Proof complexity of systems of (non-deterministic) decision trees and branching programs

Buss, Sam; Das, Anupam; Knop, Alexander

Proof complexity of systems of (non-deterministic) decision trees and branching programs

Sam Buss*
Dept. of Mathematics
UC San Diego
sbuss@ucsd.edu

Anupam Das†
Dept. of Computer Science
University of Copenhagen
anupam.das@di.ku.dk

Alexander Knop
Dept. of Mathematics
UC San Diego
aknop@ucsd.edu

October 21, 2019

Abstract

This paper studies propositional proof systems in which lines are sequents of decision trees or branching programs — deterministic and nondeterministic. The systems LDT and LNDT are propositional proof systems in which lines represent deterministic or non-deterministic decision trees. Branching programs are modeled as decision dags. Adding extension to LDT and LNDT gives systems eLDT and eLNDT in which lines represent deterministic and non-deterministic branching programs, respectively.

Deterministic and non-deterministic branching programs correspond to log-space (L) and nondeterministic log-space (NL). Thus the systems eLDT and eLNDT are propositional proof systems that reason with (nonuniform) L and NL properties.

The main results of the paper are simulation and non-simulation results for tree-like and dag-like proofs in the systems LDT, LNDT, eLDT, and eLNDT. These systems are also compared with Frege systems, constant-depth Frege systems and extended Frege systems.

*Supported in part by Simons Foundation grant 578919
†Supported by a a Marie Skłodowska-Curie fellowship, *Monotonicity in Logic and Complexity*, ERC project 753431.

1 Introduction

Propositional proof systems are widely studied because of their connections to complexity classes and their usefulness for computer-based reasoning. The first connections to computational complexity arose largely from the work of Cook and Reckhow [12, 17, 18], showing a connection to the NP-coNP question. These results, building on the work of Tseitin [38] initiated the study of the relative efficiency of propositional proof systems. The present paper introduces propositional proof systems that are closely connected to log-space (L) and nondeterministic log-space (NL).

Our original motivation for this study was to investigate propositional proof systems corresponding to the first-order bounded arithmetic theories VL and VNL for L and NL, see [16]. This follows a long line of work defining formal theories of bounded arithmetic that correspond to computational complexity classes, as well as to provability in propositional proof systems. The first results of this type were due (independently) to Paris and Wilkie [33] who gave a translation from $\mathrm{I}\Delta_0$ to constant-depth Frege (AC^0-Frege) proofs and to Cook [12] who gave a translation from PV to extended Frege ($\mathrm{e}\mathcal{F}$) proofs. Since the first-order bounded arithmetic theory S^1_2 is conservative over the equational theory PV, Cook's translation also applies to the bounded arithmetic theory S^1_2 [6]. As shown in the table below, similar propositional translations have since been given for a range of other theories, including first-order, second-order and equational theories.

Formal Theories	Propositional Proof Systems	Complexity Class	
PV, S^1_2	$\mathrm{e}\mathcal{F}$	P	[12, 6]
PSA, U^1_2	QBF	PSPACE	[19, 6]
T^i_2, S^{i+1}_2	G_i, G^*_{i+1}	$\mathrm{P}^{\Sigma^{\mathrm{p}}_i}$	[30, 31, 6]
VNC^0	Frege ($\mathcal{F}$)	ALogTime	[15, 16, 1]
VL	GL^*	L	[34, 16]
VNL	GNL^*	NL	[35, 16]

The first three theories are first-order theories; the last three theories are second-order. The last three theories could also be viewed as multi-sorted first-order theories, but their formalization as second-order theories makes it possible for them to work elegantly with weak complexity classes. (For an introduction to these and related results, see the books [6, 16, 28, 29].)

A hallmark of the propositional translations in the table above is that the lines in the propositional proofs express (nonuniform) properties in the corresponding complexity class. For instance, a line in a Frege proof is a propositional formula, and the evaluation problem for propositional formulas is complete for alternating log-time (ALogTime), cf. [7]. Likewise, a line in a e$\mathcal{F}$ proof is (implicitly) a Boolean circuit, and the Boolean circuit value problem is well known to be complete for P, cf. [32]. In the usual formulation of e$\mathcal{F}$, the lines only "implicitly" express Boolean circuits, since it is necessary to expand the definitions of extension variables to form the circuit; however, Jeřábek [24] made this connection explicit in a propositional proof system Circuit-Frege CF, in which lines are actually Boolean circuits.

The present paper's main goal is to define alternatives for the proof systems GL* and GNL* corresponding to log-space and nondeterministic log-space, see [34, 35, 13, 14]. The proof system GL* restricts cut formulas to be "ΣCNF(2)" formulas; the subformula property then implies that proofs contain only ΣCNF(2) formulas when proving ΣCNF(2) theorems. GNL* similarly restricts cut formulas to be "ΣKrom" formulas. (A ΣKrom formula has the form $\exists\vec{z}\varphi(\vec{z},\vec{x})$, where φ is a conjunction $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ with each C_i a disjunction of any number of x-literals and at most two z-literals.) ΣCNF(2) and ΣKrom have expressive power equivalent to nonuniform L and NL respectively [25, 21], but they are are somewhat ad hoc classes of quantified formulas, and their connections to L and NL are indirect. In this paper, we propose new proof systems, called eLDT and eLNDT, intended to be alternatives for GL* and GNL* respectively. The lines in eLDT and eLNDT proofs are sequents of formulas expressing *branching programs* and *nondeterministic branching programs*, respectively. This follows an earlier unpublished suggestion of S. Cook [11], who gave a system for L based on branching programs via "Prover-Liar" games (see [10]). The advantage of our systems is that deterministic and nondeterministic branching programs correspond directly to nonuniform L and NL respectively and do not require the use of quantified formulas. (See [39] for a comprehensive introduction to branching programs.)

To design the proof systems eLDT and eLNDT, we need to choose representations for branching programs. For this, we use a formula-based representation, as this fits well into the customary frameworks for proof systems. The formulas appearing in eLDT and eLNDT proofs will be descriptions of *decision trees*. Decision trees are not as powerful as branching programs since branching programs may be dags instead of trees. Accordingly, we also allow extension variables. The use of extension variables allows decision trees to express branching programs; this is similar to the way the extension variables in extended Frege proofs allow formulas to express circuits. An example is given in the figure on page 25.

We start in Section 2 describing proof systems LDT and LNDT that work with just deterministic and nondeterministic decision trees (without extension variables). Deterministic decision trees are represented by formulas using a single "case" or "if-then-else" connective, written in infix notation ApB, which means "if p is false, then A, else B". The condition p is required to be a literal, but A and B are arbitrary formulas. The system LDT is a sequent calculus system in which all formulas are decision trees. Nondeterministic decision trees are represented with formulas that may also use disjunctions, allowing formulas of the form $A \vee B$. The system LNDT is a sequent calculus in which all formulas are nondeterministic decision trees.

LDT and LNDT are weak systems; in fact, they are both polynomially simulated by depth-2 LK (the sequent calculus LK with all formulas of depth two). Figure 1 shows the equivalences between systems as currently established. The equivalences and separations that concern LDT and LNDT are proved in Section 4.

Section 5 introduces the proof systems eLDT and eLNDT for branching programs and nondeterministic branching programs. These again are sequent calculus systems. These systems are obtained from LDT and LNDT by adding the extension rule, thereby effectively changing the expressive power of formulas from decision trees to decision diagrams. (Decision diagrams are of course the same as a branching programs).

An important issue is designing these proof systems is how to handle isomorphic or bisimilar branching programs. Two branching programs A and B are isomorphic if there is an isomorphism (a bijection) between the nodes of the branching programs. The most convenient solution perhaps would be to allow the propositional proof systems to freely replace any branching program with any isomorphic branching program: for this, we could allow "isomorphism axioms" or "bisimilarity axioms" $A \leftrightarrow B$ whenever the two programs are isomorphic or bisimilar (respectively). For instance, isomorphism axioms of this type were used by Jeřábek [24] for the reformulation of extended Frege using Boolean circuits as lines. The problem with using isomorphism or bisimilarity axioms is that — as argued in the next paragraph — the isomorphism and bisimilarity problems for branching programs are known to be in NL, but they not known to be in L. In other words, it is open whether valid isomorphism or bisimilarity axioms are recognizable in log-space. This make the use of these axioms undesirable, at least for eLDT, as it is a proof system for log-space.

As a sketch of how to recognize bisimilarity with a NL algorithm, let A and B be branching programs. A "path" in either A or B is specified by some sequence of values $v_1, v_2, v_3, \ldots$ of *true* or *false* (1 or 0): a path is traversed in the obvious way, starting the source of the branching program, and using the value v_i to decide how to branch when reaching the i-th vertex. (Note this allows a variable to be given conflicting truth values at different points in the path.) Then A and B are *bisimilar* provided that any given path in A reaches a vertex labelled with a literal p or a sink vertex labelled with 1 or 0 if and only if the same path in B reaches a vertex labelled with the same literal p or a sink vertex labelled with the same value 1 or 0. This is clearly coNL verifiable; namely, co-nondeterministically choose a path to traverse simultaneously in A and B. Two branching programs are *isomorphic* provided that they are bisimilar, and that in addition, any two paths reach distinct nodes in A if and only if they reach distinct nodes in B. This property clearly can also be checked co-nondeterministically. Since NL = coNL (cf. [22, 37]), these properties are also in NL.

One way to handle isomorphism and bisimilarity would be to nonetheless use (say) isomorphism axioms, but require that they be accompanied by an explicit isomorphism. In our setting, this might mean giving an explicit renaming of extension variables that makes the two formulas and the definitions of their associated extension variables identical. We instead adopt a more conservative approach, and do not allow isomorphism axioms. Instead, the equivalence

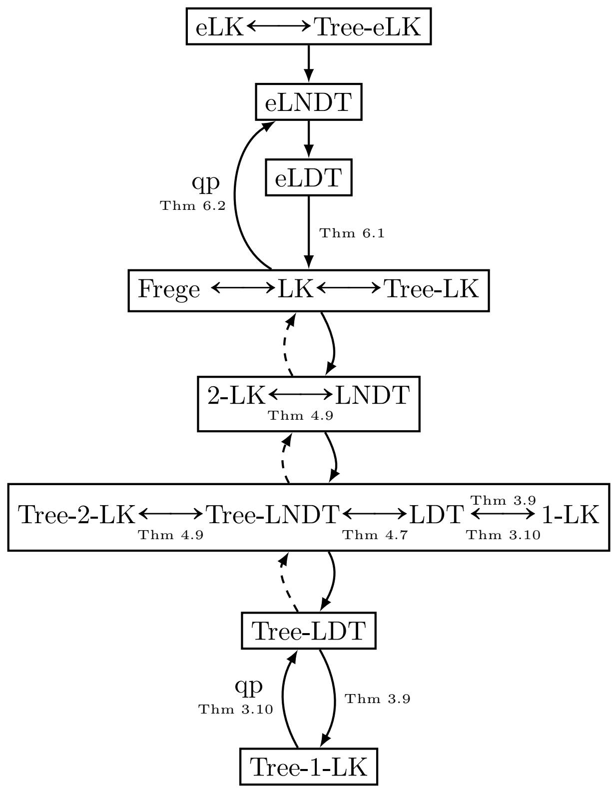


Figure 1: Relations between proof systems. $\rightarrow$ means "polynomially simulates"; $\rightarrow_{qp}$ means "quasipolynomially simulates"; $\dashrightarrow$ means "exponentially separated from". d-LK is the system of dag-like LK proofs with only depth d formulae occurring (atomic formulae have depth 0) By default, all proof systems allow dag-like proofs, unless they are labeled as "Tree".

of isomorphic branching programs (and more generally, of bisimilar branching programs) is proved explicitly, using induction on the size of the branching programs.

Since formulas in eLDT and eLNDT proofs express nonuniform L and NL properties, respectively, they are intermediate in expressive power between Boolean formulas (expressing NC^1 properties) and Boolean circuits (expressing nonuniform P properties). Thus it is not surprising that, as shown in Figure 1, these two systems are between Frege and extended Frege in strength. In addition, since NL properties can be expressed by quasipolynomial formulas, it is not unexpected that Frege proofs can quasipolynomially simulate eLNDT, and hence eLDT. These facts are proved in Section 6.

2 Decision tree formulas and LDT proofs

This section describes decision tree (DT) formulas, and the associated sequent calculus proof system LDT. All our proof systems are *propositional* proof systems with variables $x, y, z \ldots$ intended to range over the Boolean values *False* and *True*. We use 0 and 1 to denote the constants *False* and *True*, respectively. A *literal* is either a propositional variable x or a negated propositional variable $\overline{x}$. We use use variables $p, q, r, \ldots$ to range over literals.

The only connective for forming decision tree formulas (DT formulas) is the 3-ary "case" function, written in infix notation as (ApB) where A and B are formulas and p is required to be a literal. This informally means "if p is false, then A, else B". The syntax is formalized by:

Definition 2.1. The *decision tree formulas*, or DT *formulas* for short, are inductively defined by

(1) any literal p is a DT formula, and

(2) if A and B are DT formulas and p is a literal, then (ApB) is a DT formula. We call p a *decision literal*.

The parentheses in (2) ensure unique readability, but we informally write just ApB when the meaning is clear.

Suppose α is a truth assignment to the variables; the semantics of DT formulas is defined by extending α to be a truth assignment to all DT formulas by inductively defining

$$\alpha(\overline{x}) = 1 - \alpha(x) \tag{1}$$

$$\alpha(ApB) = \begin{cases} \alpha(A) & \text{if } \alpha(p) = 0 \\ \alpha(B) & \text{otherwise.} \end{cases}$$

It is important that only *literals* p may serve as the decision literals in DT formulas. Notably, for C a complex formula, an expression of the form $(A\,C\,B)$, which evaluates to A if C is true and to B if C is false, would in general be only a decision *diagram*, not a decision tree.

Although there is no explicit negation of DT formulas, we informally define the negation $\overline{A}$ of a DT formula inductively by letting $\overline{\overline{x}}$ denote x, and letting $\overline{ApB}$ denote the formula $\overline{A}\,p\,\overline{B}$. Of course $\overline{A}$ is a DT formula whenever A is, and $\overline{A}$ correctly expresses the negation of A. Notice also that negative decision literals are 'syntactic sugar', since $A\bar{p}B$ is equivalent to BpA. Nonetheless the notation is useful for making later definitions more intuitive.

Our definition of DT formulas is somewhat different from the usual definition of decision trees. The more common definition would allow 0 and 1 as atomic formulas instead of literals p as in condition (1) of Definition 2.1. We call such formulas 0/1-DT formulas. DT formulas and 0/1-DT formulas are are equivalent in expressive power. The constants 0 and 1 are equivalent to $pp\overline{p}$ and $\overline{p}pp$, for any literal p. More generally, any formula $0pA$, $1pA$, $Ap0$ or $Ap1$ is equivalent to ppA, $\overline{p}pA$, $Ap\overline{p}$, or App, respectively. Conversely, a literal p, when used as atom, is equivalent to $0p1$.

Remark 2.2 (Expressive power of decision trees)**.** It is easy to decide the validity or satisfiability of a DT formula with a log-space algorithm. A DT formula is presented as fully parenthesized, syntactically correct formula, and it is well-known that formulas can be efficiently parsed in L. To check satisfiability, for example, one examines each leaf in the formula tree (each atomic subformula p) and verifies whether the path from the root to the leaf, assigning true to the literal at the leaf, is permitted under any consistent assignment of truth values to variables.

The size of a DT formula A is the number of occurrences of atomic formulas in A. Recall that a (Boolean) CNF formula is a conjunction of disjunctions of literals; each such disjunction is called a *clause*. Likewise a (Boolean) DNF formula is a disjunction of conjunctions of literals; each such conjunction is called a *term*. A DT formula A of size n can be expressed as a DNF formula of size $O(n^2)$ with at most n disjuncts. This is defined formally as $\mathrm{Tms}(A)$ in Section 3: informally, $\mathrm{Tms}(A)$ is formed by converting the formula to a 0/1-DT formula, and then forming the disjunction, taken over all leaves labelled by a 1, of the terms expressing that that leaf is reached. A dual construction expresses a DT formula A as a CNF, denoted $\mathrm{Cls}(A)$ of size $O(n^2)$ with at most n conjuncts.

It is folklore that the construction can be partially reversed: namely any Boolean function that is equivalently expressed by a DNF φ and a CNF ψ can be represented by a DT formula of size quasipolynomial in the sizes of φ and ψ. This bound is optimal, as [26] proves a quasipolynomial lower bound.

We next define the proof system LDT for reasoning about DT formulas. Lines in an LDT proof are sequents, hence they express disjunctions of DT's. Thus lines in LDT proofs can express DNF properties: for these, the validity problem is non-trivial, in fact, coNP-complete.

Definition 2.3. A *cedent*, denoted Γ, Δ etc., is a multiset of formulas; we often use commas for multiset union, and write Γ, A for the multiset $\Gamma, \{A\}$. A *sequent* is an expression $\Gamma \longrightarrow \Delta$ where Γ and Δ are cedents. Γ and Δ are called the *antecedent* and *succedent*, respectively.

The intended meaning of $\Gamma \longrightarrow \Delta$ is that if every formula in Γ is true, then some formula in Δ is true. Accordingly, $\Gamma \longrightarrow \Delta$ is true under a truth assignment α iff $\alpha(A) = 0$ for some $A \in \Gamma$ or $\alpha(A) = 1$ for some $A \in \Delta$. A sequent is *valid* iff it is true for every truth assignment.

Definition 2.4. The sequent calculus LDT is a proof system in which lines are sequents of DT formulas. The valid initial sequents (axioms) are, for p any literal,

$$p \longrightarrow p \qquad\qquad p, \overline{p} \longrightarrow \qquad\qquad \longrightarrow p, \overline{p}.$$

The rules of inference are:

Contraction rules: $\text{c-l: } \dfrac{A, A, \Gamma \longrightarrow \Delta}{A, \Gamma \longrightarrow \Delta}$ $\qquad$ $\text{c-r: } \dfrac{\Gamma \longrightarrow \Delta, A, A}{\Gamma \longrightarrow \Delta, A}$

Weakening rules: $\text{w-l: } \dfrac{\Gamma \longrightarrow \Delta}{A, \Gamma \longrightarrow \Delta}$ $\qquad$ $\text{w-r: } \dfrac{\Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, A}$

Cut rule: $\text{cut: } \dfrac{\Gamma \longrightarrow \Delta, A \qquad A, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta}$

Decision rules: $\text{dec-l: } \dfrac{A, \Gamma \longrightarrow \Delta, p \qquad p, B, \Gamma \longrightarrow \Delta}{ApB, \Gamma \longrightarrow \Delta}$

$$\text{dec-r: } \dfrac{\Gamma \longrightarrow \Delta, A, p \qquad p, \Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, ApB}$$

Proofs are, by default, dag-like. I.e. a *proof* of a sequent S in LDT is a sequence $(S_0, \dots, S_n)$ such that S is S_n and each S_k is either an initial sequent or is the conclusion of an inference step whose premises occur amongst $(S_i)_{i<k}$. The subsystem where proofs are restricted to be tree-like (i.e. trees of sequents composed by inference steps) is denoted Tree-LDT.

The *size* of a proof is the sum of the sizes of the formulas occurring in the proof.

The inference rules that are new to LDT are the two decision rules, *dec-l* and *dec-r*. Since ApB is equivalent to $(A \vee p) \wedge (B \vee \overline{p})$, the lower sequent of a *dec-r* is true (under some fixed truth assignment) iff both upper sequents are true under the same assignment. This property of *dec-r* inferences is called "invertibility"; in particular, it means that the *dec-r* rule is sound. Similarly, since ApB is also equivalent to $(A \wedge \overline{p}) \vee (B \wedge p)$, the *dec-l* rule is also sound and invertible.

Remark 2.5 (Cut-free completeness)**.** The invertibility properties also imply that the cut-free fragment of LDT is complete. To prove this by induction on the complexity of sequents, start with a valid sequent $\Gamma \longrightarrow \Delta$; choose any non-atomic formula ApB in Γ or Δ, and apply the appropriate decision rule *dec-l* or *dec-r* that introduces this formula. The upper sequents of this inference are also valid. Since they have logical complexity strictly less then the logical complexity of $\Gamma \longrightarrow \Delta$, and thus, arguing by induction, they have cut-free proofs. The base case of the induction is when $\Gamma \longrightarrow \Delta$ contains only atomic formulas; in this case, it can be inferred from an initial sequent with weakenings. Note that this shows in fact, that any valid sequent can be proved in LDT using only decision rules, weakenings, and initial sequents. The system also enjoys a 'local' cut-elimination procedure, via standard techniques, but that is beyond the scope of this work.

Proposition 2.6. *The following have polynomial size, cut-free,* Tree-LDT *proofs:*

(a) $A \longrightarrow A$
(b) $\longrightarrow A, \overline{A}$
(c) $A, \overline{A} \longrightarrow$
(d) $A \longrightarrow p, ApB$
(e) $p, B \longrightarrow ApB$
(f) $ApB \longrightarrow A, p$
(g) $ApB, p \longrightarrow B$

Proof. To prove (a), we show by induction on the complexity of A that $\Gamma, A \longrightarrow A, \Delta$ has a polynomial size, cut-free proof. In the base case, A is a literal p, and this is an axiom. For the induction step, A has the form BpC, we use

$$\dfrac{\dfrac{B,\Gamma \longrightarrow \Delta, B, p, p \qquad p, C, \Gamma \longrightarrow \Delta, B, p}{\Gamma, BpC \longrightarrow \Delta, B, p} \qquad \dfrac{B, p, \Gamma \longrightarrow \Delta, C, p \qquad p, C, p, \Gamma \longrightarrow \Delta, C}{p, \Gamma, BpC \longrightarrow \Delta, C}}{\Gamma, BpC \longrightarrow BpC, \Delta}$$

The first and fourth upper sequents are handled by the induction hypothesis applied to B and C. The second and third upper sequents obtained from axioms by weakenings. By inspection, the resulting Tree-LDT proof has $O(n)$ lines each with $O(n)$ many symbols, where n is the size of A.

Parts (b) and (c) are proved similarly. Parts (d)-(g) are now easy to prove with a single *dec-l* or *dec-r* inference and invoking part (a). □

3 Comparing DT proof systems and LK proof systems

LK is the usual Gentzen sequent calculus for Boolean formulas over the basis $\wedge$ and $\vee$. The *Boolean formulas* are defined inductively by

(1) Any literal p is a Boolean formula, and

(2) If A and B are Boolean formulas, then so are $(A \vee B)$ and $(A \wedge B)$.

The proof system LK has the same initial sequents (axioms) as LDT, its inference rules are the contraction rules *c-l* and *c-r*, the weakening rules *w-l* and *w-r*, the cut rule, and the following Boolean rules:

Boolean rules:

$$\wedge\text{-}l\text{:}\ \dfrac{A, B, \Gamma \longrightarrow \Delta}{A \wedge B, \Gamma \longrightarrow \Delta} \qquad \wedge\text{-}r\text{:}\ \dfrac{\Gamma \longrightarrow \Delta, A \qquad \Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, A \wedge B}$$

$$\vee\text{-}l\text{:}\ \dfrac{A, \Gamma \longrightarrow \Delta \qquad B, \Gamma \longrightarrow \Delta}{A \vee B, \Gamma \longrightarrow \Delta} \qquad \vee\text{-}r\text{:}\ \dfrac{\Gamma \longrightarrow \Delta, A, B}{\Gamma \longrightarrow \Delta, A \vee B}$$

Definition 3.1. A *clause* is a disjunction of literals; a *term* is a conjunction of literals. If $\vec{p}$ is a vector of literals, we write $\bigvee \vec{p}$ to denote any disjunction of the literals $\vec{p}$, taken in the indicated order. In other words, $\bigvee p_1$ denotes p_1; and $\bigvee \vec{p}$ denotes any formula of the form $(\bigvee \vec{p}\,') \vee (\bigvee \vec{p}\,'')$ where $\vec{p}\,'$ and $\vec{p}\,''$ denote $p_1, \ldots, p_k$ and $p_{k-1}, \ldots, p_\ell$ for some $1 \leqslant k \leqslant \ell$. The notation $\bigwedge \vec{p}$ is defined similarly.

Definition 3.2. A Boolean formula is *depth one* if it is either a clause or a term. 1-LK is the fragment of LK in which all formulas appearing in sequents are depth one formulas. Tree-1-LK is the same system with the restriction that proofs are tree-like.

Although the notations $\bigvee \vec{p}$ and $\bigwedge \vec{p}$ are ambiguous about the nesting of disjunctions or conjunctions, this makes no difference in our applications since, if A and B are both of the form $\bigvee \vec{p}$ but with different orders of applications of $\lor$'s, then there are polynomial size, cut-free Tree-1-LK proofs of $A \longrightarrow B$ and $B \longrightarrow A$.

Later theorems will compare the proof theoretic strengths of various fragments and extensions of LDT to fragments of LK. Since these theories use different languages, we need to establish translations between cedents of DT formulas and (depth one) Boolean formulas.

Definition 3.3. For a (nonempty) sequence of literals $\vec{p}$ we define the DT formulas $\mathrm{Conj}(\vec{p})$ and $\mathrm{Disj}(\vec{p})$ by induction on the length of $\vec{p}$ as follows:

$$\begin{array}{rcl@{\qquad}rcl} \mathrm{Conj}(p) & := & p & \mathrm{Disj}(p) & := & p \\ \mathrm{Conj}(p, \vec{p}) & := & (pp\mathrm{Conj}(\vec{p})) & \mathrm{Disj}(p, \vec{p}) & := & (\mathrm{Disj}(\vec{p})pp) \end{array}$$

In other words, if $\vec{p} = (p_1, \ldots, p_\ell)$, for $\ell > 1$, we have:

$$\begin{array}{rcl} \mathrm{Conj}(\vec{p}) & = & (p_1p_1(p_2p_2(\cdots(p_{\ell-2}p_{\ell-2}(p_{\ell-1}p_{\ell-1}p_\ell))\cdots))) \\ \mathrm{Disj}(\vec{p}) & = & (((\cdots((p_\ell p_{\ell-1}p_{\ell-1})p_{\ell-2}p_{\ell-2})\cdots)p_2p_2)p_1p_1). \end{array}$$

It is not hard to verify that Conj and Disj correctly express the conjunction and disjunction of the literals $\vec{p}$. This is borne out by the next proposition.

Proposition 3.4. *The following sequents have polynomial size, cut-free* Tree-LDT *proofs.*

(a) $\mathrm{Conj}(\vec{p}, \vec{q}) \longrightarrow \mathrm{Conj}(\vec{p})$
(b) $\mathrm{Conj}(\vec{p}, \vec{q}) \longrightarrow \mathrm{Conj}(\vec{q})$
(c) $\mathrm{Conj}(\vec{p}), \mathrm{Conj}(\vec{q}) \longrightarrow \mathrm{Conj}(\vec{p}, \vec{q})$
(d) $\mathrm{Disj}(\vec{p}) \longrightarrow \mathrm{Disj}(\vec{p}, \vec{q})$
(e) $\mathrm{Disj}(\vec{q}) \longrightarrow \mathrm{Disj}(\vec{p}, \vec{q})$
(f) $\mathrm{Disj}(\vec{p}, \vec{q}) \longrightarrow \mathrm{Disj}(\vec{p}), \mathrm{Disj}(\vec{q})$

Proof. All six parts of the proposition are readily proved by induction on the length of $\vec{p}$, applying a *dec-l* and *dec-r* inference, and appealing to the induction hypothesis. The base cases are handled with the aid of Proposition 2.6(a). □

For the converse direction of simulating LDT (and its supersystems) by LK, we need to express a DT formula A as Boolean formulas in both CNF and DNF forms. For this we define $\mathrm{Tms}(A)$ as a multiset of terms (i.e., a multiset of conjunctions) and $\mathrm{Cls}(A)$ as a multiset of clauses (i.e., a multiset of disjunctions) so that A is equivalent to both the DNF $\bigvee \mathrm{Tms}(A)$ and the CNF $\bigwedge \mathrm{Cls}(A)$.

Definition 3.5. Let A be a DT-formula. The *terms* and *clauses* of A are the multisets $\mathrm{Tms}(A)$ and $\mathrm{Cls}(A)$ inductively defined by letting $\mathrm{Tms}(p)$ and $\mathrm{Cls}(p)$ both equal p, and letting

$$\mathrm{Tms}(BpC) \quad := \quad \{\overline{p} \wedge D : D \in \mathrm{Tms}(B)\} \cup \{p \wedge D : D \in \mathrm{Tms}(C)\} \tag{2}$$

$$\mathrm{Cls}(BpC) \quad := \quad \{p \vee D : D \in \mathrm{Cls}(B)\} \cup \{\overline{p} \vee D : D \in \mathrm{Cls}(C)\}. \tag{3}$$

The conjunctions and disjunctions are associated from right to left.

It is clear from the definition that the DNF $\bigvee \mathrm{Tms}(A)$ and the CNF $\bigwedge \mathrm{Cls}(A)$ are both equivalent to A.

Proposition 3.6. *For* DT *formulas A and B, there are polynomial size, cut-free* Tree-LK*-proofs of:*

(a) $C \longrightarrow D$*, for each* $C \in \mathrm{Tms}(A)$ *and* $D \in \mathrm{Cls}(A)$.

(b) (i) $\mathrm{Cls}(ApB) \longrightarrow D, p$*, for each* $D \in \mathrm{Cls}(A)$*;*

(ii) $p, \mathrm{Cls}(ApB) \longrightarrow D$*, for each* $D \in \mathrm{Cls}(B)$.

(iii) $\mathrm{Cls}(A) \longrightarrow D, p$*, for each* $D \in \mathrm{Cls}(ApB)$.

(iv) $p, \mathrm{Cls}(B) \longrightarrow D$*, for each* $D \in \mathrm{Cls}(ApB)$.

(c) (i) $C \longrightarrow p, \mathrm{Tms}(ApB)$*, for each* $C \in \mathrm{Tms}(A)$*;*

(ii) $p, C \longrightarrow \mathrm{Tms}(ApB)$*, for each* $C \in \mathrm{Tms}(B)$.

(iii) $C \longrightarrow p, \mathrm{Tms}(A)$*, for each* $C \in \mathrm{Tms}(ApB)$.

(iv) $p, C \longrightarrow \mathrm{Tms}(B)$*, for each* $C \in \mathrm{Tms}(ApB)$.

Part (a) of the lemma is proved by induction on the complexity of A. Parts (b) and (c) are trivial once the definitions are unwound. For example, (b.i) follows from the fact that $\mathrm{Cls}(ApB)$ contains the formula $p \vee D$. This allows (b.i) to be derived from the two sequents $p \longrightarrow p$ and $D \longrightarrow D$. The former is an axiom, and the latter has a tree-like cut-free proof by Proposition 2.6(a). The other cases are similar.

Proposition 3.7. *There are polynomial size atomic-cut* Tree-LK *proofs and polynomial size cut-free* LK *proof of the sequents* $\mathrm{Cls}(A) \longrightarrow \mathrm{Tms}(A)$ *for* DT *formulas A.*

Proof. We prove the tree-like case by giving a recursive construction. Assume A is BpC. We claim that there is a polynomial size tree-like LK derivation π_0 of the sequent

$$\{p \vee D : D \in \mathrm{Cls}(B)\} \longrightarrow \{\overline{p} \vee D : D \in \mathrm{Tms}(B)\}, p \tag{4}$$

which uses a single instance $\mathrm{Cls}(B) \longrightarrow \mathrm{Tms}(B)$ as a non-logical initial sequent. Indeed, π_0 is easily constructed by combining $\mathrm{Cls}(B) \longrightarrow \mathrm{Tms}(B)$ with initial sequents $p \longrightarrow p$ and $\longrightarrow p, \overline{p}$ using $\vee$-l and $\vee$-r inferences. Similarly, there is a polynomial size Tree-LK proof of

$$p, \{\overline{p} \vee D : D \in \mathrm{Cls}(C)\} \longrightarrow \{p \vee D : D \in \mathrm{Tms}(C)\} \tag{5}$$

which uses a single instance $\mathrm{Cls}(C) \longrightarrow \mathrm{Tms}(C)$ as a non-logical initial sequent. Combining (4) and (5) with a cut on p gives a tree-like LK derivation of $\mathrm{Cls}(A) \longrightarrow \mathrm{Tms}(A)$ which uses single instances of the sequents $\mathrm{Cls}(B) \longrightarrow \mathrm{Tms}(B)$ and $\mathrm{Cls}(C) \longrightarrow \mathrm{Tms}(C)$ as non-logical initial sequents. Proceeding recursively gives the desired polynomial size atomic-cut Tree-LK proof of $\mathrm{Cls}(A) \longrightarrow \mathrm{Tms}(A)$.

It is straightforward to give (dag-like) cut-free LK polynomial size proof of $\mathrm{Cls}(A) \longrightarrow \mathrm{Tms}(A)$, and this is omitted. Alternatively, [9] gives a general construction that, given a tree-like LK proof in which all cuts are atomic, forms a linear size dag-like LK proof. □

Proposition 3.7 can be extended to show that there are quasipolynomial size cut-free Tree-LK proofs of $\mathrm{Cls}(A) \longrightarrow \mathrm{Tms}(A)$, but it is open whether polynomial size is possible.

The next definition shows how to compare proof complexity between proof systems that work with DT formulas and ones that work with Boolean formulas.

Definition 3.8. Let P be a proof system for sequents of Boolean formulas (or at least, sequents of depth one Boolean formulas), and Q be a proof system for sequents of DT formulas. We say that P *polynomially simulates* Q if there is a polynomial time procedure which, given a Q-proof of

$$A_0, \ldots, A_{m-1} \longrightarrow B_0, \ldots, B_{n-1}, \tag{6}$$

where the A_i's and B_i's are DT-formulas, produces a P-proof of

$$\mathrm{Cls}(A_0), \ldots, \mathrm{Cls}(A_{m-1}) \longrightarrow \mathrm{Tms}(B_0), \ldots, \mathrm{Tms}(B_{n-1}). \tag{7}$$

The system Q *polynomially simulates* P if there is a polynomial time procedure which, given a P-proof of

$$\bigvee \vec{a}_0, \ldots, \bigvee \vec{a}_{m-1} \longrightarrow \bigwedge \vec{b}_0, \ldots, \bigwedge \vec{b}_{n-1}, \tag{8}$$

where the $\vec{a}_i$'s and $\vec{b}_i$'s are sequences of literals, produces a Q-proof of

$$\mathrm{Disj}(\vec{a}_0), \ldots, \mathrm{Disj}(\vec{a}_{m-1}) \longrightarrow \mathrm{Conj}(\vec{b}_0), \ldots, \mathrm{Conj}(\vec{b}_{n-1}). \tag{9}$$

The systems P and Q are *polynomially equivalent* if they polynomially simulate each other. (7) is called the *Boolean translation* of (6). (9) is called the *DT-translation* of (8). *Quasipolynomial simulation and equivalence* are defined in the same way, but using quasipolynomial time (time $2^{\log^{O(1)} n}$) procedures.[1]

[1]It turns out that all stated quasipolynomial simulations in this work (Theorems 3.10 and 6.2) take time $n^{O(\log n)} = 2^{O(\log^2 n)}$.

3.1 1-LK and LDT

Theorem 3.9. LDT *polynomially simulates* 1-LK. Tree-LDT *polynomially simulates* Tree-1-LK.

Proof. Suppose π is a 1-LK proof. Every formula in π is either a term $\bigwedge \vec{a}$ or a clause $\bigvee \vec{a}$, where $\vec{a}$ is a vector of literals. We modify π by replacing each such formula by $\mathrm{Conj}(\vec{a})$ or $\mathrm{Disj}(\vec{a})$ respectively. The initial sequents and the contraction, weakening and cut inferences in π become valid initial sequents or contraction, weakening and cut inferences for LDT.

An $\wedge$-l inference in π of the form

$$\wedge\text{-}l\text{:}\ \frac{\bigwedge \vec{a}, \bigwedge \vec{b}, \Pi \longrightarrow \Delta}{\bigwedge \vec{a} \wedge \bigwedge \vec{b}, \Pi \longrightarrow \Delta}$$

is replaced by

$$\frac{\mathrm{Conj}(\vec{a}), \mathrm{Conj}(\vec{b}), \Pi \longrightarrow \Delta}{\mathrm{Conj}(\vec{a}, \vec{b}), \Pi \longrightarrow \Delta} \tag{10}$$

This is not a valid LDT inference. To fix this, note that by parts (a) and (c) of Proposition 3.4, the cedents $\mathrm{Conj}(\vec{a}, \vec{b}) \longrightarrow \mathrm{Conj}(\vec{a})$ and $\mathrm{Conj}(\vec{a}, \vec{b}) \longrightarrow \mathrm{Conj}(\vec{a})$ have polynomial-size (cut-free) Tree-LDT proofs. Using two cut inferences with these sequents gives a valid LDT derivation of the lower sequent of (10) from the upper sequent.

An $\wedge$-r inference in π of the form

$$\wedge\text{-}r\text{:}\ \frac{\Pi \longrightarrow \Delta, \bigwedge \vec{a} \qquad \Pi \longrightarrow \Delta, \bigwedge \vec{b}}{\Pi \longrightarrow \Delta, \bigwedge \vec{a} \wedge \bigwedge \vec{b}}$$

is replaced by

$$\frac{\Pi \longrightarrow \Delta, \mathrm{Conj}(\vec{a}) \qquad \Pi \longrightarrow \Delta, \mathrm{Conj}(\vec{b})}{\Pi \longrightarrow \Delta, \mathrm{Conj}(\vec{a}, \vec{b})} \tag{11}$$

The sequent $\mathrm{Conj}(\vec{a}), \mathrm{Conj}(\vec{b}) \longrightarrow \mathrm{Conj}(\vec{a}, \vec{b})$ has a polynomial size (cut-free) Tree-LDT proof by Proposition 3.4(e). Cutting the two upper sequents of (11) against this gives a valid Tree-LDT derivation of the lower sequent.

Dual constructions allow $\vee$-l and $\vee$-r inferences in π to be converted into valid Tree-LDT derivations. The result is a valid LDT proof π' of the DT-translation of the final line of π. By construction, π' has size polynomially bounded by the size of π. Since the upper sequents of (10) and (11) were used only once when forming the Tree-LDT derivations simulating inferences of π, the LDT proof π' is tree-like whenever π is tree-like. $\square$

A converse result holds too, but we have only a quasipolynomial simulation in the tree-like case. It is open whether this can be improved to a polynomial simulation.

Theorem 3.10. 1-LK *polynomially simulates* LDT. Tree-1-LK *quasipolynomially simulates* Tree-LDT.

Proof. Suppose π is an LDT proof, possibly tree-like. We need to convert π into a 1-LK proof π'. As a first step, each sequent in π is replaced by its Boolean translation as defined in (7). Namely, every DT formula A in the antecedent, of a sequent in π is replaced by the cedent $\mathrm{Cls}(A)$; and every DT formula A in a succedent is replaced by the cedent $\mathrm{Tms}(A)$. Since $\mathrm{Cls}(p)$ and $\mathrm{Tms}(p)$ are both equal to p, the Boolean translation of an axiom in π is a valid LK axiom. Likewise, any contraction or weakening inference in π is readily replaced valid LK inferences after forming the Boolean translations. The decision rules and cut rules in π, however, need to be fixed up to make π' a valid LK-proof.

First consider a *dec-l* inference in π

$$\textit{dec-l:}\ \frac{A, \Gamma \longrightarrow \Delta, p \qquad p, B, \Gamma \longrightarrow \Delta}{ApB, \Gamma \longrightarrow \Delta} \tag{12}$$

The Boolean translation of this gives

$$\frac{\mathrm{Cls}(A), \Gamma^* \longrightarrow \Delta^*, p \qquad p, \mathrm{Cls}(B), \Gamma^* \longrightarrow \Delta^*}{ApB, \Gamma^* \longrightarrow \Delta^*} \tag{13}$$

where $\Gamma^* \longrightarrow \Delta^*$ is the Boolean translation of $\Gamma \longrightarrow \Delta$. Let $\mathrm{Cls}(A)$ equal $D_1, \ldots, D_\ell$, and $\mathrm{Cls}(B)$ equal $E_1, \ldots, E_k$, so that that $\mathrm{Cls}(ApB)$ equals the union of $\{p \vee D_i\}_{i \leqslant \ell}$ and $\{\overline{p} \vee E_i\}_{i \leqslant k}$. Starting with the upper left sequent of (13), we form an ℓ step tree-like derivation

$$\ell \text{ many } \vee\textit{-l's:}\ \frac{p \longrightarrow p \qquad \mathrm{Cls}(A), \Gamma^* \longrightarrow \Delta^*, p}{\{p \vee D_i\}_{i \leqslant \ell}, \Gamma^* \longrightarrow \Delta^*, p} \tag{14}$$

This derivation uses ℓ instances of the axiom $p \longrightarrow p$ and ℓ inferences of the form

$$\vee\textit{-l:}\ \frac{p \longrightarrow p \qquad \{p \vee D_i\}_{i<j}, D_j, \{D_i\}_{i>j}, \Gamma^* \longrightarrow \Delta^*, p}{\{p \vee D_i\}_{i<j}, p \vee D_j, \{D_i\}_{i>j}, \Gamma^* \longrightarrow \Delta^*, p}$$

A similar k step tree-like LK proof derives

$$k \text{ many } \vee\textit{-l's:}\ \frac{p, \overline{p} \longrightarrow \qquad p, \mathrm{Cls}(B), \Gamma^* \longrightarrow \Delta^*}{p, \{\overline{p} \vee E_i\}_{i \leqslant k}, \Gamma^* \longrightarrow \Delta^*} \tag{15}$$

Combining (14) and (15) with a cut on the atomic formula p gives the lower sequent, $\mathrm{Cls}(ApB)\Gamma \longrightarrow \Delta$, of (13) as desired. This gives a tree-like LK-derivation simulating (13) of size polynomially bounded by the size of the lower sequent of (12).

The case of a *dec-r* inference in π is handled dually; we omit the argument.

Now consider a cut inference in π:

$$\frac{\Gamma \longrightarrow \Delta, A \qquad A, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta} \tag{16}$$

The Boolean translation of this is

$$\frac{\Gamma^* \longrightarrow \Delta^*, \mathrm{Tms}(A) \qquad \mathrm{Cls}(A), \Gamma^* \longrightarrow \Delta^*}{\Gamma^* \longrightarrow \Delta^*} \tag{17}$$

Again let Cls(A) be $\{D_i\}_{i\leqslant\ell}$; and let Tms(A) be $\{F_i\}_{i\leqslant m}$. By Lemma 3.6(a), there are short cut-free Tree-LK proofs for each $F_i \longrightarrow D_j$. The strategy for converting (17) a valid LK-derivation is to repeatedly cut with these sequents.

There are two ways to do this. The first construction starts by deriving, for each i, the clause $F_i, \Gamma^* \longrightarrow \Delta^*$ by using ℓ cut inferences combining the sequents $F_i \longrightarrow D_j$ (for $j \leqslant \ell$) against the upper right sequent of (17). Then, combining these sequents with m cuts against the upper left sequent of (17) gives the desired sequent $\Gamma^* \longrightarrow \Delta^*$.

The second, alternative, construction is dual. It starts by deriving, for each j, the clause $\Gamma^* \longrightarrow \Delta^*, D_j$ by using m cuts inferences combining the sequents $F_i \longrightarrow D_j$ (for $i \leqslant m$) against the upper left sequent of (17). Then, combining these sequents with ℓ cuts against the upper right sequent of (17) gives the desired sequent $\Gamma^* \longrightarrow \Delta^*$.

Either of these constructions gives immediately a polynomial-size 1-LK derivation simulating the inference (17). The first construction is not tree-like since it uses the upper right sequent of (17) m times. Likewise, the second construction used the upper left sequent ℓ times. But in either case, this yields a dag-like derivation, completing the polynomial simulation of LDT by 1-LK.

The same constructions can work for the tree-like case, but this requires a more careful size analysis and gives only a quasipolynomial simulation. If π ends with a *dec-l* and *dec-r* inference, let π_0 and π_1 be the subderivations of π that end with the upper left and right sequents (respectively) of the inference (12). We use π^*, π_0^* and π_1^* to denote the Tree-1-LK proofs obtainable by the constructions above. As π ends with a decision inference, inspection of the construction above shows

$$|\pi^*| \;\leqslant\; |\pi_0^*| + |\pi_1^*| + n^{O(1)}.$$

Now suppose that π ends with the cut inference (16), and let π_0 and π_1 be the subderivations of π that end with the upper left and right sequents of (16). If $|\pi_1| \leqslant |\pi_0|$, then $|\pi_1| < |\pi|/2$; in this case, use the first construction that uses π_0^* once and π_1^* m times, to obtain a tree-like π^* of size bounded by $|\pi_0^*| + O(m \cdot |\pi_1^*|)$. Dually, if $|\pi_0| \leqslant |\pi_1|$, then $|\pi_0| < |\pi|/2$ and the second construction yields π^* of size bounded by $|\pi_1^*| + O(\ell \cdot |\pi_0^*|)$.

Let $S(n)$ be the minimal size Tree-1-LK proof required to simulate a Tree-LDT proof π of size n, namely $|\pi^*| \leqslant S(|\pi|)$. Combining the above size bounds into a single (rather crude) estimate and letting $S(0) = 0$ gives, for each n, values a and b such that $a + b < n$ and

$$S(n) \;\leqslant\; S(a) + S(b) + n^{O(1)} S(n/2).$$

From this $S(n) = n^{O(\log n)}$ follows immediately, giving the desired quasipolynomial simulation. □

4 Nondeterministic decision trees and LNDT

This section defines nondeterministic decision tree (NDT) formulas, and the associated sequent calculus LNDT. The NDT formulas have two kinds of connectives; the 3-ary case function ApB and the Boolean or gate ($\vee$). Formally,

Definition 4.1. The *nondeterministic decision tree formulas*, or NDT *formulas* for short, are inductively defined by

(1) Any literal p is a NDT formula, and

(2) If A and B are NDT formulas and p is a variable, then (ApB) is a NDT formula.

(3) If A and B are NDT formulas, then $(A \vee B)$ is an NDT formula.

A nondeterministic gate in a decision tree means a gate which is accepting exactly when at least one of its children is accepting. The corresponds exactly to an $\vee$ gate, which yields *True* exactly when at least one input is *True*. One of our motivations in defining LNDT that is will serve as a foundation for our later definition eLNDT, which will capture a logic for nondeterministic branching programs, and hence a logic for nonuniform NL.

Definition 4.2. The sequent calculus LNDT is a proof system in which lines are sequents of NDT formulas. The valid initial sequents (axioms) and rules are the same as those of LDT (Definition 2.1), along with the two $\vee$ inferences, $\vee$-l and $\vee$-r of LK as described on page 9.

For α a 0-1-truth assignment, the semantics of NDT formulas is defined extending the definition of the semantics of DT formulas, in equations 1, to include

$$\alpha(A \vee B) = \begin{cases} 1 & \text{if } \alpha(A) = 1 \text{ or } \alpha(B) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

It is straightforward to verify that LNDT is implicationally sound and implicationally complete for sequents of NDT formulas.

An important fact for NDT formulas is that we can, without loss of much generality, require the $\vee$'s to be used only as topmost connectives. This is formalized by the following definitions and theorem.

Definition 4.3. An NDT A is in *normal form* if it has the form $\bigvee_{i<n} A_i$ where each A_i is a DT formula, i.e., each A_i is $\vee$-free.

As we show below, the fact that NDT are formulas (not circuits) means that there is a polynomial time procedure to transform a a NDT formula to normal form.

Definition 4.4. We extend the definition of the multiset $\mathrm{Tms}(A)$ to NDT formulas A, by inductively defining

$$\begin{aligned} \mathrm{Tms}(BpC) &:= \{\overline{p} \wedge D : D \in \mathrm{Tms}(B)\} \cup \{p \wedge D : D \in \mathrm{Tms}(C)\} \\ \mathrm{Tms}(B \vee C) &:= \mathrm{Tms}(B) \cup \mathrm{Tms}(C). \end{aligned}$$

The multiset DTms(A) is defined to be the set of DT formulas

$$\text{DTms}(A) \;=\; \{\text{Conj}(\vec{p}) : \bigwedge \vec{p} \in \text{Tms}(A)\}.$$

Equivalently, $\text{DTms}(B \vee C) = \text{DTms}(B) \cup \text{DTms}(C)$ and

$$\text{DTms}(BpC) \;=\; \{(\overline{p}\,\overline{p}\,D) : D \in \text{DTms}(B)\} \cup \{(ppD) : D \in \text{DTms}(C)\}.$$

The normal form of an NDT formula A is defined to equal $\text{NF}(A) := \bigvee \text{DTms}(A)$. The disjunction consists of binary $\vee$ gates applied the members of DTms(A). For convenience, the disjunctions are ordered to respect the structure of the formula A. In particular, $\text{NF}(A \vee B)$ is just $\text{NF}(A) \vee \text{NF}(B)$.

The next proposition formalizes the intuition that NF(A) is equivalent to A.

Proposition 4.5. *The following have polynomial size, cut-free* Tree-LNDT *proofs:*

(a) $\text{NF}(A) \longrightarrow p, \text{NF}(ApB)$
(b) $p, \text{NF}(B) \longrightarrow \text{NF}(ApB)$
(c) $\text{NF}(ApB) \longrightarrow \text{NF}(A), p$
(d) $p, \text{NF}(ApB) \longrightarrow \text{NF}(B)$
(e) $\text{NF}(A) \longrightarrow \text{NF}(A \vee B)$
(f) $\text{NF}(B) \longrightarrow \text{NF}(A \vee B)$
(g) $\text{NF}(A \vee B) \longrightarrow \text{NF}(A), \text{NF}(B)$

Proof. We first prove (a); parts (b)-(d) are similar. For each formula D in DTms(A), the sequent $D \longrightarrow D$ has a polynomial size cut-free Tree-LDT proof by Proposition 2.6(a). From this, derive in LDT,

$$\dfrac{\textit{w-l, w-r: }\dfrac{\longrightarrow p, \overline{p}}{D \longrightarrow \overline{p}, \overline{p}, p} \qquad \textit{w-l, w-r: }\dfrac{D \longrightarrow D}{\overline{p}, D \longrightarrow p, D}}{D \longrightarrow p, (\overline{p}\,\overline{p}\,D)}\ \textit{dec-r}$$

Combining all the sequents $D \longrightarrow p, (\overline{p}\,\overline{p}\,D)$ with a tree of $\vee$-*l*, $\vee$-*r* and weakening inferences gives the desired sequent $\text{NF}(A) \longrightarrow p, \text{NF}(ApB)$.

To prove (e)-(g), note again that for each $D \in \text{Tms}(A \vee B)$, there is a polynomial size, cut-free proof of $D \longrightarrow D$. Then each of (e)-(g) can be derived by combining (some of) these sequents with a tree of $\vee$ and weakening inferences. □

We write LNDT$^{\text{NF}}$ to denote the proof system LNDT restricted to use sequents containing only NDT formulas in normal form.

Theorem 4.6. *Suppose* $\Gamma \longrightarrow \Delta$ *contains only normal form* NDT *formulas. Suppose* π *is an* LNDT *(respectively, a* Tree-LNDT*) proof of* $\Gamma \longrightarrow \Delta$*. Then* $\Gamma \longrightarrow \Delta$ *has an* LNDT$^{\text{NF}}$ *(respectively, a* Tree-LNDT$^{\text{NF}}$*) proof* π' *of size polynomially bounded by the size of* π*.*

Proof. As a first step towards forming π', replace every formula A in π with NF(A). Axioms in π are unchanged. Contraction inferences, weakening inferences, and cut inferences in π remain valid inferences. Likewise, since $\text{NF}(A \vee B)$

equals $\mathrm{NF}(A) \vee \mathrm{NF}(B)$, the $\vee$ inferences in π remain valid. However, the *dec-r* and *dec-l* may no longer be valid and need to be fixed up. Consider a *dec-r* inference in π:

$$\textit{dec-r:}\ \frac{\Pi \longrightarrow \Lambda, A, p \qquad p, \Pi \longrightarrow \Lambda, B}{\Pi \longrightarrow \Lambda, ApB}$$

This is transformed to

$$\frac{\Pi^* \longrightarrow \Lambda^*, \mathrm{NF}(A), p \qquad p, \Pi^* \longrightarrow \Lambda^*, \mathrm{NF}(B)}{\Pi^* \longrightarrow \Lambda^*, \mathrm{NF}(ApB)} \tag{18}$$

where Π^* and Λ^* are the cedents obtained after replacing each formula by its normal form. Applying cuts with the formulas (a) and (b) of Proposition 4.5 and then a cut on p gives

$$\frac{\dfrac{\Pi^* \longrightarrow \Lambda^*, \mathrm{NF}(A), p \qquad \mathrm{NF}(A) \longrightarrow p, \mathrm{NF}(ApB)}{\Pi^* \longrightarrow \Lambda^*, \mathrm{NF}(ApB), p} \qquad \dfrac{p, \Pi^* \longrightarrow \Lambda^*, \mathrm{NF}(B) \qquad p, \mathrm{NF}(B) \longrightarrow \mathrm{NF}(ApB)}{p, \Pi^* \longrightarrow \Lambda^*, \mathrm{NF}(ApB)}}{\Pi^* \longrightarrow \Lambda^*, \mathrm{NF}(ApB)}$$

This turns (18) into a LNDT derivation. □

4.1 LDT and tree-like LNDT are equivalent

Next we turn to the relative complexity of LDT and LNDT. Naturally the latter subsumes the former, but this can be strengthened as follows.[2]

Theorem 4.7. Tree-LNDT *is polynomially equivalent to* LDT *over* DT*-sequents.*

Proof. We first show Tree-LNDT polynomially simulates LDT. Suppose π is an LDT-proof (possibly dag-like) with m sequents $\Gamma_i \longrightarrow \Delta_i$ for $i = 1, \ldots, m$. Define $\overline{\Gamma}$ to be the multiset of formulas $\overline{F}$ for $F \in \Gamma$. Let A_i be $\bigvee(\overline{\Gamma} \cup \Delta)$, namely a tree of (binary) disjunctions of the formulas in $\overline{\Gamma} \cup \Delta$. (The disjunctions may be applied in any order.) Clearly, each A_i is a NDT-formula.

The next claim will help us work with disjunctions.

Claim 4.8. *Let $\Pi, \Lambda, \Gamma, \Delta$ be cedents. Suppose that for each formula $F \in \Pi$, the formula $F \in \Lambda \cup \Delta \cup \overline{\Gamma}$. (If there are multiple occurrences of F in Π it is not required to have multiple occurrences of F in $\Lambda \cup \Delta \cup \overline{\Delta}$.) Let $\mathcal{H}$ ("hypotheses") be the set containing the cedents $F \longrightarrow F$ such that $F \in \Pi \cap (\Lambda \cup \Delta)$ and the cedents $\overline{F}, F \longrightarrow$ for $F \in (\Pi \cap \overline{\Gamma})$. Then the sequent*

$$\Gamma, \bigvee \Pi \longrightarrow \bigvee \Lambda, \Delta$$

has a polynomial size, tree-like, cut-free proof from (a subset of) the initial sequents $\mathcal{H}$. using only $\vee$ inferences and weakenings.

[2]This also refines the known polynomial equivalence between 1-LK and Tree-2-LK, cf. Figure 1.

To understand the claim, note that the assumption is that any F in Π also appears in Λ or Δ or negated in Γ. The proof of the claim is by a simple application of $\vee$-*l* and $\vee$-*r* rules.

Returning to the proof of Theorem 4.7, consider some A_i. If $\Gamma_i \longrightarrow \Delta_i$ is an axiom, then A_i has the form $p \vee \overline{p}$. Clearly there is a short cut-free Tree-LNDT proof of $\longrightarrow A_i$. If $\Gamma_i \longrightarrow \Delta_i$ is inferred from $\Gamma_j \longrightarrow \Delta_j$ by a *unary* inference (with $j < i$), then by inspection of the contraction and weakening rules, $(\overline{\Gamma}_j \cup \Delta_j) \subseteq \overline{\Gamma}_i \cup \Delta_i$. Thus, by the claim, there is a polynomial size, cut-free Tree-LNDT-proof of $A_j \longrightarrow A_i$, since A_i is $\bigvee(\overline{\Gamma}_i \cup \Delta_i)$ and A_j is $\bigvee(\overline{\Gamma}_j \cup \Delta_j)$.

Finally, suppose $\Gamma_i \longrightarrow \Delta_i$ is inferred by a binary inference from $\Gamma_j \longrightarrow \Delta_j$ and $\Gamma_k \longrightarrow \Delta_k$ (with $j, k < i$). We will prove that the sequent$A_j, A_k \longrightarrow A_i$ has a polynomial size tree. Suppose A_i is inferred by a cut inference,

$$\textit{cut:}\ \frac{\Gamma_i \longrightarrow \Delta_i, C \qquad C, \Gamma_i \longrightarrow, \Delta_i}{\Gamma_i \longrightarrow \Delta_i}$$

Then A_j is $\bigvee(\Delta_i \cup \{C\} \cup \overline{\Gamma}_i)$ and A_i is $\bigvee(\Delta_i \cup \overline{\Gamma}_i)$ and the Claim 4.8 and Proposition 2.6 imply that $A_j \longrightarrow A_i, C$ has a polynomial size cut-free proof. Similarly, $\overline{C}, A_k \longrightarrow A_i$ has polynomial size, cut-free proof. Using a cut on C, gives a proof of $A_j, A_k \longrightarrow A_i$. Second, suppose A_i is inferred by a *dec-l* inference

$$\textit{dec-l:}\ \frac{A, \Gamma_i' \longrightarrow \Delta_i, p \qquad p, B, \Gamma_i' \longrightarrow \Delta_i}{ApB, \Gamma_i' \longrightarrow \Delta_i}$$

where Γ_i is ApB, Γ_i', and the upper left and right sequents are $\Gamma_j \longrightarrow \Delta_j$ and $\Gamma_k \longrightarrow \Delta_k$, respectively. Since A_j is $\bigvee\{\overline{A}, \overline{\Gamma}_i, \Delta_i, p\}$ and A_k is $\bigvee\{\overline{B}, \overline{p}, \overline{\Gamma}_i, \Delta_i\}$ and A_i is $\bigvee\{\overline{ApB}, \overline{\Gamma}_i, \Delta_i\}$, Claim 4.8 and Proposition 2.6 give polynomial size, cut-free Tree-LNDT proofs of $A, A_j \longrightarrow A_i, p$ and $p, B, A_k \longrightarrow A_i$. Applying a *dec-l* rule gives a polynomial size Tree-LNDT of $A_j, A_k \longrightarrow A_i$. The third case where A_i is inferred by a *dec-l* inference is similar, and again we obtain a polynomial size Tree-LNDT of $A_j, A_k \longrightarrow A_i$.

We have shown that for each $i \leqslant m$, there is are (up to two) values $j, k < i$ such that the sequent $A_j, A_k \longrightarrow A_i$ has a polynomial size, Tree-LNDT proof, where the formulas A_j and A_k are possibly omitted. We can now complete the proof of the first half of Theorem 4.7. By Claim 4.8, there is a polynomial size Tree-LNDT proof of $A_1, \ldots, A_m, \Gamma_m \longrightarrow \Delta_m$. Cutting with the sequents $A_j, A_k \longrightarrow A_i$ for $i = m, m{-}1, \ldots, 2, 1$, we derive successively $A_1, \ldots, A_\ell, \Gamma_m \longrightarrow \Delta_m$ for $\ell = m, \ldots, 2, 1$. With $\ell = 0$, a polynomial size Tree-LNDT proof of $\Gamma_m \longrightarrow \Delta_m$, the endsequent of π. This completes the proof that Tree-LNDT polynomially simulates LDT.

To prove the second part of Theorem 4.7, suppose π is a Tree-LNDT proof. By Theorem 4.6, we may assume that every formula in π is in normal form. That is, each sequent $\Gamma \longrightarrow \Delta$ in π has the form

$$\bigvee \Pi_1, \ldots, \bigvee \Pi_k \longrightarrow \bigvee \Lambda_1, \ldots, \bigvee \Lambda_\ell$$

where each Π_i and Λ_j is a multiset of DT-formulas. We shall prove that there is a polynomial size DT derivation π' of the sequent

$$\longrightarrow \Lambda_1, \ldots, \Lambda_\ell \tag{19}$$

from the extra hypotheses $\longrightarrow \Pi_i$. The proof is by induction on the number of lines in the proof π. If π is just an axiom, then this is trivial. Otherwise the argument splits into cases depending on the final inference of π.

For a more compact notation, we write $\mathcal{F}(\Delta)$ to denote the succedent in (19) ("$\mathcal{F}$" for "flatten"). And we write $\mathcal{H}(\Gamma)$ to denote the set of sequents $\longrightarrow \Lambda_i$ ("$\mathcal{H}$" for "hypotheses").

If π ends with a weakening or contraction inference, the argument is essentially trivial. For instance, if π ends with a *c-l* inference

$$\textit{c-l:}\ \frac{A, A, \Gamma \longrightarrow \Delta}{A, \Gamma \longrightarrow \Delta}$$

then the induction hypothesis gives a LDT proof π_0' of $\mathcal{F}(\Delta)$ from the hypotheses $\mathcal{H}(A, A, \Gamma)$. But $\mathcal{H}(A, A, \Gamma)$ is equal to $\mathcal{H}(A, \Gamma)$, we can just take π' to be π_0'. The case where π ends with a *w-l* inference is handled similarly, since $\mathcal{H}(A, \Gamma)$ is a superset of $\mathcal{H}(\Gamma)$. If π ends with a *c-r* inference or a *w-r* inferences, we form π' by adding the same kind of inference to the end of the LDT deduction π_0' given by the induction hypothesis.

Suppose the final inference of π is a cut inference

$$\textit{cut:}\ \frac{\Gamma \longrightarrow \Delta, A \qquad A, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta}$$

The cut formula A is an NDT formula, hence it is of the form $\bigvee \Lambda$ for some cedent Λ of DT formulas, and $\mathcal{F}(A) = \Lambda$.

The two upper sequents of the cut have (disjoint since tree-like) Tree-LNDT proofs π_0 and π_1. The induction hypothesis gives an LDT proof π_0' of the sequent $\longrightarrow \mathcal{F}(\Delta), \Lambda$ from the hypotheses $\mathcal{H}(\Gamma)$ and an LDT proof π_1' of $\longrightarrow \mathcal{F}(\Delta)$ from the hypotheses $\longrightarrow \Lambda$ and $\mathcal{H}(\Gamma)$. We modify π_1' to form a new LDT derivation, denoted $\pi_1' \triangleright \mathcal{F}(\Delta)$, which is formed from π_1' by replacing each sequent $\Pi \longrightarrow \Xi$ in π_1' with $\Pi \longrightarrow \Xi, \mathcal{F}(\Delta)$, and then fixing up initial sequents to be validly derived by adding weakening inferences as needed. This forms $\pi_1' \triangleright \mathcal{F}(\Delta)$ as a LDT-proof of $\longrightarrow \mathcal{F}(\Delta), \mathcal{F}(\Delta)$ from the hypotheses $\mathcal{H}$ and $\longrightarrow \mathcal{F}(\Delta), \Lambda$. We form the desired proof π' by concatenating π_0' and $\pi_1' \triangleright \mathcal{F}(\Delta)$ and concluding with contraction inferences:

$$\begin{array}{ll}
& \mathcal{H}(\Gamma) \\
& \quad \vdots \quad \pi_0' \\
& \longrightarrow \mathcal{F}(\Delta), \Lambda \\
& \quad \vdots \quad \pi_1' \triangleright \mathcal{F}(\Delta) \\
\textit{c-r:} & \dfrac{\longrightarrow \mathcal{F}(\Delta), \mathcal{F}(\Delta)}{\longrightarrow \mathcal{F}(\Delta)}
\end{array}$$

This yields π' as a polynomial size LDT proof of $\longrightarrow \mathcal{F}(\Delta)$ from the hypotheses $\mathcal{H}$.

Now suppose the final inference of π is an $\vee$-r inference

$$\lor\text{-}r\text{:}\ \frac{\Gamma \longrightarrow \Delta, A, B}{\Gamma \longrightarrow \Delta, A \lor B}$$

The NDT formulas A and B are equal to $\bigvee \Pi$ and $\bigvee \Lambda$ where Π and Λ are cedents of DT formulas. The induction hypothesis gives an LDT proof π_0' of $\longrightarrow \mathcal{F}(\Delta), \Pi, \Lambda$ from the hypotheses $\mathcal{H}(\Gamma)$. The desired proof π' is just equal to π_0.

Now suppose the final inference of π is an $\lor$-l inference

$$\lor\text{-}l\text{:}\ \frac{A, \Gamma \longrightarrow \Delta \qquad B, \Gamma \longrightarrow \Delta}{A \lor B, \Gamma \longrightarrow \Delta}$$

The NDT formulas A and B are again equal to $\bigvee \Pi$ and $\bigvee \Lambda$. The induction hypothesis gives an LDT proof π_0' of $\longrightarrow \mathcal{F}(\Delta)$ from the hypotheses $\longrightarrow \Pi$ and $\mathcal{H}(-)$, and gives an LDT proof π_1' of $\longrightarrow \mathcal{F}(\Delta)$ from the hypotheses $\longrightarrow \Lambda$ and $\mathcal{H}(\Gamma)$. We must produce an LDT proof π' of $\longrightarrow \mathcal{F}(\Delta)$ from the hypotheses $\longrightarrow \Pi, \Lambda$ and $\mathcal{H}(\Gamma)$. We form $\pi_0' \triangleright \Lambda$ by adding Λ to the antecedent of each sequent in π_0', and then fixing up all initial sequents with weakening inferences, except leaving the initial sequents $\longrightarrow \Pi, \Lambda$ as is. This makes $\pi_0 \triangleright \Lambda$ an LDT derivation of $\longrightarrow \mathcal{F}(\Delta), \Lambda$ from the hypotheses $\longrightarrow \Pi, \Lambda$ and $\mathcal{H}(\Gamma)$. We similarly form $\pi_1' \triangleright \mathcal{F}(\Delta)$ to be a LDT proof of $\longrightarrow \mathcal{F}(\Delta), \mathcal{F}(\Delta)$ from the hypotheses $\longrightarrow \mathcal{F}(\Delta), \Lambda$ and $\mathcal{H}(\Gamma)$. Putting these together as:

$$\begin{array}{c}
\longrightarrow \Pi, \Lambda \qquad \mathcal{H}(\Gamma) \\
\ddots \vdots \cdot^{\cdot^{\cdot}} \quad \pi_0' \triangleright \Lambda \\
\longrightarrow \mathcal{F}(\Delta), \Lambda \\
\ddots \vdots \cdot^{\cdot^{\cdot}} \quad \pi_1' \triangleright \mathcal{F}(\Delta) \\
c\text{-}r\text{:}\ \dfrac{\longrightarrow \mathcal{F}(\Delta), \mathcal{F}(\Delta)}{\longrightarrow \mathcal{F}(\Delta)}
\end{array}$$

forms the desired LDT proof of $\longrightarrow \mathcal{F}(\Delta)$ from the hypotheses $\longrightarrow \Pi, \Lambda$ and $\mathcal{H}(\Gamma)$.

Now suppose the final inference of π is a *dec-r* inference

$$dec\text{-}r\text{:}\ \frac{\Gamma \longrightarrow \Delta, A, p \qquad p, \Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, ApB}$$

A and B are DT formulas. The induction hypothesis gives an LDT proof π_0' of $\longrightarrow \mathcal{F}(\Delta), A, p$ from the hypotheses $\mathcal{H}(\Gamma)$ and an LDT proof π_1' of $\longrightarrow \mathcal{F}(\Delta), B$ from the hypotheses $\longrightarrow p$ and $\mathcal{H}(\Gamma)$. We form an LDT proof $p \triangleright \pi_1'$ by adding p to each antecedent, replacing the hypothesis $\longrightarrow p$ with the axiom $p \longrightarrow p$, and adding weakenings to fix up the other initial sequents. The desired LDT proof π' is formed as:

$$
\textit{dec-r:}\ \dfrac{\begin{array}{c}\mathcal{H}(\Gamma)\\ \vdots\ \pi_0'\\ \longrightarrow \mathcal{F}(\Delta), A, p\end{array}\qquad \begin{array}{c}\mathcal{H}(\Gamma)\\ \vdots\ p\triangleright\pi_1'\\ p \longrightarrow \mathcal{F}(\Delta), B\end{array}}{\longrightarrow \mathcal{F}(\Delta), ApB}
$$

Finally suppose the final inference of π is a *dec-l* inference

$$
\textit{dec-l:}\ \dfrac{A, \Gamma \longrightarrow \Delta, p \qquad p, B, \Gamma \longrightarrow \Delta}{ApB, \Gamma \longrightarrow \Delta}
$$

where A and B are again DT formulas, and the induction hypothesis gives an LDT proof π_0' of $\longrightarrow \mathcal{F}(\Delta), p$ from the hypotheses $\longrightarrow A$ and $\mathcal{H}(\Gamma)$ and an LDT proof π_1' of $\longrightarrow \mathcal{F}(\Delta)$ from the hypotheses $\longrightarrow p$ and $\longrightarrow B$ and $\mathcal{H}(\Gamma)$. We need to form an LDT proof of $\longrightarrow \mathcal{F}(\Delta)$ from the hypothesis $\longrightarrow ApB$ and $\mathcal{H}(\Gamma)$. From Proposition 2.6(f,g), there are short LDT proofs of $ApB \longrightarrow A, p$ and $ApB, p \longrightarrow B$. Similarly to the previous cases, we form an LDT proof $\pi_0'\triangleright p$ of $\longrightarrow \mathcal{F}(\Delta), p$ from the hypotheses $\longrightarrow A, p$ and $\mathcal{H}(\Gamma)$. We also form an LDT proof $p\triangleright\pi_1'$ of $p \longrightarrow \mathcal{F}(\Delta)$ from the hypotheses $p \longrightarrow B$ and $\mathcal{H}(\Delta)$. Combining all these with cuts gives the desired LDT proof π as:

$$
\textit{cut:}\ \dfrac{\begin{array}{c}\textit{cut:}\ \dfrac{\longrightarrow ApB \qquad \begin{array}{c}\vdots\ \text{Prop. 2.6(f)}\\ ApB \longrightarrow A, p\end{array}}{\longrightarrow A, p}\\ \vdots\ \pi_0'\triangleright p\\ \longrightarrow \mathcal{F}(\Delta), A, p\end{array}\qquad \begin{array}{c}\textit{cut:}\ \dfrac{\longrightarrow ApB \qquad \begin{array}{c}\vdots\ \text{Prop. 2.6(g)}\\ ApB, p \longrightarrow B\end{array}}{p \longrightarrow B}\\ \vdots\ p\triangleright\pi_1'\\ p, \longrightarrow \mathcal{F}(\Delta)\end{array}}{\longrightarrow \mathcal{F}(\Delta)}
$$

It is not hard to verify that proof π' is constructible from π in polynomial time. That completes the proof of Theorem 4.7. ☐

4.2 Equivalence of LNDT and 2-LK

A Boolean formula is *depth two* if it is depth one, or if it is a conjunction of clauses or a disjunction of terms. 2-LK is the fragment of LK in which all formulas appearing in sequents are depth two formulas. Tree-2-LK is the same system with the restriction that proofs are tree-like.

Theorem 4.9. LNDT *and* 2-LK *are polynomially equivalent.* Tree-LNDT *and* Tree-2-LK *are polynomially equivalent.*

The equivalence between LNDT and 2-LK is even stronger than is required by Definition 3.8. In fact, *any* LNDT proof can be faithfully translated into a 2-LK proof. For the converse, we sketch below how any 2-LK proof in which the final sequent is contains only disjunctions of conjunctions can be faithfully translated to a LNDT proof. This means essentially that *any* 2-LK proof can be faithfully translated to a LNDT proof, since any conjunctions of disjunctions can be moved to the other side of the sequent where they become disjunctions of conjunctions.

Proof. (Sketch) Suppose π is a LNDT proof. By Theorem 4.6, every formula in π may be assumed to be a normal form NDT formula. To convert π to a 2-LK proof π', we first replace every formula $\bigvee A_i$ in π with the depth two Boolean formula $\bigvee \mathrm{Tms}(A_i)$. Axioms and contraction, weakening, cut and $\vee$ inferences in π remain valid inferences in π'. Decision rules *dec-l* and *dec-r* in π are easily fixed to be valid derivation in π' using axioms $\overline{p}, p \longrightarrow$ and $\longrightarrow \overline{p}, p$, cuts on p, and $\wedge$ and $\vee$ inferences. The resulting 2-LK proof π' has size linearly bounded by the size of π. In addition, if π is tree-like, then so is π'.

Conversely, suppose π is a 2-LK proof, and that every formula in the conclusion of π is a disjunction of conjunctions of literals. We may assume w.l.o.g. that every formula in π is a disjunction of conjunctions of literals, since any conjunction of disjunctions can be negated and moved to the other side of the cedent as a disjunction of conjunctions. We thus can transform π into π' by replacing every formula $\bigvee A_i$ in π, where the A_i's are conjunctions of literals, with the NDT formula $\bigvee \mathrm{Conj}(A_i)$. The axioms and the contraction, weakening, cut and $\vee$ inferences in π remain valid after this transformation. The $\wedge$ rules in π can be fixed to be valid derivations in π using the derivations of Proposition 3.3(a,c,e) and cuts on formulas $\mathrm{Conj}(\vec{p})$ and $\mathrm{Conj}(\vec{q})$ for $\vec{p}$ and $\vec{q}$ vectors of literals. □

5 Proof systems for branching programs

5.1 Formulas and proofs with extension variables

We now describe the propositional proof systems eLDT and eLNDT which reason about deterministic and nondeterministic branching programs.[3] Formulas can now include extension variables, which will be denoted by the letter e, or with a subscript as e_1, e_2, etc.. It is important that the extension variables e are new variables that are distinct from the variables underlying literals p.

The purpose of extension variables is to serve as abbreviations for more complex formulas. Thus, proofs that use extension variables will be accompanied by a set of extension axioms $\{e_i \leftrightarrow A_i\}_{i<n}$, where each formula A_i may use any literals p but is restricted to use only the extension variables e_j for $j < i$. The intent is that e_i is an abbreviation for the formula A_i.

Definition 5.1. The *extended decision tree* formulas, or eDT formulas for short, are inductively defined

(1) Any literal p is an eDT formula.

[3]These systems could equally well be called LBP and LNBP, using "BP" for "branching programs", but the notations eLDT and eLNDT indicate that branching programs are represented with decision trees incorporating extension variables.

(2) Any extension variable e is an eDT formula.

(3) If A and B are eDT formulas and p is a literal, then (ApB) is a DT formula.

In particular, a decision literal p in a formula ApB is *not* allowed to be an extension variable. The intuition is that the extension variables may 'name' nodes in a branching program.

Definition 5.2. The *extended nondeterministic decision tree* formulas, or eNDT formulas for short, are inductively defined by the closure conditions (1)-(3) above (with "eDT" replaced with "eNDT") and:

(4) If A and B are eNDT formulas, then $(A \vee B)$ is an eNDT formula.

Definition 5.3. The *extended Boolean formulas* are defined inductively by

(1) Any literal p is a extended Boolean formula.

(2) Any extension variable e is an extended Boolean formula.

(3) If A and B are extended Boolean formulas, then so are $(A \vee B)$ and $(A \wedge B)$.

The notation $\{e_i \leftrightarrow A_i\}_{i<n}$ is used to indicate that $e_0, \ldots, e_{n-1}$ are extension variables and that the only extension variables allowed to appear in A_i are $e_0, \ldots, e_{i-1}$. The sequents

$$e_i \longrightarrow A_i \qquad \text{and} \qquad A_i \longrightarrow e_i$$

are called the *extension axioms*.

The eDT, eNDT and eLK formulas have truth semantics only relative to a set of extension axioms $\{e_i \leftrightarrow A_i\}_{i<n}$. Namely, for α a truth assignment, the definition of truth is extended by setting $\alpha(e_i) = \alpha(A_i)$.

Definition 5.4. An eLDT proof is a pair $(\pi, \{e_i \leftrightarrow A_i\}_{i<n})$ where each A_i is an eDT formula, all formulas in π are eDT formulas, and the permitted initial sequents and rules of DT plus the extension axioms of $\{e_i \leftrightarrow A_i\}_{i<n}$ are allowed as initial sequents in π.

The eLNDT proofs are defined similarly, but with eLNDT formulas A_i and using the eLNDT inference rules. Similarly, eLK proofs are defined by letting the A_i be eLK formulas and using the LK inference rules.

Clearly the eLK proof system is equivalent to the usual extended Frege proof system: in conjunction with a set of extension axioms, an extended Boolean formula represents a Boolean circuit over the de Morgan connectives $\wedge, \vee, \neg$.

Note that all formulas in an eLDT, eLNDT or eLK proof are based on the a single set of extension axioms $\{e_i \leftrightarrow A_i\}_{i<n}$.

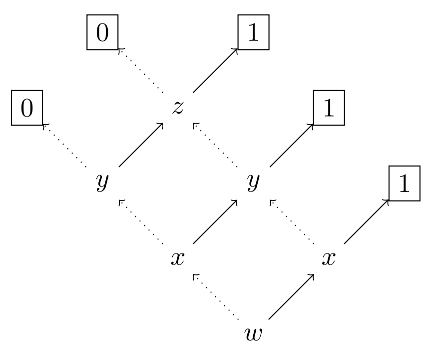
Let us discuss how the extended formulas we have introduced may be used to represent bona fide branching programs. A (deterministic) branching program is a directed acyclic graph G such that (a) G has a unique source node,

(b) sink nodes in G are labelled with either 0 or 1, (c) all other nodes are labelled with a literal p and have two outgoing edges, one labelled 0 and the other 1. A deterministic branching program G can be converted into an equivalent eDT formula with associated extension axioms $\{e_i \leftrightarrow A_i\}_{i<n}$ by introducing an extension variable e_i for every internal node in the branching program. Conversely, as is described in more detail below, any eDT formula A with extension axioms $\{e_i \leftrightarrow A_i\}_{i<n}$ can be straightforwardly transformed into a linear size deterministic branching program. For this, the nodes in the branching program correspond to the extension variables e_i and the subformulas of the formulas A_i.

Nondeterministic branching programs are defined similarly to deterministic branching programs, but further allowing the internal nodes of G to be labelled with "$\vee$" as well as literals (in this case the labelling of its outgoing edges is omitted). The semantics is that an $\vee$-node is accepting provided at least one of its children is accepting. It is straightforward to convert a nondeterministic branching program into an eLNDT formula with associated extension axioms, and vice versa.

A similar construction yields the well-known fact that extended Boolean formulas are as expressive as Boolean circuits.

Example 5.5. Consider the following branching program, which returns 1 just if at least two out of the four input variables w, x, y, z are 1.



Edges labelled with 0 are here dotted (and always left outgoing) while edges labelled 1 are here solid (and always right outgoing). In this particular case, the branching program is *ordered* (or an *OBDD*), i.e. variables occur in the same order on each branch. The program also happens to compute a monotone Boolean function.

To express the branching program above in eLDT, we introduce extension variables for each inner node of the program as follows. Write e_{ij} for the jth node of the ith layer, where i, j ranging from 0 onwards, and introduce the following extension axioms:[4]

$$\begin{array}{rcl} e_{10} & \leftrightarrow & e_{20}xe_{21} \\ e_{11} & \leftrightarrow & e_{21}x1 \\ e_{20} & \leftrightarrow & 0ye_{31} \\ e_{21} & \leftrightarrow & e_{31}y1 \\ e_{31} & \leftrightarrow & 0z1 \end{array}$$

Now the branching program is represented as the eDT formula $e_{10}we_{11}$. Notice that the orderedness of the branching program is reflected in its eLDT representation: writing (x_0, x_1, x_2, x_3) for (w, x, y, z), we have that x_i is the root of the formula that any e_{ij} abbreviates.

Other representations of this branching program are possible, for instance by renaming the extension variables or by partially unwinding the graph. In both these two latter cases, the eDT representation obtained will be provably equivalent to the one above, by polynomial-size proofs in eLDT, by virtue of Lemma 5.11 later.

5.2 Foundational issues

The fact that extension variables cannot be used as decision literals is a significant limitation on the expressiveness of DT formulas. Recall for instance that the conjunction of p_1 and p_2 can be expressed with the DT formula $\mathrm{Conj}(p_1, p_2)$, namely $(p_1p_1p_2)$. However, it is not permitted to form $(e_1e_1e_2)$; in fact, it is not possible to express the conjunction $e_1 \wedge e_2$ without taking the extension axioms defining e_1 and e_2 into account. In fact, if we could write the conjunction of e_1 and e_2 by a generic formula $A(e_1, e_1)$, then we could introduce a new extension variable representing $A(e_1, e_2)$. This would imply that eDT formulas are as expressive as extended Boolean formulas; in other words, that deterministic branching programs would be as expressive as Boolean circuits. This is a non-uniform analogue of L = P (i.e., log-space equals polynomial time), and of course is an open question.

Nonetheless, for any given extension variables e and e', there is a formula $\textsc{And}(e, e')$ expressing the conjunction of e and e' by changing the underlying set of extension axioms. The intuition is that we start with the branching program G for e, but now with sink nodes labelled with 0 or 1 instead of with variables. To form the branching program for $e \wedge e'$, we take (an isomorphic copy) of the branching program G' for e', and modify G by replacing each sink node labelled with 1 with the source node of G' (in other words, each edge directed into a sink "1" is modified to instead point to the root of G').

More formally, suppose A and B are eDT formulas defined over a set of extension axioms $\{e_i \leftrightarrow A_i\}_{i<n}$; we wish to construct an eDT formula $\textsc{And}(A, B)$. (Exactly the same construction forms an eNDT formula $\textsc{And}(A, B)$ from eNDT

[4]Formally, we are writing 0 and 1 as shorthand for $pp\bar{p}$ and $\bar{p}pp$ respectively, for some/any literal p.

formulas A and B.) We would wish to define $C[1/B]$ to be the result of replacing every "1" in C with B, but of course, "1" is not a permitted atom. Instead, we note that every atomic formula p in C is equivalent to (ppp) and to $(pp1)$. Likewise, each atomic formula p is equivalent to $(0pp)$.

Definition 5.6. Let C be an eDT or eNDT formula. $C[0/B]$ is the formula obtained by replacing (in parallel) each occurrence of a literal p as a leaf in C with the formula $(B\,p\,p)$. Similarly, $C[1/B]$ is the formula obtained by replacing each occurrence of a literal p as a leaf in C with the formula $(p\,p\,B)$.

The point of $C[0/B]$ is that $(B\,p\,p)$ evaluates to 1 if p is true, and to B otherwise. Thus, the intent is that $C[0/B]$ is equivalent $C \vee B$. Likewise, we want $C[1/B]$ to be equivalent $C \wedge B$. However, these equivalences hold only if the substitutions are applied not just in C but instead throughout the definitions of the extension axioms used in C. This is done with the following definition.

Definition 5.7. Let $\mathcal{A}$ be a set of extension axioms $\{e_i \leftrightarrow A_i\}_{i<n}$. Another set of extension axioms $\mathcal{A}[1/B]$ is defined as follows. First, let $\{e'_i\}_i$ be a set of *new* extension variables. Define $A_i[\vec{e}\,'/\vec{e}]$ to be the result of replacing each e_j in A_i with e'_j. Let A'_i be $(A_i[\vec{e}\,'/\vec{e}])[1/B]$. Then $\mathcal{A}[1/B]$ is the set of extension axioms $\{e'_i \leftrightarrow A'_i\}_{i<n} \cup \mathcal{A}$. The set $\mathcal{A}[0/B]$ is defined similarly: letting $\vec{e}\,''$ be another set of new extension variables, defining A''_i to be $(A_i[\vec{e}\,''/\vec{e}])[0/B]$, and letting $\mathcal{A}[0/B]$ be the set of extension axioms $\{e''_i \leftrightarrow A''_i\}_{i<n} \cup \mathcal{A}$.

Finally, if A and B are eDT or eNDT formulas defined using extension axioms $\mathcal{A}$, then $\textsc{And}(A,B)$ is by definition $A[1/B]$ relative to the extension axioms $\mathcal{A}[1/B]$. The formula $\textsc{Or}(A,B)$ for disjunction is defined similarly, namely, it is equal to $A[0/B]$ relative to the extension axioms $\mathcal{A}[0/B]$.

Note the two formulas $\textsc{And}(A,B)$ and $\textsc{Or}(A,B)$ introduced *different* sets of new extension variables. This allows us to use both $\textsc{And}(A,B)$ and $\textsc{Or}(A,B)$ without any clashes between extension variables. More generally, we will adopt the convention that the new extension variables are uniquely determined by the formula being constructed. In other words, for instance, e'_i could have instead been designated $e_{i,(A\wedge B)}$. When measuring proof size, we also need to count the sizes of the subscripts on the extension variables. This clearly however only increases proof size polynomially.

There are two other sources of growth of size in forming $\textsc{And}(A,B)$ and $\textsc{Or}(A,B)$. The first is that formula sizes increase since copies of B is substituted in at many places in A and $\mathcal{A}$: this potentially gives a quadratic blowup in proof size. We avoid this quadratic blowup in proof size, by always taking B to be a single variable (namely, an extension variable). The construction of $\textsc{And}(A,B)$ or $\textsc{Or}(A,B)$ also introduces many new extension variables, namely it potentially doubles the number of variables. To control this, we will ensure that the constructions of $\textsc{And}(\cdot,\cdot)$ and $\textsc{Or}(\cdot,\cdot)$ are nested only logarithmically.

Example 5.8. Consider the formula $\textsc{And}(p_1, \textsc{And}(p_2,p_3))$, which is a translation of the Boolean formula $p_1 \wedge (p_2 \wedge p_3)$ to a DT formula. To form $\textsc{And}(p_2,p_3)$, start with (p_2p_21) and substitute p_3 for "1", to obtain $(p_2p_2p_3)$. Then $\textsc{And}(p_1, \textsc{And}(p_2,p_3))$

is obtained by forming $(p_1 p_1 1)$ and replacing "1" with $\textsc{And}(p_2, p_3)$ to obtain $(p_1 p_1 (p_2 p_2 p_3))$. It is also the same as $\text{Conj}(p_1, p_2, p_3)$. A similar construction shows that $\textsc{Or}(p_1, \textsc{Or}(p_2, p_3)$ is equal to $((p_3 p_2 p_2) p_1 p_1)$. This is a translation of the Boolean formula $p_1 \vee (p_2 \vee p_3)$ to a DT formula, and is equal to $\text{Disj}(p_1, p_2, p_3)$.

Example 5.9. Let A be the formula $(p_1 p_2 (e_1 p_3 e_2))$ and B be the formula $(q_1 q_2 e_2)$ in the context of the extension axioms $\mathcal{A}$

$$e_1 \leftrightarrow (r_1 \overline{r_2} e_2) \qquad\qquad e_2 \leftrightarrow (\overline{s_1} s_2 s_3), \tag{20}$$

where p_i, q_i, r_i, s_i are literals. The formula $A[0/B]$ is formed as follows. First $\mathcal{A}(\vec{e}\,'/\vec{e})$ equals

$$e'_1 \leftrightarrow (r_1 \overline{r_2} e'_2) \qquad\qquad e'_2 \leftrightarrow (\overline{s_1} s_2 s_3)$$

Then $\mathcal{A}[0/B]$ contains the extension axioms of $\mathcal{A}$ as shown in (20) plus the extension axioms

$$e'_1 \leftrightarrow ((B r_1 r_1) \overline{r_2} e'_2) \qquad\qquad e'_2 \leftrightarrow ((B\, \overline{s_1}\, \overline{s_1}) s_2 (B s_3 s_3)).$$

Finally, $A[0/B]$ is the DT formula $((B p_1 p_1) p_2 (e'_1 p e'_2))$, namely,

$$(((q_1 q_2 e_2) p_1 p_1) p_2 (e'_1 p e'_2)),$$

relative to the four extension axioms in $\mathcal{A}[0/B]$.

5.3 Truth conditions and renaming of extension variables

We show that, despite the delicate renaming of variables required for notions such as $A[0/B]$ and $\textsc{And}(A, B)$, for DT (respectively NDT) formulas A, B, we may nonetheless realise their basic truth conditions by small eLDT (respectively eLNDT) proofs:

Lemma 5.10. *Let A and B be* eDT *formulas (respectively,* eNDT *formulas) relative to extensions axioms $\mathcal{A}$. Then, the sequents* (a)-(c) *below have polynomial size, cut free* eLDT *proofs (respectively,* eLNDT *proofs) relative to the extension axioms $\mathcal{A}[0/B]$. The same holds for the sequents* (d)-(f) *relative to $\mathcal{A}[1/B]$.*

(a) $B \longrightarrow A[0/B]$	(d) $A[1/B] \longrightarrow B$
(b) $A \longrightarrow A[0/B]$	(e) $A[1/B] \longrightarrow A$
(c) $A[0/B] \longrightarrow A, B$	(f) $A, B \longrightarrow A[1/B]$

Proof sketch. Parts (a)-(c) are proved by showing inductively that if C is a subformula of $A[0/B]$ or a subformula of any A'_i in $\mathcal{A}[0/B]$, then $C \longrightarrow A, B$ and $B \longrightarrow C$ and $A \longrightarrow C$ have short eLDT (resp., eLNDT) proofs. The base cases are just the cases where C is is the form $(B\, p\, p)$. The inductive cases are trivial. A similar argument proves cases (d)-(f). □

The proofs of Lemma 5.10 seem to be inherently dag-like, and we do not know if the lemma holds for Tree-eLDT.

As discussed above, we assume that the choice of new extension variables $\vec{e}'$ or $\vec{e}''$ depends explicitly on what formula $\textsc{And}(A, B)$ and $\textsc{Or}(A, B)$ is being formed. In other words, each e'_i or e''_i is a variable $e_{i,\textsc{And}(A,B)}$ or $e_{i,\textsc{Or}(A,B)}$. In the proof of Theorem 6.1, this means that the translations of distinct occurrences of the same Boolean formula use the same extension variables. However, this is not strictly necessary, as eLDT can prove the equivalence of formulas after a change in extension variables:

Lemma 5.11. *Suppose A is a* DT *formula w.r.t. extension axioms $\mathcal{A} = \{e_i \leftrightarrow A_i\}_i$, and that the extension variables $\vec{f}$ are distinct from the extension variables $\vec{e}$. Let B equal $A[\vec{f}/\vec{e}]$ w.r.t. the extension axioms $\mathcal{B} = \{f_i \leftrightarrow A_i[\vec{f}/\vec{e}]\}_i$. Then* eLDT *has a polynomial size, cut free (dag-like) proofs of $A \longrightarrow B$ and $B \longrightarrow A$ relative to the extension axioms $\mathcal{A} \cup \mathcal{B}$.*

Lemma 5.11 has a straightforward proof that proceeds inductively through all subformulas of the formulas A_i and A. □

6 Simulations for eLDT, eLNDT and LK

6.1 eLDT polynomially simulates LK

Theorem 6.1. eLDT *polynomially simulates* LK. *Hence,* eLNDT *also polynomially simulates* LK.

The intuition behind this theorem is that the formulas in an LK proof are Boolean formulas, and hence express NC^1 properties, while DT proofs work with DT formulas that express (nonuniform) logspace properties. Since Boolean formula evaluation can be done in logspace, it is expected that DT can directly simulate an LK proof. This is indeed how the proof goes, but it is complicated by the need to the $\textsc{And}$ and $\textsc{Or}$ constructions.

Proof. Suppose π is an LK proof of a sequent of Boolean formulas (possibly, but not necessarily of the form (8)). We wish to convert π into a eLDT proof. The main technique is to use the constructions $\textsc{And}$ and $\textsc{Or}$ of Definition 5.7 to convert the Boolean formulas in π into DT formulas over extension axioms. However, some care is needed to ensure that the resulting DT formulas and extension axioms are polynomial size.

For this, let $L(A)$ denote the *leaf size* of the formula A, namely the number of atomic subformulas of A. The leaf size $L(\mathcal{A})$ of a set of extension axioms is $\sum_i L(A_i)$. A straightforward analysis shows that Definition 5.7 constructs $\textsc{And}(A, B)$ to have leaf size $\leqslant L(A) \cdot (L(B) + 1)$, and $L(\mathcal{A}[1/B]$ to be $\leqslant L(\mathcal{A}) \cdot (L(B) + 2)$. To avoid too large formulas sizes, we will require that $L(B) = 1$. When this holds, we have $L(\textsc{And}(A, B)) \leqslant 2L(A)$ and $L(\mathcal{A})[1/B] \leqslant 3L(\mathcal{A})$. The same size bounds hold for $\textsc{Or}(A, B)$ of course.

The *height* of a Boolean formula A is the height of the binary tree corresponding to the formula A. Let's assume every formula in the endsequent of π has logarithmic height. Then by [36, 8], we may assume w.l.o.g. that every formula in π has height $O(\log|\pi|)$.[5] Each formula A in π is converted into a DT formula $\textsc{Dt}(A)$ with associated extension axioms $\mathcal{A}_A$ as defined next. The formula $\textsc{Dt}(A)$ will always be either a literal p or an extension variable e_A.

(a) Suppose A is a literal p, then $\textsc{Dt}(A)$ is just p, and $\mathcal{A}_p$ is empty (no extension axioms).

(b) If A is $B \wedge C$, then let $\textsc{Dt}(A)$ be the (new) extension variable e_A. Letting $\mathcal{A}'$ be $\mathcal{A}_B \cup \mathcal{A}_C$, set $\mathcal{A}_A$ equal to $\mathcal{A}'[1/C] \cup \{e_A \leftrightarrow \textsc{And}(\textsc{Dt}(B), \textsc{Dt}(C))\}$.

(c) The case where $B \vee C$ is exactly the same, but with $\mathcal{A}_A$ equal to $\mathcal{A}'[0/C] \cup \{e_A \leftrightarrow \textsc{Or}(\textsc{Dt}(B), \textsc{Dt}(C))\}$.

Recall the convention that the new extension variables introduced in cases (b) and (c) depend uniquely on A. This implies that every occurrence of a given formula A in the proof π has the identical translation $\textsc{Dt}(A)$. Furthermore, the formulas $\textsc{Dt}(A)$ and $\textsc{Dt}(B)$ share extension variable precisely to the extent that they share subformulas. More precisely, if C is a subformula of A, then $\textsc{Dt}(A)$ uses the extension variable e_C to denote the subformula C, using exactly the same extension axioms $\mathcal{A}_C$.

With these constructions, the LK proof π is translated to a DT proof by replacing every (Boolean) formula A in π with the DT formula $\textsc{Dt}(A)$ and using as extension axioms, the set $\bigcup_A \mathcal{A}_A$ where the union is taken over all formulas A appearing in π. This yields π', and we claim this can readily be fixed up to be a valid DT proof. For instance, an $\vee$-*r* in π

$$\vee\text{-}r\text{:}\ \frac{\Gamma \longrightarrow \Delta, A \qquad \Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, A \wedge B}$$

gets transformed to

$$\frac{\textsc{Dt}(\Gamma) \longrightarrow \textsc{Dt}(\Delta), \textsc{Dt}(A) \qquad \textsc{Dt}(\Gamma) \longrightarrow \textsc{Dt}(\Delta), \textsc{Dt}(B)}{\textsc{Dt}(\Gamma) \longrightarrow \textsc{Dt}(\Delta), \textsc{Dt}(A \wedge B)}$$

This can be fixed up to be a valid inference using cuts with the sequents $\textsc{Dt}(A), \textsc{Dt}(B) \longrightarrow \textsc{And}(\textsc{Dt}(A), \textsc{Dt}(B))$ and $\textsc{And}(\textsc{Dt}(A), \textsc{Dt}(B)) \longrightarrow \textsc{Dt}(A)$ and $\textsc{And}(\textsc{Dt}(A), \textsc{Dt}(B)) \longrightarrow \textsc{Dt}(A)$. These three sequents have polynomial size proofs by Lemma 5.10.[6]

The $\wedge$-*l*, $\vee$-*l* and $\vee$-*r* inferences in π are handled similarly. Other inferences in π are trivial to handle.

[5] We can also assume without loss of generality that π is a tree-like proof. This, however, does not help form a tree-like DT proof, since Lemma 5.10 uses dag-like proofs in an essential way.

[6] As stated in the previous footnote, this use of Lemma 5.10 is the reason the DT proof ends up dag-like instead of tree-like.

After fixing up the inferences in π' in this way, we obtain a valid DT proof π_1 of the sequent $\mathrm{DT}(\Gamma) \longrightarrow \mathrm{DT}(\Delta)$ where $\Gamma \longrightarrow \Delta$ is the final line of π.

For polynomial simulation, the last line of π is a sequent of the form (8), namely Γ is a multiset of disjunctions of literals, and Δ is a multiset of conjunctions of literals. Referring to Equation (8), a conjunct will $\bigwedge \vec{b}_i$ will have the conjunctions nested in a balanced fashion by our assumption that formulas in π have logarithmic height. However, it is straightforward to give a polynomial size, cut free DT proof of $\mathrm{DT}(\bigwedge \vec{b}_i) \longrightarrow \mathrm{Conj}(\vec{b}_i)$ for an arbitrary nesting of conjunctions in $\bigwedge \vec{b}_i$. Likewise, there are polynomial size, cut-free DT-proofs of $\mathrm{Disj}(\vec{a}_i) \longrightarrow \mathrm{DT}(\bigvee \vec{a}_i)$. Adding cuts with these to the end of π_1 gives the desired polynomial size DT proof of (9). □

6.2 LK **quasipolynomially simulates** eLNDT

The intuition for the next simulation is that eNDT formulas define nondeterministic logspace properties, and these are expressible with quasipolynomial size Boolean formulas.

Theorem 6.2. LK *quasipolynomially simulates* eLNDT*. As a result,* LK *also quasipolynomially simulates* eLDT*.*

Proof sketch. Suppose π is an eLNDT proof of a sequent $\Gamma \longrightarrow \Delta$ of eNDT formulas, and with associated extension axioms $\mathcal{A} = \{e_i \leftrightarrow A_i\}_{i \in I}$. We must construct an LK proof π' quasipolynomially simulating π. The idea for forming π' is to give truth definitions for all formulas appearing in π, and then prove that all sequents are in π are true under these truth definitions. The truth definition will be based on st-connectivity in a directed graph G_π. The nodes of G_π will be the subformulas of formulas in π or $\mathcal{A}$; the edges will be defined in terms of the literals p used in π. It is well-known that there are quasipolynomial formulas expressing st-connectivity in G_π. Furthermore, by [4], straightforward constructions of these quasipolynomial formulas can be used in LK proofs to prove basic properties of st-connectivity.[7]

We describe the direct graph G_π in more detail. Consider all distinct subformulas appearing either (a) in some formula A in π or (b) in some A_i from the extension axioms. These subformulas are vertices of the graph G_π. In addition, G_π contains one additional vertex, called 1. For example, suppose that the formula $A := (e_1\, \overline{p}\, p)$ appears in π and that $e_1 \leftrightarrow (\overline{q}\, p\, p)$ is an extension axiom in $\mathcal{A}$. These contribute the following nodes to G_π:

$$(e_1\, \overline{p}\, p), \qquad e_1, \qquad p, \qquad (\overline{q}\, p\, p), \qquad \overline{q}, \qquad \text{and} \qquad 1. \tag{21}$$

[7] The analogous results were earlier formulated within the bounded arithmetic theory U^1_2 by Beckmann-Buss [2]. U^1_2 has proof theoretic strength corresponding to polynomial space, or under the RSUV isomorphism to quasilogarithmic (that is, $(\log n)^{O(1)}$) space. Likewise, it corresponds to propositional provability with $2^{n^{O(1)}}$ size LK proofs, or under the RSUV isomorphism, with propositional provability with polynomial size LK proofs. This last claim does not appear explicitly in the literature, but see Dowd [19, 20] and Beckmann-Buss [3].

Enumerate the the vertices of G_π in any arbitrary order as $v_0, v_1, \ldots, v_m$, say with v_0 the vertex 1 and the rest of the vertices in arbitrary order. Note m is polynomially bounded (in fact, linearly bounded) by $|\pi|$.

The edges present in G_π are specified by Boolean formulas $\varphi_{i,j}$ for distinct i, j in $\{0, \ldots, m\}$, so that $\varphi_{i,j}$ is true if there is a directed edge from v_i to v_j in G_π. For a vertex v_i of G_π equal to a formula (ApB), and let the vertices v_j and $v_{j'}$ in G_π be the DT formulas A and B. Then $\varphi_{i,j}$ is the Boolean formula $\overline{p}$ and $\varphi_{i,j'}$ is the Boolean formula p. For vertex v_i equal to some e_k and vertex v_j equal to A_k, then $\varphi_{i,j}$ is the constant Boolean formula $\top$. Third, if the vertex v_i is a DT formula p with p a literal, then $\varphi_{i,0}$ is the Boolean formula p. All other formulas $\varphi_{i,j}$ are defined to equal the constant Boolean formula $\bot$. (Strictly speaking, $\top$ and $\bot$ are not allowed constants for Boolean formulas; instead, they stand for $(p \vee \overline{p})$ and $(p \wedge \overline{p})$ for some literal p.)

Returning to the example, let $v_1, \ldots, v_5$ be the five formulas in the order indicated in (21), and v_0 be 1. Then, $\varphi_{1,2}$ is p; $\varphi_{1,3}$ is $\overline{p}$; $\varphi_{2,4}$ is $\top$; $\varphi_{3,0}$ is p; $\varphi_{4,5}$ is $\overline{p}$; $\varphi_{4,3}$ is p; and $\varphi_{5,0}$ is $\overline{q}$.

Finally, for v_i a vertex in G_π, namely a subformula used in π, define Reach_i to be a Boolean formula expressing that there is a path in G_π from v_i to v_0. As discussed in [4], Reach_i can be expressed by a quasipolynomial size formula, and there are quasipolynomial size proofs of elementary properties of Reach_i, notably of

$$\mathit{Reach}_i \;\leftrightarrow\; \bigvee_{j \neq i} \big(\varphi_{i,j} \wedge \mathit{Reach}_j\big) \tag{22}$$

Each line in π is a sequent of the form

$$v_{i_1}, \ldots, v_{i_k} \longrightarrow v_{j_1}, \ldots, v_{j_\ell}.$$

To form the LK proof π', replace each such sequent with the quasipolynomial size sequent

$$\mathit{Reach}_{i_1}, \ldots, \mathit{Reach}_{i_k} \longrightarrow \mathit{Reach}_{j_1}, \ldots, \mathit{Reach}_{j_\ell}.$$

It is now easy to fix up π' be a valid LK proof. Initial sequents are handled trivially, since if v_i is p then Reach_i is also p. The only non-trivial inferences are decision rules *dec-l* and *dec-r* and these are readily handled with the aid of (22). □

7 Conclusions

This work presented sequent-style systems LDT, LNDT, eLDT and eLNDT that manipulate decision trees, nondeterministic decision trees, branching programs (via extension) and nondeterministic branching programs (also via extension) respectively. We examined their relative proof complexity and also compared them to (bounded depth) Frege systems (more precisely their representations in the sequent calculus).

In particular, since (nondeterministic) Branching Programs constitute a natural nonuniform version of (nondeterministic) L, the system eLDT (eLNDT) can be seen as a natural propositional system for (nondeterministic) logspace. This mimics the way that LK (or the Frege system) is a natural system for ALogTime (via Boolean formulas) and eLK (or extended Frege) is a natural system for P (via Boolean circuits).

We did not compare the proof complexity theoretic strength of our systems eLDT and eLNDT with the system for L in [11] and the systems for L and NL in [34, 35]. In future work we intend to show that our systems correspond to the bounded arithmetic theories VL and VNL, in the usual way. Namely, proofs of Π_1 formulas in VL translate to families of small eLDT proofs of each instance, and, conversely, VL proves the soundness of eLDT. Similarly for VNL and eLNDT. This would render our systems polynomially equivalent to their respective systems from [11, 34, 35], though this remains work in progress.

There are two natural open questions arising from this work. The first concerns the exact relationship between LDT and low-depth systems:

Question 7.1. *Does tree-*1-LK *polynomially simulate tree-*LDT*, or is there a quasipolynomial separation between the two?*

The second open question is whether tree-like systems for branching programs may polynomially simulate their corresponding dag-like ones.

Question 7.2. *Does tree-*eLDT *polynomially simulate* eLDT*? Similarly for* eLNDT

While well-defined, the systems tree-eLDT and tree-eLNDT do not seem very robust, in the sense that it is not immediate how to witness branching program isomorphisms with short proofs, cf. 5.11. Nonetheless, it would be interesting to settle their proof complexity theoretic status.

There has been much recent work on the proof complexity of systems that may manipulate OBDDs [27, 5, 23], a special kind of branching program where propositional variables must occur in the same relative order on each path through the dag. In fact, we could also define an 'OBDD fragment' of eLDT by restricting lines to eDT formulas expressing OBDDs, as alluded to in Example 5.9. It would be interesting to examine such systems from the point of view of proof complexity in the future, in particular comparing them to existing OBDD systems.

In this work we restricted the expressivity of all lines in a proof in order to define our various systems. An alternative approach is to restrict only the cut-formulas. Over conclusions of the appropriate form, this makes no difference to the notion of a proof thanks to the subformula property, but such systems have the advantage of being complete for all classes of formulas (for instance, via cut-free completeness). In this way we could have rather considered one single ambient system consisting of the connectives and rules for decision literals, disjunction and conjunction. Our various systems could thence be recovered by only restricting cut formulas. Many of our results already go through in this setting with respect to the provability of arbitrary formulas.

References

[1] T. Arai, *A bounded arithmetic AID for Frege systems*, Annals of Pure and Applied Logic, 103 (2000), pp. 155–199.

[2] A. Beckmann and S. R. Buss, *Improved witnessing and local improvement principles for second-order bounded arithmetic*, ACM Transactions on Computational Logic, 15 (2014). Article 2, 35 pages.

[3] ——, *The NP search problems of Frege and extended Frege proofs*, ACM Transactions on Computational Logic, 18 (2017), p. Article 11.

[4] S. Buss, *Quasipolynomial size proofs of the propositional pigeonhole principle*, Theoretical Computer Science, 576 (2015), pp. 77–84.

[5] S. Buss, D. Itsykson, A. Knop, and D. Sokolov, *Reordering rule makes OBDD proof systems stronger*, in 33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA, 2018, pp. 16:1–16:24.

[6] S. R. Buss, *Bounded Arithmetic*, Bibliopolis, Naples, Italy, 1986. Revision of 1985 Princeton University Ph.D. thesis.

[7] ——, *The Boolean formula value problem is in ALOGTIME*, in Proceedings of the 19-th Annual ACM Symposium on Theory of Computing, May 1987, pp. 123–131.

[8] ——, *Propositional consistency proofs*, Annals of Pure and Applied Logic, 52 (1991), pp. 3–29.

[9] ——, *Cut elimination in situ*, in Genzten's Centenary: The Quest for Consistency, R. Kahle and M. Rathjen, eds., Springer Verlag, 2015, pp. 245–277.

[10] S. R. Buss and P. Pudlák, *How to lie without being (easily) convicted and the lengths of proofs in propositional calculus*, in Proceedings of the 8th Workshop on Computer Science Logic, Kazimierz, Poland, September 1994, L. Pacholski and J. Tiuryn, eds., Lecture Notes in Computer Science #933, Berlin, 1995, Springer-Verlag, pp. 151–162.

[11] S. A. Cook, *A survey of complexity classes and their associated propositional proof systems and theories, and a proof system for log space.* Talk presented at the ICMS Workshop on Circuit and Proof Complexity, Edinburgh, October 2001. http://www.cs.toronto.edu/ sacook/.

[12] ——, *Feasibly constructive proofs and the propositional calculus*, in Proceedings of the Seventh Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, 1975, pp. 83–97.

[13] S. A. Cook and A. Kolokolova, *A second-order system for polytime reasoning based on Grädel's theorem*, Annals of Pure and Applied Logic, 124 (2003), pp. 193–231.

[14] ———, *A second-order theory for NL*, in Proc. 19th IEEE Symp. on Logic in Computer Science (LICS'04), 2004, pp. 398–407.

[15] S. A. Cook and T. Morioka, *Quantified propositional calculus and a second-order theory for NC^1*, Archive for Mathematical Logic, 44 (2005), pp. 711–749.

[16] S. A. Cook and P. Nguyen, *Foundations of Proof Complexity: Bounded Arithmetic and Propositional Translations*, ASL and Cambridge University Press, 2010. 496 pages.

[17] S. A. Cook and R. A. Reckhow, *On the lengths of proofs in the propositional calculus, preliminary version*, in Proceedings of the Sixth Annual ACM Symposium on the Theory of Computing, 1974, pp. 135–148.

[18] ———, *The relative efficiency of propositional proof systems*, Journal of Symbolic Logic, 44 (1979), pp. 36–50.

[19] M. Dowd, *Propositional representation of arithmetic proofs*, in Proceedings of the 10th ACM Symposium on Theory of Computing (STOC), 1978, pp. 246–252.

[20] ———, *Propositional Representation of Arithmetic Proofs*, PhD thesis, Dept. of Computer Science, University of Toronto, 1979.

[21] E. Grädel, *Capturing complexity classes by fragments of second order logic*, Theoretical Computer Science, 101 (1992), pp. 35–57.

[22] N. Immerman, *Nondeterministic space is closed under complement*, Tech. Rep. DCS/TR552, Yale University, July 1987.

[23] D. Itsykson, A. Knop, A. E. Romashchenko, and D. Sokolov, *On obdd-based algorithms and proof systems that dynamically change order of variables*, in 34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany, 2017, pp. 43:1–43:14.

[24] E. Jeřábek, *Dual weak pigeonhole principle, Boolean complexity, and derandomization*, Annals of Pure and Applied Logic, 124 (2004), pp. 1–37.

[25] J. Johannsen, *Satisfiability problem complete for deterministic logarithmic space*, in Proc. 21st Symp. on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science 2996, Springer, 2004, pp. 317–325.

[26] S. Jukna, A. A. Razborov, P. Savický, and I. Wegener, *On P versus NP $\cap$ co-NP for decision trees and read-once branching programs*, Computational Complexity, 8 (1999), pp. 357–370.

[27] A. Knop, *IPS-like proof systems based on binary decision diagrams.* Typeset manuscript, June 2017.

[28] J. Krajíček, *Bounded Arithmetic, Propositional Calculus and Complexity Theory*, Cambridge University Press, Heidelberg, 1995.

[29] ——, *Proof Complexity*, Cambridge University Press, 2019.

[30] J. Krajíček and P. Pudlák, *Quantified propositional calculi and fragments of bounded arithmetic*, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik, 36 (1990), pp. 29–46.

[31] J. Krajíček and G. Takeuti, *On induction-free provability*, Annals of Mathematics and Artificial Intelligence, (1992), pp. 107–126.

[32] R. E. Ladner, *The circuit value problem is log space complete for P*, SIGACT News, 7 (1975), pp. 18–20.

[33] J. B. Paris and A. J. Wilkie, *Counting problems in bounded arithmetic*, in Methods in Mathematical Logic, Lecture Notes in Mathematics #1130, Springer-Verlag, 1985, pp. 317–340.

[34] S. Perron, *A propositional proof system for log space*, in Proc. 14th Annual Conf. Computer Science Logic (CSL), Springer Verlag Lecture Notes in Computer Science 3634, 2005, pp. 509–524.

[35] ——, *Power of Non-Uniformity in Proof Complexity*, PhD thesis, Department of Computer Science, University of Toronto, 2009.

[36] R. A. Reckhow, *On the Lengths of Proofs in the Propositional Calculus*, PhD thesis, Department of Computer Science, University of Toronto, 1976. Technical Report #87.

[37] R. Szelepcsényi, *The method of forcing for nondeterminsitic automata*, Bulletin of the European Association for Theoretical Computer Science, 33 (1987), pp. 96–99.

[38] G. S. Tsejtin, *On the complexity of derivation in propositional logic*, Studies in Constructive Mathematics and Mathematical Logic, 2 (1968), pp. 115–125.

[39] I. Wegener, *Branching Programs and Binary Decision Diagrams*, SIAM, 2000.