Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 122 (2005) 23–65

www.elsevier.com/locate/entcs

A Language For Multiplicative-additive Linear Logic [1]

J.R.B. Cockett [2] C.A. Pastro [3]

Department of Computer Science, University of Calgary
2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4

Abstract

A term calculus for the proofs in multiplicative-additive linear logic is introduced and motivated as a programming language for channel based concurrency. The term calculus is proved complete for a semantics in linearly distributive categories with additives. It is also shown that proof equivalence is decidable by showing that the cut elimination rewrites supply a confluent rewriting system modulo equations.

Keywords: multiplicative-additive linear logic, linearly distributive categories, term logic, rewriting modulo equations, process semantics

1 Introduction

The main result of this paper is a decision procedure for the equality of proofs in the multiplicative-additive fragment of linear logic, where we consider this fragment to include all of the units. This is achieved by introducing a term logic, which is of independent interest, for this fragment of linear logic. Before introducing this term logic, however, it is perhaps worth sharing with the reader how we arrived at the language and why we think it is of some interest.

The idea of having a term logic for linear logic is certainly not new. In fact, almost every researcher who has been heavily involved in trying to un-

[1] Research partially supported by NSERC, Canada. Diagrams were produced with the XY-pic package of K. Rose and R. Moore and inferences with M. Tatsuya's `proof.sty`.

[2] Email: robin@cpsc.ucalgary.ca

[3] Email: pastroc@cpsc.ucalgary.ca

doi:10.1016/j.entcs.2004.06.049

derstanding the proof theory of linear logic has found it necessary to invent [4] such a calculus. Perhaps the earliest attempts at term logics for monoidal categories were by Lambek [14] and Jay [12]. They realized that ordinary algebraic terms with no variable copying or elimination would do the trick. More sophisticated attempts followed which linked the term calculus to the proof theory of various fragments of linear logic, see, e.g., Abramsky [1] and Benton, Bierman, de Paiva, and Hyland [3].

An important component of Girard's approach to linear logic [9] was the introduction of proof nets. These, of course, can also be regarded as a term logic in their own right and, indeed, in [4], were explicitly introduced as such. There, after straightening out Girard's one-sided proof nets into a two-sided form, they were used as a basis for solving the coherence problem for the units. From the point of view of a term logic, however, there is something peculiar about these nets as their definition is not local: one has to check a global correctness criterion before one can conclude that the net is valid. This condition arises as, in checking that the net corresponds to a valid proof, one has to determine that there is a valid way of assigning "scopes" to the inference rules. That this can be expressed as a combinatoric condition on the nets was, of course, one of Girard's key insights.

An interesting recent approach to providing a term logic was suggested by Koh and Ong [13]. They realized that the tricky rewiring conditions for the units which arose in [4] could be expressed directly and quite clearly with scope changing rules. The first author was very fortunate to have visited Koh and Ong in Oxford in 1996 and to have had a chance to discuss this term logic with them. He was, of course, particularly impressed by the fact that they had realized that this gave a natural term logic for linearly distributive categories. It was clear that they had a good idea. However, their term logic never found any strong resonance in the linear logic community. This was not really surprising: after all, the proof net technology and its correctness criterion had been invented precisely to remove the necessity of keeping track of scopes. The reintroduction of explicit scopes seemed like a step in the wrong direction and made the utility of such a logic rather difficult to sell. To make matters worse, the syntax of their term logic was concise to the point of being cryptic: for an outsider the terms did not invite any particular insight into their meaning.

While trying to sort out a process semantics for multiplicative-additive linear logic (with both the additive and multiplicative units) we found that

[4] Inevitably sometimes the word "reinvent" should be used here. However, in defense of reinvention it should be said that, often, to reinvent is the only way to really understand these languages.

it was very useful to have a term logic to express the processes. The use of proof nets in linear logic makes it largely unnecessary to have a term logic for the multiplicative fragment. However, when one considers the multiplicatives together with the additives the value of a term logic becomes much more compelling. Even the minimalistic approach presented by Hughes and van Glabbeek in [11] cannot hide the fact that additive proof nets are complicated combinatoric structures which are hard to create in the way a programmer might create a program.

In order to produce such a term logic we took a very literal translation of the sequent calculus. When we showed this term logic to Robert Seely, and explained the intended process semantics, he suggested —while commenting ironically on the importance of appearances rather than content in certain academic circles— that we should try presenting it in a programming language style for processes. So we fiddled around with suitable keywords and produced such a language and were horrified to find that we actually liked the result! In particular, it had an immediate resonance with process programming which made the terms almost readable to programmers.

The point is that in presenting this term logic we are happy to claim almost no originality. In fact, we would suggest that the strength of the language is exactly that it has been plagiarized from a number of sources while slavishly following the sequent calculus. The scoping techniques should be quite recognizable from the term logic of Koh and Ong. Admittedly, we have unashamedly rearranged their terms to promote, following Robert's excellent suggestion, a process reading. Finally, centre stage is the syntax for the additives which we were happy to simply borrow from the π-calculus following the lead of Bellin and Scott [2]. The result is a term calculus which has the feel of a programming language for processes.

Indeed, if you had been trying to put together a process language for channel based communication you might well have come up with a language which has at least these features. That the language is, in fact, the internal language for linearly distributive categories with additives should be, perhaps, of more than a passing interest.

1.1 Introduction to the term calculus

As far as we know there has not actually been a proposal for a term calculus for the multiplicative-additive fragment of linear logic. We should, therefore, mention that although we borrow techniques very happily from prior work there are some technical aspects which remain. For example, the proof that term normalization modulo equations works is, for this fragment, technically more challenging than in the purely multiplicative fragment (compare our

techniques to those in [13]). While it is not possible to include all the technical details in this paper the reader should be aware that many of these details are available from the second author's master's thesis [15]. The subject of this thesis was the complete additive fragment of linear logic [5] for which a full and faithful process semantics was provided. This paper is part of the continuation of that work, whose aim is to provide a full and faithful process semantics for the complete multiplicative and additive fragments of linear logic [6].

A slight warning to linear logicians: we do not assume that we have a negation in the logic. This may seem to be an enormous omission, however, those intimately familiar with the coherence issues of these settings will know that, in fact, negation is a distraction whose presence or absence is orthogonal to the real issues. If this sounds like a controversial statement to the reader then we should perhaps also quietly mention that the initial model does actually have negation (i.e., it is a $*$-autonomous category) even though no negation is mentioned: in other words, this aspect, for the initial model, comes along for the ride anyway. This remains true whenever the generating polycategory has negation. Thus, we are just being scrupulously abstract in our approach and are therefore working at the level of linearly distributive categories with additives.

The objective of this paper is to highlight the term calculus we are using and to thereby persuade you that it is quite reasonable to give a process reading to the proofs of this fragment of linear logic. To this end we start with an old example of Girard's which involves obtaining a packet of Galois from a vending machine and show how to program it in our term logic.

We wish to describe the behavior of a vending machine which allows one to select either smokers chewing gum or, for those that cannot quite kick the habit, a packet of Galois cigarettes. The machine allows you to pay by inserting a one or two dollar coin. A packet of Galois costs two dollars while the gum only costs one dollar. There are four possible outcomes:

(i) You have inserted a dollar and have requested gum: you get a pack of gum.

(ii) You have inserted a dollar and requested a packet of Galois: your dollar is returned.

(iii) You have inserted two dollars and requested some gum: you get a pack of gum and a dollar in change.

[5] The complete additive fragment includes, in particular, the additive units which not surprisingly are more technically challenging to capture.

[6] The complete multiplicative and additive fragment includes both the multiplicative units and the additive units. In particular, it should be mentioned we are fundamentally not assuming "mix", thus the complications of handling the units in their full glory is present.

(iv) You have inserted two dollars and requested a packet of Galois: you get a packet of Galois.

This means we wish to produce a process which is the proof of the following sequent:

$$(1)\quad \alpha : \{1 : \$1, 2 : \$2\} \otimes \{gal : \top, gum : \top\} \vdash \beta : \{a : \mathsf{GAL}, b : \mathsf{GUM} \otimes \{\mathsf{change} : \$1, \mathsf{nochange} : \top\}, c : \$1\}$$

The process starts with a single input channel α and a single output channel β. Each channel has an attached protocol for the interaction down that channel. The curly brackets denote coproduct types with components which are tagged: the names used for the tags are called "events" and are sent and received as messages along the channels. The tensor operation "$- \otimes -$" allows one to bundle channels together.

To model the functioning of the vending machine we need three functions which transmute our resources:

$$\mathsf{gal} : \$2 \to \mathsf{GAL}, \quad \mathsf{gum} : \$1 \to \mathsf{GUM}, \quad \text{and} \quad \mathsf{gumch} : \$2 \to \mathsf{GUM} \otimes \$1.$$

Thus, gal turns two dollars into a packet of Galois, gum turns one dollar into a stick of gum, and gumch turns two dollars into a stick of gum and one dollar of change. The fairness of the machine is guaranteed by the fact that only these exchanges are allowed, e.g., we have to ensure that when the user inserts \$2 and asks for cigarettes he will not be given gum and a dollar change instead.

This process may be written as follows:

```
split α as α1, α2 in
  input α1 of
    | 1 ↦ input α2 of
        | gal ↦ close α2 in
              output c on β in α1 = β    % return dollar
        | gum ↦ close α2 in
              output b on β in
                fork β as
                  | β1 with α1 ↦ gum(α1; β1)    % return gum
                  | β2 with    ↦ output nochange on β2 in end β2
    | 2 ↦ input α2 of
        | gum ↦ close α2 in
              output a on β in gal(α1; β)    % return cig
```

```
| gum ↦ close α2 in
      output b on β in
        on γ plug
          gumch(α2; γ)    % return gum with dollar
        to
          split γ as γ1, γ2 in
            fork β as
              | β1 with γ1 ↦ γ1 = β1
              | β2 with γ2 ↦ output change on β2 in γ2 = β2
```

Input channels which are tensored can be used by a single process as they come from independent sources: to communicate down the individual channels one must `split` the channels. However, output channels when tensored together have to be treated completely differently: they can be dependent and so if they are used by the same process this may can cause deadlock or livelock. Thus, to use such channels one must `fork` the current process into two independent processes. When one forks a process in this manner one must also decide which of the open communication channels will be attached to which of the subprocesses.

One can only close a channel when the type is a unit ($\top$ or $\bot$). When it is an input channel one can simply `close` the channel $\top$ and when it is an output one can `end` the channel provided there are no other open channels.

The process above starts by splitting its one input channel α into two distinct channels. It then looks for an input event on the first channel: this is provided by the user inserting either \$1 or \$2 into the machine. Next the machine looks for input on the buttons. If the user has input \$1 and asked for a packet of Galois then we return his coin by selecting case (c) and connecting the input channel with the coin resource directly to an output (so the coin is returned).

Let us consider what is perhaps the most complex case in the example: namely, when \$2 is entered and the gum button is pressed. In this case we start by choosing case (b), then we want to transmute \$2 into gum and \$1 (which are sent along channel γ). Next we want to pass these things to the person using the machine so we split the resources. Here we have to `fork` as the user may use these channels in such a way as to make them dependent (for example he may always look for his change before he takes his gum). We then independently return his change and provide the gum in two different processes. To provide the change we send an event `change` to indicate that change is due and then pass the change down that channel.

Finally let us consider the case where \$1 is provided and gum is selected. In this case we again have to fork the output into two processes one of which

returns the gum and the other of which handles the change. On the latter we indicate that no change is due by sending the event nochange: this gives a tensor unit channel on output which (provided that no channels are open) allows one to end the process.

Outline of this paper

The main formal result of the paper is that there is a decision procedure for this term logic. However, the fact that such a procedure exists is, we feel, a fairly standard observation and not as important as establishing the term logic itself. To establish the term logic we introduce it formally in Section 2. In fact, we introduce it in two different syntactic flavors: the programming syntax, as above, and a more succinct (and cryptic) representation to facilitate the technical arguments.

In Section 3 we introduce a rewriting system on terms. The rewriting corresponds to the cut elimination procedure and proof equivalences for multiplicative-additive linear logic.

In Section 4 we show that the terms can be interpreted in any linearly distributive category with sums and products and, furthermore, form themselves into the free representable polycategory with additives. By looking at the maps in this polycategory we obtain the corresponding free linearly distributive category with additives. These results are an extension of the results in [15] and use similar techniques.

In Section 5 we discuss the proof of categorical cut elimination. This involves showing, in proof theoretic terms, that the cut elimination procedure terminates (which is well-known). However, it also involves showing the more demanding fact these rewrites are confluent modulo the equations introduced by the permuting conversions. Once one has categorical cut elimination the equality of proofs can be determined by using the equations alone: the fact that the system is decidable follows from the subformula property which limits the number of proofs and, thus, the number of terms in an equivalence class.

This is a somewhat unsatisfactory state of affairs in which to leave things, as it does not indicate the complexity of the decision procedure. It is possible to organize the decision procedure to be much more efficient. Unfortunately, even in this reorganization the procedure has some steps to accomplish which correspond to the classical rewiring problem discussed in [4] for the multiplicatives. Thus, the complexity of this determination is dominated by the complexity of determining equality in the multiplicative fragment. The first author has conjectured (pessimistically) that this is exponential and this still remains an open problem.

$$\frac{}{A \vdash A}\ \text{(identity)}$$

$$\frac{\{\Gamma, X_i \vdash \Delta\}_{i\in I}}{\Gamma, \sum_{i\in I} X_i \vdash \Delta}\ \text{(cotuple)} \qquad \frac{\{\Gamma \vdash Y_i, \Delta\}_{i\in I}}{\Gamma \vdash \prod_{i\in I} Y_i, \Delta}\ \text{(tuple)}$$

$$\frac{\Gamma, X_k \vdash \Delta}{\Gamma, \prod_{i\in I} X_i \vdash \Delta}\ \text{(projection)} \qquad \frac{\Gamma \vdash Y_k, \Delta}{\Gamma \vdash \sum_{i\in I} Y_i, \Delta}\ \text{(injection)}$$

where $k \in I, I \neq \emptyset$

$$\frac{\Gamma, \{X_i\}_{i\in I} \vdash \Delta}{\Gamma, \bigotimes_{i\in I} X_i \vdash \Delta}\ \text{(ltensor)} \qquad \frac{\Gamma \vdash \{Y_i\}_{i\in I}, \Delta}{\Gamma \vdash \bigoplus_{i\in I} Y_i, \Delta}\ \text{(rpar)}$$

$$\frac{\{\Gamma_i, X_i \vdash \Delta_i\}_{i\in I}}{\Gamma, \bigoplus_{i\in I} X_i \vdash \Delta}\ \text{(lpar)} \qquad \frac{\{\Gamma_i \vdash Y_i, \Delta_i\}_{i\in I}}{\Gamma \vdash \bigotimes_{i\in I} Y_i, \Delta}\ \text{(rtensor)}$$

where $\Gamma = \bigsqcup_i \Gamma_i$ and $\Delta = \bigsqcup_i \Delta_i$ ($\bigsqcup$ denotes disjoint union)

$$\frac{\Gamma \vdash \Delta, Z \quad Z, \Gamma' \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \text{(cut)}$$

Table 1
Multiplicative-additive linear logic

2 Term calculi for multiplicative-additive linear logic

In this section we introduce two term calculi for multiplicative-additive linear logic. We recall the sequent rules for the multiplicative and additive fragments of linear logic [7] in Table 1.

The additive fragment consists of the cotuple, tuple, projection, and injection rules along with the identity and cut rules. The multiplicative fragment consists of the ltensor, rtensor, rpar, and lpar rules along with the identity and cut rules.

This presentation uses operations indexed by arbitrary (finite) sets. We will often write simply i (e.g., $\sum_i X_i$) or j and use it to mean $i \in I$ and $j \in J$. Except in the injection and projection rules these index sets are allowed to be empty: this gives the nullary rules. These nullary rules are usually presented

[7] We are using the symbols $+$ and $\times$ to refer to the categorical sum and product respectively, and the symbols $\otimes$ and $\oplus$ for the categorical tensor and cotensor respectively. Note then that this conflicts with Girard's notation in [9], but is consistent with [4].

separately as:

$$\frac{}{\Gamma, \mathbf{0} \vdash \Delta}\ (\text{cotuple}) \qquad \frac{}{\Gamma \vdash \mathbf{1}, \Delta}\ (\text{tuple})$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, \top \vdash \Delta}\ (\text{ltensor}) \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}\ (\text{rpar})$$

$$\frac{}{\bot \vdash}\ (\text{lpar}) \qquad \frac{}{\vdash \top}\ (\text{rtensor})$$

We shall consider various augmentations of this basic logic:

- The "initial logic" is the logic with no atoms. Notice this is still a non-trivial logic because of the symbols $\mathbf{0}$, $\mathbf{1}$, $\top$, and $\bot$. We shall denote this logic as $\mathsf{CProc}(\emptyset)$.
- The "pure logic" is the logic with an arbitrary set of atoms A: we shall denote this logic as $\mathsf{CProc}(A)$.
- The "free logic" is the logic with an arbitrary set of atoms and an arbitrary set of non-logical axioms relating lists of atoms. For example, if f is a non-logical axiom relating A, B to C, D, this may be denoted as a poly-map or an inference, i.e., as

$$f : A, B \to C, D \qquad \text{or} \qquad \frac{}{f :: A, B \vdash C, D}$$

  The atoms will be regarded as objects in a polycategory and the axioms as poly-maps in that polycategory (with the "essential cuts" being provided by composition). If the polycategory is denoted $\mathbf{A}$, the resulting logic will be denoted $\mathsf{CProc}(\mathbf{A})$.

If we think of the atoms of a pure logic as forming a discrete category (a category where the only maps are the identities), the free logic on this discrete category is just the "pure" logic. Each variant above therefore includes the previous variant, and as it is more general, we shall tend to only consider the free logic. It is worth noting that a simple inductive argument shows that the logic will have negation whenever the polycategory $\mathbf{A}$ has negation.

2.1 Formulas as protocols

The term representations are dependent on formulas annotated with "channel names" and "events" which are derived from the process semantics view in which the formulas represent protocols. The reader interested in a more complete story is referred to [15].

Each formula is assigned a channel name, which we denote using Greek

letters, e.g., if X is a formula, $\alpha : X$ is a formula attached to channel α. A formula is just formula of linear logic (without negation) which is presented using a tagged notation:

- if X is atomic then it is left as is.
- $X = \sum_i X_i$ is tagged with events as $\{a_i : X_i \mid i \in I\}$, where $a_i \neq a_j$ when $i \neq j$.
- $X = \prod_i X_i$ is tagged with events as $(a_i : X_i \mid i \in I)$, where $a_i \neq a_j$ when $i \neq j$.
- $X = \bigotimes_i X_i$ is tagged with channel names as $\bigotimes_i \alpha_i : X_i$, where $\alpha_i \neq \alpha_j$ when $i \neq j$.
- $X = \bigoplus_i X_i$ is tagged as $\bigoplus_i \alpha_i : X_i$, where $\alpha_i \neq \alpha_j$ when $i \neq j$.

For example, the linear logic formula $(W+X)\otimes(Y\oplus Z)$ in this tagged notation might become:

$$\alpha : \{a : W, b : X\} \otimes \beta : (\beta_1 : Y \oplus \beta_2 : Z)$$

The channel names occurring as the tags for the formulas in a sequent must be distinct as they are used as references. To simplify our term representation and to avoid channel name conversions we will use distinct channel names within the formulas as well. Furthermore, a convention we adopt is to assign implicit tags to multiplicatives: thus, $\alpha : \bigotimes_i X_i$ or $\alpha : \bigoplus_i X_i$ have implicitly assigned their constituents the channel names α_i, for $i \in I$.

We are now ready to introduce the term calculi. Two different syntaxes will be used to represent proofs in this system: a "programming language" syntax and a more succinct term representation. We begin with the programming syntax.

2.2 Programming language syntax

The programming syntax provides an explicit process reading for the proofs. Unfortunately, while this representation is quite intuitive, it is also rather verbose which is why we will introduce a more succinct representation in the next section.

The term formation rules for the programming language are given in Table 2. The language is self dual and so only half the rules are presented. To reduce the overload strain on colons we will use "::" to denote the term-type membership relation, e.g., $t :: U \vdash V$ will mean that t is a term of type $U \vdash V$, where U (say) may be of the form $a : X$. Note also that we introduce special syntax (`stop`, `close`, and `end`) for the nullary cases.

$$\frac{}{\alpha =_A \beta :: \alpha : A \vdash \beta : A}$$

$$\frac{\{f_i :: \Gamma \vdash \alpha : X_i, \Delta\}_i}{\texttt{input } \alpha \texttt{ of } |_i\, a_i \mapsto f_i :: \Gamma \vdash \alpha : \prod_i a_i : X_i, \Delta}$$

$$\frac{f :: \Gamma \vdash \alpha : X_k, \Delta}{\texttt{output } a_k \texttt{ on } \alpha \texttt{ in } f :: \Gamma \vdash \alpha : \sum_i a_i : X_i, \Delta}$$

where $k \in I, I \neq \emptyset$

$$\frac{f :: \Gamma \vdash \{\alpha_i : X_i\}_i, \Delta}{\texttt{split } \alpha \texttt{ as } (\alpha_i)_i \texttt{ in } f :: \Gamma \vdash \alpha : \bigoplus_i \alpha_i : X_i, \Delta}$$

$$\frac{\{f_i :: \Gamma_i \vdash \alpha_i : X_i, \Delta_i\}_i}{\texttt{fork } \alpha \texttt{ as } |_i\, \alpha_i \texttt{ with } \Lambda(\Gamma_i, \Delta_i) \mapsto f_i :: \Gamma \vdash \alpha : \bigotimes_i \alpha_i : X_i, \Delta}$$

where $\Gamma = \bigsqcup_i \Gamma_i$ and $\Delta = \bigsqcup_i \Delta_i$

$$\frac{f :: \Gamma \vdash \Delta, \gamma : Z \quad g :: \gamma : Z, \Gamma' \vdash \Delta'}{\texttt{on } \gamma \texttt{ plug } f \texttt{ to } g :: \Gamma, \Gamma' \vdash \Delta, \Delta'}$$

Notation for nullary cases:

$$\frac{}{\texttt{stop } \alpha :: \Gamma \vdash \alpha : \mathbf{1}, \Delta}$$

$$\frac{f :: \Gamma \vdash \Delta}{\texttt{close } \alpha \texttt{ in } f :: \Gamma \vdash \alpha : \bot, \Delta}$$

$$\frac{}{\texttt{end } \alpha :: \vdash \alpha : \top}$$

Table 2
"Programming language" term formation rules

The notation $\Lambda(\Gamma, \Delta)$ represents the set of domain and codomain channels of the sequent, i.e., the set of channels tags of the formulas in the sequent. For example,

$$\Lambda(\alpha_1 : \{a : W, b : X\},\ \alpha_2 : A,\ \beta : (Y \oplus Z)) = \{\alpha_1, \alpha_2, \beta\}$$

Note that the operation $\Lambda(-)$ does not extract any internal channel names.

A non-logical axiom introduced as $f : \alpha_1 : X_1, \ldots, \alpha_m : X_m \to \beta_1 : Y_1, \ldots, \beta_n : Y_n$ will be represented by the term $f(\alpha_1, \ldots, \alpha_m; \beta_1, \ldots, \beta_n)$. The

identity map on atoms, $\alpha : X \to \beta : X$, can then be represented using this notation as $1_X(\alpha;\beta)$, although we will prefer the notation $\alpha =_X \beta$. Two terms are **channel equivalent** if they are equivalent up to channel name conversion: clearly the renaming of channels does not affect the meaning of the terms.

Example 2.1 The following proof of the linear distribution map

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha_{21} : B \vdash \beta_1 : B} \quad \dfrac{\overline{\alpha_1 : A \vdash \beta_{21} : A} \quad \overline{\alpha_{22} : C \vdash \beta_{22} : C}}{\alpha_1 : A, \alpha_{22} : C \vdash \beta_2 : A \otimes C}}{\alpha_1 : A, \alpha_2 : B \oplus C \vdash \beta_1 : B, \beta_2 : A \otimes C}}{\alpha_1 : A, \alpha_2 : B \oplus C \vdash \beta : B \oplus (A \otimes C)}}{\alpha : A \otimes (B \oplus C) \vdash \beta : B \oplus (A \otimes C)}$$

is represented in the programming syntax as follows:

```
split α as α1, α2 in
   split β as β1, β2 in
      fork α2 as
         | α21 with β1 ↦ α21 =B β1
         | α22 with α1, β2 ↦ fork β2 as
                                | β21 with α1 in α1 =A β21
                                | β22 with α22 in α22 =C β22
```

Note that in view of the rewrites in Section 3 there are a number of equivalent presentations of this process.

2.3 A term calculus representation

In this section we introduce a more succinct representation for proofs which we call the term calculus representation. The term formation rules are given in Table 3; again only half the rules are presented.

In this representation (as well as in the programming language representation) the notation does not directly indicate on which side of the turnstile an active channel sits. This can be inferred from the term and also by inspecting the type. However, it is sometimes useful to make this information more explicit and to do this we will indicate domain channels by an overarrow pointing left, $\overleftarrow{\alpha}$, and codomain channels by an overarrow pointing right, $\overrightarrow{\alpha}$.

Example 2.2 The following is the term calculus representation of the proof

$$\frac{}{\alpha =_A \beta :: \alpha : A \vdash \beta : A}$$

$$\frac{\{f_i :: \Gamma \vdash \alpha : X_i, \Delta\}_i}{\alpha\{a_i \mapsto f_i\}_i :: \Gamma \vdash \alpha : \prod_i a_i : X_i, \Delta}$$

$$\frac{f :: \Gamma \vdash \alpha : X_k, \Delta}{\alpha[a_k]f :: \Gamma \vdash \alpha : \sum_i a_i : X_i, \Delta}$$

where $k \in I, I \neq \emptyset$

$$\frac{f :: \Gamma \vdash \{\alpha_i : X_i\}_i, \Delta}{\alpha\langle(\alpha_i)_i \mapsto f\rangle :: \Gamma \vdash \alpha : \bigoplus_i \alpha_i : X_i, \Delta}$$

$$\frac{\{f_i :: \Gamma_i \vdash \alpha_i : X_i, \Delta_i\}_i}{\alpha\langle\alpha_i \mid \Lambda(\Gamma_i, \Delta_i) \mapsto f_i\rangle_i :: \Gamma \vdash \alpha : \bigotimes_i \alpha_i : X_i, \Delta}$$

where $\Gamma = \bigsqcup_i \Gamma_i$ and $\Delta = \bigsqcup_i \Delta_i$

$$\frac{f :: \Gamma \vdash \Delta, \gamma : Z \quad g :: \gamma : Z, \Gamma' \vdash \Delta'}{f ;_\gamma g :: \Gamma, \Gamma' \vdash \Delta, \Delta'}$$

Table 3
Term calculus formation rules

found in Example 2.1 above:

$$\alpha\langle(\alpha_1, \alpha_2) \mapsto \beta\langle(\beta_1, \beta_2) \mapsto \alpha_2 \left\langle \begin{array}{ll} \alpha_{21}|\beta_1 & \mapsto \alpha_{21} =_B \beta_1 \\ \alpha_{22}|\alpha_1, \beta_2 & \mapsto \beta_2 \left\langle \begin{array}{ll} \beta_{21}|\alpha_1 & \mapsto \alpha_1 =_A \beta_{21} \\ \beta_{22}|\alpha_{22} & \mapsto \alpha_{22} =_C \beta_{22} \end{array} \right\rangle \end{array} \right\rangle\rangle$$

It is clear that the term calculus is more succinct and will be more convenient for manipulating the terms, accordingly we shall use this representation for the remainder of the paper.

3 Term rewrites and equivalences

Two terms f and g may be composed on a channel γ when γ is a codomain channel of f of type X and a domain channel of g of the same type. Furthermore, γ must be the only channel name f and g have in common. That is, the two terms must be of the form $f :: \Gamma_1 \to \Gamma_2, \gamma : X$ and $g :: \gamma : X, \Delta_1 \to \Delta_2$,

where γ is the only channel name in common. Composing these terms on γ is denoted:

$$f;_\gamma g :: \Gamma_1, \Delta_1 \to \Gamma_2, \Delta_2$$

The requirement that γ is the only channel name in common ensures that after composition all the channel names will be distinct. In general this means that in order to perform a composition there may be need to perform a channel name conversion to arrange that the two terms have this form.

3.1 Cut elimination rewrites

Composition is exactly the cut rule. Thus, the dynamics of composition corresponds precisely to the cut elimination process. We describe the cut elimination rewrites below. As our term calculus does not distinguish between domain and codomain channels dual proof rewrites written as terms will be identical. However, we shall be careful to indicate how the inference rules give rise to each of the rewrites.

The set of unbound channels in a term t will be denoted $\Lambda(t)$ and is the same as $\Lambda(\Gamma, \Delta)$ where $t :: \Gamma \vdash \Delta$.

The rewrites are as follows:

- Identity-sequent (sequent-identity):

  (1) $f;_\gamma 1 \Longrightarrow f$

  (2) $1;_\gamma f \Longrightarrow f$

- Cotuple-sequent (sequent-tuple), tuple-sequent (sequent-cotuple):

  (3) $\alpha\{a_i \mapsto f_i\}_i;_\gamma g \Longrightarrow \alpha\{a_i \mapsto f_i;_\gamma g\}_i$

  (4) $g;_\gamma \alpha\{a_i \mapsto f_i\}_i \Longrightarrow \alpha\{a_i \mapsto g;_\gamma f_i\}_i$

- Injection-sequent (sequent-projection), projection-sequent (sequent-injection):

  (5) $\alpha[a]f;_\gamma g \Longrightarrow \alpha[a](f;_\gamma g)$

  (6) $g;_\gamma \alpha[a]f \Longrightarrow \alpha[a](g;_\gamma f)$

- Ltensor-sequent (sequent-rpar), rpar-sequent (sequent-ltensor):

  (7) $\alpha\langle(\alpha_i)_i \mapsto f\rangle;_\gamma g \Longrightarrow \alpha\langle(\alpha_i) \mapsto f;_\gamma g\rangle$

  (8) $g;_\gamma \alpha\langle(\alpha_i)_i \mapsto f\rangle \Longrightarrow \alpha\langle(\alpha_i) \mapsto g;_\gamma f\rangle$

- Lpar-sequent (sequent-rtensor), rtensor-sequent (sequent-lpar): here we suppose $\gamma \in \Lambda_k$.

$$(9)\quad \alpha\langle \alpha_i \mid \Lambda_i \mapsto f_i\rangle_i ;_\gamma g \Longrightarrow \alpha \left\langle \begin{array}{l} \alpha_i \mid \Lambda_i \mapsto f_i \\ \alpha_k \mid (\Lambda_k \cup \Lambda(g)) \setminus \{\gamma\} \mapsto f_k ;_\gamma g \end{array} \right\rangle_{i\neq k}$$

$$(10)\ g ;_\gamma \alpha\langle \alpha_i \mid \Lambda_i \mapsto f_i\rangle_i \Longrightarrow \alpha \left\langle \begin{array}{l} \alpha_i \mid \Lambda_i \mapsto f_i \\ \alpha_k \mid (\Lambda_k \cup \Lambda(g)) \setminus \{\gamma\} \mapsto g ;_\gamma f_k \end{array} \right\rangle_{i\neq k}$$

- Injection-cotuple (tuple-projection):

$$(11)\ \gamma[a_k] f ;_\gamma \gamma\{a_i \mapsto g_i\}_i \Longrightarrow f ;_\gamma g_k$$

$$(12)\ \gamma\{a_i \mapsto g_i\}_i ;_\gamma \gamma[a_k] f \Longrightarrow g_k ;_\gamma f$$

- Rtensor-ltensor (rpar-lpar):

$$(13)\ \gamma\langle \alpha_i \mid \Lambda_i \mapsto f_i\rangle_{i\in\{1,\dots,n\}} ;_\gamma \gamma\langle (\alpha_i)_{i\in\{1,\dots,n\}} \mapsto g\rangle \Longrightarrow f_n ;_{\alpha_n} (\cdots (f_2 ;_{\alpha_2} (f_1 ;_{\alpha_1} g)) \cdots)$$

$$(14)\ \gamma\langle (\alpha_i)_{i\in\{1,\dots,n\}} \mapsto g\rangle ;_\gamma \gamma\langle \alpha_i \mid \Lambda_i \mapsto f_i\rangle_{i\in\{1,\dots,n\}} \Longrightarrow ((\cdots (g ;_{\alpha_n} f_n) \cdots) ;_{\alpha_2} f_2) ;_{\alpha_1} f_1$$

In order to obtain a normal form for sequent derivations, and hence terms, we would like to show that the cut elimination rewrites are Church-Rosser. However, this is easily seen to not be the case. Consider a derivation with a projection and an injection immediately above the cut:

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma_1, X_k \vdash \Gamma_2, Z}}{\Gamma_1, \prod_i X_i \vdash \Gamma_2, Z} \quad \dfrac{\dfrac{\pi'}{Z, \Delta_1 \vdash Y_l, \Delta_2}}{Z, \Delta_1 \vdash \sum_j Y_j, \Delta_2}}{\Gamma_1, \prod_i X_i, \Delta_1 \vdash \Gamma_2, \sum_j Y_j, \Delta_2}$$

In this case one may apply the projection-sequent rewrite or the sequent-injection rewrite to reduce the derivation, but there seems to be no way in which to resolve this pair. This motivates the use of additional rewrites which will allow us to interchange these rules (and all the other critical pairs). These rewrites (which we are denoting by $\Longmapsto$) are called the **permuting conversions** and are as follows:

- Cotuple-cotuple (tuple-tuple), cotuple-tuple interchange:

$$(15)\ \alpha\{a_i \mapsto \beta\{b_j \mapsto f_{ij}\}_j\}_i \Longmapsto \beta\{b_j \mapsto \alpha\{a_i \mapsto f_{ij}\}_i\}_j$$

- Cotuple-injection (projection-tuple), cotuple-projection (injection-tuple) interchange:

  (16) $\alpha\{a_i \mapsto \beta[b]f_i\}_i \Longrightarrow \beta[b]\alpha\{a_i \mapsto f_i\}_i$

- Cotuple-ltensor (rpar-tuple), cotuple-rpar (ltensor-tuple) interchange:

  (17) $\alpha\{a_i \mapsto \beta\langle(\beta_j)_j \mapsto f_i\rangle\}_i \Longrightarrow \beta\langle(\beta_j)_j \mapsto \alpha\{a_i \mapsto f_i\}_i\rangle$

- Cotuple-rtensor (lpar-tuple), cotuple-lpar (rtensor-tuple) interchange: here we suppose $\alpha \in \Lambda_k$.

  (18) $$\alpha\left\{a_i \mapsto \beta\left\langle \begin{array}{l} \beta_j|\Lambda_j \mapsto g_j \\ \beta_k|\Lambda_k \mapsto f_i \end{array}\right\rangle_{j\neq k}\right\}_i \Longrightarrow \beta\left\langle \begin{array}{l} \beta_j|\Lambda_j \mapsto g_j \\ \beta_k|\Lambda_k \mapsto \alpha\{a_i \mapsto f_i\}_i \end{array}\right\rangle_{j\neq k}$$

- Injection-injection (projection-projection), injection-projection interchange:

  (19) $\alpha[a](\beta[b]f) \Longrightarrow \beta[b](\alpha[a]f)$

- Injection-ltensor (rpar-projection), injection-rpar (ltensor-projection) interchange:

  (20) $\alpha[a]\beta\langle(\beta_j)_j \mapsto f\rangle \Longrightarrow \beta\langle(\beta_j)_j \mapsto \alpha[a]f\rangle$

- Injection-rtensor (lpar-projection), injection-lpar (rtensor-projection) interchange: here we suppose $\alpha \in \Lambda_k$.

  (21) $$\alpha[a]\beta\left\langle \begin{array}{l} \beta_j \mid \Lambda_j \mapsto g_j \\ \beta_k \mid \Lambda_k \mapsto f \end{array}\right\rangle_{j\neq k} \Longrightarrow \beta\left\langle \begin{array}{l} \beta_j \mid \Lambda_j \mapsto g_j \\ \beta_k \mid \Lambda_k \mapsto \alpha[a]f \end{array}\right\rangle_{j\neq k}$$

- Ltensor-ltensor (rpar-rpar), ltensor-rpar interchange:

  (22) $\alpha\langle(\alpha_i)_i \mapsto \beta\langle(\beta_j)_j \mapsto f\rangle\rangle \Longrightarrow \beta\langle(\beta_j)_j \mapsto \alpha\langle(\alpha_i)_i \mapsto f\rangle\rangle$

- Ltensor-rtensor (lpar-rpar), ltensor-lpar (rtensor-rpar) interchange:

  (23) $$\alpha\left\langle (\alpha_i)_i \mapsto \beta\left\langle \begin{array}{l} \beta_j|\Lambda_j \mapsto g_j \\ \beta_k|\Lambda_k \cup (\bigcup_i\{\alpha_i\}) \mapsto f \end{array}\right\rangle_{j\neq k}\right\rangle \Longrightarrow \beta\left\langle \begin{array}{l} \beta_j|\Lambda_j \mapsto g_j \\ \beta_k|\Lambda_k \cup \{\alpha\} \mapsto \alpha\langle(\alpha_i)_i \mapsto f\rangle \end{array}\right\rangle_{j\neq k}$$

- Rtensor-rtensor (lpar-lpar), rtensor-lpar interchange: let $\Lambda = \{\beta\} \cup \Lambda_k \cup (\bigcup_j \Lambda'_j)$ and $\Lambda' = \{\alpha\} \cup \Lambda'_{k'} \cup (\bigcup_i \Lambda_i)$.

$$(24)\ \alpha\left\langle \begin{array}{l} \alpha_i|\Lambda_i \mapsto f_i \\ \alpha_k|\Lambda \mapsto \beta\left\langle \begin{array}{l} \beta_j|\Lambda'_j \mapsto g_j \\ \beta_{k'}|\Lambda'_{k'} \mapsto h \end{array} \right\rangle_{j\neq k'} \end{array} \right\rangle_{i\neq k} \Longmapsto$$

$$\beta\left\langle \begin{array}{l} \beta_j|\Lambda'_j \mapsto g_j \\ \beta_{k'}|\Lambda' \mapsto \alpha\left\langle \begin{array}{l} \alpha_i|\Lambda_i \mapsto f_i \\ \alpha_k|\Lambda_k \mapsto h \end{array} \right\rangle_{i\neq k} \end{array} \right\rangle_{j\neq k'}$$

4 Polycategorical semantics

The reduction rules and the permuting conversions together define an equivalence relation (which we denote by $\sim$) on the derivations of a sequent. This allows us to form for each sequent a hom-set consisting of terms modulo this equivalence (based on a starting polycategory $\mathbf{A}$). The cut operation then provides polycategorical composition.

This delivers a polycategory which we shall denote $\mathsf{CProc}(\mathbf{A})$. The purpose of this section is to prove that this polycategory has additives and is representable. This amounts to saying that its maps [8] form a linearly distributive category with additives (see [6]).

Theorem 4.1 $\mathsf{CProc}(\mathbf{A})$ *is a representable additive polycategory whose objects are the formulas of the logic, and whose poly-maps are* $\sim$*-equivalence classes of derivations.*

The fact that the terms modulo equivalence provide a polycategory is proved in Appendix C. The proof consists of providing identity maps and showing that cut is associative and satisfies the interchange law. Below we discuss the remaining structure:

4.1 Additives

Our next task is to establish that the polycategory $\mathsf{CProc}(\mathbf{A})$ has additives, i.e., poly-sums and poly-products. In a polycategory $\mathbf{A}$ an object $\sum_i X_i \in \mathbf{A}$ is said to be the **poly-sum** (or **poly-coproduct**) of a family of objects $X_i \in \mathbf{A}$, for $i \in I$, in case there is a natural bijective correspondence

[8] By "map" we mean a poly-map with one domain and one codomain.

$$\frac{\{f_i :: \Gamma, \alpha : X_i \to \Delta\}_i}{\alpha\{f_i\}_i :: \Gamma, \alpha : \sum_i X_i \to \Delta}$$

where, by natural, it is meant that the equivalences

$$h;_\gamma \alpha\{f_i\}_i = \alpha\{h;_\gamma f_i\}_i \qquad \text{and} \qquad \alpha\{f_i\}_i;_\gamma h = \alpha\{f_i;_\gamma h\}_i$$

hold (when $\alpha \neq \gamma$). Poly-products in polycategories are (as might be expected) dual to poly-coproducts.

Proposition 4.2 $\mathsf{CProc}(\mathbf{A})$ *has additives.*

Proof. In order to establish that $\mathsf{CProc}(\mathbf{A})$ has finite sums and finite products it must be shown that the correspondences

$$\frac{\{\Gamma, \alpha : X_i \to \Delta\}_i}{\Gamma, \alpha : \sum_i i : X_i \to \Delta} \qquad \text{and} \qquad \frac{\{\Gamma \to \beta : Y_j, \Delta\}_j}{\Gamma \to \beta : \prod_j j : Y_j, \Delta}$$

are bijective and natural. We will prove the case for sums, products are handled dually.

Suppose we have poly-maps $t :: \Gamma, \alpha : \sum_i i : X_i \to \Delta$ and $s_i :: \Gamma, \alpha : X_i \to \Delta$, for $i \in I$. The collection of poly-maps $\{s_i\}_i$ can, via the cotupling derivation, be used to construct a poly-map

$$\psi(\{s_i\}_i) = \alpha\{i \mapsto s_i\}_i :: \Gamma, \alpha : \sum_i i : X_i \to \Delta$$

and t may be cut with $\alpha[k]1_{X_k} :: X_i \to \alpha : \sum_i X_i$, for $k \in I$, to get a poly-map

$$\varphi_k(t) = \alpha[k]1_{X_k};_\alpha t :: \Gamma, X_k \to \Delta$$

To prove that this correspondence is bijective we must establish both $\varphi_k(\psi(\{s_i\}_i)) = s_k$, for $k \in I$, and $\psi(\{\varphi_i(t)\}_i) = t$. The former follows from

$$\varphi_k(\psi(\{s_i\}_i)) \;=\; \varphi_k(\alpha\{i \mapsto s_i\}_i) \;=\; \alpha[k]1_{X_k};_\alpha \alpha\{i \mapsto s_i\}_i \;=\; 1_{X_k};_\alpha s_k \;=\; s_k$$

and the latter from

$$\begin{aligned} \psi(\{\varphi_i(t)\}_i) \;=\; \psi(\alpha[i]1_{X_i};_\alpha t) &= \beta\{i \mapsto \alpha[i]1_{X_i};_\alpha t\}_i \\ &= \beta\{i \mapsto \alpha[i]1_{X_i}\}_i;_\alpha t \\ &= 1_{\Sigma_i X_i};_\alpha t \\ &= t \end{aligned}$$

It remains to show that this correspondence is natural, however, this follows immediately from the rewrites (3) and (4). □

4.2 Representability

A multi-map $\Gamma \xrightarrow{u} X$ is said to **represent** Γ **as input** (cf. [10,5]), if cutting with u at X induces a natural bijection as follows:

$$\frac{\Gamma_1, \Gamma, \Gamma_2 \to \Delta}{\Gamma_1, X, \Gamma_2 \to \Delta}$$

Dually, a comulti-map $Y \xrightarrow{v} \Gamma$ **represents** Γ **as output**, if cutting with v at Y induces an analogous bijection. A polycategory is called **representable** if each sequence of formulas is representable as input and as output.

Theorem 4.3 $\mathsf{CProc}(\mathbf{A})$ *is representable.*

Proof. We will prove that a bundle of channels $\{\beta_i : X_i\}_i$ is represented as input by the following multi-map

$$\alpha \langle \alpha_i \mid \beta_i \mapsto \beta_i =_{X_i} \alpha_i \rangle :: \{\beta_i : X_i\}_i \to \alpha : \bigotimes_i \alpha_i : X_i$$

The symmetry of the term calculus will then deliver representability. This means that we must show the correspondences

$$\frac{\Gamma, \{\alpha_i : X_i\}_i \to \Delta}{\Gamma, \alpha : \bigotimes_i \alpha_i : X_i \to \Delta} \qquad \text{and} \qquad \frac{\Gamma \to \Delta}{\Gamma, \alpha : \top \to \Delta}$$

are bijective and natural.

The proof of the non-nullary case follows in a manner very similar to that of Proposition 4.2. Suppose we have poly-maps $t :: \Gamma, \alpha : \bigotimes_i \alpha_i : X_i \to \Delta$ and $s :: \Gamma, \{\alpha_i : X_i\}_i \to \Delta$. The poly-map s can, via the ltensor derivation, be used to construct a poly-map

$$\psi(s) = \alpha \langle (\alpha_i)_i \mapsto s \rangle :: \Gamma, \alpha : \bigotimes_i \alpha_i : X_i \to \Delta$$

and t may be cut with the representing multi-map to get a poly-map

$$\varphi(t) = \alpha \langle \alpha_i \mid \beta_i \mapsto \beta_i =_{X_i} \alpha_i \rangle_i ;_\alpha t :: \Gamma, \{\alpha_i : X_i\}_i \to \Delta$$

To prove that this correspondence is bijective we must establish both $\varphi(\psi(s)) = s$ and $\psi(\varphi(t)) = t$. Suppose $I = \{1, \ldots, n\}$. The former follows from

$$\begin{aligned} \varphi(\psi(s)) &= \varphi(\alpha \langle (\alpha_i)_i \mapsto s \rangle) \\ &= \alpha \langle \alpha_i \mid \beta_i \mapsto \beta_i =_{X_i} \alpha_i \rangle_i ;_\alpha \alpha \langle (\alpha_i)_i \mapsto s \rangle) \\ &= \beta_n =_{X_n} \alpha_n ;_{\alpha_n} (\cdots (\beta_1 =_{X_1} \alpha_1 ;_{\alpha_1} s) \cdots) \\ &= s \end{aligned}$$

and the latter from

$$\begin{aligned}\psi(\varphi(t)) &= \psi(\beta\langle \beta_i \mid \alpha_i \mapsto \alpha_i =_{X_i} \beta_i\rangle_i;_\beta t)\\ &= \alpha\langle (\alpha_i)_i \mapsto \beta\langle \beta_i \mid \alpha_i \mapsto \alpha_i =_{X_i} \beta_i\rangle_i;_\beta t\rangle\\ &= \alpha\langle (\alpha_i)_i \mapsto \beta\langle \beta_i \mid \alpha_i \mapsto \alpha_i =_{X_i} \beta_i\rangle_i\rangle;_\beta t\\ &= 1_{\bigotimes_i X_i};_\beta t\\ &= t\end{aligned}$$

To establish the nullary case suppose we have poly-maps $t :: \Gamma, \alpha : \top \to \Delta$ and $s :: \Gamma \to \Delta$, and define

$$\psi(s) = \alpha\langle (\,) \mapsto s\rangle \quad \text{and} \quad \varphi(t) = \alpha\langle\,\rangle;_\alpha t$$

To see that it is again a bijection observe

$$\varphi(\psi(s)) \;=\; \varphi(\alpha\langle (\,) \mapsto s\rangle) \;=\; \alpha\langle\,\rangle;_\alpha \alpha\langle (\,) \mapsto s\rangle \;=\; s$$

and

$$\psi(\varphi(t)) \;=\; \psi(\beta\langle\,\rangle;_\beta t) \;=\; \alpha\langle (\,) \mapsto \beta\langle\,\rangle;_\beta t\rangle \;=\; \alpha\langle (\,) \mapsto \beta\langle\,\rangle\rangle;_\beta t \;=\; 1_\top;_\beta t \;=\; t$$

It is left to show that the correspondence is natural, however, this follows immediately from the rewrites (7) and (8). □

4.3 The free additive linearly distributive category

The goal of this section is to prove:

Theorem 4.4 $\mathsf{CProc}(\mathbf{A})$ *is the free representable polycategory with additives generated by the polycategory* $\mathbf{A}$.

As any linearly distributive category with additives generates a polycategory with additives, an immediate corollary of this is:

Corollary 4.5 *The maps of* $\mathsf{CProc}(\mathbf{A})$ *form the free linearly distributive category with additives generated by the polycategory* $\mathbf{A}$.

Note that this is the usual (categorical) notion of freeness.

In order to prove this theorem we must show that in any representable polycategory with additives the identities (1)–(24) must hold.

It is routine to show (see [15]) that a polycategory has poly-sums if and only if it has a cotupling operation which is distributive on the non-sum channel, and injections which when cut against a cotuple on the sum channel delivers the appropriate component of the cotuple, and finally it satisfies "surjective

pairing" which is the requirement that the cotuple of the injections is the identity.

Now observe that the injection term $\alpha[a]f$ can be translated as $f; b_k$ where $b_k :: X_k \to \sum_i X_i$, for $k \in I$. This is a valid identification as there is a reduction of derivations

$$\dfrac{\Gamma \vdash X_k, \Delta \quad \dfrac{X_k \vdash X_k}{X_k \vdash \sum_i X_k}}{\Gamma \vdash \sum_i X_i, \Delta} \quad \Longrightarrow \quad \dfrac{\Gamma \vdash X_k, \Delta}{\Gamma \vdash \sum_i X_i, \Delta}$$

The term calculus identities for the additives express precisely the requirements described above with the exception of surjective pairing. However, recall that this is implicit in the term calculus from the manner in which the identity map for the coproduct is defined.

The only remaining difficulty is to translate the "forking" construct of the term logic. Suppose $I = \{1, \ldots, n\}$ for $n \geq 1$, then in any representable polycategory we may translate the poly-map $\alpha\langle \alpha_i \mid \Lambda_i \mapsto f_i \rangle_i$ as $f_n ;_{\alpha_n} (\ldots (f_1 ;_{\alpha_2} r) \ldots)$ where r is the representing multi-map.

$$\dfrac{\dfrac{f_1}{\Gamma_1 \vdash X_1, \Delta_1} \quad \dfrac{\dfrac{f_n}{\Gamma_n \vdash X_n, \Delta_n} \quad \dfrac{1_{\bigotimes_i X_i}}{\bigotimes_i X_i \vdash \bigotimes_i X_i}}{\{X_i\}_i \vdash_r \bigotimes_i X_i} \quad \cdots}{X_1, \Gamma_2, \cdots, \Gamma_n \vdash \bigotimes_i X_i, \Delta_2, \cdots, \Delta_n}$$
$$\Gamma \vdash \bigotimes_i X_i, \Delta$$

The identities are now an easy consequence. In particular, (9), (10), (13), and (14) are a direct consequence of the translation of "forking" described above.

5 The decision procedure

Cut elimination seen as a term rewriting supplies a way of rewriting the terms (in the initial calculus) to remove the cut completely. In $\mathsf{CProc}(\mathbf{A})$ it allows the cuts to be moved into the underlying polycategory $\mathbf{A}$. This relies on the following routine but nonetheless technical observation which also delivers a "categorical cut elimination" theorem as the equality of proofs (as determined by the equalities above) is now guaranteed to be maintained by the elimination procedure.

Theorem 5.1 *In* $\mathsf{CProc}(\mathbf{A})$*:*

(i) *The rewriting on poly-maps given by (1)–(14) terminates.*

(ii) *The rewriting on poly-maps given by (1)–(14) is confluent modulo the permuting conversions (15)–(24).*

The proof is presented in some detail in Appendix B. The argument involves resolving all the critical pairs (rewriting against rewriting and rewriting against permuting conversion) modulo the permuting conversions. To show that the rewriting terminates in forms which are equivalent with respect to the permuting conversions involves showing that the resolution of each critical pair always reduces the cost of the frontier. This does involve some subtlety as the nullary additive permuting conversions can produce some apparently non-reducing steps. However, somewhat surprisingly, a very basic multiset measure of cut heights applied carefully does suffice for the whole argument to go through quite smoothly.

The effect of this theorem is to provide a decision procedure modulo the decidability of the underlying polycategory $\mathbf{A}$. The rewriting normalizes terms by moving the cut into the non-logical axioms. The equivalence of terms is then determined by the decision procedure in $\mathbf{A}$ and the permuting conversions. The subformula property delivered by cut elimination ensures that there are only finitely many proofs which do not involve cut (excluding how the non-logical steps are filled). This means that, in principal, in order to decide the equality of two terms one can simply search all the terms equivalent by permuting conversions to one of the proofs for a term equivalent to the other. Two terms are equivalent when the proof structure of the two terms match exactly and the non-logical steps which correspond are equivalent. This observation is sufficient to establish the following formal result:

Corollary 5.2 *The equivalence of proofs in* $\mathsf{CProc}(\mathbf{A})$ *is decidable whenever the equivalence of proofs in* $\mathbf{A}$ *is decidable.*

Recall that all discrete polycategories and free polycategories (including those with negation) are decidable so that there is a ready source of decidable polycategories.

Of course, the procedure proposed is highly inefficient and we wish to finish the paper with some remarks on how one can make it more efficient. In order to decide the equality of two terms it is worth putting one term aside as a template. However, it is also sensible to ensure that this term has all tensor/par splitting done first and then all the tupling/cotupling. This structure can always be pulled up to the root of the term whenever an active multiplicative or additive type is present. This leaves the tensor/par forking and the injections/projections. The latter can be pulled up in a similar manner to [7]. Pulling up a forking to match a template term which starts with such is more complex: this is exactly where the complexity of rewirings (due to

the multiplicative units) in the purely multiplicative fragment bites. The complexity of solving this step remains an open question.

References

[1] Abramsky, S., *Computational interpretations of linear logic*, Theoretical Computer Science **111** (1993), pp. 3–57.

[2] Bellin, G. and P.J. Scott, *On the π-calculus and linear logic*, Theoretical Computer Science **135** (1994), pp. 11–65. Also at urlhttp://www.csi.uottawa.ca/∼phil/.

[3] Benton, N., G. Bierman, V. de Paiva and M. Hyland, *A term calculus for intuitionistic linear logic*, in: M. Bezem and J. F. Groote, editors, *Proc. Intl. Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science **664** (1993), pp. 75–90.

[4] Blute, R.F., J.R.B. Cockett, R.A.G. Seely and T.H. Trimble, *Natural deduction and coherence for weakly distributive categories*, Journal of Pure and Applied Algebra **3** (1996), pp. 229–296. Also at http://www.math.mcgill.ca/rags/.

[5] Cockett, J.R.B., J. Koslowski and R.A.G. Seely, *Morphisms and modules for poly-bicategories*, Theory and Applications of Categories **11** (2003), pp. 15–74.

[6] Cockett, J.R.B. and R.A.G. Seely, *Linearly distributive functors*, Journal of Pure and Applied Algebra **143** (1999), pp. 155–203. Also at http://www.math.mcgill.ca/rags/.

[7] Cockett, J.R.B. and R.A.G. Seely, *Finite sum-product logic*, Theory and Applications of Categories **8** (2001), pp. 63–99.

[8] Dershowitz, N. and Z. Manna, *Proving termination with multiset orderings*, Communications of the ACM **22** (1979), pp. 465–476.

[9] Girard, J.-Y., *Linear logic*, Theoretical Computer Science **50** (1987), pp. 1–102. Also at http://iml.univ-mrs.fr/{$\sim$}girard/.

[10] Hermida, C., *Representable multicategories*, Advances in Mathematics **151** (2000), pp. 164–225. Also at http://maggie.cs.queensu.ca/chermida/.

[11] Hughes, D. and R. van Glabbeek, *Proof nets for unit-free multiplicative-additive linear logic (extended abstract)*, in: *Proc. IEEE symposium on Logic in Computer Science* (2003). Also at http://boole.stanford.edu/$\sim$dominic/.

[12] Jay, B., *Languages for monoidal categories*, Journal of Pure and Applied Algebra **59** (1989), pp. 61–85.

[13] Koh, T.W. and C.-H.L. Ong, *Type theories for autonomous and $*$-autonomous categories: I. Types theories and rewrite systems II. Internal languages and coherence theorems* (1998). Available at http://users.comlab.ox.ac.uk/luke.ong/.

[14] Lambek, J., *Multicategories revisited*, in: J. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic: Proc. of the Joint Summer Research Conference*, American Mathematical Society, 1989 pp. 217–239.

[15] Pastro, C.A., "ΣΠ-Polycategories, additive Linear Logic, and process semantics," Master's thesis, University of Calgary (2004). Available at http://www.cpsc.ucalgary.ca/{$\sim$}pastroc/.

A The units

The way in which the cut elimination procedure handles the reductions and permuting conversions when the index sets are empty can be quite subtle. In the case of ltensor or rpar, however, it is very straightforward, however the other cases are quite tricky. To clarify this, in this section we make these special cases explicit.

The annotated nullary versions of the inference rules are:

$$\frac{}{\alpha\{\,\} :: \Gamma, \alpha : \mathbf{0} \vdash \Delta}\ (\text{cotuple}) \qquad \frac{}{\alpha\{\,\} :: \Gamma \vdash \alpha : \mathbf{1}, \Delta}\ (\text{tuple})$$

$$\frac{f :: \Gamma \vdash \Delta}{\alpha\langle(\,) \mapsto f\rangle :: \Gamma, \alpha : \top \vdash \Delta}\ (\text{ltensor}) \qquad \frac{f :: \Gamma \vdash \Delta}{\alpha\langle(\,) \mapsto f\rangle :: \Gamma \vdash \alpha : \bot, \Delta}\ (\text{rpar})$$

$$\frac{}{\alpha\langle\,\rangle :: \alpha : \bot \vdash}\ (\text{lpar}) \qquad \frac{}{\alpha\langle\,\rangle :: \vdash \alpha : \top}\ (\text{rtensor})$$

In the cotuple and tuple rules the notation is ambiguous as one cannot derive the context from the terms. To correct this we shall write the terms above as $\alpha\{\,\}_{\Gamma\vdash\Delta}$. An important observation is that the nullary rtensor and nullary lpar rules may only be applied if there are no other "active" channels present. Considering this, there are three (non-dual) reductions that are relevant to this setting corresponding to the rewrites (3), (7), and (13). The reduction (9) is not relevant to this setting as it will reduced to (13). Given a term $f :: \gamma : X, \Delta_1 \vdash \Delta_2$ the reductions are as follows:

$$(3)\ \alpha\{\,\}_{\Gamma_1\vdash\Gamma_2,\gamma:X}\ ;_\gamma f \Longrightarrow \alpha\{\,\}_{\Gamma_1,\Delta_1\vdash\Gamma_2,\Delta_2}$$

$$(7)\quad \alpha\langle(\,) \mapsto g\rangle\ ;_\gamma f \Longrightarrow \alpha\langle(\,) \mapsto g ;_\gamma f\rangle$$

$$(13)\ \gamma\langle\,\rangle\ ;_\gamma \gamma\langle(\,) \mapsto f\rangle \Longrightarrow f$$

With the interchange rules the typing of the term will determine which interchanges are allowed to take place. For example, consider (23), the ltensor-rtensor identity:

$$\alpha\left\langle (\alpha_i)_i \mapsto \beta \left\langle \begin{array}{l} \beta_j|\Lambda_j \mapsto g_j \\ \beta_k|\Lambda_k \cup (\bigcup_i\{\alpha_i\}) \mapsto f \end{array} \right\rangle_{j\neq k} \right\rangle \rightleftharpoons \beta\left\langle \begin{array}{l} \beta_j|\Lambda_j \mapsto g_j \\ \beta_k|\Lambda_k \cup \{\alpha\} \mapsto \alpha\langle(\alpha_i)_i \mapsto f\rangle \end{array} \right\rangle_{j\neq k}$$

On the left-hand side the only time the nullary rtensor rule may be applied

is if $\bigcup_i \{\alpha_i\} = \emptyset$ and no other channels are "active". This corresponds to the term

$$\alpha\langle(\,) \mapsto \beta\langle\,\rangle :: \alpha : \top \vdash \beta : \top$$

However, on the right-hand side the nullary rtensor rule will never apply as the channel α is still "active". Thus, in this case, these rules may not be interchanged. If only the ltensor is empty (on channel α) then the rules may freely be interchanged.

Considering the typing of the terms this leaves 13 permuting conversion: three versions of (15) (corresponding to the cases when only $I = \emptyset$, only $J = \emptyset$, and both $I = J = \emptyset$), (16), three variants of (17), (18) (when $I = \emptyset$), (20), three variants of (22), and finally (23) (when $I = \emptyset$). Fortunately, the form of the rewrites are quite similar, and so we present only the three variants of (15), (20), and (23). For simplicity we assume that α is a domain channel and β is a codomain channel. Here we drop the typing on the term and indicate to the right in brackets.

$$(15) \qquad \alpha\{a_i \mapsto \beta\{\,\}_{\Gamma,\alpha:X_i\vdash\Delta}\}_i \Longrightarrow \beta\{\,\}_{\Gamma,\alpha:\sum X_i\vdash\Delta}$$

$$(15) \qquad \alpha\{\,\}_{\Gamma\vdash\beta:\prod Y_j,\Delta} \Longrightarrow \beta\{b_j \mapsto \alpha\{\,\}_{\Gamma\vdash\beta:Y_j,\Delta}\}_j$$

$$(15) \qquad \alpha\{\,\}_{\Gamma\vdash\beta:\mathbf{1},\Delta} \Longrightarrow \beta\{\,\}_{\Gamma,\alpha:\mathbf{0}\vdash\Delta}$$

$$(20) \qquad \alpha[a]\beta\langle(\,) \mapsto f\rangle \Longrightarrow \beta\langle(\,) \mapsto \alpha[a]f\rangle$$

$$(23) \qquad \alpha\left\langle () \mapsto \beta\left\langle \begin{matrix}\beta_j|\Lambda_j \mapsto g_j\\ \beta_k|\Lambda_k \mapsto f\end{matrix}\right\rangle_{j\neq k}\right\rangle \Longrightarrow \beta\left\langle \begin{matrix}\beta_j|\Lambda_j \mapsto g_j\\ \beta_k|\Lambda_k \cup \{\alpha\} \mapsto \alpha\langle() \mapsto f\rangle\end{matrix}\right\rangle_{j\neq k}$$

B Proof of cut elimination and the Church-Rosser property

In this appendix we show that the cut elimination procedure terminates and that the rewrite system induced by the cut elimination rewrites and the permuting conversion has the Church-Rosser property. We begin by proving that cut elimination is a terminating procedure.

Remark B.1 The number of rewritings in our system motivates the use of a "generalized" system of rewritings which help to reduce this number. Let $\alpha(f)$ denote any of the morphisms

$$\alpha\{a_i \mapsto f_i\}_i, \qquad \alpha[a]f, \qquad \alpha\langle(\alpha_i) \mapsto f\rangle, \qquad \alpha\langle\alpha_i \mid \Lambda_i \mapsto f_i\rangle_i$$

Then $\alpha(f);_\gamma g \Longrightarrow \alpha(f;_\gamma g)$ will be used respectively to denote:

$$\alpha\{a_i \mapsto f_i\}_i;_\gamma g \xRightarrow{(3)} \alpha\{a_i \mapsto f_i;_\gamma g\}_i$$

$$\alpha[a]f;_\gamma g \xRightarrow{(5)} \alpha[a](f;_\gamma g)$$

$$\alpha\langle(\alpha_i) \mapsto f\rangle;_\gamma g \xRightarrow{(7)} \alpha\langle(\alpha_i) \mapsto f;_\gamma g\rangle$$

$$\alpha\langle\alpha_i \mid \Lambda_i \mapsto f_i\rangle_i;_\gamma g \xRightarrow{(9)} \alpha\left\langle \begin{array}{l} \alpha_i \mid \Lambda_i \mapsto f_i \\ \alpha_k \mid \Lambda_k \cup \Lambda(g) \setminus \{\gamma\} \mapsto f_k;_\gamma g \end{array} \right\rangle_{i\neq k}$$

Dually $g;_\gamma \alpha(f) \Longrightarrow \alpha(g;_\gamma f)$ will denote any of the rewrites (4), (6), (7), or (10). The rewrites (11) and (13) (and their duals) may be represented as $\gamma(f);_\gamma \gamma(g) \Longrightarrow f;_\gamma g$ and the permuting conversions (15) through (24) may be represented as $\alpha(\beta(f)) \Longleftrightarrow \beta(\alpha(f))$.

B.1 The cut measure on terms

The purpose of this section is to show that the cut elimination procedure terminates. To this end we define a bag of cut heights and show that the bag is strictly reduced on each of the cut elimination rewrites.

We begin by defining the multiset ordering of Dershowitz and Manna [8]. Let $(S, \succ)$ be a partially-ordered set, and let $\mathcal{M}(S)$ denote the multisets (or bags) over S. For $M, N \in \mathcal{M}(S)$, $M > N$ (">" is called the **multiset** (or **bag**) **ordering**), if there are multisets $X, Y \in \mathcal{M}(S)$, where $\emptyset \neq X \subseteq M$, such that

$$N = (M \setminus X) \cup Y \quad \text{and} \quad (\forall y \in Y)(\exists x \in X)\ x \succ y$$

where $\cup$ here is the multiset union.

For example,

$$[3] > [2, 2, 2, 1], \quad [7, 3] > [7], \quad [5, 2] > [5, 1]$$

Recall from [8] that if $(S, \succ)$ is a total order then $\mathcal{M}(S)$ is a total order. To see this consider $M, N \in \mathcal{M}(S)$. To determine whether $M > N$ sort the elements of both M and N and then compare the two sorted sequences lexicographically.

We now define the **height** of a term as:

- $\mathrm{hgt}[a] = 1$ when a is an atomic map (or an identity)
- $\mathrm{hgt}[\alpha\{a_i \mapsto f_i\}_{i\in I}] = 1 + \max\{\mathrm{hgt}[f_i] \mid i \in I\}$
- $\mathrm{hgt}[\alpha[a_k] \cdot f] = 1 + \mathrm{hgt}[f]$

- $\mathrm{hgt}[\alpha\langle(\alpha_i)_{i\in I} \mapsto f\rangle] = 1 + \mathrm{hgt}[f]$
- $\mathrm{hgt}[\alpha\langle\alpha_i \mid \Omega_i \mapsto f_i\rangle_{i\in I}] = 1 + \sum_{i\in I} \mathrm{hgt}[f_i]$
- $\mathrm{hgt}[f;g] = \mathrm{hgt}[f] + \mathrm{hgt}[g]$

The **height of a cut** is defined simply as its height, e.g., $\mathrm{cuthgt}[f;g] = \mathrm{hgt}[f;g]$. Define a function $\Lambda : T \to \mathrm{bag}(\mathbb{N})$ which takes a term to its bag of cut heights.

Proposition B.2

(i) *If* $t_1 \Longrightarrow t_2$ *then* $\Lambda(t_1) > \Lambda(t_2)$.

(ii) *If* $t_1 \overset{(a)}{\Longmapsto} t_2$ *and* (a) *is an interchange which does not involve the nullary cotuple or tuple then* $\Lambda(t_1) = \Lambda(t_2)$.

Proof. We begin with the proof of part (i). There are three properties that must be shown: $\mathrm{hgt}[t_1] \geq \mathrm{hgt}[t_2]$, the height of each non-principal cut does not increase, and the height of any cut produced from the principal cut is strictly less than the height of the principal cut.

A simple examination of the rewrites will confirm that if $t_1 \Longrightarrow t_2$ then $\mathrm{hgt}[t_1] \geq \mathrm{hgt}[t_2]$:

(1), dually (2)
$$\mathrm{hgt}[f;1] = \mathrm{hgt}[f] + \mathrm{hgt}[1] > \mathrm{hgt}[f]$$

(3), dually (4)
$$\begin{aligned}\mathrm{hgt}[\alpha\{a_i \mapsto f_i\}_{i\in I};_\gamma g] &= \mathrm{hgt}[\alpha\{a_i \mapsto f_i\}_{i\in I}] + \mathrm{hgt}[g]\\ &= 1 + \max\{\mathrm{hgt}[f_i] \mid i \in I\} + \mathrm{hgt}[g]\\ &= 1 + \max\{\mathrm{hgt}[f_i] + \mathrm{hgt}[g] \mid i \in I\}\\ &= \mathrm{hgt}[\alpha\{a_i \mapsto f_i;_\gamma g\}_{i\in I}]\end{aligned}$$

If $I = \emptyset$ then

$$\mathrm{hgt}[\alpha\{\,\};_\gamma g] = \mathrm{hgt}[\alpha\{\,\}] + \mathrm{hgt}[g] > \mathrm{hgt}[\alpha\{\,\}]$$

(5), dually (6)
$$\begin{aligned}\mathrm{hgt}[\alpha[a_k]f;_\gamma g] &= \mathrm{hgt}[\alpha[a_k]f] + \mathrm{hgt}[g]\\ &= 1 + \mathrm{hgt}[f] + \mathrm{hgt}[g]\\ &= \mathrm{hgt}[\alpha[a_k](f;_\gamma g)]\end{aligned}$$

(7), dually (8)
$$\begin{aligned}\mathrm{hgt}[\alpha\langle(\alpha_i)_{i\in I} \mapsto f\rangle ;_\gamma g] &= \mathrm{hgt}[\alpha\langle(\alpha_i)_{i\in I} \mapsto f\rangle + \mathrm{hgt}[g] \\ &= 1 + \mathrm{hgt}[f] + \mathrm{hgt}[g] \\ &= \mathrm{hgt}[\alpha\langle(\alpha_i)_{i\in I} \mapsto f ;_\gamma g\rangle\end{aligned}$$

(9), dually (10)

$$\begin{aligned}\mathrm{hgt}[\alpha\langle\alpha_i \mid \Omega_i \mapsto f_i\rangle_{i\in I} ;_\gamma g] &= \mathrm{hgt}[\alpha\langle\alpha_i \mid \Omega_i \mapsto f_i\rangle_{i\in I}] + \mathrm{hgt}[g] \\ &= 1 + \sum_{k\neq i\in I} \mathrm{hgt}[f_i] + \mathrm{hgt}[f_k] + \mathrm{hgt}[g] \\ &= \mathrm{hgt}\left[\alpha\left\langle \begin{array}{l}\alpha_i \mid \Omega_i \mapsto f_i \\ \alpha_k \mid \Omega_k \mapsto f_k ;_\gamma g\end{array}\right\rangle_{k\neq i\in I}\right]\end{aligned}$$

(11), dually (12)
$$\begin{aligned}\mathrm{hgt}[\gamma[a_k]f ;_\gamma \gamma\{a_i \mapsto g_i\}_{i\in I}] &= \mathrm{hgt}[\gamma[a_k]f] + \mathrm{hgt}[\gamma\{a_i \mapsto g_i\}_{i\in I}] \\ &= 1 + \mathrm{hgt}[f] + 1 + \max\{\mathrm{hgt}[g_i] \mid i \in I\} \\ &> \mathrm{hgt}[f] + \max\{\mathrm{hgt}[g_i] \mid i \in I\} \\ &\geq \mathrm{hgt}[f] + \mathrm{hgt}[g_k] \\ &= \mathrm{hgt}[f ;_\gamma g_k]\end{aligned}$$

(13), dually (14)
$$\begin{aligned}\mathrm{hgt}[\gamma\langle\alpha_i|\Omega_i \mapsto f_i\rangle_i ;_\gamma \gamma\langle(\alpha_i)_i \mapsto g\rangle] &= \mathrm{hgt}[\gamma\langle\alpha_i|\Omega_i \mapsto f_i\rangle_i] + \mathrm{hgt}[\alpha\langle(\alpha_i)_i \mapsto g\rangle] \\ &= 1 + \sum_i \mathrm{hgt}[f_i] + 1 + \mathrm{hgt}[g] \\ &> \sum_i \mathrm{hgt}[f_i] + \mathrm{hgt}[g] \\ &= \mathrm{hgt}\left[f_n ;_{\alpha_n} (\cdots (f_2 ;_{\alpha_2} (f_1 ;_{\alpha_1} g)) \cdots)\right]\end{aligned}$$

If $I = \emptyset$ then

$$\begin{aligned}\mathrm{hgt}[\gamma\langle\,\rangle_i ;_\gamma \gamma\langle(\,) \mapsto g\rangle] &= \mathrm{hgt}[\gamma\langle\,\rangle_i] + \mathrm{hgt}[\gamma\langle(\,) \mapsto g\rangle] \\ &= 1 + 1 + \mathrm{hgt}[g] \\ &> \mathrm{hgt}[g]\end{aligned}$$

Moreover, this implies that cuts below and cuts above the redex will not increase their cut height on a rewriting.

Finally, consider the principal cut of the reduction. Rewrite (1) (dually (2)) removes a cut and so strictly reduces the bag of cut heights. It is an easy observation that (5), (7), (9), and (11) (and their duals) each replace a cut with one of lesser height, and that (3) and (13) (and their duals) replace a cut with zero or more cuts of lesser height. Thus applying any of the rewrites strictly reduces the bag.

We now prove part (ii). For the equations (15), (16), (17), and (18) we assume that the index sets are non-empty. This then implies that the commuting conversions are all of the form $\alpha(\beta(f))$ and thus

$$\mathrm{hgt}[\alpha(\beta(f))] = 1 + \mathrm{hgt}[\beta(f)] = 1 + 1 + \mathrm{hgt}[f] = \mathrm{hgt}[\beta(\alpha(f))]$$

which proves that the height does not change across these (non-empty cotuple and tuple) interchanges. □

To see that the height is not invariant across the empty cotuple (dually the tuple) rule recall one of the nullary versions of the rewrite (15):

$$\alpha\{\,\} \Longmapsto \beta\{b_j \mapsto \alpha\{\,\}\}_j$$

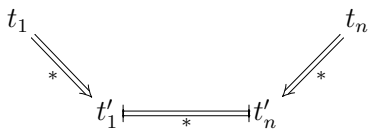
The height on the left-hand side is one, while on the right-hand side the height is two.

B.2 Proof of the Church-Rosser property

In this section we present a proof of the Church-Rosser property for morphisms. We wish to show that given any two morphisms related by a series of reductions and permuting conversions

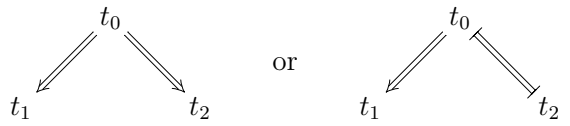
$$t_1 \Longleftarrow t_2 \Longmapsto t_3 \Longrightarrow \quad \cdots \quad \Longleftarrow t_{n-2} \Longmapsto t_{n-1} \Longrightarrow t_n$$

there is an alternative way of arranging the reductions and permuting conversions so that t_1 and t_n can be reduced to terms which are related by the permuting conversions alone. That is, we wish to show that there is a convergence of the following form:

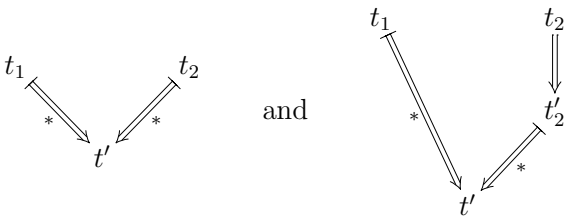


When the rewriting system terminates (in the appropriate sense) this allows the decision procedure for the equality of ΣΠ-terms to be reduced to the decision procedure for the permuting conversions (see Section 5). In order to test the equality of two terms, one can rewrite both terms into a reduced form (one from which there are no further reductions), and these will be equal if and only if the two reduced forms are equivalent through the permuting conversions alone. In the current situation the reduction process is, of course, the cut-elimination procedure.

Following [7] we say a rewrite system is **locally confluent modulo equations** if any (one step) divergence of the following form



(where "$\Longrightarrow$" denotes a reduction and "$t_1 \models\!\!\Longrightarrow\!\! t_2$" an equation) has a convergence, respectively, of the form



where the new arrow "$\models\!\!\Longrightarrow$" indicates either an equality or a reduction in the indicated direction.

This gives:

Proposition B.3 *Suppose $(N, \mathcal{R}, \mathcal{E})$ is a rewriting system with the equations equipped with a well-ordered measure on the rewrite arrows such that the measure of the divergences is strictly greater than the measure of the convergences then the system is confluent modulo equations if and only if it is locally confluent modulo equations.*

Proof. If the system is confluent modulo equations it is certainly locally confluent modulo equations. Conversely suppose we have a chain of reductions, equations, and expansions. We may associate with it the bag of measures of the arrows of the sequence.

The idea will be to show that replacing any local divergence in this chain by a local confluence will result in a new chain whose bag measure is strictly smaller. However, this can be seen by inspection as we are removing the arrows

associated with the divergence and replacing them with the arrows associated with the convergence. The measure on the arrows associated with the divergence is strictly greater then that of the measure on the arrows associated with the convergence.

Thus, each rewriting reduces the measure and, therefore, any sequence of rewriting on such a chain must terminate. However, it can only terminate when there are no local divergences to resolve. This then implies that the end result must be a confluence modulo equations. □

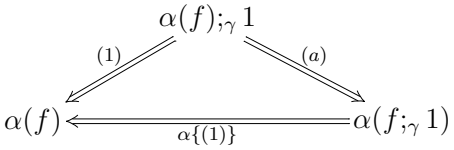
B.2.1 Resolving critical pairs locally

The proof of Church-Rosser involves examining all the possible critical pairs involving reductions or reductions and conversions, and showing that they are all of the form shown above and that they may be resolved in the way shown above. It then must be shown that there is some measure on the arrows which decreases when replacing a divergence with a convergences. This will then suffice to show that our system is locally confluent modulo equations, so that by Proposition B.3, it is confluent modulo equations. The rewrites (1)–(12) are the "reductions" and the commuting conversions (13)–(24) are the "equations". The resolutions of the critical pairs will be presented using the "generalized" rewrites (see Remark B.1). For the additive rewrites see [15] where they have been written out in detail.

The reductions are as follows. Note that these reductions assume that all index sets are non-empty. The reductions when the index set is empty are handled separately below.

Reduction diagram 0: $1;1 \underset{(2)}{\overset{(1)}{\Rightarrow}} 1$

Reduction diagram 1: Substitute (3), (5), (7), or (9) for (a) to get the reduction diagrams for (1)–(3), (1)–(5), (1)–(7), and (1)–(9).

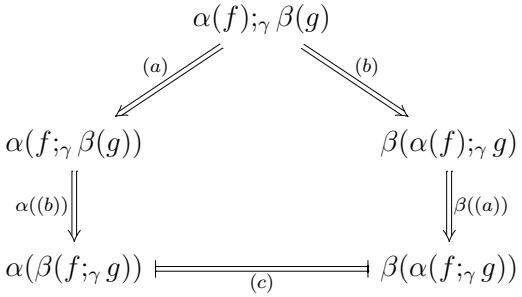


Substituting the dual rewrites in the mirror image of the diagram above give the dual reduction diagrams.

Reduction diagram 2: each row in the table corresponds to the resolution

of the critical pair (a)–(b).

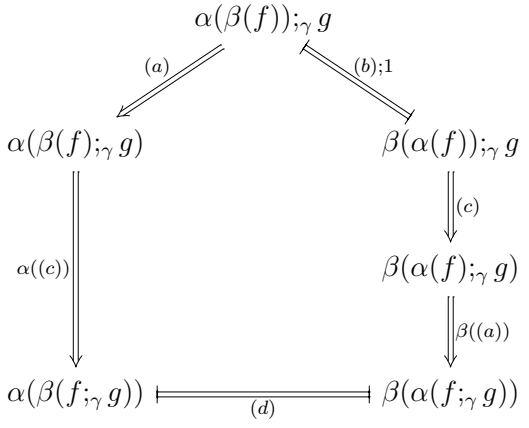
a	b	c
3	4	15
3	6	16
3	8	17
3	10	18
5	6	19
5	8	20
5	10	21
7	8	22
7	10	23
9	10	24



Reduction diagram 3: each row in the table corresponds to the resolution of the critical pair (a)–(b).

a	b	c
3	15	3
3	16	5
3	17	7
3	18	9
5	16	3
5	19	5
5	20	7
5	21	9

a	b	c
7	17	3
7	20	5
7	22	7
7	23	9
9	18	3
9	21	5
9	23	7
9	24	9

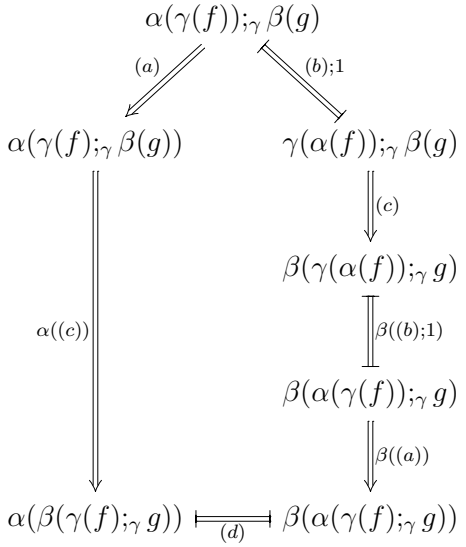


Reduction diagram 4: each row in the table corresponds to the resolution

of the critical pair (a)–(b).

a	b	c	d
3	w	4	15
3	w	6	16
3	w	8	17
3	w	10	18
5	x	4	16
5	x	6	19
5	x	8	20
5	x	10	21

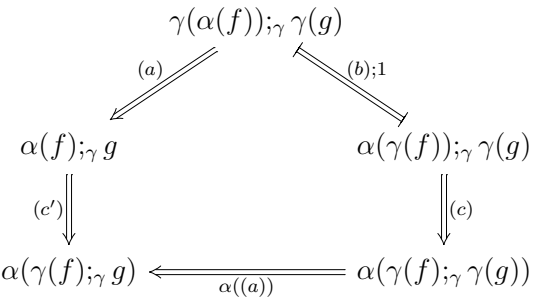
a	b	c	d
7	y	4	17
7	y	6	20
7	y	8	22
7	y	10	23
9	z	4	18
9	z	6	19
9	z	8	20
9	z	10	21



where $w \in \{15, 16, 17, 18\}$, $x \in \{16, 19, 20, 21\}$, $y \in \{17, 20, 22, 23\}$, and $z \in \{18, 21, 23, 24\}$. This means that there are 64 reductions that fit this general case!

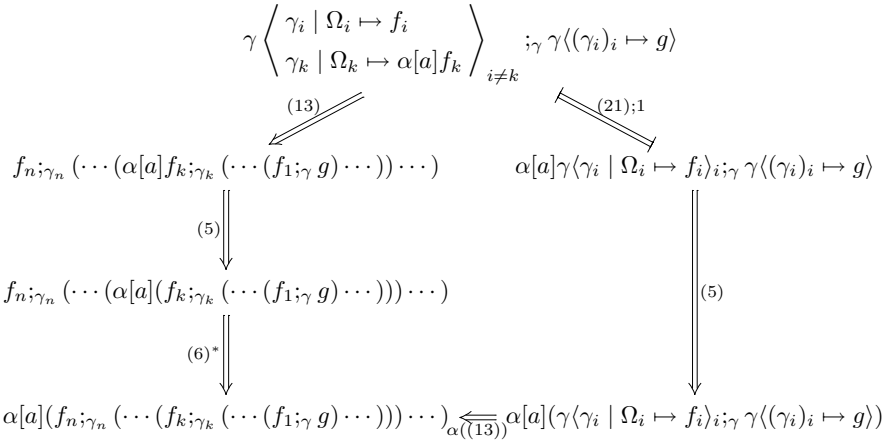
Reduction diagram 5: each row in the table corresponds to the resolution of the critical pair (a)–(b).

a	b	c	c'
11	16	3	3
11	19	5	5
11	20	7	7
11	21	9	9
13	18	3	$3^+, 4^*$
13	21	5	$5^+, 6^*$
13	23	7	$7^+, 8^*$
13	24	9	$9^+, 10^*$



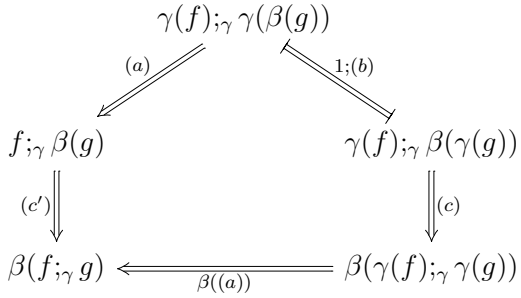
where x^+, y^* means zero or one application of the rewrite (x) and zero or more applications of the rewrite (y).

This may be a good time for a concrete example. Suppose $(a, b, c, c') = (13, 21, 5, (5^+, 6^*))$, $i \in \{1, \ldots, n\}$, and $\alpha \in \Omega_k$. The reduction diagram for this case is:



Reduction diagram 6: each row in the table corresponds to the resolution of the critical pair (a)–(b).

a	b	c	c'
11	15	4	4
11	16	6	4
11	17	8	8
11	18	10	10
13	17	4	$4^+, 3^*$
13	20	6	$6^+, 5^*$
13	22	8	$8^+, 7^*$
13	23	10	$10^+, 9^*$



where x^+, y^* means zero or one application of the rewrite (x) and zero or more applications of the rewrite (y).

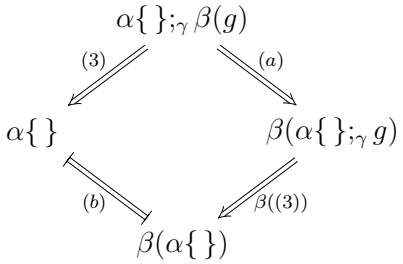
We now explore the cases when the index sets may be empty. For the empty ltensor (dually rpar) rule the rewrites fit the cases above. The cases for the empty cotuple (dually tuple) and rtensor (dually lpar) however do not. We start by first examining what happens to the reduction diagrams for the

case of the empty cotuple (dually tuple).

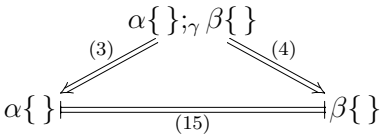
Reduction diagram 1: $\alpha\{\,\};1 \xrightarrow[(3)]{(1)} \alpha\{\,\}$

Reduction diagram 2: there are two (non-dual) cases corresponding to only $I = \emptyset$ and both $I = J = \emptyset$. Each row in the table corresponds to the resolution of the critical pair (3)–(a).

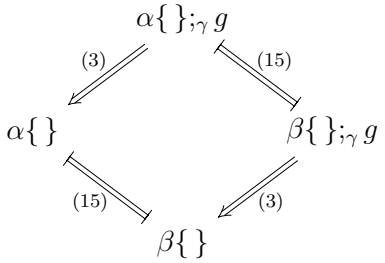
a	b
4	15
6	16
8	17
10	18



Dual to the above diagram is the nullary reductions for β. If both α and β have empty index sets:

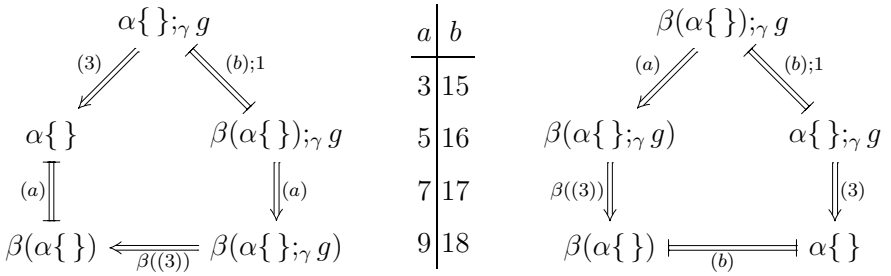


Reduction diagram 3: there are three cases. The first we describe is when both $I = J = \emptyset$. The resolution is as follows:



The two remaining cases corresponding to whether the apex (of the reduction diagram) starts with $\alpha\{\,\}$ or with $\beta(\alpha\{\,\})$. Each row in the table corresponds to the resolution of the critical pair (3)–(b) in the reduction

diagram on the left and (a)–(b) in the reduction diagram on the right.

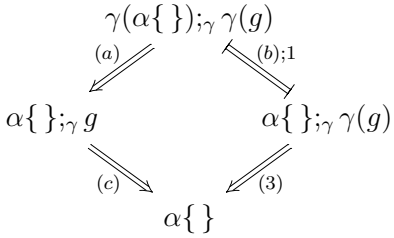


a	b
3	15
5	16
7	17
9	18

Reduction diagram 4: The channel α is non-empty and if channel γ is empty the reduction diagram is identical.

Reduction diagram 5: each row in the table corresponds to the resolution of the critical pair (a)–(b).

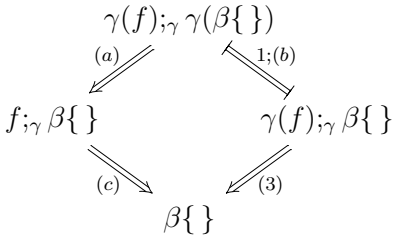
a	b	c
11	16	3
13	18	$3^+, 4^*$



where $3^+, 4^*$ means zero or one application of the rewrite (3) and zero or more applications of the rewrite (4).

Reduction diagram 6: each row in the table corresponds to the resolution of the critical pair (a)–(b).

a	b	c
11	15	4
13	17	4^*



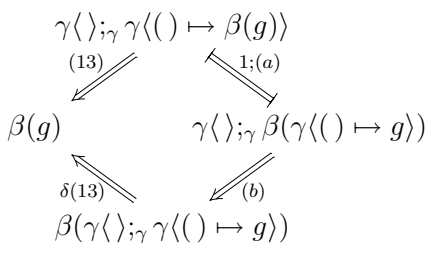
where 4^* means zero or more applications of the rewrite (4).

We now examine what happens to the reduction diagrams for the case of the empty rtensor (dually lpar). Due to the typing constraints the only reduction diagrams that need be considered are 1 and 6.

Reduction diagram 1: $\alpha\langle\,\rangle; 1 \xRightarrow[(13)]{(1)} \alpha\langle\,\rangle$

Reduction diagram 6: each row in the table corresponds to the resolution of the critical pair (13)–(a).

a	b
17	4
20	6
22	8
23	10



B.2.2 The measure on the rewriting arrows

We define a measure $\lambda : A \to \mathrm{bag}(\mathbb{N})$ on the rewriting arrows as follows:

- if $t_1 \xRightarrow{x} t_2$ then $\lambda(x) = \min\{\Lambda(t_1),\ \Lambda(t_2)\}$
- if $t_1 \xRightarrow{x}\!\!\!\!\mid\ t_2$ then $\lambda(x) = \max\{\Lambda(t_1),\ \Lambda(t_2)\}$

where $\Lambda(t)$ is the bag of cut heights of t.

A quick examination of the reduction diagrams now confirms that this measure will decrease when we replace a divergence with a convergence.

This completes the proof of the proposition:

Proposition B.4 $\mathsf{CProc}(\mathbf{A})$ *under the rewrites (1)–(14) is confluent modulo the equations (15)–(24).*

C Proof that $\mathsf{CProc}(\mathbf{A})$ is a polycategory

In order to show that $\mathsf{CProc}(\mathbf{A})$ is a polycategory three properties must be satisfied:

(i) $\mathsf{CProc}(\mathbf{A})$ must have identity maps which behave in the correct manner,
(ii) composition in $\mathsf{CProc}(\mathbf{A})$ must be associative, and
(iii) composition in $\mathsf{CProc}(\mathbf{A})$ must satisfy the interchange law.

We prove each in turn.

Lemma C.1 *The identity acts as a neutral element with respect to composition. That is, given terms of the form*

$$f :: \Gamma \to \Delta, \gamma : X \quad \textit{and} \quad 1_X :: \gamma : X \to \delta : X$$

we have $f;_\gamma 1_X = f$ *(up to renaming channels), and dually, given terms of the form*

$$1_X :: \gamma : X \to \delta : X \quad \textit{and} \quad f :: \delta : X, \Gamma \to \Delta$$

we have $1_X;_\delta f = f$ *(Upton renaming channels).*

Proof. We shall suppose the identity is on the left; duality covers the other case. The case where f is the identity is clearly true. So suppose f is of the form $\alpha(f)$, where $\alpha \neq \gamma$. If $\alpha(f)$ is a tuple or rtensor with $I = \emptyset$ then $\alpha(f);_\gamma 1_X$ will reduce to $\alpha\{\,\}$ or $\delta\langle\,\rangle$ respectively. Otherwise, any rewrite applied to $\alpha(f);_\gamma 1_X$ will move the identity in $\alpha(f;_\gamma 1_X)$ moving the cut onto a smaller term from which we may apply the inductive hypothesis. Thus, the only cases we must explore is when the term f operates on γ at the top level.

The proof will proceed by structure induction on the term $\gamma(f)$. Without loss of generality we may assume that f is cut free.

- The base case is a cut with an atomic identity: here the cut-elimination step removes the identity and the result is immediate.
- $f = \gamma\{a_i \mapsto f_i\}_{i\in I}$. There are two cases to consider corresponding to $I = \emptyset$ and $I \neq \emptyset$. In the first case $1_X = \delta\{\,\}$ and by rewrite (4) this reduces to $\delta\{\,\}$. In the second case we have $X = \prod_i a_i : X_i$ and $1_X = \delta\{a_i \mapsto \gamma[a_i]1_{X_i}\}_i$. This gives

$$\begin{aligned}\gamma\{a_i \mapsto f_i\}_i;_\gamma \delta\{a_i \mapsto \gamma[a_i]1_{X_i}\}_i &\Longrightarrow \delta\{a_i \mapsto \gamma\{a_i \mapsto f_i\}_i;_\gamma \gamma[a_i]1_{X_i}\}_i \\ &\Longrightarrow \delta\{a_i \mapsto f_i;_\gamma 1_{X_i}\}_i\end{aligned}$$

  which moves the composition onto smaller terms. Applying the inductive hypothesis now gives the desired result.
- $f = \gamma[a_k]f'$. In this case $X = \sum_i X_i$ and $1_X = \gamma\{a_i \mapsto \delta[a_i]1_{X_i}\}_i$ which gives

$$\begin{aligned}\gamma[a_k]f';_\gamma \gamma\{a_i \mapsto \delta[a_i]1_{X_i}\}_i &\Longrightarrow f';_\gamma \delta[a_k]1_{X_k} \\ &\Longrightarrow \delta[a_k](f';_\gamma 1_{X_k})\end{aligned}$$

  which moves the composition onto a smaller term. Applying the inductive hypothesis now gives the desired result.
- $f = \gamma\langle(\gamma_i)_i \mapsto f'\rangle$. In this case $X = \bigoplus_i \gamma_i : X_i$ (in f and the domain of 1_X and $X = \bigoplus_i \delta_i : X_i$ in the codomain of 1_X) and $1_X = \delta\langle(\delta_i)_i \mapsto \gamma\langle\gamma_i \mid \delta_i \mapsto 1_{X_i}\rangle_i\rangle$. In this case suppose $I = \{1, \ldots, n\}$. This gives

$$\begin{aligned}\gamma\langle(\gamma_i)_i \mapsto f'\rangle ;_\gamma \delta\langle(\delta_i)_i \mapsto \gamma\langle\gamma_i \mid \delta_i \mapsto 1_{X_i}\rangle_i\rangle \\ \Longrightarrow \delta\langle(\delta_i)_i \mapsto \gamma\langle(\gamma_i)_i \mapsto f'\rangle ;_\gamma \gamma\langle\gamma_i \mid \delta_i \mapsto 1_{X_i}\rangle_i\rangle \\ \Longrightarrow \delta\langle(\delta_i)_i \mapsto (\cdots(f' ;_{\gamma_n} 1_{X_n})\cdots) ;_{\gamma_1} 1_{X_1}\rangle\end{aligned}$$

which moves the composition onto smaller terms. Applying the inductive hypothesis now gives the desired result.

- $f = \gamma\langle\gamma_i \mid \Omega_i \mapsto f_i\rangle_{i\in I}$. There are two cases to consider corresponding to $I = \emptyset$ and $I \neq \emptyset$. In the first case $1_X = \gamma\langle(\,) \mapsto \delta\langle\,\rangle$ which by (13) reduces to $\delta\langle\,\rangle$. In the second case we have $X = \bigotimes_i \gamma_i : X_i$ (in f and the domain of 1_X and $X = \bigotimes_i \delta_i : X_i$ in the codomain of 1_X) and $1_X = \gamma\langle(\gamma_i)_i \mapsto \delta\langle\delta_i \mid \gamma_i \mapsto 1_{X_i}\rangle_i\rangle$. In this case suppose $I = \{1, \ldots, n\}$. This gives

$$\begin{aligned}\gamma\langle\gamma_i \mid \Omega_i \mapsto f_i\rangle_i ;_\gamma \gamma\langle(\gamma_i)_i \mapsto \delta\langle\delta_i \mid \gamma_i \mapsto 1_{X_i}\rangle_i\rangle \\ \Longrightarrow f_n ;_{\gamma_n} (\cdots(f_1 ;_{\gamma_1} \delta\langle\delta_i \mid \gamma_i \mapsto 1_{X_i}\rangle_i)\cdots) \\ \Longrightarrow \delta\langle\delta_i \mid \Omega_i \mapsto f_i ;_{\gamma_i} 1_{X_i}\rangle_i\end{aligned}$$

which moves the composition onto smaller terms. Applying the inductive hypothesis now gives the desired result.

This now completes the proof that the identity acts as a neutral element with respect to composition in this system. □

Lemma C.2 *Cut satisfies the associative law. That is, given terms of the form*

$$f :: \Gamma_1 \to \Gamma_2, \gamma : X \qquad g :: \gamma : X, \Delta_1 \to \Delta_2, \delta : Y \qquad h :: \delta : Y, \Phi_1 \to \Phi_2$$

the composites $(f ;_\gamma g) ;_\delta h$ *and* $f ;_\gamma (g ;_\delta h)$ *are* $\sim$*-equivalent.*

Proof. The proof proceeds by structural induction on f, g, and h; without loss of generality we may assume that f, g, and h are cut free. Recall that if two terms are $\sim$-equivalent then they must be related through the permuting conversions alone. Without explicitly mentioning where, duality will be used to reduce the number of cases presented. To show that the composites are $\sim$-equivalent we will make use of the generalized rewrites. Additionally, the notation

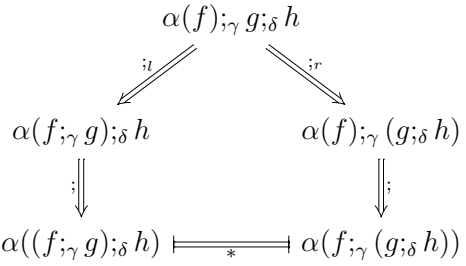
$$f ;_\gamma g ;_\delta h \overset{;l}{\Longrightarrow} (f ;_\gamma g) ;_\delta h \quad \text{and} \quad f ;_\gamma g ;_\delta h \overset{;r}{\Longrightarrow} f ;_\gamma (g ;_\delta h)$$

will be used to indicate the two possible composites of $f ;_\gamma g ;_\delta h$.

If $f = 1_X$ then $1_X ;_\gamma g ;_\delta h \overset{;l}{\underset{;r}{\Longrightarrow}} g ;_\delta h$ shows that the composites are $\sim$-equivalent.

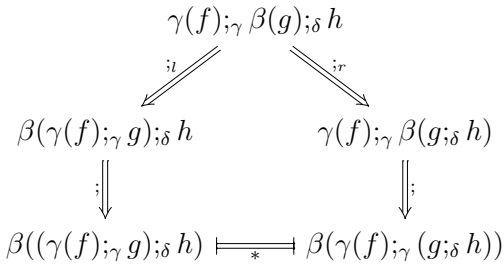
Suppose now that f is of the form $\alpha(f)$, where $\alpha \neq \gamma$. If $\alpha(f)$ is a tuple or rtensor with $I = \emptyset$ then $f ;_\gamma g ;_\delta h$ will respectively reduce to $\alpha\{\,\}$ or $\beta\langle\,\rangle$

(where β is the codomain channel of h). If $I \neq \emptyset$ then in following reduction diagram

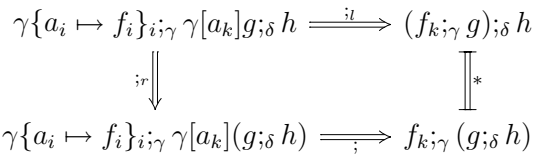


the composites are moved onto smaller terms, which by induction are satisfy the associative law, and hence, both composites are $\sim$-equivalent.

It remains to examine the case where f is of the form $\gamma(f)$. There are two cases corresponding to whether g is of the form $\beta(g)$, where $\beta \neq \gamma$ and $\beta \neq \delta$, or $\gamma(g)$. (The dual rewrites will suffice for the $\beta = \delta$ case.) In the first case if $\beta(g) = \beta\{\,\}$ then both composites will reduce to $\beta\{\,\}$. Note the $\beta(g) \neq \beta\langle\,\rangle$ as both composites are defined. So suppose that $\beta(g) \neq \beta\{\,\}$. The following reduction diagram shows that both composites are $\sim$-equivalent (by induction):



In the second case ($\beta = \gamma$) there is no need to generalize as there are only two (non-dual) rewrites (which also has the benefit of providing a "concrete" example). The first is when the left-hand side composite is the rewrite (11):



The second case is when the left-hand side composite is the rewrite (13). In this case assume that $I = \{1, \ldots, n\}$:

$$\begin{array}{ccc} \alpha\langle \alpha_i \mid \Omega_i \mapsto f_i \rangle_i ;_\gamma \gamma\langle (\gamma_i) \mapsto g \rangle ;_\delta h & \xRightarrow{;_l} & (f_n ;_{\gamma_n} (\cdots (f_1 ;_{\gamma_1} g) \cdots)) ;_\delta h \\ \Big\Downarrow {\scriptstyle ;_r} & & \Big\Vert {\scriptstyle *} \\ \alpha\langle \alpha_i \mid \Omega_i \mapsto f_i \rangle_i ;_\gamma \gamma\langle (\gamma_i) \mapsto g ;_\delta h \rangle & \xRightarrow[;]{} & (f_n ;_{\gamma_n} (\cdots (f_1 ;_{\gamma_1} (g ;_\delta h)) \cdots)) \end{array}$$

If all of f, g, and h are atomic then composition is associative as it must be associative in the underlying polycategory. If some of f, g, and h are atomic a quick check of the possibilities will show that one ends up with a case similar to one of the cases above.

Thus, composition is associative. □

Lemma C.3 *Cut satisfies the interchange property. That is, given terms of the form*

$$f :: \Gamma_1 \to \gamma : X, \Gamma_2, \delta : Y \qquad g :: \Delta_1, \gamma : X \to \Delta_2 \qquad h :: \delta : Y, \Phi_1 \to \Phi_2$$

the composites $(f ;_\gamma g) ;_\delta h$ *and* $(f ;_\delta h) ;_\gamma g$ *are* $\sim$*-equivalent. Dually, given sequents of the form*

$$f :: \Gamma_1 \to \Gamma_2, \gamma : X \qquad g :: \Delta_1 \to \delta : Y, \Delta_2 \qquad h :: \gamma : X, \Phi_1, \delta : Y \to \Phi_2$$

the composites $f ;_\gamma (g ;_\delta h)$ *and* $g ;_\delta (f ;_\gamma h)$ *are* $\sim$*-equivalent.*

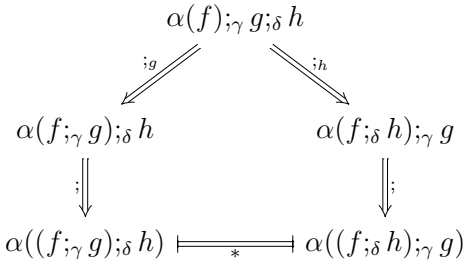
Proof. The proof proceeds by structural induction on f, g, and h; without loss of generality we may assume that f, g, and h are cut free. Without explicitly mentioning where, duality will be used to reduce the number of cases presented. In the following

$$f ;_\gamma g ;_\delta h \xRightarrow{;_g} (f ;_\gamma g) ;_\delta h \quad \text{and} \quad f ;_\gamma g ;_\delta h \xRightarrow{;_h} (f ;_\delta h) ;_\gamma g$$

will be used respectively to indicate composing first with g and composing first with h.

Notice that f may not be 1_X as it requires at least two codomain channels. So suppose that f is of the form $\alpha(f)$. If $f = \alpha\{\,\}$ then both composites will reduce to $\alpha\{\,\}$. The term f may not be the empty rtensor since, as noted above, it requires at least two codomain channels. If $I \neq \emptyset$ then in the

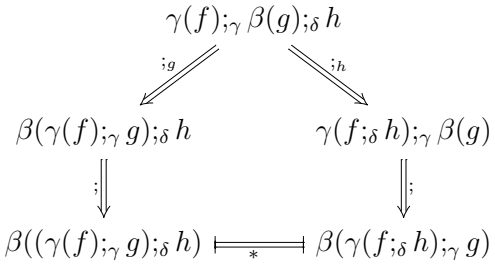
reduction diagram



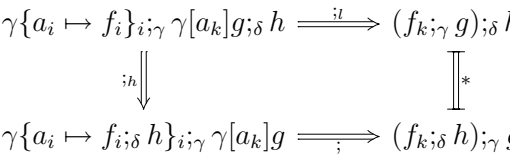
the composites are moved onto smaller terms, which by induction satisfy the interchange law, and hence, both composites are $\sim$-equivalent.

It remains to examine the case where f is of the form $\gamma(f)$. There are two cases corresponding to whether g is of the form $\beta(g)$, where $\beta \neq \gamma$ and $\beta \neq \delta$, or $\gamma(g)$. (The dual rewrites will suffice for the $\beta = \delta$ case.) In the first case if $\beta(g) = \beta\{\,\}$ then both composites will reduce to $\beta\{\,\}$. If $\beta(g) = \beta\langle\,\rangle$ (empty lpar) then both composites will reduce to $f;_\delta h$.

Now suppose that $\beta(g)$ is not a nullary operation. The following reduction diagram shows that both composites are $\sim$-equivalent (by induction):



In the second case ($\beta = \gamma$) there is only one possible choice: composition with g is an application of the rewrite (11). Explicitly,



which shows that the composites are $\sim$-equivalent.

If all of f, g, and h are atomic then composition satisfies the interchange law as it must satisfy the interchange law in the underlying polycategory. If

some of f, g, and h are atomic a quick check of the possibilities will show that one ends up with a case similar to one of the cases above.

Thus, composition satisfies the interchange law. □