MULTIPLICATIVE-ADDITIVE PROOF EQUIVALENCE IS **Logspace**-COMPLETE, *via* BINARY DECISION TREES

MARC BAGNOL

Department of Mathematics and Statistics, University of Ottawa

Abstract. Given a logic presented in a sequent calculus, a natural question is that of equivalence of proofs: to determine whether two given proofs are equated by any denotational semantics, *i.e.* any categorical interpretation of the logic compatible with its cut-elimination procedure. This notion can usually be captured syntactically by a set of rule permutations.

Very generally, proofnets can be defined as combinatorial objects which provide canonical representatives of equivalence classes of proofs. In particular, the existence of proof nets for a logic provides a solution to the equivalence problem of this logic. In certain fragments of linear logic, it is possible to give a notion of proofnet with good computational properties, making it a suitable representation of proofs for studying the `cut`-elimination procedure, among other things.

It has recently been proved that there cannot be such a notion of proofnets for the multiplicative (with units) fragment of linear logic, due to the equivalence problem for this logic being **Pspace**-complete.

We investigate the multiplicative-additive (without unit) fragment of linear logic and show it is closely related to binary decision trees: we build a representation of proofs based on binary decision trees, reducing proof equivalence to decision tree equivalence, and give a converse encoding of binary decision trees as proofs. We get as our main result that the complexity of the proof equivalence problem of the studied fragment is **Logspace**-complete.

1. Introduction

Writing a proof in a formal system, or a program following the grammar of a programming language, one often finds different equivalent ways of describing the same object.

This phenomenon can be described at a semantic level: recall that a denotational semantics of a logic is a categorical interpretation that is invariant *modulo* its `cut`-elimination procedure. We want to say that two `cut`-free proofs are equivalent if they are interpreted as the same morphism in any denotational semantics.

Definition 1.1. *We say that two* `cut`*-free proofs in a logic* **L** *presented in sequent calculus are* equivalent *if they are interpreted by the same morphism in any denotational semantics of* **L**.

Consider now that the logic at hand is given as a sequent calculus. Then we can consider a more syntactic flavor of the same idea by looking at rewriting steps that yield equivalent proofs, *e.g.* by be permuting two rules. For example, in the following proof[1]

$$\dfrac{\dfrac{\dfrac{\langle\pi\rangle}{A, B \vdash D} \qquad \dfrac{\langle\mu\rangle}{A, C \vdash D}}{A, B \oplus C \vdash D}\,\oplus^\star}{B \oplus C \vdash A \multimap D}\,\multimap$$

the $\multimap$ rule can be lifted above the $\oplus^\star$ rule, yielding the equivalent

1998 ACM Subject Classification: F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic.

Key words and phrases: Linear logic, additive connectives, proofnets, proof equivalence, logarithmic space.

Extended and updated journal version of the author's TLCA'15 article [1], including an *erratum* to a wrong proof in it.

[1] For now, we do not bother with the exchange rule for ease of presentation, but this will have to change when we get into more technical considerations. See Remark 3.17

DOI:10.2168/LMCS-???

$$\dfrac{\dfrac{\begin{array}{c}\langle\pi\rangle\\ A,B\vdash D\end{array}}{B\vdash A\multimap D}\multimap\qquad\dfrac{\begin{array}{c}\langle\mu\rangle\\ A,C\vdash D\end{array}}{C\vdash A\multimap D}\multimap}{B\oplus C\vdash A\multimap D}\oplus^\star$$

In the Curry-Howard view on logic, proofs are seen as programs, the `cut`-rule corresponds to composition and the `cut`-elimination procedure to the evaluation mechanism. The existence of equivalent but formally different objects is an issue in that during `cut`-elimination, we may have to switch between equivalent objects to be able to actually perform a reduction step. Consider

$$\dfrac{\dfrac{\begin{array}{c}\langle\nu\rangle\\ \vdash B\end{array}}{\vdash B\oplus C}\oplus\mathtt{1}\qquad\dfrac{\dfrac{\begin{array}{c}\langle\pi\rangle\\ A,B\vdash D\end{array}\qquad\begin{array}{c}\langle\mu\rangle\\ A,C\vdash D\end{array}}{A,B\oplus C\vdash D}\oplus^\star}{B\oplus C\vdash A\multimap D}\multimap}{\vdash A\multimap D}\mathtt{cut}$$

In order to go on with `cut`-elimination, one option[2] is to apply the permutation we just saw to the $\oplus^\star$ and $\multimap$ rules to get:

$$\dfrac{\dfrac{\begin{array}{c}\langle\nu\rangle\\ \vdash B\end{array}}{\vdash B\oplus C}\oplus\mathtt{1}\qquad\dfrac{\dfrac{\begin{array}{c}\langle\pi\rangle\\ A,B\vdash D\end{array}}{B\vdash A\multimap D}\multimap\qquad\dfrac{\begin{array}{c}\langle\mu\rangle\\ A,C\vdash D\end{array}}{C\vdash A\multimap D}\multimap}{B\oplus C\vdash A\multimap D}\oplus^\star}{\vdash A\multimap D}\mathtt{cut}$$

and be able to perform an actual $\oplus/\oplus^\star$ elimination step. Note how this situation mirrors the permutation we took as an example just above. These steps where some reorganization of the proof but no actual elimination occurs are called *commutative conversions*. They make the study of the `cut`-elimination procedure much more intricate, as one has to work *modulo* equivalence, and this equivalence cannot be oriented: at some point we need to have the `cut` rule in scope and realize it commutes with itself, if we want the procedure to be confluent (indeed **MALL**$^-$`cut`-elimination is only confluent *modulo* permutations)

$$\dfrac{\begin{array}{c}\langle\nu\rangle\\ \vdash B\end{array}\qquad\dfrac{\begin{array}{c}\langle\mu\rangle\\ \vdash A\end{array}\qquad\begin{array}{c}\langle\pi\rangle\\ A,B\vdash C\end{array}}{B\vdash C}\mathtt{cut}}{\vdash C}\mathtt{cut}\quad\sim\quad\dfrac{\begin{array}{c}\langle\mu\rangle\\ \vdash A\end{array}\qquad\dfrac{\begin{array}{c}\langle\nu\rangle\\ \vdash B\end{array}\qquad\begin{array}{c}\langle\pi\rangle\\ A,B\vdash C\end{array}}{A\vdash C}\mathtt{cut}}{\vdash C}\mathtt{cut}$$

This is particularly problematic in view of fine-grained analysis of the `cut`-elimination procedure, *e.g.* in terms of complexity. In any case, we can define a decision problem following the discussion above, and later discuss its decidability, complexity *etc.*

Definition 1.2 (equivalence problem)**.** *Given a logic* **L** *presented in sequent calculus, we define the* equivalence problem *of* **L** *which we write* **Leq***, as the decision problem:*

"Given two **L** *proofs* π *and* ν*, are they equivalent?"*

We write $\pi\sim\nu$ *when two proofs are equivalent.*

1.1. **Proofnets and complexity of proof equivalence.** The theory of proofnets, introduced alongside linear logic [5], aims at tackling this issue by looking for a canonical representation of proofs, usually graphical objects, with the idea that when one manipulates canonical objects the commutative conversions automatically disappear. More technically: we look for a data structure offering canonical representatives of equivalence classes of proofs. The absolute minimal requirement is a translation function $\mathbf{t}(\cdot)$ from proofs to proofnets. One usually requires a section $\mathbf{s}(\cdot)$, from proofnets to proofs (with $\mathbf{t}\circ\mathbf{s}=\mathsf{Id}$) of $\mathbf{t}(\cdot)$ which is usually called *sequentialization*, and of course the ability to implement `cut`-elimination natively on the proofnet side. All of these together allows to use proofnets as proofs on their own accord, while having only $\mathbf{t}(\cdot)$ tells us about proof equivalence and nothing more.

In this article we will only look into the complexity of computing the $\mathbf{t}(\cdot)$ function and will refer to it simply as the complexity of the proofnets at hand. For the multiplicative without units fragment of linear

[2] The standard solution would rather be to lift the `cut` above the $\multimap$, but the end result would be the same.

logic **MLL⁻**, a satisfactory notion of proofnet can be defined: it indeed offers a representation of equivalence classes of proofs, while the translation from proofs to proofnets can be computed in logarithmic space.[3]

Contrastingly, the linear logic community has struggled to extend the notion of proofnets to wider fragments: even the case of **MLL** (that is, **MLL⁻** plus the multiplicative units) could not find a satisfactory answer. A recent result [7] sheds some light on this question. Since proofnets, when they exist, are canonical representatives of equivalence classes of proofs, they offer a way to solve the equivalence problem of the logic: simply translate the two proofs and check if the resulting proofnets are the same. The authors show that **MLLeq** is actually a **Pspace**-complete problem. Hence, there is no hope for a satisfactory notion of low-complexity proofnet for this fragment

In this article, we consider the same question, but in the case of **MALL⁻**: the multiplicative-additive without units fragment of linear logic. Indeed, this fragment has so far also resisted the attempts to build a notion of proofnet that at the same time characterizes proof equivalence and has basic operations of tractable complexity: we have either canonical nets of exponential size [9] or tractable nets that are not canonical [6]. D. Hughes and W. Heijltjes [8] recently argued that it is unlikely that one can devise a notion of proofnets for **MALL⁻** that is at the same time canonical and **Ptime**.

One might suspect that we have completeness for some untractable complexity class,[4] as is the case for **MLL**. An obvious candidate in that respect would be **coNP**: as we will see, one of the two approaches to proofnets for **MALL⁻** is related to Boolean formulas, which equivalence problem is widely known to be **coNP**-complete. Moreover, the `cut`-elimination problem (given two proofs with `cut`s, do they have the same normal form) of **MALL⁻** is already known to be **coNP**-complete [13].

It actually turns out that this is not the case as we will show that the equivalence problem of **MALL⁻** is **Logspace**-complete. However the connection between the problem of proofnets for this fragment and the notion of binary decision tree established in the course of the proof provides a beginning of clarification on the matter.

1.2. Binary Decision Trees.

The problem of the representation of Boolean functions is of central importance in circuit design and has a large range of practical applications. Over the years, binary decision diagrams [15] became the most widely used data structure to treat this question. We will actually use here the simpler cousins of binary decision diagrams which do not allow sharing of subtrees: binary decision trees (BDT).

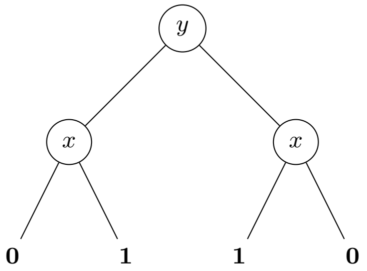
Let us set up a bit of notation and vocabulary concerning BDT.

Definition 1.3. *A binary decision tree (*BDT*) is a a binary tree with leaves labeled by* $\mathbf{0}$ *or* $\mathbf{1}$*, and internal vertices labeled by Boolean variables.*

We will only be manipulating free BDT*: no variable appears twice on a path from the root to a leave.*

A BDT represents a Boolean function in the obvious way: given a valuation of Boolean variables ($v : x_i \mapsto \mathbf{0}/\mathbf{1}$), go down from the root following the left/right branch of each internal vertex according to the assignment. When a leaf is reached one has the output of the function.

Example 1.4. *The following* BDT *represents the* $x\,\mathtt{XOR}\,y$ *function:*



[3] Given a proof and looking at its conclusion sequent, one only needs to compute when two atoms are linked by an axiom link. Which can be done by going from the root of the proof to an eventual leaf (axiom rule), potentially stopping at a ⊗ rule if the two atoms are sent to two different subproofs.

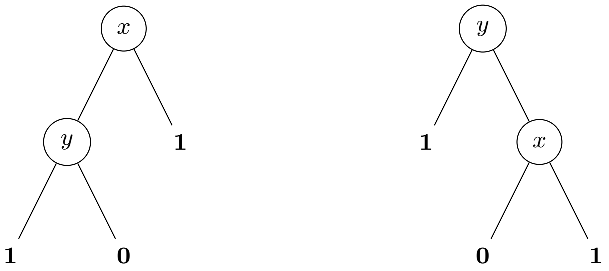
[4] The problem was first discovered to be decidable [3]. Later on the first notion of canonical proofnets for **MALL⁻** [9] implied it can be solved in exponential time and finally the **coNP** bound [13] on `cut`-elimination for the fragment, which subsumes the equivalence problem, was the best known to date.

Notation 1.5. *We use the notation* $x \rhd \phi \,[\!]\, \psi$ *to describe* BDT *inductively, with the meaning: the root is labeled by the variable* x*, with left subtree* ϕ *and right subtree* ψ*. Moreover we write* $\bar{\phi}$ *for the* negation *of* ϕ*: take* ϕ *and swap the* $\mathbf{0}/\mathbf{1}$ *at the leaves.*

Different trees can represent the same function, and we have an equivalence relation accounting for that.

Definition 1.6. *We say that two* BDT ϕ, ψ *are* equivalent *(notation* $\phi \sim \psi$*) if they represent the same Boolean function.*

Example 1.7. *The two following* BDT *represent the boolean function* x `OR` (`NOT` y) *and are therefore equivalent:*



As above, we can wonder about the complexity of the associated decision problem BDT**eq**. While the equivalence problem for Boolean formulas is **coNP**-complete in general and in many subcases including binary decision diagrams [15], we will see that the situation is different for BDT due to their simpler structure.

Outline of the paper

We begin in section 2 by covering some background material on **MALL**$^-$ and notions of proofnet for this fragment: monomial proofnets and the notion of slicing. Then, we introduce in section 3 an intermediary notion of proof representation, BDT slicings, that will help us to relate proofs in **MALL**$^-$ and BDD. In section 4, we establish further encodings and reductions.

The conference version [1] *of this work contained a mistake in a proof which we fix in this version by considering a variant of* **MALL**$^-$ *where the exchange rule is taken seriously into account. This is discussed further in Remark 3.17.*

2. Multiplicative-Additive Proof Equivalence

We will be interested in the multiplicative-additive without units part of linear logic, **MALL**$^-$, and more precisely its intuitionistic fragment. We choose to work primarily in this fragment for two reasons: first we want to avoid any impression that our constructions rely on the classical nature of **MALL**$^-$, second we believe that this makes the whole discussion more accessible outside the linear logic community. For reductions and encodings *to* **MALL**$^-$, this actually makes for stronger results than if they were formulated in the classical case. Conversely, when being limited the intuitionistic case would be restrictive (when deciding equivalence) we will explain how constructions can be extended to the classical case.

We consider formulas that are built inductively from atoms which we write $\alpha, \beta, \gamma, \dots$ and the binary connectives $\oplus$ and $\multimap$. We write formulas as $A, B, C, \dots$ unless we want to specify they are atoms and sequents as $\Gamma \vdash A$ with Γ a sequence of formulas. The $\oplus$ occurring on the of the left of the $\vdash$ symbol may be labeled by boolean variables ($\oplus_x$, we omit the label when not relevant) and we assume that different occurrences of the connective in a sequent carry a different label. We consider an η-expanded logic, which simplifies proofs and definitions. We do not include the `cut` rule in our study, since in a static situation (we are not looking at the `cut`-elimination procedure) it can always be encoded *w.l.o.g.* using the $\multimap^\star$ rule. Moreover it can be argued that proof equivalence with `cut` should include `cut`-elimination, which is known to be **coNP**-complete [13] as we already mentioned. Also we work with an explicit exchange rule, labeled by the positions i, j of the elements of the context it swaps:

$$\alpha \vdash \alpha \qquad \dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\multimap \qquad \dfrac{\Gamma \vdash A \qquad B, \Delta \vdash C}{\Gamma, A \multimap B, \Delta \vdash C}\multimap^\star \qquad \dfrac{\Gamma \vdash C}{\Gamma' \vdash C}\,\mathtt{ex}_{i,j}$$

$$\dfrac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\oplus\mathtt{l} \qquad \dfrac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}\oplus\mathtt{r} \qquad \dfrac{\Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma, A \oplus_x B \vdash C}\oplus_x^\star$$

Notation 2.1. *We will denote this proof system by $\lambda_\oplus^\multimap$ emphasizing that, even if we will not explicitly manipulate λ-terms, we are working with the Curry-Howard counterpart of a typed linear λ-calculus with sums.*

Moreover, we will use the notation $\overset{\langle\pi\rangle}{\Gamma \vdash A}$ *for "the proof π of conclusion $\Gamma \vdash A$".*

Remark 2.2. *Any time we will look at a $\lambda_\oplus^\multimap$ proof from a complexity perspective, we will consider they are represented as a tree with nodes labelled with a rule and the sequent that is the conclusion of that rule.*

We already discussed the idea of proof equivalence in general, its synthetic formulation based on denotational semantics and how it can be captured algorithmically as rule permutations in certain cases. We will not go through all the details specific to the **MALL⁻** case, as we already have an available equivalent characterization in terms of *slicing* in the literature [9, 10] which we review in subsection 2.2. Instead, in addition to the example we saw in the introduction let us rather focus on another significant case:

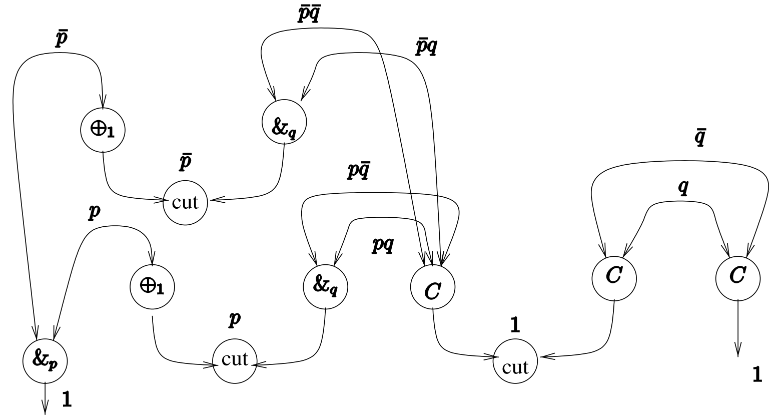
$$\dfrac{\overset{\langle\nu\rangle}{\vdash D} \qquad \dfrac{\overset{\langle\pi\rangle}{E, A \vdash C} \qquad \overset{\langle\mu\rangle}{E, B \vdash C}}{E, A \oplus_x B \vdash C}\oplus_x^\star}{D \multimap E, A \oplus_x B \vdash C}\multimap^\star \quad \sim \quad \dfrac{\dfrac{\overset{\langle\nu\rangle}{\vdash D} \qquad \overset{\langle\pi\rangle}{E, A \vdash C}}{D \multimap E, A \vdash C} \qquad \dfrac{\overset{\langle\nu\rangle}{\vdash D} \qquad \overset{\langle\mu\rangle}{E, B \vdash C}}{D \multimap E, B \vdash C}\multimap^\star}{D \multimap E, A \oplus_x B \vdash C}\oplus_x^\star$$

In this permutation, the $\multimap^\star$ rule gets lifted above the $\oplus_x^\star$ rule. But doing so, notice that we created two copies of ν instead of one, therefore the size of the prooftree has grown. Iterating on this observation, it is not hard to build pairs of proofs that are equivalent, but one of which is exponentially bigger than the other. This "distributivity phenomenon" [8] is indeed where the difficulty of proof equivalence in $\lambda_\oplus^\multimap$ (and hence **MALL⁻**) lies. As a matter of fact, this permutation of rules alone would be enough to build the encoding of binary decision trees by $\lambda_\oplus^\multimap$ proofs presented in section 4.

2.1. **Monomial proofnets.** The first attempt in the direction of a notion of proofnet for **MALL⁻** is due to J.-Y.Girard [6], followed by a version with a full `cut`-elimination procedure by O. Laurent and R. Maieli [12].

While proofnets for multiplicative linear logic without units were introduced along linear logic itself [5], extending the notion to the multiplicative-additive without units fragment proved to be a true challenge, mainly because of the *superposition* at work in the & rule ($\oplus^\star$ in the intuitionistic fragment).

Girard's idea was to represent the superposed "versions" of the proof by a graph with Boolean formulas (called a *weight*) to each edge, with one variable for each & connective in the conclusion Γ. To retrieve the version corresponding to some selection of the left/right branches of each &, one then just needs to evaluate the Boolean formulas with the corresponding valuation of their variables, keeping only the parts of the graph which weight evaluates to $\mathbf{1}$. Here is an example of monomial proofnet, from Laurent and Maieli's article:



This notion of proofnet fails to be a canonical representation of equivalence classes of proofs: for instance the rule permutation we just saw is not interpreted as an equality. Still, we will retain this idea of using Boolean formulas and put it to work in section 3: suitably regrouping and merging links, it appears that instead of a collection of scattered monomials we can use binary decision trees. This will lead us to the definition of BDT slicing.

2.2. **Slicings and proof equivalence.** The idea of slicing dates back to J.-Y. Girard's original article on proofnets for **MALL**$^-$ [6], and was even present in the original article on linear logic [5]. It has been really developed, addressing the many technical problems it poses by D. Hughes and R.van Glabbeek [9] to set up the only known notion of canonical proofnets for **MALL**$^-$. It amounts to the natural point of view already evoked just above, seeing the $\&$ rule as introducing superposed variants of the proof, which are eventually to be selected from in the course of `cut`-elimination.

But if we have two alternative *slices* for each $\&$ connective of a sequent Γ and all combinations of slices can be selected independently, we readily see that the global number of slices can be exponential in the number of $\&$ connectives in Γ. This is indeed the major drawback of the representation of proofs as a set of slices: the size of objects representing proofs may grow exponentially with the size of the original proofs, impairing the possibility of proofnets based on this idea to be low-complexity.

Let us give a variant of the definition in the case of the intuitionistic $\lambda_\oplus^\multimap$:

Definition 2.3 (slicing)**.** *Given a $\lambda_\oplus^\multimap$ sequent $\Gamma \vdash A$, a* slice *of $\Gamma \vdash A$ is a set of (unordered) pairs of occurrences of atoms in $\Gamma \vdash A$. Then, a* slicing *of $\Gamma \vdash A$ is a finite set of slices of $\Gamma \vdash A$, and to any $\lambda_\oplus^\multimap$ proof π, we associate a slicing $\mathcal{S}_\pi$ by induction:*

- *If $\pi = \alpha \vdash \alpha$ then $\mathcal{S}_\pi$ is the set containing only the linking $\{[\alpha, \alpha]\}$*
- *If* $\pi = \dfrac{\overset{\langle\mu\rangle}{\Gamma, A \vdash B}}{\Gamma \vdash A \multimap B}\multimap$ *then $\mathcal{S}_\pi = \mathcal{S}_\mu$ (seeing atoms of $A \multimap B$ as the corresponding ones of A and B)*
- *If* $\pi = \dfrac{\overset{\langle\mu\rangle}{\Gamma \vdash A} \qquad \overset{\langle\nu\rangle}{\Delta, B \vdash C}}{\Gamma, \Delta, A \multimap B \vdash C}\multimap^\star$ *then $\mathcal{S}_\pi = \{\, \lambda \cup \lambda' \mid \lambda \in \mathcal{S}_\mu \,,\ \lambda' \in \mathcal{S}_\nu \,\}$*
- *If* $\pi = \dfrac{\overset{\langle\mu\rangle}{\Gamma \vdash A}}{\Gamma' \vdash A}\mathtt{ex}_{i,j}$ *then $\mathcal{S}_\pi = \mathcal{S}_\mu$ (accordingly reindexed)*
- *If* $\pi = \dfrac{\overset{\langle\mu\rangle}{\Gamma \vdash A}}{\Gamma \vdash A \oplus B}\oplus\mathtt{l}$ *then $\mathcal{S}_\pi = \mathcal{S}_\mu$, and likewise for $\oplus\mathtt{r}$*
- *If* $\pi = \dfrac{\overset{\langle\mu\rangle}{\Gamma, A \vdash C} \qquad \overset{\langle\nu\rangle}{\Gamma, B \vdash C}}{\Gamma, \Delta, A \oplus B \vdash C}\oplus^\star$ *then $\mathcal{S}_\pi = \mathcal{S}_\mu \cup \mathcal{S}_\nu$*

Example 2.4. *The following proof π (we index occurences of atoms to differentiate them)*

$$\dfrac{\dfrac{\alpha_l \vdash \alpha_r}{\alpha_l \vdash \alpha_r \oplus \beta_r}\oplus\mathtt{l} \qquad \dfrac{\beta_l \vdash \beta_r}{\beta_l \vdash \alpha_r \oplus \beta_r}\oplus\mathtt{r}}{\alpha_l \oplus \beta_l \vdash \alpha_r \oplus \beta_r}\oplus^\star$$

gets the (two-slices) slicing $\mathcal{S}_\pi = \big\{ \{\, [\alpha_l, \alpha_r] \,\}, \{\, [\beta_l, \beta_r] \,\} \big\}$.

Remark 2.5. *In the $\multimap^\star$ rule, it is clear that the number of slices is multiplied. This is just what is needed in order to have a combinatorial explosion: pick any proof interpreted by two (or more) slices, and combine n copies of this proof to get a proof that has linear size in n, but is interpreted with 2^n slices.*

The main result we need from the work of Hughes and van Glabbeek is their canonicity result [9, 10] formulated in the case of **MALL**$^-$, that specializes readily to the intuitionistic fragment $\lambda_\oplus^\multimap$.

Theorem 2.6 (slicing equivalence)**.** Let π and ν be two $\lambda_{\oplus}^{\multimap}$ proofs. We have that π and ν are equivalent if and only if $\mathcal{S}_\pi = \mathcal{S}_\nu$.

3. BDT slicings

We now introduce an intermediate notion of representation of proofs which will be a central tool in the rest of the article. In a sense, it is a synthesis of monomial proofnets and slicings: acknowledging the fact that slicing makes the size of the representation explode, we rely on BDT to keep things more compact. Of course, the canonicity property is lost, but this is exactly the point! Indeed, deciding whether two "BDT slicings" are equivalent is the reformulation of proof equivalence we rely on in the next sections.

The basic idea is, considering a sequent $\Gamma \vdash A$, to use the boolean variable associated to $\oplus$ occurrences in Γ and associate a BDT to each pair of (occurrences of) atoms in $\Gamma \vdash A$, indicating the presence of an axiom rule linking the two, depending on which branch of the $\oplus^\star$ rules we are sitting in.

For example in the proof

$$\dfrac{\dfrac{\alpha \vdash \alpha}{\alpha \vdash \alpha \oplus \beta}\,\oplus\mathtt{l} \qquad \dfrac{\beta \vdash \beta}{\beta \vdash \alpha \oplus \beta}\,\oplus\mathtt{r}}{\alpha \oplus_x \beta \vdash \alpha \oplus \beta}\,\oplus_x^\star$$

we have a $\beta \vdash \beta$ axiom only when we are selecting the right branch of the $\oplus^\star$ rule. So the BDT $x \rhd \mathbf{0} \mathbin{[\!]} \mathbf{1}$ should be associated to the pair $[\beta, \beta]$ (here we use Notation 1.5), and conversely $[\alpha, \alpha]$ gets $x \rhd \mathbf{1} \mathbin{[\!]} \mathbf{0}$.

We can lift this idea to a complete inductive definition:

Definition 3.1 (BDT slicing)**.** *Given a $\lambda_{\oplus}^{\multimap}$ sequent $\Gamma \vdash A$, a* BDT slicing *of $\Gamma \vdash A$ is a function $\mathcal{B}$ that associates a* BDT *to every (unordered) pair $[\alpha, \beta]$ of distinct occurrences of atoms of $\Gamma \vdash A$.*

To any $\lambda_{\oplus}^{\multimap}$ proof π, we associate a BDT *slicing $\mathcal{B}_\pi$ by induction on the tree structure of π:*

- *If* $\pi = \alpha \vdash \alpha$ *then* $\mathcal{B}_\pi[\alpha, \alpha] = \mathbf{1}$
- *If* $\pi = \dfrac{\overset{\langle\mu\rangle}{\Gamma, A \vdash B}}{\Gamma \vdash A \multimap B}\,\multimap$ *then* $\mathcal{B}_\pi[\alpha, \beta] = \mathcal{B}_\mu[\alpha, \beta]$ *(seeing atoms of $A \multimap B$ as the corresponding atoms of A and B)*
- *If* $\pi = \dfrac{\overset{\langle\mu\rangle}{\Gamma \vdash A} \qquad \overset{\langle\nu\rangle}{\Delta, B \vdash C}}{\Gamma, \Delta, A \multimap B \vdash C}\,\multimap^\star$ *then* $\mathcal{B}_\pi[\alpha, \beta] = \begin{cases} \mathcal{B}_\mu[\alpha, \beta] & \textit{if } \alpha, \beta \textit{ are atoms of } \Gamma \vdash A \\ \mathcal{B}_\nu[\alpha, \beta] & \textit{if } \alpha, \beta \textit{ are atoms of } \Delta, B \vdash C \\ \mathbf{0} & \textit{otherwise} \end{cases}$
- *If* $\pi = \dfrac{\overset{\langle\mu\rangle}{\Gamma \vdash A}}{\Gamma' \vdash A}\,\mathtt{ex}$ *then* $\mathcal{B}_\pi[\alpha, \beta] = \mathcal{B}_\mu[\alpha, \beta]$ *(accordingly reindexed)*
- *If* $\pi = \dfrac{\overset{\langle\mu\rangle}{\Gamma \vdash A}}{\Gamma \vdash A \oplus B}\,\oplus\mathtt{l}$ *then* $\mathcal{B}_\pi[\alpha, \beta] = \begin{cases} \mathcal{B}_\mu[\alpha, \beta] & \textit{if } \alpha, \beta \textit{ are atoms of } \Gamma \vdash A \\ \mathbf{0} & \textit{otherwise} \end{cases}$ *(and likewise for $\oplus\mathtt{r}$)*
- *If* $\pi = \dfrac{\overset{\langle\mu\rangle}{\Gamma, A \vdash C} \qquad \overset{\langle\nu\rangle}{\Gamma, B \vdash C}}{\Gamma, A \oplus_x B \vdash C}\,\oplus_x^\star$ *then* $\mathcal{B}_\pi[\alpha, \beta] =$ $\begin{cases} x \rhd \mathcal{B}_\mu[\alpha, \beta] \mathbin{[\!]} \mathbf{0} & \textit{if } \alpha \textit{ or } \beta \textit{ is an atom of } A \\ x \rhd \mathbf{0} \mathbin{[\!]} \mathcal{B}_\nu[\alpha, \beta] & \textit{if } \alpha \textit{ or } \beta \textit{ is an atom of } B \\ x \rhd \mathcal{B}_\mu[\alpha, \beta] \mathbin{[\!]} \mathcal{B}_\nu[\alpha, \beta] & \textit{if } \alpha \textit{ or } \beta \textit{ are both atoms of } \Gamma, C \\ \mathbf{0} & \textit{otherwise} \end{cases}$

All cases except $\oplus^\star$ are just ensuring proper branching of subformulas, while the $\oplus^\star$ case implements the idea behind the example before the definition: the left branch of the rule corresponds to the left branch of the BDT.

The equivalence of BDT slicings is straightforward to define:

Definition 3.2. *We say that two* BDT *slicings* $\mathcal{M},\mathcal{N}$ *of the same* $\Gamma\vdash A$ *are* equivalent *(notation* $\mathcal{M}\sim\mathcal{N}$*) if for any pair* $[\alpha,\beta]$ *we have* $\mathcal{M}[\alpha,\beta]\sim\mathcal{N}[\alpha,\beta]$ *in the sense of Definition 1.6.*

The usefulness of the notion of BDT slicing comes from the fact that it captures exactly proof equivalence, reducing it to BDT equivalence.

Theorem 3.3 (proof equivalence)**.** Let π and ν be two $\lambda_{\oplus}^{\multimap}$ proofs, then $\pi\sim\nu$ if and only if $\mathcal{B}_\pi\sim\mathcal{B}_\nu$.

Proof. We will rely on the characterization of proof equivalence by slicing, *i.e.* Theorem 2.6 and show that $\mathcal{S}_\pi=\mathcal{S}_\nu$ if and only if $\mathcal{B}_\pi\sim\mathcal{B}_\nu$.

To a BDT slicing $\mathcal{B}$, we can associate a slice $v(\mathcal{B})$ for each valuation v of the variables occurring in $\mathcal{B}$ by setting $v(\mathcal{B})=\{\,[\alpha,\beta]\mid v(\mathcal{B}[\alpha,\beta])=\mathbf{1}\,\}$ (where $v(B[\alpha,\beta])$ denotes the result of evaluating a BDT against a valuation of variables) and then a slicing $S_{\mathcal{B}}=\{\,v(\mathcal{B})\mid v\text{ valuation}\,\}$. Note that in the process different valuations yielding identical slices might be identified. By definition, it is clear that if $\mathcal{B}$ and $\mathcal{B}'$ involve the same variables and $\mathcal{B}\sim\mathcal{B}'$ then $S_{\mathcal{B}}=S_{\mathcal{B}'}$.

It is straightforward to see that $S_{\mathcal{B}_\pi}=\mathcal{S}_\pi$ so we have that $\mathcal{B}_\pi\sim\mathcal{B}_\nu$ implies $\mathcal{S}_\pi=\mathcal{S}_\nu$

Conversely, suppose $\mathcal{B}_\pi\not\sim\mathcal{B}_\nu$, so that there is a v such that $v(\mathcal{B}_\pi)\neq v(\mathcal{B}_\nu)$. To conclude that $S_{\mathcal{B}_\pi}\neq S_{\mathcal{B}_\nu}$, we need the following lemmas:

Lemma 3.4. *Let* π *be a* $\lambda_{\oplus}^{\multimap}$ *proof of* $A_1,\dots,A_n\vdash A_0$ *and* v *a valuation of the variables of* π *then for any* i*,* $v(\mathcal{B}_\pi)$ *contains at least one pair with an atom in* A_i*.*

Lemma 3.5. *Let* π *be a* $\lambda_{\oplus}^{\multimap}$ *proof of* $\Gamma\vdash C$ *and* x *a label of some* $\oplus_x^\star$ *rule introducing the subformula* $A\oplus_x B$ *and* v *a valuation of the variables of* π *mapping* x *to* $\mathbf{0}$ *then:*

- *The pairs in* $v(\mathcal{B}_\pi)$ *contain no atom of* B*.*
- *At least one pair in* $v(\mathcal{B}_\pi)$ *contains an atom of* A*.*

(and conversely if v *maps* x *to* $\mathbf{1}$*)*

Proof of the lemmas. The first one follows from a straightforward induction on π.
The second is also by induction on π: going through the cases of Definition 3.1 we see any rule that does not introduce $A\oplus_x B$ itself but has it as a subformula will preserve these two properties. In the case of a $\oplus_x^\star$ rule branching two proofs μ and ν, by definition any pair with atoms of B will evaluate to $\mathbf{0}$ if x does, and hence cannot be part of $v(\mathcal{B}_\pi)$; conversely, the first lemma provides us with a pair containing an atom of A in $v(\mathcal{B}_\mu)$ which will still be present in $v(\mathcal{B}_\pi)$. □

A consequence of the second lemma is that for any $\lambda_{\oplus}^{\multimap}$ proofs π and ν with the same conclusion, and two *different* valuations v,w of their variables we have $v(\mathcal{B}_\pi)\neq w(\mathcal{B}_\nu)$: just apply the lemma with x a variable on which v,w differ. This means we have $v(\mathcal{B}_\pi)\neq v(\mathcal{B}_\nu)$ and also if $v\neq w$, $v(\mathcal{B}_\pi)\neq w(\mathcal{B}_\nu)$, so that $v(\mathcal{B}_\pi)\notin S_{\mathcal{B}_\nu}$ and therefore $S_{\mathcal{B}_\pi}\neq S_{\mathcal{B}_\nu}$, that is $\mathcal{S}_\pi\neq\mathcal{S}_\nu$. □

This theorem will allow us to decide proof equivalence by reducing it to BDT equivalence. Therefore let us have a look at the complexity of computing BDT slicings.

Proposition 3.6. *For any* $\lambda_{\oplus}^{\multimap}$ *proof* π *and any pair* $[\alpha,\beta]$*, we can compute* $\mathcal{B}_\pi[\alpha,\beta]$ *in* **Logspace***.*

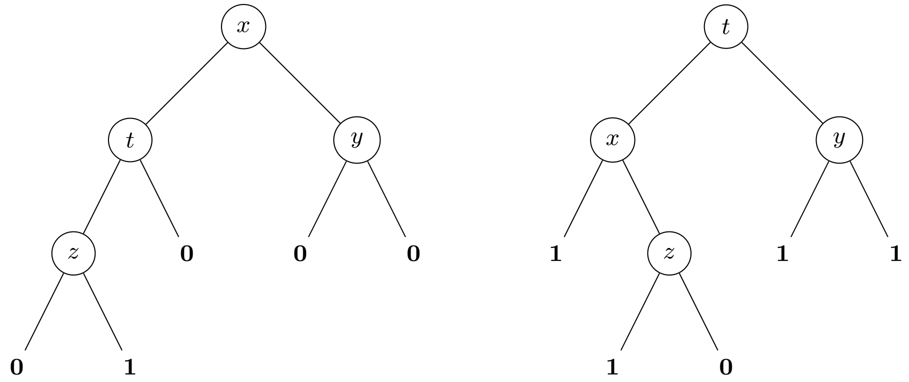
Proof. To build $\mathcal{B}_\pi[\alpha,\beta]$ we only have to go through the prooftree and apply Definition 3.1, creating new x vertices when visiting $\oplus_x^\star$ rules, passing through $\multimap$ and $\oplus$ rules, following only the relevant branch of $\multimap^\star$ rules *etc.*

At any step we do not need to remember any other information than a pointer to our position in the prooftree, which requires only logarithmic space. □

3.1. **Proof equivalence is Logspace-complete.** Thanks to the results we just established, we will be able to decide $\lambda_{\oplus}^{\multimap}$ proof equivalence in logarithmic space. Since Proposition 3.6 basically reduces proof equivalence to BDT equivalence in logarithmic space, let us focus on BDT equivalence for a moment.

Definition 3.7 (compatible leaves)**.** *Consider two leaves* l,m *of two* BDT ϕ,ψ *and the two paths* p,q *from the leaves to the root of their respective trees. We say these leaves are* compatible *if there is no variable that appears both in* p *and* q *while being reached from opposite left/right branches.*

A small example to illustrate the idea: in the following trees ϕ (left) and ψ (right)



the only **1** leaf of ϕ is compatible with the leftmost **1** leaf of ψ, but is not compatible with its only **0** leaf.

Lemma 3.8. *Two* BDT ϕ *and* ψ *are not equivalent if and only if there is a* **1** *leaf of* ϕ *and a* **0** *leaf of* ψ *that are compatible, or conversely.*

Proof. If we have compatible **0** and **1** leaves, we can build a number of valuations of the variables that will lead to these leaves when evaluating ϕ and ψ, making them not equivalent.

Conversely, if ϕ and ψ are not equivalent, then we have a valuation that have them evaluate differently. Both of these evaluations are reached by following a path from the roots to leaves with different values, and these two leaves must be compatible. □

With this lemma we can devise a decision procedure in logarithmic space for BDT**eq**.

Proposition 3.9. *The* BDT *equivalence problem is in* **Logspace**.

Proof. With the above lemma, we see that given two BDT ϕ, ψ all we have to do is look for compatible leaves with different values. To do that we can go through all the pairs of one leaf of ϕ and one leaf of ψ (two pointers) and then through all pairs of variables on the paths between the leaves and the root of their respective trees (again, two pointers), checking if the two leaves hold different values while being compatible. This needs four pointers of logarithmic size, so we can decide BDT**eq** in **Logspace**. □

Proposition 3.10. *The equivalence problem* $\lambda_{\oplus}^{\multimap}$**eq** *is in* **Logspace**.

Proof. Combine Proposition 3.6 and Proposition 3.9, relying on the fact that **Logspace** algorithms can be composed without stepping out of **Logspace**: given two $\lambda_{\oplus}^{\multimap}$ proofs π, μ with the same conclusion $\Gamma \vdash A$, go through all the pairs of atoms $[\alpha, \beta]$ of $\Gamma \vdash A$, compute the two associated BDT $\mathcal{B}_{\pi}[\alpha, \beta], \mathcal{B}_{\mu}[\alpha, \beta]$ and test them for equivalence. □

To prove the hardness direction of the completeness result, we rely on a standard **Logspace**-complete problem and reduce it to $\lambda_{\oplus}^{\multimap}$**eq**.

Definition 3.11 (**ORD**)**.** Order between vertices *(***ORD***) is the following decision problem:*

"Given a directed graph $G = (V, E)$ that is a line[5] *and two vertices $f, s \in V$*
do we have $f < s$ in the total order induced by G?"

Theorem 3.12. **ORD** is **Logspace**-complete [4].

Remark 3.13. *Since we want to prove* **Logspace**-*completeness results, the notion of reduction in what follows needs to be in a smaller complexity class than* **Logspace** *itself (indeed* any *problem in* **Logspace** *is complete under* **Logspace** *reductions).*

[5] We use the standard definition of graph as a pair (V, E) of sets of vertices and edges (oriented couples of vertices $x \to y$). A graph is a *line* if it is connected and all the vertices have in-degree and out-degree 1, except the *begin* vertex which has in-degree 0 and out-degree 1 and the *exit* vertex which has in-degree 1 and out-degree 0. A line induces a total order on vertices through its transitive closure

One among many standard notions is (uniform) **AC₀** *reduction* [2]*, formally defined in terms of uniform circuits of fixed depth and unbounded fan-in. We will not be getting into the details about this complexity class and, as we will consider only graph transformations, we will rely on the following intuitive principle: if a graph transformation locally replaces each vertex by a bounded number of vertices and the replacement depends* only *on the vertex considered and eventually its direct neighbors, then the transformation is in* **AC₀***. Typical examples of such a transformation are certain simple cases of so-called "gadget" reductions used in complexity theory to prove hardness results.*

Lemma 3.14. **ORD** *reduces to* $\lambda^{\multimap}_{\oplus}$**eq** *in* **AC₀***.*

Proof. We are going to build a local graph transformation between the two problems.

First, we assume *w.l.o.g.* that the begin b vertex of the line G is different from f and s. Consider the following proof (where the indexes are just used to identify occurences of the same atom a) that we call π_0:

$$\dfrac{\dfrac{\dfrac{a_1 \vdash a_1 \qquad a_2 \vdash a_2}{a_1, a_1 \multimap a_2 \vdash a_2}\,{\multimap^\star} \qquad a_3 \vdash a_3}{a_1, a_1 \multimap a_2, a_2 \multimap a_3 \vdash a_3}\,{\multimap^\star} \qquad a_4 \vdash a_4}{a_1, a_1 \multimap a_2, a_2 \multimap a_3, a_3 \multimap a_4 \vdash a_4}\,{\multimap^\star}$$

Then encoding relies on the exchange rule: the begin vertex b is replaced by the proof π_0, the f vertex is replaced by the $\mathtt{ex}_{2,3}$ rule with conclusion $a, a \multimap a, a \multimap a, a \multimap a \vdash a$, the s vertex is replaced by the $\mathtt{ex}_{3,4}$ rule $a, a \multimap a, a \multimap a, a \multimap a \vdash a$ and all the other vertices are replaced by a sequence of two identical $\mathtt{ex}_{1,2}$ rules (so the end result is doing nothing) again with conclusion $a, a \multimap a, a \multimap a, a \multimap a \vdash a$. Edges are kept as they were. Note that the conclusions of each "gadget" does not depend on the rest of the tree (as is the case in general) which would be problematic with respect to **AC₀** complexity.

Because the permutations corresponding to $\mathtt{ex}_{2,3}$ and $\mathtt{ex}_{3,4}$ do not commute, asking if f comes before s amounts to asking if the resulting proof is equivalent to

$$\dfrac{\dfrac{\begin{array}{c}\langle \pi_0 \rangle \\ a_1, a_1 \multimap a_2, a_2 \multimap a_3, a_3 \multimap a_4 \vdash a_4\end{array}}{a_1, a_2 \multimap a_3, a_1 \multimap a_2, a_3 \multimap a_4 \vdash a_4}\,\mathtt{ex}_{2,3}}{a_1, a_2 \multimap a_3, a_3 \multimap a_4, a_1 \multimap a_2 \vdash a_4}\,\mathtt{ex}_{3,4}$$

□

So in the end, we get:

Theorem 3.15 (**Logspace**-completeness)**.** The problem $\lambda^{\multimap}_{\oplus}$**eq** is **Logspace**-complete.

Remark 3.16. *Since the reduction we use for the hardness part uses only the rules* $\multimap^\star$ *and exchange, any subsystem of intuitionistic linear logic containing these rules will be have a* **Logspace***-hard equivalence problem. If they are moreover subsystems of* $\lambda^{\multimap}_{\oplus}$*, the problem will be* **Logspace***-complete. For instance intuitionistic multiplicative linear logic (without units) has a* **Logspace***-complete equivalence problem.*

Remark 3.17. *The choice of including an explicit exchange rule needs to be commented here. In the the conference version of this article* [1] *we tried to work* modulo *exchange to simplify proofs and definitions. But when dealing with such low complexities as* **Logspace** *and* **AC₀** *reductions, this is not something we can afford: as we just saw, the exchange rule entails* **Logspace***-hardness of equivalence, so that working* modulo *this rule amounts to work* modulo *a problem which is hard for the complexity class we are looking at.*

This is at the root of a mistake in the conference version of this work: deprived of explicit exchange rule, we were forced to try to prove hardness via *a reduction of* BDT *equivalence, which does not seem to be doable in* **AC₀** *after all. More generally, this shows that when dealing with such low complexities the details of how proofs are implemented can be fully relevant.*

3.2. **Classical case.** Exploring further the relation between BDT and $\lambda_{\oplus}^{\multimap}$ proofs, let us have a look at extending the above constructions and results to the classical case.

In **MALL**$^{-}$ we have one sided sequents $\vdash \Gamma$ (we even drop the $\vdash$ which no longer serve any purpose) of formulas built from atoms and *duals* of atoms $\alpha^\star, \beta^\star, \gamma^\star, \ldots$ and the connectives $\oplus, \&, \otimes, \parr$. The $\&$ connectives are now holding the labels. Here are the rules of **MALL**$^{-}$:

$$\alpha^\star, \alpha \qquad \frac{\Gamma, A, B}{\Gamma, A \parr B}\,\parr \qquad \frac{\Gamma, A \qquad \Delta, B}{\Gamma, \Delta, A \otimes B}\,\otimes \qquad \frac{\Gamma}{\Gamma'}\,\mathtt{ex}_{i,j} \qquad \frac{\Gamma, A}{\Gamma, A \oplus B}\,\oplus\mathtt{l} \qquad \frac{\Gamma, B}{\Gamma, A \oplus B}\,\oplus\mathtt{r} \qquad \frac{\Gamma, A \qquad \Gamma, B}{\Gamma, A \&_x B}\,\&_x$$

Notice how close this is to what we had before. We are indeed looking at nothing more than a symmetrized version of $\lambda_{\oplus}^{\multimap}$, with a one-to-one correspondence between rules: $\parr$ matches $\multimap$, $\otimes$ matches $\multimap^\star$ *etc.* this makes the extension of previous result to **MALL**$^{-}$ extremely straightforward.

Logspace-hardness is immediate since $\lambda_{\oplus}^{\multimap}$ is a subsystem of **MALL**$^{-}$.

Moreover, the slicing and proof equivalence result of subsection 2.2 were originally formulated for **MALL**$^{-}$ [9] and we did nothing but adapt them to $\lambda_{\oplus}^{\multimap}$. The notion of BDT slicing never relies on the fact that we have a separation $\Gamma \vdash A$ so extending this to **MALL**$^{-}$ is just a matter of giving the same interpretation to corresponding rules. It follows that we can still reduce **MALL**$^{-}$**eq** to BDT**eq** in logarithmic space, and therefore that **MALL**$^{-}$**eq** is in **Logspace**.

Theorem 3.18. The equivalence problem **MALL**$^{-}$**eq** is **Logspace**-complete.

4. Reductions

With the development of previous sections, we were able to show that $\lambda_{\oplus}^{\multimap}$**eq** is **Logspace**-complete. This relied on the use of BDT, which equivalence problem was show to be in **Logspace**. In this section we complete the picture by first showing that BDT equivalence is **Logspace**-hard and setting up a direct reduction from BDT to $\lambda_{\oplus}^{\multimap}$ proofs.

We start by showing that the BDT equivalence problem is **Logspace**-hard, by a similar argument to that of the proof of Lemma 4.1. Then we will show we have a converse transformation to BDT slicing: we can encode any BDT into a $\lambda_{\oplus}^{\multimap}$ proof in logarithmic space, transporting BDT**eq** to $\lambda_{\oplus}^{\multimap}$**eq**. This translation is more expansive than the one we would get from completeness results, but it preserves the tree structure it manipulates.

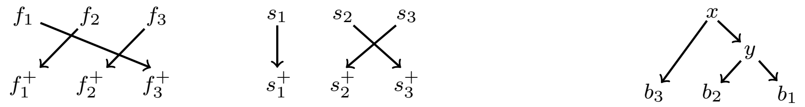
4.1. BDT**eq** is **Logspace**-complete.

Let us now show that the equivalence problem of BDT is **Logspace**-hard (and hence complete), again by reducing **ORD** (Definition 3.11) to it.

Lemma 4.1. **ORD** *reduces to* BDT**eq** *in* **AC**$_0$.

Proof. (very similar to the proof of Lemma 4.1) First, we assume *w.l.o.g.* that the begin b and the exit e vertices of the line G are different from f and s. We write f^+ and s^+ the vertices immediately after f and s in G.

Then, we perform a first transformation by replacing the graph with three copies of itself (this can be done by locally scanning the graph and create labeled copies of the vertices and edges). We write x_i to refer to the copy of the vertex x in the graph i. The second transformation is a rewiring of the graph as follows: erase the edges going out of the f_i and s_i and replace them as pictured in the two first subgraphs:

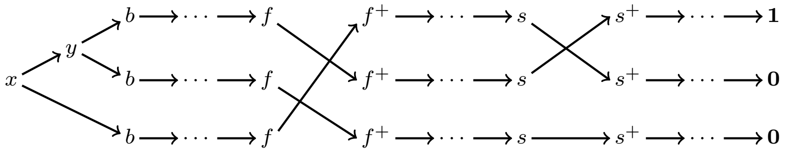


Let us call G_r the rewired graph and G_n the non-rewired graph. To each of them we add two binary vertices x and y connected to the begin vertices b_i as pictured in the third graph above.

Then we can produce two corresponding BDT ϕ_r and ϕ_n by replacing the exit vertices e_1, e_2, e_3 by $\mathbf{1}$, $\mathbf{0}$, $\mathbf{0}$ respectively, and each complete each unary (those with an out-degree 1) vertex with a $\rightarrow\mathbf{1}$ second branch, making each vertex binary.

It is then easy to see that if $f < s$ in the order induced by G if and only if ϕ_r and ϕ_n are equivalent.

Let us illustrate graphically what happens in the case where indeed $f < s$: we draw the resulting BDT as a labeled graph (with the convention that the we do not picture the extra $\to \mathbf{1}$ on unary vertices)



□

As a consequence of the lemma and the results of subsection 3.1 we get:

Theorem 4.2. The equivalence problems of BDT (and hence of BDT slicings) is **Logspace**-complete.

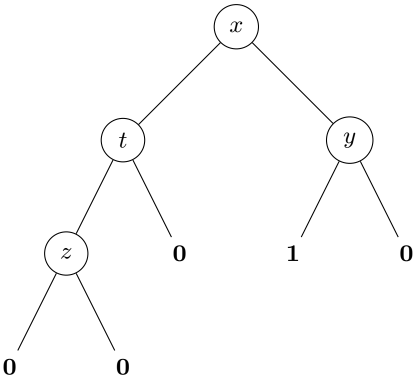
4.2. From BDT to $\lambda_{\oplus}^{\multimap}$ proofs (in Logspace).

We know now that both BDT**eq** and $\lambda_{\oplus}^{\multimap}$**eq** are **Logspace**-complete. Therefore there are reductions in both directions between these two problems. However, coming from completeness theory these reductions would not preserve the structure of the trees they manipulate.

The basic idea here is to use the $\oplus_x^\star$ rule to encode the $x \triangleright \cdot \,[\!]\, \cdot$ vertices. We will have a formula $\mathbf{B}$ which will receive the encoded BDT in the BDT slicing interpretation of the proof, and an atomic formula α_i for each variable. To be able to mix the order in which the variables appear, we will need a careful treatment of which variables have already been tested at any point.

Definition 4.3 (free variable)**.** *Given a* BDT ϕ *we call the* free variables *of a vertex or leaf* v *of* ϕ *the list of variables one encounters on the path from* v *to the root of* ϕ*.*

In other words, a variable is free at a certain point of a BDT if it needs to be tested to reach this point from the root of the tree. For instance, in the BDT



the $\mathbf{1}$ leaf has free variables x, y while the z vertex has free variables x, t. Note that at any point of the tree, the two children of an internal vertex have the same free variables.

Let us then settle a few notation to help streamline the definition of the encoding.

Notation 4.4. *We fix atomic formulas* $\alpha_1, \ldots, \alpha_n \ldots$ *and* β *and write* $\mathbf{B} = \beta \oplus \beta$*. In what follows, we will use* β^{l} *and* β^{r} *to refer respectively to the left and right copies of* β *in* $\mathbf{B}$*; and likewise* α_i^{l} *and* α_i^{r} *for copies of* α_i *in* $\alpha_i \oplus \alpha_i$ *and we implicitly associate the boolean variable* x_i *to* $\alpha_i \oplus \alpha_i$*. Moreover, we write* $\alpha_i^{\multimap}$ *for the occurrence of* α_i *in* $F_n = \alpha_n \multimap \cdots \multimap \alpha_2 \multimap \alpha_1 \multimap \beta$*.*[6]

Given n *and a subset* $I \subseteq \{x_1, \ldots, x_n\}$ *we can then identify uniquely atom occurrences in the sequents:*

- $\Lambda_I = \{\, \alpha_i \mid x_i \in I \,\}$
- $\Delta_I = \{\, \alpha_j \oplus \alpha_j \mid x_j \notin I \,\}$
- $\Gamma_{n,I} = \Lambda_I, \Delta_I, F_n$

[6] We follow the usual convention of writing the arrow $\multimap$ as right-associating: $\alpha \multimap \beta \multimap \gamma = \alpha \multimap (\beta \multimap \gamma)$

We define respectively π_1 and π_0 as the proofs $\dfrac{\beta\vdash\beta}{\beta\vdash\mathbf{B}}\oplus\mathtt{l}$ and $\dfrac{\beta\vdash\beta}{\beta\vdash\mathbf{B}}\oplus\mathtt{r}$. For any k, we write $\pi_{\oplus}^{k}$ for the proof $\dfrac{\alpha_k\vdash\alpha_k \qquad \alpha_k\vdash\alpha_k}{\alpha_k\oplus\alpha_k\vdash\alpha_k}\oplus^{\star}_{x_k}$ and π_{Id}^{k} the proof (reduced to an axiom rule) $\alpha_k\vdash\alpha_k$.

*Finally, we write R, **ex** when a rule is applied together with a series of exchanges before and after it which are obvious from context.*

We can then go on with the definition of the encoding of a BDT.

Definition 4.5. *Given a number of variables n, to any* BDT *ϕ using the variables $x_1, \ldots, x_n$, we associate a $\lambda_{\oplus}^{\multimap}$ proof π_ϕ of conclusion $\Gamma_{n,\varnothing}\vdash\mathbf{B}$ defined by induction on the tree structure of ϕ. We think of any point P in the tree as the root of a new* BDT*, augmented with the information of the free variables at that point: to P with free variables I we associate a proof of conclusion $\Gamma_{n,I}\vdash\mathbf{B}$.*

- *If P is a leaf $\mathbf{0}/\mathbf{1}$ with free variables I, we begin by setting $I_k = I\cap x_1, \ldots, x_k$ and then define π_P as*

$$\dfrac{\nu_2 \qquad \dfrac{\nu_1 \qquad \overset{\langle\pi_{\mathbf{0}/\mathbf{1}}\rangle}{\beta\vdash\mathbf{B}}}{\Gamma_{1,I_1}\vdash\mathbf{B}}\multimap^{\star}}{\Gamma_{2,I_2}\vdash\mathbf{B}}\multimap^{\star}$$

$$\cdot^{\cdot^{\cdot}}$$

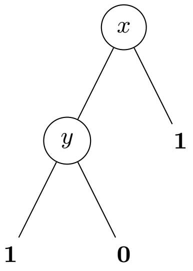
$$\dfrac{\nu_n \qquad \Gamma_{n-1,I_{n-1}}\vdash\mathbf{B}}{\Gamma_{n,I}\vdash\mathbf{B}}\multimap^{\star}$$

(with $\nu_i=\pi_{Id}^{i}$ if $x_i\in I$ and $\nu_i=\pi_{\oplus}^{i}$ otherwise)

- *If $P = x_k \rhd Q \mathbin{[\!]} R$, with free variables I, then P and Q both have free variables $I' = I, x_k$ and we define π_P as*

$$\dfrac{\overset{\langle\pi_Q\rangle}{\Gamma_{n,I'}\vdash\mathbf{B}} \qquad \overset{\langle\pi_R\rangle}{\Gamma_{n,I'}\vdash\mathbf{B}}}{\Gamma_{n,I}\vdash\mathbf{B}}\oplus^{\star}_{x_k},\ \boldsymbol{ex}$$

Example 4.6. *The* BDT *for the x `OR` (`NOT` y) function we saw in Example 1.7*



translates to the proof:

$$\dfrac{\dfrac{\alpha_x\vdash\alpha_x \qquad \dfrac{\alpha_y\vdash\alpha_y \qquad \overset{\langle\pi_{\mathbf{1}}\rangle}{\beta\vdash\mathbf{B}}}{\alpha_y,\multimap\alpha_y\multimap\beta\vdash\mathbf{B}}\multimap^{\star}}{\alpha_x,\alpha_y,\alpha_x\multimap\alpha_y\multimap\beta\vdash\mathbf{B}}\multimap^{\star} \qquad \dfrac{\alpha_x\vdash\alpha_x \qquad \dfrac{\alpha_y\vdash\alpha_y \qquad \overset{\langle\pi_{\mathbf{0}}\rangle}{\beta\vdash\mathbf{B}}}{\alpha_y,\multimap\alpha_y\multimap\beta\vdash\mathbf{B}}\multimap^{\star}}{\alpha_x,\alpha_y,\alpha_x\multimap\alpha_y\multimap\beta\vdash\mathbf{B}}\multimap^{\star}}{\alpha_x,\alpha_y\oplus\alpha_y,\alpha_x\multimap\alpha_y\multimap\beta\vdash\mathbf{B}}\oplus^{\star}_{y},\ \boldsymbol{ex} \qquad \dfrac{\alpha_x\vdash\alpha_x \qquad \dfrac{\overset{\langle\pi_{\oplus}^{y}\rangle}{\alpha_y\oplus\alpha_y\vdash\alpha_y} \qquad \overset{\langle\pi_{\mathbf{1}}\rangle}{\beta\vdash\mathbf{B}}}{\alpha_y\oplus\alpha_y,\alpha_y\multimap\beta\vdash\mathbf{B}}\multimap^{\star}}{\alpha_x,\alpha_y\oplus\alpha_y,\alpha_x\multimap\alpha_y\multimap\beta\vdash\mathbf{B}}\multimap^{\star}}{\alpha_x\oplus\alpha_x,\alpha_y\oplus\alpha_y,\alpha_x\multimap\alpha_y\multimap\beta\vdash\mathbf{B}}\oplus^{\star}_{x},\ \boldsymbol{ex}$$

We then have to check that we get indeed a faithful encoding of BDTs, reducing equivalence of BDT to equivalence of proofs.

Lemma 4.7 (representation)**.** *Writing $\mathcal{B}$ the* BDT *slicing of π_ϕ, we have*

$$\mathcal{B}[\beta,\beta^{\mathtt{l}}]=\phi \qquad \mathcal{B}[\beta,\beta^{\mathtt{r}}]=\bar{\phi}$$
$$\mathcal{B}[\alpha_i^{\mathtt{l}},\alpha_i^{\multimap}]\sim x_i\rhd\mathbf{1}\mathbin{[\!]}\mathbf{0} \qquad \mathcal{B}[\alpha_i^{\mathtt{r}},\alpha_i^{\multimap}]\sim x_i\rhd\mathbf{0}\mathbin{[\!]}\mathbf{1}$$

Proof. A straightforward induction: at the leaf level (first part of the definition) we get, if we are dealing for instance with a $\mathbf{1}$ leaf, $\mathcal{B}[\beta, \beta^{\mathtt{l}}] = \mathbf{1}$, $\mathcal{B}[\beta, \beta^{\mathtt{r}}] = \mathbf{0}$, if x_i is not free then $\mathcal{B}[\alpha_i, \alpha_i^{\multimap}] = \mathbf{1}$, if on the contrary x_i is free then $\mathcal{B}[\alpha_i^{\mathtt{l}}, \alpha_i^{\multimap}] = x_i \rhd \mathbf{1} \mathbin{[\!]} \mathbf{0}$ and $\mathcal{B}[\alpha_i^{\mathtt{r}}, \alpha_i^{\multimap}] = x_i \rhd \mathbf{0} \mathbin{[\!]} \mathbf{1}$.

From there, we can see that the induction step for variable x_i of the definition just branches together in the expected way the $\mathcal{B}[\beta, \beta^{\mathtt{l}}]$ and $\mathcal{B}[\beta, \beta^{\mathtt{r}}]$, does not change (up to equivalence) any of the $\mathcal{B}[\,\cdot\,, \alpha_j^{\multimap}]$ for $j \neq i$ and turns $\mathcal{B}[\alpha_i, \alpha_i^{\multimap}] \sim \mathbf{1}$ into $\mathcal{B}[\alpha_i^{\mathtt{l}}, \alpha_i^{\multimap}] \sim x_i \rhd \mathbf{1} \mathbin{[\!]} \mathbf{0}$ and $\mathcal{B}[\alpha_i^{\mathtt{r}}, \alpha_i^{\multimap}] \sim x_i \rhd \mathbf{0} \mathbin{[\!]} \mathbf{1}$. □

Corollary 4.8. *Two* BDT ϕ, ψ *are equivalent* iff $\pi_\phi \sim \pi_\psi$.

We can then have a look at the complexity of the reduction:

Lemma 4.9 (**Logspace** reduction)**.** *The representation* π_ϕ *of a* BDT ϕ *by a proof can be computed in logarithmic space.*

Proof. We can do this in two steps: first process the BDT to have the free variables at each vertex made explicit, which is done by listing the variables encountered between the vertex and the root of the tree and requires only to remember two positions in the tree; then each vertex of the new tree can be replaced by the corresponding piece of proof from Definition 4.5. Both these steps can be performed in logarithmic space, and since logarithmic space function compose, we are done. □

Some extra remarks about this encoding: first, because we need to keep track of which variables have been used and which have not, we do not have a linear size bound but only a quadratic one. Second, on a more positive note: we can see how closely the tree structure of π_ϕ mimics that of ϕ; in fact we believe that a form of one to one correspondance between proofs of $\Gamma_{n,I} \vdash \mathbf{B}$ and BDT using variables $x_1, \ldots, x_n$ could be worked out.

Conclusion

We established that the equivalence problem of intuitionistic additive-multiplicative (without units) linear logic is **Logspace**-complete, this was achieved by introducing an intermediate representation of proofs based on binary decision trees.

We also established low-complexity (computable in logarithmic space) correspondence between binary decision trees and proofs. This correspondance relates the tree structures of proofs and BDT very tightly amd ought to be studied further in view of potential limitation results.

However, the question of the possibility of a notion of proofnets for this logic is still unsettled: even if the equivalence problem is in **Logspace**, it might very well be that no notion of canonical representative could be built. The fact that *optimization* of BDT is a hard problem [11] (even hard to approximate [14]), could possibly be used to derive limitations on (if not impossibility of) the existence of low-complexity, canonical representatives. But this has still to be clarified.

Acknowlegments

The author would like to thank the anonymous referees the their helpful and constructive comments, and Willem Heijltjes for pointing to the idea of using the exchange rule to fix the **Logspace**-hardness proof.

References

[1] Marc Bagnol. MALL proof equivalence is logspace-complete, via binary decision diagrams. In Thorsten Altenkirch, editor, *proceedings of TLCA'15*, pages 60–75. Schloss Dagstuhl, 2015.
[2] Ashok K. Chandra, Larry J. Stockmeyer, and Uzi Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, 1984.
[3] J. R. B. Cockett and Craig A. Pastro. A language for multiplicative-additive linear logic. *Electronic Notes in Theoretical Computer Science*, 122:23–65, 2005.
[4] Kousha Etessami. Counting quantifiers, successor relations, and logarithmic space. *Journal of Computer and System Sciences*, 54(3):400–411, 1997.
[5] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–101, 1987.
[6] Jean-Yves Girard. Proof-nets: The parallel syntax for proof-theory. *Logic and Algebra*, 180:97–124, 1996.
[7] Willem Heijltjes and Robin Houston. No proof nets for MLL with units: Proof equivalence in MLL is PSPACE-complete. In Thomas A. Henzinger and Dale Miller, editors, *proceedings of CSL-LICS'14*, pages 1–10. ACM, 2014.

[8] Dominic J. D. Hughes and Willem Heijltjes. Conflict nets: Efficient locally canonical MALL proof nets. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of LICS'16*, pages 437–446. ACM, 2016.

[9] Dominic J. D. Hughes and Rob J. van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. *ACM Transactions on Computational Logic*, 6(4):784–842, 2005.

[10] Dominic J. D. Hughes and Rob J. van Glabbeek. MALL proof nets identify proofs modulo rule commutation. *available online* `http://boole.stanford.edu/~dominic/MALL-equiv.pdf`*, to appear*, 2015.

[11] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[12] Olivier Laurent and Roberto Maieli. Cut elimination for monomial MALL proof nets. In Frank Pfenning, editor, *proceedings of LICS'08*, pages 486–497. IEEE Computer Society, 2008.

[13] Harry G. Mairson and Kazushige Terui. On the computational complexity of cut-elimination in linear logic. volume 2841 of *Lecture Notes in Computer Science*, pages 23–36. Springer, 2003.

[14] Detlef Sieling. Minimization of decision trees is hard to approximate. *Journal of Computer and System Sciences*, 74(3):394–403, 2008.

[15] Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM Monographs on Discrete Mathematics and Applications, 2000.