No proof nets for MLL with units

Proof equivalence in MLL is PSPACE-complete

Willem Heijltjes
University of Bath
w.b.heijltjes@bath.ac.uk

Robin Houston
Kiln Enterprises
robin@kiln.it

Categories and Subject Descriptors F.4.1 [*Mathematical logic and formal languages*]: Mathematical logic—Proof theory; F.2.2 [*Analysis of algorithms and problem complexity*]: Nonnumerical algorithms and problems—Complexity of proof procedures

Keywords linear logic, proof equivalence, proof nets, constraint logic, PSPACE-completeness

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

The question of equivalence of proofs goes back to Lambek [19], who realised that the new tool of category-theoretic logic gave a notion of proof equivalence that was coarser and better-behaved than syntactic equality.

Later, a striking technical innovation of linear logic was the introduction of proof nets [5, 7], which define a canonical form for proofs in the unitless fragment of multiplicative linear logic—two proofs are equivalent if and only if they have the same proof net—so proof nets offer a simple decision procedure for proofs in this fragment. This naturally raises the question whether proof nets can be extended to work in the presence of units. The work in this direction begins with [3, 22] via [18] and perhaps culminates in [14, 15]; but these proof nets are not canonical and must be identified up to a rewiring equivalence.

So the question remains whether there exist fully canonical proof nets for full MLL. Canonical proof nets have been found for several other fragments of linear logic: the combined multiplicative-additive fragment without units [13], and the additive fragment, including the additive units [10].

In this paper we establish that the proof equivalence problem for multiplicative linear logic with units is PSPACE-complete. This effectively rules out the existence of a satisfactory notion of proof net for MLL with units—one that reduces proof equivalence to syntactic equality, and where the translation from proofs to proof nets and equality of proof nets are both tractable.

Constraint logic and reconfiguration problems

The proof of PSPACE-completeness relies on a polynomial reduction from the configuration-to-configuration problem in nondeterministic *constraint logic*, a graphical formalism recently introduced as a uniform tool for use in complexity reductions [6]. Constraint logic is a simple graph rewriting formalism, where weighted edges may be reversed as long as the given in-flow constraint for each vertex is satisfied; the configuration-to-configuration problem asks whether two graphs are related by a sequence of rewriting steps.

This is one of a class of problems called *reconfiguration problems* [16]: can one solution to a given problem be transformed into another by a series of elementary changes, while remaining valid throughout? For example, the reconfiguration problem for boolean satisfiability (SAT) asks whether one satisfying assignment can be transformed into another by changing the value of one atomic formula at a time, without passing via a non-satisfying assignment. It is not uncommon for an NP-complete problem to have an associated reconfiguration problem that is PSPACE-complete [16]; 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 [17, 20].

2. MLL

The formulae of unit-only multiplicative linear logic are given by the following grammar.

$$A, B, C \;:=\; \bot \;\mid\; 1 \;\mid\; A \parr B \;\mid\; A \otimes B$$

The connectives $\otimes$ and $\parr$ will be considered up to associativity, and *duality* $A^\star$ is via DeMorgan. A *sequent* Γ, Δ will be a multiset of formulae. Within a sequent, connectives and units will be *named* with distinct elements from an arbitrary set of names, e.g.

$$1_a \parr_b 1_c, \bot_d \otimes_e \bot_f \ .$$

This allows to 1) identify *occurrences* of subformulae uniquely by the name of their root connective, e.g. as A_b, 2) distinguish the two proofs of the above sequent while using standard multiset sequents, and 3) easily extract proof nets, as graphs using the names of connectives as vertices. Names will often be left implicit.

Proofs are constructed from the inference rules in Figure 1, where the names of units and connectives are preserved through

CSL-LICS 2014, July 14–18, 2014, Vienna, Austria.

ACM 978-1-4503-2886-9…$15.00.
http://dx.doi.org/10.1145/2603088.2603126

$$\frac{\Gamma}{\Gamma, \bot}\,\bot \qquad \frac{}{1}\,1 \qquad \frac{\Gamma, A, B}{\Gamma, A \parr B}\,\parr \qquad \frac{\Gamma, A \qquad \Delta, B}{\Gamma, \Delta, A \otimes B}\,\otimes$$

Figure 1. Inference rules for unit-only MLL

$$\frac{\dfrac{\Gamma}{\Gamma, \bot_a}\,\bot}{\Gamma, \bot_a, \bot_b}\,\bot \sim \frac{\dfrac{\Gamma}{\Gamma, \bot_b}\,\bot}{\Gamma, \bot_a, \bot_b}\,\bot \qquad \frac{\dfrac{\Gamma, A, B}{\Gamma, A \parr B}\,\parr}{\Gamma, A \parr B, \bot}\,\bot \sim \frac{\dfrac{\Gamma, A, B}{\Gamma, A, B, \bot}\,\bot}{\Gamma, A \parr B, \bot}\,\parr$$

$$\frac{\dfrac{\Gamma, A}{\Gamma, A, \bot}\,\bot \qquad \Delta, B}{\Gamma, \Delta, A \otimes B, \bot}\,\otimes \sim \frac{\Gamma, A \qquad \dfrac{\Delta, B}{\Delta, B, \bot}\,\bot}{\Gamma, \Delta, A \otimes B, \bot}\,\otimes \sim \frac{\Gamma, A \qquad \dfrac{\Delta, B}{\Delta, B, \bot}\,\bot}{\Gamma, \Delta, A \otimes B, \bot}\,\otimes$$

$$\frac{\dfrac{\Gamma, A, B, C, D}{\Gamma, A \parr B, C, D}\,\parr}{\Gamma, A \parr B, C \parr D}\,\parr \sim \frac{\dfrac{\Gamma, A, B, C, D}{\Gamma, A, B, C \parr D}\,\parr}{\Gamma, A \parr B, C \parr D}\,\parr$$

$$\frac{\dfrac{\Gamma, A \qquad \Delta, B, C, D}{\Gamma, \Delta, A \otimes B, C, D}\,\otimes}{\Gamma, \Delta, A \otimes B, C \parr D}\,\parr \sim \frac{\Gamma, A \qquad \dfrac{\Delta, B, C, D}{\Delta, B, C \parr D}\,\parr}{\Gamma, \Delta, A \otimes B, C \parr D}\,\otimes$$

$$\frac{\Gamma, A \qquad \dfrac{\Delta, B, C \qquad \Lambda, D}{\Delta, \Lambda, B, C \otimes D}\,\otimes}{\Gamma, \Delta, \Lambda, A \otimes B, C \otimes D}\,\otimes \sim \frac{\dfrac{\Gamma, A \qquad \Delta, B, C}{\Gamma, \Delta, A \otimes B, C}\,\otimes \qquad \Lambda, D}{\Gamma, \Delta, \Lambda, A \otimes B, C \otimes D}\,\otimes$$

Figure 2. Permutations

inferences. Only cut-free proofs are considered, and no cut-rule is added. *Permutations* of inference rules are displayed in Figure 2; the symmetric variants of the last two permutations, *par-tensor* and *tensor-tensor*, have been omitted.

Definition 1. *Equivalence* ($\sim$) of proofs in (cut-free, unit-only) multiplicative linear logic is the congruence generated by the permutations given in Figure 2. *MLL proof equivalence* is the problem of deciding whether two given proofs are equivalent.

The permutations of sequent proofs are exactly the identifications imposed by the categorical semantics of MLL, star-autonomous categories [1] (and semi-star-autonomous categories [11, 12] 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.

Proof nets

Proof nets provide a solution to proof equivalence for MLL without units. For full MLL, they reduce the proof equivalence problem to a simple rewiring relation [15].

Definition 2. For a sequent Γ,

- a *linking* ℓ is a function from the names of $\bot$-subformulae to the names of 1-subformulae,
- a *switching graph* for ℓ is an undirected graph over the names of Γ, with for every subformula $A_a \otimes_c B_b$ the edges a—c and b—c, for every subformula $A_a \parr_c B_b$ either the edge a—c or the edge b—c, and for every subformula $\bot_a$ the edge a—$\ell(a)$,
- a *proof net* ℓ or (Γ, ℓ) is a linking ℓ such that every switching graph is acyclic and connected.

An edge a—$\ell(a)$ in a proof net or switching graph is called a *link* or a *jump*. The restriction that jumps must target 1-occurrences (rather than any connective) is a convenience—it can be circumvented by replacing a subformula A by the equivalent $A \otimes 1$.

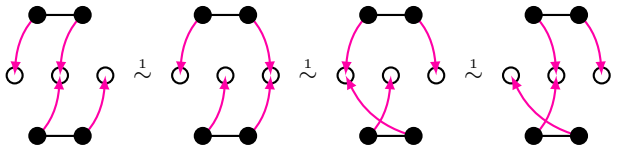


Figure 3. A rewiring sequence on proof nets

Definition 3. A *rewiring* ($\overset{1}{\sim}$) between proof nets is the redirection of exactly one link. *Equivalence* ($\sim$) of proof nets over a sequent Γ is the equivalence generated by rewiring.

An example rewiring sequence is given in Figure 3, using the notation for proof nets introduced below.

To translate a sequent proof to a proof net requires to find a target for each jump from a $\bot$-formula. The inference rule for $\bot$ introduces it into a sequent Γ; in the corresponding proof net, any occurrence of 1 in Γ may serve as the target of the jump.

Definition 4. The relation ($\Mapsto$) interprets a proof Π for a sequent Δ by a linking ℓ as follows: $\Pi \Mapsto \ell$ if for each $\bot_a$ in Δ, if Γ is the context of the inference introducing $\bot_a$, as illustrated below, then $\ell(a)$ is the name of some 1 in Γ.

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

Proposition 5 ([5], [18])**.** *If* $\Pi \Mapsto \ell$ *and* Π *has conclusion* Γ*, then* ℓ *is a proof net for* Γ*. For a net* ℓ *for* Γ*, there is a proof* Π *of* Γ *such that* $\Pi \Mapsto \ell$ *(*sequentialisation*).*

Proof nets are a canonical representation of proofs in the absence of the units: they factor out the permutations among tensor- and par-inferences, which are the last three permutations in Figure 2. Equivalence of proof nets is generated by the four remaining permutations, on $\bot$-introduction.

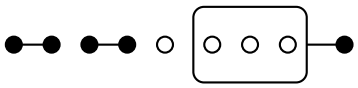
Proposition 6 ([15])**.** *For proofs* Π*,* Π' *and proof nets* ℓ*,* ℓ'*, if* $\Pi \Mapsto \ell$ *and* $\Pi' \Mapsto \ell'$*, then* $\Pi \sim \Pi'$ *if and only if* $\ell \sim \ell'$*.*

The above proposition means that MLL proof equivalence is the problem of deciding equivalence of proof nets.

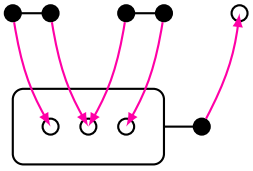
Notation

We will use a concise diagrammatic notation for sequents and proof nets. The units 1 and $\bot$ are represented by a circle ($\circ$) and a disc ($\bullet$) respectively; formulae related by a tensor will be connected by edges; formulae related by a par will be juxtaposed, and collected in a box when a par-formula is an immediate subformula of a tensor-formula (and also for illustrative purposes). For example, the following denote the same sequent:

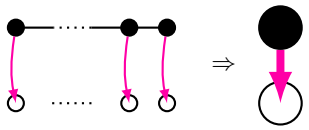
$$\bot \otimes \bot, \bot \otimes \bot, 1, (1 \parr 1 \parr 1) \otimes \bot$$



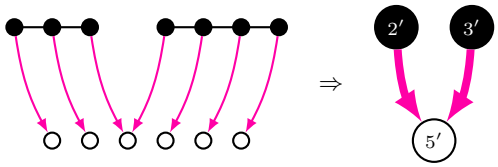
The links of a proof net are added to the sequent as coloured arrows. The following example is a proof net for the above sequent.



We will denote by A^n 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 $\parr \Gamma$ for $A_1 \parr \cdots \parr A_n$. In diagrammatic notation, a big disc will represent an n-ary tensor over only $\bot$-formulae, $\bigotimes(\bot^n)$, and a big circle will represent an n-ary par (or a sequent) over only 1-formulae, $\parr(1^n)$. A linking between $\bigotimes(\bot^n)$ and $\parr(1^n)$ will be represented by a wide arrow, as illustrated below.



Two formulae $\bigotimes(\bot^{i+1})$ and $\bigotimes(\bot^{j+1})$ may together connect to a formula $\parr(1^{i+j+1})$, as illustrated below for $i = 2$ and $j = 3$. Nodes will be labelled i' for $i + 1$, so that both the abbreviated formulae and the arithmetic of connecting them remain intuitive.



A path in a proof net or switching graph is indicated $a\text{---}b$, and illustrated as below.

3. Encoding constraint logic

Non-deterministic constraint logic [6, 8, 9] is a simple graph-rewriting formalism, used here as a convenient tool for PSPACE-hardness reduction. A *constraint graph* is a graph with weighted edges and an *inflow constraint*—a natural number that may be taken to be always 2—on each vertex. A *configuration* is an assignment of directions to the edges of the underlying undirected constraint graph. A rewrite step consists of the reversal of a single edge in a configuration, while preserving the condition that the total weight of the incoming edges at each vertex is at least its inflow constraint.

Figure 4 shows an example rewrite sequence in a part of a constraint graph. The central node has inflow constraint 2, the thick blue edge has weight 2, and the thinner red edges have weight 1.

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. A constraint graph may be encoded as a sequent, and a configuration as a proof net. It is useful for us to generalise the notion of configuration a little: we will allow *partial* configurations, where edges may be left undirected—as long as the inflow constraints are satisfied by the directed edges.

Definition 7. A *constraint graph* $G = (V, E, c, v, w)$ consists of: a set V of vertices with *inflow constraint* $c\colon V \to \mathbb{N}$; and a set E of undirected edges with *weight* $w\colon E \to \mathbb{N}$, connecting the two vertices $v(e) = \{v_1, v_2\} \subseteq V$.

A *(partial) configuration* for a constraint graph is a (partial) function $\gamma\colon E \to V$ such that

- for every edge e, if $\gamma(e)$ is defined then $\gamma(e) \in v(e)$,
- for every vertex v, the total weight of its incoming edges is at least its inflow constraint, $\sum\{w(e) \mid \gamma(e) = v\} \geq c(v)$.

A *reconfiguration step* $\gamma \overset{1}{\sim} \delta$ relates two (partial) configurations for G that differ in value (or definedness) on exactly one edge; this edge is then called *mobile* in γ and δ. The reflexive–transitive closure of $(\overset{1}{\sim})$ will be denoted $(\sim)$.

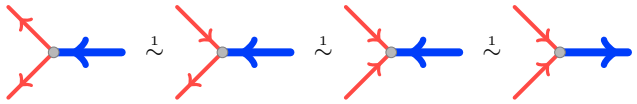


Figure 4. A series of reconfiguration steps in a constraint graph

Proposition 8. *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.*

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. □

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.

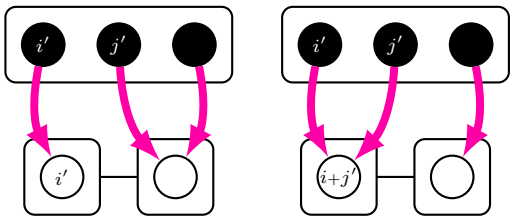
Theorem 9 ([9], Theorem 5.15)**.** *NCG-reconfiguration is PSPACE-complete.*

We will demonstrate the PSPACE-hardness of MLL proof equivalence by an encoding of NCG-reconfiguration in MLL proof nets.

The encoding

The basis of the encoding of constraint graphs in MLL is formed by *weight elements*, which encode one unit of weight on an edge, and *constraint elements*, that encode one unit towards the inflow constraint of a vertex. The arithmetic of linking formulae of the form $\bigotimes(\bot^n)$ to formulae $\parr(1^m)$ will be used to ensure that a weight element may be linked to only two vertices. Below left is a weight element, below right a constraint element.

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 of m (and thus n) will differ for each vertex. The weight element above will be able to connect naturally to those constraint elements where $m = i$ and where $m = i + j$, as illustrated below.



To ensure that no other connections can be made, values are chosen such that $m \equiv 1$ and $n \equiv 2 \pmod 3$, and accordingly $i \equiv 1$, $j \equiv 0$, and $k \equiv 2 \pmod 3$.

In a constraint graph, the sum of all weights is usually greater than the sum of all inflow constraints—otherwise, no edge can move, or no configuration exists. An encoding will therefore have weight elements not connected to constraint elements. These will instead connect to additional, separate 1-formulae, referred to as *weight*

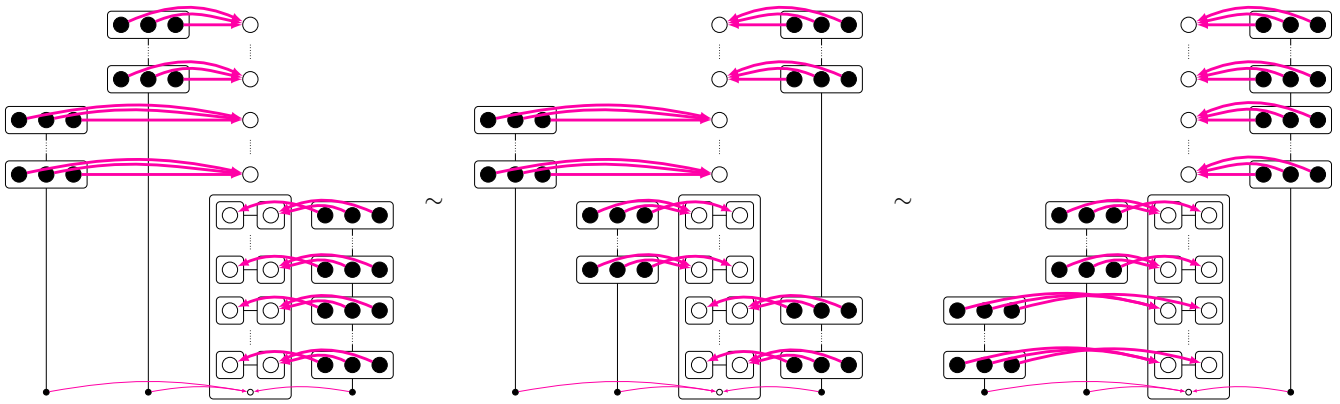
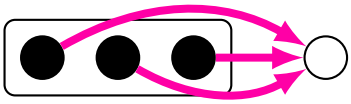
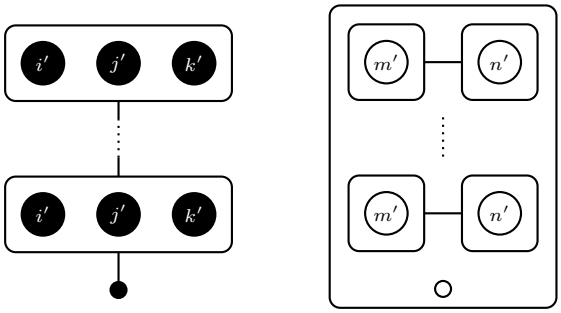


Figure 5. Rewiring three edge-gadgets connected to a single vertex-gadget

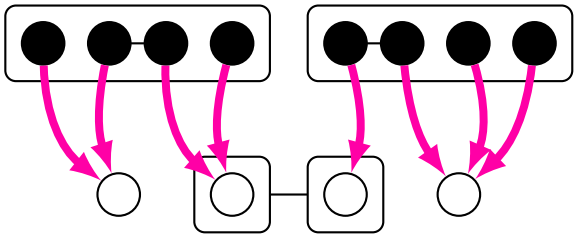
absorbers, as follows.



An edge will be encoded by an *edge-gadget*, illustrated below left, constructed by stringing together a number of similar weight elements plus a single *indicator* vertex. Illustrated below right is a *vertex-gadget* encoding a vertex, formed by a number of constraint elements plus a single *indicator target*.



It would be natural to encode an edge of weight n by an edge-gadget with n weight elements. 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 an inappropriate linking, since both halves of the constraint element are connected, the weight elements must be disconnected—otherwise, the linking would violate the switching condition. 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 $A \parr B \parr C$ 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 therefore requires $n + 1$ edges. It thus suffices to multiply the number of constraint elements by three times the number of edges, and encode a vertex with inflow constraint c by a vertex-gadget with $c \times 3 \times |E|$ constraint elements (where $|E|$ denotes the number of edges in the constraint graph).

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.

Figure 5 displays an encoding of two edges of weight 1, and one of weight 2, connecting to a central vertex with inflow constraint 2, as in the example reconfiguration sequence in Figure 4. The jumps from the three indicator vertices, at the bottom, indicate which vertex an edge is directed at. Indicator jumps can only be rewired between vertices when the edge-gadget is connected only to weight absorbers—in the first net of Figure 5, the two leftmost edges are mobile, and in the third, the rightmost edge is mobile. All three nets in Figure 5 then correspond to the third graph in Figure 4, but the first net allows rewiring according to the second and first graph, whereas the third net allows rewiring according to the fourth graph.

The encoding will be made formal in Section 5; first we will establish some basic results for rewiring on proof nets.

4. 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 5, which show that the encoding of NCG-reconfiguration in MLL proof equivalence, as described informally in the previous section, is correct.

Subnets

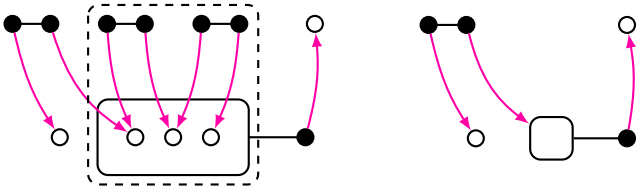
We will discuss (and adapt) some convenient standard notions for MLL proofs and proof nets, and relate them to rewiring. Firstly we will look at subnets—see also [2].

Definition 10. A *sub-sequent* $\Delta \leq \Gamma$ of a sequent Γ is a sequent consisting of disjoint subformulae of Γ, preserving names.

Definition 11. 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 12. A *boxing* s for a linking ℓ for Γ assigns a sub-sequent $s(v) \leq \Gamma$ to each par-vertex $\parr_v$ 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 $s(w) < s(v)$ is closed. The following is then a variation on the local retraction algorithm by Danos [4].

Proposition 13. *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 $\parr_v$ is found by taking the conclusion $\Delta, A \parr_v B$ of its introduction rule, below.

$$\frac{\Delta, A, B}{\Delta, A \parr_v B}\,\parr$$

□

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. We will denote the empire of v in ℓ by $\ell|_v$ (used both as a graph and a set of vertices).

Proposition 14 ([2, 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 $\ell|_v - \{v\}$.*

The following three lemmata show how empires are connected to rewiring. Firstly, a jump from $\bot_v$ may be rewired to exactly those 1-occurences that are in the empire of v (Lemma 15). Secondly, rewiring this jump preserves the empire of v, up to that rewiring (Lemma 16). Thirdly, the empire of any other vertex w will grow or shrink, or neither, but not both (Lemma 17).

Lemma 15. *For a proof net (Γ, ℓ) where $\ell(a) = v$, and w names a 1-occurrence in Γ, the following are equivalent:*

1. *$\ell \overset{1}{\sim} \ell'$ where $\ell'(a) = w$;*
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 [2, 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 in any $\Pi \Rightarrow \ell$ with as conclusion the introduction rule of $\bot_a$. By Definition 4, 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---w passes through a, then in ℓ' there is no path a---v (and two paths a---w) for that switching, so that ℓ' is not a net. □

Lemma 16. *If $\ell \overset{1}{\sim} \ell'$ by $\ell(v) \neq \ell'(v)$ 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 15 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 14 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 $\ell'|_v$). □

Lemma 17. *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 the link from a be rewired, $\ell(a) = w$ but $\ell'(a) = w'$. Using Proposition 14, there are four cases.

1. If the empire of v is propagated from a to w in ℓ, it includes the empire of a. Then by Lemma 16 $\Delta = \Delta'$.
2. If the empire of v includes w but not w', then it is propagated through a in ℓ but not in ℓ', so that $\Delta \geq \Delta'$.
3. If the empire of v includes w' but not w, then $\Delta \leq \Delta'$.
4. Otherwise, $\Delta = \Delta'$. □

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 $\parr$-occurrences. This observation gives a well-known necessary condition for the provability of a sequent.

Definition 18. The *balance* of a sequent is the number of $\bot$s minus the number of $\parr$s and commas. A sequent is *balanced* if its balance is zero.

Proposition 19. *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 $\parr$, and alternating formulae, i.e. every argument of a $\otimes$ is a $\parr$ 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 $\parr$ at the root.

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 $\parr_v$ has one switched edge, to be named $v(1)$; and $\bot_v$ 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 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 6 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.

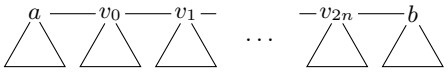
Given two proof nets ℓ and ℓ' for Γ, and a switching graph for each (not necessarily given by the same switching of Γ), we obtain two bijections between edges and (non-root) vertices. Composing these gives a permutation on the non-root vertices.

Definition 20. 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 from v—a to v—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 in Figure 7. 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.

On the right, in Figure 7, 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---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 $\parr$, 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---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 21. *Two equivalent proof nets have even parity.*

Equivalence without $\parr$

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:

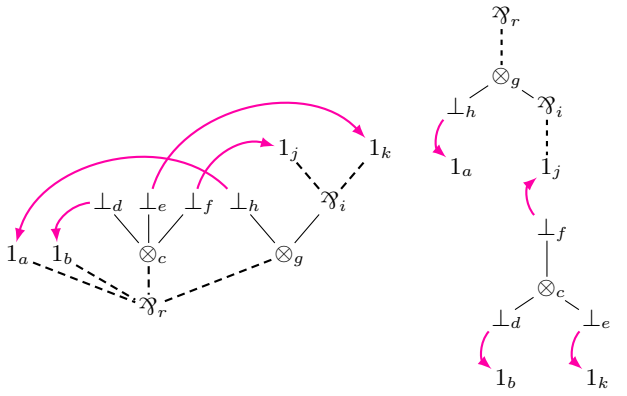


Figure 6. A switching graph of a proof net as a directed tree

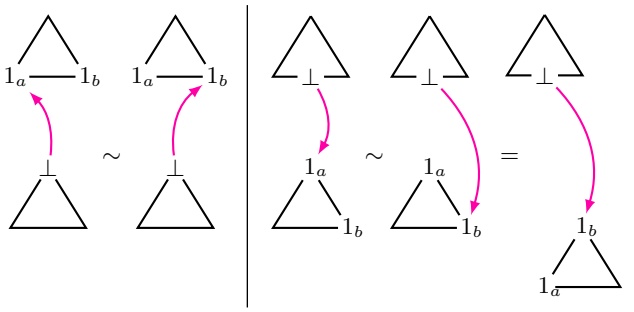


Figure 7. Rewiring on the directed tree of a switching graph

Proposition 22. *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 23. *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 24. *Let ℓ be a proof net with for every switching a path $\ell(a)$—a---b—$\ell(b)$---$\ell(c)$—c---d—$\ell(d)$. Then $\ell \sim \ell'$ where $\ell'(a) = \ell(b)$ and $\ell'(b) = \ell(a)$, $\ell'(c) = \ell(d)$, and $\ell'(d) = \ell(c)$, and $\ell'(z) = \ell(z)$ otherwise.*

Proof. By the rewiring path shown in Figure 8. □

Lemma 25. *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---z is via $\ell(a)$. If $\ell(a) = z$, we are done. Otherwise, by Lemma 15 ℓ' may be obtained from ℓ by changing only $\ell'(a) = z$.

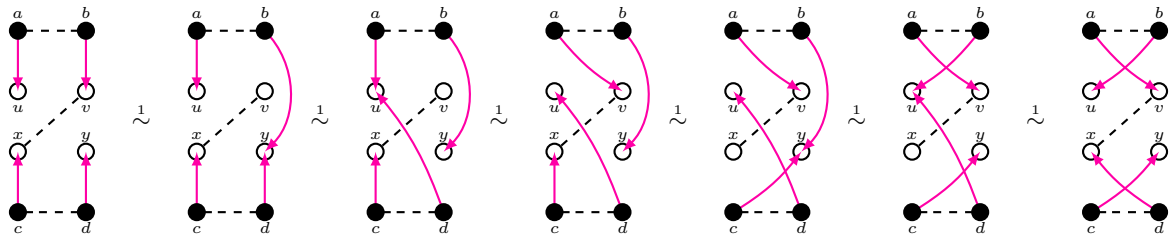


Figure 8. Double exchange of links (Lemma 24)

2) The path $a \dashrightarrow z$ is via some $\bot_b$ in A. Firstly, if $\ell(b) \neq z$, use Lemma 15 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 24 to exchange the targets of a and b, as well as those of c and d. □

Proposition 26. *For a basic sequent with at least two tensor-formulae, two proof nets with even parity are equivalent.*

Proof. By Lemma 25 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. □

5. Formalising the encoding

We will formalise the encoding of NCG-reconfiguration into MLL proof equivalence that was informally introduced in Section 3. 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:

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

Definition 27. 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

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

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

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

The encoding of an edge e connecting vertices v_i and v_j with $i < j$ 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

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

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

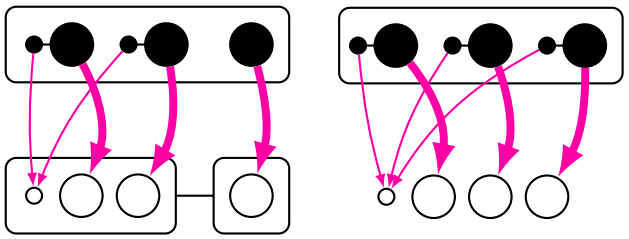
In the above definition, the final 1-subformula of a vertex-gadget $[\![v_k]\!]$ is its *indicator target*; the final $\bot$-subformula of an edge-gadget $[\![e]\!]$ is its *indicator*; and in the completed encoding $[\![G]\!]$ the p instances of 1 are the *weight absorbers*. In a constraint graph G, 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 28. For $W = W_n(i,j) = A \parr B \parr C$ 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 29. *Standard linkings are proof nets.*

The encoding of a configuration is then as follows.

Definition 30. 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 31. *If γ is a total configuration for G then $[\![\gamma]\!]$ is a proof net for $[\![G]\!]$.*

Proof. Using Proposition 13, it is sufficient to give a suitable box for each $\parr$. 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 29. 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 32. 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

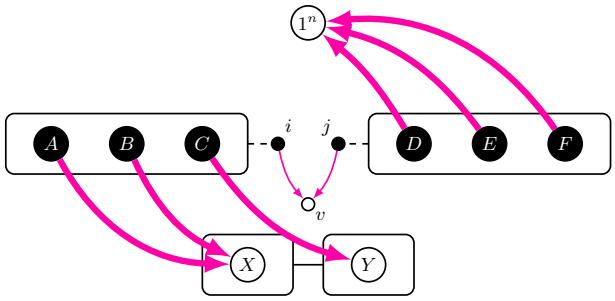
Given a reconfiguration path $\gamma \sim \delta$ over total configurations, we will demonstrate a rewiring sequence between $[\![\gamma]\!]$ and $[\![\delta]\!]_\gamma$. The central part of the argument will be to show how the weight element linking to a constraint element may be exchanged for another (Lemma 33). 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 34).

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.

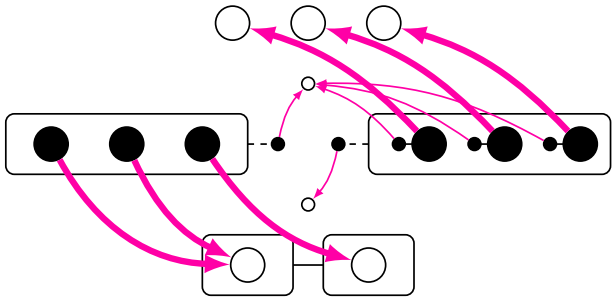
To describe the intermediate stages of such a rewiring sequence, call an edge-gadget $[\![e]\!]$ *well linked* if: 1) its indicator connects to the indicator target of an appropriate vertex-gadget $[\![v]\!]$, and 2) each weight element is either in a standard linking with a constraint element of $[\![v]\!]$, or linked only to weight absorbers (in arbitrary fashion). A well-linked edge-gadget is *mobile* if all its weight elements connect only to weight absorbers. 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 33. *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 C 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 naturally linked to C, W_i is linked to 1^n, and ℓ' agrees with ℓ otherwise.*

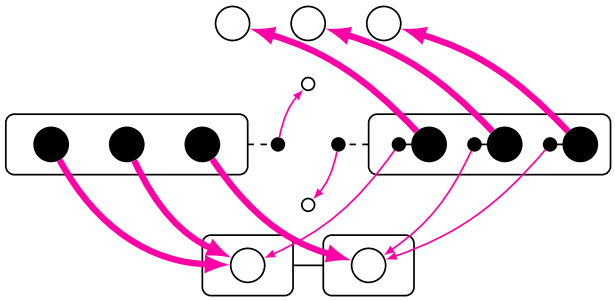
Proof. Let $W_i = A \parr B \parr C$, $W_j = D \parr E \parr F$, and $C = X \otimes Y$. The rewiring path will be illustrated for the case where A, B, X and D, X, and thus also C, Y and E, F, Y, are balanced sequents; other cases are similar.



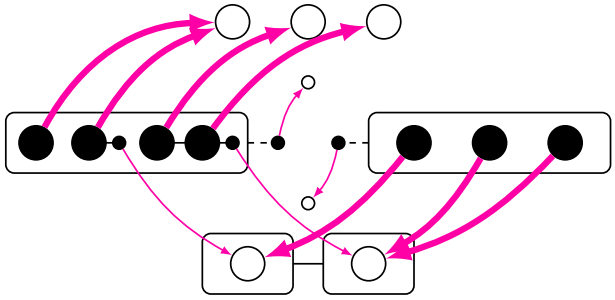
1. The initial configuration is illustrated above. Other edges, other weight and constraint elements, and the outer box of the vertex-gadget, are omitted. The vertices i, j, and v are the indicators of $[\![e_i]\!]$ and $[\![e_j]\!]$ and the indicator target of $[\![v]\!]$, respectively.



2. The link $i - v$ is re-connected to a weight absorber together with only the links from the first $\perp$ of each of D, E, and F.

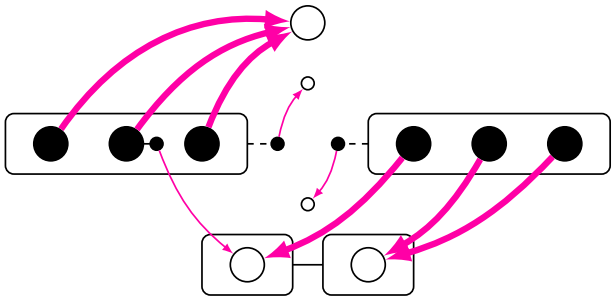


3. The link from the first $\perp$ of D is moved to X, and those from the first $\perp$ of E and F are moved to Y.

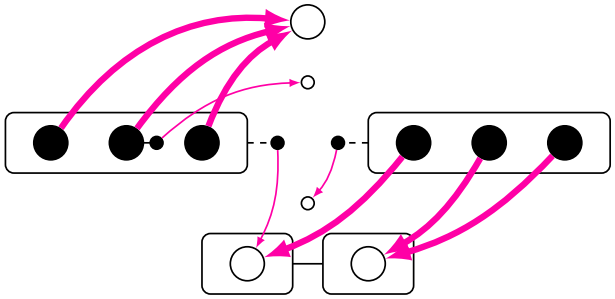


4. In the present configuration, the links from the weight elements form two subnets: one over the sub-sequent $A, B, X, D, 1^m$, and one over the sub-sequent $C, Y, E, F, 1^k$, for some m and k. By Proposition 26, these subnets are equivalent to any other over the same sequent, as long as their parity is preserved. It is then sufficient to choose linkings so that $D \parr E \parr F$ is in a a standard linking with $X \otimes Y$, with a minor adjustment: two links from D to X should remain exchanged, compared to the standard linking, for step

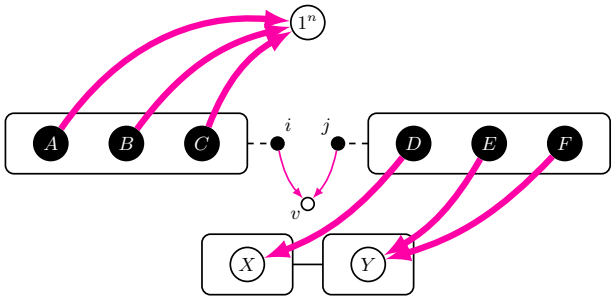
6 below. The links of A, B, C link to the weight absorbers 1^m and 1^k, with one remaining link from B to X and one from C to Y.



5. The link from C to Y is moved towards a weight absorber connected to A, B.



6. The link from B to X is the one remaining connection between the edge-gadgets $[\![e_i]\!]$ and $[\![e_j]\!]$. Lemma 24 allows to swap the targets of the link from B to X and the link from i, and simultaneously undo the exchange in the links from D to X added in step 4 above.



7. The link from i is re-attached to v to yield the final configuration. □

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 34. *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 mobile edge-gadget $[\![e]\!]$.*

Proof. To reconfigure ℓ' into ℓ, we will use the double exchange operation of Lemma 24 to exchange two arbitrary weight absorbers at a time, as well as two chosen jumps in the mobile edge-gadget $[\![e]\!]$. Firstly, let e_0 be the indicator of $[\![e]\!]$. Note that since $[\![e]\!]$ is mobile, 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 24 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. □

Lemma 35. *If $\gamma \sim \delta$ for total configurations γ and δ, then $[\![\gamma]\!] \sim [\![\delta]\!]_\gamma$.*

Proof. By Proposition 8 we may assume that $\gamma \sim \delta$ by a sequence of reconfiguration steps over total configurations. We will prove that if $\gamma \xrightarrow{1} \delta$ 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 33, 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 33, 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 34. □

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 36 ([2, 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 37. *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 36. □

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 38. *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 $W = X \parr Y \parr Z$ 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 $\pmod 3$. 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 39. 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 40. *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 17, 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 38 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. □

6. MLL proof equivalence is PSPACE-complete

We are now ready to state our main theorem.

Theorem 41. *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 [21] 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 35; the direction ($\Leftarrow$) is by Lemma 40 and the observation that $(\!|[\![\gamma]\!]|\!) = \gamma$ and $(\!|[\![\delta]\!]_\gamma|\!) = \delta$. □

Acknowledgments

We would like to thank Lutz Straßburger for comments and discussion, and the anonymous referees for their constructive observations.

References

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

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

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

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

[5] Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[6] 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.

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

[8] 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.

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

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

[11] Willem Heijltjes and Lutz Straßburger. Proof nets and free semi-star-autonomous categories. *MSCS*, 2014. To appear.

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

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

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

[15] Dominic J.D. Hughes. Simple multiplicative proof nets with units. *Annals of Pure and Applied Logic*, 2012. arXiv:math.LO/0507003.

[16] 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.

[17] Max I. Kanovich. Horn programming in linear logic is NP-complete. In *LICS*, 1992.

[18] François Lamarche and Lutz Straßburger. From proof nets to the free *-autonomous category. *LMCS*, 2(4:3):1–44, 2006.

[19] Joachim Lambek. Deductive systems and categories i. syntactic calculus and residuated categories. *Mathematical Systems Theory*, 2(4):287–318, 1968.

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

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

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