lemma 5 guarantees that in the case of a substitution context L_Δ the corresponding context N_Δ does not depend on the name.

- *Inductive step*: $E_\Delta\langle\!\langle t\rangle\!\rangle \to_{\mathtt{dB}} E_\Delta\langle\!\langle s\rangle\!\rangle$ because $t \mapsto_{\mathtt{dB}} s$. Let us recall that by definitions reductions in the π-calculus are closed by non-blocking contexts. Then:

$$[\![E_\Delta\langle\!\langle t\rangle\!\rangle]\!]_a \quad =_{Lem.5} \quad N_{\Delta\uplus\Gamma}\langle\!\langle[\![t]\!]_b\rangle\!\rangle \quad \Rightarrow_{\otimes} \quad N_{\Delta\uplus\Gamma}\langle\!\langle[\![s]\!]_b\rangle\!\rangle \quad =_{Lem.5} \quad [\![E_\Delta\langle\!\langle s\rangle\!\rangle]\!]_a$$

2. For $\to_{\mathtt{ls}}$ the inductive case is as for $\to_{\mathtt{dB}}$. The base case is $E_\Delta\langle\!\langle x\rangle\!\rangle[x/s] \multimap_{\mathtt{ls}} E_\Delta\langle\!\langle s\rangle\!\rangle[x/s]$ with $x \notin \Delta$:

$$\begin{array}{rcllcl} [\![E_\Delta\langle\!\langle x\rangle\!\rangle[x/s]]\!]_a & = & \nu x([\![E_\Delta\langle\!\langle x\rangle\!\rangle]\!]_a \mid !x(b).[\![s]\!]_b) & =_{Lem.5} & \nu x(N_{\Delta\uplus\Gamma}\langle\!\langle[\![x]\!]_c\rangle\!\rangle \mid !x(b).[\![s]\!]_b) \\ & = & \nu x(N_{\Delta\uplus\Gamma}\langle\!\langle\overline{x}\langle c\rangle\rangle\!\rangle \mid !x(b).[\![s]\!]_b) & \Rightarrow_! & \nu xN_{\Delta\uplus\Gamma}\langle\!\langle[\![s]\!]_c \mid !x(b).[\![s]\!]_b\rangle\!\rangle \\ & \equiv_{Lem.2.1} & \nu x(N_{\Delta\uplus\Gamma}\langle\!\langle[\![s]\!]_c\rangle\!\rangle \mid !x(b).[\![s]\!]_b) & =_{Lem.5} & \nu x([\![E_\Delta\langle\!\langle s\rangle\!\rangle]\!]_a \mid !x(b).[\![s]\!]_b) \\ & = & [\![E_\Delta\langle\!\langle s\rangle\!\rangle[x/s]]\!]_a & & \end{array}$$

where the $\equiv$-step is justified by the fact that by hypothesis and by Lemma 5 ($x \notin \Gamma$) we get that $(\mathtt{fv}(s)\uplus\{x,b\})\cap(\Delta\uplus\Gamma) = \emptyset$, and so we can apply Lemma 2.1. □

The converse relation. To simulate process reductions on λ-terms we need a lemma, which is a converse to Lemma 5.

Lemma 7. *Let Δ and Γ be a set of variable names and a set of special names, respectively.*

1. *If $[\![t]\!]_a = N_{\Delta\uplus\Gamma}\langle\!\langle a(y,b).P\rangle\!\rangle$ with $a \notin \Gamma$ then $\Gamma = \emptyset$ and exist s and L_Δ s.t. $P = [\![s]\!]_b$ and $t = L_\Delta\langle\!\langle\lambda y.s\rangle\!\rangle$.*
2. *If $[\![t]\!]_a = N_{\Delta\uplus\Gamma}\langle\!\langle\overline{x}\langle c\rangle\rangle\!\rangle$ with $x \notin \Delta$ then exist $\Sigma \subseteq \Delta$ and E_Σ s.t. $t = E_\Sigma\langle\!\langle x\rangle\!\rangle$ (and $x \notin \Sigma$).*

Proof. Both points are by induction on t:

- *Variable*:

1. The hypothesis is false and there is nothing to prove.
2. By definition of $[\![\cdot]\!]_a$, taking the empty context (and $\Delta = \emptyset$).

- *Abstraction*:
  1. By definition of $[\![\cdot]\!]_a$, taking the empty context (and $\Delta = \emptyset$).
  2. The hypothesis is false and there is nothing to prove.
- *Application*: if $t = ur$ then $[\![ur]\!]_a = \nu b \nu z([\![u]\!]_b \mid \overline{b}\langle z, a\rangle \mid !z(c).[\![r]\!]_c)$ with z fresh.
  1. By Lemma 4 $a \notin \mathtt{fn}([\![u]\!]_b)$, and so there is no context $N_{\Delta\uplus\Gamma}$ s. t. $[\![t]\!]_a = N_{\Delta\uplus\Gamma}(\!|a(y,b).P|\!)$, hence the hypothesis is false and there is nothing to prove.
  2. It must be that $[\![u]\!]_a = M_{\Delta'\uplus\Gamma'}(\!|\overline{x}\langle c\rangle|\!)$ with $\Delta = \Delta' \uplus \{z\}$ and $\Gamma = \Gamma' \uplus \{a\}$. Then by *i.h.* there exist $\Sigma \subseteq \Delta'$ and F_Σ s.t. $u = F_\Sigma(\!|x|\!)$. We conclude taking $E_\Sigma := F_\Sigma r$.
- *Substitution*: if $t = u[z/r]$ then $[\![u[z/r]]\!]_a = \nu z([\![u]\!]_a \mid !z(b).[\![r]\!]_b)$.
  1. If $[\![t]\!]_a = N_{\Delta\uplus\Gamma}(\!|a(y,b).P|\!)$ then it must be that exists $M_{\Delta'\uplus\Gamma}(\!|\cdot|\!)$ with $\Delta = \Delta' \uplus \{z\}$ s.t. $[\![u]\!]_b = M_{\Delta'\uplus\Gamma}(\!|a(y,b).P|\!)$ and $N_{\Delta\uplus\Gamma} = \nu z(M_{\Delta'\uplus\Gamma} \mid !z(b).[\![r]\!]_b)$. By *i.h.* we get $\Gamma = \emptyset$, $u = L'_{\Delta'}(\!|\lambda y.s|\!)$, and $P = [\![s]\!]_b$. We conclude taking $L_\Delta := L'_{\Delta'}[z/r]$.
  2. It must be that $[\![u]\!]_a = M_{\Delta'\uplus\Gamma'}(\!|\overline{x}\langle c\rangle|\!)$ with $\Delta = \Delta' \uplus \{z\}$ and $\Gamma = \Gamma' \uplus \{a\}$. Then by *i.h.* there exist $\Sigma' \subseteq \Delta'$ and $F_{\Sigma'}$ s.t. $u = F_{\Sigma'}(\!|x|\!)$. We conclude taking $\Sigma := \Sigma' \uplus \{z\}$ and $E_\Sigma := F_{\Sigma'}[z/r]$. □

Now, we can prove that any process reduction from $[\![t]\!]_a$ can be simulated by t.

Theorem 8 ($\multimap$ strongly simulates $\Rightarrow$ via $[\![\cdot]\!]_a$)**.**

1. *If* $[\![t]\!]_a \Rightarrow_\otimes Q$ *then exists* s *s.t.* $t \multimap_{\mathtt{dB}} s$ *and* $[\![s]\!]_a \equiv Q$.
2. *If* $[\![t]\!]_a \Rightarrow_! Q$ *then exists* s *s.t.* $t \multimap_{\mathtt{ls}} s$ *and* $[\![s]\!]_a \equiv Q$.

Proof. Both points are by induction on t. Cases:

- *Values*: if $t = x$ or $t = \lambda x.u$ then $[\![t]\!]_a$ cannot reduce.
- *Application*: if $t = ur$ then $[\![t]\!]_a = \nu b \nu x([\![u]\!]_b \mid \overline{b}\langle x, a\rangle \mid !x(c).[\![r]\!]_c)$ with x fresh. Then:
  1. *Multiplicative reduction.* Cases of $[\![t]\!]_a \Rightarrow_\otimes Q$:
     - *Root*: $[\![u]\!]_b = N_{\Delta\uplus\Gamma}(\!|b(y,d).P|\!)$ with $b \notin (\Delta \uplus \Gamma)$ and the process reduction is a $\Rightarrow_\otimes$ interaction with $\overline{b}\langle x, a\rangle$ on b. By Lemma 7.1 we get that $\Gamma = \emptyset$, $u = L_\Delta(\!|\lambda y.u'|\!)$, and $P = [\![u']\!]_d$. So $t = L_\Delta(\!|\lambda y.u'|\!)r$ and thus it has a $\multimap_{\mathtt{dB}}$-redex on y, which maps to the $\Rightarrow_\otimes$ communication on b exactly as in the proof of Theorem 6.1.
     - *Inductive*: because of $[\![u]\!]_b \Rightarrow_\otimes R$. Then by *i.h.* exists u' s.t. $u \to_{\mathtt{dB}} u'$ and $[\![u']\!]_b \equiv R$. We conclude by taking $s := u'r$.
  2. *Exponential reduction.* $[\![t]\!]_a \Rightarrow_! Q$ can only happen if reduction takes place in $[\![u]\!]_b$, because x is fresh by hypothesis. In such a case we conclude using the *i.h.*, as in the first sub-case of the previous point.
- *Substitution*: if $t = u[x/r]$ then $[\![t]\!]_a = \nu x([\![u]\!]_a \mid !x(b).[\![r]\!]_b)$. We have:
  1. *Multiplicative reduction.* $[\![t]\!]_a \Rightarrow_\otimes Q$ can only happen if reduction takes place in $[\![u]\!]_a$, and we conclude using the *i.h.*.
  2. *Exponential reduction.* If $[\![t]\!]_a \Rightarrow_! Q$ because reduction takes place in $[\![u]\!]_a$ we use the *i.h.*. Otherwise, $[\![u]\!]_a = N_{\Delta\uplus\Gamma}(\!|\overline{x}\langle c\rangle|\!)$ with $x \notin \Delta \uplus \Gamma$ and the process reduction is a $\Rightarrow_!$ interaction with $!x(b).[\![r]\!]_b$ on x. By Lemma 7.2 there exist Σ and E_Σ s.t. $u = E_\Sigma(\!|x|\!)$. So $t = E_\Sigma(\!|x|\!)[x/r]$ has a $\multimap_{\mathtt{ls}}$ redex on x, which maps to the $\Rightarrow_!$ communication on x exactly as in the proof of Theorem 6.2. □

According to the two theorems of this section, the relationship between the call-by-name strategy on the ordinary λ-calculus and the evaluation in the π-calculus is the same as the relationship between the call-by-name strategy and linear weak head reduction. In the strong case (*i.e.* when (head) reduction can go under lambdas), it is known that the latter can be at most quadratically longer than the former [9]. The analysis in [9] does not depend on being weak or strong. It follows that the same upper bound holds between the call-by-name strategy and its evaluation in the π-calculus.

Last, it is easy to see that linear weak head reduction is *deterministic*: every term has at most one $\multimap$ redex, since every redex writes as $E_\Delta\langle\!|v|\!\rangle$ (where v is a value, *i.e.* a variable or an abstraction) and such a decomposition is unique. This property accounts for what Milner calls *determinacy* of $[\![t]\!]_a$ in [33].

4 The call-by-value encoding

We now show that the same exact relationship can be obtained with respect to call-by-value (CBV). The CBV calculus in use here is not Plotkin's calculus $\lambda_{\beta v}$. In [10] the author and Paolini introduced the *value substitution calculus* λ_{vsub}, which is a CBV calculus with explicit substitutions containing $\lambda_{\beta v}$ as a sub-calculus and behaving better than $\lambda_{\beta v}$ with respect to the semantical notion of *solvability*. In [4, 5] we showed that λ_{vsub} has a sub-calculus, the *value substitution kernel* λ_{vker}, which has two key properties:

1. *Observational equivalence* [4]: there is a translation $\cdot^\circ : \lambda_{vsub} \to \lambda_{vker}$ s.t. t and t° are equivalent with respect to observing any termination property.
2. *Language for proof nets* [5]: λ_{vker} is an algebraic reformulation of the proof nets corresponding to the CBV translation of λ-calculus into linear logic. Namely, there is a translation $\underline{\cdot} : \lambda_{vker} \to PN$ which is a strong bisimulation.

Here, we are going to show a further property: there are a CBV analogous $\multimap_{\mathtt{v}}$ of linear weak head reduction $\multimap$ and a translation $\{\!|\cdot|\!\}^x$ from λ_{vker} to the π-calculus which is a strong bisimulation with respect to $\multimap_{\mathtt{v}}$ and $\Rightarrow$. Let us point out that in the untyped case there is a strong mismatch between Plotkin's calculus $\lambda_{\beta v}$ and the evaluation in proof nets (see [4]), thus the results of this section do not hold with respect to $\lambda_{\beta v}$ (nor with any of its refinements with explicit substitutions where β-redexes are constrained to fire on values).

The **value substitution kernel** λ_{vker} is given by the following grammar:

$$t,s,u,r \quad ::= \quad v \mid vt \mid t[x/s] \qquad\qquad v \quad ::= \quad x \mid \lambda x.t$$

Please note that the left sub-term of an application can only be a value (see [4, 5] for more details). Substitution contexts L_Δ are defined as before. Instead, the language of **evaluation contexts** changes:

$$E_\emptyset \quad ::= \quad \langle\!|\cdot|\!\rangle \mid vE_\emptyset \mid t[x/E_\emptyset] \qquad\qquad E_{\Delta\uplus\{x\}} \quad ::= \quad E_\Delta[x/t] \mid vE_{\Delta\uplus\{x\}} \mid t[y/E_{\Delta\uplus\{x\}}]$$

Next, we define **applicative contexts** as $A_\Delta\langle\!|\cdot|\!\rangle ::= E_\Delta\langle\!|\langle\!|\cdot|\!\rangle t|\!\rangle$. As for CBN, we do not define the full calculus, but only the evaluation strategy. **Linear weak applicative reduction**, noted $\multimap_{\mathtt{v}}$, is given by the rewriting rules $\multimap_{\mathtt{vdB}}$ and $\multimap_{\mathtt{vls}}$ defined as the closure by evaluation contexts of the following rules:

$$(\lambda x.t)s \quad \mapsto_{\mathtt{dB}} \quad t[x/s] \qquad\qquad A_\Delta\langle\!|x|\!\rangle[x/L_\Sigma\langle\!|v|\!\rangle] \quad \mapsto_{\mathtt{lsv}} \quad L_\Sigma\langle\!|A_\Delta\langle\!|v|\!\rangle[x/v]|\!\rangle \qquad x \notin \Delta$$

Note that the argument of a β-redex is not required to be a value, while the substitution rule can fire only in presence of a value (in a substitution context). As it was the case for the call-by-name calculus and for the π-calculus, one should also ask that $\mathtt{fv}(v) \cap \Delta = \emptyset$, $\mathtt{fv}(A_\Delta\langle\!|x|\!\rangle) \cap \Sigma = \emptyset$, and $\Delta \cap \Sigma = \emptyset$, but these side-conditions can always be satisfied by taking an α-equivalent term, and so in the following they will be taken for granted. Note that $x[x/y] \not\mapsto_{\mathtt{lsv}} y$ but $(xz)[x/y] \mapsto_{\mathtt{lsv}} yz$, because substitution has to take place

in an applicative context. This applicative restriction is a sort of converse to the head restriction used in the case of call-by-name evaluation. In terms of proof nets both these restrictions correspond to forbid reduction of cuts involving links in some !-boxes (with respect to the respective encodings of CBV and CBN), while the *weak* requirement correspond to the analogous constraint with respect to the $^{2}\!\Im$-boxes mentioned in the introduction. The applicative restriction is somehow a surprise, which is justified by the fact that it matches what happens in the π-calculus. It is a quite reasonable restriction: there is no point in substituting a value if it cannot be used in some application.

Linear weak applicative reduction enjoys a property which is the CBV analogous of the subterm porperty (deifned at the end of Section 1). Let us call a *v-subterm* a subterm which is a value.

Lemma 9 (v-subterm property)**.** *If $t \multimap_{\mathtt{v}}^{k} s$ and v is a v-subterm of s then v is a v-subterm of t.*

Proof. By induction on k. For $k = 0$ it is trivial, for $k > 0$ consider the term u s.t. $u \multimap_{\mathtt{v}} s$. The $\multimap_{\mathtt{vdB}}$ rule does not create new values. The $\multimap_{\mathtt{vls}}$ rule duplicates a v-subterm of u, which by *i.h.* is a v-subterm of t, and it does not substitute into v-subterms. So, any v-subterm of s is a v-subterm of t. □

Differently from linear weak head reduction, linear weak applicative reduction is a *non-deterministic* stretegy: just consider $t = ((\lambda x.x)(yy))[y/z]$, which has two redexes. However, a simple induction shows that reduction is confluent: there is no need to use parallel reductions or other sophisticated techniques because no redex can duplicate/erase other redexes. In fact, it is easily seen that linear weak applicative reduction enjoys the diamond property. This fact corresponds to what Milner calls *determinacy* of the CBV encoding.

The translation. Similarly to the CBV translation of the λ-calculus to linear logic, the CBV translation to the π-calculus uses an auxiliary function. The main translation function $\{\!\{t\}\!\}^{x}$ is parametrized by a variable name $x \notin \mathtt{fv}(t)$ (and not by a special name) and the auxiliary function is noted $\{\!\{\cdot\}\!\}^{a}$, *i.e.* we use the same symbol but now the parameter is a special name a:

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}l}
\{\!\{v\}\!\}^{x} & ::= & !x(a).\{\!\{v\}\!\}^{a} & \{\!\{vs\}\!\}^{x} & ::= & \nu b \nu y(\{\!\{v\}\!\}^{b} \mid \overline{b}\langle y,x\rangle \mid \{\!\{s\}\!\}^{y}) & y \text{ is fresh} \\
\{\!\{y\}\!\}^{a} & ::= & \overline{y}\langle a\rangle & \{\!\{s[y/u]\}\!\}^{x} & ::= & \nu y(\{\!\{s\}\!\}^{x} \mid \{\!\{u\}\!\}^{y}) & \\
\{\!\{\lambda y.s\}\!\}^{a} & ::= & a(y,z).\{\!\{s\}\!\}^{z} & & & &
\end{array}$$

Note that the application case uses the auxiliary function on v. Note also the difference with the call-by-name case: applications and explicit substitutions do not use replication, which is instead associated to values, with the important exception of applied values. The *applicative* restriction on the strategy $\multimap_{\mathtt{v}}$ comes from this exception: the impossibility of interacting under replication in the π-calculus reflects on terms as the fact that one can substitute only on variables in applicative contexts, because the others are under a replication prefix. Last, this encoding is a minor variation over the CBV one in [39], which is not Milner's original CBV encoding.

Lemma 10. *Let $t \in \lambda_{vker}$. Then $\mathtt{fn}(\{\!\{t\}\!\}^{x}) = \mathtt{fv}(t) \uplus \{x\}$ and $\mathtt{fn}(\{\!\{t\}\!\}^{a}) = \mathtt{fv}(t) \uplus \{a\}$.*

Proof. By mutual induction on $\{\!\{t\}\!\}^{x}$ and $\{\!\{t\}\!\}^{a}$. □

The following lemma is the call-by-value analogous of Lemma 5.

Lemma 11 (Relating E and N via $\{\!\{\cdot\}\!\}^{x}$)**.** *Let Δ be a set of variable names, x a variable name and E_{Δ} an evaluation context. There exist a set of names Γ (possibly containing both variables and special names), a non-blocking context $N_{\Delta \uplus \Gamma}$, and a variable name z s.t. $\{\!\{E_{\Delta}\langle\!\langle t \rangle\!\rangle\}\!\}^{x} = N_{\Delta \uplus \Gamma}\langle\!\langle \{\!\{t\}\!\}^{z} \rangle\!\rangle$ and $\Gamma \cap \mathtt{fv}(t) = \emptyset$ for every term t. Moreover, if E_{Δ} is a substitution context L_{Δ} then $x = z$, $\Gamma = \emptyset$, and N_{Δ} does not depend on x.*

Proof. By induction on E_Δ. The base case is given by the empty context $E_\emptyset = (\!|\cdot|\!)$, and it is trivial, just take $\Gamma := \emptyset$, $N_\emptyset := (\!|\cdot|\!)$, and $z := x$. The inductive cases:

- *Right of an application*, $E_\Delta = vF_\Delta$:
$$\begin{array}{rcll} \{\!\{E_\Delta(\!|t|\!)\}\!\}^x & = & \{\!\{vF_\Delta(\!|t|\!)\}\!\}^x & = \quad vbvy(\{\!\{v\}\!\}^b \mid \overline{b}\langle y,x\rangle \mid \{\!\{F_\Delta(\!|t|\!)\}\!\}^y) \\ & & & =_{i.h.} \quad vbvy(\{\!\{v\}\!\}^b \mid \overline{b}\langle y,x\rangle \mid M_{\Delta\uplus\Sigma}(\!|\{\!\{t\}\!\}^z|\!)) \quad = \quad N_{\Delta\uplus\Sigma\uplus\{y,b\}}(\!|\{\!\{t\}\!\}^z|\!) \end{array}$$
The *i.h.* also gives $\Sigma \cap \mathtt{fv}(t) = \emptyset$. Since $b,y \notin \mathtt{fv}(t)$ it follows that $\Gamma := \Sigma \uplus \{y,b\}$ satisfies $\Gamma \cap \mathtt{fv}(t) = \emptyset$.
- *Right of a substitution*, $E_\Delta = s[y/F_\Delta]$:
$$\begin{array}{rcll} \{\!\{E_\Delta(\!|t|\!)\}\!\}^x & = & \{\!\{s[y/F_\Delta(\!|t|\!)]\}\!\}^x & = \quad vy(\{\!\{s\}\!\}^x \mid \{\!\{F_\Delta(\!|t|\!)\}\!\}^y) \\ & & & =_{i.h.} \quad vy(\{\!\{s\}\!\}^x \mid M_{\Delta\uplus\Sigma}(\!|\{\!\{t\}\!\}^z|\!)) \quad = \quad N_{\Delta\uplus\Sigma\uplus\{y\}}(\!|\{\!\{t\}\!\}^z|\!) \end{array}$$
The *i.h.* also gives $\Sigma \cap \mathtt{fv}(t) = \emptyset$. Since $y \notin \mathtt{fv}(t)$ it follows that $\Gamma := \Sigma \uplus \{y\}$ satisfies $\Gamma \cap \mathtt{fv}(t) = \emptyset$.
- *Left of a substitution*, $E_{\Delta\uplus\{z\}} = F_\Delta[y/u]$. Then:
$$\begin{array}{rcll} \{\!\{E_{\Delta\uplus\{y\}}(\!|t|\!)\}\!\}^x & = & \{\!\{F_\Delta(\!|t|\!)[y/u]\}\!\}^x & = \quad vy(\{\!\{F_\Delta(\!|t|\!)\}\!\}^x \mid \{\!\{u\}\!\}^y) \\ & & & =_{i.h.} \quad vy(M_{\Delta\uplus\Gamma}(\!|\{\!\{t\}\!\}^z|\!) \mid \{\!\{u\}\!\}^y) \quad = \quad N_{\Delta\uplus\{y\}\uplus\Gamma}(\!|\{\!\{t\}\!\}^z|\!) \end{array}$$
The *i.h.* also gives $\Gamma \cap \mathtt{fv}(t) = \emptyset$. Now, suppose that $E_{\Delta\uplus\{y\}}$ (and thus F_Δ) is a substitution context L_Δ. Then by *i.h.* we get M_Δ not depending on x s.t.:
$$\begin{array}{rcll} \{\!\{E_{\Delta\uplus\{y\}}(\!|t|\!)\}\!\}^x & = & \{\!\{F_\Delta(\!|t|\!)[y/u]\}\!\}^x & = \quad vy(\{\!\{F_\Delta(\!|t|\!)\}\!\}^x \mid \{\!\{u\}\!\}^y) \\ & & & =_{i.h.} \quad vy(M_\Delta(\!|\{\!\{t\}\!\}^x|\!) \mid \{\!\{u\}\!\}^y) \quad = \quad N_{\Delta\uplus\{y\}}(\!|\{\!\{t\}\!\}^x|\!) \end{array}$$
where clearly $N_{\Delta\uplus\{y\}}$ does not depend on x. □

Theorem 12 ($\to_\pi$ strongly simulates $\multimap_{\mathtt{v}}$)**.**

1. *$t \multimap_{\mathtt{vdB}} s$ implies $\{\!\{t\}\!\}^x \Rightarrow_\otimes \equiv \{\!\{s\}\!\}^x$.*
2. *$t \multimap_{\mathtt{vls}} s$ implies $\{\!\{t\}\!\}^x \Rightarrow_! \equiv \{\!\{s\}\!\}^x$.*

Proof. We show the base cases, the inductive ones are as in the call-by-name case, using Lemma 11.

1. If $(\lambda y.t)s \multimap_{\mathtt{vdB}} t[y/s]$ then:
$$\begin{array}{rcllcl} \{\!\{(\lambda y.t)s\}\!\}^x & = & vbvz(\{\!\{\lambda y.t\}\!\}^b \mid \overline{b}\langle z,x\rangle \mid \{\!\{s\}\!\}^z) & = & vbvy(b(y,w).\{\!\{t\}\!\}^w \mid \overline{b}\langle z,x\rangle \mid \{\!\{s\}\!\}^z) \\ & \Rightarrow_\otimes & vbvy(\{\!\{t\}\!\}^w\{w/x\}\{y/z\} \mid \{\!\{s\}\!\}^z) & =_\alpha & vbvy(\{\!\{t\}\!\}^x \mid \{\!\{s\}\!\}^y) \\ & = & vb\{\!\{t[x/s]\}\!\}^x & \equiv_{Lem.10} & \{\!\{t[x/s]\}\!\}^x \end{array}$$
2. If $A_\Delta(\!|y|\!)[y/L_\Sigma(\!|v|\!)] \mapsto_{\mathtt{lsv}} L_\Sigma(\!|A_\Delta(\!|v|\!)[y/v]|\!)$ and $A_\Delta(\!|\cdot|\!) = E_\Delta(\!|(\!|\cdot|\!)s|\!)$ then:
$$\begin{array}{rcl} \{\!\{A_\Delta(\!|y|\!)[y/L_\Sigma(\!|v|\!)]\}\!\}^x & = & vy(\{\!\{E_\Delta(\!|ys|\!)\}\!\}^x \mid \{\!\{L_\Sigma(\!|v|\!)\}\!\}^y) \\ & =_{Lem.11} & vy(N_{\Delta\uplus\Gamma}(\!|\{\!\{ys\}\!\}^z|\!) \mid M_\Sigma(\!|\{\!\{v\}\!\}^y|\!)) \\ & = & vy(N_{\Delta\uplus\Gamma}(\!|\{\!\{ys\}\!\}^z|\!) \mid M_\Sigma(\!|!y(a).\{\!\{v\}\!\}^a|\!)) \\ & = & vy(N_{\Delta\uplus\Gamma}(\!|vbvw(\{\!\{y\}\!\}^b \mid \overline{b}\langle w,z\rangle \mid \{\!\{s\}\!\}^w)|\!) \mid M_\Sigma(\!|!y(a).\{\!\{v\}\!\}^a|\!)) \\ & = & vy(N_{\Delta\uplus\Gamma}(\!|vbvw(\overline{y}\langle b\rangle \mid \overline{b}\langle w,z\rangle \mid \{\!\{s\}\!\}^w)|\!) \mid M_\Sigma(\!|!y(a).\{\!\{v\}\!\}^a|\!)) \\ & \Rightarrow_! & vyM_\Sigma(\!|N_{\Delta\uplus\Gamma}(\!|vbvw(\{\!\{v\}\!\}^b \mid !y(a).\{\!\{v\}\!\}^a \mid \overline{b}\langle w,z\rangle \mid \{\!\{s\}\!\}^w)|\!)|\!) \\ & \equiv_{Lem.2.1} & vyM_\Sigma(\!|N_{\Delta\uplus\Gamma}(\!|vbvw(\{\!\{v\}\!\}^b \mid \overline{b}\langle w,z\rangle \mid \{\!\{s\}\!\}^w)|\!) \mid !y(a).\{\!\{v\}\!\}^a|\!) \\ & = & vyM_\Sigma(\!|N_{\Delta\uplus\Gamma}(\!|\{\!\{vs\}\!\}^z|\!) \mid !y(a).\{\!\{v\}\!\}^a|\!) \\ & = & vyM_\Sigma(\!|\{\!\{E_\Delta(\!|vs|\!)\}\!\}^x \mid !y(a).\{\!\{v\}\!\}^a|\!) \\ & \equiv_{Lem.2.3} & M_\Sigma(\!|vy(\{\!\{E_\Delta(\!|vs|\!)\}\!\}^x \mid !y(a).\{\!\{v\}\!\}^a)|\!) \\ & = & M_\Sigma(\!|\{\!\{E_\Delta(\!|vs|\!)[y/v]\}\!\}^x|\!) \\ & =_{Lem.11} & \{\!\{L_\Sigma(\!|E_\Delta(\!|vs|\!)[y/v]|\!)\}\!\}^x \\ & = & \{\!\{L_\Sigma(\!|A_\Delta(\!|v|\!)[y/v]|\!)\}\!\}^x \end{array}$$

The $\equiv$ step after the reduction is justified by the fact that b, w, and all the variables in Γ are introduced fresh and so do not belong to $\mathtt{fv}(v)$. Moreover, $\Delta \cap \mathtt{fv}(v) = \emptyset$ by hypothesis,and so we can apply Lemma 2.1. □

The converse relation. As for call-by-name, we show that linear weak applicative reduction reflects exactly evaluation in the π-calculus.

Lemma 13. *Let Δ and Γ be a set of variable names and a set of special names, respectively. Then:*

1. *If $\{\!|t|\!\}^x = N_{\Delta\uplus\Gamma}(\!|!x(a).P|\!)$ with $x \notin \Delta$ then $\Gamma = \emptyset$ and exist v and L_Δ s.t. $P = \{\!|v|\!\}^a$ and $t = L_\Delta(\!|v|\!)$.*
2. *If $\{\!|t|\!\}^x = N_{\Delta\uplus\Gamma}(\!|\overline{y}\langle a\rangle|\!)$ with $y \notin \Delta$ then exist $\Sigma \subseteq \Delta$ and A_Σ s.t. $t = A_\Sigma(\!|y|\!)$.*

Proof. Both points are by induction on t:

- *Value*: if $t = v'$ then $\{\!|t|\!\}^x = !x(a).\{\!|v'|\!\}^a$.
  1. Clearly $\Gamma = \Delta = \emptyset$, v is v', and L_Δ is the empty context.
  2. The hypothesis is false, and so there is nothing to prove.
- *Application*: if $t = v's$ then $\{\!|v's|\!\}^x = \nu b \nu z(\{\!|v'|\!\}^b \mid \overline{b}\langle z,x\rangle \mid \{\!|s|\!\}^z)$ with z and b are fresh.
  1. By definition of the translation $x \notin \mathtt{fv}(v's)$ and so by Lemma 10 $x \notin \mathtt{fn}(\{\!|v'|\!\}^b) \cup \mathtt{fn}(\{\!|s|\!\}^z)$. Consequently, there is no context $N_{\Delta\uplus\Gamma}$ s. t. $\{\!|t|\!\}^x = N_{\Delta\uplus\Gamma}(\!|!x(a).P|\!)$, so the hypothesis is false and there is nothing to prove.
  2. Two cases:
     (a) $\{\!|v'|\!\}^b = \overline{y}\langle a\rangle$ and $N_{\Delta\uplus\Gamma} = \nu b \nu z(\!|\cdot|\!) \mid \overline{b}\langle z,x\rangle \mid \{\!|s|\!\}^z)$, which imply $v' = y$, $a = b$, $\Delta = \{z\}$, and $\Gamma = \{b\}$. We conclude taking $\Sigma := \emptyset$ and $A_\emptyset := (\!|\cdot|\!)s$.
     (b) The context hole $(\!|\cdot|\!)$ is in $\{\!|s|\!\}^z$. Let $\Delta' := \Delta \setminus \{z\}$ and $\Gamma' := \Gamma \setminus \{b\}$. If $\{\!|t|\!\}^x = N_{\Delta\uplus\Gamma}(\!|\overline{z}\langle a\rangle|\!)$ then $\{\!|s|\!\}^x = M_{\Delta'\uplus\Gamma'}(\!|\overline{z}\langle a\rangle|\!)$ for some context $M_{\Delta'\uplus\Gamma'}$. The *i.h.* gives $\Sigma \subseteq \Delta'$ and an applicative context B_Σ s.t. $s = B_\Sigma(\!|y|\!)$. We conclude taking $A_\Sigma := v'B_\Sigma$.
- *Substitution*: if $t = s[z/u]$ then $\{\!|t|\!\}^x = \nu z(\{\!|s|\!\}^x \mid \{\!|u|\!\}^z)$.
  1. By definition of the translation $x \notin \mathtt{fv}(s[z/u])$ and so by Lemma 10 $x \in \mathtt{fn}(\{\!|s|\!\}^x)$ and $x \notin \mathtt{fn}(\{\!|u|\!\}^z)$. Consequently, the context hole $(\!|\cdot|\!)$ is in $\{\!|s|\!\}^x$, which then writes as $M_{\Delta'\uplus\Gamma}(\!|!x(a).P|\!)$, with $\Delta = \Delta' \uplus \{z\}$ for some context $M_{\Delta'\uplus\Gamma}$. By *i.h.* we get that there exist v and $L'_{\Delta'}$ s.t. $P = \{\!|v|\!\}^a$ and $s = L'_{\Delta'}(\!|v|\!)$. We conclude taking $L_\Delta := L'_{\Delta'}[z/u]$.
  2. Two cases:
     (a) The context hole $(\!|\cdot|\!)$ is in $\{\!|s|\!\}^x$. Let $\Delta' := \Delta \setminus \{z\}$. If $\{\!|t|\!\}^x = N_{\Delta\uplus\Gamma}(\!|\overline{z}\langle a\rangle|\!)$ then $\{\!|s|\!\}^x = M_{\Delta'\uplus\Gamma}(\!|\overline{z}\langle a\rangle|\!)$ for some context $M_{\Delta'\uplus\Gamma}$. The *i.h.* gives $\Sigma' \subseteq \Delta'$ and an applicative context $B_{\Sigma'}$ s.t. $s = B_{\Sigma'}(\!|y|\!)$. We conclude taking $\Sigma := \Sigma' \uplus \{z\}$ and $A_\Sigma := B_{\Sigma'}[z/u]$.
     (b) The context hole is in $\{\!|u|\!\}^z$. Analogous to the previous case (except that $\Sigma = \Sigma'$). □

Theorem 14 ($\multimap_{\mathtt{v}}$ strongly simulates $\Rightarrow$ via $\{\!|\cdot|\!\}^a$)**.**

1. *If $\{\!|t|\!\}^x \Rightarrow_\otimes Q$ then exists r s.t. $t \multimap_{\mathtt{vdB}} r$ and $\{\!|r|\!\}^x \equiv Q$.*
2. *If $\{\!|t|\!\}^x \Rightarrow_! Q$ then exists r s.t. $t \multimap_{\mathtt{vls}} r$ and $\{\!|r|\!\}^x \equiv Q$.*

Proof. By induction on t. Cases:

- *Values*: if t is a value then $\{\!|t|\!\}^x$ cannot reduce.
- *Application*: if $t = vs$ then $\{\!|vs|\!\}^x = \nu b \nu y(\{\!|v|\!\}^b \mid \overline{b}\langle y,x\rangle \mid \{\!|s|\!\}^y)$ with y and b fresh. Then:

1. *Multiplicative reduction.* Cases of $[\![t]\!]_x \Rightarrow_{\otimes} Q$:
   - *Root*: $\{\!\{v\}\!\}^b = b(z,w).P$ interacts with $\overline{b}\langle y,x\rangle$ on b. Clearly, v is an abstraction $\lambda z.u$ with $\{\!\{u\}\!\}^w = P$, and $t = (\lambda z.u)s$ has a root $\multimap_{\mathtt{vdB}}$ redex. Then, t and $\{\!\{t\}\!\}^x$ are related exactly as in the proof of Theorem 12.1. Note that $b \notin \mathtt{fn}(\{\!\{s\}\!\}^y)$ by Lemma 10, and so there cannot be any multiplicative root interaction involving $\{\!\{s\}\!\}^y$.
   - *Inductive*: $\{\!\{t\}\!\}^x \Rightarrow_{\otimes} Q$ because $\{\!\{s\}\!\}^y \Rightarrow_{\otimes} P$. By *i.h.* we get that there exists r' s.t. $s \to_{\mathtt{dB}} r'$ and $\{\!\{r'\}\!\}^y \equiv P$. Since $v(\!|\,\cdot\,|\!)$ is an evaluation contexts, taking $r := vr'$ we get $t \to_{\mathtt{dB}} r$ and $\{\!\{r\}\!\}^x \equiv P$.
2. *Exponential reduction.* The inductive case (*i.e.* $\{\!\{t\}\!\}^x \Rightarrow_! Q$ because $\{\!\{s\}\!\}^y \Rightarrow_{\otimes} P$) follows by the *i.h.* as in the inductive case for multiplicative reductions. In the root case there cannot be any root exponential reduction. Indeed, $\{\!\{v\}\!\}^b$ would have to be $\overline{z}\langle b\rangle$ and $\{\!\{s\}\!\}^y$ should have a $!z(c).P$ sub-process. This second requirement is only possible if s contains a value v which in $\{\!\{s\}\!\}^y$ is translated with respect to z, so that $\{\!\{v\}\!\}^z = !z(c).P$. But this is impossible because y is fresh (and so $y \neq z$) and any variable name which is used as a parameter in the translation of a subterm of s is either y or it is introduced fresh (and so cannot be equal to z).

- *Substitution*: if $t = \{\!\{s[y/u]\}\!\}^x$ then $\{\!\{t\}\!\}^x = \nu y(\{\!\{s\}\!\}^x \mid \{\!\{u\}\!\}^y)$
  1. *Multiplicative reduction.* If the reduction takes place in $\{\!\{s\}\!\}^x$ or $\{\!\{u\}\!\}^y$ we use the *i.h.* as in the previous inductive cases. And there cannot be any root multiplicative reduction. Indeed, it should be along a special name a free in both $\{\!\{s\}\!\}^x$ and $\{\!\{u\}\!\}^y$, but by Lemma 10 $\{\!\{s\}\!\}^x$ and $\{\!\{u\}\!\}^y$ have no free special name.
  2. *Exponential reduction.* If the reduction takes place in $\{\!\{s\}\!\}^x$ or $\{\!\{u\}\!\}^y$ we use the *i.h.* as in the previous inductive cases.
     Otherwise, an exponential reduction can only be along a variable name z which is free in both $\{\!\{s\}\!\}^x$ and $\{\!\{u\}\!\}^y$. Then $z \neq x$, because $x \notin \mathtt{fn}(\{\!\{u\}\!\}^y)$. Another requirement is that z has to be used as the parameter of the translation of a value v, which is the only way to get a replicated input. The only possibility then is that $z = y$, because all variable parameter names used in the translation and different from x and y are fresh and cannot be in both $\{\!\{s\}\!\}^x$ and $\{\!\{u\}\!\}^y$.
     Now, $\{\!\{s\}\!\}^x$ has to be of the form $N_{\Delta\uplus\Gamma}(\!|\overline{y}\langle a\rangle|\!)$ and $\{\!\{u\}\!\}^y$ has to be of the form $M_{\Delta'\uplus\Gamma'}(\!|!y(b).P|\!)$, for some sets of variable names Δ and Δ' and some sets of special names Γ and Γ', and with $y \notin \Delta \cup \Delta'$. By Lemma 13 we get $\Gamma' = \emptyset$ and that exist v, $L_{\Delta'}$, $\Sigma \subset \Delta$, and A_Σ s.t. $P = \{\!\{v\}\!\}^b$, $u = L_{\Delta'}(\!|v|\!)$, and $s = A_\Sigma(\!|y|\!)$. Summing up, $t = A_\Sigma(\!|y|\!)[y/L_{\Delta'}(\!|v|\!)]$ and it has a $\multimap_{\mathtt{vls}}$ redex which maps on $[\![t]\!]_x \Rightarrow_! Q$ exactly as in the proof of Theorem 12.2. □

Conclusions

We have shown how to refine the relation between the λ-calculus and the π-calculus, getting a perfect match of reductions steps in both call-by-name and call-by-value. The refinements crucially exploits rewriting rules at a distance, and unveil that the π-calculus evaluates λ-terms exactly as linear logic proof nets. A natural continuation would be to extend these relations to calculi with multiplicities [14], which are related to the study of observational equivalence. It would also be interesting to investigate linear weak applicative reduction, in particular in relation with complexity [9] or with Taylor-Ehrhard expansion [22]. Finally, given the compactness of the results and the involved reasoning about bound, free, and fresh variables, it would be interesting to try to formalize this work in Abella [25], which is a proof assistant provided with a nominal quantifier precisely developed to cope with the π-calculus [32] and where reasoning about untyped calculi with binders is very close to pen-and-paper reasoning [6].

References

[1] Samson Abramsky (1993): *Computational Interpretations of Linear Logic*. *Theor. Comput. Sci.* 111(1&2), pp. 3–57. Available at http://dx.doi.org/10.1016/0304-3975(93)90181-R.

[2] Beniamino Accattoli (2011): *Jumping around the box: graphical and operational studies on λ-calculus and Linear Logic*. PhD thesis, *La Sapienza* University of Rome.

[3] Beniamino Accattoli (2012): *An Abstract Factorization Theorem for Explicit Substitutions*. In: *RTA*, pp. 6–21. Available at http://dx.doi.org/10.4230/LIPIcs.RTA.2012.6.

[4] Beniamino Accattoli (2012): *A linear analysis of call-by-value λ-calculus*. Available at the address https://sites.google.com/site/beniaminoaccattoli/Accattoli-Alinearanalysisofcall-by-valuelambdaca

[5] Beniamino Accattoli (2012): *Proof nets and the call-by-value λ-calculus*. LSFA 2012. Available at the address https://sites.google.com/site/beniaminoaccattoli/Accattoli-Proofnetsandthecallbyvaluelambdaca

[6] Beniamino Accattoli (2012): *Proof Pearl: Abella Formalization of λ-Calculus Cube Property*. In: *CPP*, pp. 173–187. Available at http://dx.doi.org/10.1007/978-3-642-35308-6_15.

[7] Beniamino Accattoli & Stefano Guerrini (2009): *Jumping Boxes*. In: *CSL*, pp. 55–70. Available at http://dx.doi.org/10.1007/978-3-642-04027-6_7.

[8] Beniamino Accattoli & Delia Kesner (2010): *The Structural λ-Calculus*. In: *CSL*, pp. 381–395. Available at http://dx.doi.org/10.1007/978-3-642-15205-4_30.

[9] Beniamino Accattoli & Ugo Dal Lago (2012): *On the Invariance of the Unitary Cost Model for Head Reduction*. In: *RTA*, pp. 22–37. Available at http://dx.doi.org/10.4230/LIPIcs.RTA.2012.22.

[10] Beniamino Accattoli & Luca Paolini (2012): *Call-by-Value Solvability, revisited*. In: *FLOPS*, pp. 4–16. Available at http://dx.doi.org/10.1007/978-3-642-29822-6_4.

[11] Emmanuel Beffara (2006): *A Concurrent Model for Linear Logic*. *Electr. Notes Theor. Comput. Sci.* 155, pp. 147–168. Available at http://dx.doi.org/10.1016/j.entcs.2005.11.055.

[12] Gianluigi Bellin & Philip J. Scott (1994): *On the pi-Calculus and Linear Logic*. *Theor. Comput. Sci.* 135(1), pp. 11–65. Available at http://dx.doi.org/10.1016/0304-3975(94)00104-9.

[13] Gérard Boudol (1998): *The π-Calculus in Direct Style*. *Higher-Order and Symbolic Computation* 11(2), pp. 177–208. Available at http://dx.doi.org/10.1023/A:1010064516533.

[14] Gérard Boudol & Cosimo Laneve (1996): *The Discriminating Power of Multiplicities in the Lambda-Calculus*. *Inf. Comput.* 126(1), pp. 83–102. Available at http://dx.doi.org/10.1006/inco.1996.0037.

[15] Luís Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In: *CONCUR*, pp. 222–236. Available at http://dx.doi.org/10.1007/978-3-642-15375-4_16.

[16] Matteo Cimini, Claudio Sacerdoti Coen & Davide Sangiorgi (2010): *Functions as Processes: Termination and the λμμ̃-Calculus*. In: *TGC*, pp. 73–86. Available at http://dx.doi.org/10.1007/978-3-642-15640-3_5.

[17] Pierre Clairambault (2011): *Estimation of the Length of Interactions in Arena Game Semantics*. In: *FOSSACS*, pp. 335–349. Available at http://dx.doi.org/10.1007/978-3-642-19805-2_23.

[18] Vincent Danos, Hugo Herbelin & Laurent Regnier (1996): *Game Semantics & Abstract Machines*. In: *LICS*, pp. 394–405. Available at http://doi.ieeecomputersociety.org/10.1109/LICS.1996.561456.

[19] Vincent Danos & Laurent Regnier (1999): *Reversible, Irreversible and Optimal lambda-Machines*. *Theor. Comput. Sci.* 227(1-2), pp. 79–97. Available at http://dx.doi.org/10.1016/S0304-3975(99)00049-3.

[20] Vincent Danos & Laurent Regnier (2004): *Head Linear Reduction*. Technical Report.

[21] Henry DeYoung, Luís Caires, Frank Pfenning & Bernardo Toninho (2012): *Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication*. In: *CSL*, pp. 228–242. Available at http://dx.doi.org/10.4230/LIPIcs.CSL.2012.228.

[22] Thomas Ehrhard (2012): *Collapsing non-idempotent intersection types*. In: *CSL*, pp. 259–273. Available at http://dx.doi.org/10.4230/LIPIcs.CSL.2012.259.

[23] Thomas Ehrhard & Olivier Laurent (2010): *Interpreting a finitary pi-calculus in differential interaction nets*. *Inf. Comput.* 208(6), pp. 606–633. Available at http://dx.doi.org/10.1016/j.ic.2009.06.005.

[24] Thomas Ehrhard & Laurent Regnier (2006): *Böhm Trees, Krivine's Machine and the Taylor Expansion of Lambda-Terms*. In: *CiE*, pp. 186–197. Available at http://dx.doi.org/10.1007/11780342_20.

[25] Andrew Gacek (2008): *The Abella Interactive Theorem Prover (System Description)*. In: *IJCAR*, pp. 154–161. Available at http://dx.doi.org/10.1007/978-3-540-71070-7_13.

[26] Jean-Yves Girard (1987): *Linear Logic*. *Theoretical Computer Science* 50, pp. 1–102. Available at http://dx.doi.org/10.1016/0304-3975(87)90045-4.

[27] Kohei Honda & Olivier Laurent (2010): *An exact correspondence between a typed pi-calculus and polarised proof-nets*. *Theor. Comput. Sci.* 411(22-24), pp. 2223–2238. Available at http://dx.doi.org/10.1016/j.tcs.2010.01.028.

[28] John Maraist, Martin Odersky, David N. Turner & Philip Wadler (1999): *Call-by-name, Call-by-value, Call-by-need and the Linear lambda Calculus*. *Theor. Comput. Sci.* 228(1-2), pp. 175–210. Available at http://dx.doi.org/10.1016/S0304-3975(98)00358-2.

[29] Gianfranco Mascari & Marco Pedicini (1994): *Head Linear Reduction and Pure Proof Net Extraction*. *Theor. Comput. Sci.* 135(1), pp. 111–137. Available at http://dx.doi.org/10.1016/0304-3975(94)90263-1.

[30] Damiano Mazza (2003): *Pi et Lambda. Une étude sur la traduction des lambda-termes dans le pi-calcul*. Memoire de DEA (in french).

[31] Dale Miller (1992): *The pi-Calculus as a Theory in Linear Logic: Preliminary Results*. In: *ELP*, pp. 242–264. Available at http://dx.doi.org/10.1007/3-540-56454-3_13.

[32] Dale Miller & Alwen Tiu (2010): *Proof search specifications of bisimulation and modal logics for the π-calculus*. *ACM Trans. Comput. Log.* 11(2). Available at http://doi.acm.org/10.1145/1656242.1656248.

[33] Robin Milner (1992): *Functions as Processes*. *Math. Str. in Comput. Sci.* 2(2), pp. 119–141. Available at http://dx.doi.org/10.1017/S0960129500001407.

[34] Robin Milner (2007): *Local Bigraphs and Confluence: Two Conjectures*. *Electr. Notes Theor. Comput. Sci.* 175(3), pp. 65–73. Available at http://dx.doi.org/10.1016/j.entcs.2006.07.035.

[35] Gordon D. Plotkin (1975): *Call-by-Name, Call-by-Value and the lambda-Calculus*. *Theor. Comput. Sci.* 1(2), pp. 125–159. Available at http://dx.doi.org/10.1016/0304-3975(75)90017-1.

[36] Davide Sangiorgi (1994): *The Lazy Lambda Calculus in a Concurrency Scenario*. *Inf. Comput.* 111(1), pp. 120–153. Available at http://dx.doi.org/10.1006/inco.1994.1042.

[37] Davide Sangiorgi (1999): *From lambda to pi; or, Rediscovering continuations*. *Math. Str. in Comput. Sci.* 9(4), pp. 367–401. Available at http://dx.doi.org/10.1017/S0960129599002881.

[38] Davide Sangiorgi & David Walker (2001): *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press.

[39] Bernardo Toninho, Luís Caires & Frank Pfenning (2012): *Functions as Session-Typed Processes*. In: *FoSSaCS*, pp. 346–360. Available at http://dx.doi.org/10.1007/978-3-642-28729-9_23.

[40] Vasco Thudichum Vasconcelos (2005): *Lambda and pi calculi, CAM and SECD machines*. *J. Funct. Program.* 15(1), pp. 101–127. Available at http://dx.doi.org/10.1017/S0956796804005386.

Term Graph Representations for Cyclic Lambda-Terms*

Clemens Grabmayer
Department of Philosophy
Utrecht University
The Netherlands
clemens@phil.uu.nl

Jan Rochel
Department of Computing Sciences
Utrecht University
The Netherlands
jan@rochel.info

We study various representations for cyclic λ-terms as higher-order or as first-order term graphs. We focus on the relation between 'λ-higher-order term graphs' (λ-ho-term-graphs), which are first-order term graphs endowed with a well-behaved scope function, and their representations as 'λ-term-graphs', which are plain first-order term graphs with scope-delimiter vertices that meet certain scoping requirements. Specifically we tackle the question: Which class of first-order term graphs admits a faithful embedding of λ-ho-term-graphs in the sense that (i) the homomorphism-based sharing-order on λ-ho-term-graphs is preserved and reflected, and (ii) the image of the embedding corresponds closely to a natural class (of λ-term-graphs) that is closed under homomorphism?

We systematically examine whether a number of classes of λ-term-graphs have this property, and we find a particular class of λ-term-graphs that satisfies this criterion. Term graphs of this class are built from application, abstraction, variable, and scope-delimiter vertices, and have the characteristic feature that the latter two kinds of vertices have back-links to the corresponding abstraction.

This result puts a handle on the concept of subterm sharing for higher-order term graphs, both theoretically and algorithmically: We obtain an easily implementable method for obtaining the maximally shared form of λ-ho-term-graphs. Furthermore, we open up the possibility to pull back properties from first-order term graphs to λ-ho-term-graphs, properties such as the complete lattice structure of bisimulation equivalence classes with respect to the sharing order.

1 Introduction

Cyclic lambda-terms typically represent infinite λ-terms. In this paper we study term graph representations of cyclic λ-terms and their respective notions of homomorphism, or functional bisimulation.

The context in which the results presented in this paper play a central role is our research on subterm sharing as present in terms of languages such as the λ-calculus with letrec [8, 1], with recursive definitions [2], or languages with μ-recursion [3], and our interest in describing maximal sharing in such settings. Specifically we want to obtain concepts and methods as follows:

- an efficient test for term equivalence with respect to α-renaming and unfolding;
- a notion of 'maximal subterm sharing' for terms in the respective language;
- the efficient computation of the maximally shared form of a term;
- a sharing (pre-)order on unfolding-equivalent terms.

Now our approach is to split the work into a part that concerns properties specific to concrete languages, and into a part that deals with aspects that are common to most of the languages with constructs for expressing subterm sharing. To this end we set out to find classes of term graphs that facilitate faithful interpretations of terms in such languages as (higher-order, and eventually first-order) term graphs, and that are 'well-behaved' in the sense that maximally shared term graphs do always exist. In this way the

*This work was started, and in part carried out, within the framework of the project NWO project *Realising Optimal Sharing (ROS)*, project number 612.000.935, under the direction of Vincent van Oostrom and Doaitse Swierstra.

R. Echahed and D. Plump (Eds.): 7th International Workshop on Computing with Terms and Graphs
EPTCS 110, 2013, pp. 56–73, doi:10.4204/EPTCS.110.7

task can be divided into two parts: an investigation of sharing for term graphs with higher-order features (the aim of this paper), and a study of language-specific aspects of sharing (the aim of a further paper).

Here we study a variety of classes of term graphs for denoting cyclic λ-terms, term graphs with higher-order features and their first-order 'implementations'. All higher-order term graphs we consider are built from three kinds of vertices, which symbolize applications, abstractions, and variable occurrences, respectively. They also carry features that describe notions of *scope*, which are subject to certain conditions that guarantee the meaningfulness of the term graph (that a λ-term is denoted), and in some cases are crucial to define *binding*. The first-order implementations do not have these additional features, but they may contain scope-delimiter vertices.

In particular we study the following three kinds (of classes) of term graphs:

λ-Higher-order-term-graphs (Section 3) are extensions of first-order term graphs by adding a scope function that assigns a set of vertices, its scope, to every abstraction vertex. There are two variants, one with and one without an edge (a *back-link*) from each variable occurrence to its corresponding abstraction vertex. The class with back-links is related to *higher-order term graphs* as defined by Blom in [4], and in fact is an adaptation of that concept for the purpose of representing λ-terms.

Abstraction-prefix based λ-higher-order-term-graphs (Section 4) do not have a scope function but assign, to each vertex w, an abstraction prefix consisting of a word of abstraction vertices that includes those abstractions for which w is in their scope (it actually lists all abstractions for which w is in their 'extended scope' [6]). Abstraction prefixes are aggregations of scope information that is relevant for and locally available at individual vertices.

λ-Term-graphs with scope delimiters (Section 5) are plain first-order term graphs intended to represent higher-order term graphs of the two sorts above, and in this way stand for λ-terms. Instead of relying upon additional features for describing scopes, they use scope-delimiter vertices to signify the end of scopes. Variable occurrences as well as scoping delimiters may or may not have back-links to their corresponding abstraction vertices.

Each of these classes induces a notion of homomorphism (functional bisimulation) and bisimulation. Homomorphisms increase sharing in term graphs, and in this way induce a sharing order. They preserve the unfolding semantics of term graphs[1], and therefore are able to preserve λ-terms that are denoted by term graphs in the unfolding semantics. Term graphs from the classes we consider always represent finite or infinite λ-terms, and in this sense are not 'meaningless'. But this is not shown here. Instead, we lean on motivating examples, intuitions, and the concept of higher-order term graph from [4].

We establish a bijective correspondence between the former two classes, and a correspondence between the latter two classes that is 'almost bijective' (bijective up to sharing or unsharing of scope delimiter vertices). All of these correspondences preserve and reflect the sharing order. Furthermore, we systematically investigate which specific class of λ-term-graphs is closed under homomorphism and renders the mentioned correspondences possible. We prove (in Section 6) that this can only hold for a class in which both variable-occurrence and scope-delimiter vertices have back-links to corresponding abstractions, and establish (in Section 7) that the subclass containing only λ-term-graphs with eager application of scope-closure satisfies these properties. For this class the correspondences allow us:

- to transfer properties known for first-order term graphs, such as the existence of a maximally shared form, from λ-term-graphs to the corresponding classes of higher-order λ-term-graphs;
- to implement maximal sharing for higher-order λ-term-graphs (with eager scope closure) via bisimulation collapse of the corresponding first-order λ-term-graphs (see algorithm in Section 8).

We stress that this paper in its present form is only a report about work in progress, and, while a number of proofs are included, predominantly has the character of an extended abstract.

[1] While this is well-known for first-order term graphs, it can also be proved for the higher-order term graphs considered here.

2 Preliminaries

By $\mathbb{N}$ we denote the natural numbers including zero. For words w over an alphabet A we denote the length of w by $|w|$. For a function $f : A \to B$ we denote by $\mathrm{dom}(f)$ the domain, and by $\mathrm{im}(f)$ the image of f; and for $A_0 \subseteq A$ we denote by $f|_{A_0}$ the restriction of f to A_0.

Let Σ be a signature with arity function $ar : \Sigma \to \mathbb{N}$. A *term graph over* Σ is a tuple $\langle V, lab, args, r\rangle$ where V is a set of *vertices*, $lab : V \to \Sigma$ the *(vertex) label function*, $args : V \to V^*$ the *argument function* that maps every vertex v to the word $args(v)$ consisting of the $ar(lab(v))$ successor vertices of v (hence it holds $|args(v)| = ar(lab(v))$), r, the *root* is a vertex in V, and where every vertex is reachable from the root (by a path that arises by repeatedly going from a vertex to one of its successors). (Note this reachability condition, and mind the fact that term graphs may have infinitely many vertices.) By a Σ-*term-graph* we mean a term graph over Σ. And by $\mathrm{TG}(\Sigma)$ we mean the class of all term graphs over Σ.

Let G be a term graph over signature Σ. As useful notation for picking out any vertex or the i-th vertex from among the ordered successors of a vertex v in G we define the (not indexed) edge relation $\rightarrowtail \subseteq V \times V$, and for each $i \in \mathbb{N}$ the indexed edge relation $\rightarrowtail_i \subseteq V \times V$, between vertices by stipulating that:

$$\begin{aligned} w \rightarrowtail_i w' &\iff \exists w_0, \ldots, w_n \in V.\ args(w) = w_0 \ldots w_n \ \wedge\ w' = w_i \\ w \rightarrowtail w' &\iff \exists i \in \mathbb{N}.\ w \rightarrowtail_i w' \end{aligned}$$

holds for all $w, w' \in V$. We write $w \overset{f}{\rightarrowtail}_i w'$ if $w \rightarrowtail_i w' \wedge lab(w) = f$ holds for $w, w' \in V$, $i \in \mathbb{N}$, $f \in \Sigma$, to indicate the label at the source of an edge. A *path* in G is a tuple of the form $\langle w_0, k_0, w_1, k_1, w_2, \ldots, w_{n-1}, k_{n-1}, w_n\rangle$ where $w_0, \ldots, w_n \in V$ and $n, k_0, k_1, \ldots, k_{n-1} \in \mathbb{N}$ such that $w_0 \rightarrowtail_{k_0} w_1 \rightarrowtail_{k_1} w_2 \cdots w_{n-1} \rightarrowtail_{k_{n-1}} w_n$ holds; paths will usually be denoted in the latter form, using indexed edge relations. An *access path* of a vertex w of G is a path that starts at the root of G, ends in w, and does not visit any vertex twice. Note that every vertex w has at least one access path: since every vertex in a term graph is reachable from the root, there is a path π from r to w; then an access path of w can be obtained from π by repeatedly cutting out *cycles*, that is, parts of the path between different visits to one and the same vertex.

In the sequel, let $G_1 = \langle V_1, lab_1, args_1, r_1\rangle$, $G_2 = \langle V_2, lab_2, args_2, r_2\rangle$ be term graphs over signature Σ.

A *homomorphism*, also called a *functional bisimulation*, from G_1 to G_2 is a morphism from the structure $\langle V_1, lab_1, args_1, r_1\rangle$ to the structure $\langle V_2, lab_2, args_2, r_2\rangle$, that is, a function $h : V_1 \to V_2$ such that, for all $v \in V_1$ it holds:

$$\left.\begin{aligned} lab_1(v) &= lab_2(h(v)) && \text{(labels)} \\ \bar{h}(args_1(v)) &= args_2(h(v)) && \text{(arguments)} \\ h(r_1) &= r_2 && \text{(roots)} \end{aligned}\right\} \tag{1}$$

where $\bar{h}$ is the homomorphic extension of h to words over V_1, that is, to the function $\bar{h} : V_1^* \to V_2^*$, $v_1 \ldots v_n \mapsto h(v_1) \ldots h(v_n)$. In this case we write $G_1 \trianglerighteq_h G_2$, or $G_2 \trianglelefteq_h G_1$. And we write $G_1 \trianglerighteq G_2$, or for that matter $G_2 \trianglelefteq G_1$, if there is a homomorphism (a functional bisimulation) from G_1 to G_2.

Let $\mathsf{f} \in \Sigma$. We write $G_1 \trianglerighteq^{\mathsf{f}} G_2$ or $G_2 \trianglelefteq^{\mathsf{f}} G_1$ if there is a homomorphism h between G_1 and G_2 with the property that for all $w_1, w_2 \in V_1$ with $w_1 \neq w_2$ it holds that $h(w_1) = h(w_2) \Rightarrow lab_1(w_1) = lab_1(w_2) = \mathsf{f}$, that is, if h only 'shares', or 'identifies', vertices when they have label f. If h is such a homomorphism, we also write $G_1 \trianglerighteq^{\mathsf{f}}_h G_2$ or $G_2 \trianglelefteq^{\mathsf{f}}_h G_1$.

A *bisimulation* between G_1 and G_2 is a term graph $G = \langle R, lab, args, r\rangle$ over Σ with $R \subseteq V_1 \times V_2$ and $r = \langle r_1, r_2\rangle$ such that $G_1 \trianglelefteq_{\pi_1} G \trianglerighteq_{\pi_2} G_2$ where π_1 and π_2 are projection functions, defined, for $i \in \{1, 2\}$, by $\pi_i : V_1 \times V_2 \to V_i$, $\langle v_1, v_2\rangle \mapsto v_i$. In this case we write $G_1 \leftrightarrows_R G_2$. And we write $G_1 \leftrightarrows G_2$ if there is a bisimulation between G_1 and G_2.

Alternatively, bisimulations for term graphs can be defined directly as relations on the vertex sets, obtaining the same notion of bisimilarity. In this formulation, a bisimulation between G_1 and G_2 is a relation $R \subseteq V_1 \times V_2$ such that the following conditions hold, for all $\langle v, v' \rangle \in R$:

$$\begin{aligned} \langle r_1, r_2 \rangle &\in R && \text{(roots)} \\ lab_1(v) &= lab_2(v') && \text{(labels)} \\ \langle args_1(v), args_2(v') \rangle &\in R^* && \text{(arguments)} \end{aligned}$$

where $R^* := \big\{ \langle v_1 \cdots v_k, v'_1 \cdots v'_k \rangle \mid v_1, \ldots, v_k \in V_1,\, v'_1, \ldots, v'_k \in V_2 \text{ for } k \in \mathbb{N} \text{ such that} \langle v_i, v'_i \rangle \in R \text{ for all } 1 \leq i \leq k \big\}$.

Bisimulation is an equivalence relation on the class $\mathrm{TG}(\Sigma)$ of term graphs over a signature Σ. The homomorphism (functional bisimulation) relation $\twoheadrightarrow$ is a pre-order on term graphs over a given signature Σ, and it induces a partial order on isomorphism equivalence classes of term graphs over Σ. We will refer to $\twoheadrightarrow$ as the *sharing pre-order*, and will speak of it as *sharing order*, dropping the 'pre'. The bisimulation equivalence class $[[G]_{\sim}]_{\leftrightarrows} := \{[G']_{\sim} \mid G' \leftrightarrows G\}$ of the isomorphism equivalence class $[G]_{\sim}$ of a term graph G is ordered by homomorphism $\twoheadrightarrow$ such that $\langle [[G]_{\sim}]_{\leftrightarrows}, \twoheadrightarrow \rangle$ is a complete lattice [3, 10].Note that, different from e.g. [10], we use the order relation $\twoheadrightarrow$ in the same direction as $\leq$: if $G_1 \twoheadrightarrow G_2$, then G_2 is greater or equal to G_1 in the ordering $\twoheadrightarrow$ (indicating that sharing is typically increased from G_1 to G_2).

Let $\mathcal{K} \subseteq \mathrm{TG}(\Sigma)$ be a subclass of the term graphs over Σ, for a signature Σ. We say that $\mathcal{K}$ is *closed under homorphism* (*closed under bisimulation*) if $G \twoheadrightarrow G'$ (resp. $G \leftrightarrows G'$) for $G, G' \in \mathrm{TG}(\Sigma)$ with $G \in \mathcal{K}$ implies $G' \in \mathcal{K}$. Note these concepts are invariant under considering other signatures Σ' with $\mathcal{K} \subseteq \mathrm{TG}(\Sigma')$.

3 λ-higher-order-Term-Graphs

By Σ^{λ} we designate the signature $\{@, \lambda\}$ with $ar(@) = 2$, and $ar(\lambda) = 1$. By Σ^{λ}_i, for $i \in \{0,1\}$, we denote the extension $\Sigma^{\lambda} \cup \{0\}$ of Σ^{λ} where $ar(0) = i$. The classes of term graphs over Σ^{λ}_0 and Σ^{λ}_1 are denoted by $\mathcal{T}_0$ and $\mathcal{T}_1$, respectively.

Let $G = \langle V, lab, args, r \rangle$ be a term graph over a signature extending Σ^{λ} or Σ^{λ}_i, for $i \in \{0,1\}$. By $V(\lambda)$ we designate the set of *abstraction vertices* of G, that is, the subset of V consisting of all vertices with label λ; more formally, $V(\lambda) := \{v \in V \mid lab(v) = \lambda\}$. Analogously, the sets $V(@)$ and $V(0)$ of *application vertices* and *variable vertices* of G are defined as the sets consisting of all vertices in V with label @ or label 0, respectively. Whether the variable vertices have an outgoing edge depends on the value of i. The intention is to consider two variants of term graphs, one with and one without variable back-links to their corresponding abstraction.

A 'λ-higher-order-term-graph' consists of a Σ^{λ}_i-term-graph together with a scope function that maps abstraction vertices to their scopes ('extended scopes' in [6]), which are subsets of the set of vertices.

Definition 1 (λ-ho-term-graph) Let $i \in \{0,1\}$. A *λ-ho-term-graph* (short for *λ-higher-order-term-graph*) *over* Σ^{λ}_i, is a five-tuple $\mathcal{G} = \langle V, lab, args, r, Sc \rangle$ where $G_{\mathcal{G}} = \langle V, lab, args, r \rangle$ is a Σ^{λ}_i-term-graph, called the term graph *underlying* $\mathcal{G}$, and $Sc : V(\lambda) \to \wp(V)$ is the *scope function* of $\mathcal{G}$ (which maps an abstraction vertex v to a set of vertices called its *scope*) that together with $G_{\mathcal{G}}$ fulfills the following conditions: For all $k \in \{0,1\}$, all vertices $w, w_0, w_1 \in V$, and all abstraction vertices $v, v_0, v_1 \in V(\lambda)$ it holds:

$$\begin{aligned} & \Rightarrow\ r \notin Sc^-(v) && \text{(root)} \\ & \Rightarrow\ v \in Sc(v) && \text{(self)} \\ v_1 \in Sc^-(v_0) \ & \Rightarrow\ Sc(v_1) \subseteq Sc^-(v_0) && \text{(nest)} \\ w \twoheadrightarrow_k w_k \ \wedge\ w_k \in Sc^-(v) \ & \Rightarrow\ w \in Sc(v) && \text{(closed)} \end{aligned}$$

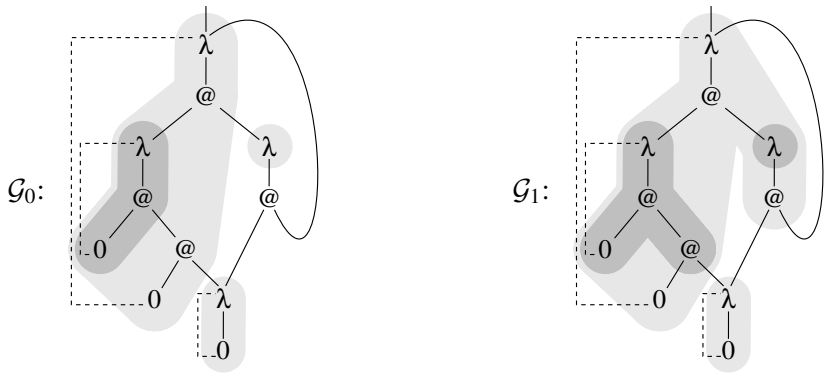


Figure 1: $\mathcal{G}_0$ and $\mathcal{G}_1$ are λ-ho-term-graphs in $\mathcal{H}_i^\lambda$ whereby the dotted back-link edges are present for $i = 1$, but absent for $i = 0$. The underlying term graphs of $\mathcal{G}_0$ and $\mathcal{G}_1$ are identical but their scope functions (signified by the shaded areas) differ. While in $\mathcal{G}_0$ scopes are chosen as small as possible, which we refer to as 'eager scope closure', in $\mathcal{G}_1$ some scopes are closed only later in the graph.

$$w \in V(0) \;\Rightarrow\; \exists v_0 \in V(\lambda).\, w \in Sc^-(v_0) \qquad \text{(scope)}_0$$

$$w \in V(0) \;\wedge\; w \twoheadrightarrow_0 w_0 \;\Rightarrow\; \begin{cases} w_0 \in V(\lambda) \;\wedge \\ \;\wedge\; \forall v \in V(\lambda). \\ \qquad (w \in Sc(v) \Leftrightarrow w_0 \in Sc(v)) \end{cases} \qquad \text{(scope)}_1$$

where $Sc^-(v) := Sc(v) \setminus \{v\}$. Note that if $i = 0$, then $(\text{scope})_1$ is trivially true and hence superfluous, and if $i = 1$, then $(\text{scope})_0$ is redundant, because it follows from $(\text{scope})_1$ in this case. For $w \in V$ and $v \in V(\lambda)$ we say that v is a *binder for* w if $w \in Sc(v)$, and we designate by $\text{bds}(w)$ the set of binders of w.

The classes of λ-ho-term-graphs over Σ_0^λ and Σ_1^λ will be denoted by $\mathcal{H}_0^\lambda$ and $\mathcal{H}_1^\lambda$.

See Fig. 1 for two different λ-ho-term-graphs over Σ_i^λ both of which represent the same term in the λ-calculus with letrec, namely **letrec** $f = \lambda x.\,(\lambda y.\, y\,(x\,g))\,(\lambda z.\, g\,f)$, $g = \lambda u.\, u$ **in** f.

The following lemma states some basic properties of the scope function in λ-ho-term-graphs. Most importantly, scopes in λ-ho-term-graphs are properly nested, in analogy with scopes in finite λ-terms.

Lemma 2 *Let $i \in \{0,1\}$, and let $\mathcal{G} = \langle V, lab, args, r, Sc\rangle$ be a λ-ho-term-graph over Σ_i^λ. Then the following statements hold for all $w \in V$ and $v, v_1, v_2 \in V(\lambda)$:*

(i) If $w \in Sc(v)$, then v is visited on every access path of w, and all vertices on access paths of w after v are in $Sc^-(v)$. Hence (since $G_\mathcal{G}$ is a term graph, every vertex has an access path) $\text{bds}(w)$ is finite.

(ii) If $Sc(v_1) \cap Sc(v_2) \neq \varnothing$ for $v_1 \neq v_2$, then $Sc(v_1) \subseteq Sc^-(v_2)$ or $Sc(v_2) \subseteq Sc^-(v_1)$. As a consequence, if $Sc(v_1) \cap Sc(v_2) \neq \varnothing$, then $Sc(v_1) \subsetneqq Sc(v_2)$ or $Sc(v_1) = Sc(v_2)$ or $Sc(v_2) \subsetneqq Sc(v_1)$.

(iii) If $\text{bds}(w) \neq \varnothing$, then $\text{bds}(w) = \{v_0, \ldots, v_n\}$ for $v_0, \ldots, v_n \in V(\lambda)$ and $Sc(v_n) \subsetneqq Sc(v_{n-1}) \ldots \subsetneqq Sc(v_0)$.

Proof. Let $i \in \{0,1\}$, and let $\mathcal{G} = \langle V, lab, args, r, Sc\rangle$ be a λ-ho-term-graph over Σ_i^λ.

For showing (i), let $w \in V$ and $v \in V(\lambda)$ be such that $w \in Sc(v)$. Suppose that $\pi : r = w_0 \twoheadrightarrow_{k_0} w_1 \twoheadrightarrow_{k_1} w_2 \cdots \twoheadrightarrow_{k_{n-1}} w_n = w$ is an access path of w. If $w = v$, then nothing remains to be shown. Otherwise $w_n = w \in Sc^-(v)$, and, if $n > 0$, then by (closed) it follows that $w_{n-1} \in Sc(v)$. This argument can be repeated to find subsequently smaller i with $w_i \in Sc(v)$ and $w_{i+1}, \ldots, w_n \in Sc^-(v)$. We can proceed as long as

$w_i \in Sc^-(v)$. But since, due to (root), $w_0 = r \notin Sc^-(v)$, eventually we must encounter an i_0 such that such that $w_{i_0+1}, \ldots, w_n \in Sc^-(v)$ and $w_{i_0} \in Sc(v) \setminus Sc^-(v)$. This implies $w_{i_0} = v$, showing that v is visited on π.

For showing (ii), let $w \in V$ and $v_1, v_2 \in V(\lambda)$, $v_1 \neq v_2$ be such that $w \in Sc(v_1) \cap Sc(v_2)$. Let π be an access path of w. Then it follows by (i) that both v_1 and v_2 are visited on π, and that, depending on whether v_1 or v_2 is visited first on π, either $v_2 \in Sc^-(v_1)$ or $v_1 \in Sc^-(v_2)$. Then due to (nest) it follows that either $Sc(v_2) \subseteq Sc^-(v_1)$ holds or $Sc(v_1) \subseteq Sc^-(v_2)$.

Finally, statement (iii) is an easy consequence of statement (ii). □

Remark 3 The notion of λ-ho-term-graph is an adaptation of the notion of 'higher-order term graph' by Blom [4, Def. 3.2.2] for the purpose of representing finite or infinite λ-terms or cyclic λ-terms, that is, terms in the λ-calculus with letrec. In particular, λ-ho-term-graphs over Σ_1^λ correspond closely to higher-order term graphs over signature Σ^λ. But they differ in the following respects:

Abstractions: Higher-order term graphs in [4] are graph representations of finite or infinite terms in Combinatory Reduction Systems (CRSs). They typically contain abstraction vertices with label □ that represent CRS-abstractions. In contrast, λ-ho-term-graphs have abstraction vertices with label λ that denote λ-abstractions.

Signature: Whereas higher-order term graphs in [4] are based on an arbitrary CRS-signature, λ-ho-term-graphs over Σ_1^λ only contain the application symbol @ and the variable-occurrence symbol 0 in addition to the abstraction symbol λ.

Variable back-links and variable occurrence vertices: In the formalization of higher-order term graphs in [4] there are no explicit vertices that represent variable occurrences. Instead, variable occurrences are represented by back-link edges to abstraction vertices. Actually, in the formalization chosen in [4, Def. 3.2.1], a back-link edge does not directly target the abstraction vertex v it refers to, but ends at a special variant vertex $\bar{v}$ of v. (Every such variant abstraction vertex $\bar{v}$ could be looked upon as a variable vertex that is shared by all edges that represent occurrences of the variable bound by the abstraction vertex v.)

In λ-ho-term-graphs over Σ_1^λ a variable occurrence is represented by a variable-occurrence vertex that as outgoing edge has a back-link to the abstraction vertex that binds the occurrence.

conditions on the scope function: While the conditions (root), (self), (nest), and (closed) on the scope function in higher-order term graphs in [4, Def. 3.2.2] correspond directly to the respective conditions in Def. 1, the difference between the condition (scope) there and $(\text{scope})_1$ in Def. 1 reflects the difference described in the previous item.

free variables: Whereas the higher-order term graphs in [4] cater for the presence of free variables, free variables have been excluded from the basic format of λ-ho-term-graphs.

Definition 4 (homomorphism, bisimulation) Let $i \in \{0,1\}$. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be λ-ho-term-graphs over Σ_i^λ with $\mathcal{G}_k = \langle V_k, lab_k, args_k, r_k, Sc_k \rangle$ for $k \in \{1,2\}$.

A *homomorphism*, also called a *functional bisimulation*, from $\mathcal{G}_1$ to $\mathcal{G}_2$ is a morphism from the structure $\langle V_1, lab_1, args_1, r_1, Sc_1 \rangle$ to the structure $\langle V_2, lab_2, args_2, r_2, Sc_2 \rangle$, that is, a function $h : V_1 \to V_2$ such that, for all $v \in V_1$ the conditions (labels), (arguments), and (roots) in in (1) are satisfied, and additionally, for all $v \in V_1(\lambda)$:

$$\bar{\bar{h}}(Sc_1(v)) = Sc_2(h(v)) \qquad \text{(scope functions)} \tag{2}$$

where $\bar{\bar{h}}$ is the homomorphic extension of h to sets over V_1, that is, to the function $\bar{h} : \wp(V_1) \to \wp(V_2)$, $A \mapsto \{h(a) \mid a \in A\}$. If there exists a homomorphism (a functional bisimulation) h from $\mathcal{G}_1$ to $\mathcal{G}_2$, then we write $\mathcal{G}_1 \rightrightarrows_h \mathcal{G}_2$ or $\mathcal{G}_2 \lhd\!\lhd_h \mathcal{G}_1$, or, dropping h as subscript, $\mathcal{G}_1 \rightrightarrows \mathcal{G}_2$ or $\mathcal{G}_2 \lhd\!\lhd \mathcal{G}_1$.

A *bisimulation* between $\mathcal{G}_1$ and $\mathcal{G}_2$ is a term graph $\mathcal{G} = \langle R, lab, args, r, Sc \rangle$ over Σ with $R \subseteq V_1 \times V_2$ and $r = \langle r_1, r_2 \rangle$ such that $\mathcal{G}_1 \lhd\!\lhd_{\pi_1} \mathcal{G} \rightrightarrows_{\pi_2} \mathcal{G}_2$ where π_1 and π_2 are projection functions, defined, for $i \in \{1,2\}$, by $\pi_i : V_1 \times V_2 \to V_i$, $\langle v_1, v_2 \rangle \mapsto v_i$. If there exists a bisimulation R between $\mathcal{G}_1$ and $\mathcal{G}_2$, then we write $\mathcal{G}_1 \leftrightarrows_R \mathcal{G}_2$, or just $\mathcal{G}_1 \leftrightarrows \mathcal{G}_2$.

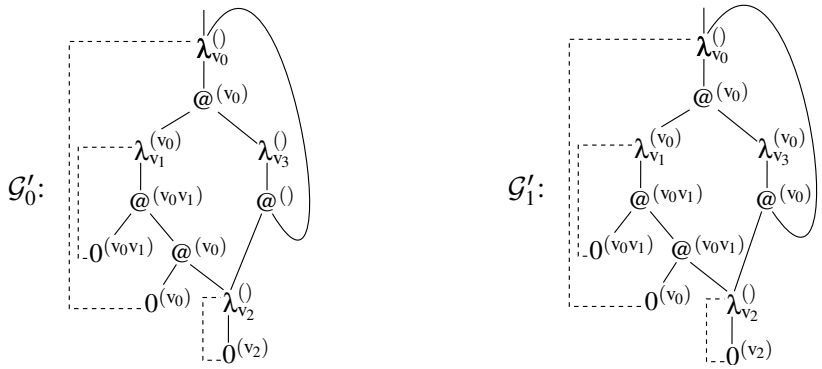


Figure 2: The λ-ap-ho-term-graphs corresponding to the λ-ho-term-graphs in Fig 1. The subscripts of abstraction vertices indicate their names. The super-scripts of vertices indicate their abstraction-prefixes. A precise formulation of this correspondence is given in Example 11.

4 Abstraction-prefix based λ-h.o.-term-graphs

By an 'abstraction-prefix based λ-higher-order-term-graph' we will mean a term-graph over Σ_i^λ for $i \in \{0,1\}$ that is endowed with a correct abstraction prefix function that maps abstraction vertices v to words of vertices that represent the sequence of abstractions that have v in their scope. The conceptual difference between the abstraction-prefix function and the scope function is that the former makes the most essential scoping information locally available. It explicitly states all 'extended scopes' (induced by the transitive closure of the in-scope relation, see [6]) in which a node resides in the order of their nesting. This approach leads to simpler correctness conditions.

Definition 5 (correct abstraction-prefix function for Σ_i^λ-term-graphs) Let $G = \langle V, lab, args, r\rangle$ be, for an $i \in \{0,1\}$, a Σ_i^λ-term-graph.

A function $P : V \to V^*$ from vertices of G to words of vertices is called an *abstraction-prefix function* for G. Such a function is called *correct* if for all $w, w_0, w_1 \in V$ and $k \in \{0,1\}$:

$$\begin{aligned}
&\Rightarrow\ P(r) = \varepsilon && \text{(root)}\\
w \in V(\lambda)\ \wedge\ w \rightarrowtail_0 w_0\ &\Rightarrow\ P(w_0) \leq P(w)w && (\lambda)\\
w \in V(@)\ \wedge\ w \rightarrowtail_k w_k\ &\Rightarrow\ P(w_k) \leq P(w) && (@)\\
w \in V(0)\ &\Rightarrow\ P(w) \neq \varepsilon && (0)_0\\
w \in V(0)\ \wedge\ w \rightarrowtail_0 w_0\ &\Rightarrow\ w_0 \in V(\lambda)\ \wedge\ P(w_0)w_0 = P(w) && (0)_1
\end{aligned}$$

Note that analogously as in Def. 1, if $i = 0$, then $(0)_1$ is trivially true and hence superfluous, and if $i = 1$, then $(0)_0$ is redundant, because it follows from $(0)_1$ in this case.

We say that G *admits* a correct abstraction-prefix function if such a function exists for G.

Definition 6 (λ-ap-ho-term-graph) Let $i \in \{0,1\}$. A *λ-ap-ho-term-graph* (short for *abstraction-prefix based λ-higher-order-term-graph*) over signature Σ_i^λ is a five-tuple $\mathcal{G} = \langle V, lab, args, r, P\rangle$ where $G_\mathcal{G} = \langle V, lab, args, r\rangle$ is a Σ_i^λ-term-graph, called the term graph *underlying* $\mathcal{G}$, and P is a correct abstraction-prefix function for $G_\mathcal{G}$. The classes of λ-ap-ho-term-graphs over Σ_i^λ will be denoted by $\mathcal{H}_i^{(\lambda)}$.

See Fig. 2 for two examples, which correspond, as we will see, to the λ-ho-term-graphs in Fig. 1.

The following lemma states some basic properties of the scope function in λ-ap-ho-term-graphs.

Lemma 7 *Let $i \in \{0,1\}$ and let $\mathcal{G} = \langle V, lab, args, r, P\rangle$ be a λ-ap-ho-term-graph over Σ^λ_i. Then the following statements hold:*

(i) Suppose that, for some $v, w \in V$, v occurs in $P(w)$. Then $v \in V(\lambda)$, occurs in $P(w)$ only once, and every access path of w passes through v, but does not end there, and thus $w \neq v$. Furthermore it holds: $P(v)\,v \leq P(w)$. In particular, if $P(w) = p\,v$, then $P(v) = p$.

(ii) Vertices in abstraction prefixes are abstraction vertices, and hence P is of the form $P : V \to (V(\lambda))^*$.

(iii) For all $v \in V(\lambda)$ it holds: $v \notin P(v)$.

(iv) While access paths might end in vertices in $V(0)$, they only pass through vertices in $V(\lambda) \cup V(@)$.

Proof. Let $i \in \{0,1\}$ and let $\mathcal{G} = \langle V, lab, args, r, P\rangle$ be a λ-ap-ho-term-graph over Σ^λ_i.

For showing (i), let $v, w \in V$ be such that v occurs in $P(w)$. Suppose further that π is an access path of w. Note that when walking through π the abstraction prefix starts out empty (due to (root)), and is expanded only in steps from vertices $v' \in V(\lambda)$ (due to (λ), $(@)$, and $(0)_1$) in which just v' is added to the prefix on the right (due to (λ)). Since v occurs in $P(w)$, it follows that $v \in V(\lambda)$, that v must be visited on π, and that π continues after the visit to v. That π is an access path also entails that v is not visited again on π, hence that $w \neq v$ and that v occurs only once in $P(w)$, and that $P(v)\,v$, the abstraction prefix of the successor vertex of v on π, is a prefix of the abstraction prefix of every vertex that is visited on π after v.

Statements (ii) and (iii) follow directly from statement (i).

For showing (iv), consider an access path $\pi : r = w_0 \rightarrowtail \cdots \rightarrowtail w_n$ that leads to a vertex $w_n \in V(0)$. If $i = 0$, then there is no path that extends π properly beyond w_n. So suppose $i = 1$, and let $w_{n+1} \in V$ be such that $w_n \rightarrowtail_0 w_{n+1}$. Then $(0)_1$ implies that $P(w_n) = P(w_{n+1})\,w_{n+1}$, from which it follows by (i) that w_{n+1} is visited already on π. Hence π does not extend to a longer path that is again an access path. □

Definition 8 (homomorphism, bisimulation) Let $i \in \{0,1\}$. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be λ-ap-ho-term-graphs over Σ^λ_i with $\mathcal{G}_k = \langle V_k, lab_k, args_k, r_k, P_k\rangle$ for $k \in \{1,2\}$.

A *homomorphism*, also called a *functional bisimulation*, from $\mathcal{G}_1$ to $\mathcal{G}_2$ is a morphism from the structure $\langle V_1, lab_1, args_1, r_1, P_1\rangle$ to the structure $\langle V_2, lab_2, args_2, r_2, P_2\rangle$, that is, a function $h : V_1 \to V_2$ such that, for all $v \in V_1$ the conditions (labels), (arguments), and (roots) in in (1) are satisfied, and additionally, for all $v \in V_1$:

$$\bar{h}(P_1(v)) = P_2(h(v)) \qquad \text{(abstraction-prefix functions)} \tag{3}$$

where $\bar{h}$ is the homomorphic extension of h to words over V_1. In this case we write $\mathcal{G}_1 \rightrightarrows_h \mathcal{G}_2$, or $\mathcal{G}_2 \leftleftarrows_h \mathcal{G}_1$. And we write $\mathcal{G}_1 \rightrightarrows \mathcal{G}_2$, or for that matter $\mathcal{G}_2 \leftleftarrows \mathcal{G}_1$, if there is a homomorphism (a functional bisimulation) from $\mathcal{G}_1$ to $\mathcal{G}_2$.

A *bisimulation* between $\mathcal{G}_1$ and $\mathcal{G}_2$ is a term graph $\mathcal{G} = \langle R, lab, args, r, Sc\rangle$ over Σ with $R \subseteq V_1 \times V_2$ and $r = \langle r_1, r_2\rangle$ such that $\mathcal{G}_1 \leftleftarrows_{\pi_1} \mathcal{G} \rightrightarrows_{\pi_2} \mathcal{G}_2$ where π_1 and π_2 are projection functions, defined, for $i \in \{1,2\}$, by $\pi_i : V_1 \times V_2 \to V_i$, $\langle v_1, v_2\rangle \mapsto v_i$. If there exists a homomorphism (a functional bisimulation) h from $\mathcal{G}_1$ to $\mathcal{G}_2$, then we write $\mathcal{G}_1 \rightrightarrows_h \mathcal{G}_2$ or $\mathcal{G}_2 \leftleftarrows_h \mathcal{G}_1$, or, dropping h as subscript, $\mathcal{G}_1 \rightrightarrows \mathcal{G}_2$ or $\mathcal{G}_2 \leftleftarrows \mathcal{G}_1$.

The following proposition defines mappings between λ-ho-term-graphs and λ-ap-ho-term-graphs by which we establish a bijective correspondence between the two classes. For both directions the underlying λ-term-graph remains unchanged. A_i derives an abstraction-prefix function P from a scope function by assigning to each vertex a word of its binders in the correct nesting order. B_i defines its scope function Sc by assigning to each λ-vertex v the set of vertices that have v in their prefix (along with v since a vertex never has itself in its abstraction prefix).

Proposition 9 *For each $i \in \{0,1\}$, the mappings A_i and B_i are well-defined between the class of λ-ho-term-graphs over Σ_i^λ and the class of λ-ap-ho-term-graphs over Σ_i^λ:*

$$\left.\begin{array}{l} A_i : \mathcal{H}_i^\lambda \to \mathcal{H}_i^{(\lambda)},\ \mathcal{G} = \langle V, lab, args, r, Sc\rangle \mapsto A_i(\mathcal{G}) := \langle V, lab, args, r, P\rangle \\ \qquad\qquad \text{where } P : V \to V^*,\ w \mapsto v_0 \cdots v_n \text{ if } \mathrm{bds}(w) \setminus \{w\} = \{v_0, \ldots, v_n\} \text{ and} \\ \qquad\qquad\qquad\qquad Sc(v_n) \subsetneqq Sc(v_{n-1}) \ldots \subsetneqq Sc(v_0) \end{array}\right\} \quad (4)$$

$$\left.\begin{array}{l} B_i : \mathcal{H}_i^{(\lambda)} \to \mathcal{H}_i^\lambda,\ \mathcal{G} = \langle V, lab, args, r, P\rangle \mapsto A_i(\mathcal{G}) := \langle V, lab, args, r, Sc\rangle \\ \qquad\qquad \text{where } Sc : V(\lambda) \to \wp(V),\ v \mapsto \{w \in V \mid v \text{ occurs in } P(w)\} \cup \{v\} \end{array}\right\} \quad (5)$$

Theorem 10 (correspondence of λ-ho-term-graphs with λ-ap-ho-term-graphs) *For each $i \in \{0,1\}$ it holds that the mappings A_i in* (4) *and B_i in* (5) *are each other's inverse; thus they define a bijective correspondence between the class of λ-ho-term-graphs over Σ_i^λ and the class of λ-ap-ho-term-graphs over Σ_i^λ. Furthermore, they preserve and reflect the sharing orders on $\mathcal{H}_i^\lambda$ and on $\mathcal{H}_i^{(\lambda)}$:*

$$\begin{array}{lll} (\forall \mathcal{G}_1, \mathcal{G}_2 \in \mathcal{H}_i^\lambda) & \mathcal{G}_1 \trianglerighteq \mathcal{G}_2 & \iff A_i(\mathcal{G}_1) \trianglerighteq A_i(\mathcal{G}_1) \\ (\forall \mathcal{G}_1, \mathcal{G}_2 \in \mathcal{H}_i^{(\lambda)}) & B_i(\mathcal{G}_1) \trianglerighteq B_i(\mathcal{G}_1) & \iff \mathcal{G}_1 \trianglerighteq \mathcal{G}_2 \end{array}$$

Example 11 The λ-ho-term-graphs in Fig. 1 correspond to the λ-ap-ho-term-graphs in Fig. 2 via the mappings A_i and B_i as follows: $A_i(\mathcal{G}_0) = \mathcal{G}'_0$, $A_i(\mathcal{G}_1) = \mathcal{G}'_1$, $B_i(\mathcal{G}_0) = \mathcal{G}'_0$, $A_i(\mathcal{G}_1) = \mathcal{G}'_1$.

For λ-ho-term-graphs over the signature Σ_0^λ (that is, without variable back-links) essential binding information is lost when looking only at the underlying term graph, to the extent that λ-terms cannot be unambiguously represented anymore. For instance the λ-ho-term-graphs that represent the λ-terms $\lambda xy.xy$ and $\lambda xy.xx$ have the same underlying term graph. The same holds for λ-ap-ho-term-graphs.

This is not the case for λ-ho-term-graphs (λ-ap-ho-term-graphs) over Σ_1^λ, because the abstraction vertex to which a variable-occurrence vertex belongs is uniquely identified by the back-link. This is the reason why the following notion is only defined for the signature Σ_1^λ.

Definition 12 (λ-term-graph over Σ_1^λ) A term graph G over Σ_1^λ is called a *λ-term-graph* over Σ_1^λ if G admits a correct abstraction-prefix function. By $\mathcal{T}_1^{(\lambda)}$ we denote the class of λ-term-graphs over Σ_1^λ.

In the rest of this section we examine, and then dismiss, a naive approach to implementing functional bisimulation on λ-ho-term-graphs or λ-ap-ho-term-graphs, which is to apply the homomorphism on the underlying term graph, hoping that this application would simply extend to the λ-ho-term-graph (λ-ap-ho-term-graph) without further ado. We demonstrate that this approach fails, concluding that a faithful first-order implementation of functional bisimulation must not be negligent of the scoping information.

Definition 13 (scope- and abstraction-prefix-forgetful mappings) Let $i \in \{0,1\}$. The *scope-forgetful mapping* ScF_i^λ and the *abstraction-prefix-forgetful mapping* $PF_1{}^{(\lambda)}$ map λ-ho-term-graphs in $\mathcal{T}_i^{(\lambda)}$, and respectively, λ-ho-term-graphs in $\mathcal{T}_i^{(\lambda)}$ to their underlying term graphs:

$$\begin{array}{l} ScF_i^\lambda : \mathcal{H}_i^\lambda \to \mathcal{T}_i,\ \langle V, lab, args, r, Sc\rangle \mapsto \langle V, lab, args, r\rangle \\ PF_i{}^{(\lambda)} : \mathcal{H}_i^{(\lambda)} \to \mathcal{T}_i,\ \langle V, lab, args, r, P\rangle \mapsto \langle V, lab, args, r\rangle \end{array}$$

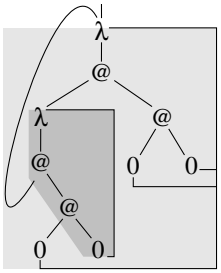
Definition 14 Let $\mathcal{G}$ be a λ-ho-term-graph over Σ_i^λ for $i \in \{0,1\}$ with underlying term graph $ScF_i^\lambda(\mathcal{G})$. And suppose that $ScF_i^\lambda(\mathcal{G}) \trianglerighteq_h G'$ holds for a term graph G' over Σ_i^λ and a functional bisimulation h. We say that *h extends to a functional bisimulation on $\mathcal{G}$* if G' can be endowed with a scope function to obtain a λ-ho-term-graph $\mathcal{G}'$ with $ScF_i^\lambda(\mathcal{G}') = G'$ and such that it holds $\mathcal{G} \trianglerighteq_h \mathcal{G}'$.

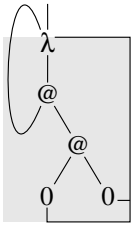
We say that a class $\mathcal{K}$ of λ-ho-term-graphs is *closed under functional bisimulations on the underlying term graphs* if for every $\mathcal{G} \in \mathcal{K}$ and for every homomorphism h on the term graph underlying $\mathcal{G}$ that witnesses $ScF_i^\lambda(\mathcal{G}) \rightrightarrows_h G'$ for a term graph G' there exists $\mathcal{G}' \in \mathcal{K}$ with $ScF_i^\lambda(\mathcal{G}') = G'$ such that $\mathcal{G} \rightrightarrows_h \mathcal{G}'$ holds, that is, h is also a homorphism between $\mathcal{G}$ and $\mathcal{G}'$.

These notions are also extended, by analogous stipulations, to λ-ap-ho-term-graphs over Σ_i^λ for $i \in \{0,1\}$ and their underlying term graphs.

Proposition 15 *Neither the class $\mathcal{H}_1^\lambda$ of λ-ho-term-graphs nor the class ${\mathcal{H}_1}^{(\lambda)}$ of λ-ap-ho-term-graphs is closed under functional bisimulations on the underlying term graphs.*

Proof. In view of Thm. 10 it suffices to show the statement for $\mathcal{H}_1^\lambda$. We show that not every functional bisimulation on the term graph underlying a λ-ho-term-graph over Σ_1^λ extends to a functional bisimulation on the higher-order term graphs. Consider the following term graphs G_0 and G_1 over Σ_1^λ (at first, please ignore the scope shading):

G_1: 

G_0: 

There is an obvious homomorphism h that witnesses $G_1 \rightrightarrows_h G_0$. Both of these term graphs extend to λ-ho-term-graphs by suitable scope functions (one possibility per term graph is indicated by the scope shadings above; G_1 actually admits two possibilities). However, h does not extend to any of the λ-ho-term-graphs $\mathcal{G}_1$ and $\mathcal{G}_0$ that extend G_1 and G_0, respectively. □

The next proposition is merely a reformulation of Prop. 15.

Proposition 16 *The scope-forgetful mapping ScF_1^λ on $\mathcal{H}_1^\lambda$ and the abstraction-prefix-forgetful mapping ${PF_1}^{(\lambda)}$ on ${\mathcal{H}_1}^{(\lambda)}$ preserve, but do not reflect, the sharing orders on these classes. In particular:*

$$(\forall \mathcal{G}_1, \mathcal{G}_2 \in {\mathcal{H}_1}^{(\lambda)}) \qquad \mathcal{G}_1 \rightrightarrows \mathcal{G}_2 \implies {PF_1}^{(\lambda)}(\mathcal{G}_1) \rightrightarrows {PF_1}^{(\lambda)}(\mathcal{G}_2)$$
$$(\exists \mathcal{G}_1, \mathcal{G}_2 \in {\mathcal{H}_1}^{(\lambda)}) \qquad \mathcal{G}_1 \not\rightrightarrows \mathcal{G}_2 \ \wedge \ {PF_1}^{(\lambda)}(\mathcal{G}_1) \rightrightarrows {PF_1}^{(\lambda)}(\mathcal{G}_2)$$

As a consequence of this proposition it is not possible to faithfully implement functional bisimulation on λ-ho-term-graphs and λ-ap-ho-term-graphs by only considering the underlying term graphs, and in doing so neglecting[2] the scoping information from the scope function, or respectively, from the abstraction prefix function. In order to yet be able to implement functional bisimulation of λ-ho-term-graphs and λ-ap-ho-term-graphs in a first-order setting, in the next section we introduce a class of first-order term graphs that accounts for scoping by means of scope delimiter vertices.

5 λ-Term-Graphs with Scope Delimiters

For all $i \in \{0,1\}$ and $j \in \{1,2\}$ we define the extensions $\Sigma_{i,j}^\lambda := \Sigma^\lambda \cup \{0, \mathsf{S}\}$ of the signature Σ^λ where $ar(0) = i$ and $ar(\mathsf{S}) = j$, and we denote the class of term graphs over signature $\Sigma_{i,j}^\lambda$ by $\mathcal{T}_{i,j}$.

[2] In the case of Σ_1^λ implicit information about possible scopes is being kept, due to the presence of back-links from variable occurrence vertices to abstraction vertices. But this is not enough for reflecting the sharing order under the forgetful mappings.

Let G be a term graph with vertex set V over a signature extending $\Sigma^{\lambda}_{i,j}$ for $i \in \{0,1\}$ and $j \in \{1,2\}$. We denote by $V(\mathsf{S})$ the subset of V consisting of all vertices with label S, which are called the *delimiter vertices* of G. Delimiter vertices signify the end of an 'extended scope' [6]. They are analogous to occurrences of function symbols S in representations of λ-terms in a nameless de-Bruijn index [5] form in which Dedekind numerals based on 0 and the successor function symbol S are used (this form is due to Hendriks and van Oostrom, see also [9], and is related to their end-of-scope symbol Λ [7]).

Analogously as for the classes $\mathcal{H}^{\lambda}_{i}$ and $\mathcal{H}_{i}{}^{(\lambda)}$, the index i will determine whether in correctly formed λ-term-graphs (defined below) variable vertices have back-links to the corresponding abstraction. Here additionally scope-delimiter vertices have such back-links (if $j = 2$) or not (if $j = 1$).

Definition 17 (correct abstraction-prefix function for $\Sigma^{\lambda}_{i,j}$-term-graphs) Let $G = \langle V, lab, args, r\rangle$ be a $\Sigma^{\lambda}_{i,j}$-term-graph for an $i \in \{0,1\}$ and an $j \in \{1,2\}$.

A function $P : V \to V^*$ from vertices of G to words of vertices is called an *abstraction-prefix function* for G. Such a function is called *correct* if for all $w, w_0, w_1 \in V$ and $k \in \{0,1\}$ it holds:

$$
\begin{array}{rcll}
 & \Rightarrow & P(r) = \varepsilon & (\text{root}) \\
w \in V(\lambda) \wedge w \rightarrowtail_0 w_0 & \Rightarrow & P(w_0) = P(w)w & (\lambda) \\
w \in V(@) \wedge w \rightarrowtail_k w_k & \Rightarrow & P(w_k) = P(w) & (@) \\
w \in V(0) & \Rightarrow & P(w) \neq \varepsilon & (0)_0 \\
w \in V(0) \wedge w \rightarrowtail_0 w_0 & \Rightarrow & w_0 \in V(\lambda) \wedge P(w_0)w_0 = P(w) & (0)_1 \\
w \in V(\mathsf{S}) \wedge w \rightarrowtail_0 w_0 & \Rightarrow & P(w_0)v = P(w) \text{ for some } v \in V & (\mathsf{S})_1 \\
w \in V(\mathsf{S}) \wedge w \rightarrowtail_1 w_1 & \Rightarrow & w_1 \in V(\lambda) \wedge P(w_1)w_1 = P(w) & (\mathsf{S})_2
\end{array}
$$

Note that analogously as in Def. 1 and in Def. 6, if $i = 0$, then $(0)_1$ is trivially true and hence superfluous, and if $i = 1$, then $(0)_0$ is redundant, because it follows from $(0)_1$ in this case. Additionally, if $j = 1$, then $(\mathsf{S})_2$ is trivially true and therefore superfluous.

Definition 18 (λ-term-graph over $\Sigma^{\lambda}_{i,j}$) Let $i \in \{0,1\}$ and $j \in \{1,2\}$. A *λ-term-graph (with scope-delimiters)* over $\Sigma^{\lambda}_{i,j}$ is a $\Sigma^{\lambda}_{i,j}$-term-graph that admits a correct abstraction-prefix function. The class of λ-term-graphs over $\Sigma^{\lambda}_{i,j}$ is denoted by $\mathcal{T}^{(\lambda)}_{i,j}$.

See Fig. 3 for examples, that, as we will see, correspond to the ho-term-graphs in Fig. 1 and in Fig. 2.

Lemma 19 *Let $i \in \{0,1\}$ and $j \in \{1,2\}$, and let $G = \langle V, lab, args, r\rangle$ be a λ-term-graph over $\Sigma^{\lambda}_{i,j}$. Then the statements* (i)–(iii) *in Lemma 7 hold, and additionally:*

(iv) Access paths may end in vertices in $V(0)$, but only pass through vertices in $V(\lambda) \cup V(@) \cup V(\mathsf{S})$, and depart from vertices in $V(\mathsf{S})$ only via indexed edges ${}^{\mathsf{S}}\!\!\rightarrowtail_0$.

(v) There exists precisely one correct abstraction-prefix function on G.

Proof. That also here statements (i)–(iii) in Lemma 7 hold, and that statement (iv) holds, can be shown analogously as in the proof of the respective items of Lemma 7. For (v) it suffices to observe that if P is a correct abstraction-prefix function for G, then, for all $w \in V$, the value $P(w)$ of P at w can be computed by choosing an arbitrary access path π from r to w and using the conditions (λ), $(@)$, and $(\mathsf{S})_0$ to determine in a stepwise manner the values of P at the vertices that are visited on π. Hereby note that in every transition along an edge on π the length of the abstraction prefix only changes by at most 1. □

Now we define a precise relationship between λ-term-graphs and λ-ap-ho-term-graphs via translation mappings between these classes:

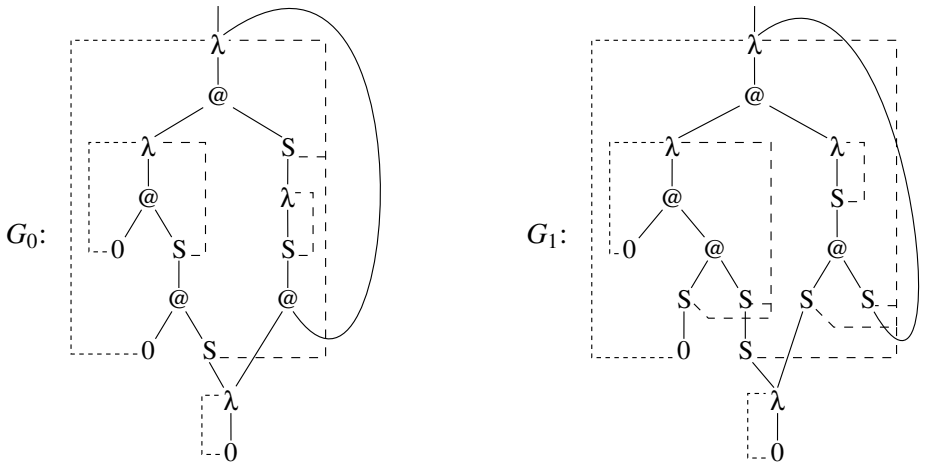


Figure 3: The λ-term-graphs corresponding to the λ-ap-ho-term-graphs from Fig 2 and the λ-ho-term-graphs from Fig 1. A precise formulation of this correspondence is given in Example 23.

The mapping $G_{i,j}$ (see Prop. 20) produces a λ-term-graph for any given λ-ap-ho-term-graph by adding to the original set of vertices a number of delimiter vertices at the appropriate places. That is, at every position where the abstraction prefix decreases by n elements, n S-vertices are inserted. In the image, the original abstraction prefix is retained as part of the vertices. This can be considered intermittent information used for the purpose of defining the edges of the image.

The mapping $\mathcal{G}_{i,j}$ (see Prop. 21) back to λ-ap-ho-term-graphs is simpler because it only has to erase the S-vertices, and add the correct abstraction prefix that exists for the λ-term-graph to be translated.

Proposition 20 *Let $i \in \{0,1\}$ and $j \in \{1,2\}$. The mapping $G_{i,j}$ defined below is well-defined between the class of λ-term-graphs over $\Sigma^{\lambda}_{i,j}$ and the class of λ-ap-ho-term-graphs over Σ^{λ}_{i}:*

$$G_{i,j} : \mathcal{H}_i^{(\lambda)} \to \mathcal{T}_{i,j}^{(\lambda)},\ \ \mathcal{G} = \langle V, lab, args, r, P\rangle \mapsto G_{i,j}(\mathcal{G}) := \langle V', lab', args', r'\rangle$$

where:

$$V' := \{\langle w, P(w)\rangle \mid w \in V\} \cup \left\{ \langle w,k,w',p\rangle \;\middle|\; w,w' \in V,\ w \rightarrowtail_k w',\ \begin{array}{l} V(w) = \lambda \ \wedge\ P(w') < p \leq P(w)\, w \\ \vee\ V(w) = @ \ \wedge\ P(w') < p \leq P(w) \end{array} \right\}$$

$$r' := \langle r, \varepsilon\rangle \qquad lab' : V' \to \Sigma^{\lambda}_{i,j},\ \ \begin{array}{l} \langle w, P(w)\rangle \mapsto lab'(\langle w, P(w)\rangle) := lab(w) \\ \langle w,k,w',p\rangle \mapsto lab'(w,k,w',p) := \mathsf{S} \end{array}$$

and $args' : V' \to (V')^*$ is defined such that for the induced indexed successor relation $\rightarrowtail'_{(\cdot)}$ it holds:

$$w \rightarrowtail_k w_k \wedge \#\mathrm{del}(w,k) = 0 \implies \langle w, P(w)\rangle \rightarrowtail'_k \langle w_k, P(w_k)\rangle$$

$$w \rightarrowtail_0 w_0 \wedge \#\mathrm{del}(w,0) > 0 \wedge lab(w) = \lambda \wedge P(w) = P(w_0)\, v\, p \implies \langle w, P(w)\rangle \rightarrowtail'_0 \langle w, 0, w_0, P(w)\, w\rangle \wedge \langle w, 0, w_0, P(w_0) v\rangle \rightarrowtail'_0 \langle w_0, P(w_0)\rangle$$

$$w \rightarrowtail_k w_k \wedge \#\mathrm{del}(w,k) > 0 \wedge lab(w) = @ \wedge P(w) = P(w_k)\, v\, p \implies \langle w, P(w)\rangle \rightarrowtail'_k \langle w, k, w_k, P(w)\rangle \wedge \langle w, k, w_k, P(w_k) v\rangle \rightarrowtail'_0 \langle w_k, P(w_k)\rangle$$

$$w \rightarrowtail_k w_k \wedge \#\mathrm{del}(w,k) > 0 \wedge \langle w,k,w_k,p\, v\rangle, \langle w,k,w_k,p\rangle \in V' \implies \langle w,k,w_k,p\, v\rangle \rightarrowtail'_0 \langle w,k,w_k,p\rangle$$

$$w \rightarrowtail_k w_k \wedge \#\mathrm{del}(w,k) > 0 \wedge \langle w,k,w_k,p\, v\rangle \in V' \wedge j = 2 \implies \langle w,k,w_k,p\, v\rangle \rightarrowtail'_1 \langle w_k, P(w_k)\rangle$$

for all $w, w_0, w_1, v \in V$, $k \in \{0,1\}$, $p \in V^*$, *and where the function* #del *is defined as:*

$$\#\mathrm{del}(w,k) := \begin{cases} |P(w)| - |P(w')| & \text{if } w \in V(@) \wedge w \rightarrowtail_k w' \\ |P(w)| + 1 - |P(w')| & \text{if } w \in V(\lambda) \wedge w \rightarrowtail_k w' \\ 0 & \text{otherwise} \end{cases}$$

Proposition 21 *Let* $i \in \{0,1\}$ *and* $j \in \{1,2\}$. *The mapping* $\mathcal{G}_{i,j}$ *defined below is well-defined between the class of* λ*-term-graphs over* $\Sigma^{\lambda}_{i,j}$ *and the class of* λ*-ap-ho-term-graphs over* Σ^{λ}_{i}*:*

$$\begin{aligned} \mathcal{G}_{i,j} : \mathcal{T}^{(\lambda)}_{i,j} \to \mathcal{H}_i^{(\lambda)},\ G = \langle V, lab, args, r\rangle \mapsto \mathcal{G}_{i,j}(G) := \langle V', lab', args', r', P'\rangle \\ \text{where } V' := V(\lambda) \cup V(@) \cup V(0),\ lab' := lab|_{V'},\ r' := r, \\ args' : V' \to (V')^* \text{ so that for the induced indexed succ. relation } \rightarrowtail'_{(\cdot)}: \\ v_0 \rightarrowtail'_k v_1 :\Leftrightarrow v_0 \rightarrowtail_k \cdot \left(\overset{\mathsf{S}}{\rightarrowtail}_0\right)^* v_1 \quad \text{(for all } v_0, v_1 \in V',\ k \in \{0,1\}) \\ P' := P|_{V'} \text{ for the correct abstraction-prefix function } P \text{ for } G. \end{aligned}$$

Theorem 22 (correspondence between λ-ap-ho-term-graphs with λ-term-graphs) *Let* $i \in \{0,1\}$ *and* $j \in \{1,2\}$. *The mappings* $\mathcal{G}_{i,j}$ *from Prop. 21 and* $G_{i,j}$ *from Prop. 20 define a correspondence between the classes of* λ*-term-graphs over* $\Sigma^{\lambda}_{i,j}$ *and of* λ*-ap-ho-term-graphs over* Σ^{λ}_{i} *with the following properties:*

(i) $\mathcal{G}_{i,j} \circ G_{i,j} = \mathrm{id}_{\mathcal{H}_i^{(\lambda)}}$.

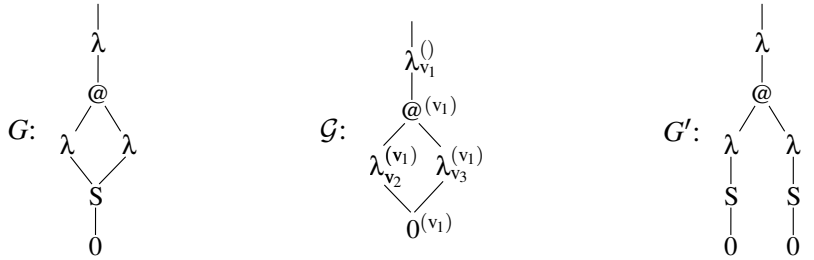
(ii) For all $G \in \mathcal{T}^{(\lambda)}_{i,j}$: $(G_{i,j} \circ \mathcal{G}_{i,j})(G) \rightrightarrows^{\mathsf{S}} G$.

(iii) $\mathcal{G}_{i,j}$ *and* $G_{i,j}$ *preserve and reflect the sharing orders on* $\mathcal{H}_i^{(\lambda)}$ *and on* $\mathcal{T}^{(\lambda)}_{i,j}$*:*

$$\begin{aligned} &(\forall \mathcal{G}_1, \mathcal{G}_2 \in \mathcal{H}_i^{(\lambda)}) && \mathcal{G}_1 \rightrightarrows \mathcal{G}_2 \iff G_{i,j}(\mathcal{G}_1) \rightrightarrows G_{i,j}(\mathcal{G}_2) \\ &(\forall G_1, G_2 \in \mathcal{T}^{(\lambda)}_{i,j}) && \mathcal{G}_{i,j}(G_1) \rightrightarrows \mathcal{G}_{i,j}(G_2) \iff G_1 \rightrightarrows G_2 \end{aligned}$$

Example 23 The λ-ap-ho-term-graphs in Fig. 2 correspond to the λ-ap-ho-term-graphs in Fig. 3 via the mappings $G_{i,j}$ and $\mathcal{G}_{i,j}$ as follows: $G_{i,j}(\mathcal{G}_0) = G'_0$, $G_{i,j}(\mathcal{G}_1) = G'_1$, $\mathcal{G}_{i,j}(G_0) = \mathcal{G}'_0$, $\mathcal{G}_{i,j}(G_1) = \mathcal{G}'_1$.

Remark 24 The correspondence in Theorem 22 is not a bijection since $\mathcal{G}_{i,j}$ is not injective. This can be seen for the following graphs (here with $i = 0$ and $j = 1$) where we have $\mathcal{G}_{0,1}(G) = \mathcal{G} = \mathcal{G}_{0,1}(G')$:



Obviously λ-ap-ho-term-graphs are not capable of reproducing the different degrees of S-sharing.

6 Not closed under bisimulation and functional bisimulation

In this section we collect all negative results concerning closedness under bisimulation and functional bisimulation for the classes of λ-term-graphs as introduced in the previous section.

Proposition 25 *None of the classes $\mathcal{T}_1^{(\lambda)}$ and $\mathcal{T}_{i,j}^{(\lambda)}$, for $i \in \{0,1\}$ and $j \in \{1,2\}$, of λ-term-graphs are closed under bisimulation.*

This proposition is an immediate consequence of the next one, which can be viewed as a refinement, because it formulates non-closedness of classes of λ-term-graphs under specializations of bisimulation, namely for functional bisimulation (under which some classes are not closed), and for converse functional bisimulation (under which none of the classes considered here is closed).

Proposition 26 *The following statements hold:*

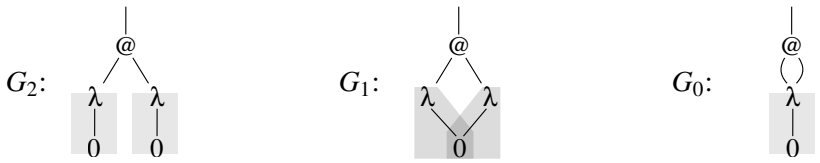
(i) None of the classes $\mathcal{T}_{\mathbf{0},j}^{(\lambda)}$ for $j \in \{1,2\}$ of λ-term-graphs are closed under functional bisimulation $\rightarrow$, or under converse functional bisimulation $\leftarrow$.

(ii) None of the classes $\mathcal{T}_{\mathbf{1}}^{(\lambda)}$ and $\mathcal{T}_{\mathbf{1},j}^{(\lambda)}$ for $j \in \{1,2\}$ of λ-term-graphs are closed under converse functional bisimulation.

(iii) The class $\mathcal{T}_{\mathbf{1},\mathbf{1}}^{(\lambda)}$ of λ-term-graphs is not closed under functional bisimulation.

(iv) The class $\mathcal{T}_{\mathbf{1},\mathbf{2}}^{(\lambda)}$ of λ-term-graphs is not closed under functional bisimulation.

Proof. For showing (i), let Δ be one of the signatures $\Sigma_{0,j}^{\lambda}$. Consider the following term graphs over Δ:



Note that G_2 represents the syntax tree of the nameless de-Bruijn-index notation $(\lambda 0)(\lambda 0)$ for the λ-term $(\lambda x.x)(\lambda x.x)$. Then it holds: $G_2 \twoheadrightarrow G_1 \twoheadrightarrow G_0$. But while G_2 and G_0 admit correct abstraction-prefix functions over Δ (nestedness of the implicitly defined scopes, here shaded), and consequently are λ-term-graphs over Δ, this is not the case for G_1 (overlapping scopes). Hence the class of λ-term-graphs over Δ is closed neither under functional bisimulation nor under converse functional bisimulation.

For showing (ii), let Δ be one of the signatures Σ_1^{λ} and $\Sigma_{1,j}^{\lambda}$. Consider the term graphs over Δ:

Then it holds: $G_1' \twoheadrightarrow G_0'$. But while G_0' admits a correct abstraction-prefix function, and therefore is a λ-term-graph, over Δ, this is not the case for G_1' (due to overlapping scopes). Hence the class of λ-term-graphs over Δ is not closed under converse functional bisimulation.

For showing (iii), consider the following term graphs over $\Sigma_{1,1}^{\lambda}$:

Then it holds that $G_1'' \trianglerighteq G_0''$. However, while G_1'' admits a correct abstraction-prefix function, and hence is a λ-term-graph over $\Sigma^{\lambda}_{1,1}$, this is not the case for G_0'' (due to overlapping scopes). Therefore the class of λ-term-graphs over $\Sigma^{\lambda}_{1,1}$ is not closed under functional bisimulation.

For showing (iv), consider the following term graphs over $\Sigma^{\lambda}_{1,2}$:

Then it holds that $G_1''' \trianglerighteq G_0'''$. However, while G_1''' admits a correct abstraction-prefix function, and hence is a λ-term-graph over $\Sigma^{\lambda}_{1,2}$, this is not the case for G_0''' (overlapping scopes). Therefore the class of λ-term-graphs over $\Sigma^{\lambda}_{1,2}$ is not closed under functional bisimulation. The scopes defined implicitly by these graphs are larger than necessary: they do not exhibit 'eager scope closure', see Section 7. □

As an easy consequence of Prop. 25, and of Prop. 26, (i) and (ii), together with the examples used in the proof, we obtain the following two propositions.

Proposition 27 *Let $i \in \{0,1\}$. None of the classes $\mathcal{H}^{\lambda}_i$ of λ-ho-term-graphs, or $\mathcal{H}_i{}^{(\lambda)}$ of λ-ap-ho-term-graphs are closed under bisimulations on the underlying term graphs.*

Proposition 28 *The following statements hold:*

(i) Neither the class $\mathcal{H}^{\lambda}_0$ nor the class $\mathcal{H}_0{}^{(\lambda)}$ is closed under functional bisimulations, or under converse functional bisimulations, on the underlying term graphs.

(ii) Neither the class $\mathcal{H}^{\lambda}_1$ of λ-ho-term-graphs nor the class $\mathcal{H}_1{}^{(\lambda)}$ of λ-ap-ho-term-graphs is closed under converse functional bisimulations on underlying term graphs.

Note that Prop. 28, (i) is a strengthening of the statement of Prop. 15 earlier.

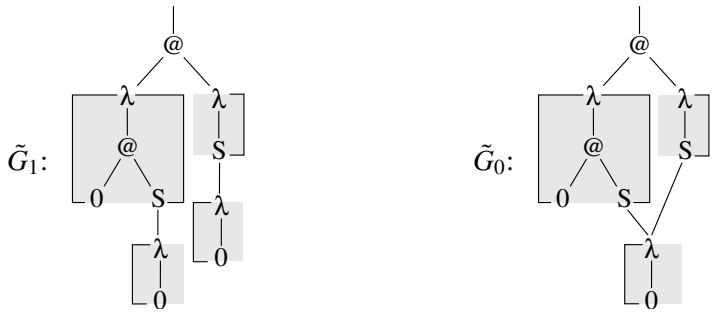
7 Closed under functional bisimulation

The negative results gathered in the last section might seem to show our enterprise in a quite poor state: For the classes of λ-term-graphs we introduced, Prop. 26 only leaves open the possibility that the class $\mathcal{T}_1{}^{(\lambda)}$ is closed under functional bisimulation. Actually, $\mathcal{T}_1{}^{(\lambda)}$ is closed (we do not prove this here), but that does not help us any further, because the correspondences in Thm. 22 do not apply to this class, and worse still, Prop. 16 rules out simple correspondences for $\mathcal{T}_1{}^{(\lambda)}$. So in this case we are left without the satisfying correspondences to λ-ho-term-graphs and λ-ap-ho-term-graphs that yet exist for the other classes of λ-term-graphs, but which in their turn are not closed under functional bisimulation.

But in this section we establish that the class $\mathcal{T}_{1,2}{}^{(\lambda)}$ is very useful after all: its restriction to term graphs with eager application of scope closure is in fact closed under functional bisimulation.

The reason for the non-closedness of $\mathcal{T}_{1,2}{}^{(\lambda)}$ under functional bisimulation consists in the fact that λ-term-graphs over $\Sigma^{\lambda}_{1,2}$ do not necessarily exhibit 'eager scope closure': for example in the term graph G_1''' from the proof of Prop. 26, (iv), the scopes of the two topmost abstractions are not closed on the paths to variable occurrences belonging to the bottommost abstractions. For the following variation $\tilde{G}_1$

of G_1 with eager scope closure the problem disappears:



Its bisimulation collapse $\tilde{G}_0$ has again a correct abstraction-prefix function and hence is a λ-term-graph.

Definition 29 (eager-scope, and fully back-linked, λ-term-graphs) Let $G = \langle V, lab, args, r\rangle$ be a λ-term-graph over $\Sigma^{\lambda}_{1,j}$ for $j \in \{1,2\}$ with abstraction-prefix function $P : V \to V^*$. We call G an *eager-scope λ-term-graph (over $\Sigma^{\lambda}_{1,j}$)* if it holds:

$$\begin{aligned}\forall w, v \in V.\ \forall p \in V^*.\ P(w) = pv \ \Rightarrow\ \exists n \in \mathbb{N}\, \exists w_1, \ldots, w_{n-1} \in V.\ & \\ w \rightarrowtail w_1 \rightarrowtail \ldots \rightarrowtail w_{n-1} \rightarrowtail_0 v & \\ \wedge\ w_{n-1} \in V(0) \ \wedge\ \forall 1 \le i \le n-1.\ P(w) \le P(w_i)\ , &\end{aligned}$$

that is, if for every vertex w in G with a non-empty abstraction-prefix $P(w)$ that ends with v there exists a path from w to v in G via vertices with abstraction-prefixes that extend $P(w)$ and finally a variable-occurrence vertex before reaching v. By ${}^{\mathrm{eag}}\mathcal{T}^{(\lambda)}_{i,j}$ we denote the subclass of $\mathcal{T}^{(\lambda)}_{i,j}$ consisting of all eager-scope-λ-term-graphs. And we say that G is *fully back-linked* if it holds:

$$\forall w, v \in V_1.\ \forall p \in V_1^*.\ P(w) = pv \ \Rightarrow\ w \rightarrowtail^* v\ , \tag{6}$$

that is, if for all vertices w of G_1, the last vertex v in the abstraction-prefix of w is reachable from v. Note that eager-scope implies fully back-linkedness for λ-term-graphs.

Lemma 30 *Let G be a fully back-linked λ-term-graph in $\mathcal{T}^{(\lambda)}_{1,2}$ with vertex set V, and let P be its abstraction-prefix function. Let G' be a term graph over $\Sigma^{\lambda}_{1,2}$ (thus in $\mathcal{T}_{1,2}$) such that $G \trianglerighteq_h G'$. Then it holds:*

$$\forall v_1, v_2 \in V.\ h(v_1) = h(v_2) \ \Rightarrow\ \bar{h}(P(v_1)) = \bar{h}(P(v_2)) \tag{7}$$

where $\bar{h}$ is the homomorphic extension of h to words over V.

Proof (Idea). Let G_1, G_2 be as assumed in the lemma, and let h be a homomorphism that witnesses $G_1 \trianglerighteq_h G_2$. We will use the following distance parameter for vertices of G_1: Let, for all $w \in V_1$, $d_{\lambda,P}(w)$ be either 0 if $P(w)$ is empty, or otherwise the minimum length of a path in G_1 from w to the last vertex in the abstraction-prefix $P(w)$. Thus due to (6), $d_{\lambda,P}(w) \in \mathbb{N}$ for all vertices w of G_1. Now (7) can be proved by induction on $\max\{d_{\lambda,P}(v_1), d_{\lambda,P}(v_2)\}$ with a subinduction on $\max\{|P(v_1)|, |P(v_2)|\}$. □

This lemma is the crucial stepping stone for the proof of the following theorem.

Theorem 31 (preservation of λ-term-graphs over $\Sigma^{\lambda}_{1,2}$ under homomorphism) *Let G and G' be term graphs over $\Sigma^{\lambda}_{1,2}$ such that G is a λ-term-graph in $\mathcal{T}^{(\lambda)}_{1,2}$, and $G \trianglerighteq_h G'$ holds for a homomorphism h.*

If G is fully back-linked, then also G' is a λ-term-graph in $\mathcal{T}^{(\lambda)}_{1,2}$, which is fully back-linked. If, in addition, G is an eager-scope λ-term-graph, then so is G'.

Corollary 32 *The subclass ${}^{\mathrm{eag}}\mathcal{T}_{1,2}^{(\lambda)}$ of the class $\mathcal{T}_{1,2}$ that consists of all eager-scope λ-term-graphs in $\mathcal{T}_{1,2}^{(\lambda)}$ is closed under functional bisimulation.*

Since the counterexample in the proof of Prop. 26, (iii) used eager-scope λ-term-graphs, it rules out a statement analogous to Cor. 32 for the class $\mathcal{T}_{1,1}^{(\lambda)}$. Such a statement for $\mathcal{T}_{0,1}^{(\lambda)}$ and $\mathcal{T}_{0,2}^{(\lambda)}$ is ruled out similarly, with respect to an appropriate definition of 'eager-scope' for λ-term-graphs over $\Sigma_{0,1}^{\lambda}$ and $\Sigma_{0,2}^{\lambda}$.

Corollary 33 *Let h be a functional bisimulation from an eager-scope λ-term-graph G over $\Sigma_{1,2}^{\lambda}$ to a term graph G' over $\Sigma_{1,2}^{\lambda}$ (h witnesses $G \rightrightarrows_h G'$). Then G' is an eager-scope λ-term-graph as well, and h extends to a functional bisimulation from $\mathcal{G}_{1,2}(G)$ to $\mathcal{G}_{1,2}(G')$ (thus h also witnesses $\mathcal{G}_{1,2}(G) \rightrightarrows_h \mathcal{G}_{1,2}(G')$).*

8 Conclusion

We first defined higher-order term graph representations for cyclic λ-terms:

- λ-ho-term-graphs in $\mathcal{H}_i^{\lambda}$, an adaptation of Blom's 'higher-order term graphs' [4], which possess a scope function that maps every abstraction vertex v to the set of vertices that are in the scope of v.
- λ-ap-ho-term-graphs in $\mathcal{H}_i^{(\lambda)}$, which instead of a scope function carry an abstraction-prefix function that assigns to every vertex w information about the scoping structure relevant for w. Abstraction prefixes are closely related to the notion of 'generated subterms' for λ-terms [6]. The correctness conditions here are simpler and more intuitive than for λ-ho-term-graphs.

These classes are defined for $i \in \{0,1\}$, according to whether variable occurrences have back-links to abstractions (for $i = 1$) or not (for $i = 0$). Our main statements about these classes are:

- a bijective correspondence between $\mathcal{H}_i^{\lambda}$ and $\mathcal{H}_i^{(\lambda)}$ via mappings A_i and B_i that preserve and reflect the sharing order (Thm. 10);
- the naive approach to implementing homomorphisms on theses classes (ignoring all scoping information and using only the underlying first-order term graphs) fails (Prop. 16).

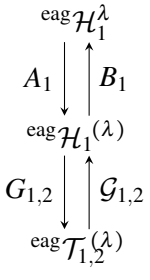
The latter was the motivation to consider first-order term graph implementations with scope delimiters:

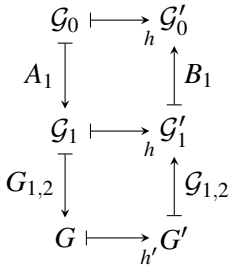
- λ-term-graphs in $\mathcal{T}_{i,j}^{(\lambda)}$ (with $i \in \{0,1\}$ and $j = 2$ or $j = 1$ for scope delimiter vertices with or without back-links, respectively), which are first-order term graphs without a higher-order concept, but for which correctness conditions are formulated via the existence of an abstraction-prefix function.

The most important results linking these classes with λ-ap-ho-term-graphs are:

- an 'almost bijective' correspondence between the classes $\mathcal{H}_i^{(\lambda)}$ and $\mathcal{T}_{i,j}^{(\lambda)}$ via mappings $G_{i,j}$ and $\mathcal{G}_{i,j}$ that preserve and reflect the sharing order (Thm. 22);
- the subclass ${}^{\mathrm{eag}}\mathcal{T}_{1,2}^{(\lambda)}$ of eager-scope λ-term-graphs in $\mathcal{T}_{1,2}^{(\lambda)}$ is closed under homomorphism (Cor. 32).

The correspondences together with the closedness result allow us to derive methods to handle homomorphisms between eager higher-order term graphs in $\mathcal{H}_1^{\lambda}$ and $\mathcal{H}_1{}^{(\lambda)}$ in a straightforward manner by implementing them via homomorphisms between first-order term graphs in $\mathcal{T}_{1,2}^{(\lambda)}$.





For example, the property that a unique maximally shared form exists for λ-term-graphs in $\mathcal{T}_{1,2}^{(\lambda)}$ (which

can be computed as the bisimulation collapse that is guaranteed to exist for first-order term graphs) can now be transferred to eager-scope λ-ap-ho-term-graphs and λ-ho-term-graphs via the correspondence mappings (see the diagram above). For this to hold it is crucial that ${}^{\mathrm{eag}}\mathcal{T}_{1,2}^{(\lambda)}$ is closed under homomorphism, and that the correspondence mappings preserve and reflect the sharing order. The maximally shared form $\max_{\mathrm{eag}}\mathcal{H}_1^{\lambda}(\mathcal{G})$ of an eager λ-ho-term-graph $\mathcal{G}$ can furthermore be computed as:

$$\max_{\mathrm{eag}}\mathcal{H}_1^{\lambda}(\mathcal{G}) = (B_1 \circ \mathcal{G}_{1,2} \circ \max_{\mathrm{eag}}\mathcal{T}_{1,2}^{(\lambda)} \circ G_{1,2} \circ A_1)(\mathcal{G}).$$

where $\max_{\mathrm{eag}}\mathcal{T}_{1,2}^{(\lambda)}$ maps every λ-term-graph in $\mathcal{T}_{1,2}^{(\lambda)}$ to its bisimulation collapse. For obtaining $\max_{\mathrm{eag}}\mathcal{T}_{1,2}^{(\lambda)}$ fast algorithms for computing the bisimulation collapse of first-order term graphs can be utilized.

While we have explained this result here only for term graphs with eager scope-closure, the approach can be generalized to non-eager-scope term graphs. To this end scope delimiters have to be placed also underneath variable vertices. Then variable occurrences do not implicitly close all open extended scopes, but every extended scope that is open at some position must be closed explicitly by scope delimiters on all (maximal) paths from that position. The resulting graphs are fully back-linked, and then Thm. 31 guarantees that the arising class of λ-term-graphs is again closed under homomorphism.

For our original intent of getting a grip on maximal subterm sharing in the λ-calculus with letrec or μ, however, only eager scope-closure is practically relevant, since it facilitates a higher degree of sharing.

Ultimately we expect that these results allow us to develop solid formalizations and methods for subterm sharing in higher order languages with sharing constructs.

Acknowledgement. We want to thank the reviewers for their helpful comments, and for pointing out a number of inaccurate details in the submission that we have remedied for obtaining this version.

References

[1] Zena M. Ariola & Stefan Blom (1997): *Cyclic Lambda Calculi*. In Martin Abadi & Takayasu Ito, editors: *Proceedings of TACS'97, Sendai, Japan, September 23–26, 1997. LNCS* 1281, Springer Berlin / Heidelberg, pp. 77–106, doi:10.1007/BFb0014548.

[2] Zena M. Ariola & Jan Willem Klop (1994): *Cyclic Lambda Graph Rewriting*. In: *Proceedings of the Symposium on Logic in Computer Science (LICS) 1994*. pp. 416 –425, doi:10.1109/LICS.1994.316066.

[3] Zena M. Ariola & Jan Willem Klop (1996): *Equational Term Graph Rewriting*. *Fundamenta Informaticae* 26(3), pp. 207–240, doi:10.3233/FI-1996-263401.

[4] Stefan Blom (2001): *Term Graph Rewriting, Syntax and Semantics*. Ph.D. thesis, Vrije Universiteit Amsterdam.

[5] N. G. de Bruijn (1972): *Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem*. *Indagationes Mathematicae* 34, pp. 381–392, doi:10.1016/1385-7258(72)90034-0.

[6] Clemens Grabmayer & Jan Rochel (2012): *Expressibility in the Lambda-Calculus with Letrec*. Technical Report arXiv:1208.2383, `http://arxiv.org`. `http://arxiv.org/abs/1208.2383`.

[7] Dimitri Hendriks & Vincent van Oostrom (2003): ß. In F. Baader, editor: *Proceedings CADE-19. Lecture Notes in Artificial Intelligence* 2741, Springer–Verlag, pp. 136–150.

[8] Simon Peyton Jones (1987): *The Implementation of Functional Programming Languages*. Prentice-Hall, Inc.

[9] Vincent van Oostrom, Kees-Jan van de Looij & Marijn Zwitserlood (2004): *Lambdascope*. Extended Abstract for the Workshop on Algebra and Logic on Programming Systems (ALPS), Kyoto, April 10th 2004. `http://www.phil.uu.nl/~oostrom/publication/pdf/lambdascope.pdf`.

[10] Terese (2003): *Term Rewriting Systems*. *Cambridge Tracts in Theoretical Computer Science* 55, Cambridge University Press.

Bigraphical Nets

Maribel Fernández Ian Mackie
Matthew Walker
École Normale Supérieure, Paris, France
École Polytechnique, Palaiseau, France
King's College London, Dept. of Informatics, London WC2R 2LS, UK

Interaction nets are a graphical model of computation, which has been used to define efficient evaluators for functional calculi, and specifically λ-calculi with patterns. However, the flat structure of interaction nets forces pattern matching and functional behaviour to be encoded at the same level, losing some potential parallelism. In this paper, we introduce bigraphical nets, or binets for short, as a generalisation of interaction nets using ideas from bigraphs and port graphs, and we present a formal notation and operational semantics for binets. We illustrate their expressive power by examples of applications.

Keywords: Interaction Net, Port Graph, Bigraph, Rewriting Calculus.

1 Introduction

Interaction nets [15] are graphical rewrite systems used for the specification of logical proof systems (e.g., [1, 16]), for the implementation of efficient evaluators for the λ-calculus (e.g., [12, 4, 18]), and for visual programming (e.g., [13, 20, 19]).

The visual nature of interaction nets makes them well suited as a specification tool, and, since *all* the computation steps are explicit and expressed in the same formalism (there is no external machinery), interaction nets are also well suited for the study of the dynamics of programming languages and rewriting systems [9, 7, 22]. However, interaction nets have some drawbacks. When the nets are large or growing during reduction, being able to *structure* the graph is crucial to understand the system modelled, but interaction nets lack mechanisms to structure the system. Moreover, to formally prove properties of the system modelled or implement reduction, a *formal, algebraic notation*, with a precise operational semantics, should also be available. In this paper, we address these two points:

- First, inspired by Milner's bigraphs [21], we define a generalisation of interaction nets, which we call bigraphical nets, or simply *binets*, where not only the connectivity but also the hierarchical structure of the system is taken into account. Binets borrow from bigraphs a notion of locality that is missing in interaction nets.
- Then, we present a formal algebraic notation for binets, with an operational semantics which can serve as a basis for their implementation.

Related Work. Binets can be seen as hierarchical graph rewriting systems that permit links between nested nets and external subgraphs (like bigraphs, and unlike hierarchical graphs [6]). Rewriting can take place across boundaries. Both of these features will be of use in our encoding of the ρ-calculus.

Binets inherit from interaction nets the notion of principal port. But, in contrast with interaction nets, binets do not force all interactions to be binary, and in contrast with bigraphs, they place restrictions on reactions to simplify the implementation of rule application.

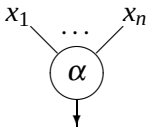
R. Echahed and D. Plump (Eds.): 7th International Workshop on Computing with Terms and Graphs
EPTCS 110, 2013, pp. 74–81, doi:10.4204/EPTCS.110.8

Interaction nets have been used as an implementation language for functional calculi, and as a tool to understand their dynamics [12, 4, 17, 18, 8, 7, 11]. Interaction net encodings of the ρ-calculus [5], an extension of the λ-calculus where we can abstract on patterns, not just on variables, shed light on the implicit parallelism present in the ρ-calculus, and at the same time, motivate a generalisation of interaction nets, as shown in [10]. In this paper, we develop and formalise this idea. Our main contribution is a formal syntax and operational semantics for binets, via a textual calculus.

The class of portgraphs defined by Andrei and Kirchner [3] can also be seen as a generalisation of interaction nets, but although binets are graphs with ports, due to their hierarchical nature they cannot be defined as portgraphs. It would be interesting to consider a generalisation of portgraphs with a notion of locality; the inclusion of this feature in PORGY [2] could serve as a starting point for the development of a specification environment based on binets.

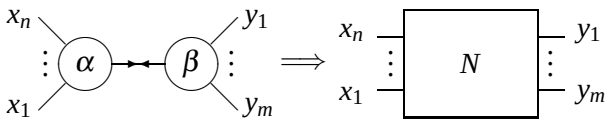
2 Background

Interaction Nets. A system of interaction nets is specified by a set Σ of symbols with fixed arities, and a set $\mathscr{R}$ of interaction rules. An occurrence of a symbol $\alpha \in \Sigma$ is called an *agent*. If the arity of α is n, then the agent has $n+1$ *ports*: a *principal port* depicted by an arrow, and n *auxiliary ports*. Such an agent will be drawn in the following way:



A net N is a graph (not necessarily connected) with agents at the vertices and each edge connecting at most 2 ports. The ports that are not connected to another agent are *free*. There are two special instances of a net: an empty net, and a net consisting only of edges (no agents). The *interface* of a net is the set of free ports of agents and free extremes of wires. We refer to [15] for more details.

An interaction rule $((\alpha,\beta) \Longrightarrow N) \in \mathscr{R}$ replaces a pair of agents $(\alpha,\beta) \in \Sigma \times \Sigma$ connected together on their principal ports (an *active pair* or *redex* and written $\alpha \bowtie \beta$) by a net N with the same interface. Rules must satisfy two conditions: all free ports are preserved during reduction (there are no global operations: only the part of the net involved in the rewrite is modified), and there is at most one rule for each pair of agents (such a rule will thus be sometimes denoted by $\alpha \bowtie \beta$). The following diagram shows the format of interaction rules (N can be any net built from Σ).



We use the notation $\Longrightarrow$ for the one-step reduction relation, or $\underset{\alpha\bowtie\beta}{\Longrightarrow}$ if we want to be explicit about the rule used, and $\Longrightarrow^*$ for its transitive and reflexive closure. If a net does not contain any active pairs then we say that it is in normal form. The key property of interaction nets is that reduction is strongly confluent. We refer the reader to [15] for more details and examples.

Bigraphs. In [21, 14] a notion of graph transformation system is defined, using nested (or hierarchical) graphs called *bigraphs*. Bigraphs represent two kinds of structure: locality (nodes may occur inside

other nodes) and connectivity (nodes have ports that may be connected by links). We recall the basic terminology of bigraphs and refer the reader to [21] for details and examples.

A bigraph is a pair of a *place graph* and a *link graph* over the same set of nodes. It has interfaces, which define the way in which it can be composed with other bigraphs. The place graph, or placing, is a set of trees with interfaces, and the link graph, or linking, is a hypergraph with interfaces. A placing has inner and outer interfaces. The inner interface corresponds to the *sites* where other graphs can be placed, and the outer interface corresponds to the *roots* of the trees. The linking also has inner and outer interfaces, which are names of ports, that is, the points where edges can be attached.

Nodes are labelled by *controls* with fixed arities; the arity of a control corresponds to the number of ports of the node. A control is *atomic* if it cannot contain a nested graph, otherwise it is non-atomic.

The reduction relation is defined by a set of reaction rules, which are pairs of bigraphs (called redex and reactum). The redex has a *width*, corresponding to the number of sites it occupies in the outer bigraph [14]. A non-atomic control K can be specified as active, in which case reactions can occur inside, or passive, in which case reactions in the internal bigraph can only occur after the control K has been destroyed.

Interaction nets can be seen as a particular kind of bigraphs without nesting: all controls (called agents in interaction nets) are atomic, and have a distinguished port (the principal port). Interaction rules can be seen as reactions in which both redex and reactum have width 1, and redexes are restricted to just two controls connected by one link through the distinguished ports.

3 Binets

3.1 Informal presentation

Bigraphs [14] introduce a notion of locality (using nesting to indicate that a graph is local to a certain node) which is missing in interaction nets. In this section, we define binets as a generalisation of interaction nets to incorporate this feature. We start with an informal definition of binets, contrasting them with interaction nets, before presenting a formal syntax and semantics for them.

A binet is a labelled graph consisting of a set of nodes (also called *agents*) and a set of edges, which are attached to nodes at connection points called *ports*. Each edge connects at most two ports. The label of a node (i.e., the agent's name) determines its arity, that is, the number of ports it has. Each agent has a distinguished port, called the *principal* port, and a (possibly empty) set of *auxiliary* ports. An agent can be located inside another agent, and edges can connect ports of agents situated at different nesting levels (i.e., edges can cross node boundaries).

Interaction rules, also called reaction rules, define interactions between two agents connected by their principal ports, or interactions of an agent with its local subnets, preserving the interfaces.

Figure 1 shows a binet representing a ρ-term. Ovals and circles represent agents, their names are written inside; principal ports are marked with an arrow, the free port at the top is marked by a dangling edge. The ε-agent is drawn outside the $\rightarrow$-agent to exploit a non-strict semantics as early in the reduction process as possible.

In contrast with interaction nets, the left-hand side of a reaction rule can specify the location in which the reacting agents are, or the locations contained in these controls, and reactions can take place across boundaries. However, reduction is still local in the sense that it only affects the nodes that match the left hand side (no global conditions or updates are specified). The latter point is relevant for implementation.

Agents in binets correspond to the notion of control in bigraphs, and binet reaction rules are a particular class of bigraph reaction rules. Each binet has an associated place graph and link graph, similarly to

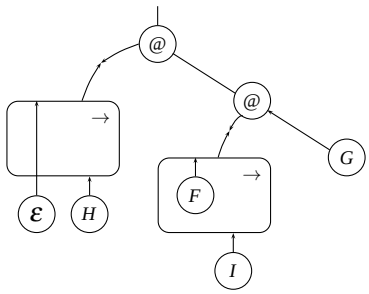


Figure 1: Binet for the ρ-term $(x \to H)((F \to I)G)$.

bigraphs. All the examples of bigraphs for the π-calculus and ambient calculus given in [14] (part I) can be recast as binets by adding principal ports and copy/erase agents (controls) to preserve the interface of the reactions.

Comparing with the properties of interaction nets, we remark that confluence does not hold in general for binets, because of the possibility of interactions across boundaries. To study the formal properties of binet reduction, below we give a calculus for binets.

3.2 A calculus for binets

As Milner [21] stated: "Diagrams are valuable for rapid appreciation of a system's structure. On the other hand, algebra is essential to express [...] the ways in which a system may be resolved into components." In this section we give a formal, algebraic presentation for binets. First we give the syntax of the language, and then we present an operational semantics for programs written in this language.

Syntax. A textual syntax for binets has to capture, dually, the connections between agents (including where those connections are principal ports), differentiating between internal or external with respect to the originating agent and also the locality of agents within a system, i.e., their physical position within other agents. It is the intention of this syntax to unambiguously state these three properties without over-complication.

We define below agents and binets over a *signature* $\mathscr{A}, L$, where $\mathscr{A}$ is a set of *agent names*, each with an associated arity (n,m) corresponding to the number of ports in its internal and external interface, respectively, and L is a set of port labels. We assume $L \cap \mathscr{A} = \emptyset$.

Definition 3.1 (Agent) *An* agent *over the signature $\mathscr{A}$, L is written $A^l\langle E \,|\, I \,|\, N\rangle$, where $A \in \mathscr{A}$ is the agent name, which determines its arity (n,m), $l \in L$ is the label given to the principal port of A, and the lists I, E of lengths n, m respectively, whose elements are port labels in L, denote the internal and external agent interfaces; the order denotes the geographic position of the ports (in similar fashion to interaction nets reading in a clockwise direction from the principal port for external ports and, without loss of precision, in an arbitrary clockwise direction for interior ports). N is a (possibly empty) list corresponding to the set of agents located within the agent.*

The definition above is inductive (due to the inclusion of the set N within A) but not recursive; agents are not permitted to be located within themselves.

Definition 3.2 (Binet) *A binet over the signature $\mathscr{A}$, L is defined as a set of agents and wires over $\mathscr{A}$, L. Agents have already been defined. A* wire *is an edge joining two ports, written $a-b$ where a, b are the*

labels of the ports. Each port label in L occurs at most twice in a binet; the net interface *of the binet is defined as the subset of labels occurring only once.*

The binet containing no agents and wires is a special case. For brevity, agents of the form $A^i\langle X\,|\,\emptyset\,|\,\emptyset\rangle$ will be denoted $A^i\langle X\rangle$ and the particular case when $X=\emptyset$ will be written $A^i\langle\rangle$.

A binet, similar to a bigraph, can be decomposed into a place graph and a link graph. The link graph is explicit in the definition of binet (a binet is a set of agents and wires); the place graph can be reconstructed from the nesting of agents.

Reduction in binets occurs on *active pairs*, which are pairs of agents connected via their principal ports, similarly to interaction nets although significantly rules in binets must be aware of locality context. More precisely, a rule may affect the agents in the place graph of the active pair. We illustrate it with an example: the interaction rule presented in [10] between the matching agent, M and any other agent α is one such occurrence and would be written:

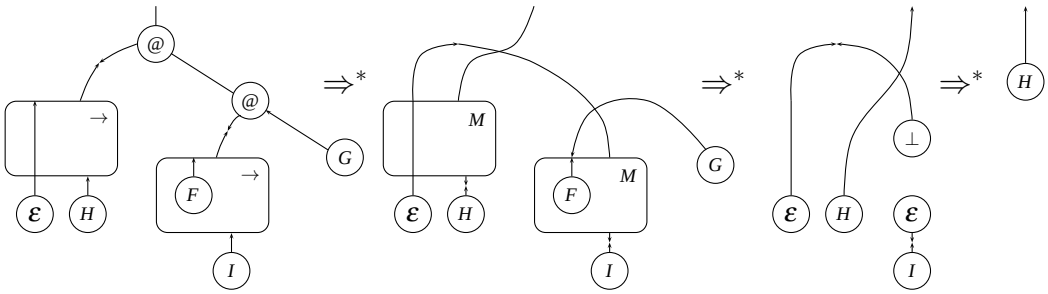
$$M^a\langle b\,|\,\emptyset\,|\,X\rangle\,,\,\alpha^a\langle Y\rangle \Rightarrow \alpha^b_M\langle Y\,|\,\emptyset\,|\,X\rangle$$

The metavariables X and Y denote a subnet and series of labels respectively that remain unchanged under graph reduction. Here, the interaction between the agents M and α causes the subgraph located within the agent M (denoted by X) to move to the new agent named α_M with principal port b.

Contrary to interaction nets, binets permit reductions to occur on certain graph configurations despite the nonexistence of an active pair, called *inactive* rewriting in the sequel. See for example the configuration in [10] of an empty matching agent, M, where the net is rewritten to a wire without interaction through active pairs. This is written as follows (the arrow explicitly shows the type of rule being applied):

$$M^a\langle b\rangle \Longrightarrow_{inactive} a-b$$

The reduction calculus is defined below, but first we give an example: a reduction sequence for the binet shown in the previous subsection, representing the ρ-term $(x\to H)((F\to I)G)$. The textual representation of each binet is shown in the table below.



First Binet	Second Binet	Third Binet	Fully Reduced
$\to^x\langle a\,\vert\,b\,\vert\,\emptyset\rangle$	$M^a\langle c\rangle$	$H^c\langle\rangle$	$H^c\langle\rangle$
$\varepsilon^b\langle\rangle$	$H^a\langle\rangle$		
$H^a\langle\rangle$		$\bot^d\langle\rangle$	
$@^x\langle c,d\rangle$	$\varepsilon^d\langle\rangle$	$\varepsilon^d\langle\rangle$	
$@^y\langle d,e\rangle$	$M^f\langle d\,\vert\,\emptyset\,\vert\,F^e\langle\rangle\rangle$		
$\to^y\langle f\,\vert\,g\,\vert\,F^g\langle\rangle\rangle$	$I^f\langle\rangle$	$I^f\langle\rangle$	
$I^f\langle\rangle$	$G^e\langle\rangle$	$\varepsilon^f\langle\rangle$	
$G^e\langle\rangle$			

The second binet contains three active pairs but parallel firing of the rule for agents M^f and I^f with the rule for F^e and G^e would clearly be incorrect; the general rule for M with α involves a rewriting that affects all of the nested nets within M, hence a reduction strategy is required. Informally, there is a choice to delay either active pair until the other has had the opportunity to reduce (although in this instance either derivation will eventually lead to the intended destruction of this disjoint net). A strategy is also required in the same configuration to either fire the rule for M^a and H^a or, as in the example, to perform an inactive rewrite on M^a and reduce it to the wire $a - c$ (see the second textual rule above).

Reduction Rules. Rewrite rules follow the simplicity of interaction nets when the reductum has no rewrite implications for any agent except for the agent (in the case of inactive rewriting) or agents (active pair rewriting) directly involved. However, due to the more expressive graph rewriting allowed by binets, rewrite rules require additional machinery to resolve rewriting of nested agents within the place graph of the net. Unlike interaction net rules they incorporate metavariables and an additional strategy language.

A priori knowledge of how a rule may affect the surrounding subnet is essential when the rule is defined. For example, the ε rule within the ρ-calculus scheme is the garbage collection agent responsible for deleting nets. This agent propagates through subnets, terminating when it forms an active pair with another ε-agent, hence when defining the interaction of ε and M, the subnets within M should also be reduced, as follows:

$$\varepsilon^a\langle\rangle, M^a\langle b \,|\, \emptyset \,|\, X\rangle \Longrightarrow \varepsilon^b\langle\rangle, X, \text{ foreach } x \text{ in } I(X)\text{: } \varepsilon^x\langle\rangle \text{ and } \varepsilon^{\bar{x}}\langle\rangle$$

where $I(X)$ is the collection of labelled ports that constitute the interface of the nested net X and $\bar{x}$ is a fresh label for the port outside of M that was connected to the interface at x.

The ability of binets to rewrite over agent boundaries means the efficiency (measured in the number of interactions: typical of interaction nets although cruder for binets) can be improved by rephrasing the rule to propagate ε only over the wires that are free in this subnet (i.e., those wires that extend beyond the locality of the M-agent). Each of these wires can be identified by a label appearing only once in the subnet X and so the above rule can be reinterpreted as follows, where X is removed in one step:

$$\varepsilon^a\langle\rangle, M^a\langle b \,|\, \emptyset \,|\, X\rangle \Longrightarrow \varepsilon^b\langle\rangle, \text{ foreach } x \text{ in } L_X \text{ where } x \text{ is unique: } \varepsilon^x\langle\rangle$$

where L_X is the multiset of labelled ports in X.

The strategy language is left informal at this stage with full details to be provided in later technical reports.

Reduction Calculus. The reduction calculus comprises four main parts: firstly populating a set of the active pairs within a binet and also those (sub)nets that are configured in a way that permits inactive rewriting. This collection of active pairs and nets is then prioritised according to a given reduction strategy and, crucially, a collection of agents and nets that can safely be rewritten in parallel is identified. Then, both active and inactive rewriting is performed and, lastly, a tidying stage is performed to eliminate every explicitly written wire in the net.

Collect Let $\mathscr{C}_A$ be the set of labels of principal ports involved in active pairs (these are easily computed: scan the graph and identify each label l that appears twice as a principal port) and $\mathscr{C}_I$ be the set of binets that are isomorphic to the left hand side of inactive rewriting rules (computed using standard subgraph matching algorithms that can be optimised due to occurrence of principal ports).

Prioritise According to the reduction strategy implemented (weighted, stochastic, typed and so on) group all *safe* nets, $\mathscr{C}_s$. A collection of safe nets is one where rewriting (either active or inactive) can occur in parallel without conflict. The safety, or otherwise, of potential net rewriting is inferred by the rules: any rule of the form $\alpha^m\langle -|-|X\rangle,\ldots \Longrightarrow \beta^n\langle -|-|X'\rangle$ where $X \neq X'$ cannot safely be rewritten at the same time (or in the same *pass*) as any rule that rewrites within X.

Rewrite For each active pair and agent within $\mathscr{C}_s$, apply rule. For simple rules where rewriting does not occur within agent borders and there are no internal edges:

$$\begin{aligned}\alpha^x\langle u_1,\ldots,u_m\rangle,\beta^x\langle v_1,\ldots,v_m\rangle \Longrightarrow & \Gamma_1^{w_1}\langle w_2,\ldots,w_p\rangle,\ldots,\\ & \Gamma_q^{w_r}\langle w_{r+1},\ldots,w_s\rangle,\\ & u_1-w_i,\ldots,u_m-w_{i'},\\ & v_1-w_{i''},\ldots,v_n-w_{i'''}\end{aligned}$$

where $\Gamma^{\mathbf{w}}$ are the (possibly empty) agents that are produced on rewriting and $\mathbf{w}$ are the intermediary labels given to the wires of the produced net. Note that the size of $\mathbf{w}$ is potentially larger than the number of ports to the left hand side of the rule. The cases for rules whose agents have internal ports and rewriting occurs across borders incorporates a richer programmatic syntax and the resulting operational calculus is more complex.

Tidy If w is a label within Γ and there exists $u-w$ then substitute w by u within Γ ($\Gamma[w/u]$).

4 Conclusion

We have presented a new visual language generalising interaction nets to incorporate features from bigraphs. Domains of application include concurrent and reactive systems. Not only can binets model these systems both graphically and textually, but they are also directly implementable. We are currently working on the implementation of an abstract machine for binets, inspired by the interaction net machines defined in [22].

References

[1] Sandra Alves, Maribel Fernández & Ian Mackie (2011): *A new graphical calculus of proofs.* In Echahed, editor: *Proceedings of TERMGRAPH 2011*, EPTCS, pp. 69–84. Available at `http://dx.doi.org/10.4204/EPTCS.48.8`.

[2] Oana Andrei, Maribel Fernández, Hélène Kirchner, Guy Melançon, Olivier Namet & Bruno Pinaud (2011): *PORGY: Strategy-Driven Interactive Transformation of Graphs.* In Echahed, editor: *Proceedings of TERMGRAPH 2011*, EPTCS, pp. 54–68. Available at `http://dx.doi.org/10.4204/EPTCS.48.7`.

[3] Oana Andrei & Hélène Kirchner (2008): *A Rewriting Calculus for Multigraphs with Ports. Electr. Notes Theor. Comput. Sci.* 219, pp. 67–82. Available at `http://dx.doi.org/10.1016/j.entcs.2008.10.035`.

[4] Andrea Asperti, Cecilia Giovannetti & Andrea Naletto (1996): *The Bologna Optimal Higher-order Machine*. *Journal of Functional Programming* 6(6), pp. 763–810. Available at `http://dx.doi.org/10.1017/S0956796800001994`.

[5] Horatiu Cirstea & Claude Kirchner (2001): *The rewriting calculus — Part I* and *II*. *Logic Journal of the Interest Group in Pure and Applied Logics* 9(3), pp. 427–498. Available at `http://dx.doi.org/10.1093/jigpal/9.3.339`.

[6] Frank Drewes, Berthold Hoffmann & Detlef Plump (2000): *Hierarchical Graph Transformation*. In Tiuryn, editor: *Proc. Foundations of Software Science and Computation Structures (FOSSACS 2000), Lecture Notes in Computer Science* 1784, pp. 98–113. Available at http://dx.doi.org/10.1007/3-540-46432-8_7.

[7] M. Fernández & L. Khalil (2003): *Interaction nets with McCarthy's amb: Properties and Applications*. *Nordic Journal of Computing* 10(2), pp. 134–162. Available at http://dx.doi.org/10.1016/S1571-0661(05)80363-9.

[8] M. Fernández & I. Mackie (1996): *From Term Rewriting to Generalised Interaction Nets*. In: *Proceedings of PLILP'96. Programming Languages: Implementations, Logics, and Programs*, *Lecture Notes in Computer Science* 1140, Springer-Verlag. Available at http://dx.doi.org/10.1007/3-540-61756-6_94.

[9] Maribel Fernández & Ian Mackie (1998): *Interaction Nets and Term Rewriting Systems*. *Theoretical Computer Science* 190(1), pp. 3–39. Available at http://dx.doi.org/10.1016/S0304-3975(97)00082-0.

[10] Maribel Fernández, Ian Mackie & François Régis Sinot (2006): *Interaction Nets vs. the rho-calculus: Introducing Bigraphical Nets*. *Electr. Notes Theor. Comput. Sci.* 154(3), pp. 19–32. Available at http://dx.doi.org/10.1016/j.entcs.2006.05.004.

[11] Fabien Fleutot (2004): *Encoding an Object Calculus into Interaction Nets*. In M. Fernandez, editor: *Proc. of the 2nd Int. Workshop on Term Graph Rewriting (TERMGRAPH 2004)*, ENTCS, Rome. Available at http://dx.doi.org/10.1016/j.entcs.2005.03.024.

[12] Georges Gonthier, Martín Abadi & Jean-Jacques Lévy (1992): *The Geometry of Optimal Lambda Reduction*. In: *Proceedings of the 19th ACM Symposium on Principles of Programming Languages (POPL'92)*, ACM Press, pp. 15–26. Available at http://dx.doi.org/10.1145/143165.143172.

[13] Abubakar Hassan, Ian Mackie & Jorge Sousa Pinto (2008): *Visual Programming with Interaction Nets*. In Gem Stapleton, John Howse & John Lee, editors: *Diagrams*, *Lecture Notes in Computer Science* 5223, Springer, pp. 165–171. Available at http://dx.doi.org/10.1007/978-3-540-87730-1_17.

[14] O. Jensen & R. Milner (2004): *Bigraphs and mobile processes (revised)*. Technical Report 580, Computer Laboratory, University of Cambridge.

[15] Yves Lafont (1990): *Interaction Nets*. In: *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, ACM Press, pp. 95–108. Available at http://dx.doi.org/10.1145/96709.96718.

[16] Yves Lafont (1995): *From Proof Nets to Interaction Nets*. In J.-Y. Girard, Y. Lafont & L. Regnier, editors: *Advances in Linear Logic*, *London Mathematical Society Lecture Note Series* 222, Cambridge University Press, pp. 225–247. Available at http://dx.doi.org/10.1017/CBO9780511629150.012.

[17] Ian Mackie (1994): *The Geometry of Implementation*. Ph.D. thesis, Department of Computing, Imperial College of Science, Technology and Medicine.

[18] Ian Mackie (2004): *Efficient λ-evaluation with interaction nets*. In V. van Oostrom, editor: *Proc. 15th Int. Conference on Rewriting Techniques and Applications (RTA'04)*, *Lecture Notes in Computer Science* 3091, Springer-Verlag, pp. 155–169. Available at http://dx.doi.org/10.1007/978-3-540-25979-4_11.

[19] Ian Mackie (2009): *A rewriting paradigm for program and algorithm animation*. In: *VL/HCC*, IEEE, pp. 170–173. Available at http://doi.ieeecomputersociety.org/10.1109/VLHCC.2009.5295272.

[20] Ian Mackie (2010): *A Visual Model of Computation*. In Jan Kratochvíl, Angsheng Li, Jirí Fiala & Petr Kolman, editors: *TAMC*, *Lecture Notes in Computer Science* 6108, Springer, pp. 350–360. Available at http://dx.doi.org/10.1007/978-3-642-13562-0_32.

[21] Robin Milner (2001): *Bigraphical Reactive Systems*. In Kim Guldstrand Larsen & Mogens Nielsen, editors: *CONCUR*, *Lecture Notes in Computer Science* 2154, Springer, pp. 16–35. Available at http://dx.doi.org/10.1007/3-540-44685-0_2.

[22] Jorge Sousa Pinto (2000): *Sequential and Concurrent Abstract Machines for Interaction Nets*. In J. Tiuryn, editor: *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS)*, *Lecture Notes in Computer Science* 1784, Springer-Verlag, pp. 267–282. Available at http://dx.doi.org/10.1007/3-540-46432-8_18.