EPTCS 408

Proceedings of the

13th International Workshop on

Developments in Computational Models

Rome, Italy, 2 July 2023

Edited by: Sandra Alves and Ian Mackie

Published: 1st October 2024
DOI: 10.4204/EPTCS.408
ISSN: 2075-2180
Open Publishing Association

Preface

This volume contains the proceedings of DCM 2023, the 13th International Workshop on Developments in Computational Models held on 2 July 2023 in Rome, Italy. DCM 2023 was organised as a one-day satellite event of FSCD 2023, the 8th International Conference on Formal Structures for Computation and Deduction.

Several new models of computation have emerged in the last few years, and many developments of traditional computational models have been proposed with the aim of taking into account the new demands of computer systems users and the new capabilities of computation engines. A new computational model, or a new feature in a traditional one, usually is reflected in a new family of programming languages, and new paradigms of software development.

The aim of this series of workshops is to bring together researchers who are currently developing new computational models or new features for traditional computational models, in order 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 this area. Topics of interest include all abstract models of computation and their applications to the development of programming languages and systems. This includes (but is not limited to):

- Functional calculi: lambda-calculus, rho-calculus, term and graph rewriting;
- quantum computation, including implementations and formal methods in quantum protocols;
- probabilistic computation and verification in modelling situations;
- chemical, biological and bio-inspired computation, including spatial models, self-assembly, growth models;
- models of concurrency, including the treatment of mobility, trust, and security;
- infinitary models of computation;
- information-theoretic ideas in computing.

The programme committee accepted 5 papers for presentation at the workshop. The program also included 3 invited talks by:

- Gabriele Vanoni, INRIA. *"What are abstract machines good for?"*
- Mariangiola Dezani-Ciancaglini, Torino University. *"Partial Typing for Open Compliance in Multiparty Sessions."*
- Clemens Grabmayer, Gran Sasso Science Institute. *"From Compactifying Lambda-letrec Terms to Recognizing Regular-Expression Processes."*

Following the workshop, authors were invited to submit a contribution for the formal proceedings. Seven papers were selected that compose the present volume.

We would like to thank all those who contributed to DCM 2023. We are grateful to the Programme Committee members and the external referees for their careful and efficient work in reviewing and selecting the submitted papers. A special thanks to EPTCS for supporting the publication of this issue.

Sandra Alves and Ian Mackie
DCM 2023 PC-chairs

S. Alves and I. Mackie (Eds.): 13th International Workshop
on Developments in Computational Models 2023 (DCM'23).
EPTCS 408, 2024, pp. i–i, doi:10.4204/EPTCS.408.0

Table of Contents

Partial Typing for Asynchronous Multiparty Sessions

Franco Barbanera *
Dipartimento di Matematica e Informatica, Università di Catania, Catania, Italy
franco.barbanera@unict.it

Mariangiola Dezani-Ciancaglini Ugo de'Liguoro †
Dipartimento di Informatica, Università di Torino, Torino, Italy
{dezani,deligu}@di.unito.it

Formal verification methods for concurrent systems cannot always be scaled-down or tailored in order to be applied on specific subsystems. We address such an issue in a MultiParty Session Types setting by devising a *partial* type assignment system for multiparty sessions (i.e. sets of concurrent participants) with asynchronous communications. Sessions are possibly typed by "asynchronous global types" describing the overall behaviour of specific subsets of participants only (from which the word "partial"). Typability is proven to ensure that sessions enjoy the partial versions of the well-known properties of lock- and orphan-message-freedom.

Keywords: MultiParty Session Types, Asynchronous Communication, Lock-freedom.

1 Introduction

When validating/verifying distributed and concurrent systems, it is often natural to identify different subsystems for which the properties we have to take into account are not those required for the whole system, if any. The system of a social media, for instance, is made of users and services the former are provided with. The users are the main concern of the social media, which hence tend to ensure to the user subsystem properties which cannot be (or need not to be) ensured to the services. This particularly applies in case services are managed by a second party not under direct control of the social media. *Lock-freedom* is a relevant specimen of such properties. It ensures that no lock is ever reached in the evolution of a system. A lock being a system's reachable configuration where a still active participant is forever prevented to perform any action in any possible continuation of the system[1]. In particular, such a configuration is called a p-lock in case the stuck participant is p. A social media would hence be focused on p-lock freedom for each $\mathsf{p} \in \mathscr{P}$, where $\mathscr{P}$ is the set of users in the current example. As far as the users cannot get into a lock, the services can behave as they like best. The social media can also be interested in that, in case of an asynchronous model of communication, the messages exchanged among the users are eventually received. This is a *partial* version of the property referred to in the literature as *orphan-message freedom*. An investigation on verification of partial properties was carried on in [1] in the setting of MultiParty Session Types (MPST for short), in particular in a *bottom-up* MPTS setting. Unlike formalisms using the notion of *projections*, the formalism in [1] enables to exploit an approach to the development and verification of distributed/concurrent system where systems (formalised here

*Partially supported by Project "National Center for HPC, Big Data e Quantum Computing", Programma M4C2, Investimento 1.3 – Next Generation EU.

†Partially supported by Project INDAM-GNCS "Fondamenti di Informatica e Sistemi Informatici".

[1] Actually several slightly different property are present in the literature under the name "lock-freedom".

S. Alves and I. Mackie (Eds.): 13th International Workshop
on Developments in Computational Models 2023 (DCM'23).
EPTCS 408, 2024, pp. 1–20, doi:10.4204/EPTCS.408.1

through the notion of "network", a parallel composition of named processes) are first developed and then subsequently proved sound with respect a specific overall description of the system's behaviour by checking the network against a *global type*. The MPST type system of [1] derives judgements of the shape

$$\vdash_{\mathscr{P}} \mathbb{N} : \mathsf{G}$$

where $\mathscr{P}$ is a set of participants, $\mathbb{N}$ is a network and G is a global type. The typing is *partial* since some communications between participants in $\mathscr{P}$ do not appear in the global type. Typing $\mathbb{N}$ with G does ensure that (a) the communications of the participants in $\mathbb{N}$ not belonging to $\mathscr{P}$ comply with the interaction scenario represented by G and (b) $\mathbb{N}$ is p-lock-free for each $\mathsf{p} \notin \mathscr{P}$.

In the present paper we push further the investigation of [1] by treating an asynchronous model of communication, instead of a synchronous one. Besides, we take into account also the partial version of the property of *orphan-message freedom*. The calculus, the global types and the type system we use are inspired by [3, 4, 7].

Contributions and structure of the paper. In Section 2 we recall from [3] the asynchronous calculus of multiparty sessions. Also, we adapt from [1] the notion of $\mathscr{P}$-lock-freedom (the absence of locks is ensured here to the participants in $\mathscr{P}$) and introduce the novel notion of $\mathscr{P}$-orphan-message freedom. An example is given to clarify the various notions and results. Section 3 is devoted to the presentation of (asynchronous) global types from [3] and the introduction of our "partial" type system, assigning global types to multiparty sessions, where some communications can be ignored. The relevant properties of partially typable sessions are proved in Section 4. In particular Subject Reduction, Session Fidelity, $\mathscr{P}$-lock-freedom and $\mathscr{P}$-orphan-message-freedom. A section summing up our results, discussing related works and possible directions for future work concludes the paper.

2 Multiparty Sessions

The calculus of multiparty sessions, as well as global types, used in the present paper are inspired by [3]. The simplicity of the calculus with respect to the original MPST calculus [9] and of many of the subsequent ones, as well as the lack of explicit channels, enables us to focus on our main concerns. Besides, it allows for a clear explanation of the type system we will introduce in the next section. All this has however the cost of preventing the representation of session interleaving and delegation.

We use the following base sets and notation: *labels*, ranged over by $\lambda, \lambda', \ldots$; *session participants*, ranged over by $\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}, \mathsf{u}, \ldots$; *processes*, ranged over by $P, Q, R, S, U, \ldots$; *networks*, ranged over by $\mathbb{N}, \mathbb{N}', \ldots$; *queues*, ranged over by $\mathscr{M}, \mathscr{M}', \ldots$; *integers*, ranged over by $i, j, l, h, k, \ldots$; *(finite) integer sets*, ranged over by $I, J, L, H, K, \ldots$.

Definition 2.1 (Processes) Processes *are defined by:*

$$P \quad ::=_{\rho} \quad \mathbf{0} \mid \mathsf{p}!\{\lambda_i.P_i\}_{i\in I} \mid \mathsf{p}?\{\lambda_i.P_i\}_{i\in I}$$

where $I \neq \emptyset$ and $\lambda_h \neq \lambda_k$ for $h, k \in I$ and $h \neq k$.

The symbol $::=_{\rho}$, in the above definition and in other definitions, indicates that the productions of the grammar should be interpreted *coinductively*. That is, they define possibly infinite processes. However, we assume such processes to be *regular*, i.e. with finitely many distinct subprocesses. In this way, we only obtain processes which are solutions of finite sets of equations, see [6]. We choose this formulation since it allows us to avoid explicitly handling variables, thus simplifying a lot the technical development.

A process of shape $\mathsf{p}!\{\lambda_i.P_i\}_{i\in I}$ (*internal choice*) chooses a label in the set $\{\lambda_i \mid i \in I\}$ to be sent to p, and then behaves differently depending on the label sent. A process of shape $\mathsf{p}?\{\lambda_i.P_i\}_{i\in I}$ (*external choice*) waits for receiving one of the labels $\{\lambda_i \mid i \in I\}$ from p, and then behaves as P_i depending on the received label λ_i. Note that the set of indexes in choices is assumed to be non-empty, and the corresponding labels to be pairwise distinct. An internal choice which is a singleton is simply written $\mathsf{p}!\lambda.P$; analogously for an external choice. The process $\mathbf{0}$ is inactive and we omit trailing $\mathbf{0}$. In a full-fledged calculus, labels would carry values, namely they would be of shape $\lambda(v)$. For simplicity, here we consider "pure" labels.

The *participants of a process* are the senders and the receivers which occur in the process itself. Their set is defined as the smallest set satisfying

$$\mathsf{Prt}(\mathbf{0}) = \emptyset \qquad \mathsf{Prt}(\mathsf{p}!\{\lambda_i.P_i\}_{i\in I}) = \mathsf{Prt}(\mathsf{p}?\{\lambda_i.P_i\}_{i\in I}) = \{\mathsf{p}\} \cup \bigcup_{i\in I} \mathsf{Prt}(P_i)$$

We use queues in order to formalise a one-to-one asynchronous model of communication. Instead of explicitly defining a queue for each possible sender and receiver, we use a single queue and equip the communicated labels with their sender and receiver names, so forming triples that we dub *messages*.

Definition 2.2 (Messages and Queues) *i)* Messages *are triples of the form* $\langle \mathsf{p}, \lambda, \mathsf{q} \rangle$ *denoting that participant* p *is the sender of label* λ *to the receiver* q.

ii) Message queues *(queues for short) are defined by the following grammar:*

$$\mathscr{M} ::= \emptyset \mid \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathscr{M}$$

Sent messages are stored in a queue, from which they are subsequently fetched by the receiver.

The order of messages in the queue is the order in which they will be read. Since order matters only between messages with the same sender and receiver, we always consider message queues modulo the following structural equivalence:

$$\mathscr{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{r}, \lambda', \mathsf{s} \rangle \cdot \mathscr{M}' \equiv \mathscr{M} \cdot \langle \mathsf{r}, \lambda', \mathsf{s} \rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathscr{M}' \text{ if } \mathsf{p} \neq \mathsf{r} \text{ or } \mathsf{q} \neq \mathsf{s}$$

Note, in particular, that $\langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{q}, \lambda', \mathsf{p} \rangle \equiv \langle \mathsf{q}, \lambda', \mathsf{p} \rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle$. These two equivalent queues represent a situation in which both participants p and q have sent a label to the other one, and neither of them has read the message. This case may happen in a multiparty session with asynchronous communication.

The *participants of queues* are the senders and the receivers which occur in the queue, i.e.

$$\mathsf{Prt}(\emptyset) = \emptyset \qquad \mathsf{Prt}(\langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathscr{M}) = \{\mathsf{p}, \mathsf{q}\} \cup \mathsf{Prt}(\mathscr{M})$$

A multiparty sessions is comprised of a network, i.e. a number of pairs participant/process of shape $\mathsf{p}[\![P]\!]$ composed in parallel, each with a different participant p, and a message queue.

Definition 2.3 (Networks and Sessions) *i)* Networks *are defined as finite parallel composition of named processes, namely*

$$\mathbb{N} = \mathsf{p}_1[\![P_1]\!] \parallel \cdots \parallel \mathsf{p}_n[\![P_n]\!]$$

where $\mathsf{p}_h \neq \mathsf{p}_k$ *and* $\mathsf{p}_h \notin \mathsf{Prt}(P_h)$ *for any* $1 \leq h \neq k \leq n$.

ii) Sessions *are defined as pairs of networks and message queues of the following form:*

$$\mathbb{N} \parallel \mathscr{M}$$

$$\mathsf{p}[\![\,\mathsf{q}!\{\lambda_i.P_i\}_{i\in I}\,]\!] \parallel \mathbb{N} \parallel \mathscr{M} \xrightarrow{\mathsf{pq}!\lambda_h} \mathsf{p}[\![\,P_h\,]\!] \parallel \mathbb{N} \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_h, \mathsf{q}\rangle \quad \text{where } h \in I \qquad [\textsc{Send}]$$

$$\mathsf{p}[\![\,\mathsf{q}?\{\lambda_i.P_i\}_{i\in I}\,]\!] \parallel \mathbb{N} \parallel \langle \mathsf{q}, \lambda_h, \mathsf{p}\rangle \cdot \mathscr{M} \xrightarrow{\mathsf{pq}?\lambda_h} \mathsf{p}[\![\,P_h\,]\!] \parallel \mathbb{N} \parallel \mathscr{M} \quad \text{where } h \in I \qquad [\textsc{Rcv}]$$

Figure 1: LTS for sessions.

The condition $\mathsf{p}_h \notin \mathsf{Prt}(P_h)$ forbids self-messages.

We assume the standard structural congruence on networks (denoted $\equiv$), that is we consider sessions modulo permutation of components and adding/removing components of the shape $\mathsf{p}[\![\,\mathbf{0}\,]\!]$.

If $P \neq \mathbf{0}$ we write $\mathsf{p}[\![\,P\,]\!] \in \mathbb{N}$ as short for $\mathbb{N} \equiv \mathsf{p}[\![\,P\,]\!] \parallel \mathbb{N}'$ for some $\mathbb{N}'$. This abbreviation is justified by the associativity and commutativity of $\parallel$.

The *participants of networks* are the participants which occur in processes, i.e.

$$\mathsf{Prt}(\mathbb{N}) = \bigcup_{\mathsf{p}[\![P]\!]\in\mathbb{N}} \{\mathsf{Prt}(P)\}$$

The *players of networks* are the participants associated with active processes, i.e.

$$\mathsf{Plays}(\mathbb{N}) = \{\mathsf{p} \mid \mathsf{p}[\![\,P\,]\!] \in \mathbb{N}\}$$

To define the asynchronous operational semantics of sessions, we use an LTS whose labels record the outputs and the inputs.

Definition 2.4 (Asynchronous Operational Semantics) *We equip sessions with the (asynchronous)* operational semantics *specified by the LTS of Figure 1. Transitions are labelled with* communications *(ranged over by β) which are either the asynchronous emission of a label λ from participant* p *to participant* q *(notation* $\mathsf{pq}!\lambda$*) or the actual reading by participant* p *of the label λ sent by participant* q *(notation* $\mathsf{pq}?\lambda$*).*

Rule [Send] in Figure 1 allows a participant p with an internal choice (a sender) to send one of its possible labels λ_h, by adding the corresponding message to the queue. Symmetrically, Rule [Rcv] allows a participant p with an external choice (a receiver) to read the first message in the queue sent to her by a given participant q, if its label λ_h is one of those she is waiting for.

The *players of communications* are the senders for the outputs and the receivers for the inputs, i.e. we define

$$\mathsf{play}(\mathsf{pq}!\lambda) = \mathsf{play}(\mathsf{pq}?\lambda) = \mathsf{p}$$

As usual we define (possibly empty) sequences of communications as traces.

Definition 2.5 (Traces) *(Finite) traces are defined by $\tau := \varepsilon \mid \beta \cdot \tau$.*

When $\tau = \beta_1 \cdot \ldots \cdot \beta_n$ $(n \geq 1)$ we write $\mathbb{N} \parallel \mathscr{M} \xrightarrow{\tau} \mathbb{N}' \parallel \mathscr{M}'$ as short for

$$\mathbb{N} \parallel \mathscr{M} \xrightarrow{\beta_1} \mathbb{N}_1 \parallel \mathscr{M}_1 \cdots \xrightarrow{\beta_n} \mathbb{N}_n \parallel \mathscr{M}_n = \mathbb{N}' \parallel \mathscr{M}'$$

With $\mathbb{N} \parallel \mathscr{M} \not\rightarrow$ we denote that the session $\mathbb{N} \parallel \mathscr{M}$ is stuck.

Example 2.6 (A social media session) A social network has two users (u_1 and u_2) that want to interact using a service s. The users exchange messages GO and STOP communicating when they like to continue or not their interaction. They "should" REQuest DATA to the service only when they both are willing to

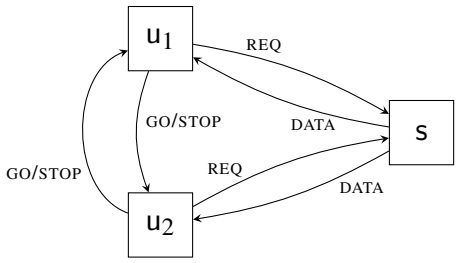


Figure 2: Representation of the session of Example 2.6.

do. The above system is roughly described (disregarding the logical order of messages) in Figure 2. A multiparty session corresponding to this system is the following.

$$\mathsf{u}_1[\![U_1]\!] \parallel \mathsf{u}_2[\![U_2]\!] \parallel \mathsf{s}[\![S]\!] \parallel \emptyset$$

$$U_1 = \mathsf{u}_2! \begin{cases} \text{GO}.\mathsf{u}_2? \begin{cases} \text{GO}.\mathsf{s}!\text{REQ}.\mathsf{s}?\text{DATA}.U_1 \\ \text{STOP}.\mathsf{s}!\text{REQ} \end{cases} \\ \text{STOP}.\mathsf{u}_2? \begin{cases} \text{GO} \\ \text{STOP} \end{cases} \end{cases}$$

$$U_2 = \mathsf{u}_1! \begin{cases} \text{GO}.\mathsf{u}_1? \begin{cases} \text{GO}.\mathsf{s}!\text{REQ}.\mathsf{s}?\text{DATA}.U_2 \\ \text{STOP} \end{cases} \\ \text{STOP}.\mathsf{u}_1? \begin{cases} \text{GO} \\ \text{STOP} \end{cases} \end{cases}$$

$$S = \mathsf{u}_2?\text{REQ}.\mathsf{u}_1?\text{REQ}.\mathsf{u}_1!\text{DATA}.\mathsf{u}_2!\text{DATA}.S$$

where both participants start sending messages, a feature which typically can be dealt only thanks to asynchronous communication. The behaviours of u_1 and u_2 only differ in that the process U_1, after sending GO to u_2 and receiving STOP from u_2, sends a REQ to the service. So the process U_1 does not precisely implement the prescribed behaviour, while U_2 does.

2.1 Partial Communication Properties

Now, we define the property of $\mathscr{P}$-lock-freedom. This property was first introduced in [1], where $\mathscr{P}$ was the set of participants whose lock-freedom we don't care about. $\mathscr{P}$-lock-freedom is a "partial" version of the standard lock-freedom [12, 13]. The latter consists in the possibility of completion of pending communications of any participant (this can be alternatively stated by saying that any participant is lock-free). We are interested instead in the progress of some explicitly specified participants only.

Definition 2.7 ($\mathscr{P}$-lock-freedom) *i) A multiparty session* $\mathbb{N} \parallel \mathscr{M}$ *is* p-lock-free *if* $\mathbb{N} \parallel \mathscr{M} \xrightarrow{\tau} \mathbb{N}' \parallel \mathscr{M}'$ *and* $\mathsf{p}[\![P]\!] \in \mathbb{N}'$ *imply* $\mathbb{N}' \parallel \mathscr{M}' \xrightarrow{\tau' \cdot \beta}$ *for some* τ' *and* β *such that* $\mathsf{p} \in \mathsf{play}(\beta)$.

ii) A multiparty session $\mathbb{N} \parallel \mathscr{M}$ *is* $\mathscr{P}$-lock-free *if it is* p-lock-free *for each* $\mathsf{p} \in \mathscr{P}$.

iii) A multiparty session $\mathbb{N} \parallel \mathscr{M}$ is a lock-free session *if it is* p-lock-free *for each* $\mathsf{p} \in \mathsf{Plays}(\mathbb{N})$.

It is natural to extend also the usual notion of Deadlock-freedom to our setting.

Definition 2.8 ($\mathscr{P}$-deadlock-freedom) *A multiparty session* $\mathbb{N} \parallel \mathscr{M}$ *is a* $\mathscr{P}$-deadlock-free *session if* $\mathbb{N} \parallel \mathscr{M} \xrightarrow{\tau} \mathbb{N}' \parallel \mathscr{M}' \not\rightarrow$ *implies* $\mathsf{p} \notin \mathsf{Plays}(\mathbb{N}')$ *for any* $\mathsf{p} \in \mathscr{P}$.

It is immediate to check that, as for standard Lock- and Deadlock-freedom, the following hold.

Fact 2.9 *$\mathscr{P}$-lock-freedom implies $\mathscr{P}$-deadlock-freedom.*

Trivially, as for the standard versions of the properties, the vice versa does not hold whenever $\mathscr{P} \neq \emptyset$.

Definition 2.10 ($\mathscr{P}$-orphan-message-freedom)

i) A multiparty session $\mathbb{N} \parallel \mathscr{M}$ *is* pq-orphan-message-free *if* $\mathbb{N} \parallel \mathscr{M} \xrightarrow{\tau} \mathbb{N}' \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathscr{M}'$ *implies* $\mathbb{N}' \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathscr{M}' \xrightarrow{\tau' \cdot \mathsf{qp}?\lambda}$ *for some* τ'.

ii) A multiparty session $\mathbb{N} \parallel \mathscr{M}$ *is* $\mathscr{P}$-orphan-message-free *if it is* pq*-orphan-message-free for each pair of participants* $\mathsf{p}, \mathsf{q} \in \mathscr{P}$.

iii) A multiparty session $\mathbb{N} \parallel \mathscr{M}$ *is* orphan-message-free *if it is* $\mathsf{Plays}(\mathbb{N}) \cup \mathsf{Prt}(\mathbb{N}) \cup \mathsf{Prt}(\mathscr{M})$*-orphan-message-free.*

Point (iii) of previous definition is justified by the example $\mathbb{N} = \mathsf{p}[\![\, \mathsf{q}!\lambda \,]\!] \parallel \emptyset \xrightarrow{\mathsf{pq}!\lambda} \mathsf{p}[\![\, \mathbf{0} \,]\!] \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle$, where the message $\langle \mathsf{p}, \lambda, \mathsf{q} \rangle$ is orphan and $\mathsf{p} \in \mathsf{Plays}(\mathbb{N})$, $\mathsf{q} \in \mathsf{Prt}(\mathbb{N})$.

Example 2.11 (Partial properties for the social media example) It is not difficult to check that the session of Example 2.6 is neither lock-free nor orphan-message-free. In fact we get an s-lock whenever at least one among u_1 and u_2 sends to the other the message STOP. In such a case the process of s is not $\mathbf{0}$, but unable to perform the input action it is willing to do. An orphan message does result present in the queue because of a "programming error": in case u_1 sends GO to u_2, receives STOP from u_2 and then sends REQ to the server, it happens that such a REQ from u_1 will never be received by s, since a REQ from u_2 should be received first, but such a message will never be sent.

The social network, however, is interested in the absence of locks for the $\{\mathsf{u}_1, \mathsf{u}_2\}$ subsystem only (i.e. $\{\mathsf{u}_1, \mathsf{u}_2\}$-lock-freedom) as well in the absence of orphan-messages only for the messages exchanged among u_1 and u_2 (i.e. $\{\mathsf{u}_1, \mathsf{u}_2\}$-orphan-message-freedom).

3 Global Types and Type System

The vast majority of global types used in the literature are independent of the synchronicity/asynchronicity of the underlying communication model. This means that, in a global type, the exchange of a message m from a participant A to a participant B is generally represented by something like $\mathtt{A} \xrightarrow{\mathsf{m}} \mathtt{B}$. This is then interpreted either as the synchronous exchange of m according to a handshaking protocol between A and B or as the simultaneous representation of two distinct asynchronous actions: the insertion of m in a communication medium (typically a queue or a bag) and the acquisition of the message from that. In [3, 4, 7] global types are instead strictly tailored for asynchronous interactions: the separate output and input actions, which together form an asynchronous communication (respectively $\mathsf{pq}!\lambda$ and $\mathsf{pq}?\lambda$ in our formalism, see below), are made visible in the global type. Even if this is actually more than what a choreographic formalism should require (our one can in fact hardly be considered a choreographic formalism in the usual sense), it allows the global types to be used in a type assignment system for asynchronous processes guaranteeing relevant (partial, in our case) communication properties. Being the asynchrony of communication syntactically evident in the global type, the formal verification of such properties can be performed without having to consider a layer of "semantic" interpretation of the types, so maintaining the complexity of proofs at the same complexity level as those for synchronous formalisms like the one in [1].

Definition 3.1 (Asynchronous Global Types) (Asynchronous) global types G *are defined by the following grammar:*

$$\mathsf{G} \quad ::=_\rho \quad \mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I} \mid \mathsf{pq}?\lambda.\mathsf{G} \mid \mathsf{End}$$

where $I \neq \emptyset$, $\mathsf{p} \neq \mathsf{q}$ *and* $\lambda_h \neq \lambda_k$ *for* $h,k \in I$ *and* $h \neq k$.

As for processes, $::=_\rho$ indicates that global types are coinductively defined *regular* terms. The global type $\mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I}$ specifies that p sends a label λ_h with $h \in I$ to q and then the interaction described by the global type G_h takes place. Dually, the global type $\mathsf{pq}?\lambda.\mathsf{G}$ specifies that q receives label λ from p and then the interaction described by the global type G takes place. The terminated global type is End and we will omit trailing End's.

Clearly message outputs must precede the corresponding inputs, since in the asynchronous communication the output puts the message on the queue and the input takes the message from the queue. Once a message is on the queue no other message can be read with the same sender and receiver. This justifies the fact that inputs in global types have no choices.

Example 3.2 (A global type for the social media example) A global type describing a possible behaviour of the network of Example 2.6 is provided in Figure 3.

$$\mathsf{G} = \mathsf{u}_1\mathsf{u}_2! \begin{cases} \textsc{go}.\mathsf{u}_2\mathsf{u}_1! \begin{cases} \textsc{go}.\mathsf{u}_1\mathsf{u}_2?\textsc{go}.\mathsf{u}_2\mathsf{u}_1?\textsc{go}.\mathsf{u}_2\mathsf{s}!\textsc{req}.\mathsf{s}\mathsf{u}_2?\textsc{req}.\mathsf{u}_1\mathsf{s}!\textsc{req}.\mathsf{s}\mathsf{u}_1?\textsc{req}. \\ \quad \mathsf{s}\mathsf{u}_1!\textsc{data}.\mathsf{u}_1\mathsf{s}?\textsc{data}.\mathsf{s}\mathsf{u}_2!\textsc{data}.\mathsf{u}_2\mathsf{s}?\textsc{data}.\mathsf{G} \\ \textsc{stop}.\mathsf{u}_1\mathsf{u}_2?\textsc{stop}.\mathsf{u}_2\mathsf{u}_1?\textsc{go}.\mathsf{u}_1\mathsf{s}!\textsc{req} \end{cases} \\ \textsc{stop}.\mathsf{u}_2\mathsf{u}_1! \begin{cases} \textsc{go}.\mathsf{u}_1\mathsf{u}_2?\textsc{go}.\mathsf{u}_2\mathsf{u}_1?\textsc{stop} \\ \textsc{stop}.\mathsf{u}_1\mathsf{u}_2?\textsc{stop}.\mathsf{u}_2\mathsf{u}_1?\textsc{stop} \end{cases} \end{cases}$$

Figure 3: A global type for the social media session.

The set of *players of a global type,* notation $\mathsf{Plays}(\mathsf{G})$, is the smallest set satisfying the following equations:

$$\mathsf{Plays}(\mathsf{End}) = \emptyset$$
$$\mathsf{Plays}(\mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I}) = \{\mathsf{p}\} \cup \textstyle\bigcup_{i\in I} \mathsf{Plays}(\mathsf{G}_i) \qquad \mathsf{Plays}(\mathsf{pq}?\lambda.\mathsf{G}') = \{\mathsf{p}\} \cup \mathsf{Plays}(\mathsf{G}')$$

Notice that the sets of players are always finite thanks to the regularity of global types.

To guarantee good communication properties for typable sessions, we require global types to satisfy a boundedness condition. To formalise boundedness we use the notion of *path* of a global type. *Paths* are actual paths in global types viewed as trees. They are possibly infinite sequences of communications, and are ranged over by ξ. Note that a finite path is a trace in the sense of Definition 2.5. We extend the notation $\cdot$ to denote also the concatenation of a finite sequence with a possibly infinite sequence. The function Paths returns the set of all the *paths* of a global type and is defined as the greatest set such that:

$$\begin{array}{ll} \mathsf{Paths}(\mathsf{End}) & = \{\varepsilon\} \\ \mathsf{Paths}(\mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I}) & = \bigcup_{i\in I}\{\mathsf{pq}!\lambda_i \cdot \xi \mid \xi \in \mathsf{Paths}(\mathsf{G}_i)\} \\ \mathsf{Paths}(\mathsf{pq}?\lambda.\mathsf{G}') & = \{\mathsf{pq}?\lambda \cdot \xi \mid \xi \in \mathsf{Paths}(\mathsf{G}')\} \end{array}$$

If $x \in \mathbf{N} \cup \{\infty\}$ is the length of ξ, i.e. $x = \mid \xi \mid$, we denote by $\xi[n]$ the n-th communication in the path ξ, where $1 \leq n < x$ if $x = \infty$ and $1 \leq n \leq x$ if $x \neq \infty$. It is handy to define the *depth* of a player p in a global type G, $\mathsf{depth}(\mathsf{G},\mathsf{p})$.

Definition 3.3 (Depth of a Player) *Let* G *be a global type. For* $\xi \in \mathsf{Paths}(\mathsf{G})$ *set*

$$\mathsf{depth}(\xi,\mathsf{p}) = \inf\{n \mid \mathsf{play}(\xi[n]) = \mathsf{p}\}$$

and define $\mathsf{depth}(\mathsf{G},\mathsf{p})$*, the* depth *of* p *in* G*, as follows:*

$$\mathsf{depth}(\mathsf{G},\mathsf{p}) = \begin{cases} \sup\{\mathsf{depth}(\xi,\mathsf{p}) \mid \xi \in \mathsf{Paths}(\mathsf{G})\} & \mathsf{p} \in \mathsf{Plays}(\mathsf{G}) \\ 0 & \textit{otherwise} \end{cases}$$

Note that $\mathsf{depth}(\mathsf{G},\mathsf{p}) = 0$ iff $\mathsf{p} \notin \mathsf{Plays}(\mathsf{G})$. Moreover, if $\mathsf{p} \neq \mathsf{play}(\xi[n])$ for all $n \in \mathbf{N}$, then $\mathsf{depth}(\xi,\mathsf{p}) = \inf \emptyset = \infty$. Hence, if p is a player of a global type G and there is some path in G where p does not occur as a player, then $\mathsf{depth}(\mathsf{G},\mathsf{p}) = \infty$.

Definition 3.4 (Boundedness) *A global type* G *is* bounded *if* $\mathsf{depth}(\mathsf{G}',\mathsf{p})$ *is finite for each participant* $\mathsf{p} \in \mathsf{Plays}(\mathsf{G})$ *and each type* G' *which occurs in* G*.*

Example 3.5 The following example shows the necessity of considering all types occurring in a global type for defining boundedness. Consider $\mathsf{G} = \mathsf{rq}!\lambda.\mathsf{qr}?\lambda.\mathsf{G}'$, where

$$\mathsf{G}' = \mathsf{pq}!\{\lambda_1.\mathsf{qp}?\lambda_1.\mathsf{qr}!\lambda_3.\mathsf{rq}?\lambda_3\,,\ \lambda_2.\mathsf{qp}?\lambda_2.\mathsf{G}'\}$$

Then we have: $\mathsf{depth}(\mathsf{G},\mathsf{p}) = 3, \mathsf{depth}(\mathsf{G},\mathsf{q}) = 2, \mathsf{depth}(\mathsf{G},\mathsf{r}) = 1$, whereas $\mathsf{depth}(\mathsf{G}',\mathsf{p}) = 1, \mathsf{depth}(\mathsf{G}',\mathsf{q}) = 2, \mathsf{depth}(\mathsf{G}',\mathsf{r}) = \infty$.

Since global types are regular, the boundedness condition is decidable.

The following notion of *weight* will be used for defining the subsequent notion of $\mathscr{P}$-soundness, a condition in the typing rules, needed to guarantee $\mathscr{P}$-orphan-message-freedom. The weight says if and where the global type prescribes an input corresponding to a message. Clearly if the message is $\langle \mathsf{p},\lambda,\mathsf{q}\rangle$ and the global type is $\mathsf{qp}?\lambda'.\mathsf{G}$ with $\lambda \neq \lambda'$, then the global type forbids to read this message.

Definition 3.6 (Weight)

$$\mathsf{weight}(\mathsf{G},\langle \mathsf{p},\lambda,\mathsf{q}\rangle) = \begin{cases} 0 & \textit{if } \mathsf{G} = \mathsf{qp}?\lambda.\mathsf{G}' \\ \infty & \textit{if } \mathsf{G} = \mathsf{End} \textit{ or } \mathsf{G} = \mathsf{qp}?\lambda'.\mathsf{G}' \textit{ with } \lambda \neq \lambda' \\ 1 + \max_{i\in I} \mathsf{weight}(\mathsf{G}_i,\langle \mathsf{p},\lambda,\mathsf{q}\rangle) & \textit{if } \mathsf{G} = \mathsf{rs}!\{\lambda_i.\mathsf{G}_i\}_{i\in I} \\ 1 + \mathsf{weight}(\mathsf{G}',\langle \mathsf{p},\lambda,\mathsf{q}\rangle) & \textit{if } \mathsf{G} = \mathsf{rs}?\lambda'.\mathsf{G}' \textit{ and } \mathsf{r} \neq \mathsf{p} \textit{ or } \mathsf{s} \neq \mathsf{q} \end{cases}$$

We consider the parallel composition of a global type with a queue that we dub *type configuration*. The $\mathscr{P}$-soundness of type configurations ensures that all messages with both participants in $\mathscr{P}$ have corresponding inputs in all the paths of the global type.

Definition 3.7 ($\mathscr{P}$-soundness) *A type configuration* $\mathsf{G} \parallel \mathscr{M}$ *is* $\mathscr{P}$-sound *if* $\mathsf{weight}(\mathsf{G},\langle \mathsf{p},\lambda,\mathsf{q}\rangle)$ *is finite for all messages* $\langle \mathsf{p},\lambda,\mathsf{q}\rangle$ *which occur in* $\mathscr{M}$ *with* $\{\mathsf{p},\mathsf{q}\} \subseteq \mathscr{P}$*.*

3.1 Partial Type System

As mentioned before, we devise a type system ensuring partial communication properties for typable sessions. Being in an asynchronous setting, some restrictions have to be imposed in order to guarantee decidability of typability. We achieve that by looking at queues as invariants for cycles. This is a quite more flexible condition than, for instance, imposing a fixed bound on the number of messages between participants. It would be rather cumbersome to guarantee our condition in a coinductive type system which, like those in [3, 4, 7, 1], suits a formalism with coinductively defined processes and types. We

hence introduce an implicitly coinductive type system, that is looking like the inductive versions of coinductive systems, as defined in [14, Section 21.9]. We define an inductive system with *histories* (see below), where the queue invariance can be immediately guarantee by the typing rule for cycles.

Definition 3.8 (Histories) *A* history $\mathscr{H}$ *is a finite set of (session, global type) pairs, namely*

$$\mathscr{H} ::= \emptyset \mid \mathscr{H}, (\mathbb{N} \parallel \mathscr{M}, \mathsf{G})$$

We define $(\mathbb{N} \parallel -, \mathsf{G}) \in \mathscr{H}$ if $(\mathbb{N} \parallel \mathscr{M}, \mathsf{G}) \in \mathscr{H}$ for some $\mathscr{M}$.

Definition 3.9 (Partial Type System) *The judgements of our partial type system have the form*

$$\mathscr{H} \vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$$

where $\mathscr{P}$ *is a set of participants (those whose properties we are interested in) and where the global type* G *is bounded. The inference rules are described in Figure 4.*

$$[\textsc{End}]\ \frac{\mathsf{Plays}(\mathbb{N}) \cap \mathscr{P} = \emptyset}{\mathscr{H} \vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{End}}\ \mathsf{End} \parallel \mathscr{M} \text{ is } \mathscr{P}\text{-sound} \qquad [\textsc{Cycle}]\ \frac{(\mathbb{N} \parallel \mathscr{M}, \mathsf{G}) \in \mathscr{H}}{\mathscr{H} \vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}}$$

$$[\textsc{Out}]\ \frac{\mathscr{H}, (\mathsf{p}[\![P]\!] \parallel \mathbb{N} \parallel \mathscr{M}, \mathsf{G}) \vdash_{\mathscr{P}} \mathsf{p}[\![P_i]\!] \parallel \mathbb{N} \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q} \rangle : \mathsf{G}_i \quad (\mathsf{Plays}(\mathbb{N}) \setminus \mathsf{Plays}(\mathsf{G})) \cap \mathscr{P} = \emptyset \quad \forall i \in I}{\mathscr{H} \vdash_{\mathscr{P}} \mathsf{p}[\![P]\!] \parallel \mathbb{N} \parallel \mathscr{M} : \mathsf{G}}$$

$$\mathsf{G} \parallel \mathscr{M} \text{ is } \mathscr{P}\text{-sound} \quad (\mathsf{p}[\![P]\!] \parallel \mathbb{N} \parallel -, \mathsf{G}) \notin \mathscr{H} \quad \mathsf{G} = \mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i \in I} \quad P = \mathsf{q}!\{\lambda_i.P_i\}_{i \in I}$$

$$[\textsc{In}]\ \frac{\mathscr{H}, (\mathsf{p}[\![P]\!] \parallel \mathbb{N} \parallel \mathscr{M}, \mathsf{G}) \vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \mathbb{N} \parallel \mathscr{M} : \mathsf{G}' \quad (\mathsf{Plays}(\mathbb{N}) \setminus \mathsf{Plays}(\mathsf{G})) \cap \mathscr{P} = \emptyset \quad h \in I}{\mathscr{H} \vdash_{\mathscr{P}} \mathsf{p}[\![P]\!] \parallel \mathbb{N} \parallel \langle \mathsf{q}, \lambda_h, \mathsf{p} \rangle \cdot \mathscr{M} : \mathsf{G}}$$

$$\mathsf{G}' \parallel \mathscr{M} \text{ is } \mathscr{P}\text{-sound} \quad (\mathsf{p}[\![P]\!] \parallel \mathbb{N} \parallel -, \mathsf{G}) \notin \mathscr{H} \quad \mathsf{G} = \mathsf{pq}?\lambda_h.\mathsf{G}' \quad P = \mathsf{q}?\{\lambda_i.P_i\}_{i \in I}$$

Figure 4: Typing rules for sessions.

In case all the participants in $\mathscr{P}$ (those we care about) terminate, we are not interested anymore in what other participants do and hence we do not record their behaviours in the global type. This is essentially what is formalised by Axiom [End]. No message with both sender and receiver in $\mathscr{P}$ must be present in the queue if we wish to ensure $\mathscr{P}$-orphan-message-freedom. This is formalised by the clause "$\mathsf{End} \parallel \mathscr{M}$ *is* $\mathscr{P}$*-sound*" of Axiom [End].

The inductive rules of our system can be looked at as a type reconstruction algorithm for a coinductively defined system.

We formalise in Axiom [Cycle] also an invariant requirement for ensuring decidability, namely the invariance of queues for cycles. This implies that any output in a cycle must have a corresponding input in the cycle itself.

Rules [Out] and [In] enable to record in the global types the actions performed by processes. Rule [Out] adds in the process and in the global type the same outputs. Rule [In] adds one input in the global type and it allows more inputs in the process, mimicking the subtyping for session types [8]. Both rules require as premises the typability of the sessions obtained by reducing the added communications. These rules ask for some conditions. The condition $(\mathsf{Plays}(\mathbb{N}) \setminus \mathsf{Plays}(\mathsf{G})) \cap \mathscr{P} = \emptyset$ ensures that the communications done by players in $\mathbb{N}$ which belong to $\mathscr{P}$ are recorded in G. The $\mathscr{P}$-soundness condition for configurations is needed to ensure absence of orphan-messages with sender and receiver

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathscr{H}_{15} \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1]\!] \parallel \mathsf{u}_2[\![U_2]\!] \parallel \mathsf{s}[\![S]\!] \parallel \emptyset : \mathsf{G}}\,[\textsc{cycle}]}{\mathscr{H}_{14} \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_6 : \mathsf{G}_{15}}}{\mathscr{H}_{13} \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S^{\mathrm{III}}]\!] \parallel \emptyset : \mathsf{G}_{14}}}{\mathscr{H}_{12} \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1^{\mathrm{IV}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S^{\mathrm{III}}]\!] \parallel \mathscr{M}_5 : \mathsf{G}_{13}}}{\mathscr{H}_{11} \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1^{\mathrm{IV}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S^{\mathrm{II}}]\!] \parallel \emptyset : \mathsf{G}_{12}}}{\mathscr{H}_{10} \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1^{\mathrm{IV}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S^{\mathrm{I}}]\!] \parallel \mathscr{M}_4 : \mathsf{G}_{11}}}{\mathscr{H}_9 \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1^{\mathrm{III}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S^{\mathrm{I}}]\!] \parallel \emptyset : \mathsf{G}_{10}}}{\mathscr{H}_7 \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1^{\mathrm{III}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_3 : \mathsf{G}_9}}{\mathscr{H}_5 \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1^{\mathrm{III}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{III}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \emptyset : \mathsf{G}_7}}{\mathscr{H}_3 \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1^{\mathrm{III}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{I}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_0 : \mathsf{G}_5}}{\mathscr{H}_2 \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1^{\mathrm{I}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{I}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_1 : \mathsf{G}_3}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathscr{H}_8 \vdash_{\mathscr{P}} \mathsf{u}_1[\![\mathbf{0}]\!] \parallel \mathsf{u}_2[\![\mathbf{0}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_7 : \mathsf{End}}\,[\textsc{end}]}{\mathscr{H}_6 \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1^{\mathrm{V}}]\!] \parallel \mathsf{u}_2[\![\mathbf{0}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \emptyset : \mathsf{G}_8}}{\mathscr{H}_4 \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1^{\mathrm{V}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{II}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_0 : \mathsf{G}_6}}{\mathscr{H}_2 \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1^{\mathrm{I}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{II}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_2 : \mathsf{G}_4}$$

$$\dfrac{\mathscr{H}_1 \vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1^{\mathrm{I}}]\!] \parallel \mathsf{u}_2[\![U_2]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_0 : \mathsf{G}_1 \qquad \mathscr{D}}{\vdash_{\mathscr{P}} \mathsf{u}_1[\![U_1]\!] \parallel \mathsf{u}_2[\![U_2]\!] \parallel \mathsf{s}[\![S]\!] \parallel \emptyset : \mathsf{G}}$$

Figure 5: Derivation for the social media example.

in $\mathscr{P}$. The condition $(\mathsf{p}[\![P]\!] \parallel \mathbb{N} \parallel -, \mathsf{G}) \notin \mathscr{H}$, together with the one for Axiom [CYCLE], is used for ensuring decidability. Our type system is in fact decidable, since global types and processes are regular. In particular, any bottom-up attempt to reconstruct a branch of a to-be derivation necessarily ends up with an application of Axiom [END], or of Axiom [CYCLE] or fails because Rules [OUT] and [IN] do not apply.

Whereas our type system enables to deal with participants whose lock-freedom we do care about, the system of [1], besides taking into account a synchronous model of communication, deals with participants whose lock-freedom we do not care about. Even if equivalent from an abstract viewpoint, these two different perspectives from which one can deal with the notion of "partiality" , bring with them pros and cons when formalised in specific MPST type systems. For instance, something like the rule [WEAK] of [1] is not needed here, so accounting for simpler proofs. On the other hand, the loose treatment of disregarded participants in [1], where one can consider different sets of participants in different branches of derivations, allows for a modular development of the derivations.

The presence of queues in our asynchronous setting makes some extra conditions – besides the regularity of global types and processes – necessary in order to get a decidable type systems. Such extra conditions are definitely easier to formalise in an inductive system rather that in a coinductive one, so accounting for the use of an inductive system, unlike a coinductive one as in [1].

Example 3.10 (Typing for the social media example) The type derivation for our social media example is shown in Figure 5 where $\mathscr{P} = \{\mathsf{u}_1, \mathsf{u}_2\}$ and $\mathscr{D}$ is the derivation with conclusion

$$\mathscr{H}_1 \vdash_{\{\mathsf{u}_1,\mathsf{u}_2\}} \mathsf{u}_1[\![U_1^{\mathrm{II}}]\!] \parallel \mathsf{u}_2[\![U_2]\!] \parallel \mathsf{s}[\![S]\!] \parallel \langle \mathsf{u}_1, \textsc{stop}, \mathsf{u}_2 \rangle : \mathsf{G}_2$$

whose detailed description we omit for the sake of readability. The abbreviations used in Figure 5 are listed in Figures 6, 7, 8, 9.

In order to show that a type configuration does represent a correct and complete description of the

$$\mathscr{H}_1 = (\mathsf{u}_1[\![U_1]\!] \parallel \mathsf{u}_2[\![U_2]\!] \parallel \mathsf{s}[\![S]\!] \parallel \emptyset, \mathsf{G})$$
$$\mathscr{H}_2 = \mathscr{H}_1, (\mathsf{u}_1[\![U_1^{\mathrm{I}}]\!] \parallel \mathsf{u}_2[\![U_2]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_0, \mathsf{G}_1)$$
$$\mathscr{H}_3 = \mathscr{H}_2, (\mathsf{u}_1[\![U_1^{\mathrm{I}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{I}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_1, \mathsf{G}_3)$$
$$\mathscr{H}_4 = \mathscr{H}_2, (\mathsf{u}_1[\![U_1^{\mathrm{I}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{II}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_2, \mathsf{G}_4)$$
$$\mathscr{H}_5 = \mathscr{H}_3, (\mathsf{u}_1[\![U_1^{\mathrm{III}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{I}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_0, \mathsf{G}_5)$$
$$\mathscr{H}_6 = \mathscr{H}_4, (\mathsf{u}_1[\![U_1^{\mathrm{V}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{II}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_0, \mathsf{G}_6)$$
$$\mathscr{H}_7 = \mathscr{H}_5, (\mathsf{u}_1[\![U_1^{\mathrm{III}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{III}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \emptyset, \mathsf{G}_7)$$
$$\mathscr{H}_8 = \mathscr{H}_6, (\mathsf{u}_1[\![U_1^{\mathrm{V}}]\!] \parallel \mathsf{u}_2[\![\mathbf{0}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \emptyset, \mathsf{G}_8)$$
$$\mathscr{H}_9 = \mathscr{H}_7, (\mathsf{u}_1[\![U_1^{\mathrm{III}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_3, \mathsf{G}_9)$$
$$\mathscr{H}_{10} = \mathscr{H}_9, (\mathsf{u}_1[\![U_1^{\mathrm{III}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S^{\mathrm{I}}]\!] \parallel \emptyset, \mathsf{G}_{10})$$
$$\mathscr{H}_{11} = \mathscr{H}_{10}, (\mathsf{u}_1[\![U_1^{\mathrm{IV}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S^{\mathrm{I}}]\!] \parallel \mathscr{M}_4, \mathsf{G}_{11})$$
$$\mathscr{H}_{12} = \mathscr{H}_{11}, (\mathsf{u}_1[\![U_1^{\mathrm{IV}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S^{\mathrm{II}}]\!] \parallel \emptyset, \mathsf{G}_{12})$$
$$\mathscr{H}_{13} = \mathscr{H}_{12}, (\mathsf{u}_1[\![U_1^{\mathrm{IV}}]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S^{\mathrm{III}}]\!] \parallel \mathscr{M}_5, \mathsf{G}_{13})$$
$$\mathscr{H}_{14} = \mathscr{H}_{13}, (\mathsf{u}_1[\![U_1]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S^{\mathrm{III}}]\!] \parallel \emptyset, \mathsf{G}_{14})$$
$$\mathscr{H}_{15} = \mathscr{H}_{14}, (\mathsf{u}_1[\![U_1]\!] \parallel \mathsf{u}_2[\![U_2^{\mathrm{IV}}]\!] \parallel \mathsf{s}[\![S]\!] \parallel \mathscr{M}_6 : \mathsf{G}_{15})$$

Figure 6: Histories for the derivation of the social media example.

$$U_1^{\mathrm{I}} = \mathsf{u}_2?\begin{cases} \textsc{go}.U_1^{\mathrm{III}} \\ \textsc{stop}.U_1^{\mathrm{V}} \end{cases} \qquad U_1^{\mathrm{II}} = \mathsf{u}_2?\begin{cases} \textsc{go} \\ \textsc{stop} \end{cases} \qquad U_1^{\mathrm{III}} = \mathsf{s}!\textsc{req}.U_1^{\mathrm{IV}} \quad U_1^{\mathrm{IV}} = \mathsf{s}?\textsc{data}.U_1 \quad U_1^{\mathrm{V}} = \mathsf{s}!\textsc{req}$$

$$U_2^{\mathrm{I}} = \mathsf{u}_1?\begin{cases} \textsc{go}.U_2^{\mathrm{III}} \\ \textsc{stop} \end{cases} \qquad U_2^{\mathrm{II}} = \mathsf{u}_1?\begin{cases} \textsc{go} \\ \textsc{stop} \end{cases} \qquad U_2^{\mathrm{III}} = \mathsf{s}!\textsc{req}.U_2^{\mathrm{IV}} \qquad U_2^{\mathrm{IV}} = \mathsf{s}?\textsc{data}.U_2$$

$$S^{\mathrm{I}} = \mathsf{u}_1?\textsc{req}.S^{\mathrm{II}} \qquad S^{\mathrm{II}} = \mathsf{u}_1!\textsc{data}.S^{\mathrm{III}} \qquad S^{\mathrm{III}} = \mathsf{u}_2!\textsc{data}.S$$

Figure 7: Processes for the derivation of the social media example.

$$\mathscr{M}_0 = \langle \mathsf{u}_1, \textsc{go}, \mathsf{u}_2 \rangle \qquad \mathscr{M}_1 = \mathscr{M}_0 \cdot \langle \mathsf{u}_2, \textsc{go}, \mathsf{u}_1 \rangle \qquad \mathscr{M}_2 = \mathscr{M}_0 \cdot \langle \mathsf{u}_2, \textsc{stop}, \mathsf{u}_1 \rangle \qquad \mathscr{M}_3 = \langle \mathsf{u}_2, \textsc{req}, \mathsf{s} \rangle$$
$$\mathscr{M}_4 = \langle \mathsf{u}_1, \textsc{req}, \mathsf{s} \rangle \qquad \mathscr{M}_5 = \langle \mathsf{s}, \textsc{data}, \mathsf{u}_1 \rangle \qquad \mathscr{M}_6 = \langle \mathsf{s}, \textsc{data}, \mathsf{u}_2 \rangle \qquad \mathscr{M}_7 = \langle \mathsf{u}_1, \textsc{req}, \mathsf{s} \rangle$$

Figure 8: Queues for the derivation of the social media example.

$$\mathsf{G}_1 = \mathsf{u}_2\mathsf{u}_1!\begin{cases}\textsc{go}.\mathsf{G}_3\\ \textsc{stop}.\mathsf{G}_4\end{cases} \qquad \mathsf{G}_2 = \mathsf{u}_2\mathsf{u}_1!\begin{cases}\textsc{go}.\mathsf{u}_1\mathsf{u}_2?\textsc{go}.\mathsf{u}_2\mathsf{u}_1?\textsc{stop}\\ \textsc{stop}.\mathsf{u}_1\mathsf{u}_2?\textsc{stop}.\mathsf{u}_2\mathsf{u}_1?\textsc{stop}\end{cases} \qquad \mathsf{G}_3 = \mathsf{u}_1\mathsf{u}_2?\textsc{go}.\mathsf{G}_5$$

$$\mathsf{G}_4 = \mathsf{u}_1\mathsf{u}_2?\textsc{stop}.\mathsf{G}_6 \qquad \mathsf{G}_5 = \mathsf{u}_2\mathsf{u}_1?\textsc{go}.\mathsf{G}_7 \qquad \mathsf{G}_6 = \mathsf{u}_2\mathsf{u}_1?\textsc{go}.\mathsf{G}_8 \qquad \mathsf{G}_7 = \mathsf{u}_2\mathsf{s}!\textsc{req}.\mathsf{G}_9$$

$$\mathsf{G}_8 = \mathsf{u}_1\mathsf{s}!\textsc{req} \qquad \mathsf{G}_9 = \mathsf{s}\mathsf{u}_2?\textsc{req}.\mathsf{G}_{10} \qquad \mathsf{G}_{10} = \mathsf{u}_1\mathsf{s}!\textsc{req}.\mathsf{G}_{11} \qquad \mathsf{G}_{11} = \mathsf{s}\mathsf{u}_1?\textsc{req}.\mathsf{G}_{12}$$

$$\mathsf{G}_{12} = \mathsf{s}\mathsf{u}_1!\textsc{data}.\mathsf{G}_{13} \qquad \mathsf{G}_{13} = \mathsf{u}_1\mathsf{s}?\textsc{data}.\mathsf{G}_{14} \qquad \mathsf{G}_{14} = \mathsf{s}\mathsf{u}_2!\textsc{data}.\mathsf{G}_{15} \qquad \mathsf{G}_{15} = \mathsf{u}_2\mathsf{s}?\textsc{data}.\mathsf{G}$$

Figure 9: Global types for the derivation of the social media example.

$$[\textsc{Top-Out}]\ \frac{}{\mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I} \parallel \mathscr{M} \xrightarrow{\mathsf{pq}!\lambda_h} \mathsf{G}_h \parallel \mathscr{M}\cdot\langle \mathsf{p},\lambda_h,\mathsf{q}\rangle}\ h\in I$$

$$[\textsc{Top-In}]\ \frac{}{\mathsf{pq}?\lambda.\mathsf{G} \parallel \langle \mathsf{q},\lambda,\mathsf{p}\rangle\cdot\mathscr{M} \xrightarrow{\mathsf{pq}?\lambda} \mathsf{G} \parallel \mathscr{M}}$$

$$[\textsc{Inside-Out}]\ \frac{\mathsf{G}_i \parallel \mathscr{M}\cdot\langle \mathsf{p},\lambda_i,\mathsf{q}\rangle \xrightarrow{\beta} \mathsf{G}_i' \parallel \mathscr{M}'\cdot\langle \mathsf{p},\lambda_i,\mathsf{q}\rangle \quad \forall i\in I}{\mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{pq}!\{\lambda_i.\mathsf{G}_i'\}_{i\in I} \parallel \mathscr{M}'}\ \mathsf{p}\neq\mathsf{play}(\beta)$$

$$[\textsc{Inside-In}]\ \frac{\mathsf{G} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathscr{M}'}{\mathsf{pq}?\lambda.\mathsf{G} \parallel \langle \mathsf{q},\lambda,\mathsf{p}\rangle\cdot\mathscr{M} \xrightarrow{\beta} \mathsf{pq}?\lambda.\mathsf{G}' \parallel \langle \mathsf{q},\lambda,\mathsf{p}\rangle\cdot\mathscr{M}'}\ \mathsf{p}\neq\mathsf{play}(\beta)$$

Figure 10: LTS for type configurations.

overall behaviour of a session (see Subject Reduction and Session Fidelity theorems), we equip type configurations with an LTS, as formally defined in Figure 10. Actually we are interested in reducing only type configurations $\mathsf{G} \parallel \mathscr{M}$ such that $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ for some $\mathscr{P}$ and $\mathbb{N}$. This justifies the shapes of message queues in Rules [Inside-Out] and [Inside-In], which mimic the message queues in Rules [Out] and [In], see Figure 4. The condition $\mathsf{p}\neq\mathsf{play}(\beta)$ in these rules ensures that β is independent of the enclosing communication.

4 Properties of Typable Sessions

We begin with a few technical lemmas enabling to prove Subject reduction and Session Fidelity, that is completeness and correctness, respectively, of type configurations with respect to sessions (by taking into account participants in $\mathscr{P}$ only). These in turn will enable us to prove partial communication properties for typable sessions.

A first lemma immediately follows by cases on the typing axioms/rules.

Lemma 4.1 *If* $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$, *then* $(\mathsf{Plays}(\mathbb{N}) \setminus \mathsf{Plays}(\mathsf{G})) \cap \mathscr{P} = \emptyset$ *and* $\mathsf{G} \parallel \mathscr{M}$ *is* $\mathscr{P}$*-sound.*

The following lemma allows to get rid of histories in particular derivations. It states that, if a judgement occurs in a proof whose conclusion is without history, then the judgement itself holds without

history. Moreover, if the premises of Rules [OUT] and [IN] hold without histories, also the conclusion holds without history.

Lemma 4.2

1. *If* $\mathscr{H} \vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ *occurs in the proof of* $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$, *then* $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$.
2. *If* $\vdash_{\mathscr{P}} \mathsf{p}[\![P_i]\!] \parallel \mathbb{N} \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q} \rangle : \mathsf{G}_i$ *for all* $i \in I$, *then* $\vdash_{\mathscr{P}} \mathsf{p}[\![\mathsf{q}!\{\lambda_i.P_i\}_{i\in I}]\!] \parallel \mathbb{N} \parallel \mathscr{M} : \mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I}$.
3. *If* $\vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ *and* $h \in I$, *then* $\vdash_{\mathscr{P}} \mathsf{p}[\![\mathsf{q}?\{\lambda_i.P_i\}_{i\in I}]\!] \parallel \mathbb{N} \parallel \langle \mathsf{q}, \lambda_h, \mathsf{p} \rangle \cdot \mathscr{M} : \mathsf{pq}?\lambda_h.\mathsf{G}$.

Proof. 1. By induction on the distance d between $\mathscr{H} \vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ and $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$ in the derivation of $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$. The case $d = 0$ is trivial.

Case $d = 1$. Then $\mathscr{H} \vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ is a premise of a rule whose conclusion is $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$, which implies $\mathscr{H} = (\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}')$. We can now build a derivation of $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ out of the derivation of $(\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}') \vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$, as follows. First we erase everywhere $(\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}')$ from the histories present in the derivation. This operation does not affect the correctness of the applicability conditions of Axiom [END] and Rules [IN] and [OUT]. Axiom [CYCLE], instead, is affected by such an erasing only in case $(\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}')$ is the triple used in the axiom, namely $\mathscr{H}', (\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}') \vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$ is the axiom conclusion. In such a case, we replace this application of Axiom [CYCLE] by a proof of $\mathscr{H}' \vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$ built out of the derivation $\mathscr{D}$ of $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$ in the following way.
Let us consider the premises of the last rule in the derivation $\mathscr{D}$. For the premises which are axioms there is nothing to do. For the other premises we need to modify the derivation as follows.
Let $(\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}') \vdash_{\mathscr{P}} \widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}} : \widehat{\mathsf{G}}$ be obtained as conclusion of either Rule [IN] or Rule [OUT] with premises having $(\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}'), (\widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}}, \widehat{\mathsf{G}})$ as histories. $\mathscr{D}$ has hence the form

$$\dfrac{\cdots \quad \dfrac{\cdots \quad (\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}'), (\widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}}, \widehat{\mathsf{G}}) \vdash_{\mathscr{P}} \_ : \_ \quad \cdots}{(\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}') \vdash_{\mathscr{P}} \widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}} : \widehat{\mathsf{G}}}\,[\text{IN}]/[\text{OUT}] \quad \cdots}{\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'}\,[\text{IN}]/[\text{OUT}]$$

Notice that this implies that $(\widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}}, \widehat{\mathsf{G}}) \in \mathscr{H}'$. We can hence transform the above derivation in a derivation of $\mathscr{H}' \vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$ as follows:

$$\dfrac{\cdots \quad \dfrac{}{\mathscr{H}', (\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}') \vdash_{\mathscr{P}} \widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}} : \widehat{\mathsf{G}}}\,[\text{CYCLE}] \quad \cdots}{\mathscr{H}' \vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'}\,[\text{IN}]/[\text{OUT}]$$

Case $d > 1$. Let $(\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}') \vdash_{\mathscr{P}} \widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}} : \widehat{\mathsf{G}}$ be an arbitrary premise of a rule whose conclusion is $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$. By the construction described in the base case, we can get a derivation $\mathscr{D}$ for $\vdash_{\mathscr{P}} \widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}} : \widehat{\mathsf{G}}$ containing a subderivation for $\mathscr{H} \setminus \{(\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}')\} \vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$. In $\mathscr{D}$ the distance between $\vdash_{\mathscr{P}} \widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}} : \widehat{\mathsf{G}}$ and $\mathscr{H} \setminus \{(\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}')\} \vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ is $d-1$. So, by the induction hypothesis, we conclude $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$.

2. Let $\mathsf{G} = \mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I}$ and $\mathbb{N}' = \mathsf{p}[\![\mathsf{q}!\{\lambda_i.P_i\}_{i\in I}]\!] \parallel \mathbb{N}$. If a statement $\mathscr{H} \vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}$ does not occur in the derivation of $\vdash_{\mathscr{P}} \mathsf{p}[\![P_i]\!] \parallel \mathbb{N} \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q} \rangle : \mathsf{G}_i$, then we can simply add $(\mathbb{N}' \parallel \mathscr{M}, \mathsf{G})$ to the histories of the derivation, so getting a still correct derivation, and then apply Rule [OUT]. Otherwise this statement must also be the conclusion of an application of Rule [OUT] with premises $\mathscr{H}, (\mathbb{N}' \parallel \mathscr{M}', \mathsf{G}) \vdash_{\mathscr{P}} \mathsf{p}[\![P_i]\!] \parallel \mathbb{N} \parallel \mathscr{M}' \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q} \rangle : \mathsf{G}_i$ for all $i \in I$. This implies $\mathscr{M}' \equiv \mathscr{M}$. Since $\mathscr{H} \vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M} : \mathsf{G}$ occurs in a derivation of $\vdash_{\mathscr{P}} \mathsf{p}[\![P_i]\!] \parallel \mathbb{N} \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q} \rangle : \mathsf{G}_i$, by Point (1) we conclude $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M} : \mathsf{G}$.

3. Let $\mathsf{G}' = \mathsf{pq}?\lambda_h.\mathsf{G}$ and $\mathbb{N}' = \mathsf{p}[\![\mathsf{q}?\{\lambda_i.P_i\}_{i\in I}]\!] \parallel \mathbb{N}$ and $\mathscr{M}' \equiv \langle \mathsf{q}, \lambda_h, \mathsf{p}\rangle \cdot \mathscr{M}$. If the derivation of $\vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ does not contain a statement $\mathscr{H} \vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}'' : \mathsf{G}'$, then we can simply add $(\mathbb{N}' \parallel \mathscr{M}, \mathsf{G}')$ to the histories of the derivation, so getting a still correct derivation, and then apply Rule [In]. Otherwise this statement must also be the conclusion of an application of Rule [In] with premise $\mathscr{H}, (\mathbb{N}' \parallel \mathscr{M}'', \mathsf{G}') \vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$. This implies $\mathscr{M}'' \equiv \mathscr{M}'$. Since $\mathscr{H} \vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$ occurs in a derivation of $\vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$, by Point (1) we conclude $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$. □

We can now show that the reductions of type configurations are matched by the reductions of the sessions.

Theorem 4.3 (Session Fidelity) *If* $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ *and* $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathscr{M}'$, *then* $\mathbb{N} \parallel \mathscr{M} \xrightarrow{\beta} \mathbb{N}' \parallel \mathscr{M}'$ *and* $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$.

Proof. The proof is by cases on the last applied axiom/rule in the derivation of $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$.

Axiom [End]. Impossible since $\mathsf{G} = \mathsf{End}$ and $\mathsf{End} \parallel \mathscr{M} \not\to$.

Axiom [Cycle]. Impossible since the history cannot be empty.

Rule [Out]. In such a case $\mathsf{G} = \mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I}$ and $\mathbb{N} = \mathsf{p}[\![P]\!] \parallel \widehat{\mathbb{N}}$ and $P = \mathsf{q}!\{\lambda_i.P_i\}_{i\in I}$ and

$$(\mathsf{p}[\![P]\!] \parallel \widehat{\mathbb{N}} \parallel \mathscr{M}, \mathsf{G}) \vdash_{\mathscr{P}} \mathsf{p}[\![P_i]\!] \parallel \widehat{\mathbb{N}} \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle : \mathsf{G}_i \quad \forall i \in I \tag{1}$$

We proceed by induction on the height t of the derivation of $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathscr{M}'$.

Case $t = 1$. Then $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathscr{M}'$ is necessarily obtained by Axiom [Top-Out], that is $\beta = \mathsf{pq}!\lambda_h$ for some $h \in I$, and

$$[\text{Top-Out}]\ \frac{}{\mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I} \parallel \mathscr{M} \xrightarrow{\mathsf{pq}!\lambda_h} \mathsf{G}_h \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_h, \mathsf{q}\rangle}\ h \in I$$

By Rule [Send] we have that

$$\mathsf{p}[\![P]\!] \parallel \widehat{\mathbb{N}} \parallel \mathscr{M} \xrightarrow{\mathsf{pq}!\lambda_h} \mathsf{p}[\![P_h]\!] \parallel \widehat{\mathbb{N}} \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_h, \mathsf{q}\rangle$$

Now, by (1) we have $(\mathsf{p}[\![P]\!] \parallel \mathbb{N} \parallel \mathscr{M}, \mathsf{G}) \vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \mathbb{N} \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_h, \mathsf{q}\rangle : \mathsf{G}_h$, and by Lemma 4.2(1) we get the rest of the thesis, namely $\vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \widehat{\mathbb{N}} \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_h, \mathsf{q}\rangle : \mathsf{G}_h$.

Case $t > 1$. Then $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathscr{M}'$ is necessarily obtained by Rule [Inside-Out], that is

$$[\text{Inside-Out}]\ \frac{\mathsf{G}_i \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle \xrightarrow{\beta} \mathsf{G}_i' \parallel \mathscr{M}' \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle \quad \forall i \in I}{\mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{pq}!\{\lambda_i.\mathsf{G}_i'\}_{i\in I} \parallel \mathscr{M}'}\ \mathsf{p} \neq \mathsf{play}(\beta)$$

From (1) and Lemma 4.2(1) we can infer that

$$\vdash_{\mathscr{P}} \mathsf{p}[\![P_i]\!] \parallel \widehat{\mathbb{N}} \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle : \mathsf{G}_i \quad \forall i \in I$$

We can now recur to the induction hypothesis, getting

$$\mathsf{p}[\![P_i]\!] \parallel \widehat{\mathbb{N}} \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle \xrightarrow{\beta} \mathsf{p}[\![P_i]\!] \parallel \widehat{\mathbb{N}}' \parallel \mathscr{M}' \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle \qquad \forall i \in I$$

and

$$\vdash_{\mathscr{P}} \mathsf{p}[\![P_i]\!] \parallel \widehat{\mathbb{N}}' \parallel \mathscr{M}' \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle : \mathsf{G}_i' \qquad \forall i \in I$$

Notice that the condition $\mathsf{p} \neq \mathsf{play}(\beta)$ ensures that, for each $i \in I$, the transition does not modify the process of participant p. Moreover, the transition does not depend on the messages $\langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle$, since these messages are at the end of the queue both before and after the transitions. So, we can infer that

$$\mathsf{p}[\![P]\!] \parallel \widehat{\mathbb{N}} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{p}[\![P]\!] \parallel \widehat{\mathbb{N}}' \parallel \mathscr{M}'$$

Lemma 4.2(2) applied to

$$\vdash_{\mathscr{P}} \mathsf{p}[\![P_i]\!] \parallel \widehat{\mathbb{N}}' \parallel \mathscr{M}' \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q} \rangle : \mathsf{G}'_i \text{ for all } i \in I$$

gives $\vdash_{\mathscr{P}} \mathsf{p}[\![P]\!] \parallel \widehat{\mathbb{N}}' \parallel \mathscr{M}' : \mathsf{G}'$.

Rule [IN]. In such a case $\mathsf{G} = \mathsf{pq}?\lambda_h.\mathsf{G}'$ and $\mathbb{N} = \mathsf{p}[\![P]\!] \parallel \widehat{\mathbb{N}}$ and $\mathscr{M} \equiv \langle \mathsf{q}, \lambda_h, \mathsf{p} \rangle \cdot \widehat{\mathscr{M}}$, with $h \in I$ and $P = \mathsf{q}?\{\lambda_i.P_i\}_{i \in I}$ and

$$(\mathsf{p}[\![P]\!] \parallel \widehat{\mathbb{N}} \parallel \mathscr{M}, \mathsf{G}) \vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}} : \mathsf{G}' \tag{2}$$

We proceed by induction on the height t of the derivation of $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathscr{M}'$.

Case $t = 1$. Then $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathscr{M}'$ is necessarily obtained by Axiom [TOP-IN], that is $\beta = \mathsf{pq}?\lambda_h$ and

$$[\text{TOP-IN}]\ \frac{}{\mathsf{pq}?\lambda_h.\mathsf{G}' \parallel \langle \mathsf{q}, \lambda_h, \mathsf{p} \rangle \cdot \widehat{\mathscr{M}} \xrightarrow{\mathsf{pq}?\lambda_h} \mathsf{G}' \parallel \widehat{\mathscr{M}}}$$

By Rule [RCV] we have that

$$\mathsf{p}[\![P]\!] \parallel \widehat{\mathbb{N}} \parallel \langle \mathsf{q}, \lambda_h, \mathsf{p} \rangle \cdot \widehat{\mathscr{M}} \xrightarrow{\mathsf{pq}?\lambda_h} \mathsf{p}[\![P_h]\!] \parallel \widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}}$$

We can now get the rest of the thesis since (2) and Lemma 4.2(1) imply

$$\vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}} : \mathsf{G}'$$

Case $t > 1$. Then $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathscr{M}'$ is necessarily obtained by Rule [INSIDE-IN], that is

$$[\text{INSIDE-IN}]\ \frac{\mathsf{G}' \parallel \widehat{\mathscr{M}} \xrightarrow{\beta} \mathsf{G}'' \parallel \widehat{\mathscr{M}'}}{\mathsf{pq}?\lambda_h.\mathsf{G}' \parallel \langle \mathsf{q}, \lambda_h, \mathsf{p} \rangle \cdot \widehat{\mathscr{M}} \xrightarrow{\beta} \mathsf{pq}?\lambda_h.\mathsf{G}'' \parallel \langle \mathsf{q}, \lambda_h, \mathsf{p} \rangle \cdot \widehat{\mathscr{M}'}}\quad \mathsf{p} \neq \mathsf{play}(\beta)$$

Now, (2) and Lemma 4.2(1) imply

$$\vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}} : \mathsf{G}'$$

We can hence recur to the induction hypothesis, getting that

$$\mathsf{p}[\![P_h]\!] \parallel \widehat{\mathbb{N}} \parallel \widehat{\mathscr{M}} \xrightarrow{\beta} \mathsf{p}[\![P_h]\!] \parallel \widehat{\mathbb{N}}' \parallel \widehat{\mathscr{M}'}$$

and

$$\vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \widehat{\mathbb{N}}' \parallel \widehat{\mathscr{M}'} : \mathsf{G}''$$

since the side condition $\mathsf{p} \neq \mathsf{play}(\beta)$ of Rule [INSIDE-IN] implies that the process of participant p is unchanged. Lemma 4.2(3) applied to

$$\vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \widehat{\mathbb{N}}' \parallel \widehat{\mathscr{M}'} : \mathsf{G}''$$

gives the rest of the thesis, namely

$$\vdash_{\mathscr{P}} \mathsf{p}[\![P]\!] \parallel \widehat{\mathbb{N}}' \parallel \langle \mathsf{q}, \lambda_h, \mathsf{p} \rangle \cdot \widehat{\mathscr{M}'} : \mathsf{pq}?\lambda_h.\mathsf{G}''$$ □

The proof of Subject Reduction requires some lemmas which are typical of our partial typing. The first lemma deals with participants which have active processes but are not players of global types. The

second lemma deals with messages whose receivers are not players of global types. The last lemma states that a player of the network whose lock-freedom must be ensured is always a player of the global type.

Lemma 4.4 *If* $\vdash_{\mathscr{P}} \mathsf{p}[\![P]\!] \parallel \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$, $P \neq \mathbf{0}$ *and* $\mathsf{p} \notin \mathsf{Plays}(\mathsf{G})$, *then* $\vdash_{\mathscr{P}} \mathsf{p}[\![P']\!] \parallel \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ *for any arbitrary* P' *such that* $\mathsf{p} \notin \mathsf{Prt}(P')$.

Proof. If $\mathsf{p} \notin \mathsf{Plays}(\mathsf{G})$, then the process P can never be involved in any occurrence of Rules [Out] or [In]. This implies that $\mathsf{p}[\![P]\!]$ must occur only in axioms. It is hence enough to replace P by P' in those axioms and modify the histories present in the derivation accordingly. □

Lemma 4.5 *If* $\vdash_{\mathscr{P}} \mathbb{N} \parallel \langle \mathsf{q}, \lambda, \mathsf{p} \rangle \cdot \mathscr{M} : \mathsf{G}$ *and* $\mathsf{p} \notin \mathsf{Plays}(\mathsf{G})$, *then* $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$.

Proof. Rule [Out] does not add messages to the queue. If $\mathsf{p} \notin \mathsf{Plays}(\mathsf{G})$, then $\langle \mathsf{q}, \lambda, \mathsf{p} \rangle$ cannot be added by Rule [In]. Then $\langle \mathsf{q}, \lambda, \mathsf{p} \rangle$ is present in all the queues of the judgements in the derivation. We remark that the removal of a message from a queue cannot alter the truth value of the $\mathscr{P}$-soundness condition, which is required for the applicability of an axiom or a rule. It is hence possible to remove $\langle \mathsf{q}, \lambda, \mathsf{p} \rangle$ from the queues in the axioms and modify the queues present in the derivation accordingly. □

Lemma 4.6 *If* $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ *and* $\mathsf{p} \in (\mathsf{Plays}(\mathbb{N}) \cap \mathscr{P})$, *then* $\mathsf{p} \in \mathsf{Plays}(\mathsf{G})$.

Proof. If $\mathsf{p} \in \mathscr{P}$, then an output with sender p can only be typed by Rule [Out] and an input with receiver p together with a message with receiver p can only be typed by Rule [In]. □

Subject Reduction ensures that a transition of a session is mimicked by a transition of the corresponding type configuration only if the player of the transition is a player of the global type.

Theorem 4.7 (Subject Reduction) *Let* $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ *and* $\mathbb{N} \parallel \mathscr{M} \xrightarrow{\beta} \mathbb{N}' \parallel \mathscr{M}'$. *If* $\mathsf{play}(\beta) \in \mathsf{Plays}(\mathsf{G})$, *then* $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathscr{M}'$ *and* $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$. *Otherwise* $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}$.

Proof. The proof is by cases on the reduction rules.

Rule [Send]. In this case

$$\mathbb{N} \equiv \mathsf{p}[\![\mathsf{q}!\{\lambda_i.P_i\}_{i\in I}]\!] \parallel \mathbb{N}_0, \quad \mathscr{M}' \equiv \mathscr{M} \cdot \langle \mathsf{p}, \lambda_h, \mathsf{q} \rangle, \quad \beta = \mathsf{pq}!\lambda_h, \quad \mathbb{N}' \equiv \mathsf{p}[\![P_h]\!] \parallel \mathbb{N}_0 \quad \text{where } h \in I.$$

By definition of network $\mathsf{p} \notin \mathsf{Prt}(\mathsf{q}!\{\lambda_i.P_i\}_{i\in I})$, which implies $\mathsf{p} \notin \mathsf{Prt}(P_h)$. If $\mathsf{p} \notin \mathsf{Plays}(\mathsf{G})$, Lemma 4.4 implies $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}$. Otherwise the proof proceeds by cases on the last typing axiom/rule used in the derivation for $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ and by induction on $d = \mathsf{depth}(\mathsf{G}, \mathsf{p})$.

Axiom [End]. Since it cannot be $\mathsf{play}(\beta) \in \mathsf{Plays}(\mathsf{G})$, the implication is vacuously satisfied.

Axiom [Cycle]. Impossible since the history cannot be empty.

Rule [Out] *and* $d = 1$. In this case $\mathsf{G} = \mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I}$. We get $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\mathsf{pq}!\lambda_h} \mathsf{G}_h \parallel \mathscr{M}'$ by Axiom [Top-Out]. Lemma 4.2(1) implies

$$\vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \mathbb{N}_0 \parallel \mathscr{M} \cdot \langle \mathsf{p}, \lambda_h, \mathsf{q} \rangle : \mathsf{G}_h$$

Rule [Out] *and* $d > 1$. In this case $\mathsf{G} = \mathsf{rs}!\{\lambda'_j.\mathsf{G}_j\}_{j\in J}$ with $\mathsf{r} \neq \mathsf{p}$ and $\mathbb{N}_0 \equiv \mathsf{r}[\![\mathsf{s}!\{\lambda'_j.R_j\}_{j\in J}]\!] \parallel \mathbb{N}_1$. Lemma 4.2(1) implies

$$\vdash_{\mathscr{P}} \mathsf{p}[\![\mathsf{q}!\{\lambda_i.P_i\}_{i\in I}]\!] \parallel \mathsf{r}[\![R_j]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M} \cdot \langle \mathsf{r}, \lambda'_j, \mathsf{s} \rangle : \mathsf{G}_j \quad \text{for all } j \in J$$

We hence get, by Rule [Send], for all $j \in J$,

$$\mathsf{p}[\![\,\mathsf{q}!\{\lambda_i.P_i\}_{i\in I}\,]\!] \parallel \mathsf{r}[\![\,R_j\,]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M}\cdot\langle \mathsf{r},\lambda'_j,\mathsf{s}\rangle \xrightarrow{\mathsf{pq}!\lambda_h} \mathsf{p}[\![\,P_h\,]\!] \parallel \mathsf{r}[\![\,R_j\,]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M}\cdot\langle \mathsf{r},\lambda'_j,\mathsf{s}\rangle\cdot\langle \mathsf{p},\lambda_h,\mathsf{q}\rangle$$

Since $\mathsf{depth}(\mathsf{G}_j,\mathsf{p}) < d$, induction implies $\mathsf{G}_j \parallel \mathscr{M}\cdot\langle \mathsf{r},\lambda'_j,\mathsf{s}\rangle \xrightarrow{\mathsf{pq}!\lambda_h} \mathsf{G}'_j \parallel \mathscr{M}\cdot\langle \mathsf{r},\lambda'_j,\mathsf{s}\rangle\cdot\langle \mathsf{p},\lambda_h,\mathsf{q}\rangle$ and

$$\vdash_{\mathscr{P}} \mathsf{p}[\![\,P_h\,]\!] \parallel \mathsf{r}[\![\,R_j\,]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M}\cdot\langle \mathsf{r},\lambda'_j,\mathsf{s}\rangle\cdot\langle \mathsf{p},\lambda_h,\mathsf{q}\rangle : \mathsf{G}'_j \quad \text{for all } j\in J$$

Let $\mathsf{G}' = \mathsf{rs}!\{\lambda'_j.\mathsf{G}'_j\}_{j\in J}$ and $\mathscr{M}' = \mathscr{M}\cdot\langle \mathsf{p},\lambda_h,\mathsf{q}\rangle$. Since the messages $\langle \mathsf{r},\lambda'_j,\mathsf{s}\rangle$ and $\langle \mathsf{p},\lambda_h,\mathsf{q}\rangle$ commute, being $\mathsf{r}\neq\mathsf{p}$, we can derive $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\mathsf{pq}!\lambda_h} \mathsf{G}' \parallel \mathscr{M}'$ using Rule [INSIDE-OUT]. Lastly, $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$ by Lemma 4.2(2).
Rule [IN] *and* $d = 1$. Impossible.
Rule [IN] *and* $d > 1$. In this case

$$\mathsf{G} = \mathsf{rs}?\lambda'_k.\mathsf{G}'' \text{ with } \mathsf{r}\neq\mathsf{p} \text{ and } \mathbb{N}_0 \equiv \mathsf{r}[\![\,\mathsf{s}?\{\lambda'_j.R_j\}_{j\in J}\,]\!] \parallel \mathbb{N}_1 \text{ and } \mathscr{M} \equiv \langle \mathsf{s},\lambda'_k,\mathsf{r}\rangle\cdot\mathscr{M}_0 \text{ with } k\in J.$$

Moreover, $\vdash_{\mathscr{P}} \mathsf{p}[\![\,\mathsf{q}!\{\lambda_i.P_i\}_{i\in I}\,]\!] \parallel \mathsf{r}[\![\,R_k\,]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M} : \mathsf{G}''$ by Lemma 4.2(1). We get

$$\mathsf{p}[\![\,\mathsf{q}!\{\lambda_i.P_i\}_{i\in I}\,]\!] \parallel \mathsf{r}[\![\,R_k\,]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M} \xrightarrow{\mathsf{pq}!\lambda_h} \mathsf{p}[\![\,P_h\,]\!] \parallel\parallel \mathsf{r}[\![\,R_k\,]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M}\cdot\langle \mathsf{p},\lambda_h,\mathsf{q}\rangle$$

Since $\mathsf{depth}(\mathsf{G}'',\mathsf{p}) < d$, induction implies

$$\mathsf{G}'' \parallel \mathscr{M} \xrightarrow{\mathsf{pq}!\lambda_h} \mathsf{G}''' \parallel \mathscr{M}\cdot\langle \mathsf{p},\lambda_h,\mathsf{q}\rangle \quad \text{and} \quad \vdash_{\mathscr{P}} \mathsf{p}[\![\,P_h\,]\!] \parallel \mathsf{r}[\![\,R_k\,]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M}\cdot\langle \mathsf{p},\lambda_h,\mathsf{q}\rangle : \mathsf{G}'''$$

Let $\mathsf{G}' = \mathsf{rs}?\lambda'_k.\mathsf{G}'''$. Being $\mathsf{r}\neq\mathsf{p}$ we can derive $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\mathsf{pq}!\lambda_h} \mathsf{G}' \parallel \mathscr{M}'$ using Rule [INSIDE-IN]. Lastly, $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$ by Lemma 4.2(3).

Rule [RCV]. In this case

$$\mathbb{N} \equiv \mathsf{p}[\![\,\mathsf{q}?\{\lambda_i.P_i\}_{i\in I}\,]\!] \parallel \mathbb{N}_0, \quad \mathscr{M} \equiv \langle \mathsf{q},\lambda_h,\mathsf{p}\rangle\cdot\mathscr{M}', \quad \beta = \mathsf{pq}?\lambda_h, \quad \mathbb{N}' \equiv \mathsf{p}[\![\,P_h\,]\!] \parallel \mathbb{N}_0 \quad \text{where } h\in I.$$

By definition of network $\mathsf{p}\notin\mathsf{Prt}(\mathsf{q}?\{\lambda_i.P_i\}_{i\in I})$, which implies $\mathsf{p}\notin\mathsf{Prt}(P_h)$. If $\mathsf{p}\notin\mathsf{Prt}(\mathsf{G})$ Lemmas 4.4 and 4.5 imply $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}$. Otherwise the proof proceeds by cases on the last axiom/rule used in the derivation for $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ and by induction on $d = \mathsf{depth}(\mathsf{G},\mathsf{p})$.
Axiom [END]. Since it cannot be $\mathsf{play}(\beta)\in\mathsf{Prt}(\mathsf{G})$, the implication is vacuously satisfied.
Axiom [CYCLE]. Impossible since the history cannot be empty.
Rule [OUT] *and* $d = 1$. Impossible.
Rule [OUT] *and* $d > 1$. In this case $\mathsf{G} = \mathsf{rs}!\{\lambda'_j.\mathsf{G}_j\}_{j\in J}$ with $\mathsf{r}\neq\mathsf{p}$ and $\mathbb{N}_0 \equiv \mathsf{r}[\![\,\mathsf{s}!\{\lambda'_j.R_j\}_{j\in J}\,]\!] \parallel \mathbb{N}_1$ and $\vdash_{\mathscr{P}} \mathsf{p}[\![\,\mathsf{q}?\{\lambda_i.P_i\}_{i\in I}\,]\!] \parallel \mathsf{r}[\![\,R_j\,]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M}\cdot\langle \mathsf{r},\lambda'_j,\mathsf{s}\rangle : \mathsf{G}_j$ for all $j\in J$ by Lemma 4.2(1). We get, for all $j\in J$,

$$\mathsf{p}[\![\,\mathsf{q}?\{\lambda_i.P_i\}_{i\in I}\,]\!] \parallel \mathsf{r}[\![\,R_j\,]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M}\cdot\langle \mathsf{r},\lambda'_j,\mathsf{s}\rangle \xrightarrow{\mathsf{pq}?\lambda_h} \mathsf{p}[\![\,P_h\,]\!] \parallel \mathsf{r}[\![\,R_j\,]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M}'\cdot\langle \mathsf{r},\lambda'_j,\mathsf{s}\rangle$$

Since $\mathsf{depth}(\mathsf{G}_j,\mathsf{p}) < d$, induction implies, for all $j\in J$,

$$\mathsf{G}_j \parallel \mathscr{M}\cdot\langle \mathsf{r},\lambda'_j,\mathsf{s}\rangle \xrightarrow{\mathsf{pq}?\lambda_h} \mathsf{G}'_j \parallel \mathscr{M}'\cdot\langle \mathsf{r},\lambda'_j,\mathsf{s}\rangle \quad \text{and} \quad \vdash_{\mathscr{P}} \mathsf{p}[\![\,P_h\,]\!] \parallel \mathsf{r}[\![\,R_j\,]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M}'\cdot\langle \mathsf{r},\lambda'_j,\mathsf{s}\rangle : \mathsf{G}'_j$$

Let $\mathsf{G}' = \mathsf{rs}!\{\lambda'_j.\mathsf{G}'_j\}_{j\in J}$. Being $\mathsf{r}\neq\mathsf{p}$ we can derive $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\mathsf{pq}?\lambda_h} \mathsf{G}' \parallel \mathscr{M}'$ using Rule [INSIDE-OUT]. Lastly, $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$ by Lemma 4.2(2).
Rule [IN] *and* $d = 1$. In this case $\mathsf{G} = \mathsf{pq}?\lambda.\mathsf{G}'$. Lemma 4.2(1) implies $\vdash_{\mathscr{P}} \mathsf{p}[\![\,P_h\,]\!] \parallel \mathbb{N}_0 \parallel \mathscr{M}' : \mathsf{G}'$. We get $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\mathsf{pq}?\lambda_h} \mathsf{G}' \parallel \mathscr{M}'$ by Axiom [TOP-IN].
Rule [IN] *and* $d > 1$. In this case

$$\mathsf{G} = \mathsf{rs}?\lambda'_k.\mathsf{G}'' \quad \mathbb{N}_0 \equiv \mathsf{r}[\![\,\mathsf{s}?\{\lambda'_j.R_j\}_{j\in J}\,]\!] \parallel \mathbb{N}_1, \quad \mathscr{M}' \equiv \langle \mathsf{s},\lambda'_k,\mathsf{r}\rangle\cdot\mathscr{M}_0 \quad \text{with } \mathsf{r}\neq\mathsf{p} \text{ and } k\in J$$

Lemma 4.2(1) implies $\vdash_{\mathscr{P}} \mathsf{p}[\![\mathsf{q}?\{\lambda_i.P_i\}_{i\in I}]\!] \parallel \mathsf{r}[\![R_k]\!] \parallel \mathbb{N}_1 \parallel \langle \mathsf{q},\lambda_h,\mathsf{p}\rangle \cdot \mathscr{M}_0 : \mathsf{G}''$. We get

$$\mathsf{p}[\![\mathsf{q}?\{\lambda_i.P_i\}_{i\in I}]\!] \parallel \mathsf{r}[\![R_k]\!] \parallel \mathbb{N}_1 \parallel \langle \mathsf{q},\lambda_h,\mathsf{p}\rangle \cdot \mathscr{M}_0 \xrightarrow{\mathsf{pq}?\lambda_h} \mathsf{p}[\![P_h]\!] \parallel \mathsf{r}[\![R_k]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M}_0$$

Since $\mathsf{depth}(\mathsf{G}'',\mathsf{p}) < d$, induction implies

$$\mathsf{G}'' \parallel \langle \mathsf{q},\lambda_h,\mathsf{p}\rangle \cdot \mathscr{M}_0 \xrightarrow{\mathsf{pq}?\lambda_h} \mathsf{G}''' \parallel \mathscr{M}_0 \quad \text{and} \quad \vdash_{\mathscr{P}} \mathsf{p}[\![P_h]\!] \parallel \mathsf{r}[\![R_k]\!] \parallel \mathbb{N}_1 \parallel \mathscr{M}_0 : \mathsf{G}'''$$

Let $\mathsf{G}' = \mathsf{rs}?\lambda'_k.\mathsf{G}'''$. Being $\mathsf{r} \neq \mathsf{p}$ we can derive $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\mathsf{pq}?\lambda_h} \mathsf{G}' \parallel \mathscr{M}'$ using Rule [INSIDE-IN]. Lastly, $\vdash_{\mathscr{P}} \mathbb{N}' \parallel \mathscr{M}' : \mathsf{G}'$ by Lemma 4.2(3). □

We conclude this section by showing the main properties of our type system: partial lock-freedom and partial orphan-message-freedom.

Theorem 4.8 (Partial Lock-freedom) *If $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$, then $\mathbb{N} \parallel \mathscr{M}$ is $\mathscr{P}$-lock free.*

Proof. Let $\mathsf{p} \in \mathscr{P}$. If $\mathsf{p} \notin \mathsf{Plays}(\mathbb{N})$, then $\mathbb{N} \parallel \mathscr{M}$ is trivially p-lock free. Otherwise $\mathsf{p} \in (\mathsf{Plays}(\mathbb{N}) \cap \mathscr{P})$ gives $\mathsf{p} \in \mathsf{Plays}(\mathsf{G})$ by Lemma 4.6. We first show by induction on $d = \mathsf{depth}(\mathsf{G},\mathsf{p})$ that $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\tau\cdot\beta}$ with $\mathsf{play}(\beta) = \mathsf{p}$ for some τ, β.

If $d = 1$, then either $\mathsf{G} = \mathsf{pq}!\{\lambda_i.\mathsf{G}_i\}_{i\in I}$ or $\mathsf{G} = \mathsf{pq}?\lambda.\mathsf{G}'$. We get $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\beta}$ with $\mathsf{play}(\beta) = \mathsf{p}$ by either Axiom [TOP-OUT] or Axiom [TOP-IN].

If $d > 1$, then $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\beta'} \mathsf{G}' \parallel \mathscr{M}'$ for some β', G' and $\mathscr{M}'$ by Axiom [TOP-OUT] or Axiom [TOP-IN]. The applicability of Axiom [TOP-IN] is ensured by the fact that $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ must be typed using Rule [IN]. Since $\mathsf{depth}(\mathsf{G}',\mathsf{p}) < d$, by induction $\mathsf{G}' \parallel \mathscr{M}' \xrightarrow{\tau'\cdot\beta}$ with $\mathsf{play}(\beta) = \mathsf{p}$ for some τ, β. We can take $\tau = \beta' \cdot \tau'$.

By Theorem 4.3 $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\tau\cdot\beta}$ implies $\mathbb{N} \parallel \mathscr{M} \xrightarrow{\tau\cdot\beta}$. □

Theorem 4.9 (Partial Orphan-message-freedom) *If $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$, then $\mathbb{N} \parallel \mathscr{M}$ is $\mathscr{P}$-orphan-message free.*

Proof. Let $\mathscr{M} \equiv \langle \mathsf{p},\lambda,\mathsf{q}\rangle \cdot \widehat{\mathscr{M}}$ and $\{\mathsf{p},\mathsf{q}\} \subseteq \mathscr{P}$. We first show that $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\tau\cdot\mathsf{qp}?\lambda}$ by induction on $\mathsf{weight}(\mathsf{G},\langle \mathsf{p},\lambda,\mathsf{q}\rangle)$. If $\mathsf{weight}(\mathsf{G},\langle \mathsf{p},\lambda,\mathsf{q}\rangle) = 0$ it is trivial. Otherwise $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathscr{M}'$ by Axiom [TOP-OUT] or Axiom [TOP-IN] and $\mathsf{weight}(\mathsf{G}',\langle \mathsf{p},\lambda,\mathsf{q}\rangle) < \mathsf{weight}(\mathsf{G},\langle \mathsf{p},\lambda,\mathsf{q}\rangle)$. The applicability of Axiom [TOP-IN] is ensured by the fact that $\vdash_{\mathscr{P}} \mathbb{N} \parallel \mathscr{M} : \mathsf{G}$ must be typed using Rule [IN]. By induction $\mathsf{G}' \parallel \mathscr{M}' \xrightarrow{\tau'\cdot\mathsf{qp}?\lambda}$, so we can take $\tau = \beta \cdot \tau'$.

Applying Theorem 4.3 to $\mathsf{G} \parallel \mathscr{M} \xrightarrow{\tau\cdot\mathsf{qp}?\lambda}$ we conclude $\mathbb{N} \parallel \mathscr{M} \xrightarrow{\tau\cdot\mathsf{qp}?\lambda}$. □

It is worth noticing that in case we were interested in $\mathscr{P}$-lock-freedom only we could simply take out the $\mathscr{P}$-soundness conditions in the type system.

Remark 4.10 (Saving $\mathscr{P}$-soundness checks) One could avoid to have the $\mathscr{P}$-soundness condition in Rules [IN] and [OUT] in case we impose $\mathscr{M} = \emptyset$ in Axiom [CYCLE]. In fact (a) $\mathsf{G} \parallel \emptyset$ is $\mathscr{P}$-sound for any G and $\mathscr{P}$ and (b) applications of Rules [IN] and [OUT] do preserve $\mathscr{P}$-soundness. Note that requiring $\mathsf{G} \parallel \mathscr{M}$ to be $\mathscr{P}$-sound only in Axioms [CYCLE] and [END] would not work. A counterexample being the obvious derivation for

$$\vdash_{\mathscr{P}} \mathsf{p}[\![P]\!] \parallel \mathsf{q}[\![Q]\!] \parallel \langle \mathsf{q},\lambda,\mathsf{p}\rangle : \mathsf{G}$$

where $P = \mathsf{q}!\lambda.P$, $Q = \mathsf{p}?\lambda.Q$ and $\mathsf{G} = \mathsf{pq}!.\mathsf{qp}?\lambda.\mathsf{G}$.

5 Conclusions

Membership of a component to a concurrent/distributed system does not imply that the component is equivalent in rights, capabilities and properties to the other components. A system can often viewed as being formed by different and heterogeneous subsystems. Formal verification techniques and methods are usually devised to ensure properties of whole systems and they cannot always be scaled down or tailored to work on specific subsystems. This is obviously due to non trivial interactions between subsystems and the rest of system components. This issue has been addressed in [1], in the development/verification framework of MPTS. The type assignment of [1], guaranteeing good communication properties, can be in fact tailored for specific subsets of participants, so disregarding the behaviour of the rest of the participants. In the present paper we extend the investigation in [1] by considering an asynchronous model of communication, which was instead synchronous in [1]. With respect to that paper we consider, besides $\mathscr{P}$-lock-freedom (absence of locks for participants in $\mathscr{P}$), also $\mathscr{P}$-orphan-message freedom. The type assignment we devise is inspired by [3, 4, 7] where, unlike most choreographic formalisms, the asynchronicity of the communication model is explicitly reflected at the level of global-behaviour descriptions, namely the global types in our case.

A MPST formalism dealing with properties holding for partial descriptions of systems was defined in [11] and further investigated in [2, 5]. In those papers, a notion of *connecting communications* enables us to consider some participants as optional, in particular the ones that are "invited" (via connecting inputs) to join some interactions. Such a feature allows for a more natural description of typical communication protocols. Connecting communications and our partial typing are sort of orthogonal. An advantage of connecting communications over partial typing (where participants offering connecting communications should be ignored) is that only participants offering connecting inputs can be stuck. The disadvantage is that the typing rules are more demanding, so many interesting sessions can be partially typed but cannot be typed using connecting communications. We definitely deem worth investigating an extension of our formalism to deal with participants offering connecting communications.

An algorithm enabling to infer all the global types for a given session – and handling, in particular, infinite expressions as sets of recursive equations – has been devised in [1], working on a similar one in [7]. We are confident that the approach of [7], for what concerns the representation of infinite terms, can be also exploited in inference algorithms for our system.

The MPTS formalism used in the present paper, unlike many MPST formalisms stemmed from [10], does not recur to projections. Extending the standard projection operator to a relation between global types and local behaviours with good partial properties would lead to a *top-down* development/verification formalism for partial properties, i.e. where local descriptions are obtained by projecting previously developed global descriptions.

The properties verified by formalisms like the present one, as well as the ones in [1, 3, 4, 7], are strictly related to LTSs on type configurations. Such LTSs are inductively defined. It is worth considering coinductively defined LTSs, so that communication properties can be ensured for wider sets of sessions.

Acknowledgements We are grateful to the anonymous referees for their comments and suggestions to improve the readability of this paper.

References

[1] Franco Barbanera & Mariangiola Dezani-Ciancaglini (2023): *Partially Typed Multiparty Sessions*. In Clément Aubert, Cinzia Di Giusto, Simon Fowler & Larisa Safina, editors: *ICE*, *EPTCS* 383, Open Pub-

lishing Association, pp. 15–34, doi:10.4204/EPTCS.383.2.

[2] Ilaria Castellani, Mariangiola Dezani-Ciancaglini & Paola Giannini (2019): *Reversible sessions with flexible choices*. *Acta Informatica* 56(7), pp. 553–583, doi:10.1007/s00236-019-00332-y.

[3] Ilaria Castellani, Mariangiola Dezani-Ciancaglini & Paola Giannini (2021): *Global types and event structure semantics for asynchronous multiparty sessions*. *CoRR* abs/2102.00865. Available at `https://arxiv.org/abs/2102.00865`.

[4] Ilaria Castellani, Mariangiola Dezani-Ciancaglini & Paola Giannini (2022): *Asynchronous sessions with input races*. In Marco Carbone & Rumyana Neykova, editors: *PLACES*, *EPTCS* 356, Open Publishing Association, pp. 12–23, doi:10.4204/EPTCS.356.2.

[5] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, Paola Giannini & Ross Horne (2020): *Global types with internal delegation*. *Theoretical Computer Science* 807, pp. 128–153, doi:10.1016/j.tcs.2019.09.027.

[6] Bruno Courcelle (1983): *Fundamental properties of infinite trees*. *Theoretical Computer Science* 25, pp. 95–169, doi:10.1016/0304-3975(83)90059-2.

[7] Francesco Dagnino, Paola Giannini & Mariangiola Dezani-Ciancaglini (2023): *Deconfined global types for asynchronous sessions*. *Logical Methods in Computer Science* 19(1), pp. 1–41, doi:10.46298/lmcs-19(1:3)2023.

[8] Romain Demangeon & Kohei Honda (2012): *Nested protocols in session types*. In Maciej Koutny & Irek Ulidowski, editors: *CONCUR*, *LNCS* 7454, Springer, pp. 272–286, doi:10.1007/978-3-642-32940-1_20.

[9] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty asynchronous session types*. In George C. Necula & Philip Wadler, editors: *POPL*, ACM Press, pp. 273–284, doi:10.1145/1328897.1328472.

[10] Kohei Honda, Nobuko Yoshida & Marco Carbone (2016): *Multiparty asynchronous session types*. *Journal of the ACM* 63(1), pp. 9:1–9:67, doi:10.1145/2827695.

[11] Raymond Hu & Nobuko Yoshida (2017): *Explicit connection actions in multiparty session types*. In: *FASE*, *LNCS* 10202, Springer, pp. 116–133, doi:10.1007/978-3-662-54494-5.

[12] Naoki Kobayashi (2002): *A type system for lock-free processes*. *Information and Computation* 177(2), pp. 122–159, doi:10.1006/inco.2002.3171.

[13] Luca Padovani (2014): *Deadlock and lock freedom in the linear π-calculus*. In Thomas A. Henzinger & Dale Miller, editors: *CSL-LICS*, ACM Press, pp. 72:1–72:10, doi:10.1145/2603088.2603116.

[14] Benjamin C. Pierce (2002): *Types and Programming Languages*. MIT Press.

From Compactifying Lambda-Letrec Terms to Recognizing Regular-Expression Processes

(Extended Abstract and Literature)

Clemens Grabmayer
Department of Computer Science
Gran Sasso Science Institute
L'Aquila, Italy

clemens.grabmayer@gssi.it

As a supplement to my talk at the workshop, this extended abstract motivates and summarizes my work with co-authors on problems in two separate areas: first, in the λ-calculus with letrec, a universal model of computation, and second, on Milner's process interpretation of regular expressions, a proper subclass of the finite-state processes. The aim of my talk was to motivate a transferal of ideas for workable concepts of structure-constrained graphs: from the problem of finding compact graph representations for terms in the λ-calculus with letrec to the problem of recognizing finite process graphs that can be expressed by regular expressions. In both cases the construction of structure-constrained graphs was expedient in order to enable to go back and forth easily between, in the first case, λ-terms and term graphs, and in the second case, regular expressions and process graphs.

The main focus here is on providing pointers to my work with co-authors, in both areas separately. A secondary focus is on explaining directions of my present projects, and describing research questions of possibly general interest that have developed out of my work in these two areas.

1 Introduction

The purpose of this extended abstract is to supplement my talk at the workshop [3] with a brief description of my work with co-authors in two areas, including ample references. While my workshop-presentation covered similar topics as my talk [1] at TERMGRAPH 2018, and while the proceedings article [2] for that workshop remains a useful resource, this article is a rewritten account with a detailed update on results that have been obtained in the meantime, and with an outlook on remaining challenging problems.

My talk [3] at the workshop aimed at motivating a fruitful transferal of ideas between two areas on which I worked in the (a bit removed, and more recent) past: λ-calculus, and the implementation of functional programming languages (2009–2014), and the process theory of finite-state processes (2005–6, and from 2015). My intention was to show, supported by many pictures: How a solution to the problem of finding adequate graph representations for terms in the λ-calculus with letrec, a universal model of computation, turned out to be very helpful in understanding process graphs that can be expressed by regular expressions (via Milner's process interpretation), a proper subclass of finite-state processes.

In both cases the definition of an adequate notion of structure-constrained (term or process) graph was the key to solve a specific practical, and respectively, a theoretical problem. It was central that the structure-constrained graphs facilitate to go back and forth easily between, on the one hand, terms in the λ-calculus with letrec and term graphs, and on the other hand, regular expressions and process graphs. The graph representations respect the appertaining operational semantics, but were conceived with specific purposes in mind: to optimize functional programs in the Lambda Calculus with letrec; and

S. Alves and I. Mackie (Eds.): 13th International Workshop
on Developments in Computational Models 2023 (DCM'23).
EPTCS 408, 2024, pp. 21–41, doi:10.4204/EPTCS.408.2

respectively, to reason with process graphs denoted by regular expressions, and to decide recognizability of these graphs. For a detailed comparison of the similarities and differences of the structure-constrained graphs as defined in the term graph semantics of terms in the λ-calculus with letrec (see Section 2), and the process (graph) semantics of regular expressions (see Section 3), we want to refer to Section 4 of [2].

Section 2 summarizes work by Jan Rochel and myself that led us to the definition, and efficient implementation of maximal sharing for the higher-order terms in the λ-calculus with letrec. Specifically we formulated a representation-pipeline: Higher-order terms can be represented by, appropriately defined, higher-order term graphs, then these can be encoded as first-order term graphs, and subsequently those can in turn be represented as deterministic finite-state automata (DFAs). Via these correspondences and DFA minimization, maximal shared forms of higher-order terms can be computed.

Section 3 gives an overview of my work, in important parts done together with Wan Fokkink, on two non-trivial problems that concern the process semantics of regular expressions. In Milner's process semantics, regular expressions are interpreted as finite process graphs (or for that matter non-deterministic finite-state automata (NFAs)) that are viewed as equal (as describing the same 'behavior') if they are bisimilar. Unlike for the standard language interpretation, not every finite process can be expressed, in this way, by a regular expression. This fact raised a non-trivial recognition (or expressibility) problem, which was formulated by Milner (1984) next to a completeness problem for an equational proof system. In Section 3 I report on the crucial steps that have led me to a solution of the completeness problem.

Finally Section 4 reports on my present projects, and lists research questions that have developed out of my work in these two areas.

References

[1] Clemens Grabmayer (2018): *Modeling Terms by Graphs with Structure Constraints (Two Illustrations)*. Invited talk at the FSCD/FLoC Workshop *TERMGRAPH 2018*, Oxford, UK, July 7. Slides are available at `https://clegra.github.io/lf/TERMGRAPH-2018-invited-talk.pdf`.

[2] Clemens Grabmayer (2019): *Modeling Terms by Graphs with Structure Constraints (Two Illustrations)*. In Maribel Fernández & Ian Mackie, editors: *Proceedings Tenth International Workshop on Computing with Terms and Graphs, TERMGRAPH@FSCD 2018, Oxford, UK, 7th July 2018*, *EPTCS* 288, pp. 1–13, doi:10.4204/EPTCS.288.1.

[3] Clemens Grabmayer (2023): *From Compactifying Lambda-Letrec Terms to Recognizing Regular-Expression Processes*. Invited talk at the 13th International Workshop on *Developments in Computational Models*, affiliated with the conference FSCD, Sapienza Università, Rome, July 2. Slides available at `https://clegra.github.io/lf/DCM-2023-invited-talk.pdf`.

2 Compactifying Lambda-Letrec Terms

This section gives an overview about work that Jan Rochel and I did in the framework of the NWO-project Realizing Optimal Sharing (ROS) at Utrecht University (2009–2014).[1] It eventually led us to the definition and practical implementation of maximal sharing for terms in the λ-calculus with letrec, the Core language for the compilation of functional programming languages.

We started with the intention to study phenomena that arise practically for optimal-sharing implementations of the λ-calculus (by graph-transformation schemes due to Lamping [18], and Kathail [17], and

[1] This project was headed jointly by Vincent van Oostrom (rewriting and λ-calculus) and Doitse Swierstra (implementation of functional languages). The project was concluded successfully in June 2016 with Jan Rochel's defense of his thesis [22].

later interaction-net formalizations by Gonthier, Abadi, Lèvy [6], and also van Oostrom, van de Looij, Zwitserlood [20]), which are implementations of optimal or parallel β-reduction (due to Lèvy [19]). For this purpose Rochel wrote an impressive visualization and animation tool [21] for transforming graphs by reducing graph-rewrite redexes per mouse-click. It produces beautifully rendered graphs that slowly float over the screen like bacteria in a liquid under a microscope. This animation tool provided us with much room for experimentation. We first tried to understand whether optimal implementations could render the so-called static-argument transformation unnecessary. When we could not establish that, we first tried to understand in how far the static-argument transformation changes the evaluation of programs with respect to usual scope-preserving graph evaluation. As a consequence, we partly turned our attention away from optimal evaluation (in the hope that we would later come back to it with a better understanding).

We started by generalizing the static-argument transformation to more general optimizations.

Parameter-dropping optimization transformations

In [23] we described an optimization transformation for the compilation of functional programs that drops parameters that are passed along unchanged between a number of recursive functions from the definitions of these functions. We used higher-order rewrite rules to describe this generalization of the static-argument transformation that permits the avoidance of repetitive evaluation patters [23]. We discovered later a close connection with Lambda Dropping due to Danvy and Schultz [5].

Realizing that we had moved on to terrain for which a strong theory had already been established, we set ourselves more ambitious goals: First, to understand formally and conceptually the relationship between terms in the λ-calculus with letrec ($\lambda_{\mathsf{letrec}}$) and the infinite λ-terms they represent (in λ^{∞}, the infinitary λ-calculus). Second, to find term graph representations of $\lambda_{\mathsf{letrec}}$-terms that are preserved under homomorphism (functional bisimularity). Finally third, we wanted to use possible answers for these two points to define maximally-shared representations of arbitrary $\lambda_{\mathsf{letrec}}$-terms. Below we report on our results concerning these three goals.

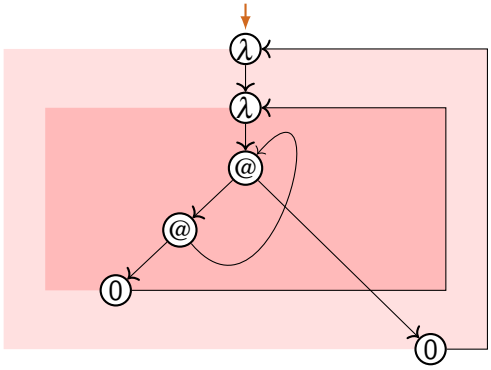
1. Expressibility of infinite λ-terms by terms in $\lambda_{\mathsf{letrec}}$ (and in λ_{μ}).

   We studied the question: Which infinite λ-terms are (infinite) unfoldings of terms in $\lambda_{\mathsf{letrec}}$, the λ-calculus with letrec, or (equivalent, but formally easier) in λ_{μ}, the λ-calculus with μ? Clearly, such infinite λ-terms have to be *regular* in the sense that their syntax-trees have only finitely many subtrees modulo α-conversion. However, while regularity is necessary for expressibility by a $\lambda_{\mathsf{letrec}}$-term under infinite unfolding, it is not sufficient. What is missing is, intuitively, that the abstraction scopes in regular infinite λ-terms are not infinitely entangled. We formulated this requirement in two different ways: that the infinite (regular) λ-term in question (i) has only finitely many 'generated subterms' that are generated by a certain decomposition rewrite system that uses eager scope closure, (ii) does not contain infinite 'binding–capturing chains'. Both conditions delineate the *strongly regular* infinite λ-terms among the regular ones. For this concept we showed that an infinite λ-term M is the unfolding of a term in $\lambda_{\mathsf{letrec}}$ (resp. a term in λ_{μ}) if and only if M is strongly regular. For $\lambda_{\mathsf{letrec}}$-expressibility we showed that in [9], and for λ_{μ}-expressibility in [11, 12]; slides with many suggestive illustrations can be found in [7].

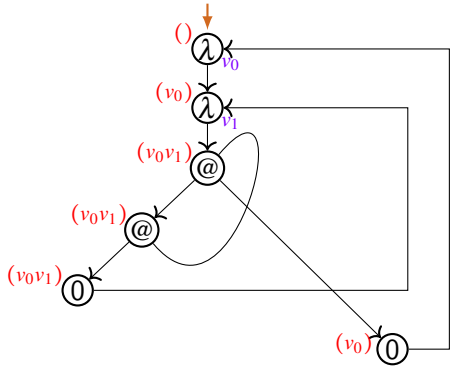
   Part of [9], and described separately in [24], is a non-trivial proof of confluence of a higher-order rewriting system that defines the unfolding semantics for $\lambda_{\mathsf{letrec}}$-terms. Furthermore in [10] we showed confluence of let-floating operations on $\lambda_{\mathsf{letrec}}$-terms, obtaining a unique-normal-form result for let-floating, by using a higher-order rewriting system for the formalization of let-floating.

2. Term graph representations of cyclic λ-terms.

   In [13, 16] we systematically investigated a range of natural options for faithfully representing the

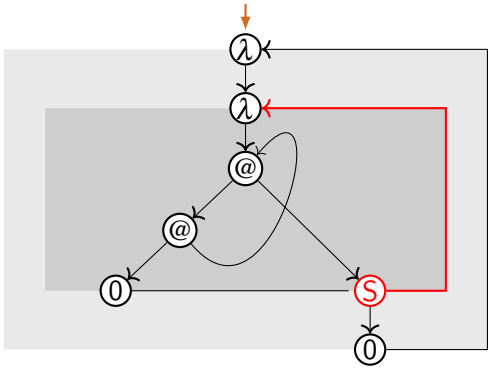


λ-higher-order term graph, version 1:
term graph with scope sets
as structure constraints

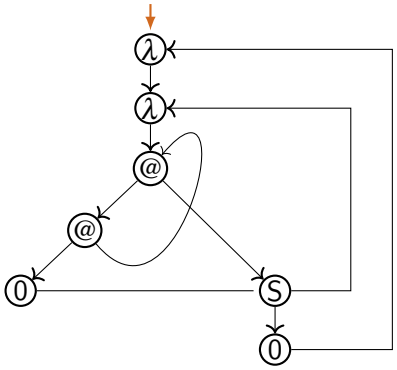


λ-higher-order term graph, version 2:
term graph with abstraction prefix function
as structure constraint

Figure 1: Translation of the $\lambda_{\mathsf{letrec}}$-term $L_0 := \lambda x.\lambda f.\mathsf{let}\ r = f\,r\,x\ \mathsf{in}\ r$ into a λ-higher-order term graph with scope sets à la Blom (left), and a λ-h-o term graph $[\![L_0]\!]_{\mathcal{H}}$ with an abstraction-prefix function (right). Note that the inner scope has been chosen minimally here, applying eager scope closure. (Non-eager scope λ-ho-term-graphs can be defined as well, but are not expedient for maximal sharing.)



first-order term graph with
scope vertices with backlinks (+ scope sets)



λ-term graph: first-order term graph
with scope vertices with backlinks

Figure 2: Translation of the $\lambda_{\mathsf{letrec}}$-term $L_0 := \lambda x.\lambda f.\mathsf{let}\ r = f\,r\,x\ \mathsf{in}\ r$ into a λ-term-graph $[\![L_0]\!]_{\mathcal{T}}$ by adding a scope vertex delimiting the inner scope to the λ-higher-order term graphs in Fig. 1, and by then dropping the scope sets (which now can be reconstructed as well as a corresponding abstraction prefix function). While the backlink from the left variable vertex to its binding abstraction vertex is drawn suggestively along the scope border, it does not target the scope-delimiting vertex, but continues invisibly below the backlink of that scope-delimiting vertex onwards to the commonly targeted abstraction vertex. (While not relevant for maximal sharing, relaxing the condition of eager scope closure for λ-ho-term-graphs can be dealt with by an adapted encoding as first-order term graph.)

cyclic λ-terms in $\lambda_{\mathsf{letrec}}$ by higher-order term graphs (first-order term graphs with additional features that describe scopes), and by first-order term graphs (with specific scope-delimiting vertices).

As the result of this analysis we arrived at a natural class of higher-order term graphs (of 'λ-ho-

term-graphs', see below) that can be implemented faithfully as first-order term graphs ('λ-term-graphs', see below). The basis of the higher-order term graphs (as well as of their first-order implementations) for representing $\lambda_{\mathsf{letrec}}$-terms are first-order term graphs with three different kinds of vertex labels:

- unary symbols λ for abstraction vertices,
- binary symbols @ for application vertices, and
- unary symbols 0 for nameless variable vertices that enable backlinks to the binding abstraction vertices.

The first-order λ-term-graphs also permit:

- binary symbols S for scope-delimiting vertices that facilitate backlinks to the abstraction vertices whose scope they close.

With this preparations we can now explain the higher-order λ-ho-term-graphs and first-order λ-term-graphs in more detail. For the precise definitions and statements we refer to [13, 16, 14].

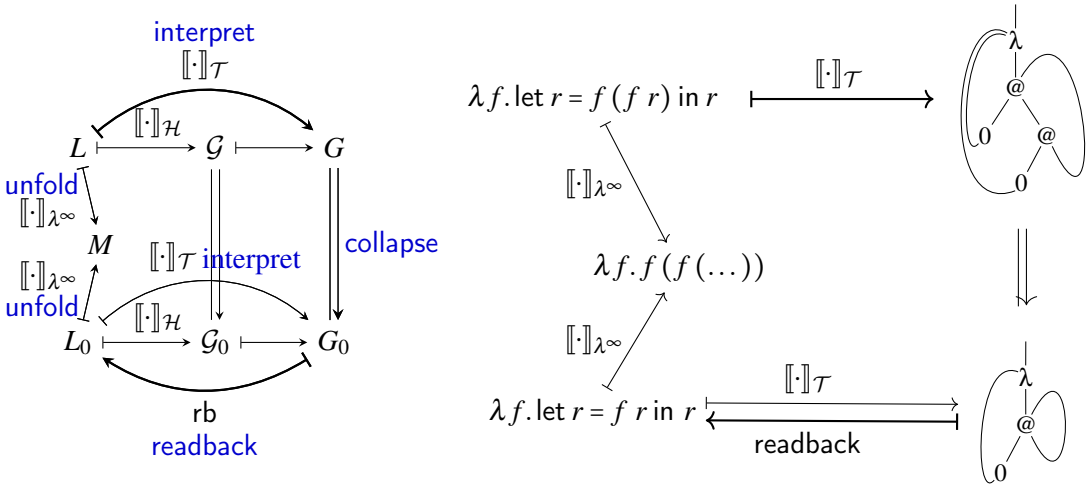
λ-ho-term-graphs appear in two versions:

λ-ho-term-graphs with scope-sets are extensions of first-order term graphs with vertex labels λ, @, and 0 by adding, to each abstraction vertex w, a scope set that consists of all vertices in the scope of w. The scope sets of abstraction vertices in a λ-ho-term-graph satisfy a number of conditions that safeguard that (i) scopes are nested, (ii) scopes arise by eager scope closure, and (iii) each variable vertex is contained in the scope of the abstraction vertex to which its backlink points to. In this way, scope sets aggregate scope information that is available locally at the abstraction vertices.
λ-term-graphs with scope sets are an adaptation of Blom's of higher-order term graphs with scope sets [4] to representing the cyclic λ-terms in $\lambda_{\mathsf{letrec}}$ (and the strongly regular infinite λ-terms in λ^{∞}). For an example, see Figure 1 on the left for the translation of a (variant) fixed-point combinator into a λ-ho-term-graph with (eager-scope) scope sets.

λ-ho-term-graphs with abstraction-prefix function are extensions of first-order term graphs with vertex labels λ, @, and 0 by adding an abstraction prefix function: That function assigns, to each vertex w, an abstraction prefix $(v_1 \dots v_n)$ consisting of a word of abstraction vertices that lists those abstractions (from the top down) for which w is in their 'extended scope' (transitive closure of scope relation) as obtained by eager scope closure. Abstraction prefixes aggregate scope information that then is locally available at individual vertices.
See Figure 1 on the right for the translation of a (variant) fixed-point combinator into a λ-ho-term-graph with abstraction prefixes (obtained by eager scope closure).

In both versions of λ-ho-term-graph, the added constraints guarantee that each variable vertex (with label 0) has a backlink to the binding λ-abstraction vertex. A bijective correspondence can be shown to exist between both versions of λ-ho-term-graphs (see [13, 16]).

λ-term-graphs are first-order term graphs that represent λ-ho-term-graphs of both kinds as above. Scopes are delimited, again by using eager scope closure, by scope-delimiting vertices (with label S) that have backlinks to the abstraction vertex whose scope they declare closed. Variable vertices (with label 0) have backlinks to the binding λ-abstraction vertex.
See Figure 2 for the encoding of the λ-ho-term-graphs in Figure 1 into a λ-term-graph. For this purpose a scope-delimiter vertex with label S is used to represent the (eager) closure of the inner scope.



(1) term graph interpretations $[\![\cdot]\!]$. of $\lambda_{\mathsf{letrec}}$-term L as:
   a. higher-order term graph $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$
   b. first-order term graph $G = [\![L]\!]_{\mathcal{T}}$

(2) bisimulation collapse $\Downarrow$ of first-order term graph G with as result G_0

(3) readback rb of first-order term graph G_0 yielding $\lambda_{\mathsf{letrec}}$-term $L_0 = \mathsf{rb}(G_0)$.

Figure 3: Schematic representation of the maximal sharing method, and its application to a toy example: Maximal sharing of a $\lambda_{\mathsf{letrec}}$-term L proceeds via three steps: (1) interpretation of L as a λ-term-graph $G = [\![L]\!]_{\mathcal{T}}$, (2) collapse of G via bisimilarity to λ-term-graph G_0, and (3) readback of $\lambda_{\mathsf{letrec}}$-term L_0 from G_0. On the top right these steps are illustrated for a redundant $\lambda_{\mathsf{letrec}}$-term formulation of a fixed-point combinator, yielding an efficient representation of fixed-point combinator as $\lambda_{\mathsf{letrec}}$-term.

The conditions underlying λ-ho-term-graphs and λ-term-graphs (see [13, 16]) guarantee that they represent finite or infinite closed λ-terms; that is, they do not contain meaningless parts. Both λ-ho-term-graphs (all two versions) and λ-term-graphs induce appropriate concepts of homomorphism (functional bisimulation) and bisimulation. Homomorphisms increase sharing, and introduce a sharing (partial) order. Bisimulations preserve the unfolding semantics (as do homomorphisms). We established in [13, 16] a bijective correspondence between λ-ho-term-graphs and λ-term-graphs that preserves and reflects homomorphisms, and hence the sharing (partial) order. These results form the basis of the maximal-sharing method, see below.

The property that is of the most central importance for the maximal-sharing method is that homomorphisms (functional bisimulations) between first-order term graphs preserve λ-term-graphs: if G_1 is a λ-term-graph, and $G_1 \trianglerighteq G_2$ for a term graph G_2 (there is a homomorphism from G_1 to G_2), then also G_2 is a λ-term-graph. For this property to hold, eager scope closure is crucial.[2]

[2]While that is not relevant for the maximal-sharing method (for which use of eager scope closure is essential), we mention as an aside that this restriction can be circumnavigated: a generalization of the preservation property can also be shown for a different kind of encoding of (also non-eager-scope) λ-ho-term-graphs into first-order term graphs (see Remark 7.10 in [16]).

3. Maximal sharing in $\lambda_{\mathsf{letrec}}$.

For defining maximally shared versions of terms in $\lambda_{\mathsf{letrec}}$ in a natural way we defined a 'representation pipeline' in [14, 15] (see Figure 3 for a suggestive illustration): First we linked $\lambda_{\mathsf{letrec}}$-terms by an interpretation function $[\![\cdot]\!]_{\mathcal{H}}$ to the class $\mathcal{H}$ of λ-ho-term-graphs that we formulated earlier in [13, 16]. Then we extended $[\![\cdot]\!]_{\mathcal{H}}$ by using the representation of λ-ho-term-graphs as λ-term-graphs (first-order term graphs) from [13, 16] to define an interpretation function $[\![\cdot]\!]_{\mathcal{T}}$ of $\lambda_{\mathsf{letrec}}$-terms to the class $\mathcal{T}$ of λ-term-graphs. For this representation pipeline we showed that unfolding equivalence $=_{[\![\cdot]\!]_{\lambda^\infty}}$ of $\lambda_{\mathsf{letrec}}$-terms is faithfully represented by bisimulation equivalence $\leftrightarrows$ on λ-ho-term-graphs, and equally, by bisimulation equivalence $\leftrightarrows$ on λ-term-graphs.

Then we defined a readback operation rb on λ-term-graphs in the class $\mathcal{T}$ (see also in Figure 3) with the property that the interpretation operation $[\![\cdot]\!]_{\mathcal{T}}$ is a left-inverse of rb on $\mathcal{T}$:

$$[\![\cdot]\!]_{\mathcal{T}} \circ \mathsf{rb} = \mathrm{id}_{\mathcal{T}} \quad \text{(modulo isomorphism)}.$$

These three operations facilitate to compute, for any given $\lambda_{\mathsf{letrec}}$-term L, a maximally shared form L_0, by the following three-step procedure (see Figure 3):

(interpret) from L its interpretation $[\![L]\!]_{\mathcal{T}}$ as λ-term-graph is obtained,

(collapse) from the λ-term-graph $[\![L]\!]_{\mathcal{T}}$ its bisimulation collapse G_0 is computed, which is again a λ-term-graph in $\mathcal{T}$ (due to preservation of λ-term-graphs along functional bisimulations),

(readback) from the collapsed λ-term-graph G_0 its readback $\mathsf{rb}(G_0)$ is computed, thereby obtaining the term $L_0 := \mathsf{rb}(G_0)$ as a maximally shared form L with $L_0 =_{[\![\cdot]\!]_{\lambda^\infty}} L$ (and hence so that L_0 has the same infinite unfolding as L).

This procedure permits an efficient implementation. We could derive its complexity as (at about) quadratic in the size of the input $\lambda_{\mathsf{letrec}}$-term.

See Figure 4 for an example of the collapse step on the λ-term-graph interpretation $[\![L]\!]_{\mathcal{T}}$ of an inefficient version L of a (slight variation of a) fixed-point combinator to obtain the λ-term-graph interpretation $[\![L_0]\!]_{\mathcal{H}}$ that obtains a more efficient and compact version L_0 of such a combinator.

A straightforward adaptation of this procedure permits to obtain also an efficient algorithm for deciding unfolding-semantics equality $=_{[\![\cdot]\!]_{\lambda^\infty}}$ of any two given $\lambda_{\mathsf{letrec}}$-terms L_1 and L_2 by the following two-step procedure:

(interpret) obtain the λ-term-graph interpretations $G_1 := [\![L_1]\!]_{\mathcal{T}}$ of L_1 and $G_2 := [\![L_2]\!]_{\mathcal{T}}$ of L_2;

(check-bisim) check bisimilarity of G_1 and G_2; if $G_1 \leftrightarrows G_2$ holds, conclude that $L_1 =_{[\![\cdot]\!]_{\lambda^\infty}} L_2$ holds (that is, L_1 and L_2 have the same infinite unfolding), otherwise $L_1 \neq_{[\![\cdot]\!]_{\lambda^\infty}} L_2$ holds.

We implemented both the maximal-sharing method and the decision procedure for unfolding equivalence $=_{[\![\cdot]\!]_{\lambda^\infty}}$ by a prototype implementation [25] that is available on Haskell's Hackage platform. For the efficient implementation of these methods we extended the representation pipeline from λ-term-graphs further to λ-DFAs, by which we mean representations of $\lambda_{\mathsf{letrec}}$-terms as deterministic finite-state automata. In this way, unfolding equivalence $=_{[\![\cdot]\!]_{\lambda^\infty}}$ of $\lambda_{\mathsf{letrec}}$-terms is represented as language equivalence of λ-DFAs, and so we could use for the implementation [25] that bisimulation collapse of λ-term-graphs is faithfully represented by state minimization of λ-DFAs.

At the end of this section I want to mention a concept that Vincent van Oostrom suggested after seeing the concept of λ-term-graphs in Jan Rochel's thesis [22]: the concept of 'nested term graphs'.

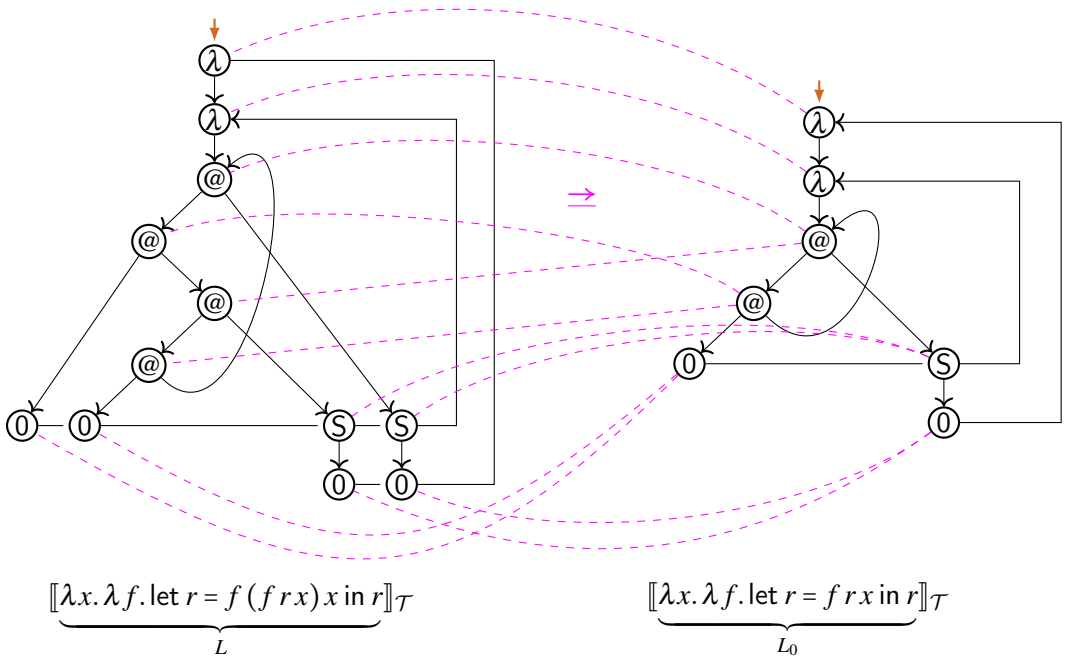


Figure 4: Compactification of the $\lambda_{\mathsf{letrec}}$-term L, a redundant form of a variant fixed-point combinator (compare with the forms in Figure 3), to the more compact $\lambda_{\mathsf{letrec}}$-term L_0. The λ-term-graph interpretations $[\![L]\!]_{\mathcal{T}}$ and $[\![L_0]\!]_{\mathcal{T}}$ of the $\lambda_{\mathsf{letrec}}$ terms L and L_0 are bisimilar. Indeed the links ---- form a functional bisimulation $\twoheadrightarrow$ from $[\![L]\!]_{\mathcal{T}}$ to $[\![L_0]\!]_{\mathcal{T}}$, of which the λ-term-graph $[\![L_0]\!]_{\mathcal{T}}$ is in bisimulation-collapsed form.

Nested Term Graphs

Motivated by the results on term graph representations and maximal sharing for $\lambda_{\mathsf{letrec}}$-terms, Vincent van Oostrom and I formulated a concept of nested term graph [8]. Instead of describing scopes by additional features like scope sets or an abstraction-prefix function in order to define constraints that guarantee that scopes are nested, we introduced 'nesting' itself as a structuring concept. This means that we permitted nesting of first-order term graphs into vertices of other first-order term graphs. In this manner, well-foundedly nested first-order term graphs can be defined by induction. We studied the behavioral semantics of nested term graphs in [8], and also showed, in analogy with the faithful encoding of λ-ho-term-graphs as λ-term-graphs, that nested term graphs can be encoded by first-order term graphs faithfully (in the sense of preserving the respective unfolding semantics).

Nested term graphs not only provide a natural formalization the maximal-sharing method developed in [13, 14], but they make it much more broadly applicable, also outside of Lambda Calculus.

References

[4] Stefan Blom (2001): *Term Graph Rewriting, Syntax and Sematics*. Ph.D. thesis, Vrije Universiteit Amsterdam. Available at `https://ir.cwi.nl/pub/29853/29853D.pdf` from webpage `https://ir.cwi.nl/pub/29853` for this thesis at CWI Amsterdam.

[5] Olivier Danvy & Ulrik P. Schultz (2000): *Lambda-Dropping: Transforming Recursive Equations into Programs with Block Structure*. Theoretical Computer Science 248(1), pp. 243–287, doi:10.1016/S0304-3975(00)00054-2. PEPM'97.

[6] Georges Gonthier, Martin Abadi & Jean-Jacques Lévy (1992): *The Geometry of Optimal Lambda Reduction*. In: *Proceedings of POPL'92*, pp. 15–26, doi:10.1145/143165.143172.

[7] Clemens Grabmayer (2019): *Modeling Terms in the λ-Calculus with letrec*. Invited talk at the Workshop *Computational Logic and Applications*, Université de Versailles, France, July 1–2. Slides available at `https://clegra.github.io/lf/CLA-2019-invited-talk.pdf`.

[8] Clemens Grabmayer & Vincent van Oostrom (2015): *Nested Term Graphs*. In Aart Middeldorp & Femke van Raamsdonk, editors: Post-Proceedings 8th International Workshop on Computing with Terms and Graphs, Vienna, Austria, July 13, 2014, *Electronic Proceedings in Theoretical Computer Science* 183, Open Publishing Association, pp. 48–65, doi:10.4204/EPTCS.183.4. ArXived at:1405.6380v2.

[9] Clemens Grabmayer & Jan Rochel (2012): *Expressibility in the Lambda-Calculus with Letrec*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.1208.2383. arXiv:1208.2383.

[10] Clemens Grabmayer & Jan Rochel (2013): *Confluent Let-Floating*. In: *Proceedings of IWC 2013 (2^{nd} International Workshop on Confluence)*, pp. 59–64. Available at `http://www.jaist.ac.jp/~hirokawa/iwc2013/iwc2013.pdf`. Available at `http://www.jaist.ac.jp/~hirokawa/iwc2013/iwc2013.pdf`.

[11] Clemens Grabmayer & Jan Rochel (2013): *Expressibility in the Lambda Calculus with Mu*. In Femke van Raamsdonk, editor: *24th International Conference on Rewriting Techniques and Applications (RTA 2013)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 21, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 206–222, doi:10.4230/LIPIcs.RTA.2013.206. Available at `http://drops.dagstuhl.de/opus/volltexte/2013/4063`.

[12] Clemens Grabmayer & Jan Rochel (2013): *Expressibility in the Lambda Calculus with μ*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.1304.6284. arXiv:1304.6284. Extends [11].

[13] Clemens Grabmayer & Jan Rochel (2013): *Term Graph Representations for Cyclic Lambda Terms*. In: *Proceedings of TERMGRAPH 2013*, *EPTCS* 110, pp. 56–73, doi:10.4204/EPTCS.110. ArXived at:1302.6338v1.

[14] Clemens Grabmayer & Jan Rochel (2014): *Maximal Sharing in the Lambda Calculus with Letrec*. In: *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming*, ICFP '14, ACM, New York, NY, USA, pp. 67–80, doi:10.1145/2628136.2628148.

[15] Clemens Grabmayer & Jan Rochel (2014): *Maximal Sharing in the Lambda Calculus with letrec*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.1401.1460. arXiv:1401.1460. Extends [14].

[16] Clemens Grabmayer & Jan Rochel (2014): *Term Graph Representations for Cyclic Lambda-Terms*. Technical Report, arxiv.org, doi:10.48550/arXiv.1304.6284. arXiv:1304.6284. Report extending [13] (proofs of main results added).

[17] Vinod Kumar Kathail (1990): *Optimal Interpreters for Lambda-calculus Based Functional Languages*. Ph.D. thesis, MIT. Available at `https://dspace.mit.edu/bitstream/handle/1721.1/14040/23292041-MIT.pdf?sequence=2`.

[18] John Lamping (1989): *An Algorithm for Optimal Lambda Calculus Reduction*. In: *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, Association for Computing Machinery, New York, NY, USA, p. 16–30, doi:10.1145/96709.96711.

[19] Jean-Jacques Lévy (1978): *Réductions correctes et optimales dans le λ-calcul*. Ph.D. thesis, Université, Paris VII.

[20] Vincent van Oostrom, Kees-Jan van de Looij & Marijn Zwitserlood (2004): *J*. Extended Abstract for the Workshop on Algebra and Logic on Programming Systems (ALPS), Kyoto, April 10th 2004. Available at `http://www.phil.uu.nl/~oostrom/publication/pdf/lambdascope.pdf`.

[21] Jan Rochel (2010): *Port Graph Rewriting in Haskell*. Implementation with an explanatory technical report at `http://rochel.info/docs/graph-rewriting.pdf` tool on HackageDB with packages

$$\frac{}{1\Downarrow} \qquad \frac{e_i\Downarrow}{(e_1+e_2)\Downarrow}\,(i\in\{1,2\}) \qquad \frac{e_1\Downarrow \qquad e_2\Downarrow}{(e_1\cdot e_2)\Downarrow} \qquad \frac{}{(e^*)\Downarrow}$$

$$\frac{}{a \xrightarrow{a} 1} \qquad \frac{e_i \xrightarrow{a} e_i'}{e_1+e_2 \xrightarrow{a} e_i'}\,(i\in\{1,2\}) \qquad \frac{e_1 \xrightarrow{a} e_1'}{e_1\cdot e_2 \xrightarrow{a} e_1'\cdot e_2} \qquad \frac{e_1\Downarrow \qquad e_2 \xrightarrow{a} e_2'}{e_1\cdot e_2 \xrightarrow{a} e_2'} \qquad \frac{e \xrightarrow{a} e'}{e^* \xrightarrow{a} e'\cdot e^*}$$

Figure 5: Transition system specification $\mathcal{T}$ for computations enabled by regular expressions.

graph-rewriting, graph-rewriting-layout, graph-rewriting-gl, graph-rewriting-ski, graph-rewriting-trs, graph-rewriting-lambdascope, maxsharing. The rewrite system uses Stewart's port graph grammars (PGGs) [59].

[22] Jan Rochel (2016): *Unfolding Semantics of the Untyped λ-Calculus with* letrec. Ph.D. thesis, Utrecht University, doi:10.48550/arXiv.1610.05954. Defended on June 20, 2016. Available at `http://rochel.info/thesis/thesis.pdf`, and as report `arXiv:1610.05954` on `arxiv.org`.

[23] Jan Rochel & Clemens Grabmayer (2011): *Repetitive Reduction Patterns in Lambda Calculus with Letrec*. In Rachid Echahed, editor: *Proceedings of the workshop TERMGRAPH 2011, 2 April 2011, Saarbrücken, Germany*, *Electronic Proceedings in Theoretical Computer Science* 48, Open Publishing Association, pp. 85–100, doi:10.4204/EPTCS.48.9. ArXived at:1102.2656v1.

[24] Jan Rochel & Clemens Grabmayer (2013): *Confluent Unfolding in the λ-calculus with* letrec. In: *Proceedings of IWC 2013 (2nd International Workshop on Confluence)*, pp. 17–22. Available at `http://www.jaist.ac.jp/~hirokawa/iwc2013/iwc2013.pdf`.

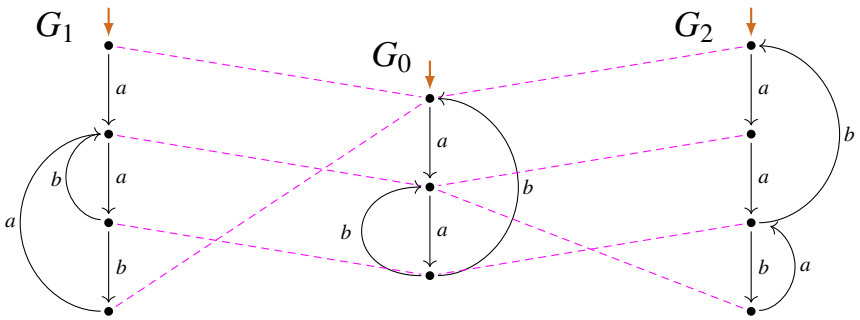
[25] Jan Rochel & Clemens Grabmayer (2014): *Maximal Sharing in the Lambda Calculus with letrec*. Implementation of the maximal sharing method described in [14, 15], available at `http://hackage.haskell.org/package/maxsharing/`.

3 Proving Bisimilarity between Regular-Expression Processes

This section motivates, summarizes, and provides references to my work on Milner's process semantics of regular expressions [52]. An important part of it (leading to [49, 50]) was done in close collaboration (2015–2020) with Wan Fokkink who had stimulated me to work on Milner's question already in 2005. While this section focuses on my work on Milner's axiomatization questions (see **(A)** below), my current work on the expressibility question (see **(E)** below) will be mentioned in Section 4.

Milner introduced a process semantics $[\![\cdot]\!]_P$ in [52] for regular expressions (conceived by Kleene [51]) that refines the standard language semantics $[\![\cdot]\!]_L$ (defined by Copi, Elgot, Wright [32]). For regular expressions e that are constructed from constants 0, 1, letters over a given set A with the binary operators + and $\cdot$, and the unary operator $(\cdot)^*$, Milner first defined a process interpretation $P(e)$ that can informally be described as follows: 0 is interpreted as a deadlocking process without any observable behavior, 1 as a process that terminates successfully immediately, letters from the set A stand for atomic actions that lead to successful termination; the binary operators + and $\cdot$ are interpreted as the operations of choice and concatenation of two processes, respectively, and the unary star operator $(\cdot)^*$ is interpreted as the operation of unbounded iteration of a process, but with the option to terminate successfully before each iteration.

Milner formalized this process interpretation in [52] as process graphs that are defined by induction on the structure of regular expressions. But soon afterwards a formal definition by means of a transition system specification (TSS) that defines a labeled transition system (LTS) became more common. The TSS $\mathcal{T}$ in Figure 5 defines, via derivations that it permits from its axioms, labeled transitions $\xrightarrow{a}$ for



$P((a\cdot(a\cdot(b+b\cdot a))^*)\cdot 0)$	$? \in im(P(\cdot))\ ?$	$P((1\cdot(((a\cdot a)\cdot(b\cdot a)^*)\cdot b)^*)\cdot 0)$
P-expressible, hence $[\![\cdot]\!]_P$-expr.	$[\![\cdot]\!]_P$-expressible	P-expressible, hence $[\![\cdot]\!]_P$-expr.
$(a\cdot(a\cdot(b+b\cdot a))^*)\cdot 0$	$=_{[\![\cdot]\!]_P}$	$(1\cdot(((a\cdot a)\cdot(b\cdot a)^*)\cdot b)^*)\cdot 0$

Figure 6: Two process graphs G_1 and G_2 that are P-expressible, and hence $[\![\cdot]\!]_P$-expressible, because they are the process interpretations of regular expressions as indicated. G_1 and G_2 are bisimilar via bisimulations that are drawn as links - - - to their joint bisimulation collapse G_0 (of which P-expressibility is at first unclear). It follows that also G_0 is $[\![\cdot]\!]_P$-expressible, and that process semantics equality holds between the regular expressions with interpretations G_1 and G_2, respectively. In this example G_0 is actually also in the image of $P(\cdot)$, hence P-expressible, as witnessed for example by $G_0 = P(((1\cdot a)\cdot(a\cdot(b+b\cdot a))^*)\cdot 0)$.

actions a that occur in a regular expressions, and immediate successful termination via the unary predicate $\Downarrow$. The process interpretation $P(e)$ of a regular expression e is then defined as the sub-LTS that is induced by e in the LTS on regular expressions that is defined via derivability in $\mathcal{T}$. See Figure 6 for suggestive examples of (bisimilar) process interpretations of two simple regular expressions. In process graph illustrations there and later we indicate the start vertex by a brown arrow ➙, and the property of a vertex v to permit immediate successful termination by emphasizing v in brown as ◉ with a boldface ring.

It is interesting to note that the so-defined process interpretation of regular expressions corresponds directly to non-deterministic finite-state automata (NFAs) that are defined via iterations of Antimirov's partial derivatives [27].[3]

Based on the process interpretation $P(\cdot)$, Milner then defined the process semantics of a regular expression e as $[\![e]\!]_P := [P(e)]_{\leftrightarrows}$ where $[P(e)]_{\leftrightarrows}$ is the equivalence class of $P(e)$ with respect to bisimilarity $\leftrightarrows$. In analogy to how language-semantics equality $=_{[\![\cdot]\!]_L}$ of regular expressions is defined from the language semantics $[\![\cdot]\!]_L$ (namely as $e =_{[\![\cdot]\!]_L} f$ if $L(e) = [\![e]\!]_L = [\![f]\!]_L = L(e)$, for all regular expressions e and f, where $L(g)$ is the language defined by a regular expression g) Milner was then interested in process-semantics equality $=_{[\![\cdot]\!]_P}$ that is defined, for all regular expressions e and f by:

$$\begin{aligned} e =_{[\![\cdot]\!]_P} f \ &:\Longleftrightarrow\ [\![e]\!]_P = [\![f]\!]_P \\ &\Longleftrightarrow\ P(e) \leftrightarrows P(f)\,. \end{aligned}$$

As the process interpretations of the regular expressions in Figure 7 are bisimilar, it follows that these regular expressions are linked by $=_{[\![\cdot]\!]_P}$.

[3] Antimirov did not have a process semantics in mind, but he had set out to define, for every regular expression e, an NFA that is typically smaller than the deterministic automaton (DFA) as usually associated with e in automata and language theory.

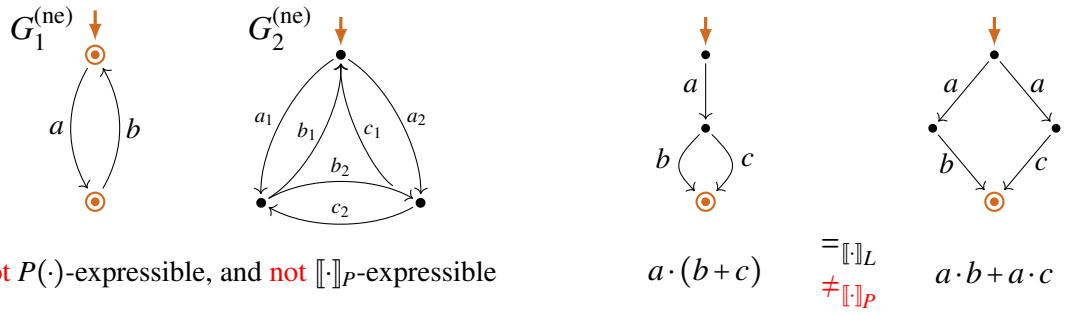


Figure 7: On the left: Two process graphs that are neither $P(\cdot)$-expressible (that is, not in the image of the process interpretation P) nor $[\![\cdot]\!]_P$-expressible (that is, not bisimilar to the process interpretation of any regular expression). On the right: two regular expressions with the same language semantics (associated language) but different process semantics, since the process interpretations are not bisimilar; therefore right-distributivity does not hold for $=_{[\![\cdot]\!]_P}$, which entails that fewer identities hold for $=_{[\![\cdot]\!]_P}$ than for $=_{[\![\cdot]\!]_L}$.

Milner realized in [52] that the process semantics $[\![\cdot]\!]_P$ of regular expressions differs from the language semantics $[\![\cdot]\!]_L$ in at least two respects: first, $[\![\cdot]\!]_P$ is incomplete, and second, process-semantics equality $=_{[\![\cdot]\!]_P}$ satisfies fewer identities than language-semantics equality $=_{[\![\cdot]\!]_L}$.

We start by explaining incompleteness of $[\![\cdot]\!]_P$. Language semantics $[\![\cdot]\!]_L$ is complete in the following sense: every language that is accepted by some finite-state automaton (FA) is the language that is defined by some regular expression; that is, every FA-accepted language is $[\![\cdot]\!]_L$-*expressible*. However, an analogous statement does not hold for the process interpretation: not every finite process graph is '$[\![\cdot]\!]_P$-expressible' in the sense of that it is 'P-expressible' by a regular expression. Here we call a finite process graph $[\![\cdot]\!]_P$-*expressible* if it is bisimilar to a P-*expressible* process graph, by which we mean the process interpretation of some regular expression (and hence a graph in the image of $P(\cdot)$. That not every finite process graph is P-expressible follows from the fact that there are finite process graphs that are not $[\![\cdot]\!]_P$-expressible, either. Indeed, Milner proved in [52] that the process graph $G_2^{(ne)}$ in Figure 7 not only is not P-expressible, but that it is not $[\![\cdot]\!]_P$-expressible, either. He also conjectured that also $G_1^{(ne)}$ in Figure 7 is not $[\![\cdot]\!]_P$-expressible. That was later shown by Bosscher [31].

Milner also noticed in [52] that some identities that hold for language-semantics equality $=_{[\![\cdot]\!]_L}$ are not true any longer for process semantics equality $=_{[\![\cdot]\!]_P}$. Most notably this is the case for right-distributivity $e \cdot (f+g) = e \cdot f + e \cdot g$, which is violated just as for the comparison of process terms via bisimilarity; see the well-known counterexample in Figure 7. The language-semantics identity $e \cdot 0 = 0$ is also violated in the process semantics. In order to define a natural sound adaptation (that we here designated by) Mil, see Figure 8, of the complete axiom systems for $=_{[\![\cdot]\!]_L}$ by Aanderaa [26] and Salomaa [53], Milner dropped these two identities from Aanderaa's system, but added the sound identity $0 \cdot e = 0$.

These two pecularities of the process semantics led Milner to formulating two questions concerning recognizability of expressible process graphs, and axiomatizability of process-semantics equality:

(E) How can $[\![\cdot]\!]_P$-expressible process graphs be characterized structurally, that is, those finite process graphs that are bisimilar to process interpretations of regular expressions?

(A) Is the natural adaptation Mil to process-semantics equality $=_{[\![\cdot]\!]_P}$ (see Figure 8 for Mil) of Salomaa's and Aanderaa's complete proof systems for language-semantics equality $=_{[\![\cdot]\!]_L}$ complete for $=_{[\![\cdot]\!]_P}$?

The expressibility question **(E)** seems to have received only limited attention at first. The reason may have been because it asks for a structural property of (the $[\![\cdot]\!]_P$-expressible) process graphs that is

(A1) $e+(f+g)=(e+f)+g$
(A2) $e+0=e$
(A3) $e+f=f+e$
(A4) $e+e=e$
(A5) $e\cdot(f\cdot g)=(e\cdot f)\cdot g$
(A6) $(e+f)\cdot g=e\cdot g+f\cdot g$
(A7) $e=1\cdot e$
(A8) $e=e\cdot 1$
(A9) $0=0\cdot e$
(A10) $e^*=1+e\cdot e^*$
(A11) $e^*=(1+e)^*$

$$\frac{e=f\cdot e+g}{e=f^*\cdot g}\ \mathrm{RSP}^*\ (\text{if } f\not\Downarrow)$$

Figure 8: Milner's equational proof system Mil for process semantics equality $=_{[\![\cdot]\!]_P}$ of regular expressions with the fixed-point rule RSP* in addition to the (not shown) basic rules for reasoning with equations (which guarantee that derivability in Mil is a congruence relation). From Mil the complete proof system for language equivalence $=_{[\![\cdot]\!]_L}$ due to Aanderaa arises by adding the axioms $e\cdot(f+g)=e\cdot f+e\cdot g$ and $e\cdot 0=0$ (which are not sound for $=_{[\![\cdot]\!]_P}$) and by dropping (A9) (which then is derivable).

invariant under bisimilarity. This is a difficult aim, because bisimulations can significantly distort the topological structure of labeled transition graphs. Two variants of **(E)** have been solved after some time: First, the question for a natural sufficient condition for $[\![\cdot]\!]_P$-expressibility of process graphs was answered by Baeten and Corradini in [28] by the definition of process graphs that satisfy 'well-behaved' recursive specifications. Second, the question of whether $[\![\cdot]\!]_P$-expressibility of finite process graphs is decidable was answered by Baeten, Corradini, and myself in [29] by giving a decision procedure (unfortunately it is highly super-exponential) that is based on minimizing well-behaved specifications under bisimilarity.

For the axiomatization problem **(A)** at first only a string of partial results have been obtained. In particular Milner's proof system Mil has initially been shown to be complete for $=_{[\![\cdot]\!]_P}$ for the following subclasses of regular expressions:

(a) without 0 and 1, but with binary star iteration $e_1{}^{\circledast}e_2$ with iteration-part e_1 and exit-part e_2 instead of unary star (Fokkink and Zantema, 1994, [36]),

(b) with 0, and with iterations restricted to exit-less ones $(\cdot)^*\cdot 0$ in absence of 1 (Fokkink, 1997, [35]) and in the presence of 1 (Fokkink, 1996 [34]),

(c) without 0, and with restricted occurrences of 1 (Corradini, De Nicola, and Labella, 2002 [33]),

(d) 1-*free* expressions formed with 0, without 1, but with binary iteration $\circledast$ (G, Fokkink, 2020, [49, 50], also showing the completeness of a proof system by Bergstra, Bethke, and Ponse [30]).

While the maximal subclasses in (c) and (d) are incomparable, these results can be joined to apply to an encompassing class that is still a proper subclass of the regular expressions, see [49]. Independently of these partial results concerning completeness of Milner's system Mil for subclasses of regular expressions, I noticed in [37] that from Mil a proof system that is complete for $=_{[\![\cdot]\!]_P}$ arises when the single-equation fixed-point rule RSP* is replaced by a unique-solvability principle USP for systems of guarded equations. Also in [37] I formulated a coinductively motivated proof system for process-semantics equality $=_{[\![\cdot]\!]_P}$ that utilizes Antimirov's partial derivatives [27] of regular expressions.

The principal new idea that facilitated the partial completeness result (d) in [49, 50] of Mil for 1-free regular expressions consisted in formulating a natural structural condition that is sufficient (but not necessary) for $[\![\cdot]\!]_P$-expressibility of process graphs: the *Loop Existence and Elimination Condition* LEE, and its layered form LLEE. This condition is based on the concept of 'loop (process) graph', and an elimination process of 'loop subgraphs' from a given process graph. A process graph G is said to have

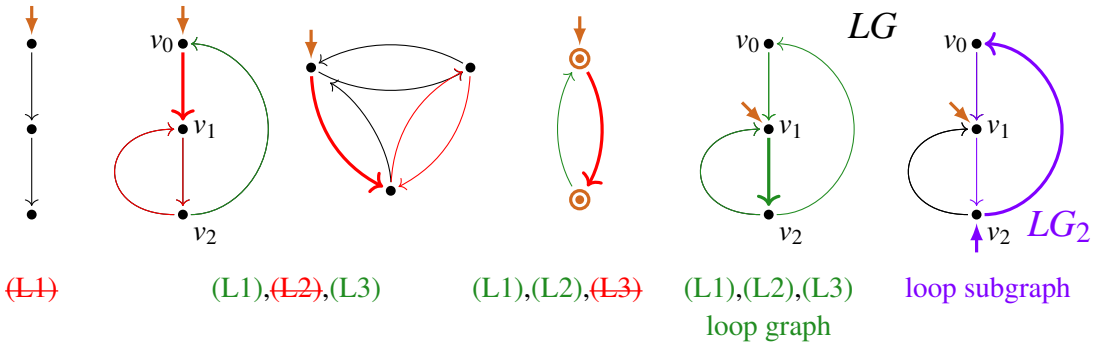


Figure 9: Four process graphs (action labels ignored) that violate at least one loop graph condition (LG1), (LG2), or (LG3), and a loop graph LG with one of its loop subgraph LG_2.

the property LEE if the non-deterministic iterative procedure, started on G, of repeatedly eliminating loop subgraphs is able to obtain a process graph without an infinite behavior (that is, a graph without infinite paths and traces). We explain the definitions in some more detail below, and provide examples.

A process graph LG is called a *loop (process) graph* if it satisfies the following three conditions:

(LG1) There is an infinite trace from the start vertex of LG.

(LG2) Every infinite trace from the start vertex v_s of LG returns to v_s.

(LG3) Immediate successful termination is only possible at the start vertex of LG.

In such a loop graph LG, the transitions from the start vertex are called *loop-entry transitions*, and all other transitions are called *loop-body transitions*. By a *loop subgraph* of a process graph G we mean a graph LG such that with respect to a vertex v of G, and a non-empty set T of transitions of G that depart from v the following three conditions are satisfied:

(LSG1) LG is a subgraph of G with start vertex v (which may be different from the start vertex v_s of G).

(LSG2) LG is generated by the transitions T from v in the following sense: LG contains all vertices and transitions of G that are reachable on traces that start from v via transitions in T, and continue onward until v is reached again for the first time.

(LSG3) LG is a loop graph.

In accordance with the stipulation for loop graphs, in such a loop subgraph LG the transitions in T are called *loop-entry* transitions of LG, and all others *loop-body* transitions of LG. In Figure 9 we have gathered, on the left, four examples of process graphs (with action labels ignored) that are *not* loop graphs: each of them violates one of the conditions (LG1), (LG2), or (LG3). The paths in red indicate violations of (LG2), and (LG3), respectively, where the thicker arrows from the start vertex indicate transitions that would need to be (but are not) loop-entry transitions. However, the loop subgraph LG_2 in Figure 9 is indeed a loop graph.

Based on these concepts, elimination of loop subgraphs is then defined as follows. We say that G' is the result of *eliminating a loop subgraph* LG with set T of loop-entry transitions *from* a process graph G, and denote such an elimination step by $G \Rightarrow_{\text{elim}} G'$, if G' results from G by first removing the transitions in T and by then applying garbage collection of vertices and transitions that have become unreachable from the start vertex of G due to the transition removals. See Figure 10 for an example of three loop elimination steps. As for non-examples, note that neither of two not $[\![\cdot]\!]_P$-expressible graphs $G_1^{(\text{ne})}$ and

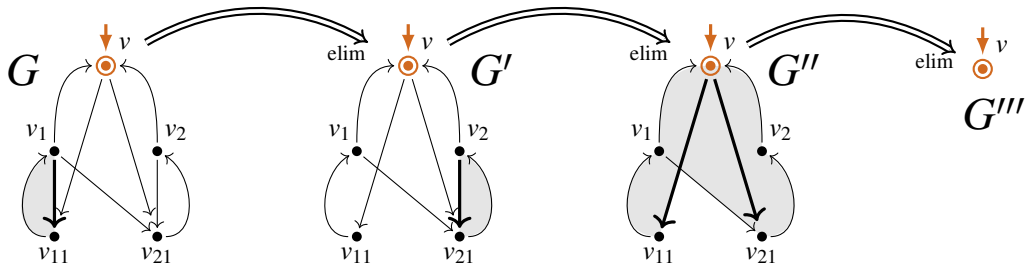


Figure 10: Example of successful loop elimination from the process graph G: three elimination steps of loop subcharts, which are represented as shaded gray areas, lead to the process graph G''' without infinite behaviour. These steps witness that G satisfies the properties LEE and LLEE (as well as do G', G'', G''').

$G_2^{(\mathrm{ne})}$ in Figure 7 are loop graphs, nor do they contain loop subgraphs; hence neither of $G_1^{(\mathrm{ne})}$ and $G_2^{(\mathrm{ne})}$ permits a loop-elimination step.

We say that a process graph G *has the property* LEE (resp. *has the property* LLEE (*layered* LEE)) if there is a finite sequence of loop-elimination steps $G \Rightarrow^*_{\mathrm{elim}} G'$ from G such that the resulting graph G' does not permit an infinite trace (and resp., if additionally during the elimination steps in $G \Rightarrow^*_{\mathrm{elim}} G'$ it never happens that a transition is removed that was a loop-body transition of a loop subgraph that was eliminated in an earlier step). It can be shown that although the property LLEE is a formally stronger requirement than the property LEE, which often helps to simplify proofs, both properties are equivalent. See Figure 10 for an example of a process graph G with the properties LEE and LLEE as is witnessed there by a sequence of three loop elimination steps that lead to graph G''' without infinite traces. The not $[\![\cdot]\!]_P$-expressible graphs $G_1^{(\mathrm{ne})}$ and $G_2^{(\mathrm{ne})}$ in Figure 7 do not satisfy LLEE and LEE, since loop elimination is not successful on them: they do not enable loop-elimination steps, but facilitate infinite traces.

The reason why the definition of the properties LEE and LLEE has facilitated progress concerning the problem **(A)** was that they define manageable conditions that could be used for proofs about process graphs that are linked by functional bisimulations. Specifically for obtaining the partial result (d) in [49, 50] it was crucial that we could prove the following facts:

(I)$_1$ Process interpretations of 1-free regular expressions satisfy LLEE (see [50, 49]).

(E)$_1$ Finite process graphs with LLEE are $[\![\cdot]\!]_P$-expressible, by 1-free regular expressions (see [50, 49]).

(C) LLEE is preserved along functional bisimilarity, and consequently, also by the operation of bisimulation collapse (see [50, 49]).

Additionally, the property LLEE permitted me to formulate a coinductive version cMil of Milner's system Mil that also permits cyclic derivations of the form of process graphs with the property LLEE, see [40, 39, 46]. The system cMil could be viewed as being located proof-theoretically half-way in between Mil and bisimulations between process interpretation. As such it could be expected to form a natural beachhead for a completeness proof of Mil.

These results raised my hope that the argumentation could be extended quite directly to the full set of regular expressions (including 1 and with unary iteration instead of binary iteration) as well as to process graphs with 1-transitions and with the property LEE. While the generalization of **(I)**$_1$ to all regular expressions does not hold, this obstacle could be overcome by defining a refined process interpretation with the desired property. Together with a rather straightforward generalization of **(E)**$_1$ we obtained:

~~**(I)**~~ The process interpretation $P(e)$ of a regular expression e does not always satisfy LLEE (nor LEE) (see [38, 42]).

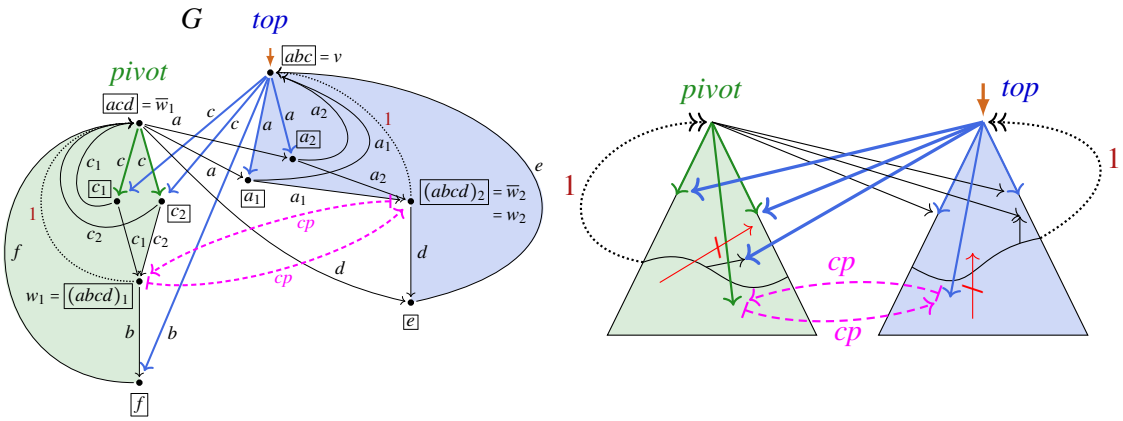


Figure 11: On the left: a finite process graph G with 1-transitions (drawn dotted, representing empty steps) that satisfies LLEE, but cannot be minimized under bisimilarity while preserving LLEE. It is a prototypical example of a twin-crystal. As such it consists of two interlinked parts, the top-part and the pivot-part, which by themselves are bisimulation collapsed, but contain vertices that have bisimilar counterparts in the opposite part of the twin-crystal. The self-inverse counterpart function *cp* links bisimilar vertices in the two parts. On the right: schematic illustration of a twin-crystal with suggestive drawing of its top-part and its pivot-part, together with interconnecting proper transitions from *top* and *pivot*.

(RI)$_1$ There is a refined process interpretation $\underline{P}(\cdot)$ that produces finite process graphs with 1-transitions such that, for every regular expression e, $\underline{P}(e)$ satisfies LLEE, $\underline{P}(e)$ is a refinement of $P(e)$ by sharing transitions by means of added 1-transitions, and $\underline{P}(e) \leftrightarrows P(e)$, that is, $\underline{P}(e)$ is bisimilar to $P(e)$ when 1-transitions are interpreted as empty steps (see [47], and a slightly weaker statement in [38, 42]).

(E)$_1$ Finite process graphs with 1-transitions and with LLEE are $[\![\cdot]\!]_P$-expressible. (See [39, 40, 46].)

However, critically, a direct generalization of our argument broke down dramatically due to the fact that the collapse statement **(C)** did not generalize to process graphs with LLEE that contain 1-transitions:

~~**(C)**$_1$~~ LLEE is not preserved under bisimulation collapse of process graphs with 1-transitions. A counterexample holds for the process graph G on the left in Figure 11. (See [43, 44, 47].)

As a consequence of this statement the image of the process interpretation is not closed under bisimulation, see [47]. This, however, contrasts with the image of a 'compact' version of the process interpretation that, when restricted to 'under-star-1-free' regular expressions, is closed under bisimulation collapse, see [47, 48]. Now due to ~~**(C)**$_1$~~ the proof strategy we used in [49, 50] for showing completeness of Mil for 1-free regular expressions, turned out not to work for showing completeness of Mil for the full class of regular expressions. At the very least it was in need of a substantial refinement.

What came to my rescue here was that the counterexample for LLEE-preserving collapse of process graphs with 1-transitions and LLEE, the graph G in 11, is of a specific symmetric form. It is a *twin-crystal*, a process graph with 1-transitions and with LLEE that is near-collapsed in the sense that non-identical bisimilar vertices appear only as pairs. More precisely, twin-crystals are process graphs with 1-transitions and with LLEE that consist of a single strongly connected component (scc), and of two parts, the top-part and the pivot-part (see in Figure 11 on the right). Each part by itself is bisimulation collapsed, and hence any two bisimilar vertices in the twin-crystal must occur in different of the top and pivot parts, and are linked by a self-inverse (partial) counterpart function. Process graphs with 1-transitions and with LLEE that are collapsed apart from within scc's, and in which all scc's are either collapsed or twin-crystals, we called *crystallized.* For this concept it was possible to show:

(NC)$_1$ Every finite process graph with 1-transitions and with LLEE can be minimized under bisimilarity to obtain a crystallized process graph (see [43, 44, 45]).

This statement is based on an effective *crystallization procedure* of process graphs with LLEE and with 1-transitions: it minimizes all scc's of the graph either to twin-crystals or collapsed parts of the graph, and also guarantees that the resulting graph is collapsed apart from within those scc's that are twin-crystals. The symmetric structure of twin-crystals can then be used to show that self-bisimulations of crystallized process graphs are of a particularly easy kind, which can be assembled from bisimulation slices that act on the twin-crystal-scc's [41]. This result on crystallized versions of process interpretations permitted me to adapt the proof strategy that Fokkink and I had used previously to also show completeness of Mil for $=_{[\![\cdot]\!]_P}$ on the full class of regular expressions, see [43, 44], and the poster [45].

There is now much hope that the crystallization technique that we developed for solving the axiomatization question **(A)** may turn out to facilitate also significant improvements for answers to the expressibility question **(E)**. We return to the expressibility question **(E)** at the end of the next section.

References

[26] Stål Aanderaa (1965): *On the Algebra of Regular Expressions*. Technical Report, Applied Mathematics, Harvard University.

[27] Valentin Antimirov (1996): *Partial Derivatives of Regular Expressions and Finite Automaton Constructions*. *Theoretical Computer Science* 155(2), pp. 291–319, doi:10.1016/0304-3975(95)00182-4.

[28] Jos Baeten & Flavio Corradini (2005): *Regular Expressions in Process Algebra*. In: *Proceedings of LICS 2005*, IEEE Computer Society 2005, pp. 12–19, doi:10.1109/LICS.2005.43.

[29] Jos Baeten, Flavio Corradini & Clemens Grabmayer (2007): *A Characterization of Regular Expressions Under Bisimulation*. *Journal of the ACM* 54(2), pp. 1–28, doi:10.1145/1219092.1219094.

[30] Jan Bergstra, Inge Bethke & Alban Ponse (1994): *Process Algebra with Iteration and Nesting*. *The Computer Journal* 37(4), pp. 243–258, doi:10.1093/comjnl/37.4.243.

[31] Doeko Bosscher (1997): *Grammars Modulo Bisimulation*. Ph.D. thesis, University of Amsterdam.

[32] Irving M. Copi, Calvin C. Elgot & Jesse B. Wright (1958): *Realization of Events by Logical Nets*. *Journal of the ACM* 5(2), doi:10.1007/978-1-4613-8177-8_1.

[33] Flavio Corradini, Rocco De Nicola & Anna Labella (2002): *An Equational Axiomatization of Bisimulation over Regular Expressions*. *Journal of Logic and Computation* 12(2), pp. 301–320, doi:10.1093/logcom/12.2.301.

[34] Wan Fokkink (1996): *An Axiomatization for the Terminal Cycle*. Technical Report, *Logic Group Preprint Series*, Vol. 167, Utrecht University.

[35] Wan Fokkink (1997): *Axiomatizations for the Perpetual Loop in Process Algebra*. In: *Proc. ICALP'97*, *LNCS* 1256, Springer, Berlin, Heidelberg, pp. 571–581, doi:10.1007/3-540-63165-8_212.

[36] Wan Fokkink & Hans Zantema (1994): *Basic Process Algebra with Iteration: Completeness of its Equational Axioms*. *The Computer Journal* 37(4), pp. 259–267, doi:10.1093/comjnl/37.4.259.

[37] Clemens Grabmayer (2006): *A Coinductive Axiomatisation of Regular Expressions under Bisimulation*. Technical Report, University of Nottingham. Short Contribution to CMCS 2006, March 25-27, 2006, Vienna Institute of Technology, Austria, `https://clegra.github.io/lf/sc.pdf`, slides for the talk available at `https://clegra.github.io/lf/cmcs06.pdf`.

[38] Clemens Grabmayer (2020): *Structure-Constrained Process Graphs for the Process Semantics of Regular Expressions*. Technical Report, `arxiv.org`, doi:https://doi.org/10.48550/arXiv.2012.10869. arXiv:2012.10869. Report version of [42].

[39] Clemens Grabmayer (2021): *A Coinductive Version of Milner's Proof System for Regular Expressions Modulo Bisimilarity*. Technical Report, arxiv.org, doi:10.48550/arXiv.2108.13104. arXiv:2108.13104. Extended report for [40].

[40] Clemens Grabmayer (2021): *A Coinductive Version of Milner's Proof System for Regular Expressions Modulo Bisimilarity*. In Fabio Gadducci & Alexandra Silva, editors: *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 211, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 16:1–16:23, doi:10.4230/LIPIcs.CALCO.2021.16. Extended report see [39].

[41] Clemens Grabmayer (2021): *Bisimulation Slices and Transfer Functions*. Technical report, Reykjavik University. Extended abstract for the 32nd Nordic Workshop on Programming Theory (NWPT 2021), http://icetcs.ru.is/nwpt21/abstracts/paper5.pdf.

[42] Clemens Grabmayer (2021): *Structure-Constrained Process Graphs for the Process Semantics of Regular Expressions*. *Electronic Proceedings in Theoretical Computer Science* 334, p. 29–45, doi:10.4204/eptcs.334.3. Extended report for [38].

[43] Clemens Grabmayer (2022): *Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete (Crystallization: Near-Collapsing Process Graph Interpretations of Regular Expressions)*. In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, Association for Computing Machinery, New York, NY, USA, pp. 1–13, doi:10.1145/3531130.3532430.

[44] Clemens Grabmayer (2022): *Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete (Crystallization: Near-Collapsing Process Graph Interpretations of Regular Expressions)*. Technical Report, arxiv.org, doi:10.48550/arXiv.2209.12188. arXiv:2209.12188. Technical report version of [43].

[45] Clemens Grabmayer (2022): *Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete (Crystallization: Near-Collapsing Process Graph Interpretations of Regular Expressions)*. Poster presented at LICS'22, Technion, Haifa, Israel, August 5. https://clegra.github.io/lf/poster-lics-2022.pdf.

[46] Clemens Grabmayer (2023): *A Coinductive Reformulation of Milner's Proof System for Regular Expressions Modulo Bisimilarity*. *Logical Methods in Computer Science* Volume 19, Issue 2, doi:10.46298/lmcs-19(2:17)2023. Available at https://lmcs.episciences.org/11519.

[47] Clemens Grabmayer (2023): *The Image of the Process Interpretation of Regular Expressions is Not Closed Under Bisimulation Collapse*. Technical Report, arxiv.org, doi:10.48550/arXiv.2303.08553. arXiv:2303.08553.

[48] Clemens Grabmayer (2024): *Closing the Image of the Process Interpretation of 1-Free Regular Expressions Under Bisimulation Collapse*. Preliminary Proceedings of the Workshop TERMGRAPH 2024 associated with ETAPS 2024, Luxembourg, April 7, 2024. Extended abstract https://clegra.github.io/lf/closing-bc-i-pi-us1f.pdf and presentation slides https://clegra.github.io/lf/TG-2024.pdf.

[49] Clemens Grabmayer & Wan Fokkink (2020): *A Complete Proof System for 1-Free Regular Expressions Modulo Bisimilarity*. In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, Association for Computing Machinery, New York, NY, USA, p. 465–478, doi:10.1145/3373718.3394744.

[50] Clemens Grabmayer & Wan Fokkink (2020): *A Complete Proof System for 1-Free Regular Expressions Modulo Bisimilarity*. Technical Report, arxiv.org, doi:10.48550/arXiv.2004.12740. arXiv:2004.12740. Report version of [49].

[51] Stephen C. Kleene (1951): *Representation of Events in Nerve Nets and Finite Automata*. In: *Automata Studies*, Princeton University Press, Princeton, New Jersey, USA, pp. 3–42, doi:10.1515/9781400882618-002.

[52] Robin Milner (1984): *A Complete Inference System for a Class of Regular Behaviours*. *Journal of Computer and System Sciences* 28(3), pp. 439–466, doi:10.1016/0022-0000(84)90023-0.

[53] Arto Salomaa (1966): *Two Complete Axiom Systems for the Algebra of Regular Events*. Journal of the ACM 13(1), pp. 158–169, doi:10.1145/321312.321326.

4 Current and Future Work

This section touches on my current research, and lists as well as briefly motivates three research questions and projects that have developed out of the work that we summarized in the previous two sections. This is organized in two subsections below that refer to the topics of Section 2 and Section 3, respectively.

4.1 Maximal Sharing at Run Time

Apart from using the maximal-sharing method for functional programs as a static-analysis based optimization transformation during compilation, one of the ideas for applications that Rochel and I gathered in [14] was that maximal sharing could be used as an optimization transformation also repeatedly at run-time. Making that idea fruitful, however, requires that representations of programs that are used in graph evaluators can be linked closely with λ-term-graph representations of $\lambda_{\mathsf{letrec}}$-terms on which the maximal-sharing method operates. This is necessary because graph evaluators in implementations of functional languages typically use supercombinator representations of $\lambda_{\mathsf{letrec}}$-terms, and much computational overhead is to be expected in transformations to and from λ-term-graphs. Yet any such overhead is highly undesirable during program execution. Now supercombinator reduction as carried out by graph evaluators intuitively corresponds to *scope-sharing* forms of β-reduction.[4] And so, since λ-term-graphs contain neatly described scopes of λ-abstractions, the implementation of a scope-sharing form of evaluation on λ-term-graphs is conceivable. These considerations lead me to the following research question.

Research Question 1. *Coupling of maximal sharing with evaluation, generally, and more specifically:*

(i) Can the maximal-sharing method for terms in the λ-calculus with letrec *be coupled naturally with an efficient evaluation method (such as a standard graph-evaluation implementation)?*

(ii) Do λ-term-graphs (which represent $\lambda_{\mathsf{letrec}}$-terms) permit a representation as interaction nets or as port graphs [59] for which a form of β-reduction can be defined that preserves both λ-term-graph form and represented λ-abstraction scopes by adequately chosen multi-steps of interactions?

In communication after the workshop, Ian Mackie pointed me to his interaction-net based implementation [56, 57] of an evaluation method for the λ-calculus. I am grateful for this reference, first, because this interaction-net representation of λ-terms bears a close resemblance with λ-term-graphs, and second, because it provides a mechanism for implementing scope-preserving forms of β-reduction. Nevertheless it remains a challenging question to relate the two formalisms (λ-term-graphs and interaction-net representations of λ-terms in [56]) closely together. Yet an interaction-net representation of λ-term-graphs close to the representation of λ-terms as used in [56] seems to me to be a plausible and promising in-road for approaching part (ii) of Research Question 1.

Regarding graph evaluators that implement scope-preserving forms of β-reduction it will also be important to explore correspondences on the rewrite-step level between graph evaluation steps and steps of, on the one hand, leftmost-outermost β-reduction, and on the other hand, reduction on super-combinator representations of λ-terms. Such connections were recently outlined by van Oostrom in [58]: a correspondence between β-reduction steps and combinator-reduction steps, as well as a correspondence

[4] Note that scope-sharing is distinct from the *context-sharing* forms of graph reduction on which implementations of *parallel* or *optimal β-reduction* are based.

between, β-reduction and combinator-reduction multi-steps on the one hand, and graph-rewriting multi-steps on the other hand. In doing so, van Oostrom carried a decisive step further the idea that I suggested in [55] of using supercombinator representations for obtaining an alternative proof, based on graph-rewriting on supercombinator representations of λ-terms, of an invariance result for leftmost-outermost β-reduction by Accattoli and Dal Lago in [54]. Namely, of the result that leftmost-outermost β-reduction in the λ-calculus can be implemented on every reasonable machine with only a polynomial overhead in the number of computation steps. In preparation for such a proof I had used supercombinator representations of λ-terms in [55] for developing the result that the depth increase along any leftmost-outermost β-reduction sequence from a λ-term is always bounded linearly in the number of steps of the sequence.

4.2 Crystallization: Proof Verification, and Application to the Expressibility Problem

Currently I am writing two articles that will provide the details of the completeness proof of Milner's proof system Mil. The first article will explain the motivation of the crystallization process for process interpretations of regular expressions: a limit to minimization under bisimilarity of P-expressible process graphs. This limit will be established specifically for the process graph G in Figure 11 with 1-transitions. The second article will detail the crystallization procedure by which process graphs with the property LLEE (which are $[\![\cdot]\!]_P$-expressible) are minimized under bisimulation to obtain process graphs with LLEE that are close to their bisimulation collapse. This central result will then be used, as explained in [43], to show that Milner's proof system Mil is complete with respect to process semantics equality $=_{[\![\cdot]\!]_P}$.

This completeness proof can be explained with clear conceptual concepts, and with convincing details, answering Milner's question **(A)** positively. However, a verification of the crystallization procedure and the completeness proof of Mil with respect to $=_{[\![\cdot]\!]_P}$ forms an important goal for me.

Research Project 2. *Formalization of the proofs for crystallization, and completeness of* Mil*:*

(a) Develop formalizations of structure constraints for process graphs in order to verify the correctness of the crystallization procedure for process graphs with LEE *by a proof assistant.*

(b) Use the correctness proof of crystallization to verify the completeness proof of Milner's proof system Mil *by a proof assistant.*

Separately I am working out a proof of the fact that the loop existence and elimination property LEE (and equivalently LLEE) can be decided in polynomial time. For this result the observation is crucial that loop elimination $\Rightarrow_{\mathrm{elim}}$ can be completed to obtain a confluent rewrite system (which is obviously terminating). As a consequence of the efficient decidability of LLEE it follows that the restriction of the expressibility problem **(E)** to expressibility by regular expressions that are under-star-1-free (but with unary iteration, see [47, 48]) can be solved efficiently. This is because the methods and results in [49, 50] permit to show that a finite process graph G is $[\![\cdot]\!]_P$-expressible by a regular expression that is under-star-1-free if and only if the bisimulation collapse of G satisfies LLEE. Then it follows that expressibility of finite process graphs by regular expressions that are under-star-1-free can be decided in polynomial time.

The crystallization procedure that we use in the completeness proof of Mil with respect to $=_{[\![\cdot]\!]_P}$ suggests that an extension of this characterization statement to one for $[\![\cdot]\!]_P$-expressibility in full generality is conceivable. We formulate that as our final research question.

Research Question 3. *Is the problem of whether a finite process graph is $[\![\cdot]\!]_P$-expressible efficiently decidable? That is, is there a polynomial decision algorithm for it? Or is $[\![\cdot]\!]_P$-expressibility at least fixed-parameter tractable (in* FPT*) for interesting parameterizations?*

References

[54] Beniamino Accattoli & Ugo Dal Lago (2016): *(Leftmost-Outermost) Beta Reduction is Invariant, Indeed*. *Logical Methods in Computer Science* Volume 12, Issue 1, doi:10.2168/LMCS-12(1:4)2016. Available at `https://lmcs.episciences.org/1627`.

[55] Clemens Grabmayer (2019): *Linear Depth Increase of Lambda Terms in Leftmost-Outermost Beta-Reduction Rewrite Sequences*. Technical Report `arXiv:1604.07030`, `arxiv.org`, doi:10.48550/arXiv.1604.07030.

[56] Ian Mackie (1998): *YALE: Yet Another Lambda Evaluator Based on Interaction Nets*. In: *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*, ICFP '98, Association for Computing Machinery, New York, NY, USA, p. 117–128, doi:10.1145/289423.289434.

[57] Ian Mackie (2004): *Efficient lambda evaluation with interaction nets*. In Vincent van Oostrom, editor: *Proceedings of RTA 2004, Aachen, Germany, June 3-5, 2004*, *LNCS* 3091, pp. 155–169, doi:10.1007/978-3-540-25979-4_11.

[58] Vincent van Oostrom (2024): *On naïvely implementing the $\lambda\beta$-calculus*. Extended Abstract for TERM-GRAPH 2024, Luxembourg, April 7, 2024, `http://www.javakade.nl/research/pdf/naive-v2.pdf`, presentation slides `http://www.javakade.nl/research/talk/termgraph070424.pdf`.

[59] Charles Stewart (2002): *Reducibility between Classes of Port Graph Grammar*. *Journal of Computer and System Sciences* 65(2), pp. 169 – 223, doi:10.1006/jcss.2002.1814.

Logic Programming with Multiplicative Structures

Matteo Acclavio*
University of Sussex, Brighton (UK)
and
University of Southern Denmark, Odense (DK)

Roberto Maieli†
Dipartimento di Matematica e Fisica
Università Roma Tre
Roma, Italy

In the logic programming paradigm, a program is defined by a set of methods, each of which can be executed when specific conditions are met during the current state of an execution. The semantics of these programs can be elegantly represented using sequent calculi, in which each method is linked to an inference rule. In this context, proof search mirrors the program's execution. Previous works introduced a framework in which the process of constructing proof nets is employed to model executions, as opposed to the traditional approach of proof search in sequent calculus.

This paper further extends this investigation by focussing on the pure multiplicative fragment of this framework. We demonstrate, providing practical examples, the capability to define *logic programming methods* with context-sensitive behaviors solely through specific resource-preserving and context-free operations, corresponding to certain generalized multiplicative connectives explored in existing literature. We show how some of these methods, although still multiplicative, escape the purely multiplicative fragment of Linear Logic (MLL, containing only $\parr$ and $\otimes$).

1 Introduction

Proof theory provides various paradigms for interpreting computations as proofs and their transformations. The renowned Curry-Howard correspondence interprets proofs as programs, and proof reduction (i.e., cut-elimination) as program execution. This correspondence offers an elegant model for *functional programming*, where the primary computation mechanism is substitution. In this context, well-typed programs are expected to terminate their execution after computing results derived by the complete initial information. However, this paradigm appears to face challenges in representing programs where the main computational mechanism is not substitution, as well as the ones characterized by non-termination, partial information, and strong concurrency (see, e.g., distributed systems, database servers, and microservices architectures). By means of example, consider the way of modeling the Curry-Howard correspondence in the case of non-terminating programs, where we need to consider *infinitary proof systems* to be able to represent infinite programs as *non-wellfounded derivations*. In these systems, even basic results like soundness, completeness, and cut-elimination require complex techniques [17, 22, 23, 4, 56, 5]. Therefore, it may appear more intuitive to interpret the rules of operational semantics for these programs as rules of a sequent system, and the program execution as the process of proof search [51, 50, 13, 15]. This approach naturally handles issues related to partial information, the concurrency of rule applications, and the possibility of non-termination.

Alternatively, in the *logic programming* paradigm, a program is provided by a set of *methods*, which are elementary programs that can be executed when specific preconditions are met. The conventional proof-theoretical interpretation of logic programming associates a sequent of formulas with each state of the computation, and a sequent calculus rule with each program method. This establishes an intuitive connection between program execution and the process of proof search within the calculus. In this

*The author is supported by Villum Fonden, grant no. 50079.
†The author is supported by INdAM-GNSAGA.

S. Alves and I. Mackie (Eds.): 13th International Workshop
on Developments in Computational Models 2023 (DCM'23).
EPTCS 408, 2024, pp. 42–61, doi:10.4204/EPTCS.408.3

Paradigm	Curry-Howard	Logic programming (sequent calculus)	Logic programming (multiplicative structures)
State	Proof - with cuts; - without proper axioms (i.e., without open premises);	Proof - without cuts; - possibly with proper axioms (i.e., with open premises);	Multiplicative structure: - (transitory) component; - possibly with inputs;
Computation step	Proof reduction: Cut-elimination	Proof construction: Proper axioms elimination	Proof net expansion: Proof structure expansion
Final state	Cut-free proof	Derivation with closed branches	Structure without inputs
Program type	Formula	Formula	Network behavior

Figure 1: A summary of the interpretation of proofs-as-programs in the paradigms of functional programming, and logic programming using sequent calculus and using proof net expansion.

context, a derivation tree where all leaves are axioms of the system represents a successfully completed computation.

It's worth noting that two forms of non-determinism arise during the process of proof search, corresponding to two distinct notions of non-determinism in program executions. Using the terminology from [12, 38, 43], The first type of non-determinism arises from the possibility of applying multiple methods to separate sub-sequents, which is a consequence of the limitations of sequent calculus[1]. The second form of non-determinism is observed when different methods are applied to non-disjoint sequents.

In this paper we continue the investigation on the interpretation of logic programming based on linear logic *proof structure expansion* instead of sequent calculus proof search, as in [14, 15, 29, 35]. In this approach, the set of inputs of a proof structure is interpreted as the current state of an execution, and the process of connecting new proof structures to its input (called *expansion*) is interpreted as the application of a method. The motivation to employ proof structures is due to their efficacy in capturing the non-determinism arising from the constraints of sequent calculus syntax: the graphical syntax relieves us from the bureaucracy of rules permutations between independent rule applications. Additionally, proof structures offer a more flexible structure, enabling us to define methods corresponding to the expansion of multiple branches simultaneously.

Contributions of the paper. In this paper, we study a logic programming framework built upon linear logic proof structures, offering a generalization of the standard MLL-*proof structure* [30] and the focused bipolar proof structures [15]. We focus on the multiplicative fragment of this framework, where the Danos-Regnier *correctness criterion* [21] can be easily generalized.

We introduce the concept of a *component* as an acyclic multiplicative structure where each of its part can interact with a context, analogous to the notion of an *open derivation* in sequent calculus. After establishing the topological conditions necessary for ensuring the composability of components, we proceed to define the foundational blocks of a logic programming framework based in the expansion of proof structures. Within this framework, as main novelty, we offer a computational interpretation of a specific family of *generalized multiplicative connectives*, which are connectives provided with linear and context-free introduction rules proposed in early works on linear logic [21, 32], but which lacked of any concrete computational interpretation prior to this work. We conclude by illustrating methods, defined within a linear and context-free setting, whose behavior is "locally additive" (in the sense of [32]), which cannot be expressed using the conventional MLL connectives $\parr$ and $\otimes$ (see [44, 9, 47]).

[1] In sequent calculus, two independent rules which can be applied to a same sequent must be sequentialized because the syntax does not allow for the application of rules to portions of a sequent. At the same time, if proof search produces a branching, then the two branches of the proof search can be performed independently in a true concurrent way.

Outline of the paper. In Section 2 we recall some notions on hypergraphs, partitions and the definition of multiplicative linear logic and its proof nets. In Section 3 we recall the definition and results from [44, 9] on the generalized multiplicative connectives (theorized in [21, 32]) we use in this paper. In Section 4 we give an overview of the way logic programming program executions are represented in using sequent calculi and how this paradigm has been extended in [15] to proof net expansion. In Section 5 we define our framework by extending the definition of proof structures, and proving the results about their compositionality. In Section 6 we provide a computational interpretation of certain generalized multiplicative connectives in our framework. We show that these connectives, beside being linear and context-free operators, are still able to capture non-linear and context-sensitive behaviors.

2 Preliminary Notions

In this section we recall basic definitions for hypergraphs and partitions we use in this paper. We then recall the definition of multiplicative linear logic and the syntax of its proof nets.

2.1 Hypergraphs

A **hypergraph** $\mathcal{G} = \langle V_\mathcal{G}, E_\mathcal{G} \rangle$ is given by a set of **vertices** $V_\mathcal{G}$ and a set of **hyperedges** $E_\mathcal{G}$, that is, a set of pairs of list of vertices in $V_\mathcal{G}$. In a hyperedge $\mathsf{h} = \langle \mathsf{in}(\mathsf{h}), \mathsf{out}(\mathsf{h}) \rangle \in E_\mathcal{G}$ we call the vertices occurring in $\mathsf{in}(\mathsf{h})$ (resp. in $\mathsf{out}(\mathsf{h})$) the **inputs** (resp. the **outputs**) of h and we define the **border** $\mathsf{B}(\mathsf{h})$ as the multiset of vertices occurring in $\mathsf{in}(\mathsf{h})$ and in $\mathsf{out}(\mathsf{h})$. A **sub-hypergraph** of a hypergraph $\mathcal{G} = \langle V_\mathcal{G}, E_\mathcal{G} \rangle$ is a hypergraph $\mathcal{G}' \langle V_{\mathcal{G}'}, E_{\mathcal{G}'} \rangle$ such that $V_{\mathcal{G}'} \subseteq V_\mathcal{G}$ and $E_{\mathcal{G}'} \subseteq \{\mathsf{h} \in E_\mathcal{G} \mid \mathsf{B}(\mathsf{h}) \subseteq V_{\mathcal{G}'}\}$.

An **input** (resp. **output**) of the hypergraph $\mathcal{G}$ is a vertex which does not occur as output (resp. input) of any hyperedge of $\mathcal{G}$. We denote by $\mathsf{I}_\mathcal{G}$ (resp. $\mathsf{O}_\mathcal{G}$) the set of inputs (resp. outputs) of $\mathcal{G}$, and we define the **border** of $\mathcal{G}$ as the set of its inputs and the outputs, that is, $\mathsf{B}_\mathcal{G} = \mathsf{I}_\mathcal{G} \cup \mathsf{O}_\mathcal{G}$. A hypergraph is **linear** if each vertex occurs at most once as an input of a hyperlink and as an output of another link[2].

In a hypergraph $\mathcal{G}$, a **path** (of length n) from $x \in V_\mathcal{G}$ to $y \in V_\mathcal{G}$ is an alternating list of vertices and hyperedges of the form $x = v_0 \mathsf{h}_1 v_1 \cdots \mathsf{h}_n v_n = y$ such that $v_{i-1} = \mathsf{out}(\mathsf{h}_i)$ and $v_i = \mathsf{in}(\mathsf{h}_i)$ for all $i \in \{1, \ldots, n\}$; in this case we say that x is connected to y. A **cycle** is a path with $\mathsf{h}_1 = \mathsf{h}_n$, or $n > 0$ and $v_0 = v_n$; it is **elementary** if there are no i and j such that $i \neq j$ and $v_i = v_j$ or $\mathsf{h}_i \neq \mathsf{h}_j$. A hypergraph is **acyclic** if it contains no elementary cycles.

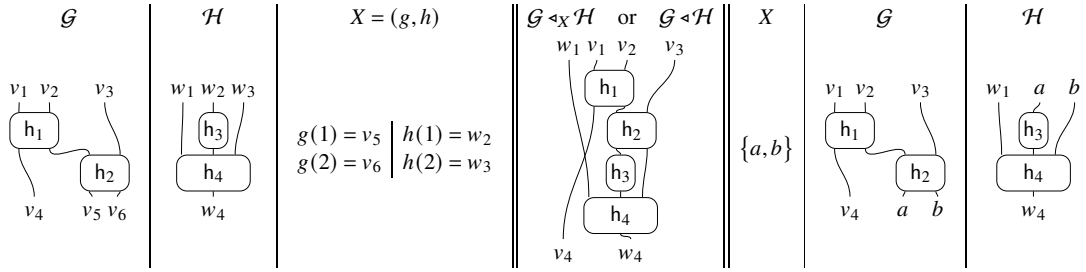
An **undirected hypergraph** $\mathcal{G} = \langle V_\mathcal{G}, E_\mathcal{G} \rangle$ is given by a set of **vertices** $V_\mathcal{G}$ and a set of **undirected hyperedges** $E_\mathcal{G}$, that is, a set of subset of vertices in $V_\mathcal{G}$. A **graph** is an undirected hypergraph in which each hyperedge is an **edge**, that is, a set of two vertices $\{v_1, v_2\}$. **Paths** and **cycles** in an undirected hypergraph are defined analogously to the ones in hypergraph. Two vertices are **connected** if there is a path from one to the other. A **connected component** of an undirected hypergraph is a maximal subset of pairwise connected vertices. The **undirected hypergraph** associated to a linear hypergraph $\mathcal{G} = \langle V_\mathcal{G}, E_\mathcal{G} \rangle$ is defined as the undirected hypergraph $\langle V_\mathcal{G}, \{\mathsf{B}(\mathsf{h}) \mid \mathsf{h} \in E_\mathcal{G}\} \rangle$. We say that a hypergraph $\mathcal{G}$ is **connected** if the undirected hypergraph associated to it is connected.

Notation 1. We drawing hyperedges with inputs on top and output on the bottom. By convention, we enumerate the inputs and outputs from left to right.

Definition 2. Let $\mathcal{G} = \langle V_\mathcal{G}, E_\mathcal{G} \rangle$ and $\mathcal{H} = \langle V_\mathcal{H}, E_\mathcal{H} \rangle$ be two hypergraphs with disjoint sets of vertices. The **disjoint union** of $\mathcal{G}$ and $\mathcal{H}$ (denoted $\mathcal{G} \| \mathcal{H}$) is defined as the union of vertices and hyperedges of

[2] In works on hypergraphs with interfaces (e.g., [19]), this property is referred to as *linearity* or *monogamicity*. Note that, by definition, no vertex can occur at the same time as an input and an output of a linear hyperedge.

$\mathcal{G}$ and $\mathcal{H}$ that is, $\mathcal{G}\|\mathcal{H} \coloneqq \langle V_\mathcal{G} \cup V_\mathcal{H}, E_\mathcal{G} \cup E_\mathcal{H} \rangle$. Note that in defining $\mathcal{G}\|\mathcal{H}$ we always assume $\mathcal{G}$ and $\mathcal{H}$ having disjoint sets of nodes. An **(linear) interface** $X = (g, h)$ is a pair of bijections from a finite set $\{1, \dots, n\}$ to $V_\mathcal{G}$ and $V_\mathcal{H}$ respectively. We define the **composition** of $\mathcal{G}$ and $\mathcal{H}$ via an interface X as the hypergraph $\mathcal{G} \triangleleft_X \mathcal{H}$ obtained by identifying in $\mathcal{G}\|\mathcal{H}$ the vertices $g(i)$ and $h(i)$ for each $i \in \{1, \dots, n\}$, as shown on the left of Equation (1).



(1)

To improve readability, we may simply identify the vertices $g(i)$ and $h(i)$ for all $i \in \{1, \dots, n\}$, therefore considering X as a set of vertices in $V_\mathcal{G} \cap V_\mathcal{H}$ and simply writing $\mathcal{G} \triangleleft \mathcal{H}$, as shown in the right of Equation (1).

2.2 Partitions

Given a set X, a **partition** of X is a set of disjoint subsets of X (we call each a **block**) such that their union is X. In order to improve readability, when writing sets of partitions, in which three parentheses are nested inside each other, even if blocks and partitions are sets (not permutations, nor multisets), we use parentheses $(-)$ to denote blocks (subsets of X), and square brackets $[-]$ to denote partitions (sets of subsets of X). For example, we write $[(1,3),(2)]$ to denote the partition of the set $\{1,2,3\}$ with one block containing 1 and 3 and one block containing only 2. We denote by $\mathbb{P}_X$ (resp. $\mathbb{P}_n$) the set of partitions over X (resp. over $\{1, \dots, n\}$). If $p \in \mathbb{P}_X$ and $Y \subset X$, we define the **restriction of** p **on** Y as the partition $p_{|_Y} \in \mathbb{P}_Y$ such that $x, y \in Y$ belongs to the same block $\gamma_{|_Y} \in p_{|_Y}$ iff x and y belongs in a same block $\gamma \in p$. By means of example, if $p = [(1,3,4),(2,5),(6)] \in \mathbb{P}_6$, then $p_{|_{\{1,2,3\}}} = [(1,3),(2)]$.

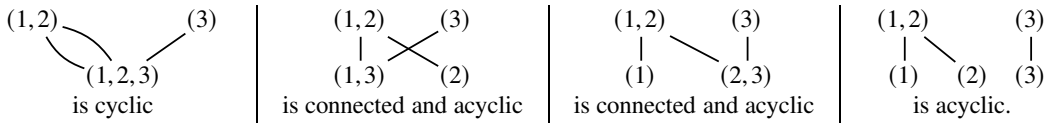
Definition 3 (Orthogonality [21])**.** Let X be a set and $p, q \in \mathbb{P}_X$. We define **graph of incidence of** p **and** q as the graph $\mathcal{G}(p,q)$ with vertices the blocks in p and in q and with an edge between a block $\gamma_p \in p$ and a block in $\gamma_q \in q$ for each $i \in \gamma_p \cap \gamma_q$ (see examples in Equation (2)). That is, the graph $\mathcal{G}(p,q) = \langle V_{\mathcal{G}(p,q)}, E_{\mathcal{G}(p,q)} \rangle$ has set of vertices and edges respectively

$$V_{\mathcal{G}(p,q)} = \{\gamma \mid \gamma \in p \text{ or } \gamma \in q\} \quad \text{and} \quad E_{\mathcal{G}(p,q)} = \{\{\gamma_i^p, \gamma_i^q\} \mid \gamma_i^p \in p \text{ and } \gamma_i^q \in q \text{ and } i \in \gamma_i^p \cap \gamma_i^q\}\,.$$

We say that $p, q \in \mathbb{P}_X$ are **weakly orthogonal**, denoted $p \perp_w q$, if their graph of incidence $\mathcal{G}(p,q)$ is acyclic. They are **orthogonal**, denoted $p \perp q$, if their graph of incidence is connected and acyclic.

The notion of weak orthogonality and orthogonality extends to sets of partitions: if $P, Q \subset \mathbb{P}_X$ then $P \perp_w Q$ (respectively $P \perp Q$) if $p \perp_w q$ (respectively $p \perp q$) for all $p \in P$ and for all $q \in Q$. The **orthogonal set** of a set of partitions $P \subset \mathbb{P}_n$ is defined as $P^\perp = \{q \in \mathbb{P}_n \mid p \perp q \text{ for all } p \in P\}$. We write $P \perp\!\!\!\perp Q$ if $P \perp Q$ and $P^\perp \perp Q^\perp$.

Example 4. Consider the partitions $p = [(1,2),(3)]$, $q_1 = [(1,2,3)]$, $q_2 = [(1,3),(2)]$ $q_3 = [(1),(2,3)]$ and $q_4 = [(1),(2),(3)]$. We have that $p \not\perp q_1$, $p \perp q_2$, $p \perp q_3$, $p \perp_w q_4$ because



(2)

$$\frac{}{\vdash a, a^\perp}\,\mathsf{ax} \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\,\parr \qquad \frac{\vdash \Gamma, A \quad \vdash B, \Delta}{\vdash \Gamma, A \otimes B, \Delta}\,\otimes \qquad \Bigg| \qquad \frac{\vdash \Gamma \quad \vdash \Delta}{\vdash \Gamma, \Delta}\,\mathsf{mix} \qquad \Bigg| \qquad \frac{\vdash \Gamma, A \quad \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}\,\mathsf{cut}$$

Figure 2: Sequent calculus rules for the multiplicative linear logic, and rules mix and cut.

2.3 Multiplicative Linear Logic

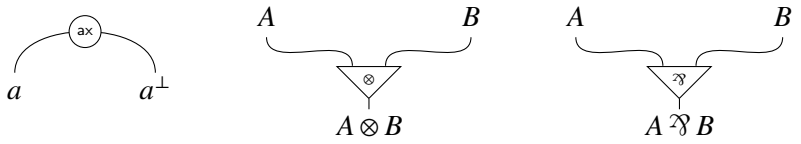
Multiplicative linear logic formulas are generated from a countable set $\mathcal{A} = \{a, b, \dots\}$ of **propositional variables** by the following grammar:

$$A, B ::= a \mid A^\perp \mid A \parr B \mid A \otimes B$$

We consider formulas modulo the **involutivity of the negation** $A^{\perp\perp} = A$ and the **De Morgan** laws $(A \otimes B)^\perp = A^\perp \parr B^\perp$ and $(A \parr B)^\perp = A^\perp \otimes B^\perp$.

A **sequent** is a set of occurrences of formulas (as in, e.g., [17, 5]). The **sequent systems** $\mathsf{MLL} = \{\mathsf{ax}, \parr, \otimes\}$ and $\mathsf{MLL}^\circ = \mathsf{MLL} \cup \{\mathsf{mix}\}$ are defined using the rules in Figure 2[3]. We call **active** (resp. **principal**) a formula occurrence in one of the premises (resp. in the conclusion) of a rule, not occurring in conclusion (resp. in any of its premises).

Multiplicative Proof Nets *Proof nets* are a graphical syntax for multiplicative linear logic proofs capturing the proof equivalence generated by independent rules permutations (see, e.g., [31, 41, 36, 40, 10, 37, 1, 11]). A **proof structure** $\mathcal{S} = \langle V_\mathcal{S}, E_\mathcal{S} \rangle$ is a hypergraph whose vertices are labeled by MLL-formulas and whose hyperedges (called links) are labeled by rules in MLL (such labels are called **types**) in such a way the following local constraints are respected:



(3)

with A and B formulas and $a \in \mathcal{A}$.

Since we are considering a sequent as a set of occurrences of formulas, it is possible to easily trace formula occurrences in a derivation, defining a proof structure that encodes a given derivation.

Definition 5. Let π be a derivation in MLL. We define the proof structure **representing** π as the proof structure $\mathcal{P}_\pi$ having a vertex for each occurrence of an active formulas of a rule in π, and a link of type ρ with inputs (resp. with output) the vertices corresponding to the active formulas (resp. the principal formula) for each occurrence of a rule ρ in π. A **proof net** is a proof structure $\mathcal{S} = \mathcal{P}_\pi$ representing a derivation π in MLL.

By definition not all proof structures are proof nets. For this reason, a **correctness criterion**, that is, a topological characterization of those proof structures which are proof nets, is needed. Beside various criteria have been developed in the literature [30, 21, 55, 34], we here report the so-called **Danos-Regnier criterion** (or **DR-criterion** for short), which is the most relevant to our purposes.

[3] In the figure we include the rule cut required to define compositionality of proofs via *modus ponens*. We do not include it in the definition of MLL and MLL° since this rule is proven to be admissible [30] and it plays no role in the framework we are presenting in this paper.

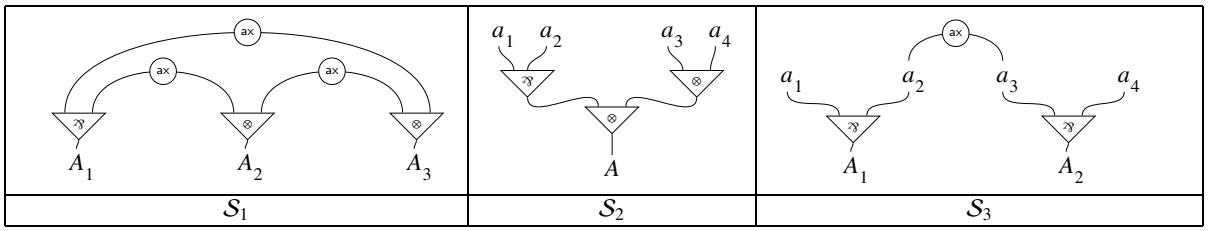


Figure 3: Examples of proof structures: $\mathcal{S}_1$ is a proof net (therefore a module), $\mathcal{S}_2$ is a module, and $\mathcal{S}_3$ is not a module (it admits a test in which a_2 and a_3 are disconnected from any other vertex, therefore $\mathcal{S}_3$ cannot be a sub-hypergraph of a proof net).

Definition 6. Let $\mathcal{M}$ be a proof structure. A **test** for $\mathcal{M}$ is the undirected hypergraph with the same vertices of $\mathcal{M}$ having a hyperedge $\{a, a^\perp\}$ for each ax-link $(\emptyset, \langle a, a^\perp\rangle)$, a hyperedge $\{A, B, A \otimes B\}$ for each $\otimes$-link $(\langle A, B\rangle, \langle A \otimes B\rangle)$, and either one edge $\{A, A \parr B\}$ or one edge $\{B, A \parr B\}$ for each $\parr$-link $(\langle A, B\rangle, \langle A \parr B\rangle)$. The proof structure $\mathcal{M}$ is **DR-correct** if it has no inputs and if all of its tests are connected and acyclic (undirected) hypergraph.

Theorem 7 ([21])**.** *A proof structure $\mathcal{S}$ is a proof net iff $\mathcal{S}$ is DR-correct.*

It is worth noticing that by dropping connectedness condition in Definition 6, we obtain a notion of correctness for MLL°-proof net, that is, if any test of a proof structure $\mathcal{M}$ is an acyclic hypergraph, then we can associate to $\mathcal{M}$ a derivation in MLL°.

Definition 8. A **module** is a proof structure which is a connected sub-hypergraph of a proof net.

Remark 9 (Definitions of module in the literature)**.** The definition of module we consider in this paper differs from the definition of module given in [21] where a module is defined as a pair $\langle \mathcal{S}, \mathsf{Y}\rangle$ with $\mathcal{S}$ a proof structure such that $\sigma(\mathcal{S})$ is acyclic for all $\sigma \in \Sigma(\mathcal{S})$, and a subset of its border $\mathsf{Y} \subseteq \mathsf{B}_\mathcal{S}$.

3 Generalized Connectives in Multiplicative Linear Logic

The notion of generalized (multiplicative) connectives for multiplicative linear logic was introduced since the early works on linear logic [21]. We say that an inference rule of the sequent calculus is **linear** if each occurrence of subformula (except the principal formula of the rule) occurring in the conclusion of the rule occurs exactly once in its premise(s), and it is **context-free** if no conditions on the non-principal formulas affect the application of the rule. A rule is **multiplicative** if linear and context-free.

Example 10. Consider the three rules in Equation (4) below. Only the leftmost is multiplicative: the central one is not linear since the subformula B does not occur in the premise, while the rightmost one is not context-free since the rule requires the sequent to contain an odd number of formulas.

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B, C}{\vdash \Gamma, \Delta, A \otimes (B \parr C)}\, a \otimes (b \parr c) \qquad \frac{\Gamma, A}{\vdash \Gamma, A \parr B}\, \mathsf{W}_\parr \qquad \frac{\vdash C_1, \ldots, C_{2k-1}, A \quad \vdash C_2, \ldots, C_{2k}, B}{\vdash C_1, C_2, \ldots, C_{2k-1}, C_{2k}, A \otimes B}\, \mathsf{odd}_\otimes \tag{4}$$

In [21] the authors observe that any multiplicative rule can be fully described by a partition (having a block for each of the rule premises) keeping track of how active formulas are distributed among the premise of the rule. Thus, we can define so called **synthetic rules** (see, e.g., [21, 32, 52]), allowing us to gather multiple inference of multiplicative rules to construct a formula by a single rule application, as shown in the following example.

Example 11. Consider the following formulas, their *synthetic rules* and the associated partitions.

Formula	$F = a \otimes (b \parr c)$	$G = (a \parr b) \otimes (c \parr d)$	$H = a \parr (b \otimes c)$
Derivation(s)	$\dfrac{\dfrac{\Delta, b, c}{\Delta, b \parr c}\parr \quad \Gamma, a}{\Gamma, \Delta, a \otimes (b \parr c)}\otimes$	$\dfrac{\dfrac{\Delta, a, b}{\Delta, a \parr b}\parr \quad \dfrac{\Delta, c, d}{\Delta, c \parr d}\parr}{\Gamma, \Delta, (a \parr b) \otimes (c \parr d)}\otimes$	$\dfrac{\Gamma, a, b \quad \Delta, c}{\Gamma, \Delta, a, b \otimes c}\otimes$ and $\dfrac{\Gamma, a, c \quad \Delta, b}{\Gamma, \Delta, a, b \otimes c}\otimes$
Synthetic Rule(s)	$\dfrac{\vdash \Gamma, a \quad \vdash \Delta, b, c}{\vdash \Gamma, \Delta, a \otimes (b \parr c)}\mathcal{R}_F$	$\dfrac{\vdash \Gamma, a, b \quad \vdash \Delta, c, D}{\vdash \Gamma, \Delta, (a \parr b) \otimes (c \parr D)}\mathcal{R}_G$	$\dfrac{\vdash \Gamma, a, b \quad \vdash \Delta, c}{\vdash \Gamma, \Delta, a \parr (b \otimes c)}\mathcal{R}_{H_1}$ and $\dfrac{\vdash \Gamma, a, c \quad \vdash \Delta, b}{\vdash \Gamma, \Delta, a \parr (b \otimes c)}\mathcal{R}_{H_2}$
Associated partitions	$[(1), (2,3)]$	$[(1,2), (3,4)]$	$[(1,2), (3)]$ and $[(1,3), (2)]$

Conversely, given a set of partitions P in $\mathbb{P}_n$, we can define a rule introducing an n-ary **generalized connective** C_P for each partition in P. In this case, we say that P is the **behavior** of C_P. Consider the following examples.

Behavior	$P = \{[(1,2), (3,4)], (1,4), (2,3)\}$	$Q = \{[(1,3), (2), (4)], [(1), (2,4), (3)]\}$
Rules	$\dfrac{\vdash \Gamma, A, B \quad \vdash \Delta, C, D}{\vdash \Gamma, \Delta, \mathsf{C}_P(A, B, C, D)}\,[(1,2), (3,4)]$ $\dfrac{\vdash \Gamma, A, D \quad \vdash \Delta, B, C}{\vdash \Gamma, \Delta, \mathsf{C}_P(A, B, C, D)}\,[(1,4), (2,3)]$	$\dfrac{\vdash \Gamma, A, C \quad \vdash \Delta, B \quad \vdash \Sigma, D}{\vdash \Gamma, \Delta, \Sigma, \mathsf{C}_Q(A, B, C, D)}\,[(1,3), (2), (4)]$ $\dfrac{\vdash \Gamma, A \quad \vdash \Delta, B, D \quad \vdash \Sigma, C}{\vdash \Gamma, \Delta, \Sigma, \mathsf{C}_Q(A, B, C, D)}\,[(1), (2,4), (3)]$

(5)

However, as shown in various works [21, 32, 44, 9], not all sets of partitions can be considered to be satisfactory in order to define connectives. In fact, we allow to use a set of partitions $P \subset \mathbb{P}_n$ to describe a connective only if P admits a set Q such that $P \perp Q$. This condition is mandatory to guarantee the possibly to define a dual connective whose rules well-behave with respect to cut-elimination.

In [44, 9] it has been proved that there are families of sets of partitions which can be used to describe behaviors different from any synthetic rule defined using $\otimes$ and $\parr$ rules. Moreover, in [9] it is also shown that no satisfactory sequent calculus can be defined in presence of generalized connectives due to the lack of the so-called **initial coherence** [52, 16] (also called *packaging problem* in [21]), that is, the possibility of having a proof system in which it is always possible to prove "A implies A" using atomic axiom only.

3.1 Generalized Connectives in Multiplicative Proof Nets

Sets of partitions have been used to define generalized connectives in the proof net syntax in [21, 32, 48, 44, 9], overcoming the aforementioned problem of initial coherence. We here give some intuitions on these connectives, while more precise definitions are provided in Section 5 where we properly define the formal setting required to accommodate them.

Generalized connectives in multiplicative proof structures use sets of partitions to define the *behavior* of new connectives, that is, the way *tests* are constructed. Intuitively, the behavior of $\parr$ (defined as $\{[(1), (2)]\}$) and $\otimes$ (defined as $\{[(1,2)]\}$) provide the topological constraints of the definition of the test: for $\parr$ the link is replaced by a hyperedge connecting only one of the two inputs (connecting the output to one of the two blocks) while for the $\otimes$ the link is replaced by hyperedge connecting both inputs (since both belong to the same block). Similarly, in defining a test for a link with a given behavior is replaced by certain hyperedges connecting the vertices in a same block.

In this case, given an n-ary connective and a partition $P \subset \mathbb{P}_n$, the condition of the existence of a Q such that $P \perp Q$ is not enough to guarantee the existence of a dual connective well-behaving with respect

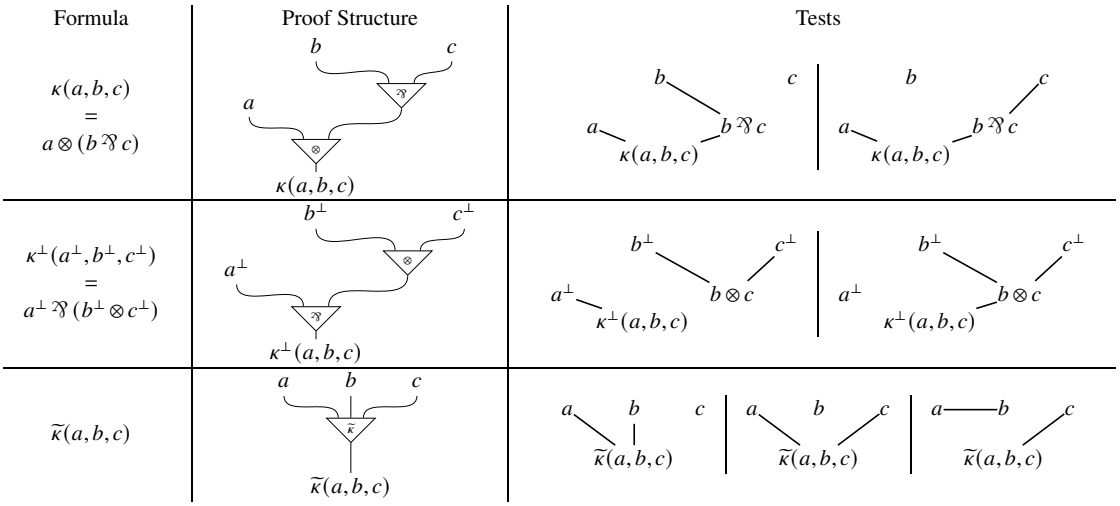


Figure 4: The behaviors of the synthetic connectives associated to the formula $F(a,b,c) = a \otimes (b \parr c)$, to its dual formula, and to a generalized multiplicative connective whose behavior strictly contains the one of $F(a,b,c)$ in such a way is the same of $F(a,b,c)$ if restricted on the inputs only.

to cut-elimination. Thus the stronger condition $P \Subset Q$ is needed[4].

Remark 12. As noticed in [44, 9], the definition of more-than-binary generalized connectives requires to include the information about which block of inputs is connected to the output of the link connective. This information is only required to ensure a sound cut-elimination procedure, and it is lost after removing cuts. Said differently, the contextual equivalence defined by cut-elimination is not able to distinguish certain behaviors differing in the way set of inputs are connected to the output. This information is not relevant for the standard MLL connectives nor for *synthetic connectives* (that is, the ones which can indirectly defined by means of combination of $\otimes$ and $\parr$; see Example 11), since it can be indirectly derived using the less complex nature of these connectives, which are defined inductively using binary ones. Nevertheless, this information is not negligible in the general case, since this information may define different tests as shown in the following example explaining in detail Figure 4.

Example 13. Consider the formula $F(a,b,c) = a \otimes (b \parr c)$ and the synthetic connectives $\kappa(a,b,c) \coloneqq F(a,b,c)$ and $\kappa^\perp(a^\perp,b^\perp,c^\perp) \coloneqq F^\perp(a^\perp,b^\perp,c^\perp)$ respectively associated to the sets of partitions $\mathfrak{B}_\kappa = \{[(1,2),(3)],[(1,3),(2)]\}$ and $\mathfrak{B}_{\kappa^\perp} = \{[(1),(2,3)]\}$ (see Figure 4).

We can now define the connective $\widetilde{\kappa}$ associated to the same set of partitions of κ (that is, the set of partitions $\mathfrak{B}_{\widetilde{\kappa}} = \mathfrak{B}_\kappa = \{[(1,2),(3)],[(1,3),(2)]\}$) but in which we allow an extra test which enforces no new partitions among the inputs. See the bottom-most row of the table in Figure 4, where the new test (the right-most one) enforces the partition $[(1,2),(3)] \in \mathfrak{B}_\kappa$ over inputs.

Since κ and $\widetilde{\kappa}$ are defined by the same set of partitions over their inputs, they are both orthogonal to $\kappa^\perp$. Moreover, both DR-correctproof structures of $\kappa(a,b,c) \parr \kappa^\perp(a^\perp,b^\perp,c^\perp)$ and $\widetilde{\kappa}(a,b,c) \parr \kappa^\perp(a^\perp,b^\perp,c^\perp)$ are correct, and the result of cut-elimination of a κ- or a $\widetilde{\kappa}$-gate against a $\kappa^\perp$-gate reduces

[4]Note that in [21, 32] each multiplicative connective is defined by a pair of sets of dual partitions over the same finite set satisfying an orthogonality condition. This condition is sufficient to fully describe these connectives in sequent calculus style, and we here show that it is also sufficient for our proof net expansion paradigm. However, it is well-known that in a Curry-Howard oriented interpretation of proof-as-program paradigm stronger conditions are required in order to guarantee a sound dynamic of cut-elimination (that is, not only the two partitions must be orthogonal, but also their orthogonal sets must be so).

to a proof structure with the same behavior. Note that this implies that κ and $\widetilde{\kappa}$ are indistinguishable with respect to the notion of context equivalence usually considered on proof structures (see, e.g., [33, 25, 27]).

4 Logic Programming with Multiplicative Structures

In this section we recall the results from [14, 15] (restricted to the multiplicative linear logic fragment) on the possibility to define a logic programming framework based on proof net construction.

The classical interpretation of logic programming (see, e.g., [12, 14, 52]), a program is defined by a set of sequent calculus rules and its execution is conceived as the process of expanding the open branches of the derivation tree of a given formula. This correspondence can easily be extended using synthetic (linear) inference rules as the ones from Example 11 to define the following methods:

$$\mathbf{F} :- \mathbf{a}, (\mathbf{b} \parr \mathbf{c}) \quad \Big| \quad \mathbf{G} :- (\mathbf{a} \parr \mathbf{b}), (\mathbf{c} \parr \mathbf{d}) \quad \Big| \quad \mathbf{H_1} :- (\mathbf{a} \parr \mathbf{b}), \mathbf{c} \qquad \mathbf{H_2} :- (\mathbf{a} \parr \mathbf{c}), \mathbf{b} \tag{6}$$

In particular, a specific family of formulas (called *bipoles*) can be used to define methods.

Definition 14 ([14])**.** Given a set of **negative** atoms $\mathcal{A}$ whose negations are **positive**, a **monopole** is a disjunction ($\parr$) of negative atoms. A **bipole** is a conjunction ($\otimes$) of monopoles and positive atoms which contains at least one positive atom. Given a set of bipoles $\mathfrak{F}$, the **focussing bipolar sequent calculus** $[\mathfrak{F}, \mathcal{A}]$ is given by the set of inference rules of the following form, where F is a bipole in $\mathfrak{F}$.

$$\frac{\vdash \mathsf{i}_{1,1}, \ldots, \mathsf{i}_{1,k_1} \quad \cdots \quad \vdash \mathsf{i}_{i,1}, \ldots, \mathsf{i}_{i,k_i}}{\vdash \mathsf{o}_1, \ldots, \mathsf{o}_n} \; F = \mathsf{o}_1^\perp \otimes \cdots \otimes \mathsf{o}_n^\perp \otimes (\mathsf{i}_{1,1} \parr \cdots \parr \mathsf{i}_{1,k_1}) \otimes \cdots \otimes (\mathsf{i}_{i,1} \parr \cdots \parr \mathsf{i}_{i,k_i}) \tag{7}$$

As shown in [12], a bipole F can be seen as a logic programming method having as **head** (or **trigger**) the subformula containing the positive atoms of F (a conjunction), and as **body** the subformula containing the negative atoms F (a CNF formula). Intuitively, each bipole F induces a synthetic rule with principal formula F and and whose active formulas are its positive atoms gathered in a same premise if they belong to the same conjunct. By means of example the rule for the bipole F in Equation (7) can be seen as a synthetic rule introducing the formula F corresponding to the following derivation

$$\dfrac{\dfrac{\dfrac{\mathsf{i}_{1,1}, \ldots, \mathsf{i}_{1,k_1}}{\mathsf{i}_{1,1} \parr \cdots \parr \mathsf{i}_{1,k_1}} \parr \quad \cdots \quad \dfrac{\mathsf{i}_{i,1}, \ldots, \mathsf{i}_{i,k_i}}{\mathsf{i}_{i,1} \parr \cdots \parr \mathsf{i}_{i,k_i}} \parr}{(\mathsf{i}_{1,1} \parr \cdots \parr \mathsf{i}_{1,k_1}) \otimes \cdots \otimes (\mathsf{i}_{i,1} \parr \cdots \parr \mathsf{i}_{i,k_i})} \otimes \quad \dfrac{\dfrac{}{\vdash \mathsf{o}_1^\perp, \mathsf{o}_1} \mathsf{ax} \quad \cdots \quad \dfrac{}{\vdash \mathsf{o}_n^\perp, \mathsf{o}_n} \mathsf{ax}}{\mathsf{o}_1^\perp \otimes \cdots \otimes \mathsf{o}_n^\perp, \mathsf{o}_1, \cdots, \mathsf{o}_n} \otimes}{\mathsf{o}_1^\perp \otimes \cdots \otimes \mathsf{o}_n^\perp \otimes (\mathsf{i}_{1,1} \parr \cdots \parr \mathsf{i}_{1,k_1}) \otimes \cdots \otimes (\mathsf{i}_{i,1} \parr \cdots \parr \mathsf{i}_{i,k_i}), \mathsf{o}_1, \cdots, \mathsf{o}_n} \otimes \tag{8}$$

In [13] it has been proved that the focussing bipolar sequent system with one rule for each MLL bipole is sound and complete with respect to MLL.

4.1 Bipolar Proof Nets

The idea of using the focussing bipolar sequent calculus has been further developed in [15], where the authors proposed to model such a framework using proof nets construction instead of proof search in sequent calculi. The main advantage of the graphical syntax with respect to the bipolar sequent calculus is that in this latter, even if this rule admits a non-singleton trigger, a rule can expand only a single branch of a derivation. In fact, the tree-like structure of sequent calculus syntax allows us to expand one leaf of the derivation tree at a time by applying a rule. For instance, consider Figure 5 where the concurrent

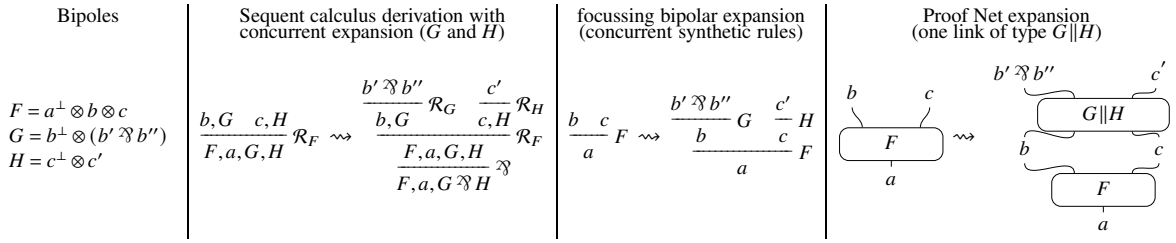
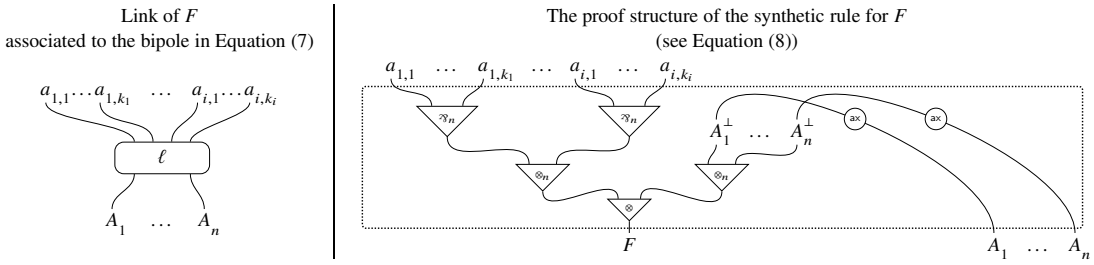


Figure 5: A concurrent application of the bipoles G and H after F represented in different formalisms

application of two methods on two different branches of a derivation can be represented by an expansion of a proof structure with a single link.

More precisely, we can use bipoles to define new link types in the same manner a bipole defines a new synthetic inference rule in the sequent calculus. Each test replaces such a link with one of the possible tests of the proof structure representing the bipole. By means of example, the bipolar rule from Equation (7) could be used to define the link below on the left, and tests would replace such a link with any test of the proof structure below on the right, which represents the open derivation in Equation (8).



(9)

Note that the link above on the left has outputs $o_1, \ldots, o_n$, while the proof structure on the right has an additional output F. In the next section we provide a solution to address this mismatch (see no-output links in Definition 16), but it is worth noting that we can define links representing concurrent application of bipoles by simply connecting those additional outputs via a $\parr$-link. Analyzing the shape of the proof structure describing a *concurrent bipole*.

Definition 15. We introduce the following naming for specific proof structure (see examples in Figure 6):

- **body**: a $\otimes_n$-link collecting the outputs of $\parr_n$-links (i.e., the proof structure of a CNF-formula). The body gathers the clauses corresponding to the body of a method;
- **header**: a bundle of ax-links attached to a $\otimes_n$-link by exactly one of their two outputs each. The header gathers the outputs corresponding to the head of a method;
- **synchronizer**: a $\parr_n$-link collecting the outputs of $\otimes$-links (i.e., the proof structure of a DNF-formula). The synchronizer establishes a connection between headers of methods and their bodies.

A **concurrent bipole** is a proof structure made of a synchronizer whose inputs are attached to headers

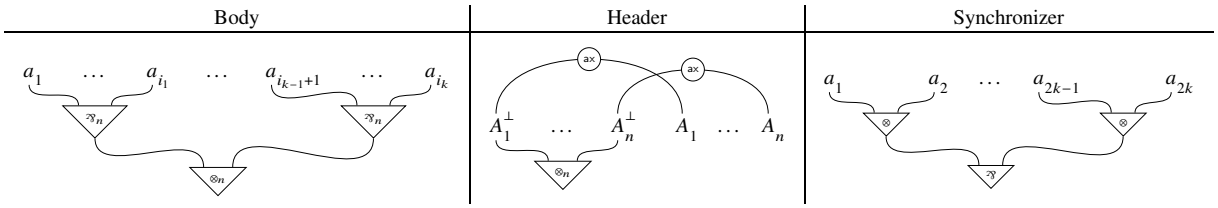
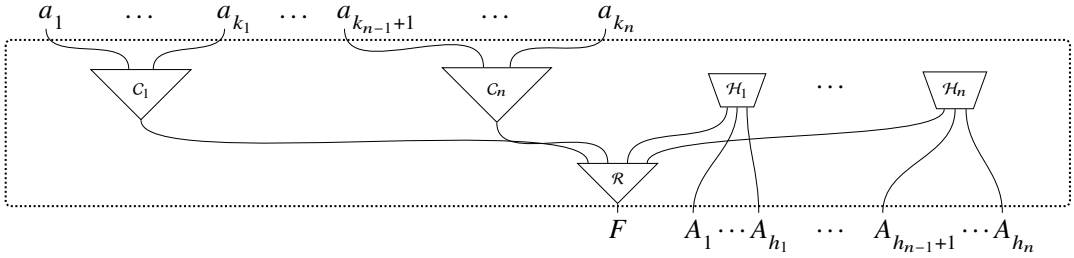


Figure 6: Components of a concurrent bipolar link.

and bodies. That is, a proof structure of the following shape:



(10)

where $\mathcal{H}_1, \ldots, \mathcal{H}_n$ are headers, $C_1, \ldots, C_n$ are bodies, and $\mathcal{R}$ is a synchronizer.

5 Generalizing Multiplicative Proof Structures

In this section we provide a general setting to define hypergraphs with hyperedges labeled by sets of partitions generalizing the syntax of proof structures, allowing us to accommodate generalized connectives, generalize the DR-correct, and define the notion of a *component* as a "proof structure which may be a piece of a proof net".

Definition 16. A **link type** (or simply **type**) is a triple $\langle n, m, \mathfrak{B} \rangle$ given by two natural numbers $n, m \in \mathbb{N}$ and a **behavior** $\mathfrak{B} \subseteq \mathbb{P}_{\{\mathsf{i}_1,\ldots,\mathsf{i}_n,\mathsf{o}_1,\ldots,\mathsf{o}_m\}}$. We define the following link types:

$$\mathsf{ax} = \langle 0, 2, \{[(\mathsf{o}_1, \mathsf{o}_2)]\} \rangle \qquad \otimes = \langle 2, 1, \{[(\mathsf{i}_1, \mathsf{i}_2, \mathsf{o}_1)]\} \rangle \qquad \parr = \langle 2, 1, \{[(\mathsf{i}_1, \mathsf{o}_1), (\mathsf{i}_2)], [(\mathsf{i}_1), (\mathsf{i}_2, \mathsf{o}_1)]\} \rangle$$
$$\mathsf{cut} = \langle 2, 0, [(\mathsf{o}_1, \mathsf{o}_2)] \rangle \qquad \parr^\bullet_n = \langle n, 0, [(\mathsf{o}_1), \ldots, (\mathsf{o}_n)] \rangle \qquad \otimes^\bullet_n = \langle n, 0, [(\mathsf{o}_1, \ldots, \mathsf{o}_n)] \rangle$$
$$\otimes_n = \langle n, 1, \{[(\mathsf{i}_1, \ldots, \mathsf{i}_n, \mathsf{o}_1)]\} \rangle \qquad \parr_n = \left\langle n, 1, \{[(\mathsf{i}_1), \ldots, (\mathsf{i}_{k-1}), (\mathsf{i}_k, \mathsf{o}_1), (\mathsf{i}_{k+1}), \ldots, (\mathsf{i}_n)]\}_{k \in \{1,\ldots,n\}} \right\rangle$$

Remark 17. By definition, $\otimes_2 = \otimes$ and $\parr_2 = \parr$ and $\mathsf{cut} = \otimes^\bullet_2$. The type $\otimes_1 = \parr_1$ can be thought as an edge connecting the input with the output since they both have behavior $[(\mathsf{i}_1, \mathsf{o}_1)]$. The type $\otimes^\bullet_1 = \parr^\bullet_1$ can be thought as a "dead-end" hyperedge with one input and no output (their behavior is $[(\mathsf{i}_1)]$).

Definition 18. A **multiplicative structure** over a signature Σ is a linear hypergraph $\mathcal{H}$ such that each hyperedge ℓ is labelled with a $\langle n, m, \mathfrak{B} \rangle \in \Sigma$ such that $|\mathsf{in}(\mathsf{h})| = n$ and $|\mathsf{out}(\mathsf{h})| = m$.

In drawing multiplicative structures, we label hyperedges by the corresponding type. The definition of **sub-multiplicative structure**, as well as the definition of **sequential** and **parallel composition** of multiplicative structures are defined extending the ones for hypergraphs.

In order to extend the notion of DR-correct, we need to provide a way to define tests.

Definition 19. Let $\mathcal{S} = \langle V_\mathcal{G}, E_\mathcal{S} \rangle$ be a multiplicative structure. A **switching** for $\mathcal{S}$ is a map σ assigning to each link ℓ a single partition $\sigma(\ell) \in \mathfrak{B}_\ell$. We denote by $\Sigma(\mathcal{S})$ the set of all possible switchings for $\mathcal{S}$. The

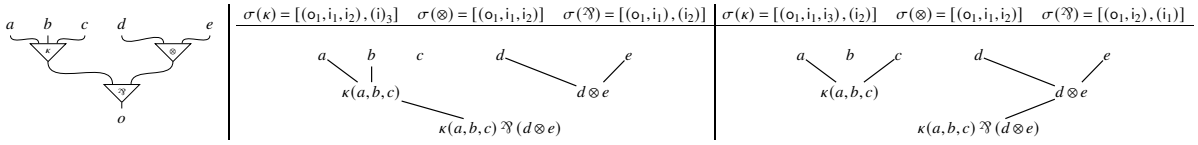


Figure 7: A proof structure where $\kappa = \langle 3, 1, \{[(\mathsf{o}_1, \mathsf{i}_1, \mathsf{i}_2), (\mathsf{i})_3], [(\mathsf{o}_1, \mathsf{i}_1, \mathsf{i}_3), (\mathsf{i}_2)]\}\rangle$ and two of its test

test $\sigma(\mathcal{S})$ defined by the switching σ is the undirected hypergraph obtained by replacing each link in $E_{\mathcal{S}}$ with one undirected hyperedge for each block in $\sigma(\ell)$. Formally, $\sigma(\mathcal{S}) = \left\langle V_{\mathcal{S}} \, , \, \bigcup_{\ell \in E_{\mathcal{S}}} \{\gamma \mid \gamma \in \sigma(\ell)\} \right\rangle$.

The **behavior** of a test σ is the partition p_σ of the border of $\mathcal{S}$ defined by the connected components of $\sigma(\mathcal{S})$. That is, $x, y \in \mathsf{B}_{\mathcal{S}}$ belongs to the same block $\gamma \in p_\sigma$ iff the vertices x and y are connected of $\sigma(\mathcal{S})$. The **behavior** of a multiplicative structure $\mathcal{S}$ is defined as the set of behaviors of its tests, that is, $\mathfrak{B}_{\mathcal{S}} = \left\{p_\sigma \in \mathbb{P}_{\mathsf{B}_{\mathcal{S}}} \mid \sigma \in \Sigma(\mathcal{S})\right\}$.

The behavior of a multiplicative structure is the collection of the information on how a multiplicative structure interacts with any possible context. It keeps track of the connectivity the vertices in its border in all its tests (see an example in Figure 7).

Definition 20. Let $\mathcal{S}$ be a multiplicative structure. We say that $\mathcal{S}$ is **correct** if $\sigma(\mathcal{S})$ is connected and acyclic for any test $\sigma \in \Sigma(\mathcal{S})$. It is **multiplicative net** if correct and if $\mathcal{S}$ has no inputs and at least one output. If each test $\sigma(\mathcal{S})$ of $\mathcal{S}$ is acyclic and each of its vertices is connected to a vertex of the border, then we say that $\mathcal{S}$ is a **(multiplicative) component**. A **transitory** component (or **⊤-component**) is a component such that each input admits a test where it is connected to an output. A **module** $\mathcal{M}$ is a connected non-empty multiplicative structure such that $\mathcal{M} \subset \mathcal{S}$ for a multiplicative net $\mathcal{S}$.

Example 21. Consider the examples in Figure 3. The multiplicative structure $\mathcal{S}_1$ is a multiplicative net (therefore a ⊤-component). $\mathcal{S}_2$ is a component and a module, but not a ⊤-component (it has no outputs). The multiplicative structure $\mathcal{S}_3$ is not a component (it admits a test in which a_2 and a_3 are disconnected from any other vertex) nor a module (there is no multiplicative net containing $\mathcal{S}_3$ as a sub-multiplicative structure).

Theorem 22. *All modules are components.*

Proof. If $\mathcal{M}$ is a module, then each test $\sigma(\mathcal{M})$ is acyclic, otherwise there should be a cycle in a test of $\mathcal{S}$. Moreover, as consequence of the fact that $\mathcal{S}$ is connected, no sub-multiplicative structure $\mathcal{S}'$ of $\mathcal{S}$ such that $\mathsf{B}_{\mathcal{S}'} = \emptyset$. Therefore each vertex in $\mathcal{M}$ must be connected to a vertex in $\mathsf{B}_{\mathcal{M}}$ in each test $\sigma(\mathcal{M})$ otherwise $\mathcal{S}$ would not be connected. □

Definition 23. Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be components and $X \subseteq \left(\mathsf{I}_{\mathcal{M}_1} \cap \mathsf{O}_{\mathcal{M}_2}\right)$ non-empty. We say that $\mathcal{M}_2$ **expands** $\mathcal{M}_1$ (on X) if $\mathcal{M}_1 \triangleleft_X \mathcal{M}_2$ is a component.

Theorem 24. *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be components and $X \subseteq \left(\mathsf{I}_{\mathcal{M}_1} \cap \mathsf{O}_{\mathcal{M}_2}\right)$ non-empty. Then*

$$\mathcal{M}_2 \textit{ expands } \mathcal{M}_1 \textit{ on } X \iff \begin{cases} X \neq \mathsf{B}_{\mathcal{M}_1} \cup \mathsf{B}_{\mathcal{M}_2} \\ \mathfrak{B}_{\mathcal{M}_1|_X} \perp_w \mathfrak{B}_{\mathcal{M}_2|_X} \\ \textit{each } x \in X \textit{ is connected either to a } y \in \left(\mathsf{B}_{\mathcal{M}_1} \setminus X\right) \textit{ in each test of } \mathcal{M}_1, \\ \qquad\qquad \textit{or to a } z \in \left(\mathsf{B}_{\mathcal{M}_2} \setminus X\right) \textit{ in each test of } \mathcal{M}_2 \end{cases}$$

Proof. By definition of component, letting $\mathcal{M} = \mathcal{M}_1 \triangleleft_X \mathcal{M}_2$.

($\Rightarrow$) If $\mathcal{M}$ is a component, then $\mathsf{B}_{\mathcal{M}} = \left(\mathsf{B}_{\mathcal{M}_1} \cup \mathsf{B}_{\mathcal{M}_2}\right) \setminus X$ must be non-empty, thus $X \neq \left(\mathsf{B}_{\mathcal{M}_1} \cup \mathsf{B}_{\mathcal{M}_2}\right)$.

If we assume that $\mathfrak{B}_{\mathcal{M}_1|_X} \not\perp_w \mathfrak{B}_{\mathcal{M}_2|_X}$, then there are $x, y \in X$ such that $\{x, y\} \subset \gamma_1 \in p \in \mathfrak{B}_{\mathcal{M}_1|_X}$ and $\{x, y\} \subset \gamma_2 \in q \in \mathfrak{B}_{\mathcal{M}_2|_X}$. That is, there is a path between x and y in both $\sigma_1(\mathcal{M}_1)$ and $\sigma_2(\mathcal{M}_2)$ with $\sigma_i \in \Sigma(\mathcal{M}_i)$ and $i \in \{1,2\}$. Thus there is a switch $\sigma \in \Sigma(\mathcal{M})$ such that $\sigma(\mathcal{M})$ contains a cycle. This contradicts the hypothesis of $\mathcal{M}$ being a module.

Finally, since each vertex of $\mathcal{M}$ is connected to a vertex in $\mathsf{B}_{\mathcal{M}} = \left(\mathsf{B}_{\mathcal{M}_1} \cup \mathsf{B}_{\mathcal{M}_2}\right) \setminus X$ in each test, then, each $x \in X$ is.

($\Leftarrow$) Since $\mathcal{M}_1$ and $\mathcal{M}_2$ are components, then $\sigma(\mathcal{M}_1)$ and $\sigma(\mathcal{M}_2)$ does not contain cycles (where, for both $i \in \{1,2\}$. Let us denote by $\sigma(\mathcal{M}_i)$ the test defined from $\mathcal{M}_i$ by the switch obtained by restringing $\sigma \in \Sigma(\mathcal{M})$ to $\mathcal{M}_i$). Then, if $\sigma(\mathcal{M})$ contains a cycle, then there are $x, y \in X$ such that x and y are connected in $\sigma(\mathcal{M}_1)$ and in $\sigma(\mathcal{M}_2)$. This implies the existence of a $\gamma_1 \in p \in \mathfrak{B}_{\mathcal{M}_1|_X}$ and a $\gamma_2 \in q \in \mathfrak{B}_{\mathcal{M}_2|_X}$ with γ_1 and γ_2 both containing x and y, therefore $p \not\perp_w q$, contradicting the hypothesis.

The fact that each vertex in $\mathcal{M}$ is connected to a vertex of the border in any test is consequence of the fact that each vertex in $\mathcal{M}$ (then, in particular, each $x \in X$) is connected to a vertex in $\mathsf{B}_{\mathcal{M}} = \left(\mathsf{B}_{\mathcal{M}_1} \setminus X\right) \cup \left(\mathsf{B}_{\mathcal{M}_2} \setminus X\right)$. □

Corollary 25. *Let $\mathcal{M}$ be a component, and $\mathcal{T}$ be T-component and $(\mathsf{I}_{\mathcal{M}} \cap \mathsf{O}_{\mathcal{T}}) \supseteq X \neq \emptyset$. If $\mathcal{M}$ is a transitory, then $\mathcal{T} \triangleleft_X \mathcal{M}$ is so.*

Proof. In light of Theorem 24, it suffices to remark that if $\mathcal{M}$ is a T-component, then every input $\mathsf{i} \in \mathsf{I}_{\mathcal{M}}$ is connected to an output $\mathsf{o} \in \mathsf{O}_{\mathcal{M}}$ in a test $\sigma_2(\mathcal{M})$. If $\mathsf{o} \in \mathsf{O}_{\mathcal{M}}$ we conclude. Otherwise $\mathsf{o} \notin \mathsf{O}_{\mathcal{M}}$ and $\mathsf{o} = x \in X$. Since $\mathcal{T}$ is T-component, then by definition there is a test $\sigma_1(\mathcal{T})$ in which x is connected to a vertex $\mathsf{o} \in \mathsf{O}_{\mathcal{T}}$. We conclude since each input of $\mathcal{T} \triangleleft_X \mathcal{M}$ is either an input of $\mathcal{T}$, or an input of $\mathcal{M}$; and in the latter case we have a switching σ defined as the union of σ_1 and σ_2 such that i is connected to an output $\mathsf{o} \in \mathsf{O}_{\mathcal{T}} \subseteq \mathsf{O}_{\mathcal{T} \triangleleft_X \mathcal{M}}$. □

Proposition 26. *We can check if a component $\mathcal{M}'$ expands a component $\mathcal{M}$ in polynomial time with respect to $|\mathsf{O}_{\mathcal{M}'}| + |\mathfrak{B}_{\mathcal{M}'}| + |\mathfrak{B}_{\mathcal{M}}|$.*

Proof. To check if $\mathfrak{B}_{\mathcal{M}'} \perp_w \mathfrak{B}_{\mathcal{M}|_{\mathsf{O}_\ell}}$ requires $|\mathfrak{B}_{\mathcal{M}'}| \times |(\mathfrak{B}_{\mathcal{M}})_{|_{\mathsf{O}_{\mathcal{M}'}}}| \leq |\mathfrak{B}_{\mathcal{M}'}| \times |\mathfrak{B}_{\mathcal{M}}|$ orthogonality tests on partitions. Each test requires to build the graph $\mathcal{G}(p,q)$ (linear on $|\mathsf{O}_\ell|$) and check graph acyclicity since graph traversal is linear in $|V_{\mathcal{G}(p,q)}| + |\overset{\mathcal{G}(p,q)}{\frown}| \leq 2|\mathsf{O}_\ell|$ and $|\overset{\mathcal{G}(p,q)}{\frown}| = |\mathsf{O}_\ell|$, see [20]. □

6 Modelling behaviors beyond the scope of MLL-proof structures

In this section we recall the definition of two classes of generalized multiplicative connectives from [9], showing how the corresponding links can be used to define methods whose behaviors exhibit unexpected context-sensitive and non-linear characteristics in a multiplicative setting.

Definition 27. Let $n = uv$ be the product of two prime numbers $u, v \in \mathbb{N}$. We define a **basic partition with u blocks of v elements** to be a partition $p \in \mathbb{P}_n$ such that each block $\gamma \in p$ is either of the form $\gamma = (i, i+1, \ldots, i+v-1)$ if $i+v < n$, or of the form $\gamma = (i, \ldots, n, 1, 2, \ldots, i+v-(n+1))$ for a $i \in \{1, \ldots, n\}$ otherwise. We denote by $\mathfrak{B}_{\langle u,v \rangle} \subset \mathbb{P}_n$ the set of basic partitions with u blocks of v elements and $\mathfrak{B}^{\perp}_{\langle u,v \rangle}$ its orthogonal set of partitions.

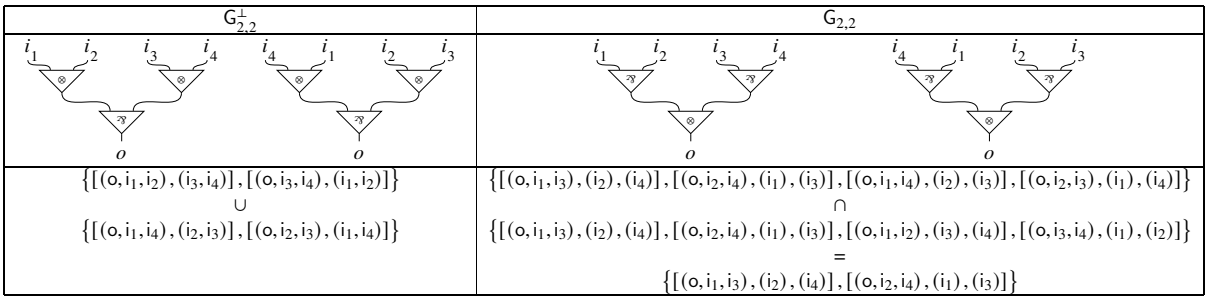


Figure 8: Connective $\mathsf{G}^{\perp}_{2,2}$ and its dual connective $\mathsf{G}_{2,2}$ interpreted as the union of the behaviors of DNF formula-trees and as the intersection of the behaviors of CNF formula-trees respectively.

For every $n = uv$ we define the following sets of partitions of the set $\{0,\dots,n\}$:

$$\mathfrak{G}_{\langle u,v\rangle}(0,1,\dots,n) = \bigcup_{k=1}^{u}\left\{p_k = \left[\gamma_{p_k},(i_1),\dots,(i_{n-u})\right] \mid p_{k|_{\{1,\dots,n\}}} \in \mathfrak{B}^{\perp}_{\langle u,v\rangle} \text{ and } 0 \in \gamma_{p_k}\right\}$$
$$\mathfrak{G}^{\perp}_{\langle u,v\rangle}(0,1,\dots,n) = \bigcup_{k=1}^{u}\bigcup_{j=1}^{v}\left\{p^j_k = [\gamma_1,\dots,\gamma_v] \mid p^j_{k|_{\{1,\dots,n\}}} \in \mathfrak{B}_{\langle u,v\rangle} \text{ and } 0 \in \gamma_j\right\}$$

and we define the following **Girard's types**: $\mathsf{G}_{u,v} \coloneqq \left\langle uv, 1, \mathfrak{G}_{\langle u,v\rangle}\right\rangle$ and $\mathsf{G}^{\perp}_{u,v} \coloneqq \left\langle uv, 1, \mathfrak{G}^{\perp}_{\langle u,v\rangle}\right\rangle$. [5]

Remark 28. The behavior $\mathfrak{G}_{\langle u,v\rangle}$ is the same of the intersection of the behavior of specific DNF-formulas, while $\mathfrak{G}^{\perp}_{\langle u,v\rangle}$ is the same of the union of behavior of specific CNF-formulas (see Figure 8). More precisely,

$$\mathfrak{G}_{\langle u,v\rangle} = \bigcap_{\tau\in \mathsf{Cy}_n} \mathfrak{B}_{\mathsf{DNF}(i_{\tau(1)},\dots,i_{\tau(n)})} \qquad \text{and} \qquad \mathfrak{G}^{\perp}_{\langle u,v\rangle} = \bigcup_{\tau\in \mathsf{Cy}_n} \mathfrak{B}_{\mathsf{CNF}(i_{\tau(1)},\dots,i_{\tau(n)})} \qquad \text{where}$$

- $\mathfrak{B}_{\mathsf{DNF}(1,\dots,n)}$ be the behavior of the multiplicative structure representing the formula tree of the disjunctive normal form formula $\mathsf{DNF}(1,\dots,n) = \left(a_1 \otimes \dots \otimes a_v\right) \parr \dots \parr \left(a_{n-v+1} \otimes \dots \otimes a_n\right)$;
- $\mathfrak{B}_{\mathsf{CNF}(1,\dots,n)}$ be the behavior of the multiplicative structure representing the formula tree of the conjunctive normal form formula $\mathsf{CNF}(a_1,\dots,a_n) = \left(a_1 \parr \dots \parr a_v\right) \otimes \dots \otimes \left(a_{n-v+1} \parr \dots \parr a_n\right)$;
- Cy_n be the set of cyclic permutations over the set $\{1,\dots,n\}$ (assuming the standard order on $\mathbb{N}$).

Theorem 29 ([9])**.** *There is no multiplicative structure $\mathcal{S}$ over the signature $\{\parr,\otimes\}$ such that $\mathfrak{B}_{\mathcal{S}} = \mathfrak{B}_{\mathsf{G}_{u,v}}$ or $\mathfrak{B}_{\mathcal{S}} = \mathfrak{B}_{\mathsf{G}^{\perp}_{u,v}}$ for any $u,v \in \mathbb{N}$ prime numbers.*

Definition 30. We generalize the components of bipoles (see Definition 15) as follows:

- A **generalized body** is a component made of a $\otimes_n$ collecting the outputs of a multiplicative structure representing a CNF-formula (i.e., bodies) or $\mathfrak{G}^{\perp}_{\langle u,v\rangle}$-links.
- A **generalized synchronizer** is a component made of a $\parr^{\bullet}_n$ collecting the outputs of a multiplicative structure representing a DNF-formula (i.e., synchronizers) or $\mathfrak{G}_{\langle u,2\rangle}$-links.

A **generalized bipole** is a component made of a generalized synchronizer $\mathcal{R}$ collecting the outputs of certain generalized bodies $\mathcal{C}_1,\dots,\mathcal{C}_n$ and headers $\mathcal{H}_1,\dots,\mathcal{H}_n$ whose structure is similar to the one in Equation (10), but where no output F occurs (thanks to the $\parr^{\bullet}_n$ in the generalized synchronizer).

[5] In [9] it has been shown that there is no module $\mathcal{M}$ containing only $\parr$- and $\otimes$-link such that $\mathfrak{B}_{\mathcal{M}} = \mathfrak{B}_{\mathsf{G}_{u,v}}$ or $\mathfrak{B}_{\mathcal{M}} = \mathfrak{B}_{\mathsf{G}^{\perp}_{u,v}}$.

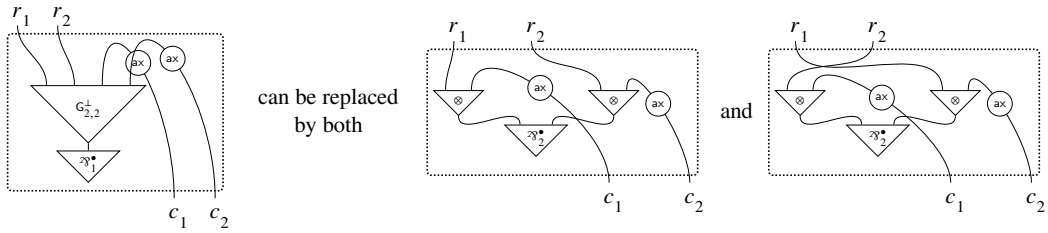


Figure 9: A multiplicative structure representing a non-deterministic application of two concurrent methods.

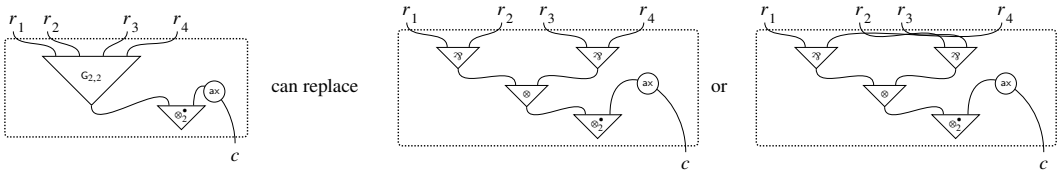


Figure 10: A link representing a method application with a dependent choice.

Remark 31. Headers and Generalized bodies and synchronizes are modules.

The following result is a corollary of Theorem 29.

Corollary 32. *There are generalized bipoles whose behavior is different from any behavior of a* MLL*-proof structure.*

Remark 33. In [15] the bipolar links have an additional output (the one labeled by the formula) as shown in Equation (9) because the syntax from [15] is closer to the representation of methods we use in Equation (8), where the name of the applied method (the formula F identifying the rule) occurs in the final sequent. In our syntax this superfluous information is discarded in the same way as in Equation (7), where the formula F does not occur in the conclusion. That is, the additional output F in Equation (10) would not occur in our generalized bipoles because in the definition of generalized synchronizes we use $⅋^{\bullet}_{n}$ instead of $⅋_{n}$, allowing us to formally discard this output.

We conclude this section by providing two toy-examples describing the way a server manages access requests to critical sections. The naïve idea behind these models is that if the vertex corresponding to a client is connected to the vertex corresponding to a resource, then there is a configuration of the model in which only that specific client accesses the resource.

Example 34 (basic union link)**.** Consider a server receiving a request from two different clients c_1 and c_2 to access, at the same time, to one resource r_1 or r_2 in a critical section. In this case the server can execute four different methods of the form $\mathbf{r_i} : - \mathbf{c_j}$ each of which represents the resource r_i being allocated to the client c_j (for some $i, j \in \{1, 2\}$). Once any one of these methods is executed, the condition of critical section requires that no other user can access to this resource (until it is released). Therefore, either the server authorizes c_1 to access r_1, and authorizes c_2 to access r_2, or the server authorizes c_1 to access r_2, and authorizes c_2 to access r_1.

In both cases, the two methods representing the clients accessing the resources can be applied concurrently, and the multiplicative structures on the right-hand side of Figure 9 represent these two configurations. Note that none of the two multiplicative structures fully capture the described configuration: each solution makes a choice about which client has access to which resource since, and this kind of choices are beyond the scope of multiplicative linear logic. Using the basic union link $\mathsf{G}^{\perp}_{2,2}$ we can define

the multiplicative structure in the left-hand side of Figure 9, whose behavior is the same of the union of the two multiplicative structures describing the two possible choices (see Remark 28). That is, this the $\mathsf{G}^{\perp}_{2,2}$-link can be interpreted as a synchronizer allowing such a concurrent choice.

Example 35 (basic intersection link)**.** Consider a server receiving a request from a single client c to access the set of resources $\{r_1, r_2, r_3, r_4\}$ with goal of either collect r_1 and r_3 (that is, to apply the method $\mathbf{c} :- \mathbf{r_1}, \mathbf{r_3}$), or to collect r_2 and r_4 (that is, to apply the method $\mathbf{c} :- \mathbf{r_2}, \mathbf{r_4}$). Because of our constraints, the server can either grant access to a resource in $\{r_1, r_2\}$ and one in $\{r_3, r_4\}$, or grant access to a resource in $\{r_1, r_4\}$ and one in $\{r_2, r_3\}$. These two different solutions are represented by the multiplicative structures in the right-hand side of Figure 10. However, if we consider singularly each multiplicative structure, it models more permissive configurations. For example, the right-most multiplicative structure admits a test in which c is connected to r_1 and r_4, which is not a configuration we want to allow. Indeed, the configurations we want to allow are exactly the configurations which are valid in both multiplicative structures. However, using the basic intersection link $\mathsf{G}_{2,2}$ we can define the multiplicative structure in the left of Figure 10, whose behavior is the same of the intersection of the two multiplicative structures on the right-hand side: the client c can only access at the same time to either r_1 and r_3, or r_2 and r_4.

7 Conclusion

In this paper, we extended the multiplicative fragment of the logic programming framework studied in [13, 14, 15, 29, 35]. Within this framework, as main novelty, we offered a computational interpretation of the generalized connectives discussed in [45, 9]. These connectives were initially introduced in early works on linear logic, but prior to this work, they had not been given a concrete computational interpretation: they cannot be expressed using combinations of the multiplicative connectives $\parr$ and $\otimes$ and they describe "locally additive" behaviors [42, 2] such as non-deterministic or conditional choices. It is worth noting that the methodology used to define our framework appears to align with the definition of the basic building block of the *transcendental syntax* [33, 27] and its extensions [26, 25, 28].

Future works. As observed in Remark 28, the behavior of a basic union link can be seen as the union of behaviors of bodies (see Figure 8). This allows us to replace any basic union link within the multiplicative structure of any of these bodies, and preserving correctness. Leveraging this intuition, we could define a non-deterministic *expansion* operation on basic union link, returning their set of bodies. This operation can be further extended to multiplicative structures, providing a notion of expansion for multiplicative structures similar to the concept of *Taylor expansion* in *differential linear logic* [24]. Such an expansion, could be interpreted as representing specific instances of *delayed choice* [18]: during proof (net) construction we do not need to specify which of the possible bodies we want to use in a specific step, but we can use a basic union link containing it instead, delaying this decision. We also envision an extension of our model where links have probabilistic distributions on the set of switchings. In this setting basic union links could be equipped with probability distribution functions, transforming the non-deterministic expansion operator into a probabilistic one. Consequently, multiplicative structures could be employed to model Bayesian networks [54, 46]. Additionally, we foresee the possibility of defining refinements of the *attack trees* syntax with a linear treatment of resources but including specific non-deterministic choices [49, 39, 53]. Similarly, the linear constraints on the hypergraphs used in our model allow us to define a concurrent computational model with a more granular management of resource consumption, akin to what we experience in the management of critical sections in concurrent systems. Another possible direction is to study modules for proof structures built using the graphical connectives from [7, 8, 6, 3] to provide them with a computational meaning based on resources separation.

References

[1] V. Michele Abrusci & Roberto Maieli (2019): *Proof nets for multiplicative cyclic linear logic and Lambek calculus*. *Mathematical Structures in Computer Science* 29(6), p. 733–762, doi:10.1017/S0960129518000300.

[2] Vito Michele Abrusci & Roberto Maieli (2016): *Cyclic Multiplicative-Additive Proof Nets of Linear Logic with an Application to Language Parsing*. In Annie Foret, Glyn Morrill, Reinhard Muskens, Rainer Osswald & Sylvain Pogodalla, editors: *Formal Grammar*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 43–59, doi:10.1007/978-3-662-53042-9_3.

[3] Matteo Acclavio (2024): *Graphical Proof Theory I: Sequent Systems on Undirected Graphs*. arXiv:2305.12975.

[4] Matteo Acclavio, Gianluca Curzi & Giulio Guerrieri (2023): *Infinitary cut-elimination via finite approximations (extended version)*. arXiv:2308.07789.

[5] Matteo Acclavio, Gianluca Curzi & Giulio Guerrieri (2024): *Infinitary Cut-Elimination via Finite Approximations*. In Aniello Murano & Alexandra Silva, editors: *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 288, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 8:1–8:19, doi:10.4230/LIPIcs.CSL.2024.8. Available at https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CSL.2024.8.

[6] Matteo Acclavio, Ross Horne, Sjouke Mauw & Lutz Straßburger (2022): *A Graphical Proof Theory of Logical Time*. In Amy P. Felty, editor: *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 228, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 22:1–22:25, doi:10.4230/LIPIcs.FSCD.2022.22. Available at https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.FSCD.2022.22.

[7] Matteo Acclavio, Ross Horne & Lutz Straßburger (2020): *Logic Beyond Formulas: A Proof System on Graphs*. In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, Association for Computing Machinery, New York, NY, USA, p. 38–52, doi:10.1145/3373718.3394763.

[8] Matteo Acclavio, Ross Horne & Lutz Straßburger (2022): *An Analytic Propositional Proof System on Graphs*. *Logical Methods in Computer Science* Volume 18, Issue 4, doi:10.46298/lmcs-18(4:1)2022. Available at https://lmcs.episciences.org/10186.

[9] Matteo Acclavio & Roberto Maieli (2020): *Generalized Connectives for Multiplicative Linear Logic*. In Maribel Fernández & Anca Muscholl, editors: *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 152, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 6:1–6:16, doi:10.4230/LIPIcs.CSL.2020.6. Available at https://drops.dagstuhl.de/opus/volltexte/2020/11649.

[10] Matteo Acclavio & Lutz Straßburger (2018): *From Syntactic Proofs to Combinatorial Proofs*. In Didier Galmiche, Stephan Schulz & Roberto Sebastiani, editors: *Automated Reasoning*, Springer International Publishing, Cham, pp. 481–497, doi:10.1007/978-3-319-94205-6_32.

[11] Matteo Acclavio & Lutz Straßburger (2022): *Combinatorial Proofs for Constructive Modal Logic*. In David Fernández-Duque, Alessandra Palmigiano & Sophie Pinchinat, editors: *Advances in Modal Logic, AiML 2022, Rennes, France, August 22-25, 2022*, College Publications, pp. 15–36. Available at http://www.aiml.net/volumes/volume14/06-Acclavio-Strassburger.pdf.

[12] Jean-Marc Andreoli (1992): *Logic programming with focusing proofs in linear logic*. *Journal of logic and computation* 2(3), pp. 297–347, doi:10.1093/logcom/2.3.297.

[13] Jean-Marc Andreoli (2001): *Focussing and proof construction*. *Annals of Pure and Applied Logic* 107(1), pp. 131 – 163, doi:10.1016/S0168-0072(00)00032-4.

[14] Jean Marc Andreoli (2002): *Focussing proof-net construction as a middleware paradigm*. In: *International Conference on Automated Deduction*, Springer, pp. 501–516, doi:10.1007/3-540-45620-1_39.

[15] Jean-Marc Andreoli & Laurent Mazaré (2003): *Concurrent Construction of Proof-Nets*. In Matthias Baaz & Johann A. Makowsky, editors: *Computer Science Logic*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 29–42, doi:10.1007/978-3-540-45220-1_3.

[16] Arnon Avron & Iddo Lev (2001): *Canonical Propositional Gentzen-Type Systems*. In: *in Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR 2001) (R. Goré, A Leitsch, T. Nipkow, Eds), LNAI 2083*, Springer Verlag, pp. 529–544, doi:10.1007/3-540-45744-5_45.

[17] David Baelde, Amina Doumane & Alexis Saurin (2016): *Infinitary proof theory : the multiplicative additive case* . Available at https://hal.science/hal-01339037. Working paper or preprint.

[18] J. C. M. Baeten & S. Mauw (1995): *Delayed choice: an operator for joining Message Sequence Charts*, pp. 340–354. Springer US, Boston, MA, doi:10.1007/978-0-387-34878-0_27.

[19] F. Bonchi, F. Gadducci, A. Kissinger, P. Sobocinski & F. Zanasi (2016): *Rewriting modulo symmetric monoidal structure*. *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–10, doi:10.1145/2933575.2935316.

[20] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein (2001): *Introduction to Algorithms*, 2nd edition. The MIT Press.

[21] Vincent Danos & Laurent Regnier (1989): *The structure of multiplicatives*. *Archive for Mathematical logic* 28(3), pp. 181–203, doi:10.1007/BF01622878.

[22] Anupam Das (2021): *On the Logical Strength of Confluence and Normalisation for Cyclic Proofs*. In Naoki Kobayashi, editor: *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, *LIPIcs* 195, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 29:1–29:23, doi:10.4230/LIPIcs.FSCD.2021.29.

[23] Anupam Das & Damien Pous (2018): *Non-Wellfounded Proof Theory For (Kleene+Action)(Algebras+Lattices)*. In Dan Ghica & Achim Jung, editors: *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 119, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 19:1–19:18, doi:10.4230/LIPIcs.CSL.2018.19. Available at http://drops.dagstuhl.de/opus/volltexte/2018/9686.

[24] T. Ehrhard & L. Regnier (2006): *Differential interaction nets*. *Theoretical Computer Science* 364(2), pp. 166–195, doi:10.1016/j.tcs.2006.08.003. Available at https://www.sciencedirect.com/science/article/pii/S0304397506005299. Logic, Language, Information and Computation.

[25] Boris Eng (2020): *Stellar Resolution: Multiplicatives - for the linear logician, through examples*. Available at https://hal.science/hal-02977750. Working paper or preprint.

[26] Boris Eng & Thomas Seiller (2020): *Stellar Resolution: Multiplicatives*. arXiv:2007.16077.

[27] Boris Eng & Thomas Seiller (2021): *A gentle introduction to Girard's Transcendental Syntax*. In: *5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)*, Rome (virtual), Italy. Available at https://hal-lirmm.ccsd.cnrs.fr/lirmm-03271496.

[28] Boris Eng & Thomas Seiller (2021): *Multiplicative Linear Logic from Logic Programs and Tilings*. Available at https://hal.science/hal-02895111. Working paper or preprint.

[29] Christophe Fouquere & Virgile Mogbil (2004): *Modules and Logic Programming*. arXiv:cs/0411029.

[30] Jean-Yves Girard (1987): *Linear logic*. *Theoretical Computer Science* 50(1), pp. 1–101, doi:10.1016/0304-3975(87)90045-4.

[31] Jean-Yves Girard (1987): *Multiplicatives*. In G. Lolli, editor: *Logic and Computer Science: New Trends and Applications*, Rosenberg & Sellier, pp. 11–34.

[32] Jean-Yves Girard (2000): *On the meaning of logical rules II: multiplicatives and additives*. *NATO ASI Series F Computer and Systems Sciences* 175, pp. 183–212.

[33] Jean-Yves Girard (2017): *Transcendental syntax I: deterministic case*. Math. Struct. Comput. Sci. 27(5), pp. 827–849, doi:10.1017/S0960129515000407.

[34] S. Guerrini (1999): *Correctness of multiplicative proof nets is linear*. In: Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158), pp. 454–463, doi:10.1109/LICS.1999.782640.

[35] Rémy Haemmerlé, François Fages & Sylvain Soliman (2007): *Closures and Modules Within Linear Logic Concurrent Constraint Programming*. In V. Arvind & Sanjiva Prasad, editors: FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 544–556, doi:10.1007/978-3-540-77050-3_45.

[36] Willem Heijltjes & Robin Houston (2014): *No Proof Nets for MLL with Units: Proof Equivalence in MLL is PSPACE-Complete*. In: Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Association for Computing Machinery, New York, NY, USA, doi:10.1145/2603088.2603126.

[37] Willem B. Heijltjes, Dominic J. D. Hughes & Lutz StraBburger (2019): *Intuitionistic proofs without syntax*. In: 2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), pp. 1–13, doi:10.1109/LICS.2019.8785827.

[38] David Hemer, Robert Colvin, Ian Hayes & Paul Strooper (2002): *Don't Care Non-determinism in Logic Program Refinement*. Electronic Notes in Theoretical Computer Science 61, pp. 101–121, doi:10.1016/S1571-0661(04)00308-1. Available at https://www.sciencedirect.com/science/article/pii/S1571066104003081. CATS'02, Computing: the Australasian Theory Symposium.

[39] Ross Horne, Sjouke Mauw & Alwen Tiu (2017): *Semantics for Specialising Attack Trees based on Linear Logic*. Fundamenta Informaticae 153, pp. 57–86, doi:10.3233/FI-2017-1531.

[40] Dominic Hughes & Willem Heijltjes (2016): *Conflict Nets: Efficient Locally Canonical MALL Proof Nets*. In: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, Association for Computing Machinery, New York, NY, USA, p. 437–446, doi:10.1145/2933575.2934559.

[41] Dominic J.D. Hughes (2006): *Towards Hilbert's 24th Problem: Combinatorial Proof Invariants: (Preliminary version)*. Electronic Notes in Theoretical Computer Science 165, pp. 37–63, doi:10.1016/j.entcs.2006.05.036. Available at https://www.sciencedirect.com/science/article/pii/S1571066106005135. Proceedings of the 13th Workshop on Logic, Language, Information and Computation (WoLLIC 2006).

[42] Olivier Laurent & Roberto Maieli (2008): *Cut Elimination for Monomial MALL Proof Nets*. In: 2008 23rd Annual IEEE Symposium on Logic in Computer Science, pp. 486–497, doi:10.1109/LICS.2008.31.

[43] Chuck Liang & Dale Miller (2021): *Focusing Gentzen's LK proof system*. Available at https://hal.science/hal-03457379. Working paper or preprint.

[44] Roberto Maieli (2014): *Construction of Retractile Proof Structures*. In Gilles Dowek, editor: Rewriting and Typed Lambda Calculi, Springer International Publishing, pp. 319–333, doi:10.1007/978-3-319-08918-8_22.

[45] Roberto Maieli (2019): *Non decomposable connectives of linear logic*. Annals of Pure and Applied Logic 170(11), p. 102709, doi:10.1016/j.apal.2019.05.006.

[46] Roberto Maieli (2021): *Probabilistic logic programming with multiplicative modules*. In Christian Retoré & E. Pimentel, editors: 5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021), Rome (virtual), Italy. Available at https://hal-lirmm.ccsd.cnrs.fr/lirmm-03271511.

[47] Roberto Maieli (2022): *A Proof of the Focusing Theorem via MALL Proof Nets*. In Agata Ciabattoni, Elaine Pimentel & Ruy J. G. B. de Queiroz, editors: Logic, Language, Information, and Computation - 28th International Workshop, WoLLIC 2022, Iaşi, Romania, September 20-23, 2022, Proceedings, Lecture Notes in Computer Science 13468, Springer, pp. 1–17, doi:10.1007/978-3-031-15298-6_1.

[48] Roberto Maieli & Quintijn Puite (2005): *Modularity of proof-nets*. Archive for Mathematical Logic 44(2), pp. 167–193, doi:10.1007/s00153-004-0242-2.

[49] Sjouke Mauw & Martijn Oostdijk (2006): *Foundations of Attack Trees*. 3935, pp. 186–198, doi:10.1007/11734727_17.

[50] Dale Miller (1995): *A survey of linear logic programming*. *Computational Logic: The Newsletter of the European Network of Excellence in Computational Logic* 2(2), pp. 63–67.

[51] Dale Miller (2004): *Overview of Linear Logic Programming*. In Thomas Ehrhard, editor: *Linear Logic in Computer Science*, Cambridge University Press, pp. 316–119, doi:10.1017/CBO9780511550850.004.

[52] Dale Miller & Elaine Pimentel (2013): *A formal framework for specifying sequent calculus proof systems*. *Theoretical Computer Science* 474, pp. 98–116, doi:10.1016/j.tcs.2012.12.008.

[53] Stefano M. Nicoletti, E. Moritz Hahn & Mariëlle Stoelinga (2022): *BFL: a Logic to Reason about Fault Trees*. In: *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 441–452, doi:10.1109/DSN53405.2022.00051.

[54] Judea Pearl (1988): *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[55] Christian Retoré (1993): *Réseaux et séquents ordonnés*. Theses, Université Paris-Diderot - Paris VII. Available at https://theses.hal.science/tel-00585634.

[56] Alexis Saurin (2023): *A Linear Perspective on Cut-Elimination for Non-wellfounded Sequent Calculi with Least and Greatest Fixed-Points*. In Revantha Ramanayake & Josef Urban, editors: *Automated Reasoning with Analytic Tableaux and Related Methods*, Springer Nature Switzerland, Cham, pp. 203–222, doi:10.1007/978-3-031-43513-3_12.

Developments in Sheaf-Theoretic Models of Natural Language Ambiguities

Kin Ian Lo Mehrnoosh Sadrzadeh
University College London
London, UK
{kin.lo.20,m.sadrzadeh}@ucl.ac.uk

Shane Mansfield
Quandela
Paris, France
shane.mansfield@quandela.com

Sheaves are mathematical objects consisting of a base that constitutes a topological space and the data associated with each open set thereof, e.g. continuous functions defined on the open sets. Sheaves have originally been used in algebraic topology and logic. Recently, they have also modelled events such as physical experiments and natural language disambiguation processes. We extend the latter models from lexical ambiguities to discourse ambiguities arising from anaphora. To begin, we calculated a new measure of contextuality for a dataset of basic anaphoric discourses, resulting in a higher proportion of contextual models–82.9%–compared to previous work which only yielded 3.17% contextual models. Then, we show how an extension of the natural language processing challenge, known as the Winograd Schema, which involves anaphoric ambiguities can be modelled on the Bell-CHSH scenario with a contextual fraction of 0.096.

1 Background

We introduce the basic elements of presheaf and sheaf theory, review sheaf theoretic models of quantum scenarios and of lexical ambiguities of natural language.

1.1 Sheaf Theory

The *presheaf* of events is a contravariant functor $\mathscr{E}$ from subsets of a set X to the category Set:

$$\mathscr{E}\colon \mathscr{P}(X)^{op} \to Set$$

Given a set O of outcomes, the functor acts as follows on objects, i.e. subsets U of X:

$$\mathscr{E}\colon U \mapsto O^U$$

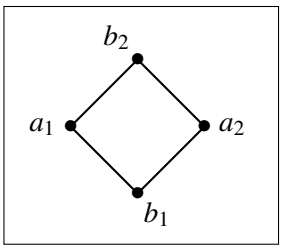
Given $U, U' \subseteq X$ and for $res_U^{U'}$ the restriction map, $\mathscr{E}$ acts as follows on morphisms:

$$U \subseteq U' \implies res_U^{U'} : \mathscr{E}(U') \to \mathscr{E}(U) :: s \mapsto s|U$$
$$U \subseteq U' \subseteq U'' \implies res_U^{U'} \circ res_U^{U'} = res_U^{U''}$$

That is, it assigns to each subset $U \subseteq X$ a map $s\colon U \to O$. This map describes the event in which an element $m \in U$ is performed with the outcome $s(m)$. It describes a section of the data that is being modelled and is thus referred to as a *section*. A presheaf is a *sheaf* if every compatible family of sections can be glued together to form a global section. This means that for some locally compatible sections $\{s_i \in \mathscr{E}(U_i)\}_{i\in I}$, there is a unique global section $s \in \mathscr{E}(U)$ such that $s|U_i = s_i$ for all $i \in I$. In other words,

S. Alves and I. Mackie (Eds.): 13th International Workshop
on Developments in Computational Models 2023 (DCM'23).
EPTCS 408, 2024, pp. 62–72, doi:10.4204/EPTCS.408.4

(a)



(b)

	$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
(a_1,b_1)	$1/2$	0	0	$1/2$
(a_1,b_2)	$3/8$	$1/8$	$1/8$	$3/8$
(a_2,b_1)	$3/8$	$1/8$	$1/8$	$3/8$
(a_2,b_2)	$1/8$	$3/8$	$3/8$	$1/8$

Figure 1: The geometric representation of the Bell-CHSH scenario and an empirical model for it.

local data must agree on overlaps. If this is the case, we can *glue* it together to create a (unique) *global* section $s \in \mathscr{E}(U)$ such that $s|U_i = s_i$, for all $i \in I$.

We can assign to each set of measurements U, the set of probability distributions over the sections of U by composing the sheaf functor $\mathscr{E}$ with a distribution functor $\mathscr{D}_R$. This provides us with the functor $\mathscr{D}_R(\mathscr{E}(U))$. The distribution functor is over a semiring R. It assigns to a set X of measurements, a set of functions $d\colon X \to R$ with finite support, such that $\sum_{x\in X} d(x) = 1$. The resulting composed functor has the type $\mathscr{D}_R\mathscr{E}\colon \mathscr{P}(X)^{op} \to Set$ and is a presheaf, as for $U \subseteq U'$, we can define a restriction map as follows:

$$\mathscr{D}_R\mathscr{E}(U') \to \mathscr{D}_R\mathscr{E}(U) :: d \mapsto d|U$$

This map sends a section $s \in \mathscr{E}(U)$ to its marginal $d|U(s)$, that is the sum of all probabilities in the larger sections $d(s')$ for $s' \in \mathscr{E}(U')$ whenever those larger sections s' restricted to s, i.e. $s' \mid U = s$.

1.2 Sheaf Models of Quantum Scenarios

An example of a sheaf of events with distributions is the entanglement protocols of quantum mechanics. Here, we have measurement scenarios, which are modelled by tuples of the form $\langle X, \mathscr{M}, O\rangle$, where $\mathscr{X}$ is the set of observables of the scenario, $\mathscr{M}$ is a cover of X, and O is the set of measurement outcomes. A subset of simultaneously measurable observables of X is called a measurement context. Thus, one can define a (local) joint probability distribution over the observables of a context. Such a joint probability distribution can either be estimated by performing the measurements in an experiment, or by calculating according to the theory of the system of interest. A collection of all joint probability distributions is called an *empirical model*.

Sheaf theory can be used to model and reason about a fundamental phenomenon of quantum mechanics known as *contextuality*. This phenomenon was observed as early as 1935 by Einstein, Podolsky and Rosen (EPR) [10]. It occurs when the quantum mechanical description of a scenario is incomplete. A typical scenario that exhibits contextuality has two space-like separated parties allowed to make measurements on an entangled system. In a sheaf theoretic setting, contextuality becomes the fact that there does not exist a global joint distribution over all the observables of the scenario that marginalises to every local joint distribution in the empirical model. This means failure of finding a classical explanation for the empirical model of the scenario. In conclusion, a scenario can host any contextual model if and only if its distribution presheaf is not a sheaf.

As an example, consider the Bell-CHSH scenario [4]. It is specified by $X = \{a_1, a_2, b_1, b_2\}$; $\mathscr{M} = \big\{\{a_1,b_1\},\{a_1,b_2\},\{a_2,b_1\},\{a_2,b_2\}\big\}$; $O = \{0,1\}$. $\mathscr{M}$ has a geometric realisation as the boundary of a square (a cycle of order 4), where each vertex is an observable, and each edge is a context (see Figure 1.2). The Bell state $|\Psi\rangle = \big(|00\rangle + |11\rangle\big)/\sqrt{2}$ produces the empirical model shown in Figure 1.2, which is contextual and moreover violates the Bell-CHSH inequality maximally [2, 4].

An empirical model is *signalling* if the marginalised distribution of a set of observables differs from one context to another. Intuitively speaking, non-signalling means that the observed probabilities are context-invariant and thus the choice of context cannot be used to transmit information over parties. A degree of contextuality can be defined for a model using its *contextual fraction* $\mathsf{CF}(e)$, introduced in [1]. Given an empirical model e, the contextual fraction $\mathsf{CF}(e)$ is the minimum λ such that e admits a convex decomposition:

$$e = (1-\lambda)e^{NC} + \lambda e^{C}, \tag{1}$$

where e^{NC} is a non-contextual empirical model and e^{C} is an empirical model that may be contextual. Suppose a given model e is non-contextual, then λ can be set to zero by choosing $e^{NC} = e$. Otherwise, λ must be greater than zero to make the decomposition valid. In a nutshell, for all contextual models, we have:

$$\mathsf{CF}(e) > 0 \tag{2}$$

The CF of a model has a nice interpretation as the maximum amount of *normalised violation* of all possible general Bell's inequalities [1]. For signalling models, the above decomposition would never hold as e^{NC} and e^{C} are non-signalling by definition. We could instead allow e^{C} to be signalling, but doing so would lead to the erroneous conclusion that all signalling models are contextual, assuming we still interpret CF as a measure of contextuality for signalling models.

Emeriau et al. [18] proposed a criterion for contextuality in the presence of signalling, making use of a measure called the *signalling fraction* $\mathsf{SF}(e)$, defined as the minimum λ such that e admits a convex decomposition:

$$e = (1-\lambda)e^{NS} + \lambda e^{S}, \tag{3}$$

where e^{NS} is a non-signalling empirical model and e^{S} is an empirical model that may be signalling. The signalling fraction of a model is zero if and only if the model is non-signalling. The contextuality criterion of Emeriau requires the following inequality to hold:

$$\mathsf{CF}(e) > 2|\mathscr{M}|\mathsf{SF}(e). \tag{4}$$

The intuition is that the degree of signalling (SF) can be regarded as the magnitude of perturbation on the empirical model. The perturbation in turn affects the contextual fraction (CF) of the model. Emeriau et al. proved a continuity result that bounds the change to the contextual fraction given the magnitude of a perturbation of the empirical model. The factor of $|\mathscr{M}|$ comes from the fact that every context in the scenario could be perturbed independently, and thus their effects could accumulate. The factor of 2 comes from the use of *total variation* as a distance measure between empirical models. The total variation distance between two probability distributions is defined as half the sum of the absolute differences between their probabilities.

1.3 Sheaf Models of Lexical Ambiguities

Lexical ambiguity, where a word has multiple meanings, is one of the most common types of ambiguity in natural language. One way to formalise lexical ambiguity in natural language is to consider an ambiguous word as an observable. The possible interpretations of the ambiguous word are treated as the possible outcomes of the observable. As an example, consider the ambiguous word *pitcher*, which means *a certain type of jug* or *or a type of baseball player*. These interpretations are modelled as outcomes. Reading the word *pitcher* in a piece of text thus becomes the event of performing a measurement. The word itself becomes an observable. Probabilities are assigned to outcomes by asking a group of readers

to read a word and rate the likelihood of its interpretations over one another. This way of treating ambiguous words is inspired by the Bell-CHSH scenario and was first considered in [19, 20]. The authors considered subject-verb and verb-object phrases where each word in the phrase had at least two possible interpretations. The measurement contexts were constructed by selecting different pairs of nouns and verbs in a way similar to how Alice and Bob select their measurements in the Bell-CHSH scenario. The phrases were uploaded into the crowd-sourcing Amazon Mechanical Turk and probability judgments were collected from human subjects. The resulting empirical models were all signalling and the authors had to analyze the data in the framework of Contextuality-by-Default [7, 8, 9]. This setting is able to compute a degree of contextuality in the presence of signalling. According to this measure, 4.5% of the phrases (13 out of 290) in [20] were contextual.

2 Methodology

In this section, we introduce the basic anaphoric ambiguity schema and the Winograd Schema. We then describe how we modelled them using the sheaf framework and the Contextuality-by-Default framework.

2.1 Sheaf Models of Basic Anaphoric Ambiguities

Another type of ambiguity in natural language is *anaphoric ambiguity*. Here, a pronoun can refer to different expressions that come before it in a piece of text. For instance, the pronoun *it* (aka *anaphora*) can refer to the *dog* or the *cat* (aka *anaphors*) in the text *The dog chased the cat. It barked.*. A piece of text that contains an anaphora together with its possible anaphors is often called a *discourse*.

In a previous work [15], we discovered that a simpler scenario than Bell-CHSH can be used to model basic anaphoric ambiguities. This scenario has three (rather than four) observables $\mathscr{X} = \{x_1, x_2, x_3\}$, which give rise to measurement contexts $\mathscr{M} = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_1\}\}$. We proved that this scenario is the only strongly contextual scenario up to relabelling and called the models of the scenario *PR prism*, as an analogy to the PR boxes. We then built schemas demonstrating anaphoric ambiguities, where outcomes O_1 and O_2 were the choice of referents are noun phrases and the observables X_1, X_2, X_3 are any of their *basic modifiers*, i.e. adjective, verb, and prepositional phrases. We refer to the ambiguities resulting from these constructions as *basic anaphoric ambiguities*.

We then built schemas in English demonstrating the basic anaphoric ambiguities. The schemas were instantiated using the large pre-trained transformer-based language model BERT [5]. The masked word prediction capability of BERT provided us with instances of the modifiers and their probability distributions.

In order to build discourses with basic anaphoric ambiguities, we used the template shown in Figure 2 which involves two nouns (O_1 and O_2), and three modifiers (X_1, X_2, and X_3) which can be instantiated by any of the following types of modifier:

1. adjectives, e.g. *red*, *round*, *sweet*; (an example of which is shown in Figure 2)
2. participial adjectives (verb-derived adjectives), e.g. *running*, *broken*, *frozen*;
3. prepositional phrases, e.g. *in the box*, *on the table*, *under the bed*.

In [15], we constructed 11,052 examples of the schema with adjective modifiers. As the examples were in general signalling, we used the criterion of [18], which makes use of the signalling fraction to determine if the examples were contextual. We found that only 350 examples (3.17%) were contextual. Since the criterion of [18] is a lower bound, the actual percentage of contextual examples could be higher.

There is an O_1 and an O_2.

1. It is X_1 and the same one is X_2.
2. It is X_2 and the same one is X_3.
3. It is X_3 and the other one is X_1.

There is an *apple* and a *strawberry*.

1. It is *red* and the same one is *round*.
2. It is *round* and the same one is *sweet*.
3. It is *sweet* and the other one is *red*.

```
There is an apple and a strawberry.  The [MASK] is red and the same one is round.
There is an apple and a strawberry.  The [MASK] is round and the same one is sweet.
There is an apple and a strawberry.  The [MASK] is sweet and the other one is red.
```

Figure 2: The left box shows the general PR prism schema of basic anaphoric ambiguities. The two outcomes O_1 and O_2 are two nouns which are the referents of the anaphor. The three observables X_1, X_2, and X_3 are the modifiers of the nouns. The right box shows an instance of the schema with adjective modifiers.

2.2 The Contextuality-by-Default (CbD) framework

In the Contextuality-by-Default framework, one has two important notions: *contents* q_i, which are questions about the system; and *contexts* c^j, which represent the conditions under which the questions are asked. A question q_i asked in a context c^j gives rise to a random variable R_i^j taking values in $\{\pm 1\}$, and representing possible answers and their probabilities. All random variables in a given context are jointly distributed.

The simplest types of CbD systems are n-cyclic. Here, each context has exactly 2 contents, and every content is exactly in 2 contexts. It has been proven in [12] that these systems are contextual if and only if:

$$\mathsf{CNT}_1 = \mathsf{CNT}_2 = s_{odd}\left(\left\{\left\langle R_{i_j}^j R_{i'_j}^j \right\rangle\right\}_{j=1,\ldots,n}\right) - \Delta - n + 2 > 0 \tag{5}$$

where $j_i \neq j'_i$ for all i and $R_{i_j}^j, R_{i'_j}^j$ are well-defined for all j. Quantities $s_{odd} : \mathbb{R}^n \to \mathbb{R}$ and Δ are defined as follows:

$$s_{odd}\left(\underline{x}\right) = \max_{\substack{\underline{\sigma}\in\{\pm 1\}^k; \\ \mathfrak{p}(\underline{\sigma}=-1)}} \underline{\sigma}\cdot\underline{x}\,; \qquad \Delta = \sum_{i=1}^{n}\left|\left\langle R_i^{j_i}\right\rangle - \left\langle R_i^{j'_i}\right\rangle\right| \tag{6}$$

where $\mathfrak{p}(\underline{\sigma}) = \prod_{i=1}^n \sigma_i$ ($\mathfrak{p}$ is the parity function of $\underline{\sigma}$). The quantity Δ is called *Direct Influence* and it measures the degree of signalling in the system (in a non-signalling system $\Delta = 0$). In the Bell-CHSH scenario, $n = 4$ and S_{odd} becomes the maximum violation of the inequality.

Following Wang et al. [19, 20], instead of using the signalling fraction of sheaf theoretic models for our basic anaphoric ambiguities, we use the CbD framework. We modelled our basic anaphoric ambiguity dataset, described in Section 3.1, by taking q_i to be the anaphors, i.e. the adjective, verb, and prepositional phrase modifiers. We took c^j to be the different sentences in which the anaphors appear.

2.3 Sheaf Theoretic Models of the Original Winograd Schema

The Winograd Schema Challenge (WSC) consists of 285 descriptions with anaphoric ambiguities in them, followed by questions about the anaphors. The questions cannot be answered by syntactic reasoning only and extra information about the semantic context of the descriptions are needed. WSC was proposed as a test for human intelligence and an alternative to Turing's test in 2012 [13]. An example

description is *The trophy doesn't fit into the suitcase because it's too [small / large].* The questions following this description are *What is too [small / large]?* The correct answer for *small* is *suitcase*, and for *large* is *trophy* . In 2014, a cash prize was put forwards for an AI that could solve the WSC with an accuracy close to humans. The prize was withdrawn in 2018 after failures of machine learning algorithms of the time. In 2019, however, the deep learning transformer algorithm RoBERTa [14] fine-tuned on large amounts of data, got close to solving it. Nevertheless, WSC still poses a challenge for AI's that do not have access to the large resources of these algorithms [11]. In this subsection, we describe and model the challenge. A valid Winograd schema must satisfy the following requirements:

1. It consists of a pair of sentences. The first sentence contains a *special* word. The second sentence is obtained by replacing the *special* word with an *alternate* word. In the *trophy-suitcase* example, the *special* word is *small* and the *alternate* word is *large*.
2. There should be two noun phrases in the sentences. In the *trophy-suitcase* example, the two noun phrases are *the trophy* and *the suitcase*.
3. A pronoun must appear in the sentences. The pronoun must agree with the two noun phrases in terms of number and gender. In the *trophy-suitcase* example, the pronoun is *it* which agrees with both *the trophy* and *the suitcase* in terms of number and gender.
4. The referent of the pronoun should be easily identifiable from the sentence. The correct referent should be different in the two sentences.
5. Both sentences in the pair must be read in a natural way, i.e. similar to what appears in common sources of text such as news articles and Wikipedia.

Previous work modelled an ambiguous word as an observable, but they had two or more of them. In our case, we have ambiguous pronouns, but only one. In order not to end up with a scenario with only one observable, we define an observable to be a pair: (**pronoun**, *special word*) or (**pronoun**, *alternate word*). The possible outcomes of each of these observables are the candidate referents of the pronoun.

Definition 1 (Winograd Schema scenario) *Given a Winograd Schema with two noun phrases A and B; an ambiguous pronoun* $\boldsymbol{p}$ *which refers to either A or B; a special word (s) and an alternate word (a), the corresponding measurement scenario is defined by the data:*

- *observables* $X = \{(\boldsymbol{p}, s), (\boldsymbol{p}, a)\}$*;*
- *contexts* $\mathscr{M} = \big\{\{(\boldsymbol{p}, s)\}, \{(\boldsymbol{p}, a)\}\big\}$*;*
- *outcomes* $O = \{A, B\}$.

Such a measurement scenario is called a Winograd Schema scenario*, or a WS scenario in short.*

With the *trophy-suitcase* example, the measurement scenario would be given by the data:

- observables $X = \{(\textbf{it},\ \textit{large}),\ (\textbf{it},\ \textit{small})\}$;
- contexts $\mathscr{M} = \big\{\{(\textbf{it},\ \textit{large})\},\ \{(\textbf{it},\ \textit{small})\}\big\}$;
- outcomes $O = \{\text{trophy},\ \text{suitcase}\}$.

One can see that there is no overlap between the contexts and thus the empirical model corresponding to the schema is deterministic. It is shown in [6] that deterministic systems are not contextual and thus WS scenarios are not either.

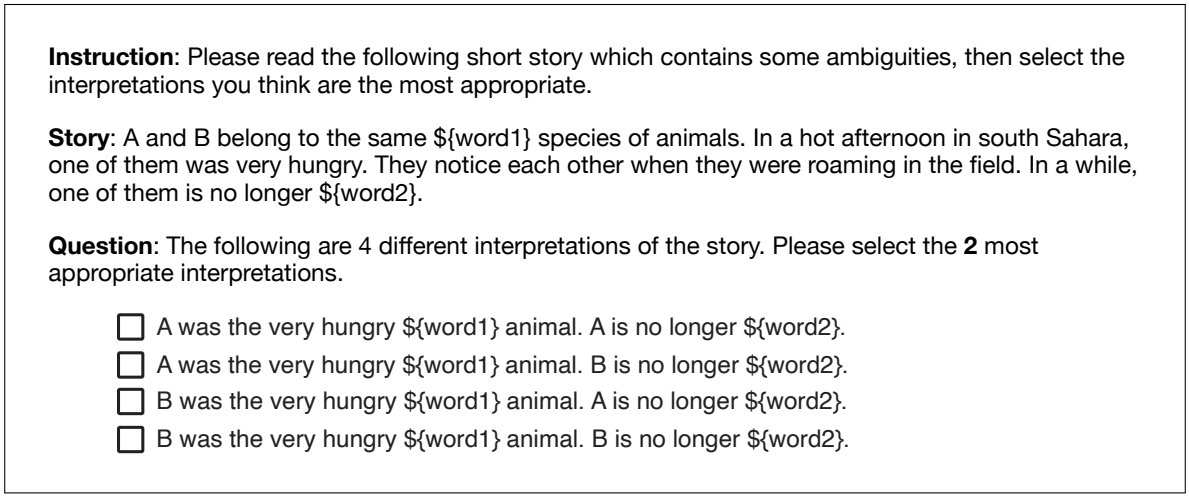


Figure 3: A screenshot of the template of the questionnaire used to collect human judgments on the generalised Winograd Schema.

2.4 Extending Winograd to Generalised Winograd

We generalised the Winograd Schema scenario such that contextuality can be exhibited in a recent work [16]. The original WS is analogous to an imaginary physical experiment with only one experimenter, who decides whether to measure the pronoun with the special word, or with the alternate word, by choosing between the two observables: $(\mathbf{p}, s)$ and $(\mathbf{p}, a)$. A natural way of generalising it would be to introduce one more experimenter, resulting in the Bell-CHSH scenario. This means that we need to add one more pronoun, one more special word and its alternate word. We use the subscript 1 to denote components relating to the first pronoun and the subscript 2 to those relating to the second pronoun. We obtain a new set of requirements for the resulting schema, which we call *generalised Winograd Schema*:

1. A generalised Winograd schema consists of four sentences. The first sentence contains two special words s_1 and s_2. Similar to the original Winograd Schema, s_1 can be replaced by an alternate word a_1, and s_2 can be replaced by a_2. Replacing special words with alternate words creates the rest of the four sentences.
2. There are two pairs of noun phrases. The first pair can be identical to the second pair.
3. There are two pronouns in the sentences. The first one refers to one of the noun phrases in the first pair of noun phrases. The second pronoun refers to either one of the noun phrases in the second pair of noun phrases.
4. All four sentences should be natural to read.

Definition 2 (Generalised Winograd Schema scenario) *Given a Generalised Winograd Schema with two noun phrases A and B; two ambiguous pronouns $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ can each refers to either A or B; two special words (s_1) and (s_2); two alternate words (a_1) and (a_2), the corresponding measurement scenario is defined by the data:*

- *observables* $X = \{(\boldsymbol{p}_1, s_1), (\boldsymbol{p}_1, a_1), (\boldsymbol{p}_2, s_2), (\boldsymbol{p}_2, a_2)\}$
- *contexts* $\mathscr{M} = \big\{\{(\boldsymbol{p}_1, s_1), (\boldsymbol{p}_2, s_2)\}, \{(\boldsymbol{p}_1, s_1), (\boldsymbol{p}_2, a_2)\}, \{(\boldsymbol{p}_1, a_1), (\boldsymbol{p}_2, s_2)\}, \{(\boldsymbol{p}_1, a_1), (\boldsymbol{p}_2, a_2)\}\big\}$;
- *outcomes* $O = \{A, B\}$.

Such a measurement scenario is called a Generalised Winograd Schema scenario*, or a generalised WS scenario in short.*

The generalised WS scenario is isomorphic, i.e. identical upon relabelling, to the Bell-CHSH scenario and should thus be able to host contextuality.

3 Results

3.1 Basic Anaphoric Ambiguities revisited

As discussed in Section 2.1, the criterion of [18] is sufficient but not necessary. To address this issue, we reanalysed the basic anaphoric ambiguities dataset using the CbD framework [7, 8, 9], which provides a tight criterion for contextuality. Indeed our analysis revealed that 9,159 examples (82.87%) were contextual in the CbD framework. Distributions of the signalling fractions and the Direct Influence of the CbD framework are shown in Figure 4.

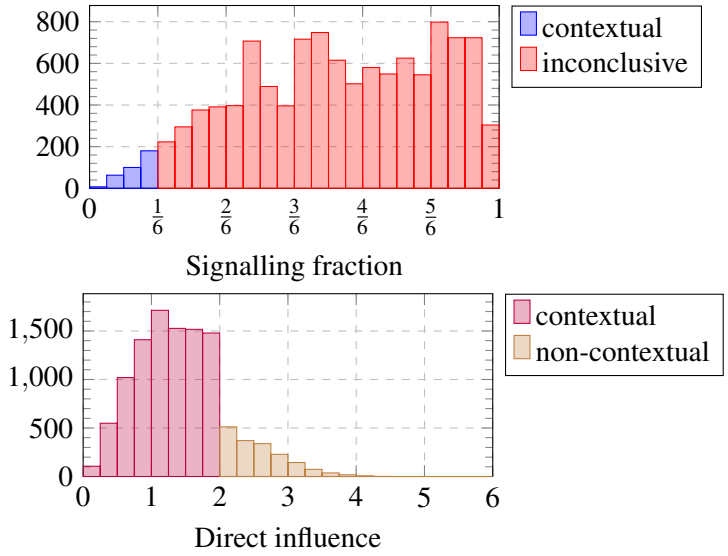


Figure 4: Distributions of 11,052 examples of basic anaphoric ambiguities. The top histogram shows the distribution of the signalling fractions. We observed that 350 examples (3.17%) have a signalling fraction less than $1/6$, which is the threshold for conclusive contextuality according to the criterion of [18]. The bottom histogram shows the distribution of the Direct Influence of the CbD framework. We observed that 9159 examples (82.87%) have a Direct Influence of less than 2, which is the threshold for contextuality in the CbD framework.

3.2 Human Judgments on the Generalised Winograd Schema

We constructed an example of the generalised Winograd Schema and collected human judgments on this example on the crowd-sourcing platform Amazon Mechanical Turk in the form of a questionnaire. The example is:

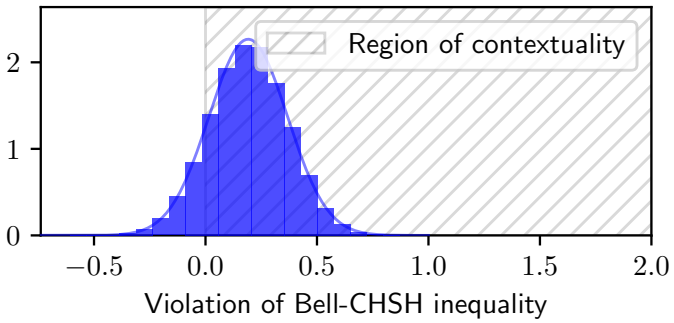


Figure 5: A histogram of violation of Bell-CHSH inequality for 100,000 bootstrap samples.

> *A and B belong to the same [cannibalistic / herbivorous]*$_1$ *species of animal. On a hot afternoon south of Sahara,* ***one of them***$_1$ *was very hungry. They noticed each other when they were roaming in the field. In a while,* ***one of them***$_2$ *is no longer [hungry / alive]*$_2$*.*

There were four versions of the questionnaire, each corresponding to one of the four contexts in the generalised WS scenario. The respondents were asked to read the example and answer a question about the correct referents, A or B, of the two referring phrases **one of them**$_1$ and **one of them**$_2$. A screenshot of the questionnaire is shown in Figure 3. The placement holders `${word1}` and `${word2}` are instantiated with the two special words or the alternate words of the generalised Winograd Schema. In this example, `${word1}` can be either *cannibalistic* or *herbivorous* and `${word2}` can be either *hungry* or *alive*. Four versions of the questionnaire were created, each corresponding to one of the four contexts in the generalised WS scenario.

Since each referring phrase can be interpreted in two ways, there are 4 possible combinations of interpretations, (A, A), (A, B), (B, A), (B, B), of the two referring phrases. The symmetry between A and B in the example ensures that the combinations (A, A) and (B, B) are equally plausible and (A, B) and (B, A) are also equally plausible. Therefore we asked the respondents to pick two out of the four combinations. This design choice also allows the detection of invalid answers, i.e. answers that are not symmetric with respect to A and B.

A total of 410 responses were collected on 20 Oct and 23 Nov 2022 from Amazon Mechanical Turk; 110 were for the context (*cannibalistic*, *hungry*) and 100 each for the rest of the three contexts. From these, 348 were valid. The respondents were each rewarded USD 1.00, regardless of the validity of their responses. The valid data was used to build an estimated probability distribution for each of the four contexts. The resulting empirical model is shown in Table 1. The model violates the Bell-CHSH inequality by 0.192 with a standard deviation of 0.176 and is thus contextual. We conducted bootstrap resampling to establish the statistical significance of this result. The distribution of the violation of the resampled models is shown in Figure 5. We see that of 87% of the models have a positive violation with a standard deviation of 0.176. Our experimental model is thus contextual with a significance level of 87%.

a)		(A, A)	(A, B)	(B, A)	(B, B)	b)	(A, A)	(A, B)	(B, A)	(B, B)
(*canni*,	*hungry*)	0.402	0.097	0.097	0.402	...	$1/2$	0	0	$1/2$
(*canni*,	*alive*)	0.044	0.455	0.455	0.044	...	0	$1/2$	$1/2$	0
(*herbi*,	*hungry*)	0.345	0.154	0.154	0.345	...	$1/2$	0	0	$1/2$
(*herbi*,	*alive*)	0.344	0.155	0.155	0.344	...	$1/2$	0	0	$1/2$

Table 1: (a) Empirical model constructed with human judgments from Amazon Mechanical Turk. The violation of Bell's inequality of the model is 0.192 ± 0.176. For brevity, the special word *cannibalistic* is shortened to *canni* and the alternate word *herbivorous* is shortened to *herbi*. (b) Empirical model of the PR box.

4 Discussion and Future Work

We assumed that the alphabetic symbols A, B used to model the generalised winograd scenarios are linguistic variables and thus interchangeable. This had the advantage that the model became non-signalling and thus the contextual fraction remains a valid measure of contextuality. Symmetrising psychological experiments has its criticisms, see [3]. We are, however, unaware of the existence of similar criticisms to a linguistic setting. The symmetry in the outcomes allows the violation to saturate the bound defined by CF [1] and the following equality is attained

$$\max\left\{0, \frac{1}{2}\,\text{violation of Bell-CHSH inequality}\right\} = \mathsf{CF}. \tag{7}$$

The CbD contextuality measures CNT_1 and CNT_2 coincide with the above degree of violation [12]. Thus, our model is considered contextual in both the sheaf-theoretic framework and the CbD framework.

The approach presented in this paper consists of deliberately constructing sentences that exhibit contextuality. One may criticise this as producing unnatural text. In future work, we will find naturally occurring language data that exhibits contextuality with the help of state-of-the-art generative large language models such as GPT-4 [17].

Acknowledgements

We are grateful to Daphne Wang for insightful discussions and the anonymous reviewers for their constructive comments. KL is supported by the Engineering and Physical Sciences Research Council [grant number EP/S021582/1]. MS is supported by the Royal Academy of Engineering research chair RCSRF2122-14-152 on Engineered Mathematics for Modelling Typed Structures.

References

[1] Samson Abramsky, Rui Soares Barbosa & Shane Mansfield (2017): *Contextual Fraction as a Measure of Contextuality*. *Physical Review Letter* 119, p. 050504, doi:10.1103/PhysRevLett.119.050504.

[2] J. S. Bell (1964): *On the Einstein Podolsky Rosen paradox*. *Physics Physique Fizika* 1(3), pp. 195–200, doi:10.1103/PhysicsPhysiqueFizika.1.195.

[3] Víctor H. Cervantes & Ehtibar N. Dzhafarov (2018): *Snow queen is evil and beautiful: Experimental evidence for probabilistic contextuality in human choices*. *Decision* 5(3), pp. 193–204, doi:10.1037/dec0000095.

[4] John F. Clauser, Michael A. Horne, Abner Shimony & Richard A. Holt (1969): *Proposed Experiment to Test Local Hidden-Variable Theories*. *Physical Review Letters* 23(15), pp. 880–884, doi:10.1103/PhysRevLett.23.880.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee & Kristina Toutanova (2019): *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota, pp. 4171–4186, doi:10.18653/v1/N19-1423.

[6] Ehtibar N. Dzhafarov (2019): *The Contextuality-by-Default View of the Sheaf-Theoretic Approach to Contextuality*. doi:10.48550/arXiv.1906.02718.

[7] Ehtibar N. Dzhafarov & Janne V. Kujala (2013): *All-Possible-Couplings Approach to Measuring Probabilistic Context*. *PLoS ONE* 8(5), p. e61712, doi:10.1371/journal.pone.0061712.

[8] Ehtibar N. Dzhafarov & Janne V. Kujala (2017): *Contextuality-by-Default 2.0: Systems with Binary Random Variables*. In Jose Acacio de Barros, Bob Coecke & Emmanuel Pothos, editors: *Quantum Interaction*, Springer International Publishing, pp. 16–32, doi:10.1007/978-3-319-52289-0_2.

[9] Ehtibar N. Dzhafarov, Janne V. Kujala & Victor H. Cervantes (2015): *Contextuality-by-Default: A Brief Overview of Ideas, Concepts, and Terminology*. *Lecture Notes in Computer Science 9535, 12-23, 2016*, doi:10.1007/978-3-319-28675-4-2.

[10] Albert Einstein, Boris Podolsky & Nathan Rosen (1935): *Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?* *Phys. Rev.* 47, pp. 777–780, doi:10.1103/PhysRev.47.777.

[11] Vid Kocijan, Ernest Davis, Thomas Lukasiewicz, Gary Marcus & Leora Morgenstern (2023): *The defeat of the Winograd Schema Challenge*. *Artificial Intelligence* 325, p. 103971, doi:10.1016/j.artint.2023.103971.

[12] Janne V. Kujala & Ehtibar N. Dzhafarov (2019): *Measures of Contextuality and Noncontextuality*. *Philosophical Transactions of the Royal Society A 377:20190149, 2019* 377(2157), p. 20190149, doi:10.1098/rsta.2019.0149.

[13] Hector J. Levesque, Ernest Davis & Leora Morgenstern (2012): *The Winograd Schema Challenge*. In: *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'12, AAAI Press, pp. 552–561, doi:10.5555/3031843.3031909.

[14] Zhuang Liu, Wayne Lin, Ya Shi & Jun Zhao (2021): *A Robustly Optimized BERT Pre-training Approach with Post-training*. In Sheng Li, Maosong Sun, Yang Liu, Hua Wu, Liu Kang, Wanxiang Che, Shizhu He & Gaoqi Rao, editors: *Chinese Computational Linguistics*, Springer International Publishing, Cham, pp. 471–484, doi:10.1007/978-3-030-84186-7_31.

[15] Kin Ian Lo, Mehrnoosh Sadrzadeh & Shane Mansfield (2022): *A Model of Anaphoric Ambiguities using Sheaf Theoretic Quantum-like Contextuality and BERT*. *EPTCS* 366, pp. 23–34, doi:10.4204/EPTCS.366.5.

[16] Kin Ian Lo, Mehrnoosh Sadrzadeh & Shane Mansfield (2023): *Generalised Winograd Schema and its Contextuality*. *Electronic Proceedings in Theoretical Computer Science* 384, pp. 187–202, doi:10.4204/EPTCS.384.11.

[17] OpenAI (2024): *GPT-4 Technical Report*. doi:10.48550/arXiv.2303.08774.

[18] Kim Vallée, Pierre-Emmanuel Emeriau, Boris Bourdoncle, Adel Sohbi, Shane Mansfield & Damian Markham (2024): *Corrected Bell and Non-Contextuality Inequalities for Realistic Experiments*. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 382(2268), p. 20230011, doi:10.1098/rsta.2023.0011.

[19] Daphne Wang, Mehrnoosh Sadrzadeh, Samson Abramsky & Victor Cervantes (2021): *On the Quantum-like Contextuality of Ambiguous Phrases*. In: *Proceedings of the 2021 Workshop on Semantic Spaces at the Intersection of NLP, Physics, and Cognitive Science (SemSpace)*, Association for Computational Linguistics, Groningen, The Netherlands, pp. 42–52, doi:10.48550/arXiv.2107.14589.

[20] Daphne Wang, Mehrnoosh Sadrzadeh, Samson Abramsky & Víctor H. Cervantes (2021): *Analysing Ambiguous Nouns and Verbs with Quantum Contextuality Tools*. *Journal of Cognitive Science* 22(3), pp. 391–420, doi:10.17791/jcs.2021.22.3.391.

When Do You Start Counting?
Revisiting Counting and Pnueli Modalities in Timed Logics

Hsi-Ming Ho
Department of Informatics, University of Sussex
United Kingdom
hsi-ming.ho@sussex.ac.uk

Khushraj Madnani
Max Planck Institute for Software Systems
Germany
kmadnani@mpi-sws.org

Pnueli first noticed that certain simple 'counting' properties appear to be inexpressible in popular timed temporal logics such as Metric Interval Temporal Logic (MITL). This interesting observation has since been studied extensively, culminating in strong timed logics that are capable of expressing such properties yet remain decidable. A slightly more general case, namely where one asserts the existence of a sequence of events in an arbitrary interval of the form $\langle a, b\rangle$ (instead of an upper-bound interval of the form $[0, b\rangle$, which starts from the current point in time), has however not been addressed satisfactorily in the existing literature. We show that counting in $[0, b\rangle$ is in fact as powerful as counting in $\langle a, b\rangle$; moreover, the general property 'there exist $x', x'' \in I$ such that $x' \leq x''$ and $\psi(x', x'')$ holds' can be expressed in Extended Metric Interval Temporal Logic (EMITL) with only $[0, b\rangle$.

1 Introduction

Timed logics. Temporal logics provide constructs to specify *qualitative* ordering between events in time. Timed logics extend classical temporal logics with the ability to specify *quantitative* timing constraints between events. *Metric Interval Temporal Logic* (MITL) [2] is amongst the best studied of timed logics. It extends the 'until' (**U**) and 'since' (**S**) modalities of *Linear Temporal Logic* (LTL) [37] with *non-singular* intervals to specify timing constraints. For example, $P\,\mathbf{U}_I\,Q$ states that an event where Q holds should occur in the future within a time interval I, and P should hold continuously till then.

Specifying multiple events. In many practical scenarios, e.g, those involving resource-bounded computations, the ability to specify not just one but a sequence of events within a given time interval can be crucial. For example, in a multi-threaded environment, a desired property for scheduling algorithms could be to have at most k context switches in every M time units. Such properties, however, cannot be expressed in MITL [9, 18, 27]. In particular, the *counting* (**C** and $\overleftarrow{\mathbf{C}}$) and *Pnueli* (**P** and $\overleftarrow{\mathbf{P}}$) modalities that specify event occurrences within the *next* or *previous* unit interval (i.e. within $[t_0, t_0+1)$ or $(t_0-1, t_0]$, where the current time is t_0) are studied in [18], and it turned out that for MITL extended with these modalities (called TLC and TLP, respectively), the satisfiability problem remains EXPSPACE-complete.[1] Moreover, it turned out that TLC and TLP, while the latter is syntactically more general, are equally expressive in the continuous semantics. This is shown by proving that both TLC and TLP are expressively complete for a natural fragment of *Monadic First-Order Logic of Order and Metric* (FO[<, +1]) called Q2MLO, where one can specify that the sequence of events between the current time t_0 and $t \in t_0 + I$ (for a *non-singular* interval I) satisfies a first-order formula $\vartheta(x_0, x)$.

[1] The exponential blow-up comes from the succinct encodings of both constants in intervals of the form $\langle a, b\rangle$ in MITL and constants k in $\mathbf{C}_I^k$; for more details, see [38].

S. Alves and I. Mackie (Eds.): 13th International Workshop
on Developments in Computational Models 2023 (DCM'23).
EPTCS 408, 2024, pp. 73–89, doi:10.4204/EPTCS.408.5

$$
\begin{aligned}
\mathsf{LTL} &= \mathsf{Propositional\ Logic} \cup \{\varphi_1 \mathbf{U} \varphi_2, \varphi_1 \mathbf{S} \varphi_2 \mid \varphi_1, \varphi_2 \in \mathsf{LTL}\} \\
\mathsf{MITL} &= \mathsf{LTL} \cup \{\varphi_1 \mathbf{U}_I \varphi_2, \varphi_1 \mathbf{S}_I \varphi_2 \mid \varphi_1, \varphi_2 \in \mathsf{MITL}, I = \langle a, b\rangle, a, b \in \mathbb{N} \cup \{\infty\}, a < b\} \\
\mathsf{TLC} &= \mathsf{MITL} + \{\mathbf{C}_I^k \varphi, \overleftarrow{\mathbf{C}}_I^k \varphi \mid \varphi \in \mathsf{TLC}, I = [0, b\rangle, b \geq 1, k \geq 1\} \\
\mathsf{TLP} &= \mathsf{MITL} + \{\mathbf{P}_I^k \varphi, \overleftarrow{\mathbf{P}}_I^k \varphi \mid \varphi \in \mathsf{TLP}, I = [0, b\rangle, b \geq 1, k \geq 1\} \\
\mathsf{TLCI} &= \mathsf{MITL} + \{\mathbf{C}_I^k \varphi, \overleftarrow{\mathbf{C}}_I^k \varphi \mid \varphi \in \mathsf{TLCI}, I = \langle a, b\rangle, a, b \in \mathbb{N} \cup \{\infty\}, a < b\} \\
\mathsf{TLPI} &= \mathsf{MITL} + \{\mathbf{P}_I^k \varphi, \overleftarrow{\mathbf{P}}_I^k \varphi \mid \varphi \in \mathsf{TLPI}, I = \langle a, b\rangle, a, b \in \mathbb{N} \cup \{\infty\}, a < b\}
\end{aligned}
$$

Fig. 1: Some timed temporal logics considered in this paper. Note that the definitions of TLC and TLP in [18] are less general but equally expressive in the continuous semantics.

Expressiveness. It is of course trivial to see that Q2MLO subsumes TLC, but it is unclear (at least to us) whether Q2MLO can express the more general modalities $\mathbf{C}_I^k$ and their past counterparts, which count event occurrences within arbitrary non-singular intervals I of the form $\langle a, b\rangle$ with $0 \leq a < b$—on the face of it, we seem to need a first-order formula $\vartheta(x_1, x_2)$ along with two quantified instants $t_1, t_2 \in t_0 + I$, which is not allowed by the syntax of Q2MLO. In [38], it is claimed (without proof) that in the continuous semantics, MITL extended with such modalities (TLCI) is equally expressive as the fragment with only the most basic versions of the counting modalities (allowing only $I = (0, 1)$). By contrast, Krishna *et al.* [27] showed that in the pointwise semantics, $\mathbf{C}_I^k$ with $I = \langle a, b\rangle$ cannot be expressed in the *future* fragment of TLCI with only counting modalities with $I = [0, b\rangle$. In this paper, we reconcile these results and reaffirm the claim, i.e. we prove that $\mathbf{C}_I^k$ with $I = \langle a, b\rangle$ is indeed expressible in (future) Q2MLO in both the pointwise and continuous semantics. This suggests that Q2MLO is a very expressive and robust logic in both the pointwise and continuous semantics. From [22], we also know that in the pointwise semantics, $\mathbf{C}_I^k$ with $I = \langle a, b\rangle$ is expressible in the fragment of TLCI with (both future and past) counting modalities with $I = [0, b\rangle$.

Contributions. We argue that the folklore belief—$\mathbf{C}_I^k$ with $I = \langle a, b\rangle$ can be rewritten into formulae using only $\mathbf{C}_I^k$ with $I = [0, b\rangle$ in about the same way as $\mathbf{U}_I$ with $I = \langle a, b\rangle$ can be rewritten into $\mathbf{U}_I$ with $I = [0, b\rangle$—is not correct. We however show that by allowing *automata modalities* (or, equivalently, Q2MLO or Q2MSO [28]), one can indeed enforce that a sequence of events specified lies in the required interval; the proof is based on a generalisation of the techniques developed in [21] to show that *Extended Metric Interval Temporal Logic* (EMITL [42]) remains as expressive when restricted to only *unilateral* intervals, i.e. in the form of $[0, b\rangle$ or $\langle a, \infty)$. Building upon this insight, we 'correct' the folklore belief by showing that $\mathbf{C}_I^k$ with $I = \langle a, b\rangle$ can actually be expressed in $\mathbf{C}_I^k$ with $I = [0, b\rangle$ (without using $\overleftarrow{\mathbf{C}}_I^k$) in a more involved way (in the pointwise semantics as well, under some extra conditions).

Related work. Hirshfeld and Rabinovich [15, 16, 17, 18, 19, 20, 38] pioneered the research on decidable timed logics that extends MITL with counting and Pnueli modalities, which culminates in the strong metric predicate logic Q2MLO. Hunter [23] later proved that if MTL [25] (which is exactly like MITL, but singular I's are allowed) is extended in the same way, or equivalently if singular I's are allowed in Q2MLO, one obtains a logic that is expressively complete for FO[<, +1] (in the continuous semantics).

In the context of temporal logics and model checking, there are also some closely related results that are not directly comparable with the present paper. Extending LTL with *threshold counting* is first done

by Laroussinie *et al.* [29] where the 'until' (**U**) modality is extended with counting specifications. The timed versions of such modalities $\mathbf{UT}_I$ are studied by Krishna *et al.* in [27]. Another type of counting specification is *modulo counting*, which counts the number of events (seen so far) satisfying some monadic predicate modulo a given constant N. LTL extended with modulo counting modalities is first considered by Baziramwabo *et al.* [6], and Lodaya and Sreejith [30] showed that N can be encoded succinctly yet still retaining the PSPACE upper bound. Bednarczyk and Charatonik [7] studied the complexity of the satisfiability problem of the two variable fragment of first-order logic extended with modulo counting quantifiers interpreted over both trees and words. Similar operations also appear in other contexts, such as *temporal aggregation* [8] in databases and knowledge graphs.

2 Preliminaries

We give a brief account of the required background on timed logics. For more detailed reviews and comparisons of relevant results, we refer the readers to [10, 17]. Note that, in contrast with [18, 38, 42], we focus mainly on the *future* fragments of metric temporal logics.

Timed languages. A *timed word* over a finite alphabet Σ is an ω-sequence of *events* $(\sigma_i, \tau_i)_{i\geq 1}$ over $\Sigma \times \mathbb{R}_{\geq 0}$ with $(\tau_i)_{i\geq 1}$ a non-decreasing sequence of non-negative real numbers ('*timestamps*') such that for each $r \in \mathbb{R}_{\geq 0}$, there is some $j \geq 1$ with $\tau_j \geq r$ (i.e. we require all timed words to be '*non-Zeno*'). We denote by $T\Sigma^\omega$ the set of all the timed words over Σ. A *timed language* is a subset of $T\Sigma^\omega$.

Metric predicate logics. *Monadic Second-Order Logic of Order and Metric* (MSO[<,+1]) [4, 42] formulae over a finite set of atomic propositions (monadic predicates) AP are generated by

$$\vartheta ::= \top \mid X(x) \mid x < x' \mid d(x,x') \in I \mid \vartheta_1 \wedge \vartheta_2 \mid \neg\vartheta \mid \exists x\, \vartheta \mid \exists X\, \vartheta$$

where $X \in \mathsf{AP}$, x, x' are first-order variables, d is the distance predicate, $I \subseteq \mathbb{R}_{\geq 0}$ is an interval with endpoints in $\mathbb{N} \cup \{\infty\}$, and $\exists x$, $\exists X$ are first- and second-order quantifiers, respectively.We write, e.g., $(a, b\rangle$, to refer to (a, b) or $(a, b]$. We say that x (respectively X) is a *free* first-order (respectively second-order) variable in ϑ if it does not appear in the scope of $\exists x$ (respectively $\exists X$) in ϑ. We usually write $\vartheta(x_1, \ldots, x_m, X_1, \ldots, X_n)$ for ϑ, if $x_1, \ldots, x_m$ and $X_1, \ldots, X_n$ are free in ϑ. We say that an MSO[<,+1] formula $\vartheta(x)$ with only a free first-order variable x is a *future formula* if all the quantifiers appearing in $\vartheta(x)$ are relativised to (x, ∞), i.e. if $\exists x'\, \theta$ (respectively $\forall x'\, \theta$) is a subformula of $\vartheta(x)$, then θ is of the form $x < x' \wedge \theta'$ (respectively $x < x' \implies \theta'$). The fragment of MSO[<,+1] without second-order quantifiers is the *Monadic First-Order Logic of Order and Metric* (FO[<,+1]). The fragment of FO[<,+1] without the distance predicate is the *Monadic First-Order Logic of Order* (FO[<]). Q2MLO [15] is a fragment of FO[<,+1] obtained from FO[<] by allowing only non-singular I's (for the sake of decidability [4, 33]) and a restricted use of distance predicates. More precisely, Q2MLO is the smallest syntactic fragment of FO[<,+1] satisfying the following conditions:

- All FO[<] formulae $\vartheta(x)$ with only a free first-order variable x are Q2MLO formulae.
- If $\vartheta(x_0, x)$ is an FO[<] formula (possibly with Q2MLO formulae used as monadic predicates) where x_0, x are the only free first-order variables, then
  - $\exists x \left(x_0 < x \wedge d(x_0, x) \in I \wedge \vartheta(x_0, x)\right)$ and
  - $\exists x \left(x < x_0 \wedge d(x_0, x) \in I \wedge \vartheta(x_0, x)\right)$,

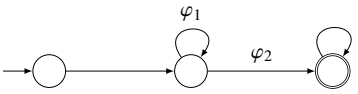


Fig. 2: The NFA $\mathcal{A}^{\mathbf{U}}$ for $\varphi_1 \mathbf{U}_I \varphi_2$.

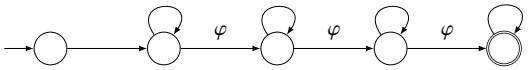


Fig. 3: The NFA $\mathcal{A}^{\mathbf{C},3}$ for $\mathbf{C}^3_I \varphi$.

where I is non-singular, are also Q2MLO formulae (with free first-order variable x_0).

The future fragment Q2MLO$^{\text{fut}}$ is obtained by allowing only $\vartheta(x)$ and $\exists x \left(x_0 < x \wedge d(x_0, x) \in I \wedge \vartheta(x_0, x)\right)$ above and also requiring them to be future formulae. In the same way we can define the corresponding fragments MSO[<], Q2MSO, and Q2MSO$^{\text{fut}}$ [28] of MSO[<, +1].

Metric temporal logics. A *non-deterministic finite automaton* (NFA) over Σ is a tuple $\mathcal{A} = \langle \Sigma, S, s_0, \Delta, F \rangle$ where S is a finite set of locations, $s_0 \in S$ is the initial location, $\Delta \subseteq S \times \Sigma \times S$ is the transition relation, and F is the set of final locations. We say that $\mathcal{A}$ is *deterministic* (a DFA) iff for each $s \in S$ and $\sigma \in \Sigma$, $|\{(s, \sigma, s') \mid (s, \sigma, s') \in \Delta\}| \leq 1$. A *run* of $\mathcal{A}$ on $\sigma_1 \dots \sigma_n \in \Sigma^+$ is a sequence of locations $s_0 s_1 \dots s_n$ where there is a transition $(s_i, \sigma_{i+1}, s_{i+1}) \in \Delta$ for each i, $0 \leq i < n$. A run of $\mathcal{A}$ is *accepting* iff it ends in a final location. A finite word is *accepted* by $\mathcal{A}$ iff $\mathcal{A}$ has an accepting run on it.

(Future) *Extended Metric Interval Temporal Logic* (EMITL$^{\text{fut}}$) [42] formulae over a finite set of atomic propositions AP are generated by

$$\varphi ::= \top \mid P \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathcal{A}_I(\varphi_1, \dots, \varphi_n)$$

where $P \in \mathsf{AP}$, $\mathcal{A}$ is an NFA over the n-ary alphabet $\{1, \dots, n\}$, and $I \subseteq \mathbb{R}_{\geq 0}$ is a non-singular interval with endpoints in $\mathbb{N} \cup \{\infty\}$.[2] We sometimes omit the subscript I when $I = [0, \infty)$ and write pseudo-arithmetic expressions for lower or upper bounds, e.g., '< 3' for $[0, 3)$. We also omit the arguments $\varphi_1, \dots, \varphi_n$ and simply write $\mathcal{A}_I$, if clear from the context. (Future) *Metric Interval Temporal Logic* (MITL$^{\text{fut}}$) [2] is the fragment of EMITL$^{\text{fut}}$ with only the 'until' modalities defined by the NFA $\mathcal{A}^{\mathbf{U}}$ in Fig. 2 (usually written in infix notation as $\varphi_1 \mathbf{U}_I \varphi_2$). We also use the usual shortcuts like $\bot \equiv \neg\top$, $\mathbf{X}_I \varphi \equiv \bot \mathbf{U}_I \varphi$, $\mathbf{F}_I \varphi \equiv \top \mathbf{U}_I \varphi$, $\overline{\mathbf{F}}_I \varphi \equiv \varphi \vee \mathbf{F}_I \varphi$, $\mathbf{G}_I \varphi \equiv \neg \mathbf{F}_I \neg\varphi$, and $\varphi_1 \mathbf{R}_I \varphi_2 \equiv \neg\big((\neg\varphi_1) \mathbf{U}_I (\neg\varphi_2)\big)$. (Future) *Linear Temporal Logic* (LTL$^{\text{fut}}$) [37] is the fragment of MITL$^{\text{fut}}$ where all modalities are labelled by $[0, \infty)$. TLC$^{\text{fut}}$ [18] is the fragment of EMITL$^{\text{fut}}$ obtained from MITL$^{\text{fut}}$ by adding the *counting modalities* $\mathbf{C}^k_I$, where I is a non-singular upper-bound interval (i.e. of the form $[0, b\rangle$ for some $b \in \mathbb{N}_{>0} \cup \{\infty\}$) and $k \geq 1$.[3] For example, $\mathbf{C}^3_I \varphi$ ('φ happens at least 3 times in I in the future') is defined by the NFA $\mathcal{A}^{\mathbf{C},3}$ in Fig. 3.

The definitions above are for the future versions of the modalities, but we note that we can also define the *past* versions of the modalities and correspondingly the full fragments of logics (denoted by names with no 'fut' superscripts), e.g., EMITL [42] and MITL [3].

Semantics. With each timed word $\rho = (\sigma_i, \tau_i)_{i \geq 1}$ over $\Sigma_{\mathsf{AP}} = 2^{\mathsf{AP}}$ we associate a structure M_ρ whose universe U_ρ is $\{i \mid i \geq 1\}$. The order relation $<$ and atomic propositions in AP are interpreted in the expected way, e.g., $P(i)$ holds in M_ρ iff $P \in \sigma_i$. The distance predicate $d(x, x') \in I$ holds iff $|\tau_x - \tau_{x'}| \in I$. The satisfaction relation for MSO[<, +1] is defined inductively as usual: we write $M_\rho, j_1, \dots, j_m, J_1, \dots, J_n \models \vartheta(x_1, \dots, x_m, X_1, \dots, X_n)$ (or simply $\rho, j_1, \dots, j_m, J_1, \dots, J_n \models \vartheta(x_1, \dots, x_m, X_1, \dots, X_n)$) if $j_1, \dots, j_m \in U_\rho$,

[2] For notational simplicity, we also use $\varphi_1, \dots, \varphi_n$ directly as transition labels (instead of $1, \dots, n$) in the figures.

[3] This definition is a mild generalisation of the modalities $\mathbf{C}_I$ in [18, 19] where I must be $(0, 1)$. Note that TLC is equivalent to the unilateral fragment of TLCI (defined later in Section 3), as intervals of the form $\langle a, \infty)$ can easily be eliminated in general.

$J_1, \ldots, J_n \subseteq U_\rho$, and $\vartheta(j_1, \ldots, j_m, J_1, \ldots, J_n)$ holds in M_ρ. We say that two $\mathsf{MSO}[<,+1]$ formulae $\vartheta_1(x)$ and $\vartheta_2(x)$ are *equivalent* if for all timed words $\rho = (\sigma_i, \tau_i)_{i \geq 1}$ and $j \in U_\rho$,

$$\rho, j \models \vartheta_1(x) \iff \rho, j \models \vartheta_2(x) \,.$$

Given a $\mathsf{EMITL}^{\mathsf{fut}}$ formula φ over AP, a timed word $\rho = (\sigma_i, \tau_i)_{i \geq 1}$ over $\Sigma_{\mathsf{AP}} = 2^{\mathsf{AP}}$ and a *position* $i \geq 1$, we define the satisfaction relation $\rho, i \models \varphi$ as follows:

- $\rho, i \models \top$;
- $\rho, i \models p$ iff $p \in \sigma_i$;
- $\rho, i \models \varphi_1 \wedge \varphi_2$ iff $\rho, i \models \varphi_1$ and $\rho, i \models \varphi_2$;
- $\rho, i \models \neg\varphi$ iff $\rho, i \not\models \varphi$;
- $\rho, i \models \mathcal{A}_I(\varphi_1, \ldots, \varphi_n)$ iff there exists $j \geq i$ such that (i) $\tau_j - \tau_i \in I$ and (ii) there is an accepting run of $\mathcal{A}$ on $a_i \ldots a_j$ where $\rho, \ell \models \varphi_{a_\ell}$ ($a_\ell \in \{1, \ldots, n\}$) for each ℓ, $i \leq \ell \leq j$.

We say that ρ *satisfies* φ (written $\rho \models \varphi$) iff $\rho, 1 \models \varphi$.

The definitions above correspond to the so-called *pointwise* semantics of timed logics [4, 5, 34, 42]. It is also possible to define the *continuous* semantics of timed logics over timed words by taking $\mathbb{R}_{\geq 0}$ instead of $\{i \mid i \geq 1\}$ as the universe and $d(x, x') = |x - x'|$; we refer the readers to [9, 11, 32] for details. While we focus on the former in this paper, it is clear that all of our results carry over to the continuous interpretations of timed logics where system behaviours are modelled as (finitely variable) signals.

Expressiveness. We say that a metric logic L' is *expressively complete* for a metric logic L iff for any formula $\vartheta(x) \in L$, there is an equivalent formula $\varphi(x) \in L'$.[4] We say that L' is *at least as expressive as* (or *more expressive than*) L (written $L \subseteq L'$) iff for any formula $\vartheta(x) \in L$, there is an *initially equivalent* formula $\varphi(x) \in L'$ (i.e., $\vartheta(1)$ and $\varphi(1)$ evaluate to the same truth value for any timed word). If $L \subseteq L'$ but $L' \not\subseteq L$ then we say that L' is *strictly more expressive than* L (or L is *strictly less expressive than* L'). We write $L \equiv L'$ iff $L \subseteq L'$ and $L' \subseteq L$. For the purpose of this paper, the most relevant known expressiveness results are $\mathsf{EMITL}^{\mathsf{fut}} \equiv \mathsf{Q2MSO}^{\mathsf{fut}}$ and *aperiodic* [31, 40] $\mathsf{EMITL}^{\mathsf{fut}} \equiv \mathsf{Q2MLO}^{\mathsf{fut}}$ [28], and thus we will freely mix the use of them.

3 Expressing counting modalities

Counting events in arbitrary intervals. We start by giving an alternative and more general definition (in terms of $\mathsf{FO}[<,+1]$) of what do we mean by counting events in an interval I. Note that the following definition of $\mathbf{C}_I^k \varphi$ is equivalent to the definition based on automata modalities in Section 2 for the special case where I is of the form $[0, b\rangle$.

Definition 1 ($\mathsf{TLCI}^{\mathsf{fut}}$ [38])**.** $\mathsf{TLCI}^{\mathsf{fut}}$ is obtained from $\mathsf{MITL}^{\mathsf{fut}}$ by adding the (one-place) modalities $\mathbf{C}_I^k$ defined by the following formula (where I is non-singular):

$$\vartheta_I^{\mathbf{C},k}(x, X) = \exists x_1 \ldots \exists x_k \left(x < x_1 < \cdots < x_k \wedge d(x, x_1) \in I \wedge d(x, x_k) \in I \wedge \bigwedge_{1 \leq i \leq k} X(x_i)\right) .$$

TLCI is obtained by adding the past counterparts of the modalities above (defined symmetrically).

[4] Formulae of metric temporal logics in this paper are $\mathsf{MSO}[<,+1]$ formulae with a single free first-order variable.

We first note that while $\vartheta_I^{\mathbf{C},k}(x,X)$ is in FO[<,+1], it is not in Q2MLO (at least syntactically), thus it is not immediately clear how to express it in TLC (with both the future and past modalities) even in the continuous semantics, as the translation from Q2MLO to TLC in [18,20] does not apply. It should also be clear that the trivial attempt of simply decorating $\mathcal{A}^{\mathbf{C},k}$ with an arbitrary non-singular I would not give a formula equivalent to $\vartheta_I^{\mathbf{C},k}$. For example, the following timed word

$$(\emptyset,0)(\{P\},0.5)(\{P\},1.5)(\{P\},2.5)(\{P\},3.5)\dots$$

satisfies $\mathcal{A}^{\mathbf{C},3}_{(2,3)}\,P$, but clearly $\rho,1 \not\models \vartheta^{\mathbf{C},3}_{(2,3)}(x,X)$. In [38], it is stated that TLC is as expressive as TLCI, but no complete proof is given. In [12] the following equivalence, which is reminiscent of how MITL and Q2MLO with arbitrary non-singular intervals can be reduced to their base versions using only $I=(0,1)$ in the continuous semantics [14, 16, 18], is proposed:

$$\mathbf{C}^k_{(a,a+1)}\,P \iff \mathbf{G}_{(0,1)}\,\mathbf{F}_{(0,a)}\,\mathbf{C}^k_{(0,1)}\,P\,. \tag{1}$$

This is, however, not correct in either the pointwise or the continuous semantics—for instance, if $k=2$ and $a=2$, then any timed word with only one event at τ_1+1, two P-events in $\tau_1+(1,2)$, and no P-event in $\tau_1+(2,3)$ satisfies the right-hand side of (1), but not its left-hand side; if $k=2$ and $a=1$, then

$$(\emptyset,0)(\emptyset,0.6)(\emptyset,0.7)(\{P\},0.8)(\{P\},0.9)(\emptyset,1.6)(\{P\},1.7)(\{P\},2.1)\dots$$

satisfies the right-hand side of (1), but not its left-hand side.

In the study of timed logics, it is common to rule out constraints involving singular ('*punctual*') intervals as they can easily render the *satisfiability problem* undecidable (or have prohibitively high complexity [34]). If we do however allow singular intervals, then the following equivalence clearly holds in the continuous semantics:

$$\mathbf{C}^k_{(a,a+1)}\,P \iff \mathbf{F}_{=a}\,\mathbf{C}^k_{(0,1)}\,P\,. \tag{2}$$

Indeed, the main difficulty in expressing (2) in TLC is the lack of ability to express punctuality—roughly speaking, $\mathbf{G}_{(0,1)}\,\mathbf{F}_{(0,1)}\,\varphi$ is a weaker requirement than $\mathbf{F}_{=1}\,\varphi$: the former is also satisfied by two points that both satisfy φ, surround $t+1$ (where t is the current time), and separated by less than 1. Therefore, while $\mathbf{G}_{(0,1)}\,\mathbf{F}_{(0,1)}\,\mathbf{C}^k_{(0,1)}\,P$ implies $\mathbf{F}_{(1,2)}\,\mathbf{C}^k_{(0,1)}\,P$ or $\mathbf{F}_{=1}\,\mathbf{C}^k_{(0,1)}\,P$, it does not guarantee that all the k 'witnesses' lie within $t+(1,2)$ in the former case. On the other hand, $\mathbf{F}_{(0,1)}\,\mathbf{G}_{(0,1)}\,\mathbf{C}^k_{(0,1)}\,P$ does not necessarily hold when $\mathbf{F}_{=1}\,\mathbf{C}^k_{(0,1)}\,P$ holds, as $\mathbf{F}_{(0,1)}\,\mathbf{G}_{(0,1)}\,\varphi$ is a stronger requirement than $\mathbf{F}_{=1}\,\varphi$.

Before we explain how to express $\mathbf{C}^k_I$ for the general case where $I=\langle a,b\rangle$ with $a<b$ in Q2MLO$^{\text{fut}}$ in the next section, let us first mention two simple ways that do not involve punctuality to express them in non-trivial extensions of MITL.

Counting events in I by automata modalities. In the case of counting where each witness is 'context free', instead of trying to locate a suitable point where $\mathbf{C}^k_{(0,1)}\,P$ holds (like in (1)), we can specify that there are k *distinct* P-events in $t+(a,a+1)$—this can be done with k modulo-k counters, similar to an idea used in [27]. For example, if $k=3$ we use three automata modalities that accept every $(3n)$-th, $(3n+1)$-th, and $(3n+2)$-th P-event, respectively, and then specify that each of them has a run that ends in $t+(a,a+1)$. The following theorem is then immediate.

Theorem 1. TLCI$^{\text{fut}} \subseteq$ *aperiodic* EMITL$^{\text{fut}} \equiv$ Q2MLO$^{\text{fut}}$.

This idea, however, does not easily generalise to TLPI, which we discuss in the next section.

Counting events in I by rational constants. Recall from [24] that $\mathbf{C}^2_{(0,1)}\,P$ can be expressed as the disjunction of $\mathbf{F}_{(0,\frac{1}{2})}(P \wedge \mathbf{F}_{(0,\frac{1}{2})}\,P)$, $\mathbf{F}_{(\frac{1}{2},1)}(P \wedge \overleftarrow{\mathbf{F}}_{(0,\frac{1}{2})}\,P)$, and $\mathbf{F}_{(0,\frac{1}{2})}\,P \wedge \mathbf{F}_{(\frac{1}{2},1)}\,P$ (where $\overleftarrow{\mathbf{F}}$ is the past version of $\mathbf{F}$). This can easily be generalised (like in [24], but with trivial modifications to avoid using punctualities) to arbitrary non-singular I and larger values of k, e.g., for $\mathbf{C}^3_{(1,2)}\,P$, we partition $(1,2)$ into 6 subintervals and consider the cases where 1) all three witnesses lie within one of the three subintervals covering $(1,1.5)$; 2) all three witnesses lie within one of the three subintervals covering $(1.5,2)$; and 3) not all witnesses lie within a single subinterval.

Theorem 2. MITL *(with both the future and past modalities) is expressively complete for* TLCI*, if rational constants are allowed.*

This also applies straightforwardly to TLPI. On the other hand, MITL with *only one of* these extensions—i.e. either past modalities [35] or rational constants [9]— is insufficient for expressing TLPI.

4 Expressing $\mathbf{P}^2_I$ in Q2MLO$^{\mathsf{fut}}$

A more general form of counting, where one can specify a sequence of distinct events, is enabled by the *Pnueli modalities* $\mathbf{P}^k_I$ defined below. Once again, [38] states that they are expressible in TLC without proof.

Definition 2 (TLPI$^{\mathsf{fut}}$ [38])**.** TLPI$^{\mathsf{fut}}$ is obtained from MITL$^{\mathsf{fut}}$ by adding the (k-place) modalities $\mathbf{P}^k_I$ defined by the following formula (where I is non-singular):

$$\vartheta^{\mathbf{P},k}_I(x,X_1,\ldots,X_k) = \exists x_1 \ldots \exists x_k \left(x < x_1 < \cdots < x_k \wedge d(x,x_1) \in I \wedge d(x,x_k) \in I \wedge \bigwedge_{1\le i\le k} X_i(x_i)\right).$$

TLPI is obtained by adding the past counterparts of the modalities above (defined symmetrically).

The modulo-k trick that we used earlier to express $\mathbf{C}^k_I$ no longer works in the case of Pnueli modalities, as obviously we must also ensure that $X_1,\ldots,X_k$ are satisfied *in this order* by a sequence of events in I. We now describe a general construction of Q2MLO$^{\mathsf{fut}}$ formulae (or, equivalently, aperiodic EMITL$^{\mathsf{fut}}$ formulae where all automata modalities are definable by LTL$^{\mathsf{fut}}$ or future FO[<] formulae [28]) that specify sequences of events in arbitrary non-singular intervals. For simplicity, we will use $\mathbf{P}^2_{(a,a+1)}(P,Q)$ with $a \ge 1$ as an example to explain the ideas involved before we extend the construction to the general case where the sequence of events is specified by a first- or second-order formula in the next section.

Let us call a pair of positive integers $\langle h,\ell\rangle$ where $h \le \ell$ a *segment*. Given a timed word $\rho = (\sigma_i,\tau_i)_{i\ge1}$ over Σ_{AP} where $\mathsf{AP} = \{P,Q\}$, we say that a segment $\langle h,\ell\rangle$ is a *witness* for $\mathbf{P}^2_{(a,a+1)}(P,Q)$ at i if $h < \ell$, $P \in \sigma_h$, $Q \in \sigma_\ell$, $\langle h,\ell\rangle$ is *minimal* in the sense that there is no h',ℓ' such that $h \le h' \le \ell' \le \ell$, either $h < h'$ or $\ell' < \ell$, and $\langle h',\ell\rangle$ also satisfies the conditions above, and both $\tau_h,\tau_\ell \in \tau_i + (a,a+1)$. In other words, $\rho,h \models \exists x'\,\varphi_1(x,x')$ where

$$\varphi_1(x,x') = x < x' \wedge P(x) \wedge Q(x') \wedge \neg\exists y \left(x < y < x' \wedge \big(P(y) \vee Q(y)\big)\right).$$

The idea is that $\exists x'\,\varphi_1(x,x')$ holds at the starting points h of all the *potential witnesses* (witnesses but without the timing requirement in relation to τ_i) for $\mathbf{P}^2_{(a,a+1)}(P,Q)$. For each $i \ge 1$, we either have $\rho,i \models \exists x'\,\varphi_1(x,x')$ or $\rho,i \not\models \exists x'\,\varphi_1(x,x')$, and this gives rise to a (finite or infinite) sequence of potential witnesses for $\mathbf{P}^2_{(a,a+1)}(P,Q)$:

$$\langle h_1,\ell_1\rangle\langle h_2,\ell_2\rangle\ldots$$

where $h_1 < h_2 < \ldots$. From the definition of φ_1, it is clear that $\ell_j \leq h_{j+1}$ for all j (i.e. the potential witnesses for $\mathbf{P}^2_{(a,a+1)}(P,Q)$ do not overlap except possibly on the endpoints).

Now, to specify that $\rho, i \models \mathbf{P}^2_{(a,a+1)}(P,Q)$, we want to express the condition that some potential witness $\langle h_j, \ell_j \rangle$ for $\mathbf{P}^2_{(a,a+1)}(P,Q)$ actually satisfies the timing requirement $\tau_{h_j}, \tau_{\ell_j} \in \tau_i + (a, a+1)$. We start from this initial attempt to express $\mathbf{P}^2_{(a,a+1)}(P,Q)$:

$$\varphi_{wit} = \mathbf{F}_{(a,a+1)}\, \varphi_1 \wedge \mathcal{A}^1_{(a,a+1)}$$

where φ_1 is the LTL formula equivalent to $\exists x'\, \varphi_1(x,x')$, $\mathcal{A}^1$ is the equivalent NFA for $\varphi'_1(x,x') = \exists y \left(x < y < x' \wedge \varphi_1(y,x')\right)$.[5] Intuitively, $\mathbf{F}_{(a,a+1)}\, \varphi_1$ says that $d(i,h_j) \in (a,a+1)$ for some j, and $\mathcal{A}_{(a,a+1)}$ says that $d(i,\ell_{j'}) \in (a,a+1)$ for some j'. But it is not hard to see that an undesired scenario (illustrated in Fig. 4), where no potential witness $\langle h, \ell \rangle$ for $\mathbf{P}^2_{(a,a+1)}(P,Q)$ lies completely within $\tau_i + (a,a+1)$, also satisfies φ_{wit}. To capture and rule out this undesired scenario, note that in Fig. 4 it is clear that the time elapsed between h_j and ℓ_{j+1} is greater or equal than 1. Based on this observation, we can write a formula involving the two adjacent potential witnesses $\langle h_j, \ell_j \rangle$ and $\langle h_{j+1}, \ell_{j+1} \rangle$ for $\mathbf{P}^2_{(a,a+1)}(P,Q)$:

$$\varphi_2(x,y') = \exists x' \exists y \left(x < y \wedge x \leq x' \wedge y \leq y' \wedge \varphi_1(x,x') \wedge \varphi_1(y,y') \wedge \neg \exists z \exists z' \left(x < z < y \wedge z \leq z' \wedge \varphi_1(z,z') \right) \right).$$

To express $d(h_j, \ell_{j+1}) \geq 1$, we just check if the Q2MLO$^{\text{fut}}$ formula

$$\varphi_2^{\geq 1}(x) = \exists y' \left(x < y' \wedge d(x,y') \geq 1 \wedge \varphi_2(x,y') \right)$$

holds at position h_j. It remains to enforce the following conditions:

- $\langle h_j, \ell_j \rangle$ is the *last* segment $\langle h, \ell \rangle$ with $\tau_h \leq \tau_i + a$.
- $\tau_{\ell_{j+1}} \geq \tau_i + (a+1)$; see Fig. 5 for an example when $\langle h_{j+1}, \ell_{j+1} \rangle$ lies completely within $\tau_i + (a,a+1)$ but $\varphi_2^{\geq 1}(x)$ holds at h_j.

We now use the following crucial lemma to locate the last $\langle h, \ell \rangle$ with $\tau_h \leq \tau_i + a$.

Lemma 1. *For any $\rho = (\sigma_i, \tau_i)_{i \geq 1}$ over Σ_{AP} where $\mathsf{AP} = \{P, Q\}$, the* Q2MLO$^{\text{fut}}$ *formula $\varphi_2^{\geq 1}(x)$ is satisfied by at most $2a+2$ positions $j > i$ with $d(i,j) \in [0,a]$ for any $i \geq 0$.*

Proof. Let $\langle h_1, \ell_1 \rangle \langle h_2, \ell_2 \rangle \ldots$ be the sequence of potential witnesses for $\mathbf{P}^2_{(a,a+1)}(P,Q)$ as described above. If $\rho, h_j \models \varphi_2^{\geq 1}(x)$, then either there is no $\langle h_{j+2}, \ell_{j+2} \rangle$ or $\tau_{h_{j+2}} \geq \tau_{h_j} + 1$. It follows that if there are $2a+3$ positions satisfying $\varphi_2^{\geq 1}(x)$, then the first and the last of them must be more than a apart. □

It follows that the undesired scenario #1 is captured by

$$\varphi_{\text{out}} = \bigvee_{1 \leq k \leq 2a+2} \left(\mathbf{C}^k_{\leq a}(\mathcal{A}^2_{\geq 1}) \wedge \neg \mathbf{C}^{k+1}_{\leq a}(\mathcal{A}^2_{\geq 1}) \wedge \mathcal{B}^k_{\geq a+1} \right)$$

where $\mathcal{A}^2$ is the equivalent NFA for $\varphi_2(x,y')$ (i.e. $\mathcal{A}^2_{\geq 1} \equiv \varphi_2^{\geq 1}(x)$) and $\mathcal{B}^k$ is the equivalent NFA for

$$\varphi_2^k(x,x') = \exists x_1 \ldots \exists x_k \Big(x < x_1 < \cdots < x_k < x' \wedge \varphi_2^{\geq 1}(x_1) \wedge \cdots \wedge \varphi_2^{\geq 1}(x_k) \wedge \varphi_2(x_k, x') \\ \wedge \neg \exists y \left(x \leq y \leq x_k \wedge \bigwedge_{1 \leq j \leq k} (y \neq x_k) \wedge \varphi_2^{\geq 1}(y) \right) \Big);$$

[5] Technically, we can use a theorem in [13] to get equivalent finite-word LTL formulae (over infinite-word LTL formulae as monadic predicates) for FO[<] formulae of the form $\varphi(x,x')$.

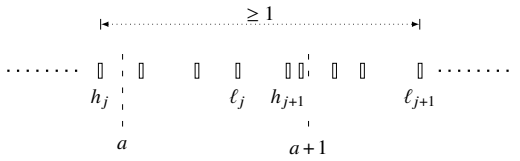


Fig. 4: Undesired scenario #1.

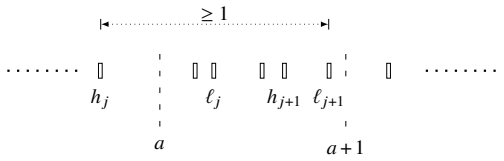


Fig. 5: Desired scenario with $d(h_j, \ell_{j+1}) \geq 1$.

it can be obtained by regarding $\varphi_2^{\geq 1}$ as an atomic proposition and replace it afterwards by $\mathcal{A}^2_{\geq 1}$. Specifically, the first two conjuncts specify that the number of positions satisfying $\varphi_2^{\geq 1}(x)$ before $\tau_i + a$ is exactly k, and the last conjunct ensures that the second potential witness in this pair is out of bounds, i.e. $\tau_{\ell_{j+1}} \geq \tau_i + (a+1)$. The desired formula is

$$\varphi^{\mathbf{P},2}_{(a,a+1)}(P,Q) = \varphi_{wit} \wedge \neg \varphi_{out} \,.$$

Proposition 1. $\varphi^{\mathbf{P},2}_{(a,a+1)}(P,Q) \equiv \mathbf{P}^2_{(a,a+1)}(P,Q)$.

Proof. If $\varphi^{\mathbf{P},2}_{(a,a+1)}(P,Q)$ holds at i then either there is a potential witness $\langle h, \ell \rangle$ for $\mathbf{P}^2_{(a,a+1)}(P,Q)$ that lies completely within $\tau_i + (a, a+1)$ (in which case $\mathbf{P}^2_{(a,a+1)}(P,Q)$ holds), or we are in the scenario in Fig. 4—but this is impossible, as one of the disjuncts of φ_{out} must hold at i, as argued above. If $\mathbf{P}^2_{(a,a+1)}(P,Q)$ holds at i, then we have a witness $\langle h, \ell \rangle$ for $\mathbf{P}^2_{(a,a+1)}(P,Q)$ at i that lies completely within $\tau_i + (a, a+1)$, and φ_{wit} clearly holds at i too. If $\mathbf{C}^k_{\leq a}(\mathcal{A}^2_{\geq 1}) \wedge \neg \mathbf{C}^{k+1}_{\leq a}(\mathcal{A}^2_{\geq 1})$ indeed holds at i for some k then $\mathcal{B}^k_{\geq a+1}$ must not hold at i: if $\langle h_j, \ell_j \rangle$ and $\langle h_{j+1}, \ell_{j+1} \rangle$ are potential witnesses for $\mathbf{P}^2_{(a,a+1)}(P,Q)$ and h_j is the k-th point satisfying $\varphi_2^{\geq 1}(x)$, we must have $h_{j+1} \leq h$ and $\tau_{\ell_{j+1}} \in \tau_i + (a, a+1)$. □

5 Expressing more general properties in Q2MLO$^{\text{fut}}$

We now consider the more general case where the desired behaviour in I is specified as a future FO[<] formula $\psi(x', x'')$.[6] Formally, the property that we want to express is

$$\vartheta^{\psi}_I(x) = \exists x' \, \exists x'' \left(x < x' \leq x'' \wedge d(x, x') \in I \wedge d(x, x'') \in I \wedge \psi(x', x'') \right) .$$

To simplify the analysis, we first modify $\psi(x', x'')$ into $\psi_1(x', x'')$ to rule out witnesses that are not minimal:

$$\psi_1(x', x'') = \psi(x', x'') \wedge \neg \Big(\exists y \, \exists z \left(x' \leq y \leq z \leq x'' \wedge (x < y \vee z < x') \wedge \psi(y, z) \right) \Big) .$$

Similarly as before, $\exists x'' \, \psi_1(x', x'')$ holds at the starting points of all the potential witnesses for ϑ^{ψ}_I. However, as opposed to the case of $P^2_{(a,a+1)}(P,Q)$, now the potential witnesses may overlap non-trivially. In particular, if $\rho, i \models \psi_{wit}$ where ψ_{wit} is defined in the same way as φ_{wit} in the last section, there is one more possible undesired scenario (illustrated in Fig. 6; note in particular that $\psi_1(h_{j+2}, \ell_j)$ does not hold). Thanks to the finite-state nature of $\psi_1(x', x'')$, the scenario in Fig. 6 can also be ruled out in the same way: in this particular case, either $d(h_j, \ell_{j+1}) \geq 1$ or $d(h_{j+1}, \ell_{j+2}) \geq 1$ must hold. This is made possible by the following lemma that gives an upper bound on the number of positions satisfying $\psi_2^{\geq 1}(x)$ (defined from $\psi_1(x', x'')$ in the same way as $\varphi_2^{\geq 1}(x)$) before $\tau_i + a$.[7]

[6] The proof applies also to the case where $\psi(x', x'')$ is a second-order formula.

[7] Similar observations based on Shelah's *composition method* [41] have also been used in [18, 20].

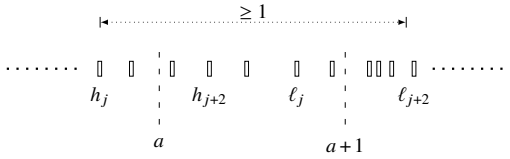


Fig. 6: Undesired scenario #2 where $\psi_1(h_{j+2}, \ell_j)$ does not hold.

Lemma 2. *For any $\rho = (\sigma_i, \tau_i)_{i\geq 1}$ over Σ_{AP}, the* Q2MLO *formula $\psi_2^{\geq 1}(x)$ over* AP *is satisfied by at most $(m+1)\cdot(a+1)$ positions $j > i$ (where m is the number of locations in the minimal equivalent* DFA *for $\psi_1(x,x')$) with $d(i,j) \in [0,a]$ for any $i \geq 0$.*

Proof sketch. Any point cannot intersect with more than m potential witnesses for ϑ_I^{ψ} (otherwise there will be a contradiction with the minimality of potential witnesses), and this implies that if $\rho, h_j \models \psi_2^{\geq 1}(x)$, then either there is no $\langle h_{j+m+1}, \ell_{j+m+1}\rangle$ or $\tau_{h_{j+m+1}} \geq \tau_{h_j} + 1$. □

We then obtain the following theorem.

Theorem 3. *The property 'the future* FO[<,+1] *formula $\psi(x',x'')$ is satisfied by positions x', x'' in I in the future' can be expressed in* $\mathsf{EMITL}^{\mathsf{fut}} \equiv \mathsf{Q2MLO}^{\mathsf{fut}}$.

The theorem also holds for the general case where $\psi(x',x'')$ is a non-future FO[<,+1] formula; in this case, the property can be expressed in EMITL ≡ Q2MLO.

6 Expressing $\mathbf{C}_I^k$ in $\mathsf{TLC}^{\mathsf{fut}}$

From [21] we know that in the pointwise semantics, (aperiodic) EMITL (or Q2MLO) formulae can be rewritten into simpler equivalent formulae where all intervals are unilateral, and in fact it suffices to use $[0,b\rangle$ and $[0,\infty)$ [22]. For the aperiodic case, such a formula can even be expressed with the simpler counting modalities as below, if we allow both the future and past versions of them:

- $\mathbf{C}_I^k$ and $\overleftarrow{\mathbf{C}}_I^k$ with $I = (0,1)$ in the continuous semantics [18, 20]; or
- $\mathbf{C}_I^k$ and $\overleftarrow{\mathbf{C}}_I^k$ with $I = [0,b\rangle$ in the pointwise semantics [22].

We now show that for the special case of $\mathbf{C}_I^k$, i.e. when the $\mathsf{Q2MLO}^{\mathsf{fut}}$ formula in question is a $\mathsf{TLCI}^{\mathsf{fut}}$ formula, we can do the same *with only the future modalities*; this can be seen as a strict generalisation of the 'well-known' reduction from $\mathbf{U}_I$ with $\langle a,b\rangle$ to $\mathbf{U}_I$ with $I = [0,b\rangle$ discussed earlier [14, 16, 18]. In the presentation below we will focus on the pointwise case, where some additional conditions must be satisfied (as explained below), but these conditions are automatically satisfied in the continuous semantics.

Expressing $\mathbf{F}_I$ with $I = \langle a,b\rangle$. We start by rewriting the 'eventually' modalities $\mathbf{F}_I$, which can actually be regarded as a special case of $\mathbf{C}_I^k$ with $k = 1$ [18]; for simplicity, let us consider a subformula $\mathbf{F}_I\varphi$ where φ is in unilateral $\mathsf{MITL}^{\mathsf{fut}}$ and $I = (a, a+1)$, $a \geq 1$. It is well known that in the pointwise semantics, such modalities cannot be expressed in unilateral MITL [39]. To overcome this apparent difficulty, let us define a family of formulae for all $m \in \{0,\ldots,a-1\}$:

$$\Phi^0 = \{\varphi\}\,,$$
$$\Phi^{m+1} = \{\mathbf{X}_{>0}\top \wedge \neg\varphi^m \mathbf{U}_{\leq 1}\varphi^m \wedge \neg\varphi^m \mathbf{U}_{\geq 1}\varphi^m, \mathbf{G}_{(0,1)}\varphi^m \mid \varphi^m \in \Phi^m \text{ or } \neg\varphi^m \in \Phi^m\}\,.$$

All these formulae are in unilateral $\mathsf{MITL}^{\mathsf{fut}}$: $\mathbf{G}_{(0,1)}\varphi^m \equiv (\mathbf{X}_{>0}\top \wedge \mathbf{G}_{[0,1)}\varphi^m) \vee \mathbf{F}_{\leq 0}\mathbf{G}_{[0,1)}\varphi^m$. Additionally, we assume that the timed word $\rho = (\sigma_i,\tau_i)_{i\geq 1}$ in question satisfies the following condition:

- For every $m \in \{0,\dots,a-1\}$ and $\varphi^m \in \Phi^m$, if $\rho, j \models \varphi^m$ and $\rho, j' \not\models \varphi^m$ for all $j' < j$ with $d(j', j) < 1$, then there exists i in ρ such that $d(i, j) = 1$ (unless $d(1, j) < 1$).

We note that in practical applications, this should not be a severe limitation—for example in model checking, if the system is modelled as a timed automaton [1], one can simply add a self-loop labelled with an extra 'empty' letter ϵ to each location, and use the following formula (which is easily expressible in unilateral $\mathsf{MITL}^{\mathsf{fut}}$) as a precondition:

$$\vartheta^{\mathbf{F}} = \bigwedge_{\substack{\varphi^m \in \Phi^m \\ m \in \{0,\dots,a-1\}}} \neg\exists x\, \exists x' \Big(x < x' \wedge \nexists x'' \, (x < x'' < x') \\ \wedge \exists y \Big(x < y \wedge d(x,y) > 1 \wedge d(x',y) < 1 \wedge \varphi^m(y) \wedge \nexists z \, (x < z < y \wedge \varphi^m(z)) \Big) \Big) .$$

Intuitively, $\vartheta^{\mathbf{F}}$ rules out the situations when $\mathbf{X}_{>0}\top \wedge \neg\varphi^m \mathbf{U}_{\leq 1} \varphi^m \wedge \neg\varphi^m \mathbf{U}_{\geq 1} \varphi^m \in \Phi^{m+1}$ should hold at x'', but x'' does not exist in ρ. With the condition in place, we now show that $\mathbf{F}_{\langle a-m, a-m+1\rangle} \varphi^{m'}$ where $\varphi^{m'} \in \Phi^{m'}$ can be expressed in unilateral $\mathsf{MITL}^{\mathsf{fut}}$ for $m \in \{0,\dots,a\}$ and $m' \leq m$. For the base step $m = a$, note that $\mathbf{F}_{(0,1\rangle} \varphi^{m'} \equiv (\mathbf{X}_{>0}\top \wedge \mathbf{F}_{[0,1\rangle} \varphi^{m'}) \vee \mathbf{F}_{\leq 0}(\mathbf{X}_{>0}\top \wedge \mathbf{F}_{[0,1\rangle} \varphi^{m'})$. For the inductive step (from $m+1$ to m), suppose that we want to express $\rho, i \models \mathbf{F}_{(a-m, a-m+1)} \varphi^m$ where $\varphi^m \in \Phi^m$ and let $\ell > i$ be the *minimal* position such that $\rho, \ell \models \varphi^m$ and $d(i, \ell) \in (a-m, a-m+1)$ (the arguments for other types of intervals are exactly similar). We can then essentially follow [21] but only need to consider the cases below:

- There exists (a maximal) j, $i < j < \ell$ such that $d(j, \ell) = 1$ and $\rho, j \models \mathbf{X}_{>0}\top \wedge \neg\varphi^m \mathbf{U}_{\leq 1} \varphi^m \wedge \neg\varphi^m \mathbf{U}_{\geq 1} \varphi^m$: we have

  $$\rho, i \models \zeta_1 = \mathbf{F}_{(a-m-1, a-m)} (\mathbf{X}_{>0}\top \wedge \neg\varphi^m \mathbf{U}_{\leq 1} \varphi^m \wedge \neg\varphi^m \mathbf{U}_{\geq 1} \varphi^m)$$

  where $\mathbf{X}_{>0}\top \wedge \neg\varphi^m \mathbf{U}_{\leq 1} \varphi^m \wedge \neg\varphi^m \mathbf{U}_{\geq 1} \varphi^m \in \Phi^{m+1}$.
- There exists j, $i < j < \ell$ such that $d(j, \ell) < 1$, $d(i, j) \in (a-m-1, a-m]$ and $\rho, j \models \varphi^m$: we have

  $$\rho, i \models \zeta_2 = \mathbf{F}_{(a-m-1, a-m]} \varphi^m \wedge \neg \mathbf{F}_{(a-m-1, a-m]} \mathbf{G}_{(0,1)} (\neg\varphi^m)$$

  where $\mathbf{G}_{(0,1)} (\neg\varphi^m) \in \Phi^{m+1}$.

The equivalent formula is $\zeta_1 \vee \zeta_2$, which can be rewritten into a unilateral $\mathsf{MITL}^{\mathsf{fut}}$ formula by the induction hypothesis. It follows that $\mathbf{F}_{\langle a, a+1\rangle} \varphi^0$, where $\varphi^0 = \varphi \in \Phi^0$ is an arbitrary $\mathsf{MITL}^{\mathsf{fut}}$ formula, can be expressed in unilateral $\mathsf{MITL}^{\mathsf{fut}}$, as desired.

Expressing $\mathbf{C}^k_I$ with $I = \langle a, b\rangle$. We now consider a subformula $\mathbf{C}^k_I \psi$ where ψ is in $\mathsf{TLC}^{\mathsf{fut}}$, $k \geq 2$, and $I = (a, a+1)$, $a \geq 1$. Define a family of formulae for all $m \in \{1,\dots,a-1\}$:

$$\Psi^1 = \{(\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]} \psi \wedge \neg\, \mathbf{C}^k_{[0,1)} \psi\} ,$$
$$\Psi^{m+1} = \{\mathbf{X}_{>0}\top \wedge \neg\psi^m \mathbf{U}_{\leq 1} \psi^m \wedge \neg\psi^m \mathbf{U}_{\geq 1} \psi^m, \mathbf{G}_{(0,1)} \psi^m \mid \psi^m \in \Psi^m \text{ or } \neg\psi^m \in \Psi^m\} .$$

All these formulae are in $\mathsf{TLC}^{\mathsf{fut}}$. Now we assert that $\rho = (\sigma_i, \tau_i)_{i \geq 1}$ satisfies the following conditions:

(C1) If $\rho, j \models \psi$ and there are

- less than k positions $j' < j$ with $0 < d(j', j) < 1$ such that $\rho, j' \models \psi$, and

- at least k positions $j' \le j$ with $0 \le d(j', j) < 1$ such that $\rho, j' \models \psi$,

then there exists i in ρ such that $d(i, j) = 1$ (unless $d(1, j) < 1$).

(C2) For every $m \in \{1, \ldots, a-1\}$ and $\psi^m \in \Psi^m$, if $\rho, j \models \psi^m$ and $\rho, j' \not\models \psi^m$ for all $j' < j$ with $d(j', j) < 1$, then there exists i in ρ such that $d(i, j) = 1$ (unless $d(1, j) < 1$).

As before, we can use $\vartheta^{\mathbf{F}}$ (trivially modified so that the conjunction ranges over $m \in \{1, \ldots, a-1\}$) to enforce the second condition. For the first condition we assert the formula

$$\varphi^{\mathbf{C}} = \neg\Big(\overline{\mathbf{F}}\left(\mathbf{X}_{>0}(\neg\psi) \wedge \neg\, \mathbf{C}^k_{[0,1]}\,\psi \wedge \mathbf{X}\,\mathbf{C}^k_{[0,1)}\,\psi\right) \vee \overline{\mathbf{F}}(\mathbf{X}_{>0}\,\psi \wedge \neg\, \mathbf{C}^k_{[0,1]}\,\psi \wedge \mathbf{X}\,\mathbf{C}^{k-1}_{[0,1)}\,\psi)\Big)\,.$$

Lemma 3. *$\rho, 1 \models \varphi^{\mathbf{C}}$ iff the first condition above holds.*

Proof. Assume that the first condition is violated and there are two adjacent positions $x, x' < j$ such that $d(x, j) > 1$ and $d(x', j) < 1$. Consider the following cases:

- $\rho, x' \not\models \psi$: It is clear that $\rho, x' \models \mathbf{C}^k_{[0,1)}\,\psi$, since the covered period may contain positions $j' > j$ with $d(j, j') > 0$, and excluding x' makes no difference. It is also clear that $\rho, x \not\models \mathbf{C}^k_{[0,1]}\,\psi$ as the covered period may only contain fewer positions. We thus have $\rho, i \not\models \varphi^{\mathbf{C}}$.
- $\rho, x' \models \psi$: It is clear that $\rho, x' \models \mathbf{C}^{k-1}_{[0,1)}\,\psi$ as the covered period must contain at least $k-1$ positions satisfying ψ after excluding x'. It is also clear that $\rho, x \not\models \mathbf{C}^k_{[0,1]}\,\psi$ as the covered period may only contain fewer positions. We thus have $\rho, i \not\models \varphi^{\mathbf{C}}$.

For the other direction, consider the following cases:

- $\rho, x \models \mathbf{X}_{>0}(\neg\psi) \wedge \neg\, \mathbf{C}^k_{[0,1]}\,\psi \wedge \mathbf{X}\,\mathbf{C}^k_{[0,1)}\,\psi$ for some position x: Let the next position be x'. It is clear that there is at least one position satisfying ψ in $(\tau_x + 1, \tau_{x'} + 1)$. Let j be the position such that $|\{j' \mid x < j' \le j \text{ and } \rho, j' \models \psi\}| = k$. It is clear that j satisfies the statements in the condition, but by assumption, there is no i in ρ such that $d(i, j) = 1$.
- $\rho, x \models \mathbf{X}_{>0}\,\psi \wedge \neg\, \mathbf{C}^k_{[0,1]}\,\psi \wedge \mathbf{X}\,\mathbf{C}^{k-1}_{[0,1)}\,\psi$ for some position x: Let the next position be x'. Once again it is clear that there is at least one position satisfying ψ in $(\tau_x + 1, \tau_{x'} + 1)$. The argument is identical to the previous case. □

We say that a segment $\langle h, \ell\rangle$ is a witness for $\mathbf{C}^k_{(a,a+1)}\,\psi$ at i if $h < \ell$, $\rho, h \models \psi$, $\rho, \ell \models \psi$, $|\{j \mid h \le j \le \ell \text{ and } \rho, j \models \psi\}| = k$, and both $\tau_h, \tau_\ell \in \tau_i + (a, a+1)$. Similarly as before, we can write an untimed (finite-word) $\mathsf{LTL}^{\mathsf{fut}}$ formula ψ_1 that holds at all the starting points h of all the potential witnesses (ignoring the timing requirement) for $\mathbf{C}^k_{(a,a+1)}\,\psi$—in this case, it is simply an untimed (finite-word) LTL formula that counts exactly k occurrences of ψ. Based on this, we can give an initial attempt to express $\mathbf{C}^k_{(a,a+1)}\,\psi$, similar to what we did for $\mathbf{P}^2_{(a,a+1)}\,(P, Q)$ using $\mathsf{Q2MLO}^{\mathsf{fut}}$ in Section 4:

$$\varphi_{wit} = \mathbf{F}_{(a,a+1)}\,\psi_1 \wedge \Big(\mathbf{F}_{(a-1,a)}\left((\mathbf{X}_{>0}\top \vee \mathbf{X}_{\le 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\,\psi \wedge \neg\, \mathbf{C}^k_{[0,1)}\,\psi\right) \vee \left(\mathbf{F}_{(a-1,a]}\,\psi \wedge \mathbf{G}_{(a-1,a]}(\psi \implies \mathbf{C}^k_{[0,1)}\,\psi)\right)\Big)\,.$$

Here, however, the second conjunct is more involved as we must refrain from using (aperiodic) automata modalities. We now prove some propositions about the correctness of φ_{wit}, based on the assumption that ρ satisfies **(C1)** and **(C2)**.

Proposition 2. *$\rho, i \models \mathbf{F}_{(a-1,a]}\,\psi \wedge \mathbf{G}_{(a-1,a]}(\psi \implies \mathbf{C}^k_{[0,1)}\,\psi)$ implies $\rho, i \models \mathbf{C}^k_{(a,a+1)}\,\psi$.*

Proof. Let j be the *maximal* position in ρ such that $d(i,j) \in (a-1,a]$ and $\rho, j \models \psi$. We have $\rho, j \models \mathbf{C}^k_{[0,1)}\psi$, and it is clear that $\tau_i + (a,a+1)$ contains at least k positions satisfying ψ. □

Proposition 3. *$\rho, i \models \mathbf{C}^k_{(a,a+1)}\psi \wedge \neg \mathbf{F}_{(a-1,a]}\psi$ implies that $\rho, i \models \mathbf{F}_{(a-1,a)}\left((\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi\right)$.*

Proof. Let j_1 be the minimal position in ρ such that $d(i,j_1) \in (a,a+1)$ and $\rho, j_1 \models \psi$, and j_k be the position in ρ such that $\sigma_{j_1} \ldots \sigma_{j_k} \models \psi_1$ and $\rho, j_k \models \psi$. By Lemma 3, the first condition above holds and there is a position j' such that $d(j', j_k) = 1$ and $\rho, j' \models (\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi$. □

Proposition 4. *$\rho, i \models \mathbf{C}^k_{(a,a+1)}\psi \wedge \mathbf{F}_{(a-1,a]}\psi$ implies that $\rho, i \models \mathbf{F}_{(a-1,a)}\left((\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi\right)$ or $\rho, i \models \mathbf{F}_{(a-1,a]}\psi \wedge \mathbf{G}_{(a-1,a]}(\psi \implies \mathbf{C}^k_{[0,1)}\psi)$.*

Proof. Let j_1 be the minimal position in ρ such that $d(i,j_1) \in (a,a+1)$ and $\rho, j_1 \models \psi$, and j_k is the minimal position in ρ such that there exists $j_1 < \cdots < j_k$ where $\rho, j_i \models \psi$ for all $i \in \{1, \ldots, k\}$. Let ℓ be the *maximal* position in ρ such that $d(i,\ell) \in (a-1,a]$ and $\rho, \ell \models \psi$. Consider the following cases:

- $d(\ell, j_k) \geq 1$: By Lemma 3, there exists a position $\ell' \geq \ell$ in ρ such that $d(\ell', j_k) = 1$ and $\rho, \ell' \models (\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi$.
- $d(\ell, j_k) < 1$: We have $\rho, \ell \models \mathbf{C}^k_{[0,1)}\psi$. Now consider j_{k-1} and the largest position $\ell' < \ell$ such that $d(i,\ell') \in (a-1,a]$ and $\rho, \ell' \models \psi$. If $d(\ell', j_{k-1}) < 1$ then clearly $\rho, \ell' \models \mathbf{C}^k_{[0,1)}\psi$. If $d(\ell', j_{k-1}) \geq 1$, then by Lemma 3, there exists $\ell'' \geq \ell'$ such that $d(i,\ell'') \in (a-1,a]$, $d(\ell'', j_{k-1}) = 1$, and $\rho, \ell'' \models (\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi$. The argument is repeated until some position in $\tau_i + (a-1,a]$ satisfies $(\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi$, or all positions in $\tau_i + (a-1,a]$ satisfying ψ also satisfy $\mathbf{C}^k_{[0,1)}\psi$. □

It remains to strengthen $\mathbf{F}_{(a,a+1)}\psi_1 \wedge \mathbf{F}_{(a-1,a)}\left((\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi\right)$ so that it implies $\mathbf{C}^k_{(a,a+1)}\psi$. As before in Section 4, we need a formula ψ_2^k that refers to two neighbouring potential witnesses—in this case, it is simply an untimed (finite-word) $\mathsf{LTL}^{\mathsf{fut}}$ formula that counts exactly $k+1$ occurrences of ψ. We can then argue that there is an upper bound on the number of positions satisfying $\psi_2^{k,\geq 1}$ (easily expressible in $\mathsf{TLC}^{\mathsf{fut}}$) before $\tau_i + a$. In contrast to Section 4, however, we need an alternative way to express $\mathcal{B}^k_{\geq a+1}$.

Lemma 4. *For any $\rho = (\sigma_i, \tau_i)_{i\geq 1}$ over Σ_{AP}, the $\mathsf{TLC}^{\mathit{fut}}$ formula $\psi_2^{k,\geq 1}$ over AP is satisfied by at most $k \cdot a$ positions $j > i$ with $d(i,j) \in [0,a)$ for any $i \geq 0$.*

Lemma 5. *For any $\rho = (\sigma_i, \tau_i)_{i\geq 1}$ over Σ_{AP}, the $\mathsf{TLC}^{\mathit{fut}}$ formula $(\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi$ over AP is satisfied by at most $k \cdot (a+1) + 1$ positions $j > i$ with $d(i,j) \in [0,a]$ for any $i \geq 0$.*

Proof sketch. Each occurrence j of $(\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi$, except for possibly the first one, happens 'between' two neighbouring (minimal) potential witnesses $\langle h_i, \ell_i\rangle$ and $\langle h_{i+1}, \ell_{i+1}\rangle$ with $d(h_i, \ell_{i+1}) \geq 1$: either $j = h_i$ or $h_i < j < h_{i+1}$. The claim holds by (a trivial modification of) Lemma 4. □

Now suppose that $\rho, i \models \mathbf{F}_{(a,a+1)}\psi_1 \wedge \mathbf{F}_{(a-1,a)}\left((\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi\right)$ and let j be the maximal position in $\tau_i + (a-1,a)$ such that $\rho, j \models (\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi$. The undesired scenario is when there is a maximal $j' > j$ with $\tau_{j'} \in \tau_i + (a-1,a]$ such that $\rho, j' \models \psi$, and there are less than k positions in $\tau_i + (a,a+1)$ satisfying ψ. In this case, it is clear that $\rho, j' \models \psi_2^{k,\geq 1}$. To

rule this scenario out we employ the following strategy, which can be implemented as a $\mathsf{TLC}^{\mathsf{fut}}$ formula (which we opt to explain in English, for the sake of readability; we count events at positions $> i$):

(1) Count the number of occurrences of $\psi_2^{k,\geq 1}$ in $\tau_i + [0,a)$ and $\tau_i + [0,a]$. If they do not match, then we are in the undesired scenario.

(2) Count the number of occurrences of $(\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi$ in $\tau_i + [0,a)$.

(3) Take a disjunction over all the possible ways in which these occurrences may interleave in $\tau_i + [0,a)$ (note that they may hold simultaneously on the same position). Those ending with $\psi_2^{k,\geq 1}$ but not $(\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi$ are in the undesired scenario.

Let us call this formula (which captures the undesired scenario) φ'_{out}.

Proposition 5. $\mathbf{F}_{(a,a+1)}\psi_1 \wedge \mathbf{F}_{(a-1,a)}\left((\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi\right) \wedge \neg\varphi'_{out}$ *implies* $\rho, i \models \mathbf{C}^k_{(a,a+1)}\psi$.

Proof. Consider the conditions above that form φ'_{out}. First note that if the number of occurrences of $\psi_2^{k,\geq 1}$ in $\tau_i + [0,a]$ is 0, then it is easy to see that $\rho, i \models \mathbf{C}^k_{(a,a+1)}\psi$. To see (1), note that if $\psi_2^{k,\geq 1}$ holds at some position j' with $\tau_{j'} = \tau_i + a$, then $\tau_i + (a, a+1)$ may contain at most $k-1$ positions satisfying ψ. So for (3), first assume that $\psi_2^{k,\geq 1}$ does not hold at any position at $\tau_i + a$. Let j be the maximal position with $\tau_j \in \tau_i + (a-1, a)$ such that $\rho, j \models (\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi$ and $j' > i$ be the maximal position with $\tau_{j'} \in \tau_i + [0,a)$ and $\rho, j' \models \psi_2^{k,\geq 1}$.

If ψ holds at some maximal position ℓ at $\tau_i + a$, since $\rho, i \models \mathbf{F}_{(a,a+1)}\psi_1$, we have $\rho, \ell \models \psi_2^{k,<1}$ (defined in the expected way) and thus $\rho, i \models \mathbf{C}^k_{(a,a+1)}\psi$; we argue that $\psi_2^{k,\geq 1}$ cannot hold at any ℓ' where $j < \ell' < \ell$. Suppose to the contrary that $\rho, \ell' \models \psi_2^{k,\geq 1}$ (Wlog. let ℓ' be the largest such position at the same timestamp $\tau_{\ell'}$). If $\rho, \ell' \models \psi_2^{k,=1}$, we have $\rho, \ell' \models (\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi$, contradicting the maximality of j. If $\rho, \ell' \models \psi_2^{k,>1}$ then by Lemma 3, there exists a position $\ell'' > \ell'$ such that $\rho, \ell'' \models (\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi$, again contradicting the maximality of j. Now we assume that ψ does not hold at any position at $\tau_i + a$. Consider the following cases:

- $j' = j$: We know that $\rho, j \models \psi$. Consider the following subcases:
  - j' is not the maximal position with $\tau_{j'} \in \tau_i + (a-1, a)$ such that $\rho, j' \models \psi$: There is a maximal j'' with $\tau_{j''} \in \tau_i + (a-1, a)$ and $\rho, j'' \models \psi$. Since $\rho, i \models \mathbf{F}_{(a,a+1)}\psi_1$ and thus $\rho, j'' \models \psi_2^{k,<1}$, it follows that $\rho, i \models \mathbf{C}^k_{(a,a+1)}\psi$.
  - j' is the maximal position with $\tau_{j'} \in \tau_i + (a-1, a)$ such that $\rho, j' \models \psi$: Since $\rho, j' \models \mathbf{C}^k_{[0,1]}$ but there is no $j'' > j$ with $\tau_{j''} \in \tau_i + (a-1, a]$ such that $\rho, j'' \models \psi$, we have $\rho, i \models \mathbf{C}^k_{(a,a+1)}\psi$.
- $j' < j$: If there is a maximal $j'' > j$ with $\tau_{j''} \in \tau_i + (a-1, a)$ and $\rho, j'' \models \psi$, Since $\rho, i \models \mathbf{F}_{(a,a+1)}\psi_1$ we have $\rho, j'' \models \psi_2^{k,<1}$, and it follows that $\rho, i \models \mathbf{C}^k_{(a,a+1)}\psi$. If there is no such j'', since $\rho, j \models \mathbf{C}^k_{[0,1]}$ we also have $\rho, i \models \mathbf{C}^k_{(a,a+1)}\psi$. □

Our final formula for $\mathbf{C}^k_{(a,a+1)}\psi$ is

$$\varphi'_{wit} = \mathbf{F}_{(a,a+1)}\psi_1 \wedge \Big(\Big(\mathbf{F}_{(a-1,a)}\left((\mathbf{X}_{>0}\top \vee \mathbf{X}_{\leq 0}\psi) \wedge \mathbf{C}^k_{[0,1]}\psi \wedge \neg \mathbf{C}^k_{[0,1)}\psi\right) \wedge \neg\varphi'_{out}\Big) \vee \left(\mathbf{F}_{(a-1,a]}\psi \wedge \mathbf{G}_{(a-1,a]}(\psi \implies \mathbf{C}^k_{[0,1)}\psi)\right)\Big).$$

To see that $\mathbf{C}^k_{(a,a+1)}\,\psi$ implies φ'_{wit}, observe that Propositions 3 and 4 still hold if the conjunct $\neg\varphi'_{wit}$ is added. We apply the equivalence repeatedly from the innermost subformula $\mathbf{C}^k_I\,\psi$ where ψ is in $\mathsf{TLC}^{\mathsf{fut}}$, and then work outwards until there is no $\mathbf{C}^k_I\,\psi$ with $I = \langle a, b\rangle$. In the process, we also need to ensure that **(C1)** and **(C2)** are satisfied for various ψ. Finally, we rewrite $\mathbf{F}_I$ with $I = \langle a, b\rangle$ into $\mathbf{F}_I$ with $I = [0, b\rangle$.

Theorem 4. *Given a* TLCI^{fut} *formula* φ*, there is a* TLC^{fut} *formula* φ' *such that* φ *and* φ' *are equivalent over timed words satisfying* **(C1)** *and* **(C2)** *(for some finite set of* ψ*).*

Finally, this result carries over to the case of the continuous interpretations of TLCI, as the 'positions' postulated by **(C1)** and **(C2)** automatically exist.

Corollary 1. $\mathsf{TLCI}^{\mathsf{fut}} \subseteq \mathsf{TLC}^{\mathsf{fut}}$ *in the continuous semantics.*

7 Conclusion and future work

It turned out that allowing $\langle a, b\rangle$ in counting modalities only makes them more intricate to express in (aperiodic) automata modalities (or Q2MLO), which necessarily 'start' from the current point; in other words, the relevant claims in [38] are indeed correct. More generally, we have shown that the existence of two 'witnesses' $x' \leq x''$ for a first-order formula $\varphi(x', x'')$ in $t_0 + \langle a, b\rangle$ can also be captured in aperiodic $\mathsf{EMITL}^{\mathsf{fut}}$ (or $\mathsf{Q2MLO}^{\mathsf{fut}}$). This is somewhat surprising, as the timing constraints on both x' and x'' does seem to require the use of punctualities or non-trivial extensions. Our second main result gives a satisfactory correction to the folklore belief, at least in the case of continuous semantics. We list below some possible further directions:

- MITL with both the future and past modalities and rational constants appears to be very expressive with EXPSPACE-complete satisfiability and model-checking problems (through a simple scaling argument). We also know from Theorem 2 and [23] that it can be made expressively complete for $\mathsf{FO}[<, +1]$ by adding punctualities in the continuous semantics. Can it express some decidable fragments of 1-TPTL[**U**, **S**] [26] with rational constants (i.e. without using automata modalities)?
- The properties considered in Section 5 can be seen as a special case of a decidable fragment of the logic PnEMTL recently proposed in [26]. Can we extend the ideas presented here to handle more general PnEMTL properties, where automata modalities do not start from the current point?
- Can the construction in Section 6 lead to a future (or 'almost future' [36]) metric temporal logic that is expressively complete for $\mathsf{Q2MLO}^{\mathsf{fut}}$, or more generally a separation result akin to [13] or [24]?

References

[1] Rajeev Alur & David L. Dill (1994): *A Theory of Timed Automata*. *Theoretical Computer Science* 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[2] Rajeev Alur, Tomás Feder & Thomas A. Henzinger (1996): *The Benefits of Relaxing Punctuality*. *Journal of the ACM* 43(1), pp. 116–146, doi:10.1145/227595.227602.

[3] Rajeev Alur & Thomas A. Henzinger (1992): *Back to the Future: Towards a Theory of Timed Regular Languages*. In: *FOCS*, IEEE Computer Society, pp. 177–186, doi:10.1109/SFCS.1992.267774.

[4] Rajeev Alur & Thomas A. Henzinger (1993): *Real-Time Logics: Complexity and Expressiveness*. *Information and Computation* 104(1), pp. 35–77, doi:10.1006/inco.1993.1025.

[5] Rajeev Alur & Thomas A. Henzinger (1994): *A Really Temporal Logic*. *Journal of the ACM* 41(1), pp. 164–169, doi:10.1145/174644.174651.

[6] A. Baziramwabo, P. McKenzie & D. Therien (1999): *Modular temporal logic*. In: *LICS*, IEEE Computer Society, pp. 344–351, doi:10.1109/LICS.1999.782629.

[7] Bartosz Bednarczyk & Witold Charatonik (2017): *Modulo Counting on Words and Trees*. In: *FSTTCS*, *LIPIcs* 93, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 12:1–12:16, doi:10.4230/LIPIcs.FSTTCS.2017.12.

[8] Luigi Bellomarini, Markus Nissl & Emanuel Sallinger (2021): *Monotonic Aggregation for Temporal Datalog*. In: *RuleML+RR*, *CEUR Workshop Proceedings* 2956, CEUR-WS.org.

[9] Patricia Bouyer, Fabrice Chevalier & Nicolas Markey (2010): *On the expressiveness of TPTL and MTL*. *Information and Computation* 208(2), pp. 97–116, doi:10.1016/J.IC.2009.10.004.

[10] Patricia Bouyer, François Laroussinie, Nicolas Markey, Joël Ouaknine & James Worrell (2017): *Timed Temporal Logics*. In: *Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His* 60*th Birthday*, *LNCS* 10460, Springer, pp. 211–230, doi:10.1007/978-3-319-63121-9_11.

[11] Deepak D'Souza & Pavithra Prabhakar (2007): *On the expressiveness of MTL in the pointwise and continuous semantics*. *International Journal on Software Tools for Technology* 9(1), pp. 1–4, doi:10.1007/S10009-005-0214-9.

[12] Carlo A. Furia & Matteo Rossi (2008): *MTL with Bounded Variability: Decidability and Complexity*. Technical Report 2008.10, Dipartimento di Elettronica e Informazione, Politecnico di Milano, doi:10.1007/978-3-540-85778-5_9.

[13] Dov Gabbay, Amir Pnueli, Sharanon Shelah & J. Stavi (1980): *On the Temporal Analysis of Fairness*. In: *POPL*, ACM Press, pp. 163–173, doi:10.1145/567446.567462.

[14] Thomas A. Henzinger, Jean-François Raskin & Pierre-Yves Schobbens (1998): *The Regular Real-Time Languages*. In: *ICALP*, *LNCS* 1443, Springer, pp. 580–591, doi:10.1007/BFB0055086.

[15] Yoram Hirshfeld & Alexander Rabinovich (1999): *A framework for decidable metrical logics*. In: *ICALP*, *LNCS* 1644, Springer, pp. 422–432, doi:10.1007/3-540-48523-6_39.

[16] Yoram Hirshfeld & Alexander Rabinovich (1999): *Quantitative Temporal Logic*. In: *CSL*, *LNCS* 1683, Springer, pp. 172–187, doi:10.1007/3-540-48168-0_13.

[17] Yoram Hirshfeld & Alexander Rabinovich (2004): *Logics for Real Time: Decidability and Complexity*. *Fundamenta Informaticae* 62(1), pp. 1–28.

[18] Yoram Hirshfeld & Alexander Rabinovich (2006): *An Expressive Temporal Logic for Real Time*. In: *MFCS*, *LNCS* 4162, Springer, pp. 492–504, doi:10.1007/11821069_43.

[19] Yoram Hirshfeld & Alexander Rabinovich (2008): *Decidable metric logics*. *Information and Computation* 206(12), pp. 1425–1442, doi:10.1016/J.IC.2008.08.004.

[20] Yoram Hirshfeld & Alexander Rabinovich (2012): *Continuous time temporal logic with counting*. *Information and Computation* 214, pp. 1–9, doi:10.1016/J.IC.2011.11.003.

[21] Hsi-Ming Ho (2019): *Revisiting timed logics with automata modalities*. In: *HSCC*, ACM, pp. 67–76, doi:10.1145/3302504.3311818.

[22] Hsi-Ming Ho & Khushraj Madnani (2023): *More Than 0s and 1s: Metric Quantifiers and Counting over Timed Words*. In: *TIME*, *LIPIcs* 278, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 7:1–7:15, doi:10.4230/LIPICS.TIME.2023.7.

[23] Paul Hunter (2013): *When is Metric Temporal Logic Expressively Complete?* In: *CSL*, *LIPIcs* 23, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 380–394, doi:10.4230/LIPICS.CSL.2013.380.

[24] Paul Hunter, Joël Ouaknine & James Worrell (2013): *Expressive Completeness for Metric Temporal Logic*. In: *LICS*, IEEE Computer Society, pp. 349–357, doi:10.1109/LICS.2013.41.

[25] Ron Koymans (1990): *Specifying Real-time Properties with Metric Temporal Logic*. *Real-Time Systems* 2(4), pp. 255–299, doi:10.1007/BF01995674.

[26] Shankara Narayanan Krishna, Khushraj Madnani, Manuel Mazo Jr. & Paritosh Pandya (2022): *From Non-Punctuality to Non-Adjacency: A Quest for Decidability of Timed Temporal Logics with Quantifiers. Formal Aspects of Computing*, doi:10.1145/3571749.

[27] Shankara Narayanan Krishna, Khushraj Madnani & Paritosh K. Pandya (2016): *Metric Temporal Logic with Counting*. In: *FoSSaCS*, *LNCS* 9634, Springer, pp. 335–352, doi:10.1007/978-3-662-49630-5_20.

[28] Shankara Narayanan Krishna, Khushraj Madnani & Paritosh K. Pandya (2018): *Logics Meet 1-Clock Alternating Timed Automata*. In: *CONCUR*, *LIPIcs* 118, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 39:1–39:17, doi:10.4230/LIPICS.CONCUR.2018.39.

[29] François Laroussinie, Antoine Meyer & Eudes Petonnet (2010): *Counting LTL*. In: *TIME*, IEEE Computer Society, pp. 51–58, doi:10.1109/TIME.2010.20.

[30] Kamal Lodaya & A. V. Sreejith (2010): *LTL Can Be More Succinct*. In: *ATVA*, *LNCS* 6252, Springer, pp. 245–258, doi:10.1007/978-3-642-15643-4_19.

[31] Robert McNaughton & Seymour Papert (1971): *Counter-free automata*. The MIT Press.

[32] Joël Ouaknine, Alexander Rabinovich & James Worrell (2009): *Time-Bounded Verification*. In: *CONCUR*, *LNCS* 5710, Springer, pp. 496–510, doi:10.1007/978-3-642-04081-8_33.

[33] Joël Ouaknine & James Worrell (2006): *On Metric Temporal Logic and Faulty Turing Machines*. In: *FoSSaCS*, *LNCS* 3921, Springer, pp. 217–230, doi:10.1007/11690634_15.

[34] Joël Ouaknine & James Worrell (2007): *On the Decidability and Complexity of Metric Temporal Logic over Finite Words*. *Logical Methods in Computer Science* 3(1), doi:10.2168/LMCS-3(1:8)2007.

[35] Paritosh K. Pandya & Simoni S. Shah (2011): *On Expressive Powers of Timed Logics: Comparing Boundedness, Non-punctuality, and Deterministic Freezing*. In: *CONCUR*, *LNCS* 6901, Springer, pp. 60–75, doi:10.1007/978-3-642-23217-6_5.

[36] Dorit Pardo & Alexander Rabinovich (2016): *No Future without (a hint of) Past: A Finite Basis for 'Almost Future' Temporal Logic*. *Information and Computation* 247, pp. 203–216, doi:10.1016/j.ic.2016.01.002.

[37] Amir Pnueli (1977): *The temporal logic of programs*. In: *FOCS*, IEEE Computer Society, pp. 46–57, doi:10.1109/SFCS.1977.32.

[38] Alexander Rabinovich (2010): *Complexity of metric temporal logics with counting and the Pnueli modalities*. *Theoretical Computer Science* 411(22–24), pp. 2331–2342, doi:10.1016/J.TCS.2010.03.017.

[39] Jean-François Raskin (1999): *Logics, automata and classical theories for deciding real time*. Ph.D. thesis, FUNDP (Belgium).

[40] Marcel-Paul Schützenberger (1965): *On finite monoids having only trivial subgroups*. *Information and Control* 8, pp. 190–194, doi:10.1016/S0019-9958(65)90108-7.

[41] Saharon Shelah (1975): *The Monadic Theory of Order*. *The Annals of Mathematics* 102(3), p. 379, doi:10.2307/1971037.

[42] Thomas Wilke (1994): *Specifying timed state sequences in powerful decidable logics and timed automata*. In: *FTRTFT*, *LNCS* 863, Springer, pp. 694–715, doi:10.1007/3-540-58468-4_191.

Conditional Nested Pattern Matching in Interaction Nets

Shinya Sato
Institute for Liberal Arts Education
Ibaraki University, Ibaraki, Japan

Interaction nets are a form of restricted graph rewrite system that can serve as a graphical or textual programming language. As such, benefits include one-step confluence, ease of parallelism and explicit garbage collection. However, some of these restrictions burden the programmer, so they have been extended in several ways, notably to include data types and conditional rules. This paper introduces a further extension to allow nested pattern matching and to do so in a way that preserves these benefits and fundamental properties of interaction nets. We also show that by introducing a translation to non-nested matching, this extension is conservative in rewriting. In addition, we propose a new notation to express this pattern matching.

1 Introduction

Interaction nets, proposed by Lafont [6], can be considered as an execution model of programming languages, where programs are described as graphs and computation is realised by graph reduction. They have been used for optimal reduction [7, 3] and other efficient implementations [8] of λ-calculus, the basis of functional programming languages. Indeed, as a programming language, interaction nets have several attractive features:

- A simple graph rewriting semantics;
- A complete symmetry between constructors and destructors;
- They are Turing complete;
- Reduction steps are local, lending them to parallel execution without amending the algorithm being executed;
- Memory management, including garbage collection, is explicitly part of the execution, improving speed and, again, not requiring any changes in a parallel environment.

When writing programs in interaction nets, it is useful to have some extensions to the basic net structure, to facilitate the process. It is analogous to PCF [11], which is an extension of the pure typed λ-calculus obtained by adding some constants. For example, "pure" interaction nets do not have built-in constants, data types or conditional branching. Data types, such as integers, were introduced in [2], and conditional rewriting rules on values were introduced in [12].

Although net reduction rules have a basic, depth-one pattern matching, a nested version has been introduced in [4] as a conservative extension, *i.e.* although it provides a new feature to the programmer, it can be implemented using pure interaction nets and thus retains fundamental properties of interaction nets such as the one-step confluence property (defined in Section 2.1).

The aim of this paper is to introduce conditional nested pattern matching on values, as a further extension of [4], whilst preserving one-step confluence. We show that this extension is conservative by introducing a translation that maps the nested pattern matching back to pure nets. We also propose a new

S. Alves and I. Mackie (Eds.): 13th International Workshop
on Developments in Computational Models 2023 (DCM'23).
EPTCS 408, 2024, pp. 90–106, doi:10.4204/EPTCS.408.6

notation for the pattern matching. This notation is similar to the well-known case expressions in many programming languages.

One of the goals of this work is to show how to represent functional programs (and more generally term rewriting systems) that contain nested pattern matching in interaction nets. This is part of ongoing work to demonstrate a real-world functional programming language that can take full advantage of interaction nets' built-in parallelism.

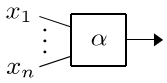
This paper is structured as follows. In the next section we give a overview of interaction nets and the extension for values. In Section 3 we introduce the conditional nested pattern matching. Section 4 introduces a translation that removes the nested pattern matching and shows that this extended matching is a conservative extension. Then Section 5 proposes a notation to easily express the matching and Section 6 discusses some implementation issues. The paper concludes in Section 7 with an outlook for the future.

2 Background

In this section we review the interaction net paradigm, and describe some known extensions to it.

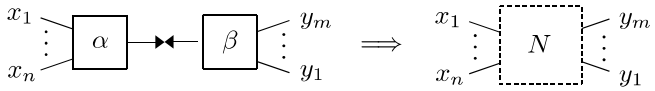
2.1 Interaction nets

Interaction nets are graph rewriting systems [6]. Each node has a name α and one or more *ports* which can be connected to ports of other nodes. The number of ports is determined by the node name α and is called the node's *arity*, written as $\mathsf{ar}(\alpha)$. When $\mathsf{ar}(\alpha) = n$, then the node α has $n+1$ ports: n *auxiliary* ports and a distinguished one called the *principal* port, labelled with an arrow. Nodes are drawn as follows:



We may put different labels on the ports to distinguish them.

We have a set Σ of *symbols*, which are the names of nodes. A *net* built on Σ is an undirected graph: the vertices are nodes, and the edges connect nodes at the ports. There is at most one edge at every port. A port which is not connected is called a *free* port. A pair of nodes $(\alpha,\beta) \in \Sigma \times \Sigma$ connected via their principal ports forms an *active* pair, which is the interaction nets analogue of a redex pattern. We refer to such a connected pair (α,β) as $\alpha \bowtie \beta$. A rule $((\alpha,\beta) \Rightarrow N)$ describes how to replace the pair (α,β) with the net N. N can be any net as long as the set of free ports are preserved during the reduction. The following diagram illustrates the idea:



There is at most one rule for each pair of nodes, and the pairs are matched symmetrically, that is to say, (β,α) is also replaced by N by a rule $((\alpha,\beta) \Rightarrow N)$. If the pair is symmetric, such as (α,α), then N must be symmetric. We write $N_1 \to N_2$ when a net N_1 is reduced to N_2 by a rule. One of the most powerful properties of this graph rewriting system is that it is one-step confluent; if $N \to N_1$ and $N \to N_2$ $(N_1 \neq N_2)$, then there exists a net N' such that $N_1 \to N'$ and $N_2 \to N'$. Therefore, if a normal form exists, it is uniquely determined and any reduction path to the normal form has the same number of steps.

Here, as shown in Figure 1, we give an example of the interaction net system for unary number multiplication, represented by the following term rewriting system:

- $\texttt{Mult}(\texttt{Z}, y) = y$
- $\texttt{Mult}(\texttt{S}(x), y) = \texttt{Add}(y, \texttt{Mult}(x, y))$

This system has nodes Z, S, Add and Mult for arithmetic expressions and δ, ε for duplication and elimination where the α in the δ and ε rules stands for either Z or S. This rewrites $\texttt{Mult}(\texttt{S}(\texttt{S}(\texttt{Z})), \texttt{S}(\texttt{S}(\texttt{S}(\texttt{Z}))))$ to $\texttt{Add}(\texttt{S}(\texttt{S}(\texttt{S}(\texttt{Z}))), \texttt{Add}(\texttt{S}(\texttt{S}(\texttt{S}(\texttt{Z}))), \texttt{Z}))$. Whilst in term rewriting systems $\texttt{Mult}(\texttt{S}(x), A)$ can be rewritten as $\texttt{Add}(A, \texttt{Mult}(x, A))$ even if the A has redexes, in this system the duplications are performed only for nets that have no active pairs, *i.e.* nets that are built from S and Z. We use δ to explicitly duplicate terms and ε to remove them when not needed in the result—an example of the explicit memory management. In addition, evaluated results in the interaction net can be quickly used for other computations without waiting for the whole computation to finish, *i.e.* as a *wait-free* algorithm. This is called a *stream operation* [9], and Figure 1 shows that in the second rewrite step, the S duplicated by $\delta \bowtie \texttt{S}$ can be used with Add.

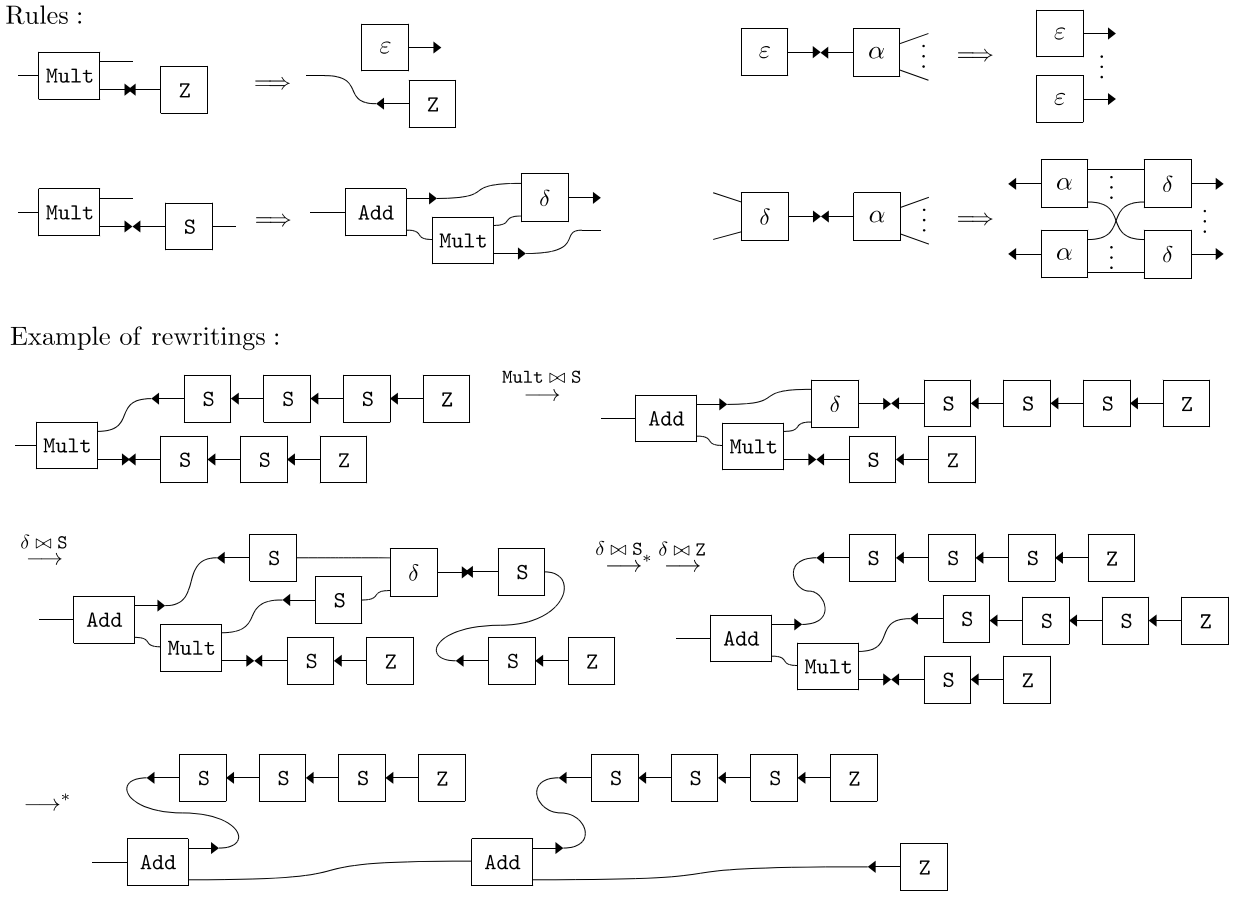
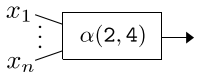


Figure 1: An example of rules and rewritings of interaction nets

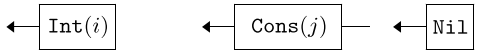
We call the system *pure* to mean that no extensions are applied. From here, we show some extensions to the pure interaction nets system.

2.2 Attributes

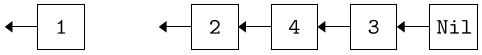
Nodes can be extended to contain additional information [2, 12], and here we review an extension called *attributes* [12]. This extension is considered as an analogue of PCF obtained by adding some constants to the pure typed λ-calculus. Attribute values are integers and are written in parentheses following agent symbols. For instance, $\alpha(2,4)$ is a node which holds 2 and 4 as attribute values, and is drawn as follows:



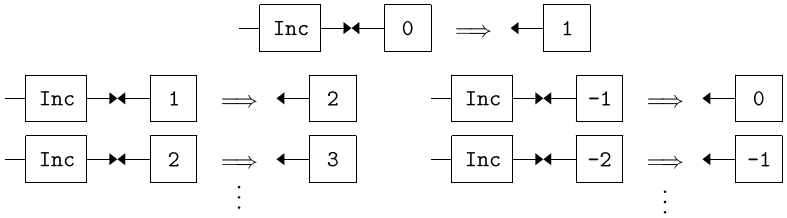
Integers and their lists are represented using built-in nodes $\texttt{Int}(i)$, $\texttt{Cons}(j)$ and `Nil` where i, j are integer numbers, and are drawn as follows:



To simplify the diagram, the nodes $\texttt{Int}(i)$ and $\texttt{Cons}(j)$ are often drawn without the symbols when no confusion will happen. For instance, an integer `1` and a list of `2,4,3` are written in the following way:



Rewriting rules are also extended to deal with attributes by considering a node name with attribute values as one symbol. For instance, the name of the node $\alpha(2,4)$, which holds attribute values `2` and `4`, is considered as a symbol "$\alpha(2,4)$". We soon see that we need to add extra power to this system. For example, the increment operator for integer numbers can be defined by the following rules between a node `Inc` and each integer number:



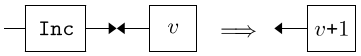
This demonstrates an obvious problem in practical using—if we want to define the increment for any integer, we need not only an infinite set of symbols such as $\texttt{Int}(0)$, $\texttt{Int}(1)$, $\texttt{Int}(-1)$, $\texttt{Int}(2)$, ..., but also an infinite number of rules. To deal with these in finite schemes, we introduce meta variables v for them, called *attribute variables* and *expressions* e on attribute variables as follows:

$e ::= v \mid i \mid -e \mid \texttt{not}\ e \mid e\ \ op\ \ e \mid (e)$

where v is an attribute variable, i is an integer number, e is an expression, and op is defined as follows:

$op ::= \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \texttt{mod} \mid \texttt{==} \mid \texttt{!=} \mid \texttt{<} \mid \texttt{<=} \mid \texttt{>} \mid \texttt{=>} \mid \texttt{and} \mid \texttt{or}.$

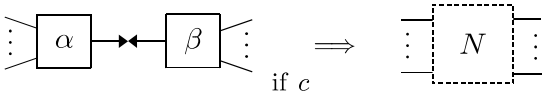
The definition of expressions may be extended as long as the computation is decidable. We will abbreviate in the following the left- and right-hand sides of a rule by LHS and RHS, respectively. Attribute variables can be placed on the LHS nodes, expressions on the variables can be placed on the RHS nodes, just like attributes. The LHS nodes do not have the same attribute variables to ensure that each variable matches any attribute value; for example, $\alpha(i,i)$ is not allowed as the name of the LHS node because it has the same variable as i, whereas $\alpha(i,j)$ is allowed. Now the increment operator is represented as the following one rule, in which an attribute value v is replaced by a value obtained by executing v+1:



Addition of integers on two node attributes is represented as the following two rules:

2.2.1 Conditional rules on attributes

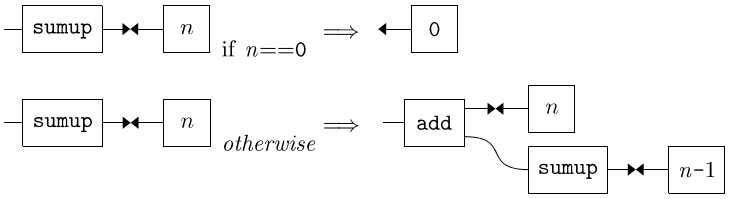
Conditional rules were proposed in [12], and here we introduce a refined version. We write conditional rules as $((\alpha,\beta)$ if $c \Rightarrow N)$, where the c is an expression on attribute variables of α, β. The c is called the *conditional expression* of a rule, and the rule becomes available to be applied if c is calculated to true, where false is represented by 0 and true by any other value. We also refer to the active pair with a conditional expression as $(\alpha \bowtie \beta$ if $c)$, and call it a *conditional* active pair. The rule is drawn as follows:



There may exist several conditional rules for the same active pair, but they must be *disjoint*, *i.e.* only at most one condition among the rules must be true. This can be realised by introducing a decidable evaluation strategy for the conditions. We use `true` and `false` that are evaluated as true and false, respectively. In addition, as a refined version, we introduce a special conditional rule $((\alpha,\beta)$ if *otherwise* $\Rightarrow N)$ which becomes available if any other conditional expressions for the $\alpha \bowtie \beta$ become false. We may write "if *otherwise*" as just "*otherwise*" and omit "if `true`" when no confusion will arise.

For instance, a function `sumup` that takes a natural number and computes the sum up to the number can be realised by the following conditional rules:

```
sumup 0 = 0
sumup n = n + sumup (n-1)
```



Conditional rules are disjoint for the same active pairs, therefore the one-step confluence property is preserved.

3 Conditional Nested Pattern Matching

In this section we introduce conditional nested pattern matching on attributes, as an extension of the nested matching introduced in [4], so that we can put a condition on each nested pattern. We write NAP as an abbreviation of "nested active pair".

Attribute values can be considered as parts of symbols, so nested pattern matching can also manage attributes values. Here, we extend the NAP to conditional ones. For simplicity, we consider all rules $((\alpha,\beta) \Rightarrow N)$ in the pure system as conditional rules with the `true` expression such as $((\alpha,\beta)$ if `true` $\Rightarrow N)$.

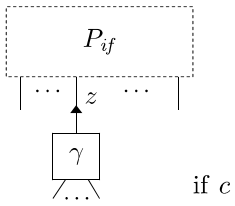
Definition 3.1 (Conditional NAPs)**.** *Conditional* NAPs $\langle P_{if} \rangle$ are inductively defined as follows:

Base: Every conditional active pair $(\alpha \bowtie \beta$ if $c)$ is a conditional NAP. We write this as:

$$\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c \rangle$$

where $\vec{x},\vec{y}$ are distinct names, corresponding to the occurrences of the auxiliary ports of α and β, respectively.

Step: We assume that $\langle P_{if} \rangle$ is a conditional NAP, γ is an agent, c is an expression on attribute variables of agents occuring in $\langle P_{if} \rangle$ and a free port in $\langle P_{if} \rangle$ has a name z. A net obtained by connecting the principal port of γ with the expression c to the free port z is also a conditional NAP.



We write this as:

$$\langle P_{if},\ z - \gamma(\vec{w}) \text{ if } c \rangle$$

where $\vec{w}$ are distinct names and fresh for $\langle P_{if} \rangle$, corresponding to the occurrences of the auxiliary ports of γ.

We call the form "$z - \gamma(\vec{w})$ if c" a *connection* in a conditional NAP. We use u to range over connections.

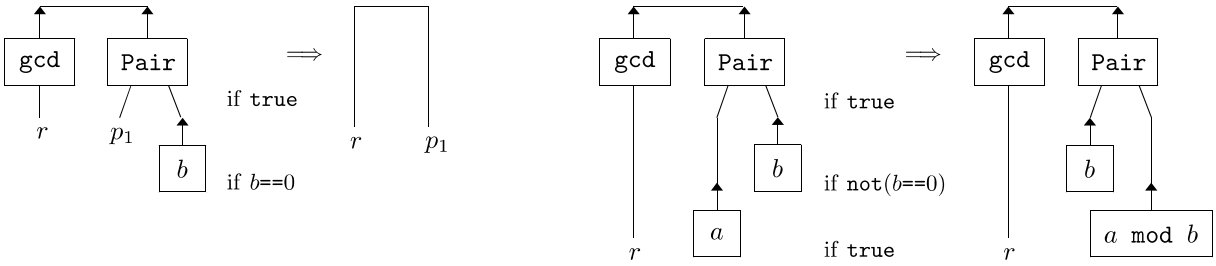
Expressions that occur as "if c" in conditional NAPs are also called *conditional* expressions. If all conditional expressions in a conditional NAP evaluate to true, the NAP is called *available*.

We build just a NAP from a conditional NAP by removing all conditional expressions. For a conditional NAP $\langle P_{if} \rangle$, we write the (non-conditional) $\langle P \rangle$ as the *condition dropped* NAP for $\langle P_{if} \rangle$. A rewriting rule on a conditional NAP $(\langle P_{if} \rangle \Rightarrow N)$ replaces a matched net by the condition dropped $\langle P \rangle$ to the net N if $\langle P_{if} \rangle$ is available. Nested nets are also symmetrically matched and replaced.

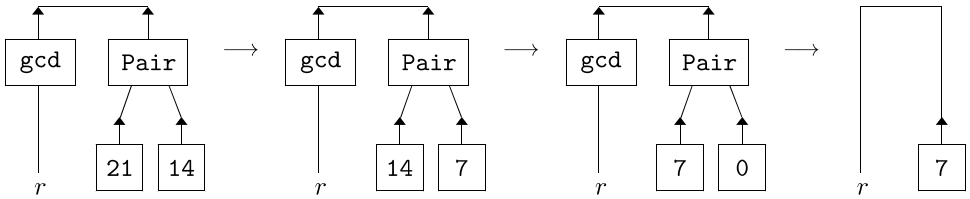
Example 3.2. We take the following program `gcd'` which obtains the greatest common divisor of two given natural numbers:

```
gcd' (a,b) = if b==0 then a
             else gcd' (b, a `mod` b)
```

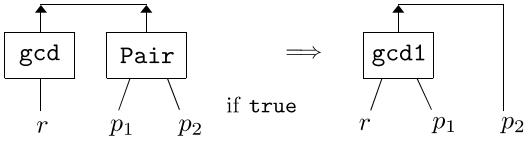
This is written as the following rewriting rules on conditional NAPs:

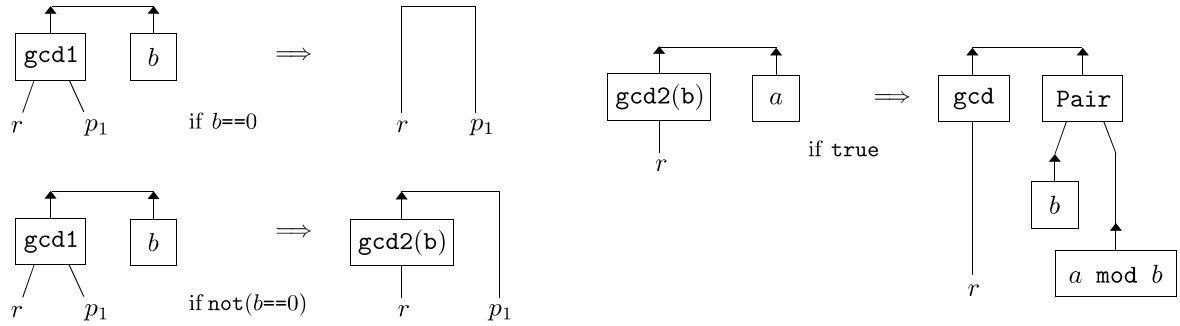


The following is an example of rewritings for `gcd'(21,14)`:

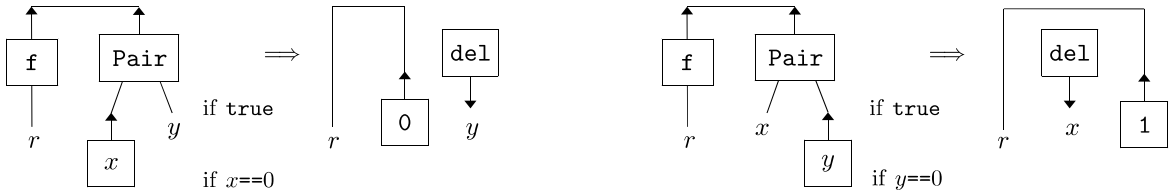


Of course, we can realise the same computation for the `gcd'` without conditional NAPs, but this requires several additional rules that obscure the overall meaning:

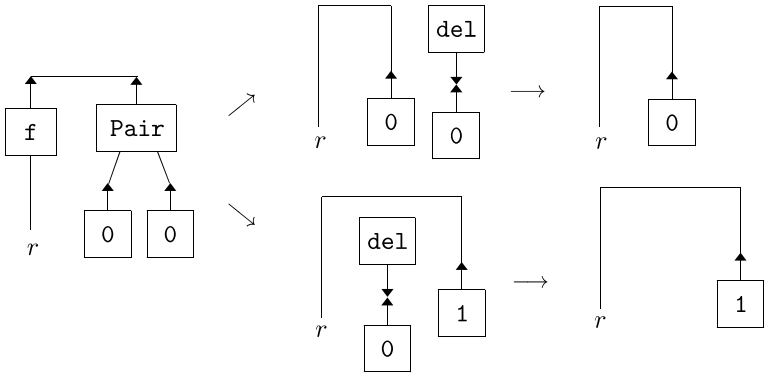




Example 3.3. Consider the following non-confluent map: `f(0,y)=0`, `f(x,0)=1`.
This map is encoded as the following rules where the agent `del` deletes built-in nodes `Int`(i):



As shown below, the two rewriting paths from `f(0,0)` are not confluent:



To preserve the one-step confluence property, it is sufficient to introduce *local sequentiality* [6], *i.e.* which port is looked first. We extend several properties in [4]. First, we instantiate the sequentiality to sets of conditional NAPs:

Definition 3.4 (Local sequentiality)**.** Let $\mathscr{P}_{if}$ be a set of conditional NAPs. $\mathscr{P}_{if}$ is *local sequential* if:

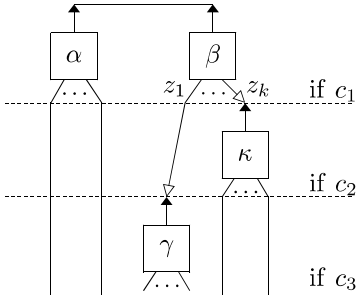
(1) If $\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c \rangle \in \mathscr{P}_{if}$:

- **(1a)** All conditional expressions c_k such that $\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c_k \rangle \in \mathscr{P}_{if}$ are disjoint, *i.e.* only at most one conditional expression must be true.

(2) If $\langle P_{if},\ z - \gamma(\vec{w}) \text{ if } c \rangle \in \mathscr{P}_{if}$:

- **(2a)** All conditional expressions c_k such that $\langle P_{if},\ z - \gamma(\vec{w}) \text{ if } c_k \rangle \in \mathscr{P}_{if}$ are disjoint,
- **(2b)** $\langle P_{if} \rangle \in \mathscr{P}_{if}$,
- **(2c)** $\langle P_{if},\ z_j - \gamma_j(\vec{w}_j) \text{ if } c_j \rangle \notin \mathscr{P}_{if}$ for any free port z_j in the $\langle P_{if} \rangle$ except the z, where γ_j is an agent, c_j is a conditional expression.

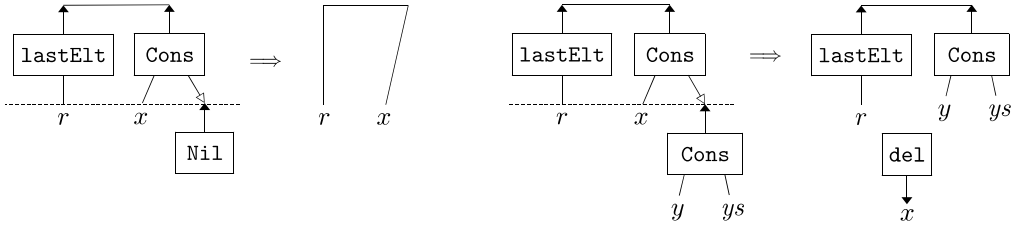
We may write "local sequential" as just "sequential" when no confusion will arise. If a local sequential set has an element $\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c,\ \vec{u},\ z - \gamma(\vec{w}) \text{ if } c \rangle$ where $\vec{u}$ is a sequence of connections, then every element started from the origin pair $\alpha \bowtie \beta$ is along the path to the z. For clarity, we draw triangles on auxiliary ports connected to agents, and horizontal dotted lines for the sequential order. As an example, we depict a NAP which is textually represented $\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c_1,\ z_k - \kappa(\vec{w}) \text{ if } c_2,\ z_1 - \gamma(\vec{z}) \text{ if } c_3 \rangle$ as follows:



Example 3.5. The following program `lastElt` returns the last element of a given non-empty list:

```
lastElt ([x]) = x
lastElt (x:y:ys) = lastElt (y:ys)
```

This is written as the following rewriting rules on (non-conditional) NAPs where `del` erases any agent:

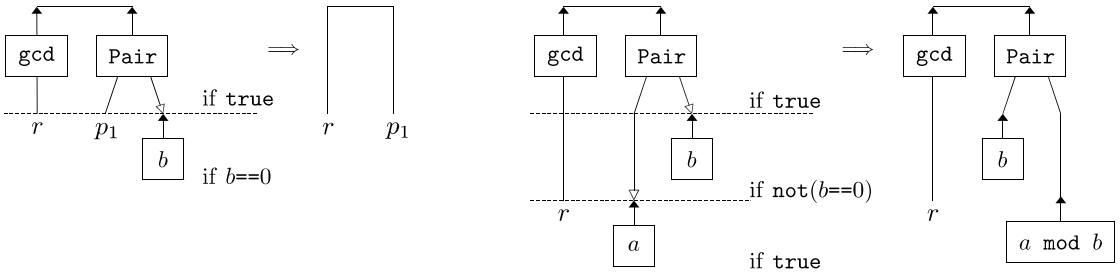


These are written as follows (omitting "if `true`") and the set of the LHS nodes of the rules is sequential:

- $(\langle \mathtt{lastElt}(r) \bowtie \mathtt{Cons}(x,xs),\ \ xs - \mathtt{Nil} \rangle \Rightarrow N_1)$,
- $(\langle \mathtt{lastElt}(r) \bowtie \mathtt{Cons}(x,xs),\ \ xs - \mathtt{Cons}(y,ys) \rangle \Rightarrow N_2)$.

Example 3.6. The rewriting rules in Example 3.2 are textually written and graphically drawn as follows and the set of the LHS nodes of the rules is also sequential:

- $(\langle \mathtt{gcd}(r) \bowtie \mathtt{Pair}(p_1,p_2) \text{ if } \mathtt{true},\ \ p_2 - \mathtt{Int}(b) \text{ if } b\mathtt{==}0 \rangle \Rightarrow N_1)$,
- $(\langle \mathtt{gcd}(r) \bowtie \mathtt{Pair}(p_1,p_2) \text{ if } \mathtt{true},\ \ p_2 - \mathtt{Int}(b) \text{ if } \mathtt{not}(b\mathtt{==}0),\ \ p_1 - \mathtt{Int}(a) \text{ if } \mathtt{true} \rangle \Rightarrow N_2)$.



Next, we extend well-formed rule sets [4] as sets of pairwise distinct rules on conditional NAPs:

Definition 3.7 (Subnets of conditional NAPs)**.** Let $\langle P_{if} \rangle$ and $\langle P'_{if} \rangle$ be conditional NAPs. $\langle P_{if} \rangle$ is a *subnet* of $\langle P'_{if} \rangle$ if:

- $\langle P \rangle$ is a subnet of $\langle P' \rangle$, where $\langle P \rangle$ and $\langle P' \rangle$ are condition dropped NAPs for $\langle P_{if} \rangle$ and $\langle P'_{if} \rangle$, respectively.
- $\langle P_{if} \rangle$ is available if $\langle P'_{if} \rangle$ is available.

Definition 3.8 (Sets of pairwise distinct rules)**.** A set of *pairwise distinct* rules on conditional NAPs $\mathscr{R}_{if}$ is defined if:

- There is a local sequential set which contains every conditional NAP $\langle P_{if} \rangle$ such that $(\langle P_{if} \rangle \Rightarrow N) \in \mathscr{R}_{if}$,
- For any $(\langle P_{if} \rangle \Rightarrow N) \in \mathscr{R}_{if}$, $\mathscr{R}_{if}$ has no $(\langle P'_{if} \rangle \Rightarrow N')$ where $\langle P'_{if} \rangle$ is a subnet of $\langle P_{if} \rangle$.

We call a rule set *pairwise distinct* if the set satisfies conditions in Definition 3.8. We also call rules *pairwise distinct* if there is a set of pairwise distinct rules containing them.

As long as a rule set is pairwise distinct, one-step confluence property is preserved as follows:

Proposition 3.9 (One-step confluent)**.** *When a given set of rules on conditional* NAP*s $\mathscr{R}_{if}$ is pairwise distinct, then all reductions using rules in $\mathscr{R}_{if}$ are confluent in one step.*

Proof. We assume that $\mathscr{R}_{if}$ has two rules $(\langle P_{if} \rangle \Rightarrow N)$ and $(\langle P'_{if} \rangle \Rightarrow N')$ which rewrite the same NAP into different nets N and N'. Let $\langle P \rangle$ and $\langle P' \rangle$ be condition dropped NAPs for $\langle P_{if} \rangle$ and $\langle P'_{if} \rangle$, respectively.

First, we suppose that $\langle P \rangle$ and $\langle P' \rangle$ are the same. If both $\langle P_{if} \rangle$ and $\langle P'_{if} \rangle$ are available, then conditional expressions in $\langle P_{if} \rangle$ and $\langle P'_{if} \rangle$ are not disjoint, and this contradicts that $\langle P_{if} \rangle$ and $\langle P'_{if} \rangle$ are elements of the same sequential set. Otherwise, at most only one rule can be applied to the $\langle P \rangle$, and thus no critical pairs will be created by using both rules.

Next, we suppose that the $\langle P \rangle$ is a subnet of $\langle P' \rangle$. If both $\langle P_{if} \rangle$ and $\langle P'_{if} \rangle$ are available, then $\langle P_{if} \rangle$ is a subnet of $\langle P'_{if} \rangle$, and it contradicts that the $\mathscr{R}_{if}$ is well-formed. Otherwise, there is at most only one rule that can be applied to both $\langle P \rangle$ and $\langle P' \rangle$. As a result, there are no critical pairs by using both rules. □

4 Translation of nested conditional rules into non-nested ones

In this section we introduce a translation **T** of a rewriting rule on a conditional NAP $(\langle P_{if} \rangle \Rightarrow N)$ into non-nested conditional rules and show properties of the translation **T**. Generally nested pattern matching can be unfolded by introducing fresh symbols into less nested ones [5]. In NAPs each nested agent is inductively connected, and thus by considering such induction steps as the local sequential order, we realise the nested matching as non-nested ones by introducing fresh agents.

Definition 4.1. We define a translation **T** of a rewriting rule on a conditional NAP $(\langle P_{if} \rangle \Rightarrow N)$ into non-nested conditional rules by structural induction on conditional NAPs:

Base case: If the $\langle P_{if} \rangle$ is $\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c \rangle$, then the translation **T** works as the identity function.

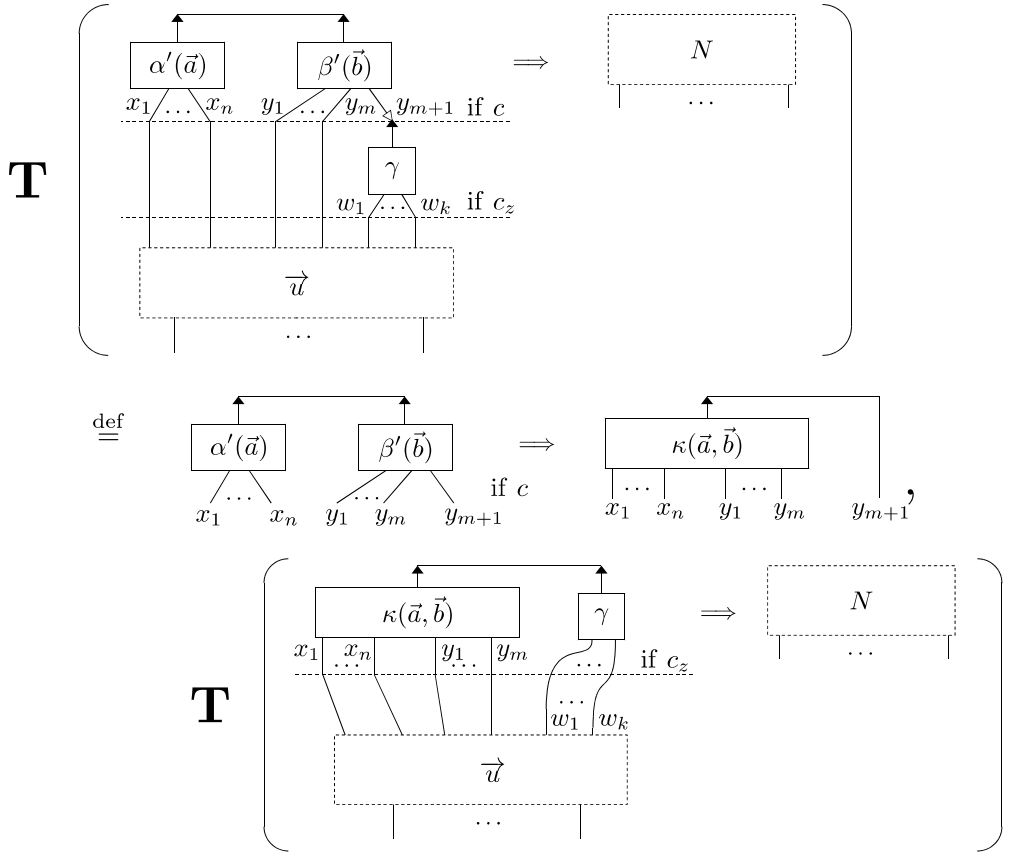
Step case: If the $\langle P_{if} \rangle$ is $\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c,\ z - \gamma(\vec{w}) \text{ if } c_z,\ \vec{u} \rangle$ where z is an auxiliary port in $\alpha(\vec{x}) \bowtie \beta(\vec{y})$ and $\vec{u}$ is a sequence of connections such as "$z_1 - \gamma(\vec{w}_1)$ if c_1, $z_2 - \gamma(\vec{w}_2)$ if c_2, …".

To draw graphs simply, we suppose that $\vec{y}$ is $y_1, \ldots, y_m, y_{m+1}$ and the z is y_{m+1}. Other cases can be defined in a similar way. We also write the α and β as $\alpha'(\vec{a})$ and $\beta'(\vec{b})$, respectively, in order to show the attribute variables explicitly.

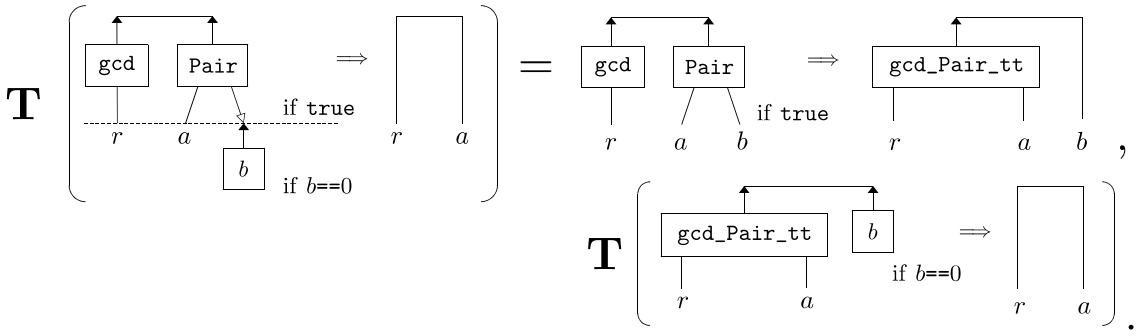
The **T** generates the following rules:

- $(\alpha'(\vec{a})(\vec{x}) \bowtie \beta'(\vec{b})(\vec{y}) \text{ if } c \Rightarrow M)$ where M is a net obtained by connecting the y_{m+1} to the principal port of an agent $\kappa(\vec{a},\vec{b})(\vec{x}, y_1, \ldots, y_m)$ and the κ is fresh for Σ and the previous occurring symbols and depending uniquely on the symbols α', β' and the conditional expression c,

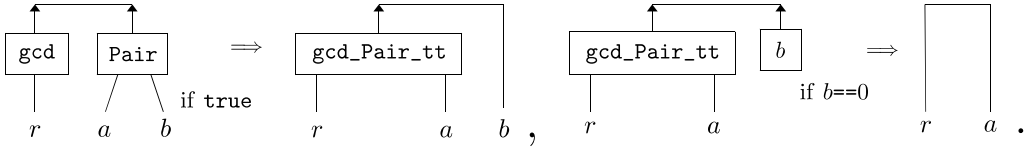
- $\mathbf{T}[(\langle \kappa(\vec{a},\vec{b})(\vec{x},y_1,\ldots,y_m) \bowtie \gamma(\vec{w}) \text{ if } c_z,\ \vec{u}\rangle \Rightarrow N)]$.



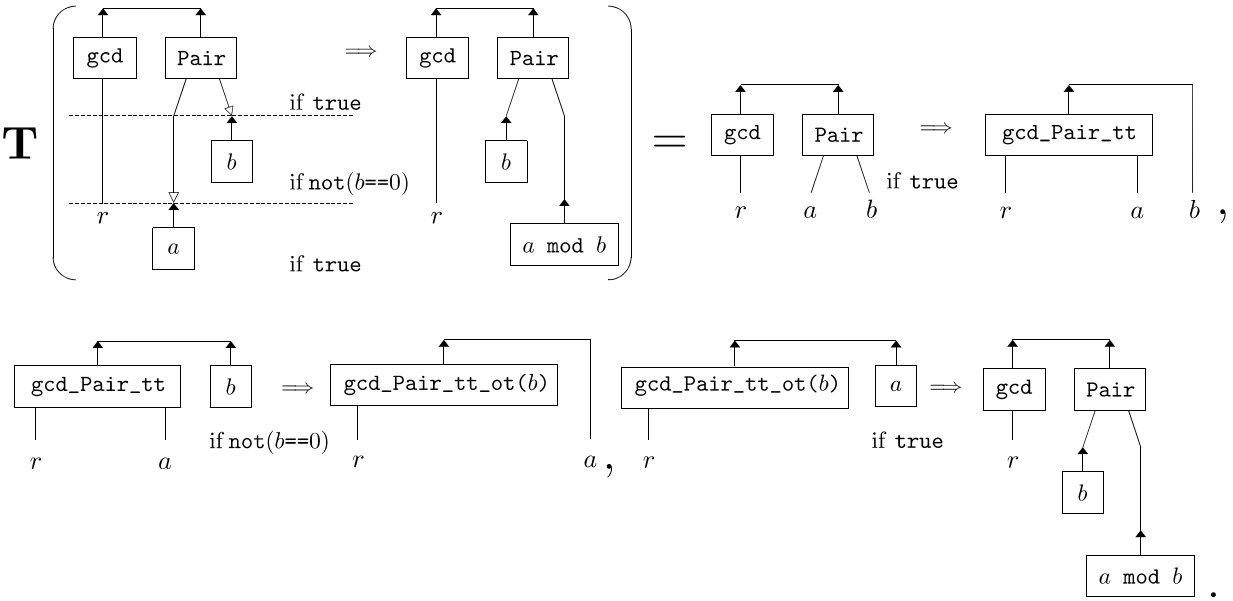
Example 4.2. We show the translation of the rules for $\texttt{gcd} \bowtie \texttt{Pair}$ in Example 3.2. First, we take the following rule: $(\langle \texttt{gcd}(r) \bowtie \texttt{Pair}(p_1,p_2) \text{ if } \texttt{true},\ p_2 - \texttt{Int}(b) \text{ if } b{==}0\rangle \Rightarrow N_1)$. By applying the translation to the rule, we have the following rule and translation:



T works for $\langle \texttt{gcd\_Pair\_tt} \bowtie \texttt{Int}(b) \text{ if } \texttt{true}\rangle$ as the identity function, so we obtain the following rules as the result:



The other rule for gcd ⋈ Pair is similarly translated as follows:



By the obtained conditional rules for gcd_Pair_tt ⋈ Int(b), we have two branches according to the value of b—return the result a if b is 0, otherwise proceed to the calculation. We note the four rules obtained are the same as the non-nested rules in Example 3.2, by changing gcd_Pair_tt and gcd_Pair_tt_ot into gcd1 and gcd2, respectively.

For each connection of a conditional NAP, the translation T introduces a rule in which one of the active pair agents has a fresh symbol for Σ and the previous occurring ones. Therefore, the following holds:

Lemma 4.3. *Let R_{if} be a rule on a conditional* NAP. $\mathrm{T}[R_{if}]$ *does not have two rules for the same active pair P such that* $(P \text{ if } c_1 \Rightarrow N_1)$ *and* $(P \text{ if } c_2 \Rightarrow N_2)$. □

Lemma 4.4. *Let R_1, R_2 be pairwise distinct rules on conditional* NAPs. *When for the same active pair P,* $(P \text{ if } c_1 \Rightarrow N_1)$ *and* $(P \text{ if } c_2 \Rightarrow N_2)$ *are introduced by* $\mathrm{T}[R_1]$ *and* $\mathrm{T}[R_2]$, *then c_1, c_2 are disjoint or $N_1 = N_2$.* □

We suppose an interaction net that has a symbol set Σ and a set of pairwise distinct rules on conditional NAPs $\mathscr{R}_{if}$. By applying the translation T to each rule in $\mathscr{R}_{if}$, we obtain two sets $\mathscr{R}'_{if}$ and Σ'—a set of (non-nested) conditional rules and a set of symbol introduced during the translation, respectively. The system with a symbol set $\Sigma \cup \Sigma'$ and a rule set $\mathscr{R}'_{if}$ is also an interaction net by Lemma 4.3 and 4.4. In addition, the following proposition shows that the conditional NAP extension is conservative, *i.e.* an interaction net with $\Sigma \cup \Sigma'$ and $\mathscr{R}_{if}$ is realised without the extension, thus as an interaction net with $\Sigma \cup \Sigma'$ and $\mathscr{R}'_{if}$.

Proposition 4.5. *Let $(\langle P_{if} \rangle \Rightarrow N)$ be a rule on conditional* NAPs *and $\langle P \rangle$ be the condition dropped* NAP *for $\langle P_{if} \rangle$. If $\langle P \rangle$ is reduced to N by using this rule, $\langle P \rangle$ is also reduced to N by using only rules obtained from* $\mathbf{T}[(\langle P_{if} \rangle \Rightarrow N)]$.

Proof. By structural induction on conditional NAPs.

Base case: The translation T works as the identity function, so the $\langle P \rangle$ is reduced to N by using rules obtained from the translation.

Step case: We assume that $\langle P_{if}\rangle$ is $\langle P_1 \text{ if } c_1,\ z-\gamma(\vec{w}) \text{ if } c,\ \vec{u}\rangle$ and the condition dropped NAP $\langle P\rangle$ is reduced to N by the rule $(\langle P_{if}\rangle \Rightarrow N)$. T generates the following rules:

(1) $(P_1 \text{ if } c_1 \Rightarrow M)$ where M is a net obtained by connecting the z to the principal port of an agent $\kappa(\vec{z})$, where the symbol κ is a fresh and uniquely depending on the agents symbols in P_1 and the c_1.

(2) $\mathrm{T}[\langle \kappa(\vec{z}) \bowtie \gamma(\vec{w}) \text{ if } c,\ \vec{u}\rangle \Rightarrow N]$.

By the rule of (1), P_1 is reduced to an active pair (κ,γ), and it is iteratively reduced to N by the induction hypothesis. □

5 A notation for rules on conditional NAPs

In this section, we introduce a notation to facilitate expressing sets of pairwise distinct rules.

It is practical to write rules for an active pair $\alpha \bowtie \beta$ with conditional expressions $c_1,\ldots,c_n$ as *guards* like in Haskell [10], which are evaluated one at a time from c_1 to c_n, as follows:

$$\begin{array}{l}\alpha(\vec{x}) \bowtie \beta(\vec{y})\\ \quad \mid\ c_1 \Rightarrow N_1\\ \quad \mid\ c_2 \Rightarrow N_2\\ \qquad \vdots\\ \quad \mid\ c_n \Rightarrow N_n.\end{array}$$

The last placed "*otherwise*" is evaluated as true. We can suppose that this is translated into the following rules where the conditional expressions do not overlap:

$$\begin{array}{l}((\alpha,\beta) \text{ if } c_1 \Rightarrow N_1),\\ ((\alpha,\beta) \text{ if } \mathtt{not}(c_1)\ \mathtt{and}\ c_2 \Rightarrow N_2),\ \ldots,\\ ((\alpha,\beta) \text{ if } \mathtt{not}(c_1\ \mathtt{or}\ c_2\ \mathtt{or}\ \cdots\ \mathtt{or}\ c_{n-1})\ \mathtt{and}\ c_n \Rightarrow N_n).\end{array}$$

As another example, we take the following two rules whose bases of conditional NAPs are the same ones which are followed by connections to the same port:

$$\begin{array}{l}(\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c,\ z-\gamma(\vec{z_1}) \text{ if } c_1\rangle \Rightarrow N_1),\\ (\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c,\ z-\kappa(\vec{z_2}) \text{ if } c_2\rangle \Rightarrow N_2)\end{array}$$

It is also convenient to group these connections by the same port z with guards and a `case`-expression as follows:

$$\begin{array}{l}\alpha(\vec{x}) \bowtie \beta(\vec{y})\\ \quad \mid\ c\ \mathtt{->}\ \mathtt{case\ of}\ z\\ \qquad\qquad \gamma(\vec{z_1}) \quad \mid\ c_1 \Rightarrow N_1\\ \qquad\qquad \kappa(\vec{z_2}) \quad \mid\ c_2 \Rightarrow N_2.\end{array}$$

We may omit the guards if there is only one condition `true`.

Example 5.1. By using this notation, the rules for $\mathtt{gcd} \bowtie \mathtt{Pair}$ in Example 3.2 is written in the following single sentence:

$$\begin{array}{l}\mathtt{gcd}(r) \bowtie \mathtt{Pair}(p_1,p_2)\\ \quad \mathtt{->}\ \mathtt{case\ of}\ p_2\\ \qquad \mathrm{Int}(b) \quad \mid\ b\mathtt{==}0 \Rightarrow N_1\\ \qquad\qquad\quad \mid\ \mathit{otherwise}\ \mathtt{->}\ \mathtt{case\ of}\ p_1\\ \qquad\qquad\qquad\qquad\qquad \mathrm{Int}(a) \Rightarrow N_2.\end{array}$$

5.1 Definition and translations of the notation

First, we define the rule notation by the following *Rule* with *Spray*s, which are branches connected to nets:

$$\begin{array}{rcl} \mathit{Rule} & ::= & \alpha(\vec{x}) \bowtie \beta(\vec{y}) \;\; \begin{array}{|ll} c_1 & \mathit{Spray}_1 \\ c_2 & \mathit{Spray}_2 \\ \vdots & \\ c_m & \mathit{Spray}_m \end{array} \end{array} \qquad \begin{array}{rcl} \mathit{Spray} & ::= & \Rightarrow N \\ & | & \texttt{->}\ \texttt{case}\ z\ \texttt{of} \\ & & \quad \begin{array}{rl|ll} \gamma_1(\vec{w}_1) & & c_{11} & \mathit{Spray}_{11} \\ & & \vdots & \\ & & c_{1m_1} & \mathit{Spray}_{1m_1} \\ \vdots & & & \\ \gamma_n(\vec{w}_n) & & c_{n1} & \mathit{Spray}_{n1} \\ & & \vdots & \\ & & c_{nm_n} & \mathit{Spray}_{nm_n} \end{array} \end{array}$$

where $\alpha(\vec{x})$, $\beta(\vec{y})$ are agents, c_i is a conditional expression, N is a net, z is a free port, $\gamma_1(\vec{w}_1), \ldots, \gamma_n(\vec{w}_n)$ are distinct agents, and the conditional expression "*otherwise*" can appear last. We may write *Spray* as S when there is no confusion.

Translation of the rule notation to rules on conditional NAPs: We define the following translations of the rule notation to obtain rules on conditional NAPs:

$$\mathrm{T_R} : \mathit{Rule} \mapsto \overrightarrow{\mathrm{T_S}[\langle P_{if}\rangle,\ \mathit{Spray}]}, \quad \mathrm{T_S} : (\langle P_{if}\rangle,\ \mathit{Spray}) \mapsto \vec{r}$$

where $\langle P_{if}\rangle$ is a conditional NAP, $\vec{r}$ is a rule sequence. If the $\langle P_{if}\rangle$ obtained during translation do not satisfy the conditions of given in Definition 3.1, then the translation will fail.

We may write "$\texttt{not}(c_1\ \texttt{or}\ c_2\ \texttt{or}\ \cdots\ \texttt{or}\ c_{n-1})\ \texttt{and}\ c_n$" as "$\texttt{not}(c_1,\ c_2, \ldots,\ c_{n-1})\ \texttt{\&}\ c_n$" to save space. In addition, we may write "$\texttt{not}(c_1,\ c_2, \ldots,\ c_{n-1})\ \texttt{\&}$ *otherwise*" just as "*otherwise*".

$$\mathrm{T_R}\left[\begin{array}{rl|ll} \alpha(\vec{x}) \bowtie \beta(\vec{y}) & & c_1 & S_1 \\ & & c_2 & S_2 \\ & & \vdots & \\ & & c_n & S_n \end{array}\right] \stackrel{\text{def}}{=} \begin{array}{l} \mathrm{T_S}[\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c_1\rangle,\ S_1], \\ \mathrm{T_S}[\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } \texttt{not}(c_1)\ \texttt{\&}\ c_2\rangle,\ S_2] \\ \qquad\qquad \vdots \\ \mathrm{T_S}[\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } \texttt{not}(c_1,\ c_2, \ldots,\ c_{n-1})\ \texttt{\&}\ c_n\rangle,\ S_n]. \end{array}$$

$$\mathrm{T_S}\left[\langle P_{if}\rangle,\ \Rightarrow N\right] \stackrel{\text{def}}{=} (\langle P_{if}\rangle \Rightarrow N),$$

$$\mathrm{T_S}\left[\langle P_{if}\rangle,\ \begin{array}{rl|ll} \texttt{->}\ \texttt{case}\ z\ \texttt{of} & & & \\ \quad \gamma_1(\vec{w}_1) & & c_{11} & S_{11} \\ & & \vdots & \\ & & c_{1m_1} & S_{1m_1} \\ \vdots & & & \\ \quad \gamma_n(\vec{w}_n) & & c_{n1} & S_{n1} \\ & & \vdots & \\ & & c_{nm_n} & S_{nm_n} \end{array}\right] \stackrel{\text{def}}{=} \begin{array}{l} \mathrm{T_S}[\langle P_{if},\ z - \gamma_1(\vec{w}_1) \text{ if } c_{11}\rangle,\ S_{11}], \\ \qquad\qquad \vdots \\ \mathrm{T_S}[\langle P_{if},\ z - \gamma_1(\vec{w}_1) \text{ if } \texttt{not}(c_{11}, \ldots,\ c_{nm_1-1})\ \texttt{\&}\ c_{1m_1}\rangle,\ S_{1m_1}], \\ \qquad\qquad \vdots \\ \mathrm{T_S}[\langle P_{if},\ z - \gamma_n(\vec{w}_n) \text{ if } c_{n1}\rangle,\ S_{n1}], \\ \qquad\qquad \vdots \\ \mathrm{T_S}[\langle P_{if},\ z - \gamma_1(\vec{w}_n) \text{ if } \texttt{not}(c_{n1}, \ldots,\ c_{nm_n-1})\ \texttt{\&}\ c_{nm_n}\rangle,\ S_{nm_n}]. \end{array}$$

Example 5.2. The following is a translation of the rule notation in Example 5.1:

$$\mathrm{T_R}\left[\begin{array}{l}\texttt{gcd}(r) \bowtie \texttt{Pair}(p_1,p_2)\\ \quad \texttt{| true -> case of } p_2\\ \qquad\qquad \mathrm{Int}(b) \texttt{ | } b\texttt{==0} \Rightarrow N_1\\ \qquad\qquad\quad \texttt{| } \mathit{otherwise} \texttt{ -> case of } p_1\\ \qquad\qquad\qquad\qquad \mathrm{Int}(a) \texttt{ | true} \Rightarrow N_2\end{array}\right]$$

$$= \mathrm{T_S}\left[\langle \texttt{gcd}(r) \bowtie \texttt{Pair}(p_1,p_2) \text{ if } \texttt{true}\rangle, \begin{array}{l}\texttt{-> case of } p_2\\ \quad \mathrm{Int}(b) \texttt{ | } b\texttt{==0} \Rightarrow N_1\\ \qquad \texttt{| } \mathit{otherwise} \texttt{ -> case of } p_1\\ \qquad\qquad \mathrm{Int}(a) \texttt{ | true} \Rightarrow N_2\end{array}\right]$$

$$= \mathrm{T_S}\left[\langle \texttt{gcd}(r) \bowtie \texttt{Pair}(p_1,p_2) \text{ if } \texttt{true},\ p_2 - \mathrm{Int}(b) \text{ if } b\texttt{==0}\rangle,\ \Rightarrow N_1\right],$$
$$\mathrm{T_S}\left[\langle \texttt{gcd}(r) \bowtie \texttt{Pair}(p_1,p_2) \text{ if } \texttt{true},\ p_2 - \mathrm{Int}(b) \text{ if } \mathit{otherwise}\rangle, \begin{array}{l}\texttt{-> case of } p_1\\ \quad \mathrm{Int}(a) \texttt{ | true} \Rightarrow N_2\end{array}\right]$$

$$= (\langle \texttt{gcd}(r) \bowtie \texttt{Pair}(p_1,p_2) \text{ if } \texttt{true},\ p_2 - \mathrm{Int}(b) \text{ if } b\texttt{==0}\rangle \Rightarrow N_1),$$
$$\mathrm{T_S}[\langle \texttt{gcd}(r) \bowtie \texttt{Pair}(p_1,p_2) \text{ if } \texttt{true},\ p_2 - \mathrm{Int}(b) \text{ if } \mathit{otherwise},\ p_1 - \mathrm{Int}(a) \text{ if } \texttt{true}\rangle, \Rightarrow N_2]$$

$$= (\langle \texttt{gcd}(r) \bowtie \texttt{Pair}(p_1,p_2) \text{ if } \texttt{true},\ p_2 - \mathrm{Int}(b) \text{ if } b\texttt{==0}\rangle \Rightarrow N_1),$$
$$(\langle \texttt{gcd}(r) \bowtie \texttt{Pair}(p_1,p_2) \text{ if } \texttt{true},\ p_2 - \mathrm{Int}(b) \text{ if } \mathit{otherwise},\ p_1 - \mathrm{Int}(a) \text{ if } \texttt{true}\rangle \Rightarrow N_2).$$

Thus, we derive the same rules as in Example 3.2.

5.2 Properties of the rule notation

In this section we show properties of the notation. First, we show that the translation $\mathrm{T_R}$ is a function if $\mathrm{T_R}$ does not fail, thus we have the same result for the same rule notation.

Lemma 5.3. *Let* Rule *be a string accepted by the rule notation, and let* $\mathrm{T_R}[\mathrm{Rule}]$ *not fail. If during expansions of* $\mathrm{T_R}[\mathrm{Rule}]$ *to rules, the followings occur for the same conditional* NAP $\langle P_{if}\rangle$*, then* S_1 *and* S_2 *are the same spray:*

- $\mathrm{T_S}[\langle P_{if}\rangle,\ S_1]$,
- $\mathrm{T_S}[\langle P_{if}\rangle,\ S_2]$.

Proof. By structural induction on the conditional NAPs $\langle P_{if}\rangle$. □

From now on, we will only consider the case where the translation $\mathrm{T_R}$ does not fail. By Lemma 5.3, we can show the property more precisely:

Lemma 5.4. *Let* Rule *be a string accepted by the rule notation. If during expansions of* $\mathrm{T_R}[\mathrm{Rule}]$ *to rules, the followings occur for the same conditional* NAP $\langle P_{if}\rangle$*, then* y_1 *and* y_2 *are the same:*

- $\mathrm{T_S}[\langle P_{if},\ y_1 - \gamma_1(\vec{w}_1) \text{ if } c_1\rangle,\ S_1]$,

- $\mathrm{T_S}[\langle P_{if},\ y_2 - \gamma_2(\vec{w}_2) \text{ if } c_2\rangle,\ S_2]$.

In addition, when $\gamma_1(\vec{w}_1)$ and $\gamma_2(\vec{w}_2)$ are the same, c_1 and c_2 are not overlapped.

Proof. By the definition of $\mathrm{T_S}$, $\mathrm{T_S}[\langle P_{if},\ y_1 - \gamma_1(\vec{w}_1) \text{ if } c_1\rangle,\ S_1]$ and $\mathrm{T_S}[\langle P_{if},\ y_2 - \gamma_2(\vec{w}_2) \text{ if } c_2\rangle,\ S_2]$ are directly derived from $\mathrm{T_S}[\langle P_{if}\rangle,\ S'_1]$ and $\mathrm{T_S}[\langle P_{if}\rangle,\ S'_2]$ for some S'_1 and S'_2, respectively. By Lemma 5.3, $S'_1 = S'_2$, and therefore this lemma holds. □

Here we form a set whose elements are conditional NAPs that appear during expansions of $\mathrm{T_R}$ to rules, and show that the set is sequential.

Definition 5.5. For conditional NAPs $\langle P_{if}\rangle$ and $\langle P_{if},\ \vec{u}\rangle$ for any sequence of connections $\vec{u}$ (which may be empty), we write $\langle P_{if}\rangle \subseteq \langle P_{if},\ \vec{u}\rangle$. We use $\mathrm{AllSub}(\langle P_{if}\rangle)$ as a set $\{P'_{if} \mid P'_{if} \subseteq P_{if}\}$.

Example 5.6. The followings are the results of applying AllSub to rules obtained in Example 5.2:

- Elements of $\mathrm{AllSub}(\langle \mathtt{gcd}(r) \bowtie \mathtt{Pair}(p_1,p_2) \text{ if } \mathtt{true},\ p_2 - \mathrm{Int}(b) \text{ if } b\mathtt{==}0\rangle)$:
  - $\langle \mathtt{gcd}(r) \bowtie \mathtt{Pair}(p_1,p_2) \text{ if } \mathtt{true}\rangle$,
  - $\langle \mathtt{gcd}(r) \bowtie \mathtt{Pair}(p_1,p_2) \text{ if } \mathtt{true},\ p_2 - \mathrm{Int}(b) \text{ if } b\mathtt{==}0\rangle$.
- Elements of $\mathrm{AllSub}(\langle \mathtt{gcd}(r) \bowtie \mathtt{Pair}(p_1,p_2) \text{ if } \mathtt{true},\ p_2 - \mathrm{Int}(b) \text{ if } \mathit{otherwise},\ p_1 - \mathrm{Int}(a) \text{ if } \mathtt{true}\rangle)$:
  - $\langle \mathtt{gcd}(r) \bowtie \mathtt{Pair}(p_1,p_2) \text{ if } \mathtt{true}\rangle$,
  - $\langle \mathtt{gcd}(r) \bowtie \mathtt{Pair}(p_1,p_2) \text{ if } \mathtt{true},\ p_2 - \mathrm{Int}(b) \text{ if } \mathit{otherwise}\rangle$,
  - $\langle \mathtt{gcd}(r) \bowtie \mathtt{Pair}(p_1,p_2) \text{ if } \mathtt{true},\ p_2 - \mathrm{Int}(b) \text{ if } \mathit{otherwise},\ p_1 - \mathrm{Int}(a) \text{ if } \mathtt{true}\rangle$.

Proposition 5.7. *Let* Rule *be a string accepted by the rule notation. When we obtain rules $(\langle P_{if_1}\rangle \Rightarrow N_1), \ldots, (\langle P_{if_m}\rangle \Rightarrow N_m)$ from $\mathrm{T_R}$[Rule], then $\mathrm{AllSub}(\langle P_{if_1}\rangle) \cup \cdots \cup \mathrm{AllSub}(\langle P_{if_m}\rangle)$ is sequential.*

Proof. We examine whether the set $\mathscr{A} = \mathrm{AllSub}(\langle P_{if_1}\rangle) \cup \cdots \cup \mathrm{AllSub}(\langle P_{if_m}\rangle)$ satisfies sequentiality conditions in Definition 3.4.

First, we check the condition (1a). Each $\langle P_{if_i}\rangle$ is obtained from the same rule notation Rule by using the translation of $\mathrm{T_R}$. Therefore, when $\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c_1\rangle, \langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c_2\rangle \in \mathscr{A}$, c_1 and c_2 are disjoint, by the definition of $\mathrm{T_R}$.

Next, we verify conditions in (2). We suppose that $(\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c,\ u_1,\ u_2, \ldots,\ u_n\rangle \Rightarrow N)$ is derived from $\mathrm{T_R}$[Rule]. All elements of $\mathrm{AllSub}(\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c,\ u_1,\ u_2, \ldots,\ u_n\rangle)$ are related by $\subseteq$ as follows:

$$\langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c\rangle \subseteq \langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c,\ u_1\rangle \subseteq \cdots \subseteq \langle \alpha(\vec{x}) \bowtie \beta(\vec{y}) \text{ if } c,\ u_1,\ u_2, \ldots,\ u_n\rangle.$$

By the definition of $\mathrm{T_R}$, for each two elements such that $\langle P_{if}\rangle \subseteq \langle P_{if},\ u\rangle$ in the AllSub, there is the following expansion:

$$\mathrm{T_S}[\langle P_{if}\rangle,\ S] = \ldots,\ \mathrm{T_S}[\langle P_{if},\ u\rangle,\ S'],\ \ldots.$$

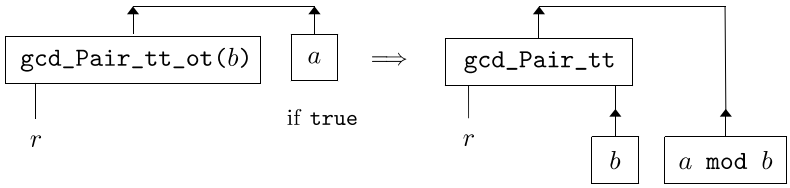
Thus every element must occur as the first argument of $\mathrm{T_S}$ during the applications of $\mathrm{T_R}$. Therefore, by Lemma 5.4, each $\mathrm{AllSub}(\langle P_{if_i}\rangle)$ satisfies (2a) and (2c). (2b) also is satisfied by the definition of AllSub. $\mathrm{AllSub}(\langle P_{if_i}\rangle)$ and $\mathrm{AllSub}(\langle P_{if_j}\rangle)$ are disjoint because these are derived for the same Rule by $\mathrm{T_R}$[Rule]. Therefore, $\mathscr{A}$ also satisfies all these conditions. □

We suppose that $\mathscr{R}_{if}$ is a rule set whose elements are obtained from the same rule notation Rule by using $\mathrm{T_R}$[Rule]. Rules in $\mathscr{R}_{if}$ do not have any overlapped conditions, and thus there are not any rules $(\langle P_{if}\rangle \Rightarrow N)$ such that the $\langle P_{if}\rangle$ is a subnet of $\langle P'_{if}\rangle$ for other rules $(\langle P'_{if}\rangle \Rightarrow N')$ in $\mathscr{R}_{if}$. Therefore, as long as we use the rule notation, the rule set derived from $\mathscr{R}_{if}$ will be pairwise distinct.

Proposition 5.8. *Let* Rule *be a string accepted by the rule notation, and $\mathscr{R}_{if}$ a set whose elements are obtained from $\mathrm{T_R}$*[Rule]. *Then, $\mathscr{R}_{if}$ is pairwise distinct.* □

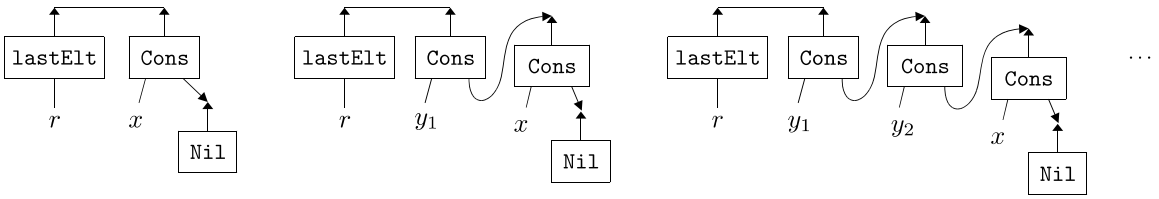
6 Discussion

Implementation: Using rules on conditional NAPs allows us to write algorithms naturally and reduces the number of reduction steps in comparison with using non-nested ones. However, when it comes to the implementation of these rules, we need an efficient matching mechanism so that the rule application waits until each nested agent is connected to all its ports and avoids repeatedly checking that these connections have been made. This method can be realised by the T translation, since it ensures that the nested agents are connected. Given this, the matching can then be performed by the net evaluator. Another advantage of using the T translation is that it is then possible to perform some rule optimisation, by combining rules. For instance, we take the translated rules in Example 4.2. The RHS of the rule $\texttt{gcd\_Pair\_tt\_ot}(b) \bowtie \texttt{Int}(a)$ has an agent pair $\texttt{gcd} \bowtie \texttt{Pair}$. Thus, by applying a rule for $\texttt{gcd} \bowtie \texttt{Pair}$ to the RHS, we can have a one-step reduced optimisation rule as follows:



Therefore we can say that by using the translation T, conditional NAPs can be realised, taking advantage of the expressiveness of the case notation and the already implemented evaluator such as Inpla [12] [1].

Related work: It is also possible to realise nested pattern matching for pure interaction nets by using other approaches [1, 13] where agents are allowed to have more than one principal port. However, there are some limitations, as discussed in [4]. For example, there is no natural way to express the function `lastElt`, which returns the last element of a list. The last element of a list $[x]$ is matched by using a pattern, which is the LHS of a rule, with `Cons` agent having two principal ports and `Nil` agent. However, to keep the one-step confluence, the patterns must be such that every principal port is connected to the principal port of another agent. Thus, because two principal ports of `Cons` cannot be free, we need to have every pattern for lists such as $[y_1,x]$, $[y_1,y_2,x]$, ...



7 Conclusion

In this paper we introduced conditional nested pattern matching as an extension of [4], and showed that as long as a rule set is pairwise distinct, the rewritings can continue to be one-step confluent. This not only allows programs to be more easily written in interaction net form, but also allows the execution to be performed by existing evaluators. Most well-known algorithms contain computations of conditional nested pattern matching, and these can now be realised in interaction nets. We expect this will contribute to finding new ways of implementing such algorithms, taking advantage of the benefits of interaction nets, such as built-in parallelism and internal garbage collection.

[1] An up-to-date implementation is available from `https://github.com/inpla/inpla`

References

[1] Denis Béchet (1992): *Partial Evaluation of Interaction Nets*. In: *Actes WSA'92 Workshop on Static Analysis, Series Bigre* 81-82, Atelier Irisa, IRISA, Campus de Beaulieu, pp. 331–338.

[2] M. Fernández, I. Mackie & J. Sousa Pinto (2001): *Combining interaction nets with externally defined programs,*. In: *Electronic proceedings of the APPIA-GULP-PRODE Joint Conference on Declarative Programming*, pp. 297–312. Available at `http://hdl.handle.net/1822/776`.

[3] G. Gonthier, M. Abadi & J.-J. 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, doi:10.1145/143165.143172.

[4] Abubakar Hassan & Shinya Sato (2007): *Interaction Nets With Nested Pattern Matching*. In: *Proceedings of the Fourth International Workshop on Computing with Terms and Graphs, TERMGRAPH 2007*, *Electronic Notes in Theoretical Computer Science* 203, Elsevier, pp. 79–92, doi:10.1016/J.ENTCS.2008.03.035.

[5] J. R. Kennaway (1988): *Implementing term rewrite languages in Dactl*. In M. Dauchet & M. Nivat, editors: *CAAP '88*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 102–116, doi:10.1007/BFb0026099.

[6] Y. Lafont (1990): *Interaction Nets*. In: *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, ACM Press, pp. 95–108, doi:10.1145/96709.96718.

[7] J. Lamping (1990): *An Algorithm for Optimal Lambda Calculus Reduction*. In: *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, ACM Press, pp. 16–30, doi:10.1145/96709.96711.

[8] I. Mackie (1998): *YALE: Yet Another Lambda Evaluator Based on Interaction Nets*. In: *Proceedings of the 3rd ACM SIGPLAN International Conference on Functional Programming (ICFP'98)*, ACM Press, pp. 117–128, doi:10.1145/291251.289434.

[9] I. Mackie & S. Sato (2015): *Parallel Evaluation of Interaction Nets: Case Studies and Experiments*. *Electronic Communications of the EASST* 73, doi:10.14279/tuj.eceasst.73.1034.

[10] Simon Marlow et al. (2010): *Haskell 2010 language report*. Available at `https://www.haskell.org/onlinereport/haskell2010/`.

[11] Gordon D. Plotkin (1977): *LCF Considered as a Programming Language*. *Theoretical Computer Science* 5, pp. 223–255, doi:10.1016/0304-3975(77)90044-5.

[12] S. Sato (2015): *Design and implementation of a low-level language for interaction nets*. Ph.D. thesis, University of Sussex, UK. Available at `https://hdl.handle.net/10779/uos.23417312.v1`.

[13] François-Régis Sinot & Ian Mackie (2004): *Macros for Interaction Nets: A Conservative Extension of Interaction Nets*. In: *Proceedings of the 2nd International Workshop on Term Graph Rewriting, TERMGRAPH 2004*, *Electronic Notes in Theoretical Computer Science* 127, Elsevier, pp. 153–169, doi:10.1016/j.entcs.2005.02.016.

Random Graph Generation in Context-Free Graph Languages

Federico Vastarini
University of York
York, UK
federico.vastarini@york.ac.uk

Detlef Plump
University of York
York, UK
detlef.plump@york.ac.uk

We present a method for generating random hypergraphs in context-free hypergraph languages. It is obtained by adapting Mairson's generation algorithm for context-free string grammars to the setting of hyperedge replacement grammars. Our main results are that for non-ambiguous hyperedge replacement grammars, the method generates hypergraphs uniformly at random and in quadratic time. We illustrate our approach by a running example of a hyperedge replacement grammar generating term graphs.

1 Introduction

We present a novel approach to the generation of random hypergraphs in user-specified domains. Our approach extends a method of Mairson for generating strings in context-free languages [11] to the setting of context-free hypergraph languages specified by hyperedge replacement grammars. Generating (or "sampling") graphs and hypergraphs according to a given probability distributions is a problem that finds application in testing algorithms and programs working on graphs. Molecular biology and cryptography are two fields of potential application where our methods could find a concrete use besides the mere software testing. In [9] Kajino presents a novel approach for the representation of molecules through hypergraphs. Specifically adapting our method to this setting would provide an instrument for the exploration of new compounds in the field of molecular biology. The uniformity of the distribution of our method is a fundamental requirement for the development of cryptographic protocols. In [6] and [12] we may find some useful insights on how to model graph based algorithms in that domain. In the setting of hyperedge replacement grammars, we believe that there is an opportunity for the development of one-way functions.

Our generation algorithm uses as input a hyperedge replacement grammar in Chomsky normal form [2] and a positive integer n. The former specifies the hypergraph language to sample from, the latter the size of the hypergraph to be generated. The algorithm then chooses a hypergraph at random from the slice of the language consisting of all members of size n. We show that if the grammar is non-ambiguous, the generated samples are uniformly distributed. The only requirements for our method are that the properties sought for the generated hypergraphs are representable by a hyperedge replacement language and that, to guarantee a uniform distribution, a non-ambiguous grammar is used as input.

We also show that our method generates a random hypergraph of size n in time $O(n^2)$. This is the same time bound established by Mairson (for the first method) in the setting of random string generation in context-free languages.

S. Alves and I. Mackie (Eds.): 13th International Workshop
on Developments in Computational Models 2023 (DCM'23).
EPTCS 408, 2024, pp. 107–120, doi:10.4204/EPTCS.408.7

2 Hyperedge Replacement Grammars

This section gives a concise overview of the definitions needed to understand the generation process. We also introduce our running example of a language of term graphs specified by hyperedge replacement. For comprehensive treatments of the theory of hyperedge replacement grammars and languages, we refer to Courcelle [3], Drewes et al. [4] and Engelfriet [5].

Let $type\colon C \to \mathbb{N}_0$ be a typing function for a fixed set of labels C, then a *hypergraph* over C is a tuple $H = (V_H, E_H, att_H, lab_H, ext_H)$ where V_H is a finite set of *vertices*, E_H is a finite set of *hyperedges*, $att_H\colon E_H \to V_H^*$ is a mapping assigning a sequence of *attachment nodes* to each $e \in E_H$, $lab_H\colon E_H \to C$ is a function that maps each hyperedge to a *label* such that $type(lab_H(e)) = |att_H(e)|$, $ext_H \in V_H^*$ is a sequence of pairwise distinct *external nodes* (Figure 1). The class of all hypergraphs over C is denoted by $\mathscr{H}_C$.

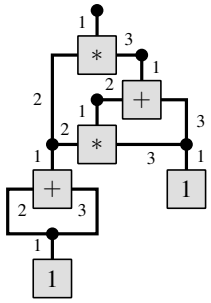


Figure 1: A term graph

We write $type(H)$ for $|ext_H|$ and call H an n-hypergraph if $type(H) = n$. The length of the sequence of *attachments* $|att_H(e)|$ is the *type* of e. Hyperedge e is an m-hyperedge if $type(lab(e)) = m$. We also write $type(e) = m$ if the context is clear. If an n-hypergraph has exactly 1 hyperedge and all its nodes are external, that is $E_H = \{e\}$ and $|V_H| = n$, it is called the *handle* induced by e and denoted by I_e. Moreover if $type(e) = n$ and $ext_H = att(e)$ such a hypergraph is called the handle induced by A and denoted by $A^\bullet$. We write $att_H(e)_i$ for the ith attachment node of $e \in E_H$ and $ext_{H,i}$ for the ith external node of H. The set $E_H^X = \{e \in E \mid lab_H(e) \in X\}$ is the subset of E_H with labels in $X \subseteq C$. We define $|H| = |V_H| + |E_H|$ as the *size* of H and we call H a size-n-hypergraph if $|H| = n$.

Figure 1 shows a *term graph*, a form of acyclic hypergraphs that represent functional expressions with possibly shared subexpressions. (See [13] for an introduction to the area of term graph rewriting.) Grey boxes represent hyperedges labelled with the function symbols $*$, $+$ and 1, while nodes are drawn as black bullets. Lines connect hyperedges with their attachment nodes, whose position in the attachment sequence is given by small numbers.

Two hypergraphs $H, H' \in \mathscr{H}_C$ are *isomorphic*, denoted $H \cong H'$, if there are bijective mappings $h_V\colon V_H \to V_{H'}$ and $h_E\colon E_H \to E_{H'}$ such that $h_V^*(att_H(e)) = att_{H'}(h_E(e))$ and $lab_H(e) = lab_{H'}(h_E(e))$ for each $e \in E_H$ and $h_V^*(ext_H) = ext_{H'}$. Two isomorphic hypergraphs are considered to be the same. A hypergraph H is a subgraph of H', denoted as $H \subseteq H'$ if $V_H \subseteq V_{H'}$ and $E_H \subseteq E_{H'}$.

Hypergraphs are generated by replacement operations. Let $H \in \mathscr{H}_C$ and $B \subseteq E_H$ and let $repl\colon B \to \mathscr{H}_C$ be a mapping with $type(repl(e)) = type(e)$ for each $e \in B$. Then the *replacement* of the hyperedges in B with respect to $repl(e)$ is defined by the operations: remove the subset B of hyperedges from E_H; for each $e \in B$, disjointly add the vertices and the hyperedges of $repl(e)$; for each $e \in B$ and $1 \leq i \leq type(e)$, fuse the ith external node $ext_{repl(e),i}$ with the ith attachment node $att_B(e)_i$.

We denote the resulting hypergraph by $H[e_1/R_1,\ldots,e_n/R_n]$, where $B=\{e_1,\ldots,e_n\}$ and $repl(e_i)=R_i$ for $1\leq i\leq n$, or $H[repl]$. The replacement preserves the external nodes, thus $ext_{H[repl]}=ext_H$.

Given the subsets $\Sigma,N\subseteq C$ used as terminal and non-terminal labels, with $\Sigma\cap N=\emptyset$, we denote E_H^Σ and E_H^N respectively the subsets of terminal and non-terminal hyperedges of H.

The replacements applied during the generation are defined in productions: $p=(A,R)$ is a *production* over N, where $lhs(p)=A\in N$ is the label of the replaced hyperedge and $rhs(p)=R\in\mathscr{H}_C$ is a hypergraph with $type(R)=type(A)$. If $|ext_R|=|V_R|$ and $E_R=\emptyset$, then p is said to be *empty*.

Let $H\in\mathscr{H}_C$ and let $p=(lab(e),R)$, with $e\in E_H$, then a *direct derivation* $H\Rightarrow_p H'$ is obtained by the replacement $H'=H[e/R]$.

A sequence d of direct derivations $H_0\Rightarrow_{p_1}\cdots\Rightarrow_{p_k}H_k$ of length k with $(p_1,\ldots,p_k)\in P$ is denoted as $H\Rightarrow^k H_k$ or $H\Rightarrow_P^* H_k$ if the length is not relevant. We denote it as $H\Rightarrow^* H_k$ if the sequence is clear from the context.

A derivation $H\Rightarrow^* H'$ of length 0 is given if $H\cong H'$.

Given an ordered set $\{\alpha_1,\ldots,\alpha_n\}$ where $a_i<a_j$ if $i<j\in\mathbb{N}$ we define a *hyperedge replacement grammar*, or *HRG* as a tuple $G=(N,\Sigma,P,S,(mark_p)_{p\in P})$ where $N\subseteq C$ is a finite set of non-terminal labels, $\Sigma\subseteq C$ is a finite set of terminal labels with $N\cap\Sigma=\emptyset$, P is a finite set of productions, $S\in N$ is the starting symbol, $(mark_p)_{p\in P}$ is a family of functions $mark_p:E_R\to\{\alpha_1,\ldots,\alpha_n\}$ assigning a mark to each hyperedge in the right-hand side of a production p (Figure 2). For each pair $e_i,e_j\in E_R$ with $i\neq j$, $mark(e_i)\neq mark(e_j)$.

We denote as $P^A\subseteq P$ the subset of productions where $lhs(p)=A$. We call a production $p=(A,R)\in P_N\subseteq P$ *non-terminal* if $E_R^N\neq\emptyset$ or *terminal* if $p=(A,R)\in P_\Sigma=P\backslash P_N$.

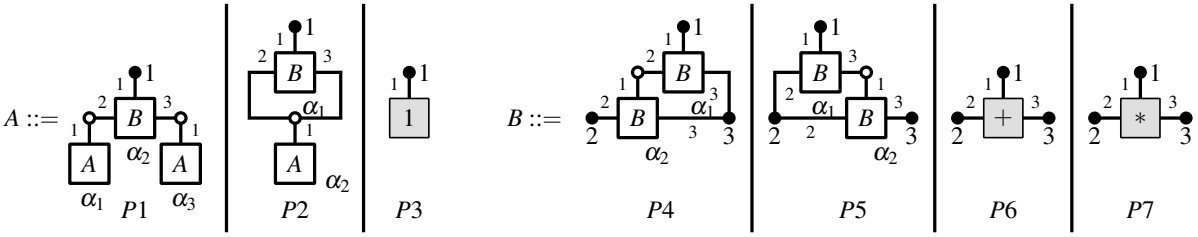


Figure 2: An ambiguous hyperedge replacement grammar for term graphs

The marking of the hyperedges in the *rhs* of each production, represents the order in which the replacements are carried out $(\alpha_1,\alpha_2,\ldots,\alpha_{n-1},\alpha_n)$. It allows for the definitions of ordered derivation tree and leftmost derivation.

Given a set of productions P, we denote by T_P the set of all ordered trees over P which is inductively defined as follows: for each $p\in P$, $p\in T_P$; for $t_1,\ldots,t_n\in T_P$ and $p\in P$, $p(t_1,\ldots,t_n)\in T_P$.

Then, given an *HRG* G, an *ordered derivation tree* t for e such that $lab(e)=X\in N$, is a tree $p(t_{\alpha_1},\ldots,t_{\alpha_n})$ in T_P, such that $p=(X,R)$ is a production in P, and $t_{\alpha_1},\ldots,t_{\alpha_n}$ are derivation trees for $e_1\ldots e_n$, such that $X_1\ldots X_n$ are the labels of the non-terminal hyperedges in R marked with $\alpha_1\ldots\alpha_n$, respectively (Figure 11).

We define the *yield* of an ordered derivation tree t, denoted with $yield(t)$, as the sequence of replacements: $yield(p(t_{\alpha_1},\ldots,t_{\alpha_n}))=rhs(p)[e_1/yield(t_{\alpha_1}),\ldots,e_n/yield(t_{\alpha_n})]$.

Let t be an ordered derivation tree for a hypergraph H obtained from a derivation $d=S^\bullet\Rightarrow_P^* H$ and $trav(t)$ its pre-ordered visit. Then d is said to be a *leftmost derivation*, denoted as $lmd(H)$, if and only if the order of the applied productions of d corresponds to $trav(t)$.

Since we need a measure to ensure the termination of the proposed algorithm, we define an *HRG*

to be *non-contracting* if for each direct derivation $H \Rightarrow_p H'$, $|H| \leq |H'|$. We call a grammar *essentially non-contracting* if there exists $p = (S,R) \in P$ such that p is the empty production.

The *hyperedge replacement language* (*HRL*) generated by an *HRG* is the set $L(G) = \{H \in \mathscr{H}_\Sigma \mid S^\bullet \Rightarrow_P^* H\}$. We define for each $A \in N$, $L^A(G) = \{H \in \mathscr{H}_\Sigma \mid A^\bullet \Rightarrow_P^* H\}$. We also define for $n \in \mathbb{N}$, $L_n^A(G) = \{H \in \mathscr{H}_\Sigma \mid A^\bullet \Rightarrow_P^* H \wedge |H| = n\}$. Clearly $L_n^A(G) \subseteq L^A(G)$. We denote as $|L_n^A|$ the size of the set of all size-n-hypergraphs in L that can be derived from $A^\bullet$.

For example, the hyperedge replacement grammar in Figure 2 generates the class of all term graphs with function symbols in $\{*,+,1\}$. Note that hyperedges with non-terminal labels are depicted as white boxes. A derivation with the Chomsky normal form version of this grammar is given in Figure 10.

We define a grammar G to be *ambiguous* if there are ordered derivation trees $t_1, t_2 \in T_P$, such that $t_1 \neq t_2$ and $yield(t_1) \cong yield(t_2)$, or equivalently, if there exist $H, H' \in L(G)$ such that $H \cong H'$ and $lmd(H) \neq lmd(H')$. If $yield(t_1), yield(t_2) \in L_n(G)$ we say that G is *n-ambiguous*. A non-ambiguous version of the term graphs grammar is given in Figure 3.

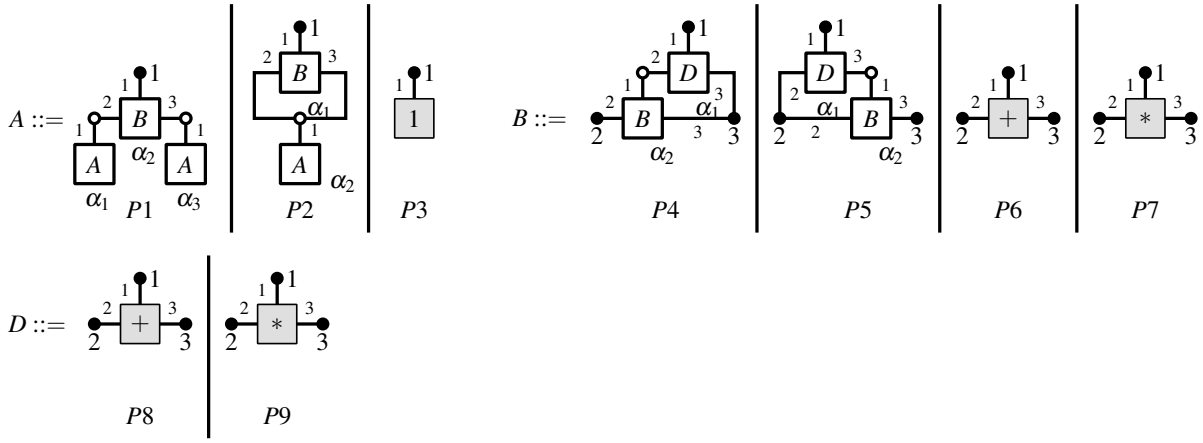


Figure 3: A non-ambiguous hyperedge replacement grammar for term graphs

3 Random hypergraph generation

In 1994, Mairson proposed a pair of methods for the sampling of strings from context-free grammars [11] . His approach requires, as input, a grammar G in Chomsky normal form and the length n of the word to be generated. He proves that, if G is non-ambiguous, such a word is generated uniformly at random. The first method has a time complexity of $O(n^2)$ while requiring $O(n)$ space, and vice versa, the second method runs in linear time using quadratic space. In the following we adapt the first of Mairson's methods to hyperedge replacement grammars. We use our running example of a term graph language to illustrate the generation process.

We define a *Chomsky normal form* (*CNF*) for hyperedge replacement grammars as a tuple $G_{CNF} = (N, \Sigma, P, S, (mark_p)_{p \in P})$ where:

- $N \subseteq C$ is a finite set of non-terminal labels
- $\Sigma \subseteq C$ is a finite set of terminal labels with $N \cap \Sigma = \emptyset$
- P is a finite set of productions
- $S \in N$ is the starting symbol

- $(mark_p)_{p\in P}$ is a family of functions $mark_p : E_R \to \{\alpha,\beta\}$ assigning a mark to each hyperedge in the right-hand side of a production p

Each production $p = (A,R) \in P$ satisfies one of the following constraints:

- $E_R = \{e_1,e_2\}$ where $lab(e_1), lab(e_2) \in N$ and $mark(e_1) \neq mark(e_2)$, in which case the replacement is firstly carried out on the hyperedge marked with α, then on the one marked with β
- $E_R = \{e_1\}$ where $lab(e_1) \in \Sigma$ and $mark(e_1) = \alpha$
- $E_R = \emptyset$, $|V_R| > |ext_R|$
- $A = S$, p is the empty production and for each $q \in P$, for each $e \in rhs(q)$, $lab(e) \neq S$

Note that in the first two cases, $rhs(p)$ contains either exactly two non-terminal hyperedges or a single terminal hyperedge and may also contain isolated nodes. Productions according to the third case are considered as *terminal* productions. The last case specifies that the empty production is only allowed if there is no other production having the starting symbol in its right-hand side. The grammar in Figure 4 is the *CNF* version of the term graph grammar in Figure 2.

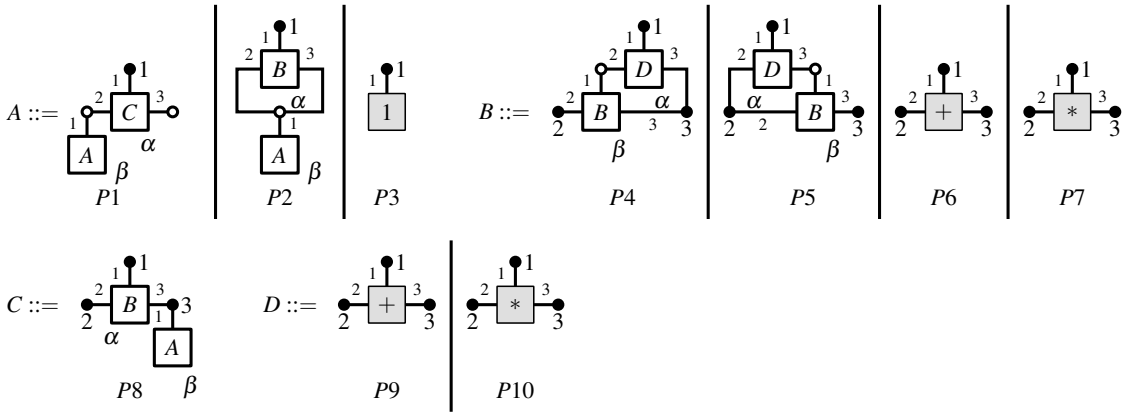


Figure 4: *CNF* of the grammar in Figure 2

Lemma 3.1. *There exists an algorithm that for every hyperedge replacement grammar G produces a grammar G' in CNF such that $L(G) = L(G')$.*

Proof. We present a set of rules to transform any grammar G, into an equivalent grammar G' such that, for each direct derivation $H \Rightarrow_p H'$ with $p \in P_G$, it exists an equivalent derivation $H \Rightarrow^*_Q H'$ with $Q \subseteq P_{G'}$. The proof is provided with a running example showing the application of the rules. The grammar in Figure 5 contains productions that are not in *CNF*: $P1$ has more than 2 hyperedges; $P2$ has a single non-terminal hyperedge; $P3$ is an empty production, but its *lhs* is not S; $P4$ has 2 hyperedges one of which is terminal.

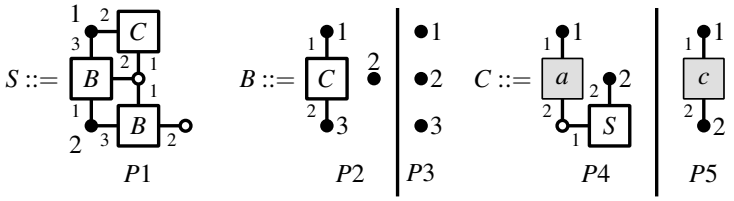


Figure 5: Starting grammar for the proof of *CNF* equivalence.

For a production $p = (A,R) \in P$, that is not already in *CNF*, we consider the following set of rules, applied in this order, to obtain a corresponding equivalent set of productions P' in *CNF*:

1. If p is the empty production, for each production $q = (B,X) \in P$ having $e \in E_X$ with $lab(e) = A$ in its *rhs*, for each production $q' = (A,Y) \in P$ having A in its *lhs* we apply the substitution $R' = X[e,Y]$ and add the productions $p = (B,R')$. We then remove the productions that are no longer needed. The proof of equivalence of the derivations $H \Rightarrow_q H' \Rightarrow_{q'} H''$ and $H \Rightarrow_{p'} H''$ is the following: if e' with $lab(e') = B$ is the hyperedge involved in the derivation $H \Rightarrow_q H'$ then $H'' = H[e'/X[e/Y]] = H[e'/R']$ since $R' = X[e,Y]$.

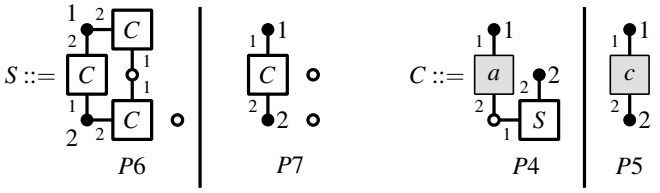


Figure 6: Removal of the empty production $P3$.

In order to remove the empty production $P3$ (Fig. 6) we apply the replacements of all the productions having B as their *lhs* to all the productions having a hyperedge labelled as B in their *rhs*. We remove $P1$ and introduce the productions $P6$ and $P7$. We then remove $P2$ and $P3$ since they are no longer needed.

2. If $E_R = \{e'\}$ with $lab(e') \in N$ for each production $q = (lab(e'),X) \in P$ we add the production $p' = (lab(e),R')$ with $R' = R[e'/X]$. If $E_{R'} = \{e''\}$ with $lab(e'') \in N$ this step is iterated and terminates when $|E_{R'}| > 1$ or $E_{R'} = \{e_t\}$ with $lab(e_t) \in \Sigma$ or $|E_{R'}| = 0$ and $|V_{R'}| > ext_{R'}$. The proof of equivalence of the derivations $H \Rightarrow_p H' \Rightarrow_q H''$ and $H \Rightarrow_{p'} H''$ is the following: if e' is the hyperedge involved in the derivation $H' \Rightarrow_q H''$ then $H'' = H'[e/R[e'/X]] = H[e/R']$ since $R' = R[e'/X]$.

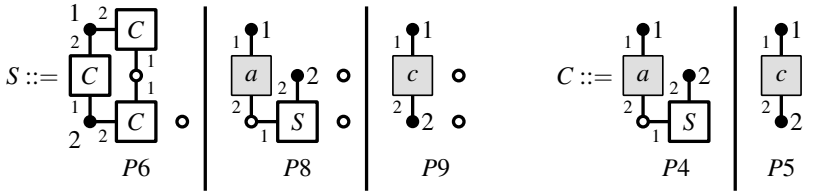


Figure 7: Removal of production $P7$

Since $P7$ has a single non-terminal hyperedge C (Fig. 7), we apply a replacement for each production that has C on its *lhs*. In our case, using the replacements of $P4$ and $P5$, we obtain $P8$ and $P9$. The production $P7$ is removed from the grammar.

3. If $|E_R| = k > 2$ we consider the subgraph X of R composed by the subset $E_X \subset E_R$ of hyperedges $e_2,\ldots,e_k$ and their attachment nodes. We introduce a new label T so that $N' = N \cup \{T\}$ and a new handle $T^\bullet$ of e_T with $ext(e_T) = \bigcup_{2 \le i \le k} att(e_i)$ such that $type(e_T) = type(X)$. We then consider the hypergraph R' composed by $R \backslash X$ and $T^\bullet$ where $V_{R'} = V_{R\backslash X} \cup V_{T^\bullet}$ and $E_{R'} = E_{R\backslash X} \cup E_{T^\bullet}$. Finally we add the productions $p' = (A,R'), p'' = (T,X)$ to P'. If $|E_X| > 2$ this step is iterated. The proof of equivalence of the derivations $H \Rightarrow_p H'$ and $H \Rightarrow^*_{P'} H'$ is the following: if e_a is the handle

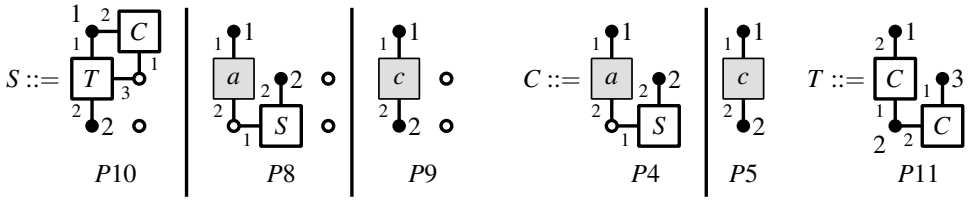


Figure 8: Removal of production $P6$

of the *lhs* of p we consider the following equivalence of the replacements then $H' = H[e_a/R] = H[e_a/R'[e_T/X]]$ since $R = R'[e_T/X]$.

Since production $P6$ has three non-terminal hyperedges (Fig. 8), we create a new label T, a new handle $T^\bullet$ and the production $P11$. Then we add the production $P10$ so that the replacement of the hyperedge labelled as T by the *rhs* of $P11$ results in the *rhs* of $P6$. The production $P6$ is then removed from the grammar.

4. If $|E_R| > 1$ and exists $e' \in E_R$ such that $lab(e') \in \Sigma$ a new label T is introduced so that $N' = N \cup \{T\}$. We add 2 new productions $p' = (A, R')$ to P' where $R' = R$ with $lab(e') = T$ and $p'' = (T, e'^\bullet)$. This step is repeated for each $e' \in E_R$ with $lab(e') \in \Sigma$. Due to the confluence property [3] of *HRG*s the order in which the terminal hyperedges are chosen is irrelevant. The proof of equivalence of the derivations $H \Rightarrow_p H''$ and $H \Rightarrow_{p'} H' \Rightarrow_{p''} H''$ is the following: if $e' \in E_R$ with $lab(e') \in \Sigma$ is the hyperedge involved in the derivation $H \Rightarrow_p H''$ then $H' = H[e/R] = H[e/R'[e'/e'^\bullet]]$ since $R = R'[e'/e'^\bullet]$.

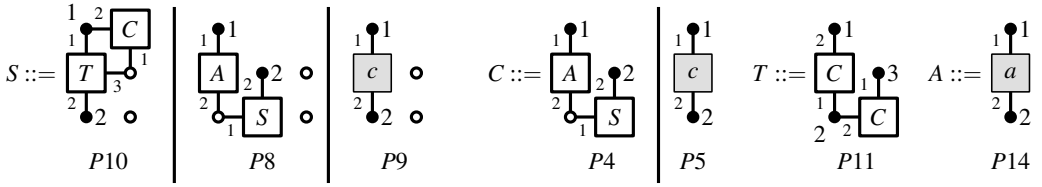


Figure 9: Removal of productions $P8$ and $P4$

Both *rhs* of productions $P4$ and $P8$ are composed by a terminal and a non-terminal hyperedge. We introduce a new label A and a its handle $A^\bullet$ along with the production $P14$ (Figure 9). We then add the productions $P12$ and $P13$ resulting from the substitution of the terminal hyperedges labelled with a by the non-terminal hyperedges labelled with A. Productions $P4$ and $P8$ are then removed from the grammar.

□

From this point on, if not explicitly specified, we always refer to an *HRG* as an *HRG* in *CNF*. We stress that the input of the method must be already provided in this form, that is, the time required for the transformation is not taken into account during the evaluation of the time complexity.

In order to complete the adaptation of the grammar we propose a more suitable short-hand representation of the productions that only extracts the necessary information, so, for each $p = (A, R) \in P$ and $i \in \mathbb{N}_0$ we use the following notations:

- $A \xrightarrow{p} BC, i$ for a *non-terminal* production where $B, C \in N$ are the labels of the marked hyperedges $e_\alpha, e_\beta \in R$ with $mark(e_\alpha) = \alpha$, $mark(e_\beta) = \beta$ and $i = |V_R \backslash ext_R|$.

- $A \xrightarrow{p} a, i$ for a *terminal* production where $a \in \Sigma$ is the label of the marked hyperedge $e_\alpha \in R$ and $i = |V_R \backslash ext_R|$.
- $A \xrightarrow{p} \lambda, i$ for a *terminal* production where $E_R = \emptyset$ and $i = |V_R \backslash ext_R|$.

Matrix M_2 (Tab. 1) shows the short-hand representation of the productions of the grammar in Figure 4. Considering a second input n as the size of the hypergraph to be generated, we are ready to describe a pair of algorithms (**Pre**, **Gen**) for the random sampling of a hypergraph H from a grammar G. Such a hypergraph is sampled in $L_n^A(G)$, where $A \in N$ is the non-terminal we begin the sampling from. If $A = S$ and G is n-unambiguous, H is sampled uniformly at random among all the hypergraphs in $L_n(G)$.

3.1 Pre-processing phase

The Pre-processing phase is used to construct a pair of matrices M_1, M_2 needed in the generation phase. Let $G = (N, \Sigma, P, S, (mark_p)_{p \in P})$ be an *HRG*, let $n \in \mathbb{N}$ be the size of the hypergraph $H \in L_n(G)$ we would like to generate, then the algorithm **Pre** (Alg. 1) produces the structures required for the generation.

Table 1: Matrices M_1 and M_2 resulting from $\mathbf{Pre}(G', 12)$

M_1

N	1	2	3	4	5	6	7	8	9	10	11	12
A	1	0	2	0	14	0	92	0	616	0	3920	0
B	2	0	8	0	32	0	128	0	256	0	512	0
C	0	0	2	0	32	0	76	0	488	0	2928	0
D	2	0	0	0	0	0	0	0	0	0	0	0

M_2

P	1	2	3	4	5	6	7	8	9	10	11	12
$A \xrightarrow{P1} CA, 1$	0	0	0	0	2	0	16	0	128	0	992	0
$A \xrightarrow{P2} BA, 1$	0	0	2	0	12	0	76	0	488	0	2928	0
$B \xrightarrow{P4} DB, 1$	0	0	4	0	16	0	64	0	128	0	256	0
$B \xrightarrow{P5} DB, 1$	0	0	4	0	16	0	64	0	128	0	256	0
$C \xrightarrow{P8} BA, 1$	0	0	2	0	12	0	76	0	488	0	2928	0
$A \xrightarrow{P3} 1, 0$	1	0	0	0	0	0	0	0	0	0	0	0
$B \xrightarrow{P6} +, 0$	1	0	0	0	0	0	0	0	0	0	0	0
$B \xrightarrow{P7} *, 0$	1	0	0	0	0	0	0	0	0	0	0	0
$D \xrightarrow{P9} +, 0$	1	0	0	0	0	0	0	0	0	0	0	0
$D \xrightarrow{P10} *, 0$	1	0	0	0	0	0	0	0	0	0	0	0

We begin initializing the entries of two matrices $M_1 = (N \times \mathbb{N})$ and $M_2 = (P \times \mathbb{N})$ to 0. Each entry (A, ℓ) of M_1, also denoted as $A[\ell]$, represents the number of derivations yielding a hypergraph of size $\ell + type(A)$, from a non-terminal $A \in N$. Each entry (p, ℓ) of M_2, also denoted as $p[\ell]$ represents the number of derivations yielding a hypergraph of size $\ell + |ext_R|$, from a production $p \in P$. According to the type of production they are also denoted as $A \xrightarrow{p} \lambda, i[\ell]$ or $A \xrightarrow{p} a, i[\ell]$ for terminal productions and $A \xrightarrow{p} BC, i[\ell]$ for a non-terminal production. Considering each terminal production $p \in P_T$, either yielding a single terminal hyperedge $A \xrightarrow{p} a, i$ or at least a single isolated node $A \xrightarrow{p} \lambda, i$, the corresponding M_2 entry $p[i+1]$ in the former case, or $p[i]$ in the latter, is set to 1. Then, for each $\ell \in \mathbb{N}$ in $1 \leq \ell \leq n$, for each non-terminal $A \in N$, $A[\ell] = \sum_{p \in P^A} p[\ell]$ and for each production $p \in P_N$, $p[\ell] = \sum_{0<k<\ell} B, [k] \cdot C[\ell - k]$.

The matrices can be used to generate hypergraphs in $L^A(G)$ of size $\ell + type(A)$, with $1 \leq \ell \leq n$ from any non-terminal $A \in N$. If the non-terminal A is chosen before the pre-processing phase we can reduce the size of the tables to $n - type(A)$. Table 1 shows the result of running the algorithm **Pre** using the grammar G' in Figure 4 and a size of 12 as input.

3.2 Generation phase

In the generation phase a non-terminal $\bar{A} \in N$ is chosen and a size-$\bar{n}$-hypergraph H, with $1 \leq \bar{n} \leq n + type(A)$, is generated using the data collected in the matrices M_1, M_2 and a pseudo-random number generator *RNG*. The algorithm **Gen** (Alg. 2) describes this process.

On input $\mathbf{Gen}(G, \langle M_1, M_2 \rangle, \bar{A}, \bar{n} - type(A))$, if $\bar{A}[\bar{n} - type(A)] = 0$ the generating algorithm fails, otherwise, having $\bar{A}^\bullet$ as a basis, the algorithm recursively calls the function **derH** proceeding through the following steps:

1. The *RNG* is used to choose a production $p \in P^A$ with probability $p[\ell]/A[\ell]$.
2. If $p \in P^A_\Sigma$, the replacement of e, the *handle* of A, with the hypergraph R in $rhs(p)$ is returned.
3. If $p \in P^A_T$ the *RNG* is used again to choose a "split" $0 < k < \ell'$ with $\ell' = \ell - i$ and probability $B[k] \cdot C[\ell' - k]/A \xrightarrow{p} BC, i[\ell]$. The hypergraph $rhs(p)[e_\alpha/\mathbf{derH}(B,k), e_\beta/\mathbf{derH}(C, \ell' - k)]$ produced by the replacement of e_α with the result on the recursive function on input $\mathbf{derH}(B,k)$ and the replacement of e_β with the result of the recursive function on input $\mathbf{derH}(C, \ell' - k)$ is computed. Then, the replacement of the hyperedge e, the *handle* of A, with the aforementioned hypergraph is returned. We use the notation $B_k C_{\ell' - k}$ to indicate such a split.

The derivation $d = A^\bullet \Rightarrow^*_P H$ in Figure 10 corresponds to the sequence of replacements computed by the recursive function **derH** to generate the size-12-hypergraph H in Figure 1, using non-terminal A as input. For each step we show the probability of the production p to be chosen and the choice of the split and its probability if $p \in P^N$. Since G is non-ambiguous, the first step shows that $|L_{12}(G)| = 3920$, that is, there are 3920 unique size-12-hypergraphs to choose from, each having a different ordered derivation tree. Figure 11 shows the tree t for which $yield(t) = H$, so that $trav(t)$, or equivalently $lmd(H)$, corresponds to the unique sequence of productions applied by the generation algorithm to produce H. In the figure are also indicated the starting symbol A and the replaced hyperedges e_α and e_β, respectively on the edges connecting the left and right child of each node. The proof of termination of the Generation algorithm is based on the assumption that the input grammar is non-contracting:

Algorithm 1: Pre - Pre-processing phase

Input: (G,n), where $G = (N, \Sigma, P, S, (mark_p)_{p \in P})$ and $n \in \mathbb{N}$, $n \geq 1$
Output: $\langle M_1, M_2 \rangle$

```
for 1 ≤ ℓ ≤ n do
    foreach A ∈ N do
        A[ℓ] := 0;
    end
    foreach p ∈ P do
        p[ℓ] := 0;
    end
end
foreach A --p--> a, i ∈ P_Σ do
    A --p--> a, i[i+1] := 1;
end
foreach A --p--> λ, i ∈ P_Σ do
    A --p--> λ, i[i] := 1;
end
for 1 ≤ ℓ ≤ n do
    foreach A ∈ N do
        foreach p ∈ P^A do
            A[ℓ] := A[ℓ] + p[ℓ];
        end
    end
    foreach A --p--> BC, i ∈ P_N do
        for 1 ≤ k < ℓ do
            A --p--> BC, i[ℓ+i] := A --p-->
              BC, i[ℓ+i] + B[k] · C[ℓ−k];
        end
    end
end
```

Proof. Let's consider a measure equivalent to the size of a hypergraph $|H|$. To each application of the recursive function **derH** in each step of the algorithm **Gen**, corresponds a direct derivation between two

Algorithm 2: Gen - Generation phase

Input: $(G, \langle M_1, M_2\rangle, \bar{A}, \bar{n})$, where $G = (N, \Sigma, P, S, (mark_p)_{p\in P})$, $\langle M_1, M_2\rangle := \mathbf{Pre}(G, n)$, $\bar{A} \in N$ and $\bar{n} \in \mathbb{N}$, $1 \leq \bar{n} \leq n + type(\bar{A})$

Output: $H \in L^{\bar{A}}_{\bar{n}}(G)$

$\ell = \bar{n} - type(\bar{A})$
if $\bar{A}[\ell] = 0$ **then**
  return $\perp$;
end
Recursively generate H using $(\bar{A}, \ell)$ as first input as follows:
function derH (A, ℓ)**:**
  $p \longleftarrow RNG$ with $p \in P^A$ and probability $p[\ell]/A[\ell]$;
  if $p \in P_T$ **then**
    return $A^\bullet[e/R]$;
  else
    $\ell' = \ell - i$;
    $k \longleftarrow RNG$ with $0 < k < \ell'$ and probability $B[k] \cdot C[\ell' - k]/(A \xrightarrow{p} BC, i)[\ell]$;
    return $A^\bullet[e/R[e_\alpha/\mathbf{derH}(B,k), e_\beta/\mathbf{derH}(C, \ell' - k)]]$;
  end
end function

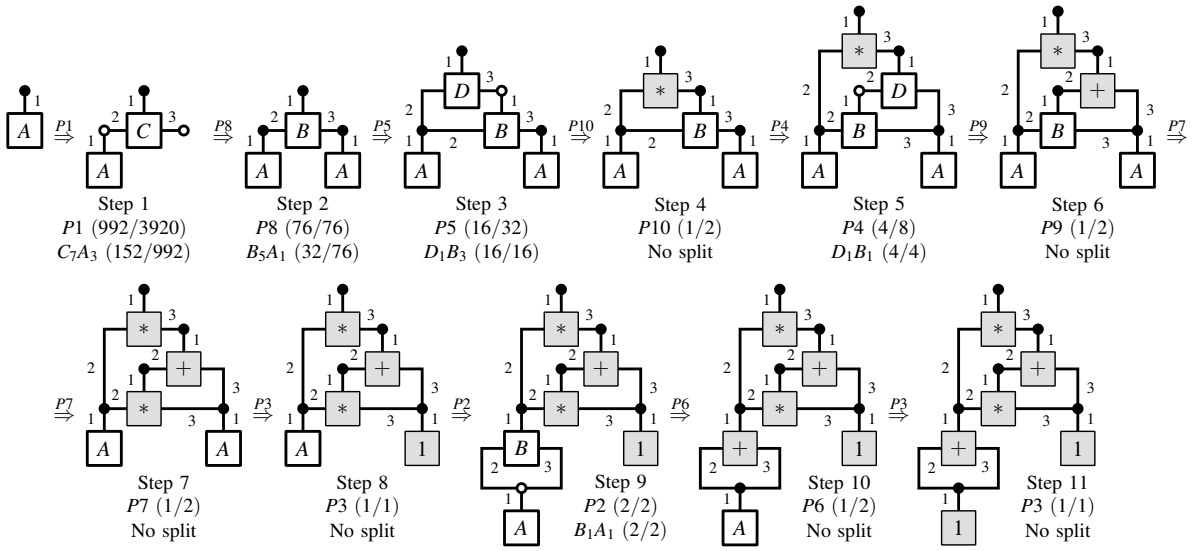


Figure 10: A derivation $d = A^\bullet \Rightarrow^*_P H$ using the grammar of Figure 4

sentential forms $F \Rightarrow F'$ such that $F \leq F'$. Since the grammar is in *CNF*, at each step there are two possible cases:

1. **derH** chooses a non-terminal production. In this case a single hyperedge $e \in F$ is replaced with a hypergraph $R \subseteq F'$ containing 2 hyperedges and 0 or more internal nodes. Clearly $|F| < |F'|$, meaning that the size of the sentential forms gets progressively close to n.
2. **derH** chooses a terminal production. A hyperedge is replaced by a terminal hyperedge or a single node and 0 or more additional internal nodes. In this case $|F| \leq |F'|$. Even if the size is not incremented, being a terminal production, the recursion does not progress any further.

If it is not possible to generate a size-n-hypergraph using the input grammar G the algorithm trivially ends in one step. □

4 Uniform distribution and time complexity

We now state our first main result, the uniform generation guarantee for Algorithm 2.

Theorem 4.1. *Given a grammar $G = (N, \Sigma, P, S, (mark_p)_{p \in P})$, Algorithm 2 generates from every non-terminal $A \in N$ a size-n-hypergraph $H \in L_n^A(G)$, provided that $L_n^A(G) \neq \emptyset$. If G is n-unambiguous and RNG is a uniform random number generator, the hypergraph is chosen uniformly at random.*

Proof. Let G be an n-unambiguous grammar in *CNF*, the recursive function **derH** derives a hypergraph $H \in L_{\bar{n}}^{\bar{A}}(G)$ simulating $trav(t)$ where $yield(t) = H$ and let $P(c_j)$ denote the probability of the jth choice c made using the *RNG* at each step of the recursion, for a production or a split, according to $lmd(H)$.

Let's recall that for the parallelization, confluence and associativity properties of context-free hyperedge replacement grammars [3], the sequence of replacements associated to a derivation preserves the result of the derivation, despite of the order in which the replacements are applied. Thus, we are able to discuss each of its steps independently.

By definition, since the grammar is n-unambiguous, for any non-terminal $A \in N$ we know that the set of hypergraphs that can be generated using different productions $p \in P^A$ are pairwise distinct. Otherwise, there would exist $trav(t') \neq trav(t'')$ for which $yield(t') \cong yield(t'')$.

From algorithm **Pre** (Alg. 1) we know that $\sum_{p \in P^A} p[\ell] = A[\ell]$ and so the probability of the choice c_j of each production in $lmd(H)$ can be expressed by $P(c_j) = p[\ell]/A[\ell]$. Also, if $p \in P_N$, since the grammar is n-unambiguous the subsets of hypergraphs that can be derived by choosing different splits are also pairwise distinct. For a production $p \in P_N$ then $\sum_{0<k<\ell} B[k] \cdot C[\ell' - k] = A \xrightarrow{p} BC, i[\ell]$, thus a split can be chosen with probability $P(c_j) = B[k] \cdot C[\ell' - k]/p[\ell]$.

Knowing that for an *lmd*, if the grammar is n-unambiguous, both the choices of productions and splits are made from independent sets, considering the corresponding derivation tree t, the probabilities associated to the choice of a node $P(c)$ and the ones associated to its children $P(c')$ and $P(c'')$ are of the form $\frac{m}{q}$, $\frac{m'}{q'}$ and $\frac{m''}{q''}$ with $m, m', m'', q, q', q'' \in \mathbb{N}$ and $q'q'' = m$. Moreover, the probabilities of two consecutive choices $P(c)$ and $P(c')$ are bound to the law of compound probabilities [10], that is, the choice of a node given the choice of its parent is of the form $P(c'|c) = P(c' \cap c)/P(c)$. Then, considering their independence, $P(c'|c) = (P(c')P(c))/P(c) = P(c')$. The same applies for $P(c'')$. The overall probability of the choice of a node and its children is then $P(c)P(c')P(c'') = \frac{m}{q}\frac{m'}{q'}\frac{m''}{q''} = \frac{m'm''}{q}$.

Finally, considering the chain of probabilities described by an *lmd*, since for $\bar{A}$ $q = |L_{\bar{n}}(G)|$ and for each terminal production $p \in P_\Sigma$ $m = 1$, then for each $H \in L_{\bar{n}}(G)$ we can define its probability $P(H)$ to be generated as the productory of independent choices:

$$P(H) = \prod_{j=1}^{k} P(c_j) = \frac{m_1}{|L_{\bar{n}}(G)|} \cdot \frac{m_2}{q_2} \cdot \frac{m_{k-1}}{q_{k-1}} \cdots \frac{1}{q_k} = \frac{1}{|L_{\bar{n}}(G)|}$$

Each hypergraph $H \in L_{\bar{n}}(G)$ is generated over a uniform distribution given the uniformity of the sampling of the underlying *RNG*.

□

For the complexity analysis we consider the time required by the algorithm **Gen** (Alg. 2) for the generation of the hypergraph and the space required by the algorithm **Pre** (Alg. 1) to store the required data, taking into account that the input grammar is already provided in the correct *CNF* and the query to the *RNG* and the replacement operations are performed in unit time. The gaps present in the tables, that are not encountered in string method, are due to the possibility of a production to increase the size of the resulting hypergraph by more than 1 in a single step.

Theorem 4.2. *With the assumptions of Theorem 4.1, the size-n-hypergraph H is generated by* ***Gen*** *(Alg. 2) in time* $O(n^2)$.

Proof. The proof of Theorem 4.2 is based on the analysis of the following recurrence relation for the function **derH**: $T(n) \leq cn + \max_{1 \leq k < (n-i)} [T(k) + T(n-k-i)]$, where $T(k)$ and $T(n-k-i)$ are the computational steps required to process the result of the split and i is the number of internal nodes of the current production. In the worst case, we consider that $i = 0$ and that $k = 1$. A simple example is the discrete hypergraph language in which every iteration may generate a terminal hyperedge from e_α and the rest of the resulting hypergraph from e_β without adding any node. Since the choice of the production is constant, while the choice of a split is linear in n, choosing a split n times leads to a quadratic behavior.

Since $i \ll n$, we may rewrite the recursion as:

$$T(n) \leq cn + \max_{1 \leq k < n} [T(k) + T(n-k)]$$

Then, considering the worst case $k = 1$, for the next step of the recursion we obtain:

$$T(n-1) \leq c(n-1) + \max_{1 \leq k < (n-1)} [T(k) + T(n-k-1)]$$

That is, at each step the choice of a split happens on an input of size $n-1$. Since this choice requires linear time and it is taken n times, the relation has solution $O(n^2)$. □

We omit a discussion of the time complexity of the pre-processing phase (Alg. 1) which can be shown to be linear, considering that given a grammar G in *CNF*, being its size $|G|$ constant, for each production p a short form containing the information about the labels and the internal nodes is obtained in constant time.

5 Conclusion

Our main results, presented in Section 4, are that the method generates hypergraphs uniformly at random and in quadratic time. A topic for future work is to design an alternative generation algorithm that runs in linear time and quadratic space, following Mairson's second method in [11].

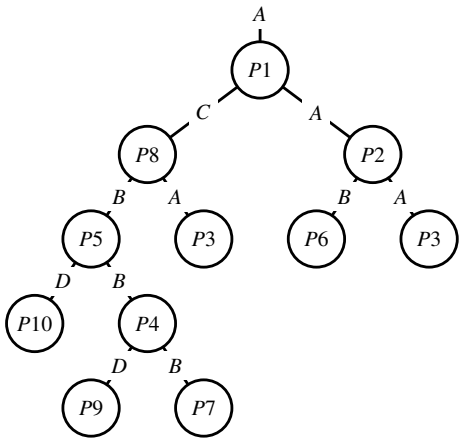


Figure 11: Ordered tree t for the derivation d in Figure 10

Another interesting topic is to extend the quasi-polynomial-time approximation algorithm of Gore et al. [7] from strings to hypergraphs. This algorithm guarantees an approximated uniform distribution even for ambiguous grammars.

Our method allows to generate strings uniformly at random in some non-context-free string languages because hyperedge replacement grammars can specify certain *string graph* languages that are not context-free. For example, this applies to the language $\{a^n b^n c^n \mid n \geq 0\}$. Moreover, our method is able to generate strings uniformly at random for a range of inherently ambiguous context-free languages.

The practically most promising application of our generation approach is the testing of programs in arbitrary programming languages that work on graphs. If the inputs of such programs are graphs in a context-free graph language, our method can generate test graphs uniformly at random in the domain of interest. This should allow to refine random testing approaches such as [1, 8].

References

[1] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. Tse. Adaptive random testing: The art of test case diversity. *Journal of Systems and Software*, 83(1):60–66, 2010. doi:10.1016/j.jss.2009.02.022.

[2] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959. doi:10.1016/S0019-9958(59)90362-6.

[3] B. Courcelle. An axiomatic definition of context-free rewriting and its application to NLC graph grammars. *Theoretical Computer Science*, 55(2):141–181, 1987. doi:10.1016/0304-3975(87)90102-2.

[4] F. Drewes, A. Habel, and H.-J. Kreowski. Hyperedge replacement graph grammars. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1, pages 95–162. World Scientific, 1997. doi:10.1142/3303.

[5] J. Engelfriet. Context-free graph grammars. In *Handbook of Formal Languages*, volume 3, pages 125–213. Springer, 1997. doi:10.1007/978-3-642-59126-6_3.

[6] O. Goldreich. Candidate one-way functions based on expander graphs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 76–87. Springer, 2011. doi:10.1007/978-3-642-22670-0_10.

[7] V. Gore, M. Jerrum, S. Kannan, Z. Sweedyk, and S. Mahaney. A quasi-polynomial-time algorithm for sampling words from a context-free language. *Information and Computation*, 134(1):59–74, 1997. doi:10.1006/inco.1997.2621.

[8] R. Hamlet. Random testing. In *Encyclopedia of Software Engineering*. John Wiley and Sons, 2002. doi:10.1002/0471028959.sof268.

[9] H. Kajino. Molecular hypergraph grammar with its application to molecular optimization. In Proceedings *36th International Conference on Machine Learning (ICML 2019)*, volume 97 of *Proceedings of Machine Learning Research*, pages 3183–3191. PMLR, 2019. URL `https://proceedings.mlr.press/v97/kajino19a.html`.

[10] P. S. le Marquis de Laplace. Théorie analytique des probabilités. In *Œuvres Completes de Laplace*, volume 7, pages 181–192. Gauthier-Villars, Imprimeur-Librarie, 3rd edition, 1820. URL `http://eudml.org/doc/203444`.

[11] H. G. Mairson. Generating words in a context-free language uniformly at random. *Information Processing Letters*, 49(2):95–99, 1994. doi:10.1016/0020-0190(94)90033-7.

[12] S. Micali and R. L. Rivest. Transitive signature schemes. In Proceedings *Topics in Cryptology (CT-RSA 2002)*, volume 2271 of *Lecture Notes in Computer Science*, pages 236–243. Springer, 2002. doi:10.1007/3-540-45760-7_16.

[13] D. Plump. Term graph rewriting. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, pages 3–61. World Scientific, 1999. doi:10.1142/9789812815149_0001.