Complexity Bounds for Sum-Product Logic via Additive Proof Nets and Petri Nets

Willem Heijltjes
Department of Computer Science
University of Bath

Dominic J. D. Hughes
Department of Mathematics
Stanford University*

***Abstract*—We investigate efficient algorithms for the additive fragment of linear logic. This logic is an internal language for categories with finite sums and products, and describes concurrent two-player games of finite choice. In the context of session types, typing disciplines for communication along channels, the logic describes the communication of finite choice along a single channel.**

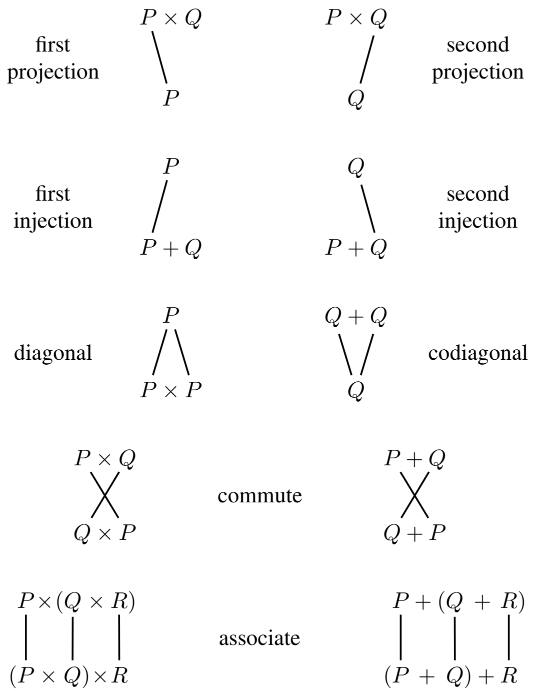
We give a simple linear time correctness criterion for unit-free propositional additive proof nets via a natural construction on Petri nets. This is an essential ingredient to linear time complexity of the second author's *combinatorial proofs* for classical logic.

For full propositional additive linear logic, including the units, we give a proof search algorithm that is linear-time in the product of the source and target formula, and an algorithm for proof net correctness that is of the same time complexity. We prove that proof search in first-order additive linear logic is NP-complete.

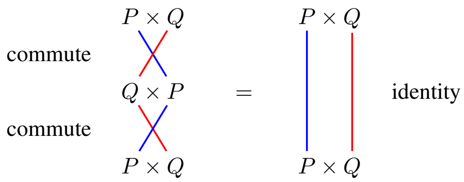
***Index Terms*—linear logic; proof complexity; Petri nets; sum–product categories; additive linear logic**

I. Introduction

Additive proof nets, as formulated in [21], provide an intuitive yet completely formal diagrammatic presentation of canonical maps such as:

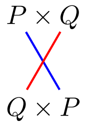


Composition is simply path composition (the standard composition of the underlying binary relations):

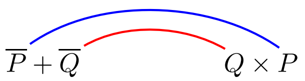


A proof net from a formula A to a formula B is a binary relation from the leaves (atom occurrences P, Q, R,...) of A to the leaves of B, satisfying the *resolution* condition [21].

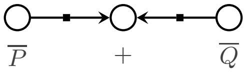
In this paper we present a new correctness criterion for additive proof nets which can be verified in linear time (in the number of edges or *links* in the binary relation), via a natural construction on Petri nets (cartesian product). We illustrate the criterion on commutativity $P \times Q \ \to \ Q \times P$:



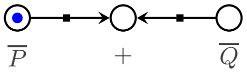
Since $P \times Q$ is a source, it is implicitly dualized. We unfold this duality and place the two formulas side by side:



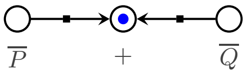
Each formula determines a Petri net. The Petri net of $\overline{P} + \overline{Q}$ has three places ○ (one per symbol "$\overline{P}$", "+", "$\overline{Q}$") and two transitions ▪ :



This Petri net $\mathcal{N}(\overline{P} + \overline{Q})$ captures the disjunctive essence of $+$ in the way it fires. Starting with a single token on $\overline{P}$,

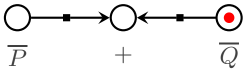


the Petri net can fire, moving the token to the central $+$:

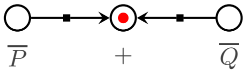


* This research was conducted as a Visiting Scholar in the Mathematics Department at Stanford. I am grateful to my host, Sol Feferman.

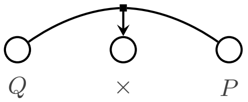
This firing sequence captures the logical disjunction rule "from $\overline{P}$ infer $\overline{P}+\overline{Q}$". The alternative firing sequence, starting with a token on the $\overline{Q}$ place,



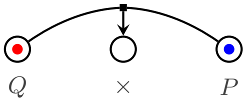
captures "from $\overline{Q}$ infer $\overline{P}+\overline{Q}$", as the token moves to the central $+$:



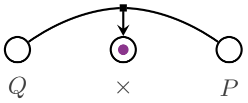
The Petri net $\mathcal{N}(Q \times P)$ has three places (one per symbol) and one transition:



Starting from two tokens, one on each of Q and P,

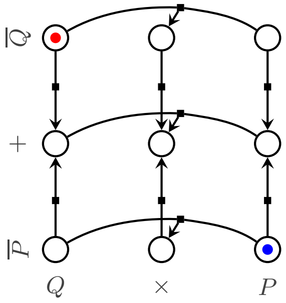


the firing results in a central token on the $\times$ place:

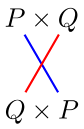


This corresponds to the logical conjunction rule "from Q and P infer $Q \times P$". Note that if we start with only one of the two tokens, the Petri net is deadlocked: we cannot infer $Q \times P$ ("Q AND P") from P alone, or from Q alone.

The next step in checking correctness is to build the cartesian product $\mathcal{N}(\overline{P}+\overline{Q}) \ast \mathcal{N}(Q \times P)$ and place the links of the proof net as tokens on the corresponding places:



Each row is a copy of $\mathcal{N}(Q \times P)$ and each column a copy of $\mathcal{N}(\overline{P}+\overline{Q})$. Each link in the commutativity proof net



determines a token in the initial state: the top-left token, in the "$\overline{Q}$ row" and "Q column", represents the link Q—Q, and the bottom-right token, in the "$\overline{P}$ row" and "P column", represents the link P—P. The proof net is correct if, upon exhaustively firing the Petri net, we end up with a single token in the center (and no other tokens).

An example firing sequence is shown in the top row of Fig. 1. First the bottom-right token fires (upwards); then the top-left token fires (downwards); finally the pair in the middle row fires (horizontally). Because only a single central token remains, we have verified that commutativity is a proof net.

Since in practice we do not have to write out the entire grid of the cartesian product at the outset, the algorithm runs in linear time in the number of tokens (i.e., linear time in the number of links in the binary relation).

A. Relationship to Danos contractibility and top-down sequentialization

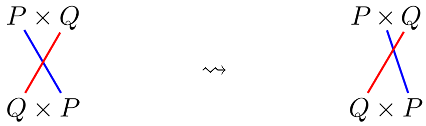
Danos' contractibility criterion for multiplicative linear logic [8] gradually produces a sequentialization by starting with the axiom links (as axiom rules) at the top then flowing downwards rule by rule until reaching the conclusion. Naively quadratic, Guerrini observed that the criterion can be checked in linear time [13]. Our Petri net condition is similar in spirit: the initial tokens (corresponding to links) provide axiom rules at the top of a sequentialization, and every Petri net firing yields a proof rule, top-down from the axiom rules.

Fig. 1 shows how the sequentialization emerges from the firing sequence presented above. It begins with two axiom rules, corresponding to the two tokens in the initial state of the Petri net (and in turn to the two links of the proof net).

When the first token fires, up the rightmost column, it corresponds to "from $\overline{P}$ infer $\overline{P}+\overline{Q}$" (as discussed earlier in the Introduction), and we write down the corresponding rule of two-sided additive linear logic (the internal logic of sum-product categories): "from $P \vdash A$ infer $P \times Q \vdash A$". When the second token fires, down the leftmost column, it corresponds to "from $\overline{Q}$ infer $\overline{P}+\overline{Q}$", and we write down the corresponding rule. When the final pair fires, along the central row, we write down the final $\times$-rule.

B. The coalescence criterion

As tokens fire (the top row of Fig. 1), or rules are written (the middle row), an alternative view is to shuffle links, as presented in the bottom row of Fig. 1. The first move shifts the source of the link P—P from P to the whole formula $P \times Q$, i.e., to the symbol $\times$, or equivalently, to the root vertex $\times$ of the parse tree of the formula:



Similarly, the second move (from the second to third column in the bottom row of Fig. 1) shifts the source of the link Q—Q from Q to the whole formula $P \times Q$. Finally, the two links from $P \times Q$, one to P and one to Q, coalesce into a single link from $P \times Q$ to $Q \times P$.

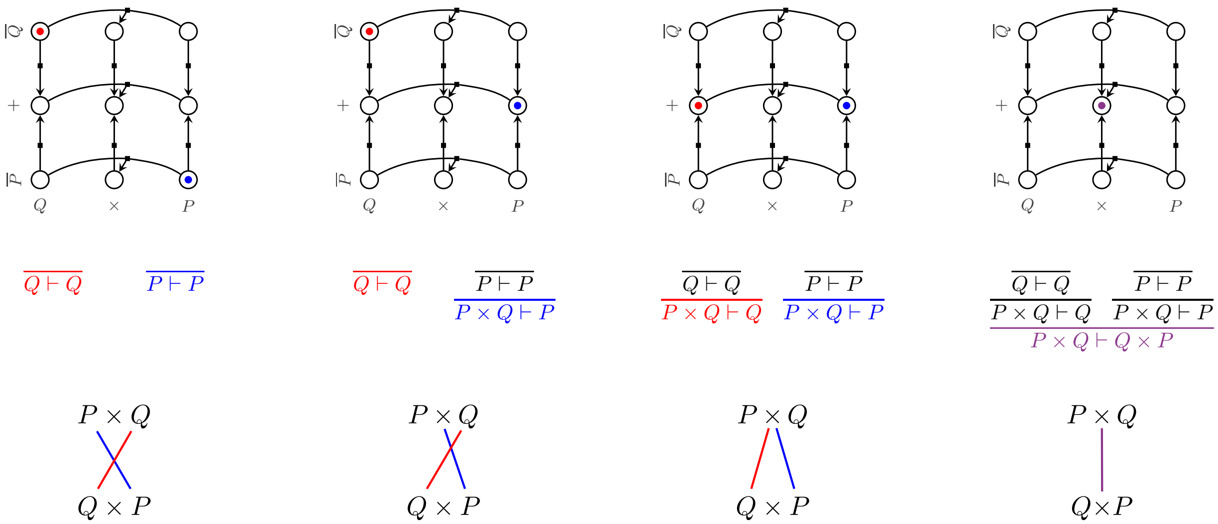


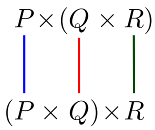
Fig. 1. A tight three-way correspondence: firing the cartesian product Petri net, sequentializing top-down, and coalescing.

This forms the basis of the *coalescence criterion*: a proof net from A to B is correct if, upon carrying out moves such as those described above (each corresponding to a Petri net fire, or to writing down a proof rule), the final result is a single link from the root vertex of A to the root vertex of B.

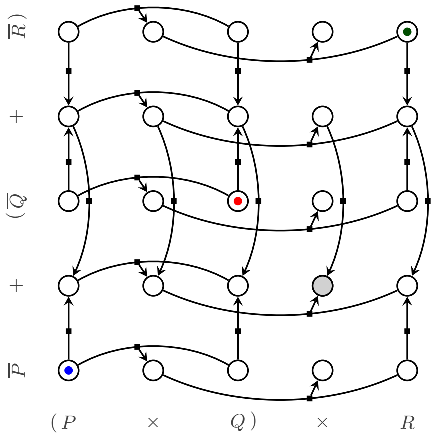
Given the tight correspondence with additive top-down sequentialization, one may consider the coalescence criterion an additive analogue of Danos' multiplicative contractibility.

C. Grid notation

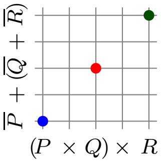
To save space, we abbreviate cartesian product Petri nets in a grid notation. For example, to verify associativity



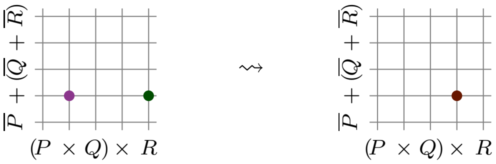
we construct the cartesian product Petri net $\mathcal{N}\left(\overline{P}+(\overline{Q}+\overline{R})\right) * \mathcal{N}\left((P \times Q) \times R\right)$ initialized with three tokens, one per link:



The goal place has been highlighted in grey. The reader may verify that, upon exhaustively firing the Petri net, a single token remains, on the goal (irrespective of firing order). We abbreviate the Petri net to the following grid[1]

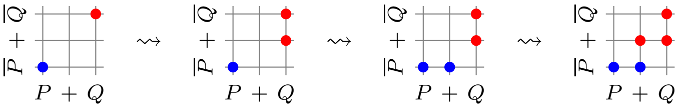


The parse trees of the two labelling formulas determine the legal firings. For example, the following firing is legal based on the structure of $(P \times Q) \times R$:


$\overline{P} + (\overline{Q} + \overline{R})$
$(P \times Q) \times R$
$\rightsquigarrow$
$\overline{P} + (\overline{Q} + \overline{R})$
$(P \times Q) \times R$

D. Efficient provability and proof search

Define *spawning* in a Petri net as the variant of firing obtained by leaving consumed tokens in place. Here is an example spawning sequence (in compact grid notation):



Spawning provides an efficient search for the provability of a sequent $A \vdash B$: initialize the Petri net $\mathcal{N}(A) * \mathcal{N}(B)$ with a token on every place which has dual atomic labels (“row $\overline{P}$

[1] Our two-dimensional grids for sequents $\vdash \overline{A}, B$ generalise to n dimensions for additive sequents $\vdash A_1, \ldots, A_n$ (given a notion of n-ary axiom).

and column P"); spawn repeatedly; the sequent is provable iff we spawn onto the goal place. The above spawning sequence shows the provability of $P \times Q \vdash P + Q$. Upon reversing a successful spawning sequence one can (non-deterministically) extract a proof. Proof search for a sequent $A \vdash B$ can thus be performed in time and space linear in $|A| \times |B|$, the size of the grid, and this remains true in the presence of the units—see Sections III-D and IV-A.

In contrast, conventional bottom-up proof search adds a logarithmic factor to this complexity, for the following reason. By softness [22] a proof of a sequent $A \times B \vdash C + D$ factors through one of the four sequents

$$A \vdash C + D \qquad B \vdash C + D \qquad A \times B \vdash C \qquad A \times B \vdash D\,.$$

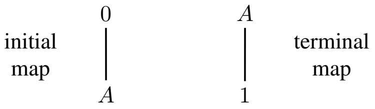
These overlap on four sub-sequents:

$$A \vdash C \qquad B \vdash C \qquad A \vdash D \qquad B \vdash D$$

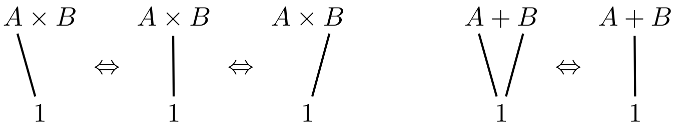
Naive inductive proof search would (in the worst case) search these sequents twice, whereas in our grid notation they are represented and searched only once. Approaches such as *focusing* [3], [5] reduce the number of instances of duplicated search, but do not ultimately solve the problem. The authors are not aware of an inductive, bottom-up search algorithm that matches our complexity.[2]

E. Units

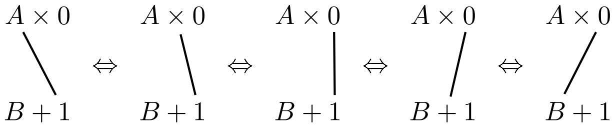
The categorical initial object 0 and terminal object 1, the nullary coproduct and product, are characterised by unique initial maps $0 \to A$ and terminal maps $A \to 1$. These are naturally rendered in proof nets as follows.



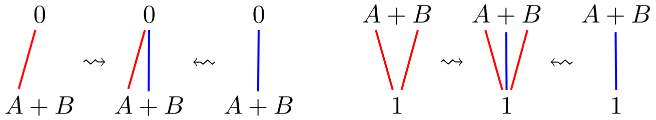
However, in this representation proof nets no longer correspond 1–1 to categorical maps, and the uniqueness property of initial and terminal maps forces an equivalence ($\Leftrightarrow$) on proof nets. For terminal maps it is generated by:



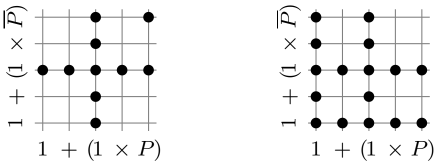
To decide whether two proof nets with units are equivalent is non-trivial, due to the interaction of initial maps and terminal maps via occurrences of the map $0 \to 1$, which is both; e.g.:



Proof net equivalence can be decided by *saturating* proof nets [14], [15]: instead of replacing one link by another in a rewrite step, both links are kept. For example:



The resulting *saturated nets* correspond 1–1 to categorical maps. Below are two example saturated nets, representing the two morphisms from $0 \times (0 + P)$ to $1 + (1 \times P)$ (in grid notation, to manage the link density).



The main result we present for sum–product logic with units, in Section IV, is that correctness of a saturated net for a sequent $A \vdash B$ can be decided in time linear in $|A| \times |B|$.

F. Quantifiers

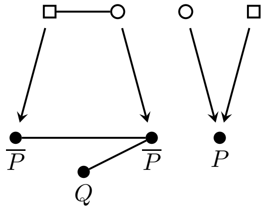
There is an interesting interaction between universal and existential quantifiers, even in the absence of any other logical connectives, or weakening and contraction [27]. In Section V we take a brief look at first-order additive linear logic, which combines the quantifiers with products and coproducts (but without the units). We show that, unlike in the propositional fragment, proof search is NP-hard.

G. Linear time combinatorial proofs for classical logic

Our result that an additive proof net can be verified in linear time implies that combinatorial proofs for propositional classical logic (*Proofs Without Syntax* [19]) can be verified in linear time. A combinatorial proof of Peirce's Law

$$\big((P \Rightarrow Q) \Rightarrow P\big) \Rightarrow P \quad = \quad \big((\overline{P} \vee Q) \wedge \overline{P}\big) \vee P$$

is shown below:



A combinatorial proof of A is a *skew fibration* (a particular kind of undirected graph homomorphism) from a coloured graph into the graph $\mathcal{G}(A)$ associated with A. In the example above, each graph has four vertices, and the upper graph has two colours. Such skew fibrations correspond [20] to homomorphisms which preserve maximal cliques, hence a skew fibration corresponds to a special kind of additive proof net: one which is *functional* in the sense that every source leaf is in one and only one link. Our theorem in this paper that an additive proof net $\lambda : C \vdash A$ can be verified in linear time in $|\lambda|$ implies that a combinatorial proof, where $|\lambda|$ is linear in the size of C, can be verified in linear time. Work in progress to extend combinatorial proofs to first-order classical logic builds on the same linear time complexity result.

[2] It is possible to re-formulate our algorithms to search the grid bottom-up.

H. Related work

An early result is Whitman's Theorem [30] on free lattices. The theorem, which gives a factorisation of lattice inequalities, corresponds closely to cut-elimination in sum–product logic.

The advent of linear logic [11], where categorical products and coproducts are captured by the additive fragment, sparked a wave of syntactic and semantic approaches, to capture the multiplicatives and exponentials, and sometimes also the additives. Joyal generalised Whitman's Theorem to bicomplete categories [22], categories with all limits and colimits, and gave an interpretation of cut-elimination in game semantics [23]. A canonical treatment of the additives, without the units, was first given by Hu in terms of *contractible* coherence spaces [17]. Canonical proof nets are a fragment of the MALL proof nets of the second author and van Glabbeek [21].

In the presence of the units, proof equivalence was investigated first by Cockett and Seely [7], and later by Cockett and Santocanale [6]. The latter give an effective, intricate decision procedure, based on a careful analysis of the structure of finite sum–product categories. A canonical syntax for additive linear logic with units, *saturated nets*, was given by the first author [14], [15].

From the perspective of game semantics, the problem of representing additive proofs canonically surfaced as the issue that Blass games [4] were not associative [1]. The problem is one of concurrency: in games, a map from A to B is interpreted as a parallel game on $\overline{A}$ and B, where the coproducts and products in $\overline{A}$ and B represent binary choice for Player and Opponent respectively. This problem was addressed in detail by Abramsky and Melliès [2].

A related interpretation of an additive sequent is as a protocol for concurrent communication along a single channel [6], which is finding its way into practial use in the idea of *session types* (see e.g. [29]).

Many results in the paper have close analogues in the multiplicative fragment, or other fragments of linear logic. The coalescence condition is related to Danos's contractibility [8]. Like the latter, which was used by Guerrini to show linear-time correctness of MLL proof nets [13], it provides an effective correctness algorithm. Provability for MLL was first shown to be NP-complete by Kanovich [24]. This and other complexity results were compounded in an early overview [25], which interestingly does not include any results for the additive fragment. The correctness of MALL proof nets was found to be NL-complete [9], but this appears not to impact the (much more restricted) purely additive fragment. Proof equivalence for MLL with units was recently shown to be PSPACE-complete, by Robin Houston and the first author [16].

In one version of proof nets, the additives are treated using *monomial weights* [12]. These have computational advantages, but they are not canonical. While Danos's contractibility has been extended to these proof nets [26], the difference in structure between weighted and non-weighted proof nets means that this approach is not related to our coalescence.

ALL:

$$\frac{}{P \vdash P} \qquad \frac{A \vdash B_i}{A \vdash B_0 + B_1} \qquad \frac{A \vdash B \qquad A \vdash C}{A \vdash B \times C}$$

$$\frac{A \vdash C \qquad B \vdash C}{A + B \vdash C} \qquad \frac{A_i \vdash B}{A_0 \times A_1 \vdash B}$$

Units:

$$\frac{}{0 \vdash A} \qquad \frac{}{A \vdash 1}$$

Quantifiers:

$$\frac{A \vdash B[t/x]}{A \vdash \exists x.B} \qquad \frac{A \vdash B[y/x]}{A \vdash \forall x.B} \quad y \text{ not free in } A$$

$$\frac{A[t/x] \vdash B}{\forall x.A \vdash B} \qquad \frac{A[y/x] \vdash B}{\exists x.A \vdash B} \quad y \text{ not free in } B$$

Identity & composition:

$$\frac{}{A \vdash A} \qquad \frac{A \vdash B \qquad B \vdash C}{A \vdash C}$$

Fig. 2. Additive linear logic.

II. Additive Linear Logic

We will consider three fragments of additive linear logic: propositional without units (ALL), propositional with units (ALLU), and first-order without units (FOALL). Fix a set of ***atoms*** $\{P, Q, R, \ldots\}$, which in the first-order case includes predicates $P(t_1, \ldots, t_n)$ over a first-order term language. We take t, u, v to range over first-order terms and x, y, z to range over term variables.

Formulas are generated by the following grammars:

$$\begin{array}{lclclcll} A, B, C & ::= & P & \mid & A + B & \mid & A \times B & \\ & & & \mid & 0 & \mid & 1 & \text{ALLU only} \\ & & & \mid & \exists x.A & \mid & \forall x.A & \text{FOALL only} \end{array}$$

A ***sequent*** $A \vdash B$ comprises a ***source*** formula A and a ***target*** formula B. Sequent calculi for the three fragments are given in Fig. 2. Each fragment includes the inference rules for ALL, while ALLU adds the unit rules and FOALL adds the quantifier rules. The rules for composition and identity are admissible in each fragment.

Theorem 1. *Cut-elimination and identity-elimination holds for* ALL, ALLU, *and* FOALL.

This result goes back to Whitman's Theorem for free lattices [30], which essentially states that the cut-free sequent calculus for ALLU is complete; see also [23, Appendix].

III. ALL PROOF NETS

By a ***subformula*** of a formula A we mean an *occurrence*, as distinguished by a rooted path in the formula tree of A. We denote the size of A by $|A|$, measured in the number of subformulae, or equivalently, connectives and atoms. Given a sequent $A \vdash B$, a ***link*** C — D connects a ***source*** subformula C in A to a ***target*** subformula D in B. An ***axiom link*** is a link between occurrences of the same atom.

Definition 2. A ***linking*** on a sequent $A \vdash B$ is a set of links on $A \vdash B$. An ***axiom linking*** is a linking whose every link is an axiom link.

We write $\lambda : A \vdash B$ to indicate that λ is a linking on $A \vdash B$.

A ***resolution*** r for an additive formula A is a function choosing one child for each subformula that is a product, i.e. either B or C for each subformula $B \times C$ [21]. A subformula C of A is ***retained*** (opposite: ***discarded***) by r if whenever C is a subformula of B_i in a subformula $B_0 \times B_1$ of A, then r chooses B_i. Dually, a ***co-resolution*** for A chooses on coproducts, and a resolution for a sequent $A \vdash B$ is a pair $r = (r_A, r_B)$ where r_A is a co-resolution for A and r_B is a resolution for B. A link in $\lambda : A \vdash B$ is retained by r if both its source is retained by r_A and its target is retained by r_B.

Definition 3. A linking $\lambda : A \vdash B$ is ***discrete*** if every resolution for $A \vdash B$ retains exactly one link in λ.

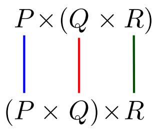
A ***proof net*** is a discrete axiom linking.

Definition 4. A proof Π of $A \vdash B$ translates to the axiom linking $[\![\Pi]\!] = \lambda : A \vdash B$ where λ collects the axioms in Π as axiom links.

A proof of ×-associativity will provide a running example:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{P\vdash P}}{P\times(Q\times R)\vdash P} \qquad \dfrac{\dfrac{\dfrac{}{Q\vdash Q}}{Q\times R\vdash Q}}{P\times(Q\times R)\vdash Q}}{P\times(Q\times R)\vdash P\times Q} \qquad \dfrac{\dfrac{\dfrac{}{R\vdash R}}{Q\times R\vdash R}}{P\times(Q\times R)\vdash R}}{P\times(Q\times R)\vdash (P\times Q)\times R}$$

It translates to the following proof net:



Proposition 5 ([18], [21])**.** *The translation $[\![\Pi]\!]$ of a proof Π is a proof net. For any proof net $\lambda : A \vdash B$ there is a proof Π of $A \vdash B$ such that $[\![\Pi]\!] = \lambda : A \vdash B$.*

The proof Π is a ***sequentialisation*** of λ.

A. Petri Net criterion

A ***transition*** on a set $\mathcal{P}$ is pair $\langle s, t\rangle$ whose ***source*** s and ***target*** t are subsets of $\mathcal{P}$.[3] A ***Petri net*** [28] $(\mathcal{P}, \mathrel{-\!\blacksquare\!\!\to})$ is a set $\mathcal{P}$ of ***places*** and a set $\mathrel{-\!\blacksquare\!\!\to}$ of transitions on $\mathcal{P}$. We abbreviate $\langle\ \{p_1,\dots,p_m\},\ \{q_2,\dots,q_n\}\ \rangle \ \in\ \mathrel{-\!\blacksquare\!\!\to}$ to $p_1,\dots,p_m \mathrel{-\!\blacksquare\!\!\to} q_1,\dots,q_n$. Example Petri nets were drawn in the Introduction with places as circles ○ and a transition $\langle s, t\rangle$ as a black square ▪ with an undirected edge from each place in s and a directed edge to each place in t.

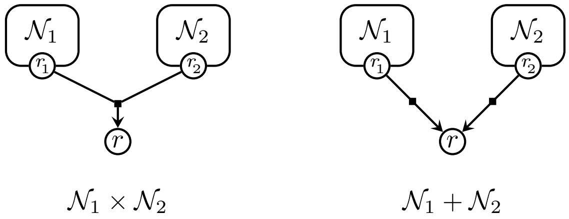
A ***marking*** is a subset M of $\mathcal{P}$. Elements of M are ***tokens***. ***Firing*** is the rewrite relation on markings defined by

$$M \quad \rightsquigarrow \quad (M \setminus s) \cup t$$

whenever $s \subseteq M$, $s \mathrel{-\!\blacksquare\!\!\to} t$, and t and M are disjoint. The top row of Fig. 1 shows a firing sequence, with tokens ●.

A ***root*** is a place r which is not in the source of any transition. A Petri net is ***rooted*** if it has a unique root. Henceforth assume every Petri net is rooted.

Every ALL formula A determines a Petri net $\mathcal{N}(A)$ upon interpeting the symbols × and + as operations on Petri nets. Let $\mathcal{N}_1 = (\mathcal{P}_1, \mathrel{-\!\blacksquare\!\!\to}_1)$ and $\mathcal{N}_2 = (\mathcal{P}_2, \mathrel{-\!\blacksquare\!\!\to}_2)$ be Petri nets with respective roots r_1 and r_2. Define $\mathcal{N}_1 \times \mathcal{N}_2$ as disjoint union plus a transition from the two roots to a new root r, and define $\mathcal{N}_1 + \mathcal{N}_2$ as disjoint union plus two transitions to a new root r, one from r_1 and the other from r_2.



Formally, where $\sqcup$ denotes disjoint union:

$$\mathcal{N}_1 \times \mathcal{N}_2 \;=\; (\,\mathcal{P}_1 \sqcup \mathcal{P}_2 \sqcup \{r\}\,,\ \mathrel{-\!\blacksquare\!\!\to}_1 \sqcup \mathrel{-\!\blacksquare\!\!\to}_2 \sqcup \mathrel{-\!\blacksquare\!\!\to}_\times\,)$$

$$\mathcal{N}_1 + \mathcal{N}_2 \;=\; (\,\mathcal{P}_1 \sqcup \mathcal{P}_2 \sqcup \{r\}\,,\ \mathrel{-\!\blacksquare\!\!\to}_1 \sqcup \mathrel{-\!\blacksquare\!\!\to}_2 \sqcup \mathrel{-\!\blacksquare\!\!\to}_+\,)$$

where $\mathrel{-\!\blacksquare\!\!\to}_\times$ and $\mathrel{-\!\blacksquare\!\!\to}_+$ are defined by:

$$r_1, r_2 \mathrel{-\!\blacksquare\!\!\to}_\times r \qquad\qquad \begin{array}{l} r_1 \mathrel{-\!\blacksquare\!\!\to}_+ r \\ r_2 \mathrel{-\!\blacksquare\!\!\to}_+ r \end{array}$$

Both Petri nets are rooted, with root r. Define the Petri net $\mathcal{N}(P)$ of an atom P as a single place with no transitions, define $\mathcal{N}(A \times B) = \mathcal{N}(A) \times \mathcal{N}(B)$ and define $\mathcal{N}(A + B) = \mathcal{N}(A) + \mathcal{N}(B)$. Examples can be found in the Introduction. By induction the places of $\mathcal{N}(A)$ are in bijection with the subformulas of A, and A itself corresponds to the root place.

The ***cartesian product*** $\mathcal{N}_1 * \mathcal{N}_2$ is $(\mathcal{P}_1 \times \mathcal{P}_2, \mathrel{-\!\blacksquare\!\!\to})$ where $\mathrel{-\!\blacksquare\!\!\to}$ is defined by

$$\{p_1\} \times s_2 \mathrel{-\!\blacksquare\!\!\to} \{p_1\} \times t_2 \qquad \text{for all } p_1 \in \mathcal{P}_1 \text{ and } s_2 \mathrel{-\!\blacksquare\!\!\to}_2 t_2$$

$$s_1 \times \{p_2\} \mathrel{-\!\blacksquare\!\!\to} t_1 \times \{p_2\} \qquad \text{for all } p_2 \in \mathcal{P}_2 \text{ and } s_1 \mathrel{-\!\blacksquare\!\!\to}_1 t_1$$

[3] Some more general definitions take s and t to be multisets, and markings to be multisets. We shall not require this level of generality.

Write $\overline{A}$ for the De Morgan dual of a formula A: $\overline{P}$ is formally dual to P, $\overline{A \times B} = \overline{A} + \overline{B}$ and $\overline{A+B} = \overline{A} \times \overline{B}$. Define $\mathcal{N}(A \vdash B) = \mathcal{N}(\overline{A}) * \mathcal{N}(B)$. (The Introduction has examples.) Via the subformula-to-place bijections, every link $A' — B'$ on $A \vdash B$ corresponds to a distinct place $\mathcal{P}(A' — B')$ in $\mathcal{N}(A \vdash B)$, and $\mathcal{P}(A — B)$ is the root.

The ***root marking*** contains only the root. Let $\lambda : A \vdash B$ be a linking. The ***link marking*** of λ is the marking of $\mathcal{N}(A \vdash B)$ determined by the links of λ: $\{\, \mathcal{P}(A' — B') : A' — B' \in \lambda \}$. A ***run*** of λ is a maximal firing sequence starting from the link marking of λ; the last marking of the sequence is the ***result***.

Definition 6. The ***Petri net criterion*** is the following function of a linking: choose a run; if the result is the root marking return PASS, otherwise return FAIL.

The results below follow from the next section, in which we recast the Petri net criterion as the coalescence criterion.

Theorem 7. *The Petri net criterion is deterministic: its* PASS/FAIL *output is independent of the choice of run.*

Theorem 8. *An axiom linking is a proof net iff it satisfies the Petri net criterion.*

B. Coalescence criterion

We recast the Petri net criterion directly in terms of linkings to define the *coalescence criterion*. Coalescence is analogous to Danos' *contractibility* for multiplicative proof nets [8]: it is a simple rewriting procedure whose rewrite steps correspond directly to ALL inference rules, which is confluent on proof nets and reduces them to trivial form. The rewrite rules, illustrated in Fig. 3, are as follows.

Definition 9. ***Coalescence*** is the rewrite relation on linkings generated by the following rewrite rules:

- replace a link $A — B$ or one $A — C$ by one $A — B + C$;
- replace two links $A — B$ and $A — C$ by one $A — B \times C$;
- replace two links $A — C$ and $B — C$ by one $A + B — C$;
- replace a link $A — C$ or one $B — C$ by one $A \times B — C$.

A linking $\lambda : A \vdash B$ ***weakly coalesces*** if there is a sequence of coalescence steps starting from λ and ending in a single link $A — B$. A linking $\lambda : A \vdash B$ ***coalesces*** if any coalescence sequence terminates in a single link $A — B$.

Fig. 4 illustrates the coalescence of our running example, the associativity proof net. The coalescence rewrite steps of Definition 9 correspond one-to-one to the inference rules of ALL in Fig. 2, where the links in the left- and right-hand side of the coalescence step correspond, respectively, to the premises and conclusion of the inference rule. It is readily observed that inference rules and coalescence steps preserve discreteness in both directions. This leads to the following proposition.

Proposition 10. *A linking is discrete iff it weakly coalesces.*

Proof. By a minor generalisation of Proposition 5, a linking $\lambda : A \vdash B$ is discrete if and only if it sequentialises to a derivation in ALL extended with arbitrary axioms $\overline{C \vdash D}$ (which correspond to links). Using the correspondence between inference rules and coalescence steps, this sequentialisation may be turned into a coalescence sequence terminating in the single link $A — B$, and vice versa. □

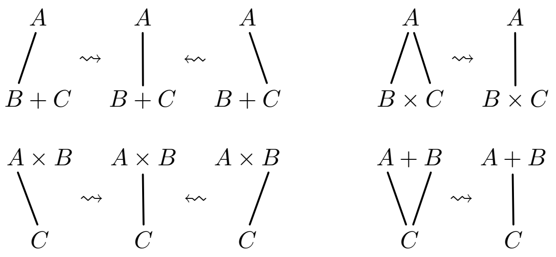


Fig. 3. Coalescence rules.

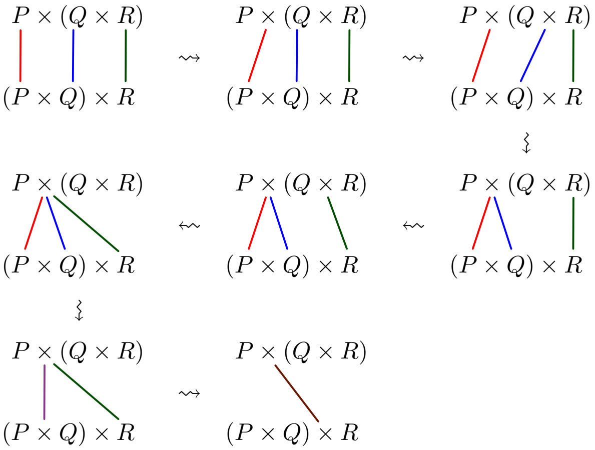
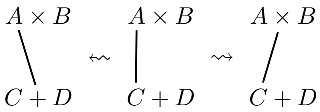


Fig. 4. Coalescing the associativity example.

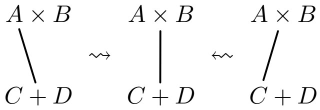
Proposition 11. *Coalescence is confluent on discrete linkings.*

Proof. The critical pairs of the coalescence relation are the pairs of rewrite steps where both replace the same link. At most two rewrite rules apply to a link, one determined by the parent formula of the source, and one by that of the target. It will be shown that every critical pair can be resolved.

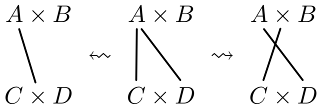
For a source with parent $A \times B$ and target with parent $C + D$, a critical pair



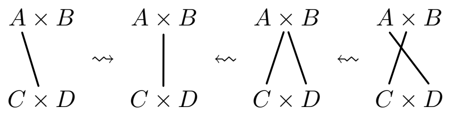
resolves immediately:



The other cases, for links $A — C$, $B — C$, and $B — D$, are similar. For $A \times B$ and $C \times D$, a critical pair

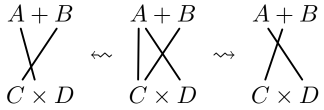


resolves as follows.



Other cases are similar, as are those for $A + B$ and $C + D$.

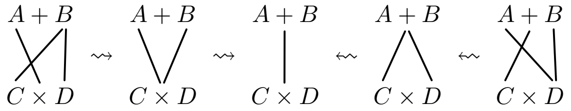
For $A + B$ and $C \times D$, a critical pair is



To resolve the pair, a link B — D is needed. We show that further coalescence exposes this link.

Let the configuration above centre occur in a linking $\lambda : X \vdash Y$. Since coalescence preserves discreteness, this linking is discrete, and any resolution r of $X \vdash Y$ retaining $B \vdash D$ must retain exactly one link. This link must lie within $B \vdash D$, for the following reason. By changing r to choose A rather than B in X, or C rather than D in Y, one of the links A — C, A — D, or B — C, is retained. Then by discreteness, no link outside $A + B \vdash C \times D$ may be retained by r. The links in $B \vdash D$ thus form a discrete linking, which by Proposition 10 coalesces to a single link B — D.

With B — D present, the critical pair is resolved as follows.



The other cases are similar. □

Our main theorem on coalescence is:

Theorem 12. *A linking is discrete if and only if it coalesces.*

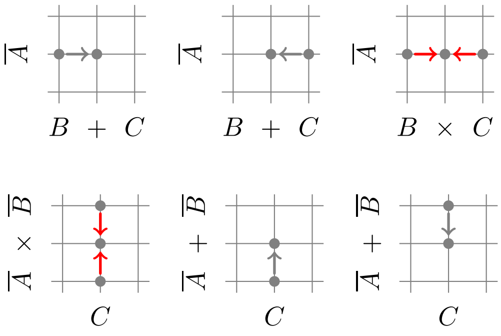
Proof. By Proposition 10 a linking is discrete if and only if it weakly coalesces, and by Proposition 11 it weakly coalesces if and only if it coalesces. □

Corollary 13. *Correctness of an* ALL *proof net* $\lambda : A \vdash B$ *is decidable in time* $\mathcal{O}(|A| \times |B|)$.

Theorem 14. *Correctness of an* ALL *proof net* $\lambda : A \vdash B$ *is decidable in time* $\mathcal{O}\big(|\lambda| \times (\mathsf{d}A + \mathsf{d}B) \times \max(\log|A| + \log|B|)\big)$.

Here $|\lambda|$ is the number of links in λ, $|C|$ is the number size of a formula C, and $\mathsf{d}C$ is the depth of C.

Proof. During coalescence the maximum number of times a token (link) can fire before termination is $\mathsf{d}A + \mathsf{d}B$, since each firing moves one step closer to the root of one of A or B. After each firing we must check if a new two-token firing becomes enabled. This requires maintaining a data structure to retrieve candidates for the other token. Employing a balanced binary tree (for example) provides worst case $\log n$ complexity for look-up among n candidates. Since every candidate is from either A or B, the look-up complexity is bounded by $\max(\log|A| + \log|B|)$. □



$\overline{A}$
$B + C$
$\overline{A}$
$B + C$
$\overline{A}$
$B \times C$
$\overline{A} \times \overline{B}$
C
$\overline{A} + \overline{B}$
C
$\overline{A} + \overline{B}$
C

Fig. 5. Inference rules in grid notation

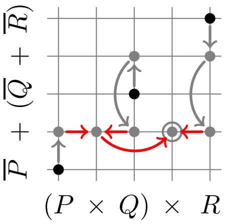
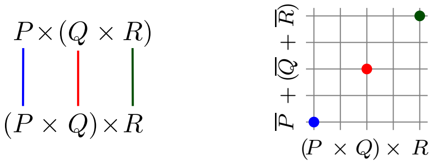


Fig. 6. The associativity proof in grid notation

C. Grid notation

In a discrete linking $\lambda : A \vdash B$ links are sparse, i.e. λ is small relative to its potential maximal size of $|A| \times |B|$. For non-sparse sets of links, where the proof net representation used thus far is not convenient, we will introduce the representation used below right.



Links are displayed as tokens on a grid. The source formula is displayed up the left side (with dualization made explicit) and the target formula along the bottom. Horizontal grid lines correspond to source subformulas, and vertical gridlines to target subformulas, so each crossing is a potential link.

To concisely represent proofs in a grid, an inference will be drawn as one or two arrows between link tokens for its premises and conclusion. Inference rules in grid notation are given in Fig. 5, and the example associativity proof is given in Fig. 6. Axiom links are drawn as black tokens, other links are in grey, and the root link is circled.

D. Efficient proof search: spawning

We define *spawning* as a variant of coalescence which performs efficient proof search. To establish provability of $A \vdash B$, a linking is generated in which a link C — D indicates the provability of a sub-sequent $C \vdash D$ of $A \vdash B$. The initial configuration is the linking $\lambda_{A,B} : A \vdash B$ comprising every possible axiom link on $A \vdash B$. The spawning rewrite relation

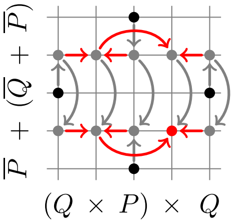


Fig. 7. Spawning a provability grid

that propagates provability is the variant of coalescence in which links are only added, but never deleted.

Definition 15. ***Spawning*** is the rewrite relation generated by the following rewrite steps:

- given a link A — B or one A — C add A — $B + C$;
- given two links A — B and A — C add A — $B \times C$;
- given two links A — C and B — C add $A + B$ — C;
- given a link A — C or one B — C add $A \times B$ — C.

The *provability grid* of a sequent $A \vdash B$ is the result of exhaustive spawning on $\lambda_{A,B} A \vdash B$.

Theorem 16. *A sequent $A \vdash B$ is provable in* ALL *if and only if its provability grid contains the root link A — B.*

Proof. By induction on the spawning relation, a link C — D exists in the provability grid iff there is a proof of $C \vdash D$. □

An algorithm implementing spawning need perform only a single pass over the grid of $A \vdash B$, by respecting the product order over the subformula relation—i.e. C — $D \leq C'$ — D' if and only if C is a subformula of C', and D one of D'.

Corollary 17. *Provability for an* ALL *sequent $A \vdash B$ is decidable in time $\mathcal{O}(|A| \times |B|)$.*

Fig. 7 shows a provability grid, as it is generated by spawning, for the sequent $P \times (Q \times P) \vdash (Q \times P) \times Q$. As the illustration suggests, to obtain a witness—an actual proof—from a provability grid, it suffices to (non-deterministically) retrace the steps taken by the spawning relation.

IV. ALLU

Canonical nets for ALLU, *saturated nets*, were introduced by the first author in [14]. They are obtained from ALL proof nets extended with *unit links* by a *saturation* rewrite procedure.

Definition 18. A ***unit link*** is a link 0 — A or A — 1. An ALLU ***proof net*** is a discrete linking in which every link is an axiom link or a unit link.

Definition 19. ***Equivalence*** ($\Leftrightarrow$) of ALLU proof nets is the equivalence relation generated by the rewrite steps:

- replace two links A — 1 and B — 1 by one $A + B$ — 1;
- replace a link A — 1 or one B — 1 by one $A \times B$ — 1;
- replace a link 0 — B or one 0 — C by one 0 — $B + C$;
- replace two links 0 — B and 0 — C by one 0 — $B \times C$.

Proof net equivalence is illustrated in Fig. 8.

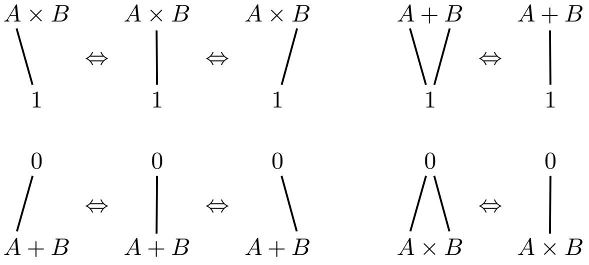


Fig. 8. Equivalence of ALLU proof nets

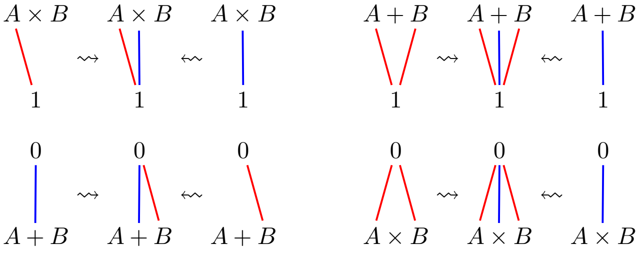


Fig. 9. Saturation (symmetric cases omitted)

Definition 20. ***Saturation*** is the rewrite relation on ALLU linkings generated by:

- given A — 1 and B — 1, add $A + B$ — 1, and vice versa;
- given A — 1 or B — 1, add $A \times B$ — 1, and vice versa;
- given 0 — B or 0 — C, add 0 — $B + C$, and vice versa;
- given 0 — B and 0 — C, add 0 — $B \times C$, and vice versa.

Saturation is illustrated in Fig. 9.

Definition 21. A ***saturated net*** is the normal form of an ALLU proof net with respect to saturation.

Another characterisation of saturated nets is as follows: a saturated net collects the links of all ALLU proof nets in an equivalence class [15, Proposition 3.3.2].

The main theorem of the first author's previous work on additive linear logic is that saturated nets are canonical for finite sum–product categories [14], [15].

Theorem 22 ([15, Theoreom 3.3.1])**.** ALLU *proof nets are equivalent if and only if they have the same saturation.*

The time complexity of saturation is as follows. A single step in the saturation process can be performed in constant time, since placing a link token requires the inspection of only direct neighbours (on the grid). A saturation algorithm that keeps a stack of potential links to place, for each link placed pushing (some of) its neighbours onto the stack, visits only a constant number of new link positions for each link in the eventual saturated net. This gives the following proposition.

Proposition 23 ([15, Section 3.5])**.** *Saturation is linear-time in the size of the saturated net.*

Before moving on, we briefly remark on ALLU proof search.

A. *Proof search*

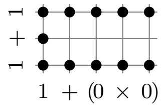
The proof search algorithm for ALL can be used to find proof nets for ALLU simply by extending it with unit links. To find a saturated net it is sufficient to saturate the proof net found by proof search.

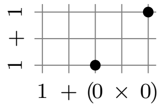
Proposition 24. *Provability of a sequent $A \vdash B$ in* ALLU *is decidable in time $\mathcal{O}(|A| \times |B|)$.*

B. *Correctness*

Neither of the two characterisations of saturated nets directly offers an effective algorithm to determine their correctness. Here, we will introduce such an algorithm: a *desaturation* procedure that given a linking $\sigma : A \vdash B$, in time $\mathcal{O}(|A| \times |B|)$ returns a proof net $\lambda : A \vdash B$ whose saturation is σ if and only if $\sigma : A \vdash B$ is a saturated net. The key ingredient to the algorithm is a *factorisation* grid similar to the provability grid used for ALL proof search, which makes the factorisation of a saturated net (through projections, injections etc.) accessible without costly backtracking.

Note that it is not sufficient for desaturation to simply pick a subnet (a subset of its links, forming a net) of the saturated net. This is illustrated by the two saturated nets below, where the second is contained in the first.





The dynamics of saturated nets are dominated by the interaction between initial and terminal maps, via the unique map $0 \to 1$. A *copointed* map is one that factors through 0, a *pointed* map one that factors through 1, and for any $A \to B$ there is at most one *bipointed* map, that factors through both.

Definition 25. The ***pointed*** respectively ***copointed*** formulae of ALLU are given by:

$$X \quad ::= \quad 1 \quad | \quad X + A \quad | \quad A + X \quad | \quad X \times X$$

$$Y \quad ::= \quad 0 \quad | \quad Y + Y \quad | \quad Y \times A \quad | \quad A \times Y$$

A sequent $X \vdash Y$ is ***bipointed***.

Definition 26. A linking $\lambda : A \vdash B$ is ***pointed*** if every resolution of B retains a link A — 1 in λ, and ***copointed*** if every co-resolution of A retains a link 0 — B. A linking that is both pointed and copointed is ***bipointed***.

Given a pointed formula X there is a ***canonical*** pointed net, which is biased to factor through the first projection of an object $X_1 \times X_2$ when both subformulae are pointed. Dually, a ***canonical*** copointed net is biased to factor through the first injection of any subformula $Y_1 + Y_2$ of the target formula.

Definition 27. A linking $\lambda : A \vdash B$ is ***full*** if λ contains every unit link in $A \vdash B$, but no axiom links (or other links).

The following proposition characterises the behaviour of pointed and copointed proof nets. The uniqueness of bipointed maps is captured simply by the saturated net being full.

Proposition 28 ([15, Lemma 4.2.3, Lemma 4.3.7])**.** *The saturation of the following nets is full:*

- *any net on a sequent $0 \vdash A$ or $A \vdash 1$;*
- *a net $\lambda : X \vdash Y$ that is copointed or pointed.*

To allow inductive reasoning on saturated nets we formalise when a linking factors through a projection or injection, or as a pair or copair. Definition 30 is chosen to be uniform for both proof nets and saturated nets. Given a linking $\lambda : A \vdash B$ and subformulae C, D of A, B respectively, let $\lambda|_{C \vdash D}$ be the sub-linking obtained by restricting λ to the sequent $C \vdash D$.

Definition 29. A linking $\lambda : A \vdash B$ is ***connected*** if it has a sub-linking $\phi \subseteq \lambda$ such that $\phi : A \vdash B$ is discrete.

Definition 30. A linking $\lambda : A \vdash B$ factors through:

- a ***projection*** π_i if $A = A_0 \times A_1$ and $\lambda|_{A_i \vdash B}$ is connected,
- an ***injection*** ι_i if $B = B_0 + B_1$ and $\lambda|_{A \vdash B_i}$ is connected,
- a ***pair*** if $B = B_0 \times B_1$ and both $\lambda|_{A \vdash B_i}$ are connected,
- a ***copair*** if $A = A_0 + A_1$ and both $\lambda|_{A_i \vdash B}$ are connected.

Proposition 31. *If a saturated net factors through a projection / injection / pair / copair, it is the saturation of a proof net that factors similarly.*

Proof. The case for pairs is immediate: a net into $A \times B$ always has an equivalent net that factors through a pair, obtained by replacing each link 0 — $A \times B$ by 0 — A and 0 — B. Dually, also the case for copairs is immediate.

For a saturated net that factors both through a pair and a projection, we split into a pair as above and reason by induction on the components. The case for a copair and injection is dual.

This leaves the case of a saturated net σ from $A \times B$ into $C + D$. If it factors through only one projection or injection, any net λ of which it is the saturation must factor similarly, since $\lambda \subseteq \sigma$. The remaining case, where Σ factors in multiple ways, is exactly Lemma 4.5.2 of [15]. □

The above gives us the prerequisites to define the *desaturation* procedure. It is non-deterministic, to allow for the fact that a saturated net may factor in multiple ways.

Definition 32. A ***desaturation*** of a linking $\lambda : A \vdash B$ is a net $d : A \vdash B$ obtained by the following procedure:

1) if λ is full, and A is copointed or B is pointed, $d : A \vdash B$ is the canonical copointed or pointed net;

otherwise, d is generated by one of the following steps:

2) if A and B are atomic, d is $\{A$ — $B\}$;
3) if $A = A_0 + A_1$ then $d = d_0 \cup d_1$, where d_i is a desaturation of $\lambda|_{A_i \vdash B}$ for $i = 0, 1$;
4) if $B = B_0 \times B_1$ then $d = d_0 \cup d_1$, where d_i is a desaturation of $\lambda|_{A \vdash B_i}$ for $i = 0, 1$;
5) if $A = A_0 \times A_1$ and λ factors through a projection π_i, then d is a desaturation of $\lambda|_{A_i \vdash B}$;

6) if $B = B_0 + B_1$ and λ factors through an injection ι_i, then d is a desaturation of $\lambda|_{A \vdash B_i}$.

The desaturation procedure is an inverse to saturation (modulo equivalence). This is expressed by the following theorem.

Theorem 33. *A saturated net $\sigma : A \vdash B$ has a desaturation, and any desaturation of it saturates to σ.*

Proof. Omitted. □

Two crucial properties of the desaturation procedure are that 1) it recurses only by restricting links to a sub-sequent, without deleting any, and 2) it relies only on how a linking factorises, whether it is full, and whether its source and target are (co)pointed. The point of 2) is that all three properties can be pre-computed in a single pass of the proof grid or the formula, and the point of 1) is that the pre-computed data remain correct when recursively finding a desaturation.

Definition 34. The ***factorisation grid*** for a linking $\lambda : A \vdash B$ is the linking obtained by exhaustive application of spawning (Definition 15) on λ.

Definition 35. The ***fullness grid*** for a linking $\lambda : A \vdash B$ contains the link C — D if and only if $\lambda|_{C \vdash D}$ is full.

As with provability search, by respecting the product order over the subformula relation both grids may be computed in a single pass. This brings us to the main result for ALLU.

Theorem 36. *Correctness of a saturated net $\sigma : A \vdash B$ is decidable in time $\mathcal{O}(|A| \times |B|)$.*

Proof. Desaturation gives a net d saturating to σ if and only if $\sigma : A \vdash B$ is correct: from left to right is immediate (if any net saturates to σ it is correct), the other direction is Theorem 33. Desaturation can be performed by a single simultaneous walk on the factorisation grid of σ, its fullness grid, and A and B annotated with copoint/point information. Generating both grids is in time $\mathcal{O}(|A| \times |B|)$, as is saturating the desaturation that is found (Proposition 23). □

V. FOALL

The main result of this section is that the provability problem for FOALL is NP-complete. Membership of NP is immediate by the size of proofs. We will show NP-hardness by a reduction from Boolean satisfyability (SAT; see [10]).

First, we will sketch how a Boolean formula A and an assignment γ for A, a function from the propositional atoms in A to truth values $\{\bot, \top\}$, may be interpreted in FOALL. The formula A is assumed to be in negation-normal form. We will encode a Boolean atom (propositional variable) p and its negation $\neg p$ by two distinct, unrelated atomic formulae, $P(\top)$ and $P(\bot)$, constructed by instantiating a predicate $P(x)$ with two distinct first-order constants $\top$ and $\bot$. Let the formula $\lfloor A \rfloor$ be the direct additive interpretation of A, as follows.

$$\lfloor p \rfloor = P(\top) \qquad \lfloor A \vee B \rfloor = \lfloor A \rfloor + \lfloor B \rfloor$$

$$\lfloor \neg p \rfloor = P(\bot) \qquad \lfloor A \wedge B \rfloor = \lfloor A \rfloor \times \lfloor B \rfloor$$

The interpretation $\lfloor \gamma \rfloor$ of the assignment γ is the product over each atomic formula, instantiated with its assigned value:

$$\lfloor \gamma \rfloor = \prod \{ P(x) \mid \gamma(p) = x \}$$

Provability of the sequent $\lfloor \gamma \rfloor \vdash \lfloor A \rfloor$ then encodes the evaluation of A under the assignment γ.

Proposition 37. *An assignment γ is satisfying for a Boolean formula A if and only if $\lfloor \gamma \rfloor \vdash \lfloor A \rfloor$ is provable in* FOALL.

Proof. For an atom p in A, the sequent $\lfloor \gamma \rfloor \vdash P(\top)$ is provable if and only if $\gamma(p) = \top$, and $\lfloor \gamma \rfloor \vdash P(\bot)$ is provable if and only if $\gamma(p) = \bot$. By induction on A it follows that $\lfloor \gamma \rfloor \vdash \lfloor A \rfloor$ is provable if and only if γ is satisfying for A. □

The next step is to encode the possibility of assigning mutually exclusive truth values to atomic formulae. The chosen encoding of Boolean atoms p and $\neg p$, as a predicate $P(x)$ over distinct constants $\bot$ and $\top$, means this can be expressed via the formula $\forall x.P(x)$, which quantifies over both truth values. For example, the Boolean formula $p \vee \neg p$ may be encoded as

$$\forall x.P(x) \ \vdash \ P(\top) + P(\bot)$$

However, in this naive formulation the interpretation of the contradiction $p \wedge \neg p$ would become:

$$\forall x.P(x) \ \vdash \ P(\top) \times P(\bot)$$

which is provable, by the following proof.

$$\dfrac{\dfrac{\dfrac{}{P(\top) \vdash P(\top)}}{\forall x.P(x) \vdash P(\top)} \qquad \dfrac{\dfrac{}{P(\bot) \vdash P(\bot)}}{\forall x.P(x) \vdash P(\bot)}}{\forall x.P(x) \vdash P(\top) \times P(\bot)}$$

The problem is that the product rule appears below the universal quantifier rule in the proof, which means the quantifier may be instantiated differently for both branches of the product. To remedy this, we introduce a "lock" construction that forces the universal quantifier to be instantiated first. The main lock mechanism consists of two existential quantifiers, inserted after the universal quantifier and before the product. Each binds the variable y in a special predicate $\mathsf{lock}(y)$, with occurrences in the antecedent and in the conclusion that must in a proof become linked by an axiom. Then the existential quantifier in the antecedent must be instantiated before that in the consequent, as both must take the same value. The interpretation of $p \wedge \neg p$ then becomes as follows.

$$\forall x \exists y.\, \mathsf{lock}(y) \times P(x) \ \vdash \ \exists y.\, \mathsf{lock}(y) \times P(\top) \times P(\bot)$$

The full construction is then the following.

Definition 38. Let $\llbracket A \rrbracket$ be the following sequent, for a Boolean formula A with atoms $p_1, \ldots, p_n$.

$$\forall x_1 \ldots \forall x_n \exists y.\, \mathsf{lock}(y) \times \prod_{1 \leq i \leq n} P_i(x_i) \ \vdash \ \exists y.\, \mathsf{lock}(y) \times \lfloor A \rfloor$$

Proposition 39. *The Boolean formula A is satisfiable if and only if $\llbracket A \rrbracket$ is provable in* FOALL.

Proof. From left to right, let γ be a satisfying assignment for A. A proof of $\llbracket A \rrbracket$ is constructed by first instantiating each universally quantified variable x_i with the constant $\gamma(a_i) \in \{\bot, \top\}$, so that the antecedent becomes the formula $\exists y.\, \mathsf{lock}(y) \times \lfloor \gamma \rfloor$. Next, there is the following derivation for the lock construction.

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{\mathsf{lock}(a) \vdash \mathsf{lock}(a)}}{\mathsf{lock}(a) \times \lfloor \gamma \rfloor \vdash \mathsf{lock}(a)} \qquad \dfrac{\begin{array}{c}\vdots \\ \lfloor \gamma \rfloor \vdash \lfloor A \rfloor\end{array}}{\mathsf{lock}(a) \times \lfloor \gamma \rfloor \vdash \lfloor A \rfloor}}{\mathsf{lock}(a) \times \lfloor \gamma \rfloor \vdash \quad \mathsf{lock}(a) \times \lfloor A \rfloor}}{\mathsf{lock}(a) \times \lfloor \gamma \rfloor \vdash \exists y.\, \mathsf{lock}(y) \times \lfloor A \rfloor}}{\exists y.\, \mathsf{lock}(y) \times \lfloor \gamma \rfloor \vdash \exists y.\, \mathsf{lock}(y) \times \lfloor A \rfloor}$$

The remaining sequent, $\lfloor \gamma \rfloor \vdash \lfloor A \rfloor$, is provable by Proposition 37, since γ is a satisfying assignment for A.

From right to left, let Π be a cut-free proof of $\llbracket A \rrbracket$. Since Π proves the consequent $\exists y.\, \mathsf{lock}(y) \times \lfloor A \rfloor$, it must contain an axiom $\mathsf{lock}(a) \vdash \mathsf{lock}(a)$. By construction, there is only one occurrence of the predicate lock in the antecedent of $\llbracket A \rrbracket$, so that Π must contain the following sub-derivation.

$$\dfrac{\dfrac{\begin{array}{c}\vdots \\ \mathsf{lock}(a) \times B \vdash \quad \mathsf{lock}(a) \times \lfloor A \rfloor\end{array}}{\mathsf{lock}(a) \times B \vdash \exists y.\, \mathsf{lock}(y) \times \lfloor A \rfloor}}{\exists y.\, \mathsf{lock}(y) \times B \vdash \exists y.\, \mathsf{lock}(y) \times \lfloor A \rfloor}$$

Here, $\mathsf{lock}(a) \times B$ is the (fully instantiated) propositional part of the antecedent of $\llbracket A \rrbracket$.

The side-condition on the left-introduction rule for the existential quantifier, that the eigenvariable a may not occur in the consequent $\exists y.\, \mathsf{lock}(y) \times \lfloor A \rfloor$, means that the two inferences above may not permute. The proof Π must then instantiate each universal quantification below the existential one, which means each is instantiated exactly once. Define an assignment γ for A as follows: $\gamma(a_i) = \top$ if x_i is instantiated to $\top$ in Π, and $\gamma(a_i) = \bot$ otherwise (note that this includes those x_i that are instantiated to anything other than $\bot$ or $\top$).

As Π is cut-free, each inference in its propositional part must either decompose $\lfloor A \rfloor$, or project onto a smaller fragment of the product B. By induction on this sub-proof it follows that γ is a satisfying assignment for A. □

Theorem 40. *Provability for* FOALL *is NP-complete.*

Proof. Since a proof for a sequent $A \vdash B$ has size no larger than $\mathcal{O}(|A| \times |B|)$, FOALL provability is in NP. Proposition 39 gives a polynomial-time reduction from the Boolean satisfiability problem, proving that FOALL provability is NP-hard. □

Acknowledgements

We thank the anonymous referees for their insightful and constructive remarks. This work was supported by EPSRC Project EP/K018868/1 *Efficient and Natural Proof Systems*.

References

[1] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *J. Symb. Log.*, 59(2):543–574, 1994.

[2] S. Abramsky and P.-A. Melliès. Concurrent games and full completeness. In *Proc. 14th Annual IEEE Symposium on Logic in Computer Science (LiCS'99)*, 1999.

[3] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *J. Logic Comput.*, 2(3):297–347, 1992.

[4] A. Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56:183–220, 1992.

[5] K. Chaudhuri, D. Miller, and A. Saurin. Canonical sequent proofs via multi-focusing. In *Fifth Ifip International Conference On Theoretical Computer Science*, pages 383–396, 2008.

[6] R. Cockett and L. Santocanale. On the word problem for $\Sigma\Pi$-categories, and the properties of two-way communication. In *Proc. Computer Science Logic (CSL'09)*, volume 5771 of LNCS, pages 194–208, 2009.

[7] R. Cockett and R. Seely. Finite sum-product logic. *Theory and Applications of Categories*, 8(5):63–99, 2001.

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

[9] P.J. De Naurois and V. Mogbil. Correctness of multiplicative additive proof structures is nl-complete. In *Proc. 23rd IEEE Symposium on Logic in Computer Science*, 2008.

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

[11] J.-Y. Girard. Linear logic. *Theor. Comput. Sci.*, 50(1):1–102, 1987.

[12] J.-Y. Girard. Proof-nets: the parallel syntax for proof-theory. *Logic and Algebra*, pages 97–124, 1996.

[13] S. Guerrini. A linear algorithm for mll proof net correctness and sequentialization. *Theor. Comput. Sci.*, 412(20):1958–1978, 2011.

[14] W. Heijltjes. Proof nets for additive linear logic with units. In *Proc. 26th Annual IEEE Symposium on Logic in Computer Science (LiCS'11)*, pages 207–216, 2011.

[15] W. Heijltjes. *Graphical representation of canonical proof: Two case studies*. PhD thesis, University of Edinburgh, 2012.

[16] W. Heijltjes and R. Houston. No proof nets for MLL with units: Proof equivalence in MLL is PSPACE-complete. In *CSL-LICS*, 2014.

[17] H. Hu. Contractible coherence spaces and maximal maps. *Elec. Notes in Theor. Comp. Sci.*, 20, 1999.

[18] D.J.D. Hughes. A canonical graphical syntax for non-empty finite products and sums. Technical report, Stanford University, 2002.

[19] D.J.D. Hughes. Proofs Without Syntax. *Annals of Mathematics*, 143:1065–1076, 2006.

[20] D.J.D. Hughes. Towards Hilbert's 24th Problem: Combinatorial Proof Invariants. In *Proc. WOLLiC'06*, volume 165 of *LNCS*, 2006.

[21] D.J.D. Hughes and R. van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. *ACM Transactions on Computational Logic*, 6(4), 2005.

[22] A. Joyal. Free bicomplete categories. *C.R. Math. Rep. Acad. Sci. Canada*, XVII(5):219–224, 1995.

[23] A. Joyal. Free lattices, communication and money games. *Proc. 10th Int. Cong. of Logic, Methodology and Philosophy of Science*, 1995.

[24] M.I. Kanovich. Horn programming in linear logic is NP-complete. In *Proc. 7th Annual IEEE Symposium on Logic in Computer Science (LiCS'92)*, 1992.

[25] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56:239–311, 1992.

[26] Roberto Maieli. Retractile proof nets of the purely multiplicative and additive fragment of linear logic. In *Proc. 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, 2007.

[27] S. Mimram. The structure of first-order causality. *Mathematical Structures in Computer Science*, 21(1):65–110, 2011.

[28] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, 1962.

[29] P. Wadler. Propositions as sessions. *ACM SIGPLAN Notices*, 47(9), 2012.

[30] P.M. Whitman. Free lattices. *Ann. Math.*, 42(1):325–330, 1941.