PROOF EQUIVALENCE IN MLL IS PSPACE-COMPLETE

WILLEM HEIJLTJES AND ROBIN HOUSTON

ABSTRACT. MLL proof equivalence is the problem of deciding whether two proofs in multiplicative linear logic are related by a series of inference permutations. It is also known as the word problem for ∗-autonomous categories. Previous work has shown the problem to be equivalent to a rewiring problem on proof nets, which are not canonical for full MLL due to the presence of the two units. Drawing from recent work on reconfiguration problems, in this paper it is shown that MLL proof equivalence is PSPACE-complete, using a reduction from Nondeterministic Constraint Logic. An important consequence of the result is that the existence of a satisfactory notion of proof nets for MLL with units is ruled out (under current complexity assumptions).

1. INTRODUCTION

Sequent calculus was originally introduced by Gentzen as a way to better understand the properties of natural deduction – in particular the eliminability of cut. In this view, a sequent calculus proof gives a recipe for the construction of a natural deduction proof. Linear logic was formulated as a sequent calculus system, and doesn't have a corresponding natural deduction in the classical sense. The role of natural deduction is taken instead by proof nets. Each sequent calculus proof gives rise to a proof net. Furthermore, at least in the case of multiplicative linear logic without units, proof nets are canonical in the sense that two sequent calculus proofs give rise to the same proof net if and only if they are equivalent.

Proof nets do not work so well when the logical units 1 and ⊥ are included. Although proof nets can still be defined in this case, they are no longer canonical. Firstly the translation from proofs to proof nets is not canonical, so a proof may be translated to several possible proof nets with no privileged choice. Also, equivalent proofs are not in general translated to the same possible proof nets. So proof nets lose their principal advantage, compared to proofs, of being canonical, and are subject to a non-trivial equivalence relation.

Equivalence of proofs (in the sense we are interested in here) was first considered by Lambek [22], who introduced the idea of category-theoretic logic and used it to define a natural notion of equivalence of proofs.

Proof nets were introduced at the birth of linear logic by Girard [10], and the correctness criterion was simplified by Danos and Regnier [7]. These proof nets did not include the logical units, and are canonical for multiplicative linear logic. Canonical proof nets also exist for larger fragments of linear logic, such as the combined multiplicative-additive fragment without units [16], and the additive fragment, including the additive units [13]. Canonical proof nets may also be defined for *intuitionistic* multiplicative linear logic with units.

Proof nets incorporating units were defined by Blute, Cockett, Seely and Trimble [30, 4], though these proof nets cease to be canonical. The 'joker games' of Koh and Ong [27] have a very similar structure to proof nets, and include units. Another explicit treatment of proof nets for MLL with units is given by Lamarche and Straßburger [21], and later a particularly simple and elegant version by Hughes [18, 17].

The idea of the present work is to look at proof equivalence through the lens of computational complexity, and use this to settle the question of whether there can be canonical proof nets for MLL with units. The canonical proof nets for MLL without units give an efficient decision procedure for proof equivalence: to decide whether two such proofs are equivalent, it is sufficient to translate both to proof nets and compare the proof nets for equality. We show that the corresponding decision procedure for MLL *with* units is PSPACE-complete, which is generally supposed to preclude the existence of a polynomial-time algorithm for this problem. So there can be no canonical proof nets in the usual sense: if the proof nets are syntactically equal just when their corresponding proofs are equivalent, then the translation from proofs to proof nets must be intractable.

Since there can be no canonical representation of multiplicative linear logic proofs in accordance with the categorical semantics, it is tempting to consider other ways of defining the identity of proofs. The literature contains several interesting ideas in this direction, such as *polarised proof nets* [23], *focused proof nets* [1], *multi-focused proofs* [5], and Paul-André Melliès's semantically appealing *tensorial logic* and *dialogue categories* [26].

Our proof of PSPACE-completeness uses the *nondeterministic constraint logic* of [11, 8, 12]. This is a graph rewriting system introduced as a tool for use in complexity proofs, originally for games and puzzles.

It is not uncommon for an NP-complete problem to have an associated reconfiguration problem that is PSPACE-complete [19]; an example of this is SAT-reconfiguration. MLL proof equivalence may be regarded as the reconfiguration problem associated with MLL proof search, which is NP-complete [20, 25].

2. MLL

In this section, we will give a brief introduction to multiplicative linear logic (MLL). For simplicity of exposition, we will work in the unit-only fragment of MLL. The formulae of this fragment are given by the following grammar.

$$A, B, C \quad ::= \quad \bot \quad | \quad 1 \quad | \quad A \,\bindnasrepma\, B \quad | \quad A \otimes B$$

The connectives tensor $(\otimes)$ and par $(\bindnasrepma)$ will be considered up to associativity, and *duality*, indicated $A^\star$, is via DeMorgan. A *sequent* Γ, Δ, Λ will be a multiset of formulae. To distinguish different occurrences of a formula within a sequent, the connectives and units in a sequent will be *named* with distinct elements from an arbitrary set of names, e.g.:

$$1_a \,\bindnasrepma\, _b\, 1_c \ , \ \bot_d \otimes_e \bot_f \ .$$

This simple technical device is introduced for a dual purpose. Firstly, it distinguishes proofs that are equal up to a symmetric exchange of formulae, such as the *identity* and the *twist* proof of the above sequent, while the sequent rules can be given using standard multiset sequents, i.e. without the need for explicit *exchange* rules. Secondly, it creates an easy way to extract proof nets, as graphs that use the names of connectives and units as vertices.

Within a proof, the names of units and connectives are preserved through inferences. Where convenient, we may leave names implicit.

$$\dfrac{\Gamma}{\Gamma, \bot} \bot \qquad \overline{\ 1\ }\, 1 \qquad \dfrac{\Gamma, A, B}{\Gamma, A \,\invamp\, B}\,\invamp \qquad \dfrac{\Gamma, A \qquad \Delta, B}{\Gamma, \Delta, A \otimes B}\,\otimes \qquad \Bigg| \qquad \overline{A, A^{\star}}\,\mathsf{ax} \qquad \dfrac{\Gamma, A \qquad \Delta, A^{\star}}{\Gamma, \Delta}\,\mathsf{cut}$$

Figure 1: Inference rules for unit-only MLL

$$\overline{1, \bot}\,\mathsf{ax} \quad \rightsquigarrow \quad \dfrac{\overline{\ 1\ }\, 1}{1, \bot}\,\bot \qquad\qquad \overline{A \otimes B, A^{\star} \,\invamp\, B^{\star}}\,\mathsf{ax} \quad \rightsquigarrow \quad \dfrac{\dfrac{\overline{A, A^{\star}}\,\mathsf{ax} \qquad \overline{B, B^{\star}}\,\mathsf{ax}}{A \otimes B, A^{\star}, B^{\star}}\,\otimes}{A \otimes B, A^{\star} \,\invamp\, B^{\star}}\,\invamp$$

Figure 2: Identity-elimination transformations

$$\dfrac{\overline{\ 1\ }\, 1 \qquad \dfrac{\Gamma}{\Gamma, \bot}\,\bot}{\Gamma}\,\mathsf{cut} \quad \rightsquigarrow \quad \Gamma$$

$$\dfrac{\dfrac{\Gamma, A \qquad \Delta, B}{\Gamma, \Delta, A \otimes B}\,\otimes \qquad \dfrac{\Lambda, A^{\star}, B^{\star}}{\Lambda, A^{\star} \,\invamp\, B^{\star}}\,\invamp}{\Gamma, \Delta, \Lambda}\,\mathsf{cut} \quad \rightsquigarrow \quad \dfrac{\Gamma, A \qquad \dfrac{\Delta, B \qquad \Lambda, A^{\star}, B^{\star}}{\Delta, \Lambda, A^{\star}}\,\mathsf{cut}}{\Gamma, \Delta, \Lambda}\,\mathsf{cut}$$

Figure 3: Cut-elimination transformations

A one-sided sequent calculus for unit-only MLL is given in Figure 1. The rules for identity (ax) and cut (cut) are admissible, and are eliminated via the identity-elimination and cut-elimination transformations displayed in Figure 2 and Figure 3 respectively. The cut-elimination process in addition makes essential use of *permutations* of inference rules, displayed in Figure 4, to match up a cut-rule with the two inferences that introduce both cut-formulae. Note that the symmetric variants of various transformations have been omitted from the figures, as well as permutations involving the cut-rule, which are similar to those of the tensor rule.

The proof system we shall be working with is that consisting of just the introduction rules for the two units and the two connectives, the four rules $(\bot)$, (1), $(\invamp)$, and $(\otimes)$ of Figure 1. The problem we consider is the following.

Definition 1 (MLL proof equivalence)**.** *Equivalence* $(\sim)$ of proofs in (cut-free, identity-free, unit-only) multiplicative linear logic is the congruence generated by the permutations given in Figure 4. *MLL proof equivalence* is the problem of deciding whether two given proofs are equivalent.

Star-autonomous categories. The permutations of sequent proofs are exactly the identifications imposed by the categorical semantics of MLL, star-autonomous categories [2] (and semi-star-autonomous categories [15, 14] for MLL without units). Proof equivalence for MLL is therefore equivalent to the *word problem* for star-autonomous categories: the problem whether two term representations denote the same morphism in any star-autonomous category.

$$\cfrac{\cfrac{\Gamma}{\Gamma, \bot_a} \bot}{\Gamma, \bot_a, \bot_b} \bot \quad \sim \quad \cfrac{\cfrac{\Gamma}{\Gamma, \bot_b} \bot}{\Gamma, \bot_a, \bot_b} \bot \qquad\qquad \cfrac{\cfrac{\Gamma, A, B}{\Gamma, A \,\mathparagraph\, B} \mathparagraph}{\Gamma, A \,\mathparagraph\, B, \bot} \bot \quad \sim \quad \cfrac{\cfrac{\Gamma, A, B}{\Gamma, A, B, \bot} \bot}{\Gamma, A \,\mathparagraph\, B, \bot} \mathparagraph$$

$$\cfrac{\cfrac{\Gamma, A}{\Gamma, A, \bot} \bot \qquad \Delta, B}{\Gamma, \Delta, A \otimes B, \bot} \otimes \quad \sim \quad \cfrac{\cfrac{\Gamma, A \qquad \Delta, B}{\Gamma, \Delta, A \otimes B} \otimes}{\Gamma, \Delta, A \otimes B, \bot} \bot \quad \sim \quad \cfrac{\Gamma, A \qquad \cfrac{\Delta, B}{\Delta, B, \bot} \bot}{\Gamma, \Delta, A \otimes B, \bot} \otimes$$

$$\cfrac{\cfrac{\Gamma, A, B, C, D}{\Gamma, A \,\mathparagraph\, B, C, D} \mathparagraph}{\Gamma, A \,\mathparagraph\, B, C \,\mathparagraph\, D} \mathparagraph \quad \sim \quad \cfrac{\cfrac{\Gamma, A, B, C, D}{\Gamma, A, B, C \,\mathparagraph\, D} \mathparagraph}{\Gamma, A \,\mathparagraph\, B, C \,\mathparagraph\, D} \mathparagraph$$

$$\cfrac{\cfrac{\Gamma, A \qquad \Delta, B, C, D}{\Gamma, \Delta, A \otimes B, C, D} \otimes}{\Gamma, \Delta, A \otimes B, C \,\mathparagraph\, D} \mathparagraph \quad \sim \quad \cfrac{\Gamma, A \qquad \cfrac{\Delta, B, C, D}{\Delta, B, C \,\mathparagraph\, D} \mathparagraph}{\Gamma, \Delta, A \otimes B, C \,\mathparagraph\, D} \otimes$$

$$\cfrac{\Gamma, A \qquad \cfrac{\Delta, B, C \qquad \Lambda, D}{\Delta, \Lambda, B, C \otimes D} \otimes}{\Gamma, \Delta, \Lambda, A \otimes B, C \otimes D} \otimes \quad \sim \quad \cfrac{\cfrac{\Gamma, A \qquad \Delta, B, C}{\Gamma, \Delta, A \otimes B, C} \otimes \qquad \Lambda, D}{\Gamma, \Delta, \Lambda, A \otimes B, C \otimes D} \otimes$$

Figure 4: Permutations

Proof nets

For MLL without units, Girard's original proof nets [10] are canonical: two proofs translate into the same proof net if and only if they are equivalent. Since the translation from proofs to proof nets and the syntactic comparison of proof nets are both effectively computable (linear-time), proof nets provide an effective solution to the proof equivalence problem for MLL without units.

There have been several proposals for (non-canonical) multiplicative proof nets with units [4, 27, 21, 17], each providing a different take on the proof equivalence problem for MLL. We will use the formulation by Hughes [17], which reduces proof equivalence to a simple rewiring relation, as the basis for our PSPACE-hardness reduction. In this section we will discuss how this notion of proof net arises from the sequent calculus, and introduce a compact notation for it.

Definition 2 (Proof nets)**.** For a sequent Γ,
- a *linking* ℓ is a function from the names of $\bot$-subformulae in Γ to the names of Γ;
- a *switching graph* for Γ and ℓ is an undirected graph over the names of Γ with:
  - for every subformula $A_a \otimes_c B_b$ the edges $c-a$ and $c-b$,
  - for every subformula $A_a \,\mathparagraph_c\, B_b$ either the edge $c-a$ or the edge $c-b$, and
  - for every subformula $\bot_a$ the edge $a-\ell(a)$;
- a linking ℓ is *correct* if every switching graph is acyclic and connected;
- a *proof net* (Γ, ℓ) consists of the sequent Γ and a correct linking ℓ for Γ.

An example proof net is given in Figure 5. The edges from a $\mathparagraph$-node, which are subject to being switched, are drawn as dashed lines, while edges from a $\otimes$-node are solid lines. An
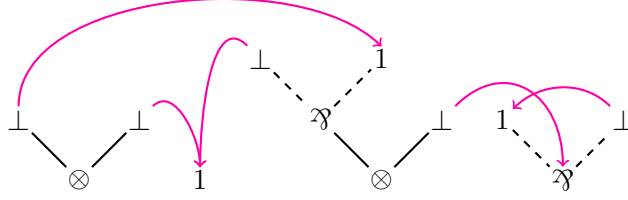
Figure 5: An example proof net

$$\cfrac{\cfrac{\overline{1_b}}{\bot, 1_b} \qquad \cfrac{\cfrac{\overline{1_a}}{1_a, \bot}}{\cfrac{\bot, 1_a, \bot}{\bot \otimes \bot, 1_a, \bot, 1_b}} \qquad \cfrac{\cfrac{\overline{1_c}}{1_c, \bot}}{\cfrac{1_c \mathbin{\rotatebox[origin=c]{180}{\&}} \bot}{\bot, 1_c \mathbin{\rotatebox[origin=c]{180}{\&}} \bot}}}{\cfrac{\bot \otimes \bot, 1_a, \bot \mathbin{\rotatebox[origin=c]{180}{\&}} 1_b}{\bot \otimes \bot, 1_a, (\bot \mathbin{\rotatebox[origin=c]{180}{\&}} 1_b) \otimes \bot, 1_c \mathbin{\rotatebox[origin=c]{180}{\&}} \bot}}$$

Figure 6: Interpreting a proof as the proof net of Figure 5

edge $a - \ell(a)$ in a proof net or switching graph is called a *link* or a *jump*, and may also be indicated $a \to \ell(a)$, to match the illustrations. Note that unlike the axiom links of proof nets without units, a jump may connect to any connective or unit, and multiple jumps may point to the same position. A path between vertices a and b in a switching graph is indicated as $a {\cdots} b$.

The present notion of proof net can be seen as a direct interpretation of the sequent calculus, in the following way. As there are inequivalent proofs of the sequent $\bot \otimes \bot, 1 \mathbin{\rotatebox[origin=c]{180}{\&}} 1$, some way of attaching the $\bot$-formulae to the remainder of the proof net appears necessary. The introduction rule for $\bot$, in Figure 1, attaches a $\bot$-formula to a sequent—but sequents are not explicit in proof nets. The notion of a jump, which attaches the $\bot$ to an arbitrary subformula in the sequent Γ, is then a natural generalisation of the axiom links of unit-free MLL proof nets. This gives the following translation from proofs to proof nets.

Definition 3. The relation $(\mapsto)$ interprets a proof Π by a linking ℓ as follows: $\Pi \mapsto \ell$ if for each $\bot_a$ in Π, $\ell(a)$ is a name in Γ, where Γ is the context of the inference introducing $\bot_a$, as illustrated below.

$$\frac{\Gamma}{\Gamma, \bot_a} \bot$$

An example is given in Figure 6, where a proof is interpreted as the proof net of Figure 5, by indicating the chosen jumps in the conclusion of each $\bot$-inference. (In the example, to trace formulae through inferences, the five $\bot$-formulae are kept in order from left to right, while the three 1-formulae have explicit names.)

Proposition 4 ([7, 21]). *If $\Pi \mapsto \ell$ and Π has conclusion Γ, then (Γ, ℓ) is a proof net. For any proof net (Γ, ℓ) there is a proof Π of Γ such that $\Pi \mapsto \ell$ (sequentialisation).*

Remark. *The proof nets of [4] may appear to be formulated in quite a distinct way from our presentation, in particular in that their jumps from $\bot$-vertices connect to edges, rather than*
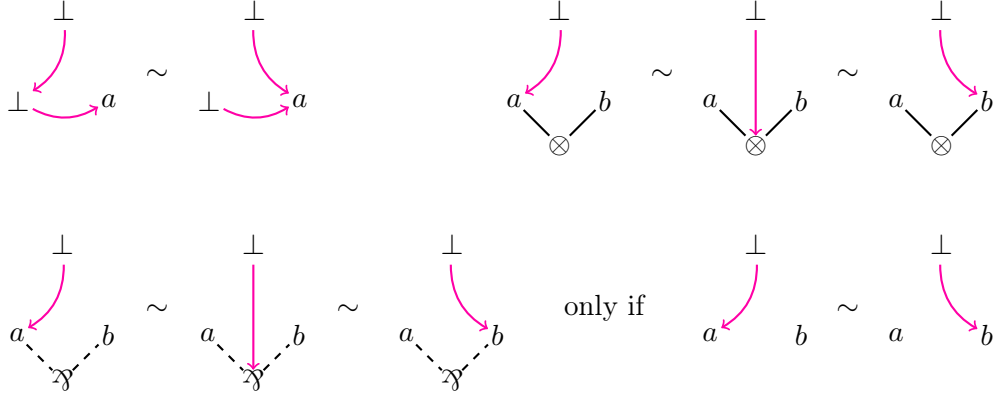
Figure 7: Rewiring steps

vertices. The difference is only superficial: in their proof nets, vertices represent inferences, and edges represent formulae; in both presentations, therefore, jumps may be seen to attach to formulae. This way of attaching jumps (to formulae, rather than inferences) is the natural choice also in their setting, which is motivated categorically, where $\perp$-introduction corresponds to the isomorphism $A \to A \,\bindnasrepma\, \perp$.

Rewiring. The use of proof nets factors out the bottom three permutations in Figure 4, $(\bindnasrepma - \bindnasrepma)$, $(\otimes - \bindnasrepma)$, and $(\otimes - \otimes)$. The remaining permutations, involving the $\perp$-introduction rule, impose an equivalence on proof nets, consisting of the *rewiring* of jumps from one target to another. This is defined as follows.

Definition 5. A *rewiring* ($\overset{1}{\sim}$) between proof nets changes the target of exactly one jump. *Equivalence* ($\sim$) of proof nets over a sequent Γ is the equivalence generated by rewiring.

We will write $\ell[a{\to}b]$ for the linking where $a{\to}b$, and $c{\to}\ell(c)$ for any $\perp_c$ other than a. Then in a rewiring $\ell \overset{1}{\sim} \ell[a{\to}b]$, the jump being rewired is that from a.

The above definition, which is due to Hughes [17], gives a "big-step" rewrite relation for equivalence. In turn, this rewiring relation is generated by smaller local steps, which correspond more closely to the permutations in the sequent calculus. These smaller steps are illustrated in Figure 7. In the top row, a jump can be rewired across another jump and across a tensor without breaking correctness; these rewiring steps can be seen to correspond to $(\perp - \perp)$ and $(\otimes - \perp)$ permutations. In the bottom row, rewiring a jump into the par, from a or b, only preserves correctness if the jump can be rewired between a and b independently of the par. Intuitively, this corresponds to the $(\bindnasrepma - \perp)$ permutation in the following way: the positions that a jump may rewire to correspond to the context of the rule introducing the $\perp$; then to have the premise of the $\bindnasrepma$-introduction as its context, the jump should be able to rewire to both a and b.

That the rewiring relation is generated by the small-step relation in Figure 7 is a main theorem in [17].

Proposition 6 ([17])**.** *Rewiring ($\overset{1}{\sim}$) is generated by the local steps in Figure 7.*

A further refinement in the proof nets introduced by Hughes [17], is that jumps may be restricted to target only 1-formulae. For the remainder of the paper, unless otherwise indicated, we will assume that proof nets are of this form.

Proposition 7 ([17]). *Any proof net is equivalent to one where the codomain of ℓ is restricted to occurrences of the formula* 1.

A main technical advantage of this restriction is that cut-elimination can be performed in a one-shot operation, by path composition over jumps. Then, without the need to define cut-elimination by local transformations, cuts may be omitted from proof nets altogether, which justifies the present definition.

Finally, the following proposition states that equivalence is preserved when translating between proofs and proof nets, which means that MLL proof equivalence is equivalent to the problem of deciding equivalence of proof nets.

Proposition 8 ([4, 17]). *If $\Pi \Rrightarrow \ell$ and $\Pi' \Rrightarrow \ell'$, then $\Pi \sim \Pi'$ if and only if $\ell \sim \ell'$.*

Diagram notation. To concisely represent larger sequents and proof nets, we will introduce a compact diagrammatic notation. Briefly, the units 1 and $\bot$ are represented by a circle ($\circ$) and a disc ($\bullet$) respectively, a tensor ($\otimes$) will be indicated by an edge, and a par ($\parr$) will be indicated by a box. For example, the sequent below left is represented as a diagram below right.

$$\bot \otimes \bot \;,\; \bot \otimes \bot \;,\; 1 \;,\; (1 \parr 1 \parr 1) \otimes \bot$$

The salient features of a diagram are the connectedness by $\otimes$-edges and containment within a $\parr$-box. Respecting these, we will lay out a diagram in the plane in any way that is convenient, while ensuring (of course) that the layout is kept consistent when comparing different proof nets over the same sequent. More formally, then, sequents and formulae are interpreted as diagrams as follows:

$$1 \Rightarrow \circ, \quad \bot \Rightarrow \bullet, \quad \text{and if} \quad A_i \Rightarrow \boxed{A_i} \quad \text{then}$$

$$A_1 \otimes A_2 \otimes \ldots \otimes A_n \;\Rightarrow\; \boxed{A_1}\!-\!\boxed{A_2}\!-\cdots-\!\boxed{A_n} \quad \text{where no } A_i \text{ is a } \otimes\text{-formula}$$

$$A_1 \parr A_2 \parr \ldots \parr A_n \;\Rightarrow\; \boxed{\;\boxed{A_1}\;\boxed{A_2}\;\cdots\;\boxed{A_n}\;} \quad \text{where no } A_i \text{ is a } \parr\text{-formula}$$

$$A_1, A_2, \ldots, A_n \;\Rightarrow\; \boxed{A_1}\;\boxed{A_2}\;\cdots\;\boxed{A_n}$$

In the diagrammatic notation, as in the regular notation before, the jumps of a proof net are added to the sequent as coloured arrows. Per the restriction mentioned previously, jumps always connect a $\bot$-formula to a 1-formula. The example below gives a proof net for the

above sequent, presented in both notational styles.
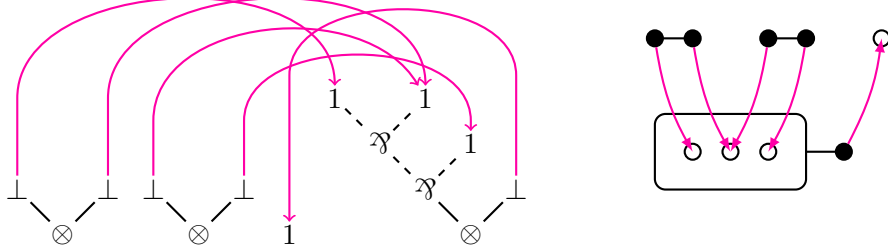
The above examples illustrate how the diagrammatic notation replaces the connectives by geometric features to allow for more convenient placement in the plane. The diagrams preserve the connectedness of non-switched edges, while switched components are placed within a box, so that a switching graph is obtained by replacing each box by an edge to exactly one of its components.

To lay out a sequent as a diagram in two dimensions may lead to ambiguity when multiple occurrences of the same formula are present, as is the case for the formula $\perp \otimes \perp$ in the above examples. Where necessary, this will be resolved by adding the names of chosen $\perp$-formulae and 1-formulae to a diagram. When discussing different proof nets over the same sequent, the sequent will be laid out in a consistent manner.

Figure 8 uses the diagrammatic notation to display the rewiring relation over the proof nets for the sequent $\perp \otimes \perp, 1, 1, 1, \perp \otimes \perp$, an interesting example also highlighted in [29]. There are 24 proof nets in total, forming two equivalence classes, each consisting of a single cyclic rewiring path.

3. Proof complexity

The problem of deciding whether a given MLL sequent is provable is NP-complete. This was shown first for an extension of MLL with weakening [24], and subsequently for unit-free MLL [20] and for unit-only MLL [25]—note that since MLL is conservative over both its unit-only and its unit-free fragment, each of the latter two implies NP-completeness for provability in full MLL.

Both Kanovich [20] and Lincoln and Winkler [25] use a reduction from (a variant of) the 3-Partition problem to show NP-hardness. Such reductions provide a good illustration of how the linearity of MLL can be used to encode combinatoric problems. In this section we will discuss a similar reduction from 3-Partition.

To encode simple numerical properties in MLL, our primary tools will be formulae of the form $\perp \otimes \ldots \otimes \perp$ and $1 \,\mathfrak{N} \ldots \mathfrak{N}\, 1$. In particular, two formulae A and B of the former kind may form a proof net with one C of the latter kind if and only if the total number of $\perp$-occurrences in A and B exceeds the number of 1-occurrences in C by 1: each 1-occurrence must be linked to by a $\perp$, and one additional $\perp$ is needed to connect A and B. We will capture this more generally by counting the number of $\perp$-occurrences in A as $i + 1$, so that if $j + 1$ is the number of $\perp$-occurrences in B, the number of 1-occurrences in C must be $i + j + 1$. Below left is an illustration for $i = 2$ and $j = 3$. In the following, we will introduce
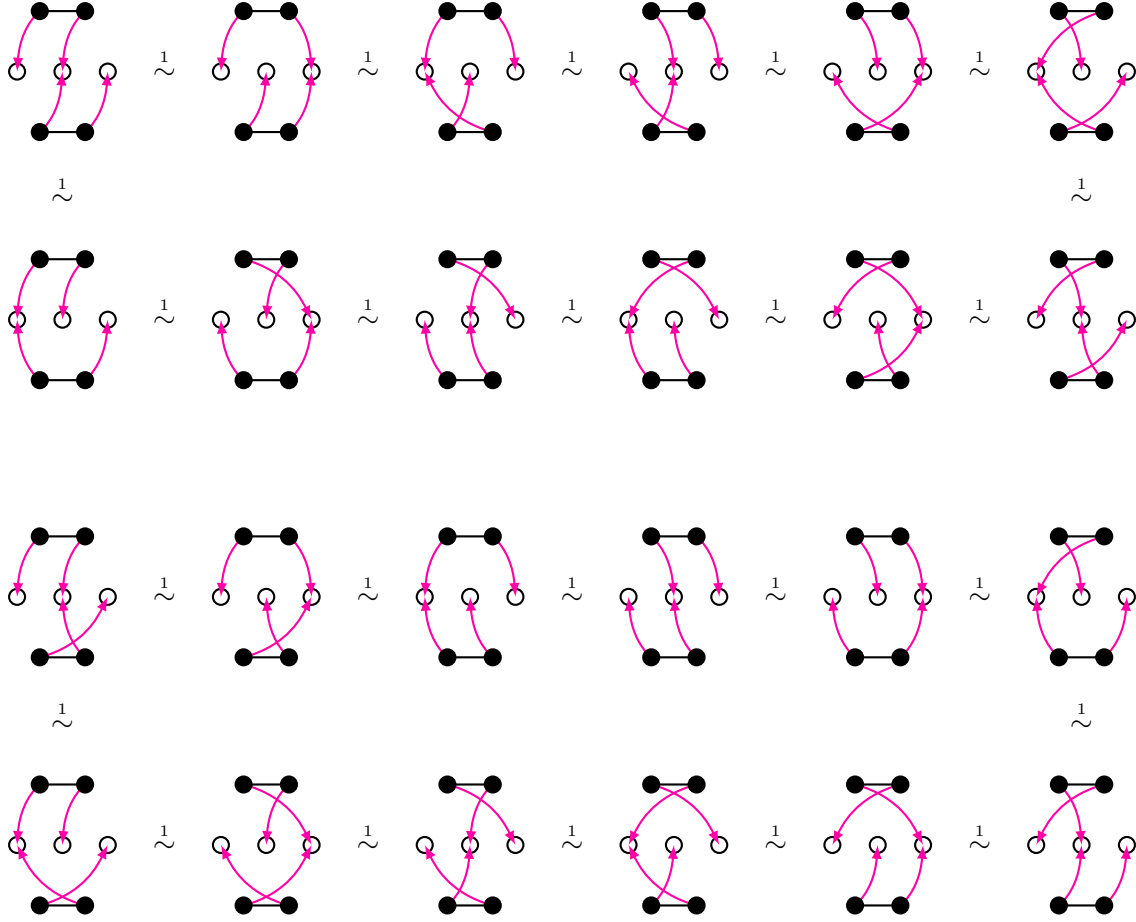
Figure 8: All proof nets for the sequent $\perp \otimes \perp, 1, 1, 1, \perp \otimes \perp$

an abbreviated notation, illustrated below right.
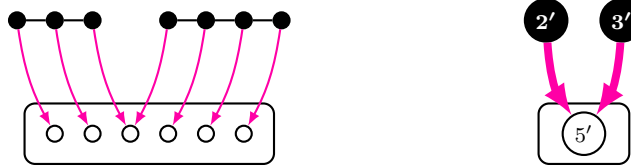
By A^n we will denote the sequent consisting of n occurrences of a formula A. Given a sequent $\Gamma = A_1, \ldots, A_n$ we will write $\bigotimes \Gamma$ for $A_1 \otimes \cdots \otimes A_n$, and $\bigparr \Gamma$ for $A_1 \parr \cdots \parr A_n$. In diagrammatic notation, a big disc labelled n' will represent the formula $\bigotimes(\perp^{n+1})$, and a big circle labelled n' will represent the sequent 1^{n+1}. A formula $\bigparr(1^{n+1})$ is represented by a disc enclosed in a box. This extends our diagrammatic notation as follows.
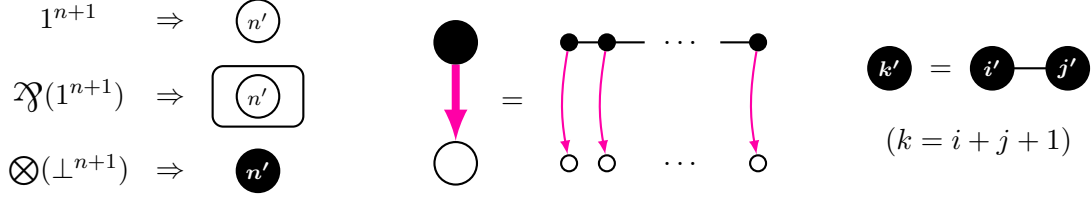
A linking between formulae $\bigotimes(\perp^n)$ and a sequent 1^n will be represented by a wide arrow, as illustrated above center. For the moment, we will not distinguish between the $n!$ different ways in which such a linking can be made, but when the distinction becomes relevant we will make an explicit choice. Above right, to link a formula $\bigotimes(\perp^{k+1})$ to several other formulae, we may display the formula as two discs according to the given arithmetic, representing the formula $\bigotimes(\perp^{i+1}) \otimes \bigotimes(\perp^{j+1})$.

The primary intention behind the chosen notation is to make the arithmetic of connecting formulae $\bigotimes(\perp^n)$ to formulae $\mathfrak{P}(1^n)$ apparent in the illustrations. Specifically, the following is a proof net if and only if $n = i_1 + \ldots + i_k$.
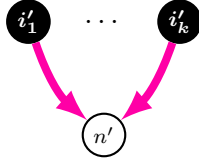
Encoding 3-Partition. An instance of the 3-Partition problem is given by a multiset $I = \{i_1, \ldots, i_{3n}\}$ of $3n$ natural number values with total sum $\sum I = nk$. The decision problem asks whether a partitioning exists of I into n triples (i_a, i_b, i_c) each with sum k. It may be assumed that each $i \in I$ lies between $k/4$ and $k/2$, so that any partitioning into *subsets* with sum k is a solution, since each subset of sum k necessarily has 3 elements. The 3-Partition problem is strongly NP-complete [9], which means it remains NP-complete with a unary encoding of the values in I.

We will give a simple encoding of 3-partition in MLL proof search, similar to those of Kanovich [20] and Lincoln and Winkler [25]. A 3-Partition instance I is encoded by a sequent consisting of:

- a formula $\bigotimes(\perp^{i+1})$ for each $i \in I$, and
- the formula $\bigotimes(K^n)$ where $K = \mathfrak{P}(1^{k+1})$.

A solution to the 3-partition problem is encoded as a proof net as follows. If the numbers a, b, and c are assigned to a triple corresponding to a formula K_j, then the corresponding formulae have $a+1$, $b+1$, and $c+1$ occurrences of $\perp$, and $k+1 = a+b+c+1$ occurrences of 1, respectively. Then as described previously, there is a proof net for these formulae, obtained by attaching each jump to a distinct 1, with the exception of three jumps, one each from a, b, and c, overlapping on exactly one intance.

The construction is illustrated below, where the numbers i are conveniently ordered to give the solution (i_1, i_2, i_3), (i_4, i_5, i_6), etc.
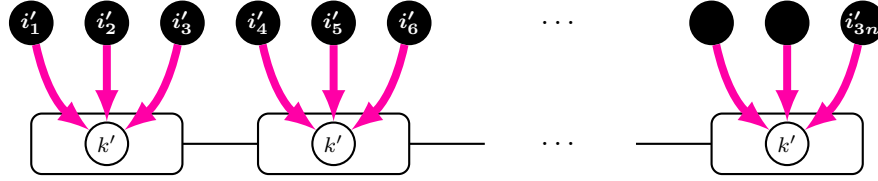
The correctness condition ensures that every proof net on a sequent encoding a 3-partition problem gives a solution that problem. If two jumps from a formula encoding a number i connect to different formulae K_j and K_m, a cycle is created immediately. Thus, the jumps from each i must all connect to the same K, and the number of occurrences of $\perp$ and 1 in the encoding then ensures that when the formulae for i_a, i_b, and i_c, connect to one formulae K, their values sum to k, $i_a + i_b + i_c = k$.

The encoding of 3-partition serves to illustrate how simple numerical constraints can be encoded in proof nets: by matching up a given number of $\perp$-occurrences and 1-occurrences and by separating formulae of the form $\mathcal{P}(1^n)$ by a tensor. This idea will form the basis of our encoding of non-deterministic constraint logic.

4. Constraint logic

Non-deterministic constraint logic (NCL) [11, 8, 12] is a simple graph-rewriting formalism, introduced as a convenient tool for PSPACE-hardness reduction. It is exactly for this purpose that we shall use it.

The graphs of NCL, called *constraint graphs*, have directed, weighted edges, and a rewrite step (or *move*) consists of reversing the direction of exactly one edge. Each vertex in a constraint graph has an *inflow constraint*, a natural number value, and moves are required to preserve the property that the *inflow* at each vertex, the sum weight of its incoming edges, never drops below the inflow constraint (i.e. the inflow constraint dictates a *minimum* inflow).

The specific problem we will use is the *configuration–to–configuration* problem, which asks whether a path of rewrite steps exists between two constraint graphs. This problem is PSPACE-complete, and remains so under various restrictions: inflow constraints can be fixed at the value 2, edge weights can be restricted to the value 1 and 2, the graphs can be required to be planar, and loops and double edges (edges connecting the same 2 vertices) can be prohibited, for example.

We will need no such restrictions, and will encode the general case. However, for simplicity of exposition, in the examples of constraint graphs in this section we will use the following conventions. Edges will have weight 1 or 2; those of weight 1 are drawn as thin red arrows, and those of weight 2 as thicker blue arrows. Vertices with inflow constraint 2 are drawn as grey circles. For convenience, we will assume that dangling edges are connected to vertices of weight zero, which are omitted from illustrations. These zero-inflow vertices can be seen as "connection points", where the graph may be connected to a wider network of graphs.

An example rewrite sequence is given in Figure 9. This particular specimen encodes a logical conjunction, and is not unlike an AND-gate: in order to invert the weight 2 edge on the right, both weight 1 edges on the left must be inverted first.

In the formal definition, a constraint graph is an *undirected* graph, for which a *configuration* assigns an orientation to each edge. It is useful for us to generalise the notion of
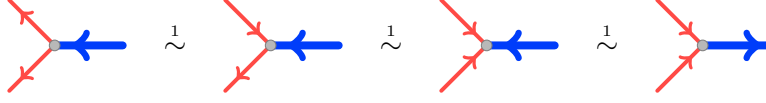
Figure 9: A series of reconfiguration steps in a constraint graph

configuration a little: we will allow *partial* configurations, where the direction of edges may be left undefined, as long as the inflow constraints are satisfied by the directed edges. We will use $\star$ to indicate an undefined value, to avoid overloading $\perp$.

Definition 9. A *constraint graph* $G = (V, E, c, v, w)$ consists of:
- V a set of vertices,
- E a set of edges,
- $c \colon V \to \mathbb{N}$ an *inflow constraint* on each vertes,
- $v \colon E \to \mathcal{P}(V)$ where $|v(e)| \in \{1, 2\}$ for $e \in E$, a set of 1 or 2 vertices for each edge,
- $w \colon E \to \mathbb{N}$ a *weight* on each edge.

Definition 10. A *(partial) configuration* for a constraint graph $G = (V, E, c, v, w)$ is a (partial) function $\gamma \colon E \to V$ such that
- for every edge e, $\gamma(e) \in v(e) \cup \{\star\}$,
- for every vertex v,
$$c(v) \leq \sum \{w(e) \mid \gamma(e) = v\} .$$

In the above definition, the first condition states that the weight of an edge may be assigned only to one of its vertices, or left undefined, but not to any other vertex in the constraint graph. The second condition states that the total weight of its incoming edges of a vertex must be at least its inflow constraint.

We will write $\gamma[e \mapsto v]$ for the partial configuration that directs e to v, and any other edge e' to $\gamma(e')$. An edge that may be reversed will be called *mobile*. Formally, an edge e in γ is *mobile* if $\gamma[e \mapsto \star]$ is a partial configuration.

Definition 11. A *reconfiguration step* ($\overset{1}{\sim}$) relates two partial configurations for G that differ in value (or definedness) on exactly one edge:
$$\gamma[e \mapsto u] \overset{1}{\sim} \gamma[e \mapsto w] \qquad \text{if } u \neq w \text{ and } u, w \in v(e) \cup \{\star\} .$$
The reflexive–transitive closure of ($\overset{1}{\sim}$) will be denoted ($\sim$).

Non-deterministic constraint graph reconfiguration or *NCG-reconfiguration* is the problem of deciding whether two total configurations of a constraint graph are connected by a sequence of reconfiguration steps. This is the problem shown to be PSPACE-complete by Hearne and Demaine [12]:

Theorem 12 ([12], Theorem 5.15)**.** *NCG-reconfiguration is PSPACE-complete.*

To show that this result extends to our setting, with partial configurations, we have the following proposition.

Proposition 13. *For total configurations γ and δ, if $\gamma \sim \delta$ then γ and δ are also connected by a sequence of reconfiguration steps over total configurations only.*
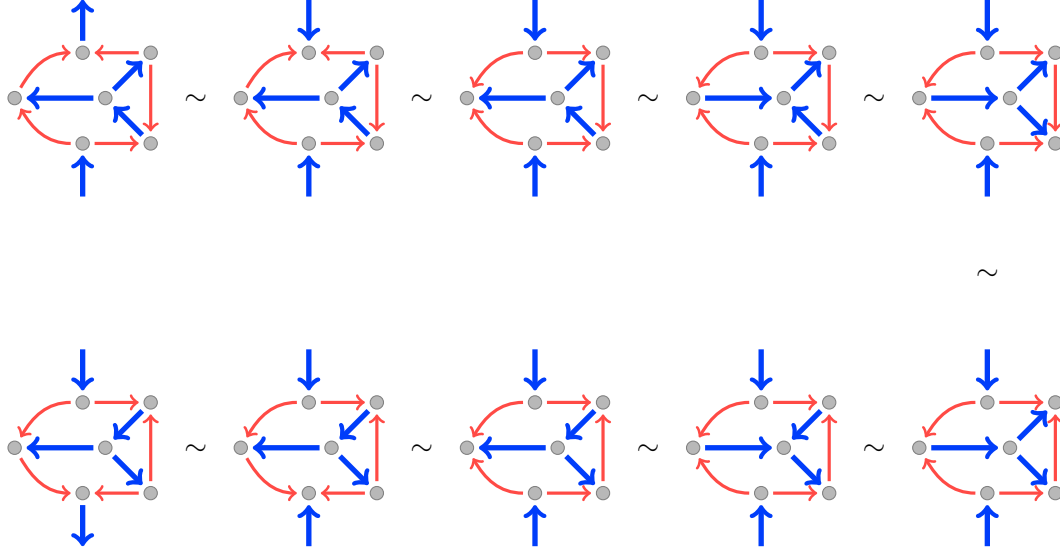
Figure 10: A reconfiguration path requiring two inversions of a single edge

Proof. By the following two observations: firstly, if $\gamma \overset{1}{\sim} \delta$ for partial configurations, then these may be completed to total configurations $\gamma' \overset{1}{\sim} \delta'$ or $\gamma' = \delta'$; and secondly, if γ' and γ'' are total configurations that both agree with a partial configuration γ where the latter is defined, then γ' and γ'' are connected by reversing the edges on which they disagree one after another. $\qquad\square$

The PSPACE-completeness of NCG-reconfiguration may seem surprising at first. To give a feel for the dynamics that make this problem PSPACE-hard, we shall exhibit an example where an exponentially-long reconfiguration sequence is needed. The existence of such rewiring sequences is an essential property for the hardness result: if reconfiguration sequences could be given a polynomial bound, they would be polynomial-time witnesses, placing the problem within NP.

The example is then as follows. Figure 10 gives a reconfiguration sequence, clockwise from the top left, on a constraint graph with two connection points, top and bottom. As such, the graph is meant to replace a single edge from top to bottom, and the reconfiguration sequence inverts the direction of this would-be edge. The key point is that in doing so, the central horizontal edge must be inverted *twice*. Then by nesting the graph within itself, where one graph replaces the central edge of another, as illustrated in Figure 11, a graph is created with an edge that requires 4 inversions. Each added level of nesting doubles the number of inversions that the central edge must make, while growing the graph by only a constant factor. Thus, the length of a rewiring path may grow exponentially with the size of the graph.
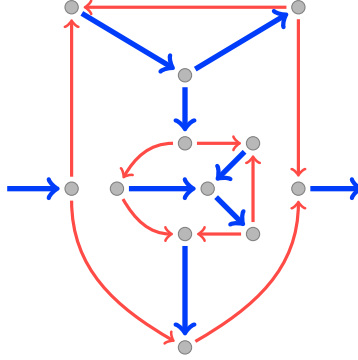
Figure 11: Nesting graphs to grow reconfiguration paths exponentially

5. Encoding constraint logic

We will demonstrate the PSPACE-hardness of MLL proof equivalence by an encoding of NCG-reconfiguration in MLL proof nets. A sequent will encode a constraint graph, and a proof net a configuration, such that rewiring the proof net corresponds to reconfiguration.

The natural basis of the encoding is the use of jumps to indicate where the weight of an edge is allocated. The main problem is then how to restrict the movement of such jumps, so that they may assign weight only to the vertices of one particular edge, and not to any other vertices. The central design idea of our encoding is to use the combinatorics of connecting formulae of the form $\bigotimes(\bot^n)$ to formulae $\bigparr(1^m)$ to solve this problem.

The two basic components of our encoding will be *weight elements*, representing the weight value of edges, and *constraint elements*, which encode the inflow constraint of vertices. For now, we will work under the assumption that each unit of weight on an edge is encoded by exactly one weight element—later, we will amend this to use multiple weight elements for each unit of weight. A weight element will be a construction over a number of $\bot$-formulae, whose jumps may connect to the 1-formulae of a constraint element.

Below top is a weight element, below bottom a constraint element.

$$\boxed{\;i'\quad j'\quad k'\;} \;=\; \bigotimes(\bot^{i+1}) \;\bigparr\; \bigotimes(\bot^{j+1}) \;\bigparr\; \bigotimes(\bot^{k+1})$$

$$\boxed{m'}\!-\!\boxed{n'} \;=\; \bigparr(1^{m+1}) \;\otimes\; \bigparr(1^{n+1})$$

For all edges and vertices in the encoding of a constraint graph, the sum $i + j + k = m + n$ will be the same—this way, a priori any weight element may connect to any constraint element. The value m will be unique for each vertex, and used for all constraint elements in the encoding of that vertex. The weight element above is for an edge connecting the two vertices for which $m = i$ and $m = i + j$: with those constraint elements it is possible to form

a proof net, as illustrated below.
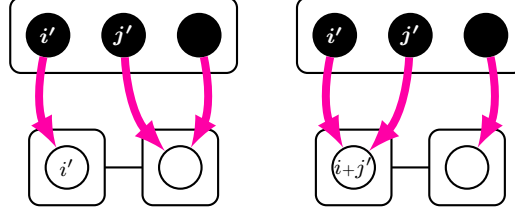
To ensure that a weight element may not form a proof net with a constraint element of another vertex, the values of m and n are chosen such that $m \equiv 1$ and $n \equiv 2 \pmod 3$, and accordingly $i \equiv 1$, $j \equiv 0$, and $k \equiv 2 \pmod 3$. It then becomes impossible for m to equal any other sum over i, j, and k, so that forming a proof net requires either $m = i$ or $m = i + j$.

Several further constructions are used in our encoding. Firstly, in a constraint graph, the sum of all weights is usually greater than the sum of all inflow constraints—otherwise, no edge can be mobile, or no configuration exists. Correspondingly, an encoding will have weight elements not connected to constraint elements. These will instead connect to additional, separate 1-formulae, referred to as *weight absorbers*, as shown below. A weight element that is connected to absorbers will be called *free*.

An edge in a constraint graph will be encoded by an *edge-gadget*, illustrated below left, constructed by stringing together a number of similar weight elements, plus a single ⊥-formula which we will refer to as the *indicator*. Illustrated below right is a *vertex-gadget* encoding a vertex, formed by a number of constraint elements plus a single *indicator target*.
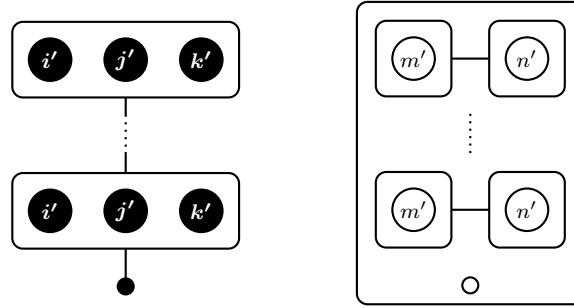
The natural way for edge-gadgets to link up with a vertex-gadget is shown in Figure 12: indicators connect to indicator targets, and weight elements connect to constraint elements or weight absorbers. The illustration depicts two equivalent proof nets, both encoding the same (fragment of a) constraint graph: a single vertex that is the target of a weight-1 edge (on the left, in red) and a weight-2 edge (on the right, in blue). The inflow of the vertex is 3, and its constraint is 2; then 2 weight elements are connected to the constraint elements in the vertex-gadget, while one remains free (connected to weight absorbers). It is essential to the functioning of the encoding that any re-distribution of weight elements gives an equivalent proof net, as illustrated by the example. Then in the second proof net, the single weight element of the left edge-gadget is free, which means the indicator jump can be rewired to connect to a different vertex.
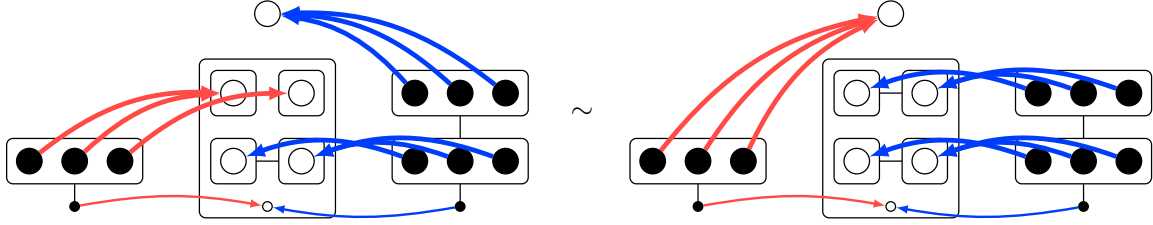
Figure 12: Two edge-gadgets connecting to one vertex-gadget

In the encoding, the indicator jumps determine which vertex is the target of the edge, and while they may in principle assign any vertex, the weight elements of the edge-gadget can only form proof nets with the constraint elements of the correct vertices. An edge-gadget will be called *free* if all its weight elements are free. The vertex-gadgets in an encoding will all be gathered in a $\otimes$-formula, which means they are connected, so that free edge-gadgets may attach their indicator to any of them.

In Figure 13 we give a complete encoding of the example rewiring sequence shown in Figure 9, a reconfiguration sequence for a constraint graph functioning as an AND-gate. The given proof nets are all equivalent, and their corresponding constraint graphs are shown near the bottom of the figure. The central vertex-gadget encodes the central constraint-2 vertex in the graph, while its three implicit constraint-0 vertices, at the dangling ends of the three edges, are encoded by the three formulae connected to the central gadget.

Weight adjustment. So far we have assumed that one weight element may encode one unit of weight in the constraint graph. However, there is a minor issue that prevents this straightforward approach. Although one weight element cannot 'fill' an inappropriate constraint element, two weight elements *can*, in the way illustrated below. In such a situation, the weight element that should have been linked to this constraint element becomes free, and the edge it belongs to may inappropriately become free, too.
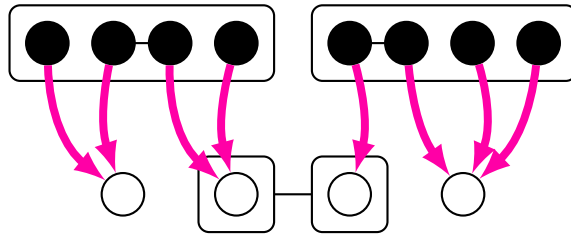
To resolve this issue, it is sufficient to increase the number of weight elements used to encode one unit of weight. In the following, we will investigate how many are needed.

In an inappropriate linking as shown above, since both halves of the constraint element are connected, the weight elements must be disconnected—otherwise, the switching condition would be violated. That means the weight elements must belong to different edges.

As the linkings above illustrate, it may occur that one subformula of a weight element $A \mathbin{⅋} B \mathbin{⅋} C$ fills one half of a constraint element. Consequently, a weight element can fill three halves, but only of different constraint elements. The other three halves may be filled by weight elements of a different edge—so to fill 3 constraint elements requires 2 inappropriate edges. To fill the next three constraint elements, at most 1 previous inappropriate edge may be used, and one additional one is needed. To fill $3n$ constraint elements inappropriately
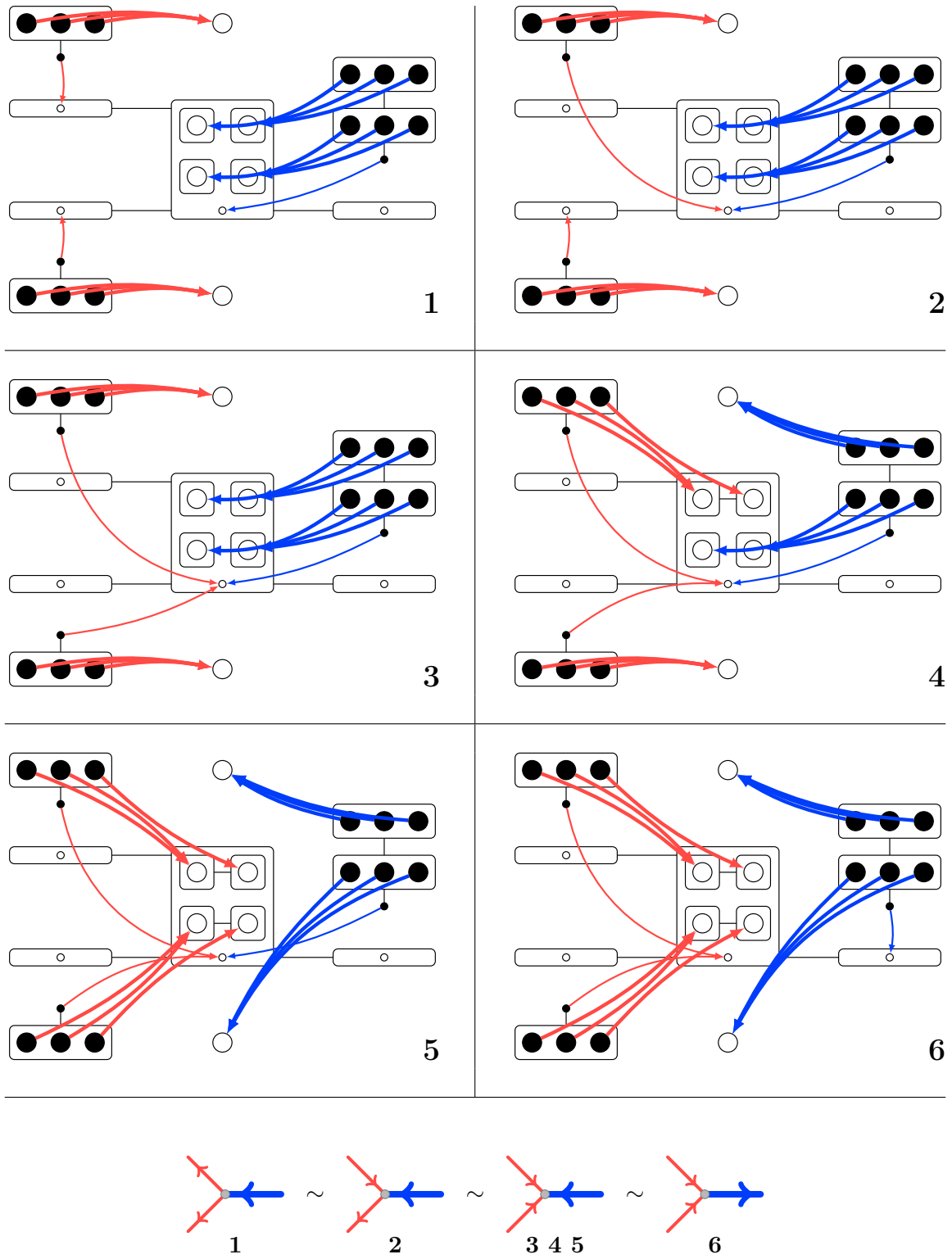
Figure 13: Encoding an AND-gate in proof nets

therefore requires $n + 1$ edges. It thus suffices to encode one unit of weight on an edge by $3 \times |E|$ weight elements (where $|E|$ is the number of edges in the constraint graph), and correspondingly to encode one unit of inflow constraint by $3 \times |E|$ constraint elements.

The complete encoding. The complete encoding of a constraint graph G will then be a sequent Γ consisting of:

(1) all vertex-gadgets, combined in a single formula via tensors,
(2) all edge-gadgets as individual formulae, and
(3) a sufficient number of weight absorbers (1-formulae).

A configuration for G will be encoded as a proof net for Γ, and conversely each proof net for Γ may be interpreted as a (partial) configuration for G. The encoding will be made formal in Section 7; first we will establish some basic results for rewiring on proof nets.

6. Rewiring proof nets

In this section we will explore the global rewiring behaviour of proof nets. We will look at notions of subnets; we will introduce a notion of relative *parity* between nets, which if odd, guarantees inequivalence; and we will give a simple account of equivalence for the fragment of MLL that omits the par.

The notions and results introduced in this section will be used in the main proofs of the paper, in Section 7, which show that the encoding of NCG-reconfiguration in MLL proof equivalence, as described informally in the previous section, is correct.

Subnets. We will discuss (and adapt) some convenient standard notions for MLL proofs and proof nets, and relate them to rewiring. Firstly we will look at subnets—see also [3].

Definition 14. A *sub-sequent* $\Delta \leq \Gamma$ of a sequent Γ is a sequent consisting of disjoint subformulae of Γ, preserving names.

Definition 15. A *subnet* $(\Delta, \ell') \leq (\Gamma, \ell)$ of a proof net is a net such that $\Delta \leq \Gamma$ and ℓ' is the restriction of ℓ to the names in Δ.

The *ports* of a sub-sequent Γ' or subnet (Γ', ℓ') are the root vertices of Γ'. For a vertex v naming a par, tensor, or bottom, the subnets of which it is a port correspond to the possible subproofs of the rule introducing v in a sequentialisation (the subproof of a 1-subformula must always be empty).

In the graph of a proof net, a chosen subnet for a par can be made explicit as a *box*, as illustrated below left. Boxes may replace the switching condition as a correctness criterion: in the example, both the outside and the inside of the box form a tree. To make this precise, we will consider the action of *closing* a box, which means it is regarded as a single vertex in the graph, as illustrated below right.
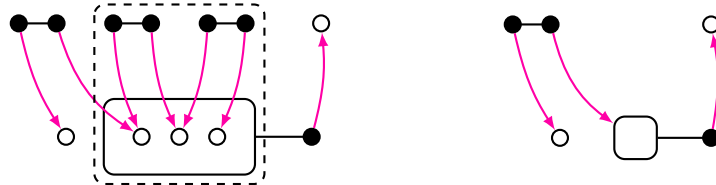
Definition 16. A *boxing s* for a linking ℓ for Γ assigns a sub-sequent $s(v) \leq \Gamma$ to each par-vertex $\mathfrak{N}_v$ such that 1) v is a port of $s(v)$ and 2) boxes are either disjoint or strictly nested: if $s(v) \cap s(w) \neq \varnothing$ then $s(w) < s(v)$ or $s(v) < s(w)$.

In the graph for ℓ and Γ, a box $s(v)$ may be *closed* by replacing the subgraph over $s(v)$ by the single vertex v, and replacing every arc into $s(v)$ by one onto v. For each box $s(v)$ we define the *local graph* to be that formed by the subgraph over $s(v)$ where each immediately smaller box $s(v) < s(v)$ is closed. The following is then a variation on the local retraction algorithm by Danos [6].

Proposition 17. *A linking ℓ for Γ is a proof net if and only if it has a boxing s such that each local graph is a tree.*

Proof. Given a boxing s, it follows by induction on the nesting of boxes that the graph over each $s(v)$ satisfies the switching condition. In the other direction, given a sequentialisation of (Γ, ℓ), a box $s(v) \leq \Gamma$ for each $\mathfrak{N}_v$ is found by taking the conclusion $\Delta, A \mathfrak{N}_v B$ of its introduction rule, below.

$$\frac{\Delta, A, B}{\Delta, A \mathfrak{N}_v B} \mathfrak{N}$$

$\square$

In a proof net, the *kingdom* and the *empire* of a vertex v are respectively the smallest and largest subnet that have v as a port. In working with the rewiring relation, the notion of empire can be particularly useful.

Definition 18. The *empire* $\ell|_v$ of a vertex v in (Γ, ℓ) is the smallest subnet (Δ, ℓ') of which v is a port.

Proposition 19 ([3, Proposition 2.b]). *The empire $\ell|_v$ is determined by propagation from v:*
  (1) *through links;*
  (2) *up towards subformulae;*
  (3) *into a tensor if one of its subformulae is in $\ell|_v - \{v\}$;*
  (4) *into a par if all its subformulae are in $\ell|_v - \{v\}$.*

It should be noted that the above characterisation of empires relies on the restriction that we have imposed, that jumps target 1-formulae only. Otherwise, the first case of the definition should be specialised so that propagation does not traverse jumps into v.

The following three lemmata will show how empires are connected to rewiring. Firstly, a jump from $\perp_v$ may be rewired to exactly those 1-occurences that are in the empire of v (Lemma 20). Secondly, rewiring a jump from $\perp_v$ preserves the empire of v, up to that rewiring (Lemma 21). Thirdly, rewiring the jump from v may add subformulae to the empire of another vertex w, or remove subformulae from it, but not both (Lemma 22).

Lemma 20. *For a proof net (Γ, ℓ) where $\ell(a) = v$, and w names a 1-occurrence in Γ, the following are equivalent:*
  (1) $\ell \overset{1}{\sim} \ell[a{\rightarrow}w]$
  (2) *w is in the empire $\ell|_a$; and*
  (3) *in any switching graph for (Γ, ℓ), the path v---w does not pass through a.*

Proof. By [3, Proposition 2.a] 2 and 3 are equivalent.

Next, it is shown that 2 implies 1. The empire $\ell|_a$ corresponds to the largest subproof Σ, with as conclusion the introduction rule of $\bot_a$, in any sequentialisation Π of ℓ. By Definition 3, in the translation of Σ to a net, a may link to any 1-occurrence, including to w.

Finally, it is shown that 1 implies 3, by contraposition. If for some switching of ℓ the path $v\text{---}w$ passes through a, then in $\ell[a{\to}w]$ there is no path $a\text{---}v$ (and two paths $a\text{---}w$) for that switching, so that $(\Gamma, \ell[a{\to}w])$ is not a net. $\qquad\square$

Lemma 21. *If $\ell \overset{1}{\sim} \ell[v{\to}w] = \ell'$ then $\ell|_v \overset{1}{\sim} \ell'|_v$.*

Proof. Since v may rewire to exactly the same 1-occurrences in ℓ as in ℓ', by Lemma 20 the empires $\ell|_v$ and $\ell'|_v$ contain the same 1-subformulae. That they also share any other subformula A follows by the observation that $A \otimes 1$ may replace A: by Proposition 19 the new 1 is in a given empire if and only if A is (unless v names A, but in this case A is included in both $\ell|_v$ and $\ell'|_v$). $\qquad\square$

Lemma 22. *If $\ell \overset{1}{\sim} \ell'$ where $\ell|_v$ is a net for the sequent Δ, and $\ell'|_v$ a net for Δ', then $\Delta \leq \Delta'$ or $\Delta \geq \Delta'$.*

Proof. Let $\ell' = \ell[a{\to}w]$, where $\ell(a) = u$ and $\ell'(a) = w$. We will distinguish five cases, depending on three factors: 1) whether $\ell|_v$ contains u; 2) if so, whether $\ell|_v$ is propagated from u to a or from a to u in case 1 of Proposition 19; and 3) whether $\ell|_v$ contains w.

- If $\ell|_v$ is propagated from a to u, then it includes $\ell|_a$ as a subnet, because the latter is generated by propagation from a. Since $\ell|_a = \ell'|_a$ by Lemma 21, then in ℓ' the empire of v is propagated from a to w, and includes $\ell'|_a$. It follows that $\Delta = \Delta'$.
- If $\ell|_v$ is propagated from u to a, and also contains w, then $\ell|_v$ has been propagated to w without passing through a; otherwise, a switching cycle would be formed by the two propagation paths from a to w, one generated by $\ell|_a$ via u, and one generated by $\ell|_v$, not via u. Then $\ell'|_v$ is propagated from v to w to a, and $\Delta = \Delta'$.
- If $\ell|_v$ is propagated from u to a, but does not contain w, then $\Delta \geq \Delta'$.
- If $\ell|_v$ contains w but not u, then $\Delta \leq \Delta'$.
- If $\ell|_v$ contains neither u nor w, then it also does not contain a. Then also $\ell'|_v$ contains neither u, w, nor a, and $\Delta = \Delta'$. $\qquad\square$

Parity. The linearity of MLL means that in a proof or proof net, there is always a certain balance to the number of $\bot$- and $\mathfrak{N}$-occurrences. This observation gives a well-known necessary condition for the provability of a sequent.

Definition 23. The *balance* of a sequent is the number of $\bot$s minus the number of $\mathfrak{N}$s and commas. A sequent is *balanced* if its balance is zero.

Proposition 24. *An unbalanced sequent is uninhabited.*

Here, we will introduce a similar necessary condition for the equivalence of two proof nets. We shall associate a *parity* with any pair of linkings ℓ and ℓ' over the same sequent Γ, and we shall find that the parity of equivalent linkings is always even.

For this argument we will work with n-ary connectives $\otimes$ and $\mathfrak{N}$, and alternating formulae, i.e. every argument of a $\otimes$ is a $\mathfrak{N}$ and vice versa. The units are given by the 0-ary connectives, and we need not rule out unary ones. We will consider a given named sequent Γ, but will assume that it consists of a single formula, if necessary by introducing a $\mathfrak{N}$ at the root.
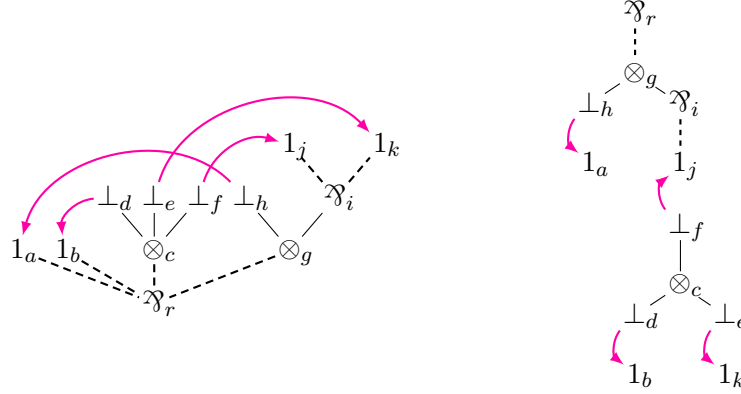
Figure 14: A switching graph of a proof net as a directed tree

To be able to compare arbitrary proof nets over Γ, we will use the following naming scheme for the edges of a switching graph, for any proof net over Γ. A tensor $\bigotimes(A_1, \ldots, A_n)$ named v has n edges, which we shall name $v(1)$ through $v(n)$; a par $\mathfrak{N}_v$ has one switched edge, to be named $v(1)$; and $\perp_v$ has the jump named $v(1)$. The naming scheme identifies edges across all switching graphs of all proof nets for Γ.

Taking a different perspective, we may alternatively consider a switching graph as a directed tree, rooted in the root connective of Γ. This establishes a bijection between the edges and the non-root vertices, which associates each edge with its target vertex.

The example in Figure 14 displays a proof net on the left, and on the right the switching graph choosing the edge $i-j$ for the par i, and the edge $r-g$ for the root par r. The induced bijection associates for example the edge named $r(1)$ with the vertex g, the edge $g(1)$ with h, and the edge $h(1)$ with a; it further associates $f(1)$ with f and $c(3)$ with c.

Given two proof nets ℓ and ℓ' for Γ, and a switching graph for each (not necessarily given by the same switching of Γ), we obtain two bijections between edges and (non-root) vertices. Composing these gives a permutation on the non-root vertices.

Definition 25. The *parity* of two switching graphs for proof nets ℓ and ℓ' for a sequent Γ is the parity of their induced permutation.

We will show that both 1) rewiring and 2) choosing a different switching induce even parity. By 2) we may define the parity of two proof nets ℓ and ℓ' to be that over arbitrary switching graphs; then by 1) it follows that proof nets with odd parity are inequivalent.

We will demonstrate 1), while 2) is similar. Let $\ell \overset{1}{\sim} \ell'$ by rewiring a jump $v \rightarrow a$ to $v \rightarrow b$. By fixing a switching for Γ, we obtain a switching graph for each of the two nets, where the jump is named $v(1)$ in each. There are two possibilities, illustrated below. On the left, if the jump $v(1)$ is directed upward, then the target of each edge in the directed switching graphs remains the same—in particular $v(1)$ has target v—and the induced permutation is the identity.
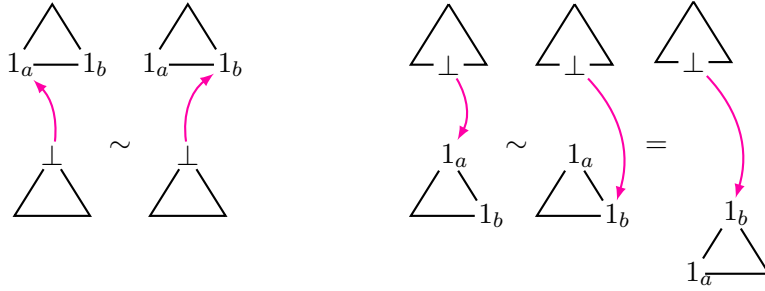
Above on the right, if the jump from v is directed downward, the subtree of a will get the new root node b. Then the vertices that are associated with a new edge are exactly those on the path $a \dashrightarrow b$ in the switching graph, as illustrated below.
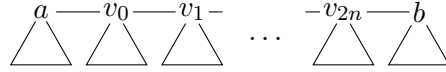
Since the connectives in Γ were assumed to be strictly alternating, there are an odd number of vertices on this path, $2n + 3$: each even v_i must be a $\bot$ or $\invamp$, while each odd v_j must be a 1 or $\otimes$. The permutation induced is then as follows. Since v has target a in the first switching graph, and target b in the second, it takes a to b. Further, since an edge connecting v_i and v_{i+1} has target v_{i+1} in the first, but target v_i in the second graph, the permutation takes v_{i+1} to v_i. The complete permutation is then a cyclic one taking each vertex on the path $a \dashrightarrow b$ to the previous, and the first to the last. A cyclic permutation of odd length has even parity.

The above argument gives us 1), that rewiring has even parity. To see that the same argument also gives 2), it is sufficient to consider that choosing a different switching for a single par is essentially the same operation as rewiring, if the par is considered a $\bot$ and the switched edge a jump. We may thus conclude that:

Proposition 26. *Two equivalent proof nets have even parity.*

Equivalence without $\invamp$. Let a *basic* sequent be one of formulae constructed only over 1, $\bot$, and $\otimes$. After removing dangling $\bot$-formulae and replacing subformulae $1 \otimes A$ with A, basic sequents consist of formulae of the form 1 or $\bigotimes(\bot^n)$ with $n \geq 2$. Provability for basic sequents is entirely determined by balance:

Proposition 27. *A balanced basic sequent is inhabited.*

We will show that, similarly, equivalence for basic sequents is determined by parity. An immediate observation is that a proof net for a basic sequent with only one tensor-formula, every 1 is linked to by exactly one jump—which means that no rewiring is possible.

Proposition 28. *A basic sequent $1^n, \bigotimes(\bot^n)$ is inhabited by exactly $n!$ inequivalent proof nets.*

In the following we will characterise equivalence for basic sequents with two or more tensor-formulae.

Lemma 29. *Let ℓ be a proof net with for every switching a path*

$$\ell(a) \leftarrow a \dashrightarrow b \to \ell(b) \dashrightarrow \ell(c) \leftarrow c \dashrightarrow d \to \ell(d) \ .$$

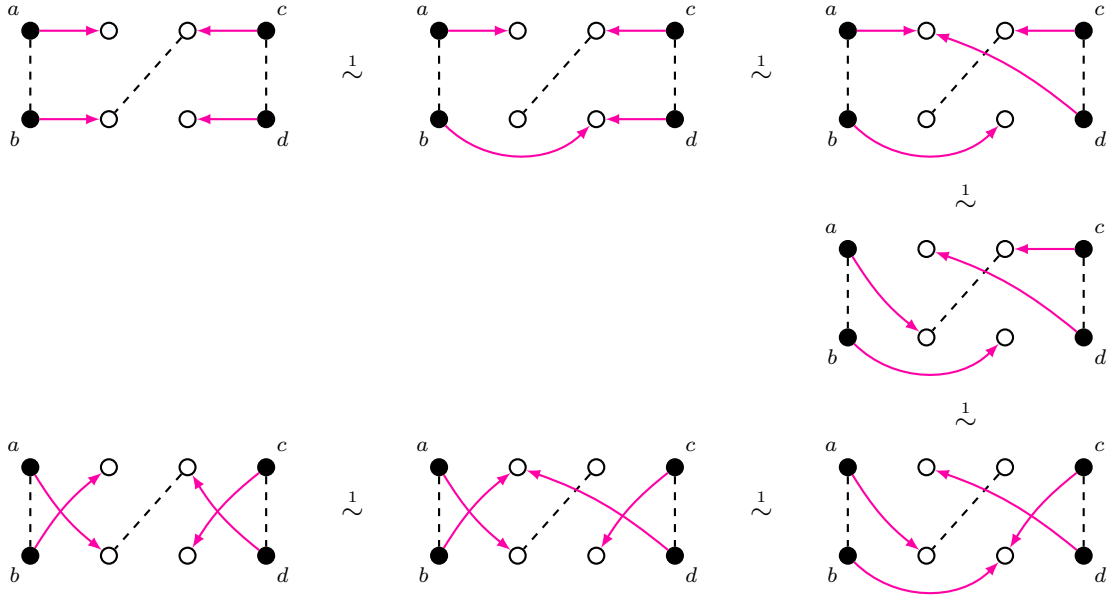*Then $\ell \sim \ell[a{\to}\ell(b)][b{\to}\ell(a)][c{\to}\ell(d)][d{\to}\ell(c)]$.*

Figure 15: Double exchange of jumps (Lemma 29)

Proof. By the rewiring path shown in Figure 15. □

Lemma 30. *A basic sequent with at least two tensor-formulae has at most two equivalence classes of proof nets.*

Proof. By induction on the size of a sequent Γ. The base case is the sequent $1, 1, \bot \otimes \bot$, which has two inequivalent proof nets. For the inductive step, let $\Gamma = \Gamma', A \otimes \bot_a, 1_z$ where $A \otimes \bot_a$ is a largest $\otimes$-formula in Γ. It will be shown that any net ℓ is equivalent to one ℓ' where $\ell'(a) = z$; then by induction, the subnet ℓ' restricted to Γ', A belongs to one of two equivalence classes. To find ℓ', there are two cases.

   1) The path $a\text{-}\text{-}\text{-}z$ is via $\ell(a)$. If $\ell(a) = z$, we are done. Otherwise, by Lemma 20 ℓ' may be obtained from ℓ by changing only $\ell'(a) = z$.

   2) The path $a\text{-}\text{-}\text{-}z$ is via some $\bot_b$ in A. Firstly, if $\ell(b) \neq z$, use Lemma 20 to re-attach b to z. Next, let $\bot_c$ and $\bot_d$ be occurrences in a separate formula B such that c links to the same 1-occurrence as some $\bot$ in A. Then ℓ' is obtained by linking c to b, and applying Lemma 29 to exchange the targets of a and b, as well as those of c and d. □

Proposition 31. *For a basic sequent with at least two tensor-formulae, two proof nets with even parity are equivalent.*

Proof. By Lemma 30 the sequent has at most 2 equivalence classes. Given two proof nets of even parity, both must be in the other equivalence class than a proof net with odd relative parity to both, which exists by exchanging two jumps from one tensor-formula. □

7. Formalising the encoding

We will formalise the encoding of NCG-reconfiguration into MLL proof equivalence that was informally introduced in Section 5. A constraint graph G will be encoded as a sequent $[\![G]\!]$, and a configuration γ for G will be encoded as a proof net $[\![\gamma]\!]$ for $[\![G]\!]$. We will show that

$\gamma \sim \delta$ if and only if $[\![\gamma]\!] \sim [\![\delta]\!]$ (modulo a small adjustment to ensure even parity between $[\![\gamma]\!]$ and $[\![\delta]\!]$).

For a constraint graph $G = (V, E, c, v, w)$, let $|V|$ and $|E|$ denote the number of vertices and edges, and let $|c|$ and $|w|$ denote the sum of all inflow constraints and the sum of all weights, respectively:

$$|c| = \sum_{v \in V} c(v) \qquad\qquad |w| = \sum_{e \in E} w(e) \ .$$

Definition 32. The *encoding* $[\![G]\!]$ of a constraint graph G is a sequent constructed as follows. Let $G = (V, E, c, v, w)$ with $|V| = n$, $|E| = m$, $V = \{v_1, \ldots, v_n\}$, and $E = \{e_1, \ldots, e_m\}$. The encoding of a vertex v_k is the formula

$$[\![v_k]\!] = \mathfrak{P} \left(C_n(k)^{3m \times c(v_k)} \right) \mathfrak{P} 1$$

where each *constraint element* $C_n(k)$ is the formula

$$C_n(k) = \mathfrak{P} \left(1^{3k+2} \right) \otimes \mathfrak{P} \left(1^{3(n-k)+3} \right) \ .$$

The encoding of an edge e connecting vertices v_i and v_j with $i \le j$ is the formula

$$[\![e]\!] = \bigotimes \left(W_n(i, j)^{3m \times w(e)} \right) \otimes \bot$$

where each *weight element* $W_n(i, j)$ is the formula

$$W_n(i, j) = \bigotimes \left(\bot^{3i+2} \right) \mathfrak{P} \bigotimes \left(\bot^{3(j-i)+1} \right) \mathfrak{P} \bigotimes \left(\bot^{3(n-j)+3} \right) .$$

The encoding of the graph G is the sequent

$$[\![G]\!] = [\![v_1]\!] \otimes \ldots \otimes [\![v_n]\!], [\![e_1]\!], \ldots, [\![e_m]\!], 1^p$$

where $p = 3m \times (|w| - |c|) \times (3n + 4)$.

In the above definition, the final 1-subformula of a vertex-gadget $[\![v_k]\!]$ is its *indicator target*; the final $\bot$-subformula of an edge-gadget $[\![e]\!]$ is its *indicator*; and in the completed encoding $[\![G]\!]$ the p instances of 1 are the *weight absorbers*. In a constraint graph G, a vertex v and an edge e will be called *appropriate* (for each other) if $v \in v(e)$, and *inappropriate* otherwise. This notion is extended to vertex-gadgets $[\![v]\!]$ and edge-gadgets $[\![e]\!]$ in $[\![G]\!]$, and their respective constraint elements and weight elements.
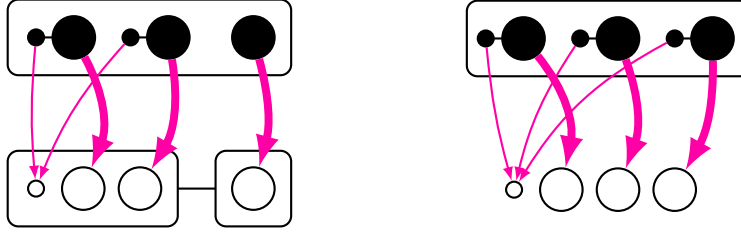
A configuration γ for a constraint graph G will be encoded as a proof net for the sequent $[\![G]\!]$. Firstly we will define a standard way of linking a weight element to a constraint element.

Definition 33. For $W = W_n(i, j) = A \mathfrak{P} B \mathfrak{P} C$ a weight element,
  (1) for a constraint element $C = C_n(i) = X \otimes Y$, the *standard linking* for the sequent W, C links the first $\bot$ in A to the first 1 in X, the first $\bot$ in B and C each to the first 1 in Y, and each remaining $\bot$ in A, B, C to a remaining 1 in X, Y in their order of occurrence;
  (2) for $C = C_n(j) = X \otimes Y$, the *standard linking* for W, C is defined as above, except the first $\bot$ in B links to the first 1 in X;
  (3) the *standard linking* for the sequent $W, 1^{3n+4}$ links the first $\bot$ in A, B, and C to the first 1, and each remaining $\bot$ to a remaining 1 in order of occurrence.

The standard linkings defined in the second and third case of the above definition are illustrated below.

Proposition 34. *Standard linkings are proof nets.*

The encoding of a configuration is then as follows.

Definition 35. The *encoding* $[\![\gamma]\!]$ of a total configuration γ for a constraint graph G is a linking ℓ for $[\![G]\!]$, constructed incrementally for each successive edge e and for each successive weight element W within e, as follows. Let $\gamma(e) = v$; firstly, the indicator of $[\![e]\!]$ is linked to the indicator target of $[\![v]\!]$. Then successively for each weight element W in e, if $[\![v]\!]$ has a first free constraint element C, extend $[\![\gamma]\!]$ to include the standard linking on W, C; otherwise, extend $[\![\gamma]\!]$ by the standard linking on the sequent consisting of W plus the first $3n + 4$ free weight absorbers.

Proposition 36. *If γ is a total configuration for G then $[\![\gamma]\!]$ is a proof net for $[\![G]\!]$.*

Proof. Using Proposition 17, it is sufficient to give a suitable box for each $\mathfrak{V}$. The box of each weight element W is the sequent W, C or $W, 1^{3n+4}$ of its standard linking, which forms a proof net by Proposition 34. The box of each vertex-gadget $[\![v]\!]$ contains the edge-gadgets $[\![e]\!]$ such that $\gamma(e) = v$, plus all the weight absorbers within boxes of weight elements inside $[\![e]\!]$. Since the weights of the connected edges e sum to more than the inflow constraint of v, there are no unused constraint elements remaining in $[\![v]\!]$. After closing the box of each W, each edge-gadget in the box of $[\![v]\!]$ becomes a single string of connected vertices, connected to other edge-gadgets only via the indicator target of $[\![v]\!]$, thus forming a tree. □

Finally, if two connected configurations $\gamma \sim \delta$ are encoded individually as $[\![\gamma]\!]$ and $[\![\delta]\!]$, these may not be equivalent simply because their parity happens to be odd. For that reason we shall adjust the encoding of the second to guarantee an even parity.

Definition 37. Let $[\![\delta]\!]'$ be $[\![\delta]\!]$ with the first two weight absorbers exchanged. For configurations γ and δ for G, let $[\![\delta]\!]_\gamma$ be $[\![\delta]\!]$ if it has even parity with $[\![\gamma]\!]$, and $[\![\delta]\!]'$ otherwise.

In the remainder, we will show that our encoding is correct, i.e. that $\gamma \sim \delta$ if and only if $[\![\gamma]\!] \sim [\![\delta]\!]_\gamma$. This will be separated into two parts: completeness ($\Rightarrow$) and soundness ($\Leftarrow$) .

Completeness. Given a reconfiguration path $\gamma \sim \delta$ over total configurations, we will demonstrate a rewiring sequence between $[\![\gamma]\!]$ and $[\![\delta]\!]_\gamma$. The central part of the argument will be to show how the weight element linking to a constraint element may be exchanged for another (Lemma 38). Before and after the exchange, the constraint element and the weight element connecting to it will be in a standard linking. The linking between weight elements and weight absorbers need not be standard: it will be shown that weight absorbers may be freely rearranged, as long as parity remains even (Lemma 39).

For a reconfiguration step $\gamma \overset{1}{\sim} \delta$ where the edge e changes direction from v to w, the rewiring sequence $[\![\gamma]\!] \sim [\![\delta]\!]_\gamma$ will be as follows. First, the weight elements of edge-gadgets

connecting to $[\![v]\!]$ are rearranged to match their target configuration, in $[\![\delta]\!]_\gamma$, which means the weight elements of $[\![e]\!]$ connect only to weight absorbers. Then $[\![e]\!]$ is moved from $[\![v]\!]$ to $[\![w]\!]$ by rewiring its indicator link, from the indicator of $[\![v]\!]$ to that of $[\![w]\!]$. Next, the weight elements connecting to $[\![w]\!]$ are rewired to match $[\![\delta]\!]_\gamma$, and finally, weight absorbers are rearranged to match $[\![\delta]\!]_\gamma$ as well.

To describe the intermediate stages of such a rewiring sequence, call an edge-gadget $[\![e]\!]$ *well linked* if: 1) its indicator connects to the indicator target of an appropriate vertex-gadget $[\![v]\!]$, and 2) each weight element is either in a standard linking with a constraint element of $[\![v]\!]$, or is free (linked only to weight absorbers, in arbitrary fashion). In the following main lemma we will exchange the weight element linking to a constraint element. The lemma considers a sequent consisting of just a vertex-gadget, some appropriate edge-gadgets, and sufficiently many weight absorbers.

Lemma 38. *In a proof net ℓ for $\Gamma = [\![v]\!], [\![e_1]\!], \ldots, [\![e_m]\!], 1^p$ where each edge-gadget is well linked, if a weight element W_i in $[\![e_i]\!]$ is linked to V in $[\![v]\!]$ and W_j in $[\![e_j]\!]$ is linked to weight absorbers 1^n, then there is a net $\ell' \sim \ell$ in which W_j is well linked to V, W_i is linked to 1^n, and ℓ' agrees with ℓ otherwise.*

Proof. Let $W_i = A \,\bindnasrepma\, B \,\bindnasrepma\, C$, $W_j = D \,\bindnasrepma\, E \,\bindnasrepma\, F$, and $V = X \otimes Y$. The rewiring path will be illustrated for the case where A, B, X and D, X, and thus also C, Y and E, F, Y, are balanced sequents; other cases are similar.

1. The initial configuration is illustrated below. Other edges, other weight and constraint elements, and the outer box of the vertex-gadget, are omitted. The vertices i, j, and v are the indicators of $[\![e_i]\!]$ and $[\![e_j]\!]$ and the indicator target of $[\![v]\!]$, respectively.
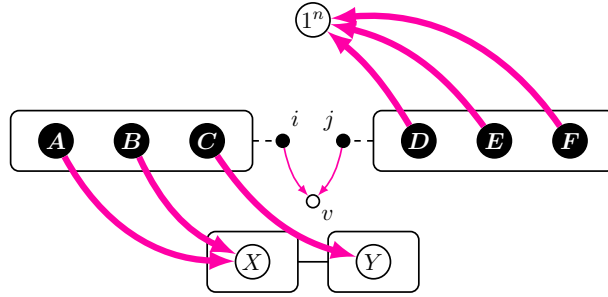

2. The jump $i \rightarrow v$ is rewired to a weight absorber together with only the jumps from the first $\perp$ of each of D, E, and F:
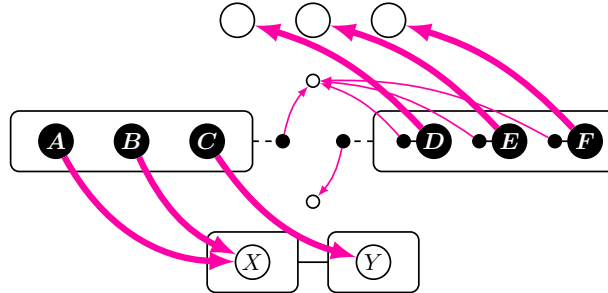

3. The jump from the first $\perp$ of D is moved to X, and those from the first $\perp$ of E and F are moved to Y:
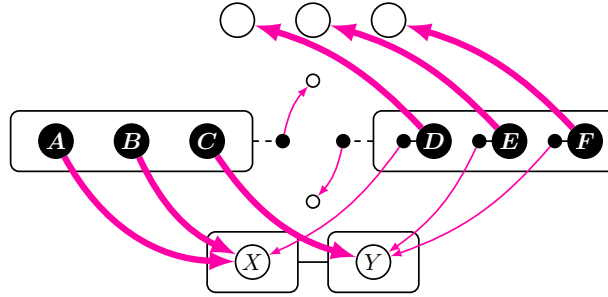
4. In the present configuration, the jumps from the weight elements form two subnets: one over the sub-sequent $A, B, X, D, 1^m$, and one over the sub-sequent $C, Y, E, F, 1^k$, for some m and k. By Proposition 31, these subnets are equivalent to any other over the same sequent, as long as their parity is preserved. It is then sufficient to choose linkings so that $D \bindnasrepma E \bindnasrepma F$ is in a a standard linking with $X \otimes Y$, with a minor adjustment: two jumps from D to X should remain exchanged, compared to the standard linking, for step 6 below. The jumps of A, B, C link to the weight absorbers 1^m and 1^k, with one remaining jump from B to X and one from C to Y.
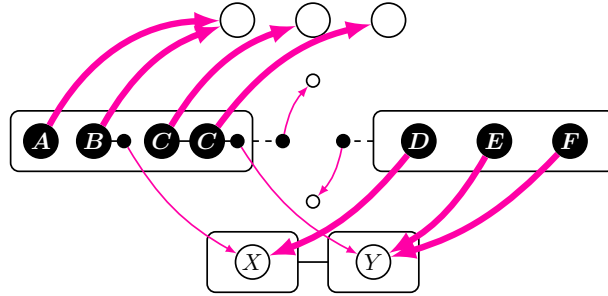

5. The jump from C to Y is moved towards a weight absorber connected to A, B.
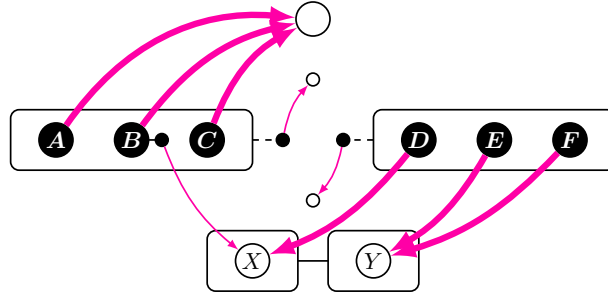
6. The jump from B to X is the one remaining connection between the edge-gadgets $[\![e_i]\!]$ and $[\![e_j]\!]$. Lemma 29 allows to swap the targets of the jump from B to X and the jump from i, and simultaneously undo the exchange in the jumps from D to X added in step 4 above.
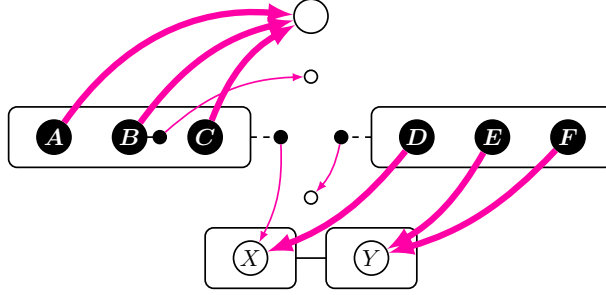
7. The jump from i is re-attached to v to yield the final configuration:

$\square$
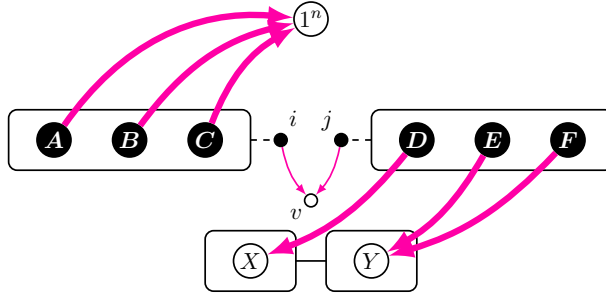
In the exchange of weight elements in the above lemma, weight elements link to weight absorbers in arbitrary fashion. To be able to correct for this, the next lemma will demonstrate that, modulo parity, weight absorbers can be re-arranged at will.

Lemma 39. *If ℓ and ℓ' are well-linked proof nets for $[\![G]\!]$ that are equal up to an even permutation of weight absorbers, then $\ell \sim \ell'$ if ℓ and ℓ' have at least one free edge-gadget $[\![e]\!]$.*

Proof. To reconfigure ℓ' into ℓ, we will use the double exchange operation of Lemma 29 to exchange two arbitrary weight absorbers at a time, as well as two chosen jumps in the free edge-gadget $[\![e]\!]$. Firstly, let e_0 be the indicator of $[\![e]\!]$. Note that since $[\![e]\!]$ is free, e_0 may re-attach anywhere within the proof net.

We will need two basic operations: 1) to exchange two arbitrary weight absorbers v and w linked from outside $[\![e]\!]$; and 2) to exchange an arbitrary weight absorber linked from inside $[\![e]\!]$ with one linked from outside $[\![e]\!]$. Since we are using double exchanges, each operation will also exchange two arbitrary jumps from $[\![e]\!]$ (but not that from e_0). By using operation 1) twice, we obtain a third, where four arbitrary weight absorbers linked from inside $[\![e]\!]$ are pairwise exchanged. Since ℓ' and ℓ differ by an even permutation of weight absorbers, these three operations together give $\ell' \sim \ell$.

To perform operation 1), exchanging v and w as well as the jumps from e_1 and e_2 in $[\![e]\!]$, first connect e_0 to v. Then apply Lemma 29 three times: exchanging v and w, and the targets of e_0 and e_1; exchanging w and v, and the targets of e_1 and e_2; and exchanging v and w, and the targets of e_2 and e_0. The result is a net exchange of v and w, and the targets of e_1 and e_2.

Operation 2) is performed similarly. In both cases, if one of the weight absorbers exchanged is linked to by multiple $\perp$-occurrences within the same weight element, these may be temporarily attached elsewhere. $\square$

Lemma 40. *If $\gamma \sim \delta$ for total configurations γ and δ, then $[\![\gamma]\!] \sim [\![\delta]\!]_\gamma$.*

Proof. By Proposition 13 we may assume that $\gamma \sim \delta$ by a sequence of reconfiguration steps over total configurations. We will prove that if $\gamma \overset{1}{\sim} \delta$ then $[\![\gamma]\!] \sim [\![\delta]\!]$ or $[\![\gamma]\!] \sim [\![\delta]\!]'$. The same proof will show the corresponding case for $[\![\gamma]\!]'$ instead of $[\![\gamma]\!]$, so that the general case for $\gamma \sim \delta$ follows by transitivity.

Let γ and δ agree on every edge except e, where $\gamma(e) = v$ and $\delta(e) = w$. Firstly, using Lemma 38, for the edges d other than e such that $\gamma(d) = v$, the weight elements of the edge-gadgets $[\![d]\!]$ may be linked to the constraint elements of $[\![v]\!]$, in accordance with the target configuration $[\![\delta]\!]$. Since e is mobile in γ, the weights of the edges d suffice to fill the inflow constraint of v, and correspondingly the weight elements of edge-gadgets $[\![d]\!]$ suffice to fill the constraint elements of $[\![v]\!]$, so that $[\![e]\!]$ is mobile. Next, the indicator vertex of $[\![e]\!]$, which links to the indicator target of $[\![v]\!]$, is re-attached to the indicator target of $[\![w]\!]$. Again using Lemma 38, the weight elements of edge-gadgets connected to $[\![w]\!]$, including $[\![e]\!]$, may be linked in accordance with $[\![\delta]\!]$. The resulting proof net is $[\![\delta]\!]$ modulo a permutation of weight absorbers; then it is equivalent to either $[\![\delta]\!]$ or $[\![\delta]\!]'$ by Lemma 39. $\qquad\square$

Soundness. It will be shown that proof net rewiring $[\![\gamma]\!] \sim [\![\delta]\!]_\gamma$ is sound for NCG-reconfiguration, $\gamma \sim \delta$. To each proof net ℓ for an encoded constraint graph $[\![G]\!]$ we will associate a partial configuration $\gamma = (\!|\ell|\!)$, such that 1) a rewiring step between ℓ and ℓ' corresponds to a reconfiguration path $(\!|\ell|\!) \sim (\!|\ell'|\!)$, and 2) the function $(\!|-|\!)$ is a retraction of the encoding of configurations, $(\!|[\![\gamma]\!]|\!) = \gamma$.

The configuration $(\!|\ell|\!)$ will assign an edge e to a vertex v when, in the proof net $(\ell, [\![G]\!])$, the edge-gadget $[\![e]\!]$ is in the empire of the vertex-gadget $[\![v]\!]$. Firstly, it will be shown that $(\!|\ell|\!)$ is a partial function.

Proposition 41 ([3, Proposition 1.i])**.** *In a proof net, if vertices v and w are joined by a tensor, then any two subnets of which they are respective ports are disjoint.*

Lemma 42. *In a proof net for $[\![G]\!]$, an edge-gadget $[\![e]\!]$ belongs to the empire of at most one vertex-gadget $[\![v]\!]$.*

Proof. Since vertex-gadgets are joined by a tensor, the lemma is immediate from Proposition 41. $\qquad\square$

Next, it will be shown that the appropriate edge-gadgets in the empire of a vertex-gadget $[\![v]\!]$ contain sufficient weight elements to fill the constraint elements of $[\![v]\!]$.

Lemma 43. *In a proof net for $[\![G]\!]$, for each node v in G, the weights of the appropriate edge-gadgets in the empire of $[\![v]\!]$ are equal to or greater than the constraint of v.*

Proof. Let $|E| = m$. We will show that the inappropriate edge-gadgets in G are insufficient to fill $3m$ weight elements in $[\![v]\!]$. Since weight elements come in multiples of $3m$, it follows that to fill the $3m \times c(v)$ constraint elements of v, there must be at least as many appropriate weight elements available.

Let e be inappropriate for v; let $C = A \otimes B$ be a constraint element in $[\![v]\!]$, and $W = X \,\mathgreek{⅋}\, Y \,\mathgreek{⅋}\, Z$ weight element in $[\![e]\!]$. To find a proof net for the sequent W, C requires to assign balanced boxes to the par-formulae A and B. Since e is inappropriate, the sequents A, X and A, X, Y are not balanced, while the balance of each of the other sequents of A

with one or more of X, Y, Z is always 1 or 2 (mod 3). It follows that there is no proof net for W, C.

Next, it will be shown that to balance $3m$ constraint elements requires at least $m + 1$ inappropriate edge-gadgets. Two edge-gadgets may balance at most three constraint elements: one weight element W has three subformulae, which may each balance at most one half of a constraint element; the other halves may be balanced by different weight elements of the second edge-gadget. In the same way, adding one further edge-gadget allows at most three further constraint elements to be filled, since previous edges may connect to only one half of each constraint element. Then to balance $3m$ constraint elements inappropriately requires $m + 1$ edge-gadgets. □

Using the above, a proof net for $[\![G]\!]$ may be interpreted as a configuration for G.

Definition 44. For a proof net ℓ for $[\![G]\!]$, let $(\!|\ell|\!)$ be the partial configuration for G where $(\!|\ell|\!)(e)$ is v if both e is appropriate for v and $[\![e]\!]$ belongs to the empire of $[\![v]\!]$, and undefined otherwise.

The following lemma then shows that rewiring corresponds to reconfiguration under $(\!|-|\!)$.

Lemma 45. *If $\ell \sim \ell'$ are proof nets for $[\![G]\!]$ then $(\!|\ell|\!) \sim (\!|\ell'|\!)$.*

Proof. The proof will consider the case where $\ell \overset{1}{\sim} \ell'$; the general case follows by transitivity. By Lemma 22, the empire of each vertex-gadget $[\![v]\!]$ contains either a subset of, a superset of, or exactly the same edge-gadgets in ℓ' as it does in ℓ. Let $[\![e_1]\!]$ through $[\![e_n]\!]$ be the edge-gadgets moving into or out of the empire of $[\![v]\!]$. By Lemma 43 other edge-gadgets must fill the constraint elements of $[\![v]\!]$. Then the corresponding edges e_1 through e_n are mobile in $(\!|\ell|\!)$. It follows that $(\!|\ell|\!) \sim (\!|\ell'|\!)$ by moving each e_i in turn, and repeating the process for other vertices. □

8. MLL proof equivalence is PSPACE-complete

We are now ready to state our main theorem.

Theorem 46. *MLL proof equivalence is PSPACE-complete.*

Proof. MLL proof equivalence has at most non-deterministic polynomial space complexity: a proof net may be represented in linear space (with respect to a proof); a single rewiring step is performed without requiring additional space; and a non-deterministic algorithm may guess the correct rewiring sequence. Then by Savitch's Theorem [28] MLL proof equivalence is in PSPACE.

PSPACE-hardness is by the encoding of NCG-reconfiguration, which in its completed form is stated:

$$\gamma \sim \delta \qquad \Longleftrightarrow \qquad [\![\gamma]\!] \sim [\![\delta]\!]_\gamma \ .$$

The direction ($\Rightarrow$) is by Lemma 40; the direction ($\Leftarrow$) is by Lemma 45 and the observation that $(\!|[\![\gamma]\!]|\!) = \gamma$ and $(\!|[\![\delta]\!]_\gamma|\!) = \delta$. □

References

[1] Jean-Marc Andreoli and Roberto Maieli. Focusing and Proof-Nets in Linear and Non-commutative Logic. In *LPAR*, volume 1705 of *LNCS*, pages 320–336, 1999.

[2] Michael Barr. *-Autonomous categories and linear logic. *MSCS*, 1:159–178, 1991.

[3] Gianluigi Bellin and Jacques van de Wiele. Subnets of Proof-nets in MLL$^-$. In *Advances in Linear Logic*, pages 249–270, 1995.

[4] Richard Blute, Robin Cockett, Robert Seely, and Todd Trimble. Natural deduction and coherence for weakly distributive categories. *JPAA*, 113:229–296, 1996.

[5] Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin. Canonical Sequent Proofs via Multi-Focusing. In *Fifth Ifip International Conference On Theoretical Computer Science*, pages 383–396, 2008.

[6] Vincent Danos. *La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du Lambda-calcul)*. PhD thesis, Université Paris 7, 1990.

[7] Vincent Danos and Laurent Regnier. The Structure of Multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[8] Erik D Demaine and Robert A Hearn. Constraint logic: A uniform framework for modeling computation as games. In *Proc. Conference on Computational Complexity (CCC'08)*, pages 149–162, 2008.

[9] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[10] Jean-Yves Girard. Linear logic. *TCS*, 50(1):1–102, 1987.

[11] Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1–2):72–96, 2005.

[12] Robert A Hearn and Erik D Demaine. *Games, puzzles, and computation*. AK Peters, Ltd., 2009.

[13] Willem Heijltjes. Proof nets for additive linear logic with units. In *LiCS'11*, pages 207–216, 2011.

[14] Willem Heijltjes and Lutz Straßburger. Proof nets and semi-star-autonomous categories. *MSCS*, FirstView, 10 November 2014.

[15] Robin Houston. *Modelling linear logic without units*. PhD thesis, University of Manchester, 2008.

[16] Dominic Hughes and Rob van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. *ACM Trans. Comp. Log.*, 6(4), 2005.

[17] Dominic J.D. Hughes. Simple multiplicative proof nets with units. *arXiv*, 2005. arXiv:math/0507003.

[18] Dominic J.D. Hughes. Simple free star-autonomous categories and full coherence. *JPAA*, 216(11):2386–2410, 2012.

[19] Takehiro Ito, Erik D. Demaine, Nicholas J.A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011.

[20] Max I. Kanovich. Horn Programming in Linear Logic is NP-complete. In *LiCS'92*, pages 200–210, 1992.

[21] François Lamarche and Lutz Straßburger. From Proof Nets to the Free *-Autonomous Category. *LMCS*, 2(4:3):1–44, 2006.

[22] Joachim Lambek. Deductive Systems and Categories I. Syntactic Calculus and Residuated Categories. *Mathematical Systems Theory*, 2(4):287–318, 1968.

[23] Olivier Laurent. Polarized Proof-Nets: Proof-Nets for LC. In *TLCA'99*, volume 1581 of *LNCS*, pages 213–227, 1999.

[24] Patrick Lincoln, John Mitchell, Andre Scedrov, and Natarajan Shankar. Decision Problems for Propositional Linear Logic. *Annals of Pure and Applied Logic*, 56:239–311, 1992.

[25] Patrick Lincoln and Timothy Winkler. Constant-only multiplicative linear logic is NP-complete. *Theoretical Computer Science*, 135:155–169, 1994.

[26] Paul-André Melliès. Game semantics in string diagrams. In *LiCS'12*, pages 481–490, 2012.

[27] A. S. Murawski and C.-H. L. Ong. Exhausting strategies, joker games and full completeness for imll with unit. In *In Proc. 8th Conf. CTCS'99*, volume 29 of *ENTCS*, pages 209–239, 1999.

[28] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 42:177–192, 1970.

[29] Lutz Straßburger and François Lamarche. On Proof Nets for Multiplicative Linear Logic with Units. In *In Proc. 13th Annual Conference of the EACSL (CSL 2004)*, volume 3210 of *LNCS*, pages 145–159, 2004.

[30] Todd Trimble. *Linear logic, bimodules, and full coherence for autonomous categories*. PhD thesis, Rutgers University, 1994.