Proof Complexity of Systems of Branching Programs

By

Avgerinos Delkos

A thesis submitted to
the University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
December 2023

Abstract

This work investigates the proof complexity of multiple systems about deterministic and non-deterministic branching programs (BPs and NBPs respectively). It is based on work by Buss, Das and Knop, where they defined the systems $\mathsf{eL(N)DT}$ reasoning with (N)BPs by using extension variables/axioms to represent the dag-structure, over a language of (non-deterministic) decision trees.

We start by focusing on positive branching programs (PBPs) i.e. NBPs where, for any 0-transition between two nodes, there is also a 1-transition. PBPs compute monotone Boolean functions, much like negation-free circuits or formulas do, but constitute a positive version of (non- uniform) $\mathbf{NL}$, rather than $\mathbf{P}$ or $\mathbf{NC}^1$, respectively. We introduce eLNDT^+, the positive fragment of eLNDT, that reasons about PBPs by considering restrictions on the form of inference rules we consider. The main result is that eLNDT^+ polynomially simulates eLNDT over positive sequents. Our proof method is inspired by a similar result for MLK ($\neg$-free LK) and LK by Atserias, Galesi and Pudlák. Along the way we formalise several properties of counting functions within eLNDT^+ by polynomial-size proofs and, as a case study, give explicit polynomial-size proofs of the propositional pigeonhole principle.

Our second contribution aims at defining Prover-Adversary games bespoke to the setting of branching programs. It builds upon work by Pudlák and Buss who defined a game with Boolean formulas as queries that corresponds to Hilbert-Frege/LK i.e. the canonical systems for reasoning about Boolean formulas. We adopt their definition to fit in our setting and provide games that correspond to eLDT and eLNDT. While the deterministic case is simple enough, the non-deterministic one requires simulating negations of NBPs for which, through a series of technical results, we formalize a (partial) non-uniform version of the Immerman-Szelepcsényi Theorem: $\mathbf{NL} = co\mathbf{NL}$.

The last part of this thesis presents the system $\mathbf{o}$-eLDT, a specifically designed fragment of eLDT that reasons about ordered BPs (OBDDs). These are BPs whose paths only exhibit variables under a fixed order. Due to their 'restricted' nature, they posses multiple nice properties, for example, easy checking for equivalence between OBDDs. They have recently received wide interest in the literature with their proof (complexity) theoretic analysis initiated by work from Atserias, Kolaitis and Vardi where they, among others, defined the system $\mathsf{OBDD}(\wedge, \mathsf{w})$. We finish by comparing the expressive strength of $\mathbf{o}$-eLDT and $\mathsf{OBDD}(\wedge, \mathsf{w})$, concluding that the former polynomially simulates the latter.

Acknowledgements

Formal acknowledgements

I would like to express my deepest gratitude to all people and entities that have helped me see through this PhD. More specifically, I would like to start by thanking my supervisor, Anupam Das, whose guidance, knowledge and attention to detail has helped me improve as a researcher and author. I want to thank in general people of the theory group (especially my RSMG) for their assistance and advice throughout my three and a half years in Birmingham. It was a pleasure to be part of this community with a lot of research and interesting topics being discussed. Lastly, I want to thank all my friends and also University of Birmingham for funding my PhD.

Personal acknowledgements

While what written above is true, it lacks personal nuance which I will now add for completeness sake. First off big thanks to my supervisor, Anupam Das, for being rich (grant wise). Without all these academic travels in interesting destinations my PhD life would have been much less enjoyable. In addition to how much I liked visiting new places for schools and conferences, I met a multitude of unique people that do interesting research and spending 'real' time together helped me get to know them. Having academic discussions can be very fun but having discussions with academics about every day things (arguing about where to eat) or arbitrary existential topics (how would you suicide) is often even better. On that end, I want to thank my partners in exploration in inverse alphabetical order: Marco, Lukas and Alakh-Akbar for exploring places, many we shouldn't be at (University of Milan), and getting kicked out by security multiple times.

A large thank you to everyone that helped me with survival. First, I include names in descending hosting-amount order: Abhishek ντε, Gianluc-lukum and Ayberkius.

Thank you to all who helped me by keeping my stuff safe, I list in random order: basement of my old place[1], Strateusimos, Chris, Sonia and Vincent in Birmingham, while: my dramatic pet, Dario of life, Kirstinenburg, Kafa and Artem (Deported) elsewhere.

A lot of appreciation goes to my collaborators (not just pertaining to this work) for being patient with my not perfectly explained ideas and forcing me to be more formal. Anupam's (academic) 'OCD' has certainly helped me become much better at expressing myself at a formal level (and not leaving double spaces in my latex code)

[1] Not a person

while, Marianna's (confused) comments on my ideas about counterfactuals, forced me to refine my definitions/notation to achieve readable levels.

I enjoyed TAing on the MLFCS module for three years. It was quite fun and for the most part well organised. Hence a big thank you to all my colleagues and students who appreciated my teaching (or took photos with my hair) is warranted.

A decently sized thanks goes to people in the Theory group for the fun discussions on random topics and having (group wide) long arguments via university email.

I should not forget to thank people in our department dealing with administrative/bureaucratic stuff that nobody likes. There are multiple people in this category but Kate certainly deserves an honorable mention. Another one would be my 'enemy by mail'/ 'friend in real life' Felipe.

Big thanks goes to friends all over who were willing to invite and host me including, in alphabetical order: Dellica, Lio de Janeiro, Kompo, Darmon and Ton Dicks. Visiting and spending time with friends played a huge role in me enjoying these past years and I am thankful for having the opportunity to do so.

This degree has been partially (and unofficially) sponsored by multiple supermarket chains. I am grateful to them all for their free resources.

There are probably more friends and acquaintances that I should mention for being there for me (or some people that deserve it multiple times) but I will stop here. I wish them to get thanked in many more theses to come.

Contents

Chapter 1

Introduction

Proof theory is the study of mathematical proofs treated as formal mathematical objects. This analysis was first necessitated by the rigorous formalization of the foundations of mathematics occurring in the early 1900's (Hilbert's program) [1]. Proofs are commonly represented linearly, as lists of statements or graphically, as trees or directed acyclic graphs whose nodes are statements. There, the final statement of the list or the root of the graph respectively serves as the *conclusion* of the proof. Informally, proof systems are sets of initial statements called axioms and rules that enable 'combining' statements and adding new steps in the proof. Some of the most studied types of proof systems are the Hilbert-Frege proof systems and Gentzen's sequent calculus LK (for a comprehensive introduction to proof theory see [21]).

Computational complexity is a field of interest to theoretical computer science. It aims to determine and classify the complexity of problems regarding finite data structures and compare the relative efficiency of algorithms solving said problems. Some typical examples of such problems are the boolean satisfiability problem (SAT), reachability of nodes in a graph, the travelling salesman problem (TSP) and prime factorisation of integers.

Studying the complexity of Boolean functions finds application in several branches of computer science. Researchers in general aim to understand the complexity of useful basic algorithms tackling arithmetic problems like addition, multiplication, squaring, division or counting which can be represented by Boolean functions. Some early works serving as introduction to Boolean function complexity are [49] and a more extensive one being [62].

Boolean circuits, Boolean formulas, decision trees and branching programs are well known models of computation commonly used to (non-uniformly) compute Boolean functions. They are commonly visualized as different types of acyclic graphs whose nodes range over Boolean connectives, variables or constants. For a thorough introduction to computational complexity see [48] and for an introduction to decision trees, branching programs their many variations and properties see [63].

Material in this thesis finds itself in the intersection of the worlds of proof theory and complexity theory. More specifically we study the complexity of proof systems reasoning about formula representations of branching programs.

1.1 Proof Complexity

Proof complexity studies the size of formal proofs with respect to the theorem proved. A well known result is that Gentzen's sequent calculus is 'polynomially equivalent' to Hilbert-Frege systems that is, a proof of some theorem in sequent calculus can be converted to a proof of the same theorem in any Hilbert-Frege style system in polynomial time and vice versa. The original motivation for the development of proof complexity is due to a theorem by S. Cook and R. Reckhow [30]. Loosely stated, the theorem says that finding superpolynomial lower bounds[1] on the size of proofs for propositional logic, directly implies $\mathbf{P} \neq \mathbf{NP}$. This gave rise to 'Cooks-program': prove superpolynomial bounds for increasingly stronger propositional proof systems until a general method is found. Multiple kinds of 'hard'-tautologies have been studied in the pursuit of general superpolynomial bounds. Notable mentions are 'Tseitin formulas', clique-coloring tautologies and the pigeonhole principle. Tseitin formulas [59], are conjunctions of connectivity conditions between nodes in a given graph. They describe parity constraints corresponding to the structure of the underlying graph and are usually written in conjunctive normal form (CNF). Clique-coloring tautologies are pairs of clauses that express the property of a graph containing a clique of a certain size versus the property of the graph being k-colorable for k some natural number. A common way to understand the pigeonhole principle is as a propositional formula that expresses the (unsatisfiable) property of mapping a set of $n+1$ elements to a set of n elements in an injective manner.

It is typical for systems in proof complexity to be parametrized by a complexity class of interest whose nonuniform counterpart comprises the objects of reasoning for the associated proof system. For example, Hilbert-Frege systems reason about boolean formulas, the nonuniform counterpart of **ALOGTIME** [19]. For the class **P** the corresponding system is *extended* Frege, employing 'Tseitin extension' to represent the dag structure of circuits. Originally, Tseitin in [59] introduced the extension rule for resolution proof systems and it was later applied in the context of Hilbert-Frege systems by Cook and Reckhow in [30]. A standard textbook in proof complexity is [44].

Monotone proof complexity

Monotone proof complexity investigates the complexity of propositional proof systems under some notion of negation-freeness. Research in monotone proof complexity was initially motivated by the famous lower bound result of Razborov [53, 54], showing that the clique function is not computable by polynomial size negation-free circuits. As a consequence of that, there has been substantial study of the negation-free fragment of Gentzen's sequent calculus LK, called MLK.

Starting from [7], the authors obtained quasi-polynomial upper bounds for the pigeonhole principle for $n+1$ pigeons and n holes (expressed as a monotone sequent in MLK). Generalising this approach, in [8] they showed that MLK effectively simulates LK proofs of monotone sequents, with quasi-polynomial increase in size while only polynomial increase in the number of proof steps. It was years later in Jerábek's work [39]

[1] Superpolynomial increase of a quantity w.r.t. original size N is not bounded by any polynomial with input the size N. Quasi-polynomial increase is of the rate: $n^{O((\log n)^c)}$ for some constant c.

where it was shown that under the assumption of existence of suitable expander graphs, MLK polynomially simulates LK. Said assumptions were proven in [24] formalising the expander-graphs from [39] used to construct the AKS sorting networks [4] and finally showing that MLK polynomially simulates LK (on monotone sequents) and improving the result of [8].

Proof systems of decision trees and branching programs

Decision trees are rooted 'tree-like' data structures commonly used to display algorithms requiring multiple decisions and calculating event outcomes. They are subsumed by branching programs (BPs), also called binary decision diagrams (BDDs) [63] which are directed acyclic graphs (dags) and have more expressive power. BPs and their many variations and corresponding properties have received increasing attention over the years. They are typically pictured as directed acyclic graphs with two sink nodes 0 and 1, inner nodes labelled by propositional variables and one root node. A particularly interesting class is the *ordered binary decision diagrams* (OBDDs) first introduced by Bryant in [16] where he imposed a fixed order on the input variables of Boolean functions. Essentially, OBDDs are deterministic branching programs in which variables (labelling nodes) occur (only once) in the same relative order in each branch. They enjoy many nice properties for example, under a certain ordering of variables, to each Boolean function corresponds a unique OBDD of minimal size computing it. Decision trees and branching programs have recently received proof theoretic treatments. We mention some notable works in the following.

S. Cook in his unpublished work [27], designed specific 'Prover-Adversary games' (first introduced by [52]) where queries are deterministic branching programs. He essentially constructed a proof system corresponding (non-uniformly) to the complexity class *logspace* (**L**) by designing it to reason with deterministic branching programs instead of the original approach of boolean formulas.

In [9], the authors aim to introduce natural ways of defining a proof system corresponding to each constraint-satisfaction problem. As a case study they investigate the proof system $\mathsf{OBDD}(\wedge, \mathsf{w})$ which reasons about *ordered binary decision diagrams* (OBDDs); in it, a proof step can only be a 'conjunction/join' step on two previously derived OBDDs or a 'weakening' step introducing a new OBDD that is semantically implied by a previous one. They compared the strength of their systems to other well-known ones, such as resolution, the Gaussian calculus, cutting planes, and Hilbert-Frege systems of bounded alternation-depth (the definitions of these systems can be found in [42]).

Building on these results, in [23] it is shown that $\mathsf{OBDD}(\wedge, \mathsf{w})$ can give exponentially shorter proofs than (dag-like) cutting planes. They also added a reordering rule that allows changing the variable order for OBDDs and proceeded to show that $\mathsf{OBDD}(\wedge, \mathsf{w}, \mathsf{ord})$ is strictly stronger than $\mathsf{OBDD}(\wedge, \mathsf{w})$.

Recently, Buss, Das and Knop [22], proposed a proof complexity theory of **L** and **NL** via first defining the systems $\mathsf{LDT}, \mathsf{LNDT}$ reasoning about deterministic and non-deterministic decision trees respectively. Utilizing formulas with extension, they introduce the 'extended' versions of said systems, eLDT and eLNDT, reasoning about deterministic branching programs (BPs) and non-deterministic branching programs (NBPs) respectively. Proofs in these systems are parametrized by sets of 'extension axioms' and 'extension variables' to essentially abbreviate complex formulas corresponding to nodes

in branching programs (in this manner 'node sharing' is permitted while preventing repetitions and avoiding superpolynomial increase in the size of formulas expressing BPs). The only logical rules in $\mathsf{LDT}, \mathsf{eLDT}$ are right and left 'decisions' that introduce an 'if then else' clause upon a propositional variable p written ApB:

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

A *decision* ApB should be understood as "if p then B else A". The systems $\mathsf{LNDT}, \mathsf{eLNDT}$ also admit the standard rules for disjunction. While no system admits conjunction natively, it is possible (under careful bookkeeping) to introduce complex extended formulas expressing the conjunction of branching programs.

In [11], Barrett and Guglielmi introduced a proof system for classical propositional logic that also reasons about decision trees. They do this by augmenting the standard language for propositional logic with a 'decision connective' similar to what is seen in [22]. Their approach is a generalisation of a method used in subatomic logic [60], where literals are considered self-dual non-commutative connectives: $0a1$ instead of a and $1a0$ instead of $\overline{a}$. Intuitively, these can be seen as simple decision trees and thus including the more general case AaB in the language, was a natural choice. Their work has proof theoretic motivations: first, enhancing the standard language for classical propositional logic means their proof system admits more proofs, some of which are provably shorter than any sequent calculus proof of the same tautology. Secondly, the inference rules in their system are linear in the usual sense encountered in structural proof theory: rules use the same variables in the premise and the conclusion, implying, that all rules can be presented in a 'medial shape'. This is then used to achieve a novel method of eliminating cuts via a simple procedure.

1.2 Outline of contributions

We mostly work with sequent-type proof systems that reason about formula representations of branching programs within the framework introduced by Buss, Das and Knop [22]. This thesis contains three main contributions: first we investigate the proof complexity of a restricted version of the system eLNDT where rules and formulas in a proof are 'positive', second we design Prover-Adversary games whose queries are e(N)DT formulas characterising the systems $\mathsf{eL(N)DT}$ and third we introduce the fragment of eLDT that reasons about OBDDs.

1. Positive proofs

Positive branching programs (PBPs) are non-deterministic branching programs where for every 0-edge from a node u to a node v there is a 1-edge from u to v. They compute monotone boolean functions, where flipping a 0 bit to a 1 bit in the input cannot decrease the output value. The first part of this work, Chapters 3, 4 and 5 is dedicated to studying eLNDT^+ the 'positive' fragment of eLNDT, reasoning about positive branching programs by requiring decisions to be in positive form, $Ap(A \vee B)$. Naturally, the decision rules of eLNDT^+ are also adapted to have positive syntax. The

main goal of this part is to show a polynomial simulation of eLNDT by eLNDT^+ over positive sequents, where all decisions are positive. The method we use is inspired by [8] where the authors showed quasi-polynomial equivalence of LK by its negation-free fragment, MLK. They first defined a proof system polynomially equivalent to LK but with negations 'pushed' to the atomic level. Following this, they define explicit monotone formulas (pseudocomplements) which simulate negated atoms and 'substitute' them by specific formulas in proofs. They crucially use (quasipoly-size) formula representations of *threshold functions* Th^n_k, monotone boolean functions which output 1 if and only if at least k of the n input bits are 1 (their proof finishes with linearly many cut steps to merge proofs and obtain the required result).

While we borrow some techniques and high level structure from [8], we need to heavily adjust them due to the peculiarities of our setting. Firstly, the 'non-duality' of the positive decision rules makes 'pushing negation to atoms' hard. To resolve this we define an intermediate proof system, an extension of eLNDT^+ which admits negated literals but whose language has formulas only appearing in a 'well behaved' normal form. Another issue is caused by the technicalities encountered when substituting negated literals with extended formulas computing threshold functions: the systems we work with do not permit decisions made upon complex formulas. We remedy this by carefully defining new extension variables and axioms encoding positive decisions made upon extended threshold formulas. We provide explicit proofs of all the 'truth conditions' our formulas should satisfy which are crucial in obtaining the desired result.

2. Alternative approaches and gaming

In many of the aforementioned works along with this thesis, reasoning about (families of) propositional proofs can be tedious and notationally heavy. This is frequent in proof complexity and can be made worse when one employs extension to represent objects with underlying dag structures as we do from Section 2.1.2 onwards. The subscripting condition of such structures can be subtle and must be carefully controlled to maintain well-foundedness, especially in cases where we substitute such structures into one another, see Section 5.4. Using extension can also further complicate the techniques used to handle equivalence or simulation between programs as will be required in Section 6.2.

A potential solution to these issues (that lies beyond the scope of this work) is the program of *bounded arithmetic* [18, 43, 29, 42]. There, (very) weak theories of arithmetic serve as uniform counterparts of propositional proof systems. We adopt a different approach and employ *Prover-Adversary games*, a well known class of games in the proof complexity literature. First formally described by Pudlák and Buss in [52] for the case of boolean formulas, they were construed as a counterpart to LK. One of the reasons they introduced these games is to prove lower bounds for Hilbert-Frege systems and their restricted versions by investigating the relation of the number of rounds in the game to the number of steps in LK (thus also Hilbert-Frege) proofs. They showed that the minimal number of rounds in a 'winning Prover-strategy' is proportional to the logarithm of the minimal number of steps in an LK proof.

These games involve two players, Prover asks queries (the objects of reasoning) and Adversary answers by assigning a value to the query, 0 or 1. Prover wins if they can force Adversary into a 'simple contradiction' induced by the corresponding proof system, (which are commonly witnessed by polynomial-size proofs). This allows recasting proofs

as *strategies*, whose depth (maximal number of rounds) is proportional to the logarithm of the number of steps in the proof. Through this lens, such games can be seen as canonical 'balanced tree-like' versions of their corresponding proof systems, [41].

Following Cook's (unpublished) idea [27] and within the framework from [22], in Chapter 6 we define the games DB, NB, which admit Boolean combinations of branching programs, non-deterministic branching programs as queries respectively. The main result of this part is the polynomial equivalence of eLDT with DB and eLNDT with NB. While the deterministic case is straightforward, the non-deterministic one requires simulating the negation of non-deterministic branching programs. We achieve this by formalising a non-uniform version of the Immerman-Szelepcsényi theorem, $co\mathbf{NL} = \mathbf{NL}$ [37, 58]. Our (partial) formalization is different in that, contrary to the original proof, the inductive counting of our argument is not encoded into the NBPs we construct but instead, in our proofs. That is, as seen in Chapter 6, we consider families of proofs parametrised by a counter k and conduct our case analysis at the level of proofs. To achieve this we rely on positive constructions akin to what was introduced in earlier Chapters 3 and 4.

3. Systems for OBDDs

The last contribution of this work is involved with defining proof systems that reason about OBDDs much like Hilbert-Frege/LK reasons about Boolean formulas. A natural candidate to look into is the system eLDT which canonically reasons about deterministic branching programs. We define its fragment, **o-**eLDT, a system about OBDDs by carefully crafting 'leveled' extension axioms sets that respect a fixed order on propositional variables. This way, **o-**eDT formulas defined over leveled sets of extension axioms represent OBDDs and every OBDD can be represented (uniquely) by an **o-**eDT formula. We finish this last part by comparing our newly defined system to previous work in the literature, specifically the system $\mathsf{OBDD}(\wedge, \mathsf{w})$ from [9] and [23]. Since our framework is different that is, $\mathsf{OBDD}(\wedge, \mathsf{w})$ is a resolution-like refutation system while **o-**eLDT is a sequent calculus type of system, we define the necessary technical tools to be able to 'translate' $\mathsf{OBDD}(\wedge, \mathsf{w})$ refutations into **o-**eLDT proofs. Chapter 7 finishes with the polynomial simulation of $\mathsf{OBDD}(\wedge, \mathsf{w})$ by **o-**eLDT.

Previously published results

Chapters 2,3,4 and 5 heavily rely upon material from [31]. Chapters 6 and 7 contain mostly unpublished material (at the time of submittal of this work).

Chapter 2

Preliminaries

In this chapter we lay out the necessary foundations for the rest of this thesis. Most notions related to proof complexity are standard with slight deviations that better suit our setting. Material pertaining to branching programs is defined akin to [22]. The final part of the preliminaries is about Prover-Adversary games, where we provide definitions and results from [52] and also introduce a more general version of the games in which queries are not necessarily Boolean formulas.

Throughout this work we make use of a countable set of **propositional variables**, written p, q etc., and two **Boolean constants** 0 and 1. **Assignments** are maps from propositional variables to $\{0,1\}$, the set of assignments is denoted by Ass. An assignment α is extended to constants in the natural way, by setting $\alpha(0) = 0$ and $\alpha(1) = 1$. We shall work with assignments that have finite support, i.e. being nonzero only on variables occurring in a formula or proof of interest. For a fixed list of variables $\mathbf{p} = p_1, \ldots, p_n$ we write $\mathsf{Ass}_{\mathbf{p}}$ for the set of assignments on $\mathbf{p}$. **Boolean functions** will come equipped with a list of propositional variables $\mathbf{p}$ and are defined to be maps from $\mathsf{Ass}_{\mathbf{p}}$ (the function's finite domain) to $\{0,1\}$. Boolean functions are often alternatively defined as maps from $\{0,1\}^n \to \{0,1\}$ for $n \in \mathbb{N}$. In this work we may consider either definition taking care to be clear when doing so.

A **polynomial-time** function f is a function for which there exists a deterministic Turing machine M such that on input x, $M(x) = f(x)$ and the length of the computation takes time polynomial in the length of x.

2.1 Proof Complexity

Formal proof systems were first defined by S. Cook and R. Reckhow in [30]; the definition we use is equivalent but slightly altered for better exposition. For Σ, Σ_1 finite alphabets and Σ^*, Σ_1^* the sets of finite words from Σ, Σ_1 respectively, a **proof system** for a language $L \subseteq \Sigma^*$ is a polynomial-time function P from Σ_1^* to L.[1]

Intuitively, the elements $\sigma \in \Sigma_1^*$ code proofs in the system, while P itself is a (efficient) 'proof-checking' algorithm that verifies that σ is a correctly written proof in which case, it returns its conclusion i.e. the theorem it proves. If not, it returns 1 (w.l.o.g. we consider 1 an element of L).

[1] In proof complexity proof systems reason about **coNP**-complete languages, motivated by [30].

We write TAUT for the set of propositional tautologies over the basis $\{1, \wedge, \vee, \neg\}$ and in what follows, $L = \mathsf{TAUT}$ (as is usually the case in proof complexity and in this work). A (propositional) proof system P is called **sound** if the conclusion of any P proof is valid i.e. a tautology, while P is called **complete** if for each tautology $\phi \in \mathsf{TAUT}$, there is a P proof concluding with ϕ.

Remark 2.1.1. Setting L to be TAUT 'insists' our proof systems be sound. Requiring P to be surjective further implies that it is a complete proof system.

The following result from [30] shows the significance of this definition:

Theorem 2.1.2 (Cook-Reckhow)**.** *There is a propositional proof system with polynomial-size proofs of each tautology if and only if* **coNP** = **NP**.

The above 'Cook-Reckhow definition' we use covers all well-studied proof systems for propositional logic, under suitable codings. It is beyond the scope of this thesis to provide any of these codings explicitly. Instead, we leave it implicit that the proofs in the systems we consider are polynomial-time proof checkable in the way described above, and thus constitute formal propositional proof systems as is standard in proof complexity.

Definition 2.1.3 (Simulation)**.** We say that a proof system P' **polynomially simulates** a proof system P if there is a polynomial-time function f such that if π is a P proof of some $\phi \in \mathsf{TAUT}$ then $f(\phi)$ is a P' proof of ϕ. If two proof systems polynomially simulate each other we call them **polynomially equivalent**.

2.1.1 Branching programs

A **deterministic branching program** (deterministic BP or just BP) is a finite directed acyclic graph G with up to two distinct **sink** nodes $0, 1$ such that:

- G has a unique root node, i.e. a unique node with no incoming edges.
- Each non-sink node of G is labelled by a propositional variable.
- Each non-sink node has exactly two outgoing edges labelled by the constants 0 and 1 respectively.[2]

A more general notion is a **non-deterministic branching program** (NBP), a branching program where non-sink nodes can have additional (i.e. more than two) outgoing edges, labelled by 0 or 1. That is, a node may have multiple edges with the same label. In this work, when represented graphically, 0-edges will appear dotted while 1-edges solid and NBPs may have multiple 1 and 0 sink nodes to ease readability, see for example Figure 2.1.

A **run** of an NBP G on an assignment α is a maximal path beginning at the root of G consistent with α. That is, at a node labelled by p, the run must follow an edge labelled by $\alpha(p) \in \{0, 1\}$. G **accepts** α if there is a run on α reaching the 1 sink. We

[2]The (trivial) BPs, 0 and 1, will be comprised of only one node (functioning as both sink and root) labelled by 0 or 1 respectively.

may extend α to a map from all NBPs to $\{0,1\}$ by setting $\alpha(G) = 1$ just if G accepts α. In this way, each NBP **computes** a unique Boolean function $\alpha \mapsto \alpha(G)$. Notice that for NBPs, G accepts as long as **there is** a run reaching the 1 sink i.e. they employ existential non-determinism.

Remark 2.1.4. The dual concept, universal non-determinism i.e. accepting just if all runs reach the 1 sink would define *co*-non-deterministic BPs but these will not be considered in this work.

An in depth introduction to many variants of branching programs and their properties can be found in [63].

2.1.2 Representation of NBPs by extended formulas

Our syntactic representation of NBPs proceeds similarly to [22] using 'formulas with extension' when working with formal proof systems. In our case, it may be convenient to consider multiple slightly different grammars depending on our setting.

First appearing in [22], **deterministic decision tree** formulas, or **DT** formulas, written A, B etc. formally represent (deterministic) decision trees and are generated from the following grammar:

$$A, B ::= \ p \ \mid \ \overline{p} \ \mid \ ApB \ \mid \ A\overline{p}B \tag{2.1}$$

where $\overline{p}$ is the **dual** of p. A **decision**, ApB, should be semantically interpreted as 'if p then B else A'. Including negative literals and not the usual constants 0 and 1, is a stylistic choice made in [22] notice though, that one may identify 0 with $pp\overline{p}$ and 1 with $\overline{p}pp$. Furthermore a decision $A\overline{p}B$ is logically equivalent to BpA.

Non-deterministic decision tree formulas, or **NDT** formulas, written A, B etc. formally represent non-deterministic decision trees and are generated from the following grammar:

$$A, B ::= \ p \ \mid \ \overline{p} \ \mid \ ApB \ \mid \ A\overline{p}B \ \mid A \vee B \tag{2.2}$$

where disjunction $\vee$, has the usual meaning.

Extended deterministic decision tree formulas, or **eDT** formulas, written A, B etc. are generated from the following grammar:

$$A, B ::= \ p \ \mid \ \overline{p} \ \mid \ ApB \ \mid \ A\overline{p}B \ \mid \ e \tag{2.3}$$

where e is an arbitrary element of a countable set of **extension variables** e_0, e_1, e_2, etc. which we require be disjoint from the set of propositional variables.

Similarly, **extended non-deterministic decision tree** formulas, or **eNDT** formulas, written A, B etc. are generated from the following grammar:

$$A, B ::= \ p \ \mid \ \overline{p} \ \mid \ ApB \ \mid \ A\overline{p}B \ \mid A \vee B \ \mid e \tag{2.4}$$

The size of a formula A, written $|A|$ is the number of symbols occurring in A.

Remark 2.1.5 (Restrictions on decisions)**.** Note that extension variables are formally distinguished from propositional variables for technical reasons. That is, we want to restrict the expressive power of our decision connective and only permit decisions on

literals. Thus, throughout this work and regardless of the choice of grammar used, we forbid formulas of the form Ae_iB or ACB for C a complex (non-literal) formula. If this were allowed, our grammars would be able to express boolean circuits succinctly, whereas the current convention ensures that eDT, $eNDT$ only express BPs and NBPs respectively.

The semantics of non-extended formulas under an assignment are standardly defined:

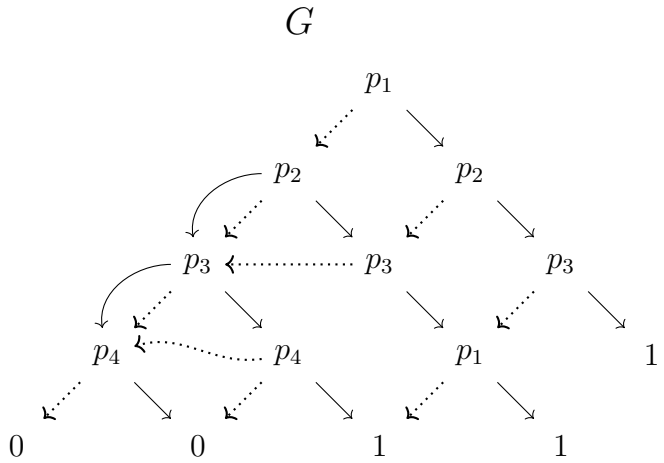
Definition 2.1.6 (Semantics of NDT formulas)**. Satisfaction** written $\models$, is a (infix) binary relation between assignments and formulas defined as follows:

- $\alpha \models p$ if $\alpha(p) = 1$ and $\alpha \models \overline{p}$ if $\alpha(p) = 0$.
- $\alpha \models A \vee B$ if $\alpha \models A$ or $\alpha \models B$.
- $\alpha \models ApB$ if either $\alpha(p) = 0$ and $\alpha \models A$, or $\alpha(p) = 1$ and $\alpha \models B$.
- $\alpha \models A\overline{p}B$ if either $\alpha(\overline{p}) = 0$ and $\alpha \models A$, or $\alpha(\overline{p}) = 1$ and $\alpha \models B$.

When including extension variables however, the interpretation is parametrised by a set of extension axioms:

Definition 2.1.7 (Extension axioms)**.** A set of **extension axioms** $\mathcal{E}$ is a set of the form $\{e_i \leftrightarrow E_i\}_{i<n}$, where E_i is a formula in our language that may only contain extension variables among $e_0, \ldots, e_{i-1}$.

With the use of extension axioms, eDT, eNDT formulas formally represent deterministic and non-deterministic branching programs respectively. Intuitively, for an NBP G, extension variables provide unique 'names' to the nodes of G.



$$
\begin{aligned}
e_{11} &\leftrightarrow e_{21}p_1e_{22}\\
e_{21} &\leftrightarrow e_{31}p_2(e_{31} \vee e_{32})\\
e_{22} &\leftrightarrow e_{32}p_2e_{33}\\
e_{31} &\leftrightarrow e_{41}p_3(e_{41} \vee e_{42})\\
e_{32} &\leftrightarrow e_{31}p_3(e_{43})\\
e_{33} &\leftrightarrow e_{43}p_31\\
e_{41} &\leftrightarrow 0p_40\\
e_{42} &\leftrightarrow (0 \vee e_{41})p_41\\
e_{43} &\leftrightarrow 1p_11
\end{aligned}
$$

Figure 2.1: An NBP G and its representation by extension axioms on the right. e_{ij} corresponds to the j-th node from the left on the i-th 'level' (top to bottom) of the graph. They are ordered lexicographically i.e. $e_{ij} < e_{lk}$ just if $i < l$ or $i = l$ and $j < k$.

Definition 2.1.8 (Semantics of eNDT formulas)**. Satisfaction** with respect to a set of extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$, written $\models_{\mathcal{E}}$, is a (infix) binary relation between assignments and formulas over $e_0, \ldots, e_{n-1}$ that extends Definition 2.1.6:

- $\alpha \models_{\mathcal{E}} p$ if $\alpha(p) = 1$ and $\alpha \models_{\mathcal{E}} \overline{p}$ if $\alpha(p) = 0$.
- $\alpha \models_{\mathcal{E}} A \vee B$ if $\alpha \models_{\mathcal{E}} A$ or $\alpha \models_{\mathcal{E}} B$.
- $\alpha \models_{\mathcal{E}} ApB$ if either $\alpha(p) = 0$ and $\alpha \models_{\mathcal{E}} A$, or $\alpha(p) = 1$ and $\alpha \models_{\mathcal{E}} B$.
- $\alpha \models_{\mathcal{E}} A\overline{p}B$ if either $\alpha(\overline{p}) = 0$ and $\alpha \models_{\mathcal{E}} A$, or $\alpha(\overline{p}) = 1$ and $\alpha \models_{\mathcal{E}} B$.
- $\alpha \models_{\mathcal{E}} e_i$ if $\alpha \models_{\mathcal{E}} E_i$.

The subscripting condition for each e_i ensures that it 'abbreviates' only formulas with 'smaller' (index wise) extension variables. This forces the underlying graphs described by formulas over a set of extension axioms to be well-founded (contain no cycles) and likewise ensures $\models_{\mathcal{E}}$ above is well defined. More formally there is an induced induction principle on extended formulas over a set of extension axioms $\mathcal{E}$:

Remark 2.1.9 ($\mathcal{E}$-induction)**.** Given a set of extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$ we may define the least strict partial order $<_{\mathcal{E}}$ on formulas over $e_0, \ldots, e_{n-1}$ satisfying:

- $p <_{\mathcal{E}} ApB$, $\overline{p} <_{\mathcal{E}} ApB$ and $A <_{\mathcal{E}} ApB$ and $B <_{\mathcal{E}} ApB$.
- $p <_{\mathcal{E}} A\overline{p}B$, $\overline{p} <_{\mathcal{E}} A\overline{p}B$ and $A <_{\mathcal{E}} A\overline{p}B$ and $B <_{\mathcal{E}} A\overline{p}B$.
- $A <_{\mathcal{E}} A \vee B$ and $B <_{\mathcal{E}} A \vee B$.
- $E_i <_{\mathcal{E}} e_i$, for each $i < n$.

Notice that $<_{\mathcal{E}}$ is indeed well-founded by the condition that each E_i must contain only extension variables e_j with $j < i$. Thus we may carry out arguments about formulas by induction on $<_{\mathcal{E}}$, which we shall simply call '$\mathcal{E}$-induction'.

We can now see Definition 2.1.8 above of $\models_{\mathcal{E}}$ as definition by $\mathcal{E}$-induction. Hence, by fixing some set of extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$, each eNDT formula A over $e_0, \ldots, e_{n-1}$ computes a unique Boolean function f by $\alpha \mapsto 1$ just if $\alpha \models_{\mathcal{E}} A$. We may then say that A **computes** f **with respect to** $\mathcal{E}$.

Since many of our future arguments are based on $\mathcal{E}$-induction we present an analysis on its complexity:

Proposition 2.1.10 (Complexity of $\mathcal{E}$-induction)**.** *Let A be an eDT or eNDT formula over $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$. Then $|\{B \mid B <_{\mathcal{E}} A\}| \leqslant |A| + \sum_{i<n} |E_i|$ and, if $B <_{\mathcal{E}} A$, then $|B| \leqslant \max\{|A|, |E_0|, \ldots, |E_{n-1}|\}$.*

Proof. If A is extension-free, then it is easy to see that $|\{B \mid B <_{\mathcal{E}} A\}| \leqslant |A|$ since it amounts to counting all distinct subformulas of A. Suppose instead there are extension variables $e_m, \ldots, e_k$ for $m < k < n$ occurring in A. Then, counting the distinct subformulas of each $E_i, i \leqslant k$ and summing them all together clearly suffices as an upper

bound since, each E_i may only contain extension variables e_j for $j < i$. We thus obtain $|\{B \mid B <_{\mathcal{E}} A\}| \leqslant |A| + \sum_{i \leqslant k} |E_i|$ which is itself upper bounded by $|A| + \sum_{i<n} |E_i|$.

Now, if $B <_{\mathcal{E}} A$, B is either a subformula of A in which case $|B| \leqslant |A|$ or B is a subformula of some E_i such that $e_i \leftrightarrow E_i$ and e_i occurs in A. In the latter case, $|B| \leqslant |E_i|$. Hence, $|B| \leqslant \max\{|A|, |E_0|, \ldots, |E_{n-1}|\}$ suffices as an upper bound. □

2.1.3 The systems eLDT, eLNDT

The language of eLDT comprises of just the eDT formulas and similarly, the language of eLNDT comprises of just the eNDT formulas. A **sequent** is an expression $\Gamma \longrightarrow \Delta$, where Γ and Δ are multisets of formulas. Commas are used to represent multiset unions (interpreted conjunctively on the left hand side and disjunctively on the right hand side of $\longrightarrow$). A sequent $\Gamma \longrightarrow \Delta$ is semantically understood as either a formula in Γ is false or a formula in Δ is true. More formally, under an assignment α we say that $\Gamma \longrightarrow \Delta$ is true if either $\alpha(A) = 0$ for some $A \in \Gamma$ or $\alpha(B) = 1$ for some $B \in \Delta$. We call a sequent **valid** if it is true under all assignments.

The semantic interpretation in 2.1.8 and 2.1.17 means that ApB is simultaneously logically equivalent to $(\overline{p} \wedge A) \vee (p \wedge B)$ and $(\overline{p} \supset A) \wedge (p \supset B)$ where $A \supset B$ is the classical implication abbreviating $\neg A \vee B$. It is this observation which naturally yields the decision rules in the following systems for eDT, eNDT formulas. We choose for negative literals to appear positively on the other hand side of the sequent since later on, negation-freeness will be of importance. See decision rules on $\overline{p}$ in Figure 2.2.

Definition 2.1.11. The system LNDT, is given by the rules in Figure 2.2 and its language comprises of the NDT formulas. An LNDT **derivation** of a sequent $\Gamma \longrightarrow \Delta$ from hypotheses $\mathcal{H} = \{\Gamma_i \longrightarrow \Delta_i\}_{i \in I}$ is defined as usual: it is a finite list of sequents, each either some $\Gamma_i \longrightarrow \Delta_i$ from $\mathcal{H}$ or following from previous ones by rules of LNDT and ending with $\Gamma \longrightarrow \Delta$.

An eLNDT, proof is just an LNDT derivation from hypotheses that are a set of extension axioms, i.e. $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$; where we construe $A \leftrightarrow B$ as an abbreviation for the pair of sequents $A \longrightarrow B$ and $B \longrightarrow A$. The **size** of a proof is simply the number of symbols in it. The **depth** of a proof is the maximum number of steps in a branch.

We note that most of the proofs in our systems will be dag-like: we must avoid duplication of subproofs with the same conclusion as, if this were recursively applied, it could lead to an exponential increase in proof size.

Remark 2.1.12 (Digression on using extension)**.** It is (typically) required that conclusions of eL(N)DT proofs are free of extension variables. This stems from a 'tradition' in the proof complexity literature that systems utilising extension operate under the above condition. For example in [30], Cook and Reckhow do not formally distinguish between extension variables and propositional variables and, among others, restrict conclusions of their proofs to be extension-free. This is done to provide for a 'smooth' soundness argument: truth assignments are defined on propositional variables and may be extended to complex formulas in the usual way; additionally if $e \leftrightarrow A$ is an extension axiom used in a proof, then for any truth assignment α, $\alpha(e) = \alpha(A)$. Therefore, a truth assignment α that satisfies all previous lines in an extended Frege proof, also satisfies

the conclusion since it is extension free. We may later though consider 'intermediate' proofs that conclude with formulas containing extension variables. In all such cases, the underlying extension axiom set will be provided explicitly.

Initial sequents and cut:

$$\mathsf{id}\,\frac{}{p \longrightarrow p} \qquad \mathsf{id}\,\frac{}{\overline{p} \longrightarrow \overline{p}} \qquad \textit{lem-r}\,\frac{}{\longrightarrow p, \overline{p}}$$

$$\textit{lem-l}\,\frac{}{p, \overline{p} \longrightarrow} \qquad \mathsf{cut}\,\frac{\Gamma \longrightarrow \Delta, A \quad \Gamma, A \longrightarrow \Delta}{\Gamma \longrightarrow \Delta}$$

Structural rules:

$$\mathsf{w}\text{-}l\,\frac{\Gamma \longrightarrow \Delta}{\Gamma, A \longrightarrow \Delta} \qquad \mathsf{w}\text{-}r\,\frac{\Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, A} \qquad \mathsf{c}\text{-}l\,\frac{\Gamma, A, A \longrightarrow \Delta}{\Gamma, A \longrightarrow \Delta} \qquad \mathsf{c}\text{-}r\,\frac{\Gamma \longrightarrow \Delta, A, A}{\Gamma \longrightarrow \Delta, A}$$

Logical rules:

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

$$\overline{p}\text{-}l\,\frac{\Gamma, A, p \longrightarrow \Delta \quad \Gamma, B \longrightarrow p, \Delta}{\Gamma, A\overline{p}B \longrightarrow \Delta} \qquad \overline{p}\text{-}r\,\frac{\Gamma, p \longrightarrow \Delta, A \quad \Gamma \longrightarrow p, \Delta, B}{\Gamma \longrightarrow \Delta, A\overline{p}B}$$

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

Figure 2.2: Rules of the system LNDT

Definition 2.1.13. The respective deterministic system LDT is given by the rules in Figure 2.2 excluding the ones for disjunction. The extended version eLDT is defined akin to the way described above for the non-deterministic case. We may at times slightly alter our exposition and permit context splitting cuts, omit structural steps or consider cedents to be sets instead of multisets/lists. This (if and whenever it occurs) will only be for improving the presentation of our proofs and arguments and has no proof complexity theoretic significance.

It was shown in [22], that the system LNDT is adequate for reasoning about non-deterministic decision trees and similarly, eLNDT adequately reasons about NBPs:

Proposition 2.1.14 (Soundness and completeness, [22])**.** eLNDT *proves a sequent* $\Gamma \longrightarrow \Delta$ *(without extension variables) if and only if* $\bigwedge \Gamma \supset \bigvee \Delta$ *is valid.*

One sanity check here is that the set of valid extension-free eLNDT sequents is indeed **coNP**-complete, and so comprises an adequate logic for proof complexity. This is shown explicitly in [22], but is also subsumed by the analogous statement for the 'positive' fragment of this language that we consider in a later section, namely Proposition 2.2.14.

While LNDT supports a propositional identity, it will be quite useful to establish a general identity rule:

Proposition 2.1.15 (General identity)**.** *Let $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$ be a set of extension axioms. There are polynomial-size* eLNDT *proofs of $A \longrightarrow A$, for formulas A containing only extension variables from $\mathcal{E}$.*

Proof. We construct dag-like proofs of the required sequent by $\mathcal{E}$-induction. That is, given eNDT formula A, we construct a proof of polynomial-size (on $|A|+|\mathcal{E}|$) containing sequents $B \longrightarrow B$ for each $B \leqslant_{\mathcal{E}} A$ by $\mathcal{E}$-induction on A.

- If A is a literal then we are done by the rule id.
- If A is an extension variable we extend the proof obtained by the inductive hypothesis by adding the step:

$$\mathsf{cut}\ \dfrac{\mathcal{E}\ \dfrac{}{e_i \longrightarrow E_i} \qquad \mathcal{E}\ \dfrac{}{E_i \longrightarrow e_i}}{e_i \longrightarrow e_i}$$

  where the premises are extension axioms from $\mathcal{E}$.
- If $A = B \vee C$ then we extend the proof obtained by the inductive hypothesis by adding the derivation:

$$\vee\text{-}r\ \dfrac{\vee\text{-}l\ \dfrac{\mathsf{w}\text{-}r\ \dfrac{\dfrac{}{B \longrightarrow B}\,IH}{B \longrightarrow B, C} \qquad \mathsf{w}\text{-}r\ \dfrac{\dfrac{}{C \longrightarrow C}\,IH}{C \longrightarrow B, C}}{B \vee C \longrightarrow B, C}}{B \vee C \longrightarrow B \vee C}$$

- If $A = BpC$ then we extend the proof obtained by the inductive hypothesis by adding the step:

$$p\text{-}l\ \dfrac{p\text{-}r\ \dfrac{\mathsf{w}\text{-}r\ \dfrac{\dfrac{}{B \longrightarrow B}\,IH}{B \longrightarrow B, p} \qquad \mathsf{w}\text{-}l,\mathsf{w}\text{-}r\ \dfrac{\dfrac{}{B \longrightarrow B}\,IH}{p, B \longrightarrow B, C}}{B \longrightarrow BpC, p} \qquad p\text{-}r\ \dfrac{\mathsf{w}\text{-}l,\mathsf{w}\text{-}r\ \dfrac{\mathsf{id}\ \dfrac{}{p \longrightarrow p}}{p, C \longrightarrow B, p} \qquad \mathsf{w}\text{-}l,\mathsf{w}\text{-}r\ \dfrac{\dfrac{}{C \longrightarrow C}\,IH}{p, C \longrightarrow B, C}}{p, C \longrightarrow BpC}}{BpC \longrightarrow BpC}$$

  In all cases sequents marked by *IH* are obtained through the inductive hypothesis.

With respect to the size of the proof provided, note that each step of the argument above, extends the proof by a constant number of lines, each of which has size polynomial in $|A|$ and $|E|$. Thus a polynomial bound follows by proposition Proposition 2.1.10. □

Example 2.1.16 (A 'medial')**.** Many classes of branching programs enjoy elegant symmetries w.r.t. permuting the order of nodes in the graph. Accordingly, in our eNDT notation, we have validity of the following pair of sequents:

$$(AqB)p(CqD) \leftrightarrow (ApC)q(BpD)$$

The above equivalence is visualised in Figure 2.3.

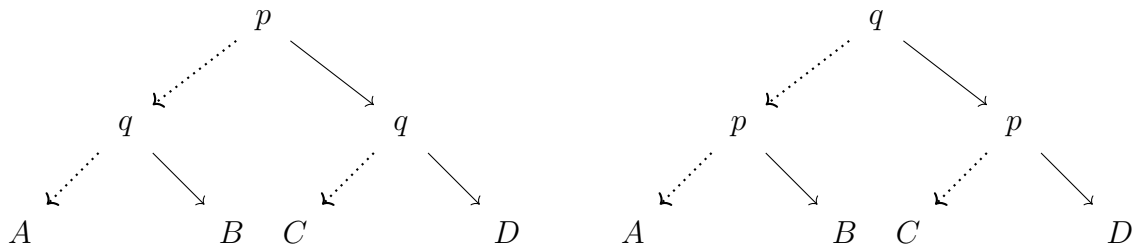


Figure 2.3: The 'medial' visualised. The two BPs compute the same Boolean function despite the change on the order of nodes.

The name medial finds its origins in the 'deep inference' community, see for example [2]. There, the medial rule permits 'exchanging' arguments that are 'deep' in the formula between conjunction ($\wedge$) and disjunction ($\vee$) connectives.

Alternative choices of grammar

It might be convenient to consider different choices for the grammar of our language. For example we may choose to either not admit constants or propositional variables natively but express them via complex formulas.

An alternative version of grammar 2.3 is the **0/1-extended deterministic decision tree** formulas, or 0/1**-eDT** formulas, written A, B etc. which are generated from the following grammar:

$$A, B ::= \ 1 \ \mid \ 0 \ \mid \ ApB \ \mid \ e \tag{2.5}$$

where e is again an arbitrary element of a countable set of **extension variables** e_0, e_1, e_2, etc. Within this grammar, we identify the propositional variable p with the formula $0p1$ and its 'dual' $\overline{p}$ with $1p0$.

0/1-extended non-deterministic decision tree formulas, or 0/1-eNDT formulas, written A, B are generated from the following grammar:

$$A, B ::= \ 1 \ \mid \ 0 \ \mid \ ApB \ \mid \ A \vee B \mid \ e \tag{2.6}$$

Another convenient choice of grammar will be made when we later consider a specific type of NBPs called 'positive branching programs'. m**-extended deterministic**

decision tree formulas, or **m-eDT** formulas, written A, B etc. are generated from the grammar for 0/1-eDT formulas but also admit propositional variables (from a countable set) hence they are generated from:

$$A, B ::= \ 1 \ | \ 0 \ | \ p \ | \ ApB \ | \ e \tag{2.7}$$

m**-extended non-deterministic decision tree** formulas, or m-eNDT formulas, written A, B etc. are generated from the following grammar:

$$A, B ::= \ 1 \ | \ 0 \ | \ p \ | \ ApB \ | \ A \vee B \ | e \tag{2.8}$$

Here, we identify $\overline{p}$ with $1p0$.

The same syntactic restrictions apply in the alternative grammars namely, decisions can only be made on literals and not complex formulas or extension variables.

Definition 2.1.17 (Semantics of 0/1-eNDT,m-eNDT formulas)**.** Most cases are covered by Definition 2.1.8 which we simply extend by considering the cases for constants:

- $\alpha \not\models_{\mathcal{E}} 0$ and $\alpha \models_{\mathcal{E}} 1$.

The systems 0/1-eLNDT**,** m-eLNDT

The language of the systems 0/1-eLNDT, m-eLNDT comprises of just the 0/1-eNDT and m-eNDT formulas respectively. Sequents and syntax are defined akin to Definition 2.1.11.

Definition 2.1.18. The systems 0/1-LNDT, m-LNDT are given by the rules in Figure 2.4. We require that cut-rules can only introduce formulas specific to each grammar. An 0/1-LNDT, m-LNDT **derivation** of a sequent $\Gamma \longrightarrow \Delta$ from hypotheses $\mathcal{H} = \{\Gamma_i \longrightarrow \Delta_i\}_{i\in I}$ is defined as usual: it is a finite list of sequents, each either some $\Gamma_i \longrightarrow \Delta_i$ from $\mathcal{H}$ or following from previous ones by rules of 0/1-LNDT, m-LNDT respectively and ending with $\Gamma \longrightarrow \Delta$.

An 0/1-eLNDT, m-eLNDT proof is just an 0/1-LNDT, m-LNDT derivation from hypotheses that are a set of extension axioms, i.e. $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$; where we construe $A \leftrightarrow B$ as an abbreviation for the pair of sequents $A \longrightarrow B$ and $B \longrightarrow A$.

The respective deterministic systems 0/1-LDT, m-LDT are given by the rules in Figure 2.4 excluding the ones for disjunction. Their extended versions 0/1-eLDT, m-eLDT are defined akin to the way described above for the non-deterministic case.

Remark 2.1.19. The different choices of grammar are to some degree 'practical'. The grammar from 2.6 can be viewed as more 'convenient' when defining Prover-Adversary games for NBPs (Chapter 6). That is because we avoid over-defining in our language: the atom p is not natively available and instead is identified with the decision $0p1$. Doing this allows us to get away with defining smaller and less convoluted sets of simple contradictions. The grammars from 2.8 and 2.3[22] may be preferable from a proof theoretic perspective since cut-free proofs satisfy the subformula property in the respective calculi. Furthermore, the grammar from 2.8 only permits decisions on 'positive' literals which accommodates the definition of positive branching programs, in Section 2.2.3. On the other hand, the grammar from 2.3[22] may be thought as more 'elegant', since it avoids overdefining with respect to constants.

Initial sequents and cut:

$$0\ \frac{}{0 \longrightarrow} \qquad 1\ \frac{}{\longrightarrow 1} \qquad \mathsf{id}\ \frac{}{A \longrightarrow A}$$

$$\mathsf{id}\ \frac{}{p \longrightarrow p} \qquad \mathsf{cut}\ \frac{\Gamma \longrightarrow \Delta, A \quad \Gamma, A \longrightarrow \Delta}{\Gamma \longrightarrow \Delta}$$

Structural rules:

$$\mathsf{w}\text{-}l\ \frac{\Gamma \longrightarrow \Delta}{\Gamma, A \longrightarrow \Delta} \qquad \mathsf{w}\text{-}r\ \frac{\Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, A} \qquad \mathsf{c}\text{-}l\ \frac{\Gamma, A, A \longrightarrow \Delta}{\Gamma, A \longrightarrow \Delta} \qquad \mathsf{c}\text{-}r\ \frac{\Gamma \longrightarrow \Delta, A, A}{\Gamma \longrightarrow \Delta, A}$$

Logical rules:

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

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

Figure 2.4: Rules of the systems 0/1-LNDT, m-LNDT. The rule exclusive to 0/1-LNDT is brown.

In this work we will make use of the grammars for 0/1-eNDT and m-eNDT formulas. Polynomial equivalence between the systems eLNDT, 0/1-eLNDT, m-eLNDT is provided in the next section.

2.1.4 Simple Simulation Results

Here for completeness, we provide the polynomial equivalence results between the systems of eLNDT, 0/1-eLNDT and m-eLNDT. We start by presenting an inductive translation (on formula structure) between eNDT and 0/1-eNDT formulas.

Remark 2.1.20. We observe that it is enough to show polynomial equivalence between eLNDT and 0/1-eLNDT. One way to see this is by observing that in Figure 2.4, all rules of m-eLNDT are also rules of 0/1-eLNDT while general identity can (as we will see later) be polynomially simulated in m-eLNDT. More formally, we can define a 'natural' translation between eNDT and m-eNDT formulas that is subsumed by the following translations in Definitions 2.1.21, 2.1.22. Hence, an equivalence result between eLNDT and m-eLNDT would accordingly be subsumed by the equivalence result for eLNDT and 0/1-eLNDT, Corollary 2.1.25.

Definition 2.1.21 (From eNDT to 0/1-eNDT)**.** We provide a (polynomial-time) translation from an eNDT formula A to a 0/1-eNDT formula A^{b} as follows:

$$\begin{array}{ll} p^{\mathsf{b}} & := 0p1 \\ \overline{p}^{\mathsf{b}} & := 1p0 \end{array} \qquad \begin{array}{ll} (ApB)^{\mathsf{b}} & := A^{\mathsf{b}}pB^{\mathsf{b}} \\ (A\overline{p}B)^{\mathsf{b}} & := B^{\mathsf{b}}pA^{\mathsf{b}} \\ (A \vee B)^{\mathsf{b}} & := A^{\mathsf{b}} \vee B^{\mathsf{b}} \end{array}$$

For a multiset $\Gamma = \{A_1, \ldots, A_n\}$, its 'translated' version Γ^{b} is $\{A_1^{\mathsf{b}}, \ldots, A_n^{\mathsf{b}}\}$. For a set of extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i\}$ we write $\mathcal{E}^{\mathsf{b}}$ for $\{e_i^{\mathsf{b}} \leftrightarrow E_i^{\mathsf{b}}\}$ where each e_i^{b} is a fresh extension variable.

Definition 2.1.22 (From 0/1-eNDT to eNDT)**.** We provide a (polynomial-time) translation from an 0/1-eNDT formula A to a eNDT formula A^{p} as follows:

$$\begin{array}{ll} 0^{\mathsf{p}} & := pp\overline{p} \\ 1^{\mathsf{p}} & := \overline{p}pp \end{array} \qquad \begin{array}{ll} (ApB)^{\mathsf{p}} & := A^{\mathsf{p}}pB^{\mathsf{p}} \\ (A \vee B)^{\mathsf{p}} & := A^{\mathsf{p}} \vee B^{\mathsf{p}} \end{array}$$

For a multiset Γ, its 'translated' version Γ^{p} is $\{A_1^{\mathsf{p}}, \ldots, A_n^{\mathsf{p}}\}$. For a set of extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i\}$ we write $\mathcal{E}^{\mathsf{p}}$ for $\{e_i^{\mathsf{p}} \leftrightarrow E_i^{\mathsf{p}}\}$ where each e_i^{p} is a fresh extension variable.

Next, we check that any LNDT proof of a sequent $\Gamma \longrightarrow \Delta$ can be simulated by an 0/1-LNDT proof of $\Gamma^{\mathsf{b}} \longrightarrow \Delta^{\mathsf{b}}$ and vice versa.

Proposition 2.1.23. *From an* LNDT *proof P for a sequent $\Gamma \longrightarrow \Delta$ we can in polynomial time construct a* 0/1-LNDT *proof P^{b} of the sequent $\Gamma^{\mathsf{b}} \longrightarrow \Delta^{\mathsf{b}}$. Similarly, from a* 0/1-LNDT *proof P for a sequent $\Gamma \longrightarrow \Delta$ we can in polynomial time construct an* LNDT *proof P^{p} of the sequent $\Gamma^{\mathsf{p}} \longrightarrow \Delta^{\mathsf{p}}$.*

Proof. We proceed by induction on the structure of the proof and start by showing polynomial simulation of LNDT by 0/1-LNDT. To do that, given an LNDT proof P of a sequent $\Gamma \longrightarrow \Delta$, we obtain an 0/1-LNDT proof P^{b} of $\Gamma^{\mathsf{b}} \longrightarrow \Delta^{\mathsf{b}}$ with only polynomial increase in size. Since both translations commute with connectives (excluding $\_^{\mathsf{b}}$ on $A\overline{p}B$), the proof is straightforward.

- A left decision rule on a negative literal $\overline{p}$:

$$\overline{p}\text{-}l\,\frac{\Gamma, p, A \longrightarrow \Delta \quad \Gamma, B \longrightarrow \Delta, p}{\Gamma, A\overline{p}B \longrightarrow \Delta}$$

  will be simulated by the following left decision:

$$p\text{-}l\,\frac{\Gamma^{\mathsf{b}}, B^{\mathsf{b}} \longrightarrow \Delta^{\mathsf{b}}, p \quad \Gamma^{\mathsf{b}}, p, A^{\mathsf{b}} \longrightarrow \Delta^{\mathsf{b}}}{\Gamma^{\mathsf{b}}, B^{\mathsf{b}}pA^{\mathsf{b}} \longrightarrow \Delta^{\mathsf{b}}}$$

- A right decision rule on a negative literal $\overline{p}$:

$$\overline{p}\text{-}r\,\frac{\Gamma, p \longrightarrow \Delta, A \quad \Gamma \longrightarrow \Delta, B, p}{\Gamma \longrightarrow \Delta, A\overline{p}B}$$

  will be simulated by the right decision:

$$p\text{-}r\,\frac{\Gamma^{\mathsf{b}} \longrightarrow \Delta^{\mathsf{b}}, B^{\mathsf{b}}, p \quad \Gamma^{\mathsf{b}}, p \longrightarrow \Delta^{\mathsf{b}}, A^{\mathsf{b}}}{\Gamma^{\mathsf{b}} \longrightarrow \Delta^{\mathsf{b}}, B^{\mathsf{b}}pA^{\mathsf{b}}}$$

- A left decision rule:

$$p\text{-}l\,\frac{\Gamma, A \longrightarrow \Delta, p \quad \Gamma, p, B \longrightarrow \Delta}{\Gamma, ApB \longrightarrow \Delta}$$

  will be simulated by the following left decision:

$$p\text{-}l\,\frac{\Gamma^{\flat}, A^{\flat} \longrightarrow \Delta^{\flat}, p \quad \Gamma^{\flat}, p, B^{\flat} \longrightarrow \Delta^{\flat}}{\Gamma^{\flat}, A^{\flat}pB^{\flat} \longrightarrow \Delta^{\flat}}$$

- A right decision rule:

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

  will be simulated by the right decision:

$$p\text{-}r\,\frac{\Gamma^{\flat} \longrightarrow \Delta^{\flat}, A^{\flat}, p \quad \Gamma^{\flat}, p \longrightarrow \Delta^{\flat}, B^{\flat}}{\Gamma^{\flat} \longrightarrow \Delta^{\flat}, A^{\flat}pB^{\flat}}$$

- A right disjunction rule:

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

  will be simulated by the right disjunction:

$$\vee\text{-}r\,\frac{\Gamma^{\flat} \longrightarrow \Delta^{\flat}, A^{\flat}, B^{\flat}}{\Gamma^{\flat} \longrightarrow \Delta^{\flat}, A^{\flat} \vee B^{\flat}}$$

- A left disjunction rule:

$$\vee\text{-}l\,\frac{\Gamma, A \longrightarrow \Delta \quad \Gamma, B \longrightarrow \Delta}{\Gamma, A \vee B \longrightarrow \Delta}$$

  will be simulated by the right disjunction:

$$\vee\text{-}l\,\frac{\Gamma^{\flat}, A^{\flat} \longrightarrow \Delta^{\flat} \quad \Gamma^{\flat}, B^{\flat} \longrightarrow \Delta^{\flat}}{\Gamma^{\flat}, A^{\flat} \vee B^{\flat} \longrightarrow \Delta^{\flat}}$$

- If a cut-rule introduces a formula A

$$\mathsf{cut}\,\frac{\Gamma \longrightarrow \Delta, A \quad \Gamma, A \longrightarrow \Delta}{\Gamma \longrightarrow \Delta}$$

  it will be simulated by a cut-rule on the translated formula $A^{\flat}$ with the translated contexts:

$$\mathsf{cut}\,\frac{\Gamma^{\flat} \longrightarrow \Delta^{\flat}, A^{\flat} \quad \Gamma^{\flat}, A^{\flat} \longrightarrow \Delta^{\flat}}{\Gamma^{\flat} \longrightarrow \Delta^{\flat}}$$

- The four initial sequents:

$$\mathsf{id}\ \frac{}{p \longrightarrow p} \qquad \mathsf{id}\ \frac{}{\overline{p} \longrightarrow \overline{p}} \qquad \textit{lem-r}\ \frac{}{\longrightarrow p, \overline{p}} \qquad \textit{lem-l}\ \frac{}{p, \overline{p} \longrightarrow}$$

  will be simulated respectively by instances of the general identity rule for $A = 0p1, 1p0$:

$$\mathsf{id}\ \frac{}{1p0 \longrightarrow 1p0} \qquad \mathsf{id}\ \frac{}{0p1 \longrightarrow 0p1}$$

  The following constant size derivation:

$$\dfrac{\dfrac{}{\longrightarrow 0p1, 1, p}\,\mathsf{1,w\text{-}}r \qquad \dfrac{}{p \longrightarrow 0, p}\,\mathsf{id,w\text{-}}r}{\longrightarrow 0p1, 1p0}\,p\text{-}r$$

  And lastly:

$$\dfrac{\dfrac{}{0, 1p0 \longrightarrow p}\,0 \qquad \dfrac{\dfrac{}{p, 1, 1 \longrightarrow p}\,\mathsf{id} \qquad \dfrac{}{p, p, 0 \longrightarrow}\,0}{p, 1p0, 1 \longrightarrow}\,p\text{-}l}{0p1, 1p0 \longrightarrow}\,p\text{-}l$$

With this we conclude that the 0/1-LNDT proof P^{b} is clearly polynomial-time computable from P.

The other direction i.e. polynomial simulation of 0/1-LNDT by LNDT proceeds similarly and is not explicitly provided. The only cases worth mentioning are for initial rules. These are either simulated by LNDT derivations of constant size or derivations employing general identity requiring proofs of polynomial size.

- The initial rules

$$0\ \frac{}{0 \longrightarrow} \qquad 1\ \frac{}{\longrightarrow 1}$$

  can be respectively simulated by LNDT derivations of fixed size with conclusions $pp\overline{p} \longrightarrow$ and $\longrightarrow \overline{p}pp$ respectively:

$$\dfrac{\mathsf{id}\ \dfrac{}{p \longrightarrow p} \qquad \textit{l-lem}\ \dfrac{}{p, \overline{p} \longrightarrow}}{pp\overline{p} \longrightarrow}\,p\text{-}r \qquad\qquad \dfrac{\textit{l-rem}\ \dfrac{}{\longrightarrow p, \overline{p}} \qquad \mathsf{id}\ \dfrac{}{p \longrightarrow p}}{\longrightarrow \overline{p}pp}\,p\text{-}l$$

  For a formula A, the generalized identity rule:

$$\mathsf{id}\ \frac{}{A \longrightarrow A}$$

in 0/1-LNDT is simulated by a derivation of the sequent $A^{\mathsf{p}} \longrightarrow A^{\mathsf{p}}$ of polynomial size in LNDT. This follows by a more general result for the system eLNDT, proved in Proposition 2.1.15. The required corresponding proof for LNDT would be slightly simpler: the cases of extension variables need not be considered and the proof would proceed by induction on the complexity of formulas instead of $\mathcal{E}$-induction. □

Remark 2.1.24. We remind that to avoid 'explosion' in size our proofs are dag-like. Furthermore, note that besides being a very natural property for our systems to satisfy, general identity also implies that replacing a variable (not occurring deeply within formulas) with a formula in a proof will only result in polynomial increase in size.[3] This will in particular be of use later, for example in Chapter 3 and also in Chapter 6 while a slightly adjusted general identity result for a fragment of eLNDT will appear in Chapter 5 .

From Proposition 2.1.23 we immediately obtain the same result for the extended systems:

Corollary 2.1.25. The systems eLNDT and 0/1-eLNDT are polynomially equivalent.

Proof. To prove the statement it is enough to show that from an eLNDT proof P for a sequent $\Gamma \longrightarrow \Delta$ with hypotheses from a set of extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i\}$, we can in polynomial time construct an 0/1-eLNDT proof P^{b} of the sequent $\Gamma^{\mathsf{b}} \longrightarrow \Delta^{\mathsf{b}}$ with hypotheses from $\mathcal{E}^{\mathsf{b}}$ and vice versa.

The proof proceeds in the same way Proposition 2.1.23 does, with the addition that in case an extension variable e_i is occurring at some point in a sequent, it must be accordingly substituted with its translation e_i^{b}. The other direction is done similarly but replacing an extension variable e_i with e_i^{p} instead. □

It is also easy to see that the same should hold for the deterministic fragments of the systems since the rules for disjunction are 'independent' to other rules and vice versa:

Corollary 2.1.26. The systems eLDT and 0/1-eLDT polynomially simulate each other.

Proof. We make the observation that in the proof of Proposition 2.1.23, disjunction is only used and required in disjunction steps. Hence, in the deterministic case, disjunction is simply not considered while everything else remains the same. □

Remark 2.1.27. In light of the results of this section we may freely reason within any of the systems eLNDT, m-eLNDT, 0/1-eLNDT interchangeably. To avoid ambiguity, we will make clear which grammar we are using at the beginning of each section and only write eLNDT, eLDT afterwards.

[3] Since id rules at leaves of the proof will be replaced by polynomial size derivations of general identity.

2.2 Monotone functions and positive proofs

Several models of monotone computation have been proposed and utilized in the literature. Examples include negation free circuits and formulas, monotone span programs introduced in [40] and positive branching programs which will be of use to us.

This section opens by recapping notions of monotonicity, regarding computation and boolean functions. We introduce positive branching programs (PBPs) computing monotone Boolean functions and define the system eLNDT^+, a fragment of eLNDT that restricts reasoning to only formulas representing PBPs. It is noted that we formally distinguish 'monotone' and 'positive', reserving the former for semantics and the latter for syntax.

2.2.1 Monotone Boolean functions and positive programs

Given $\mathbf{x} = (x_1, \ldots, x_n), \mathbf{y} = (y_1, \ldots, y_n)$ where $x_i, y_i \in \{0, 1\}$, we write $\mathbf{x} \leqslant \mathbf{y}$ if each $x_i \leqslant y_i$, i.e. $\mathbf{y}$ may be obtained from $\mathbf{x}$ by flipping 0s to 1s. A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is called **monotone** if, whenever some tuple $\mathbf{y} \in \{0,1\}^n$ is obtained from tuple $\mathbf{x} \in \{0,1\}^n$ by flipping 0s to 1s, we have $f(\mathbf{x}) \leqslant f(\mathbf{y})$. It is more suitable to our setting (accepting runs of NBPs) for the inputs of Boolean functions to be finitely supported assignments. Under this convention, we consider the following alternative definition:

Given a list of propositional variables $\mathbf{p} = \{p_1, \ldots p_n\}$, for $\alpha, \beta \in \mathsf{Ass}_{\mathbf{p}}$, we write $\alpha \leqslant \beta$ if for all p_i, we have $\alpha(p_i) \leqslant \beta(p_i)$. A Boolean function $f : \mathsf{Ass}_{\mathbf{p}} \to \{0,1\}$ is **monotone** if for all $\alpha, \beta \in \mathsf{Ass}_{\mathbf{p}}$, if $\alpha \leqslant \beta$ then $f(\alpha) \leqslant f(\beta)$.

The two definitions of monotonicity are equivalent and we may use either while taking care to be clear when necessary.

Definition 2.2.1. [Positive BPs, e.g. [33]] A **positive** BP is an NBP such that for every 0-edge in its graph from a node u to a node v, there is a 1-edge from u to v.

Proposition 2.2.2. *Positive BPs compute monotone boolean functions.*

Intuitively, this is easy to see, by Definition 2.2.1, any leaf that can be reached via a path π with any number of 0-edges can also be reached by another path π' which is obtained by flipping some 0-edges in π to 1-edges. More formally:

Proof. Assume we are given two assignments $\alpha \leqslant \beta$ and α is accepting for G i.e. $\alpha(G) = 1$; it is enough to show $\beta(G) = 1$. That $\alpha(G) = 1$ means there is an accepting run $\mathbf{r} = (r_1, \ldots, r_m, 1)$ of G consistent with α where for $i < m$, each r_i is labelled by some propositional variable p_i. To finish the proof, from $\mathbf{r}$ we will inductively construct an accepting run $\mathbf{r}' = (r'_1, \ldots, r'_m, 1)$ consistent with β. Since there is only one root node in G, we must have $r'_1 = r_1$. Then, for each i such that $\alpha(p_i) = \beta(p_i)$, set $r'_i = r_i$. In the case when $\alpha(p_i) = 0$ and $\beta(p_i) = 1$, by Definition 2.2.1, we know there will be a 1-edge from node r_i to node r_{i+1} and we set $r'_{i+1} = r_{i+1}$. We observe that $\mathbf{r}'$ is the same run as $\mathbf{r}$ and it is also accepting for β.

□

2.2.2 Monotone complexity and positive closures

A family of NBPs (or circuits) $\{G_0, G_1, \ldots\}$ is called logspace (**L**)-**uniform** if there is a **L**-Turing machine M such that for each $n \in \mathbb{N}$, $M(n) = G_n$. Textbook references to circuit/branching program complexity include: [56, 61].

Grigni and Sipser in [33, 32] developed and studied a comprehensive theory of monotone models of computation with the general goal of answering whether containment relations between complexity classes also hold for their monotone versions. They defined monotone[4] non-deterministic Turing machines (MNTM), where if transition is possible when reading 0, the same transition is also possible when reading a 1. They named the class of languages decided by such machines with logarithmic size work-tape **mNL** and additionally provide a non-uniform analogue to this class namely, **L**-computable families of monotone non-deterministic logarithmic-width polynomial-sized leveled circuits.

Deterministic branching programs are a **non-uniform analogue** of **L**: for every language $L \in \mathbf{L}$ there is an **L**-computable family of BPs $\{G_0, G_1, \ldots\}$ (thus with size polynomial w.r.t. input size) such that on input x of length n, G_n decides whether $x \in L$ and conversely, the language $E = \{(G, x) \mid G(x) = 1\}$ is **L**-complete[5] where G is a BP and x an appropriate input. Comparably, NBPs are a non-uniform version of non-deterministic logspace (**NL**). In this work we will provide a non-uniform analogue to **mNL** by defining **L**-computable families of positive branching programs (PBPs).

To define PBPs we make use of a natural construction that is available in the setting of BPs in contrast to Boolean formulas or circuits. That is, the 'positive closure' which given a BP, constructs a 'positive' version of it by adding, for every 0-edge from a node u to a node v, a 1-edge from u to v (if there is not already one). Formally:

Definition 2.2.3 (Positive closure of an NBP)**.** Given NBP G, if the sets of 0-edges and 1-edges of G are E_0 and E_1 respectively, then its **positive closure** G^+ is the NBP with the same vertex set as G and 0-edges E_0 but 1-edges $E_0 \cup E_1$.

We may call edges **parallel** if they have the same starting and ending nodes. We note that through the positive closure construction we always obtain a positive BP thus giving us a 'canonical' positive version of an NBP.

There is a closely related semantic notion of closure for boolean functions, the 'monotone closure'.

Definition 2.2.4 (Monotone closure)**.** For a Boolean function $f : \{0,1\}^n \to \{0,1\}$, we define its **monotone closure** $m(f)$, by $m(f)(\mathbf{x}) = 1$ just if $\exists \mathbf{y} \leqslant \mathbf{x}$ such that $f(\mathbf{y}) = 1$. Alternatively, if we consider the inputs of f ranging over Ass_n, we define $m(f)(\alpha) = 1$ just if $\exists \beta \leqslant \alpha$ such that $f(\beta) = 1$.

Remark 2.2.5. While it will not be of use to this work, we mention that one can define the **dual monotone closure** of f by: $\overline{m}(f)(\mathbf{x}) = 0$ just if $\exists \mathbf{y} \geqslant \mathbf{x}$ such that $f(\mathbf{y}) = 0$. It is the 'greatest' monotone function dominated by f and holds a similar relation with 'positive co-NBPs' Remark 2.1.4. That is, being given an NBP G computing f, we consider its **dual positive closure** i.e. the positive co-NBP G^+.

[4] They use the word "monotone" for both semantic and syntactic notions and reserve "positive" for a class of monotone functions.

[5] Under $\mathbf{NC}^1$ reductions, see [28]

The monotone closure of a function f essentially acts as a least monotone upper bound for f. In many cases thanks to the monotone closure we can exactly characterise the semantic effect of positive closures of NBPs. That is, there are specific classes of NBPs for which the positive closure of an NBP G in that class, computes exactly the monotone closure of the function computed by G. We call a (deterministic) BP G **read-once** if each path of G contains at most one instance of each propositional variable. Then:

Proposition 2.2.6 ([33])**.** *If G is a read-once BP computing a Boolean function f then G^+ computes exactly $m(f)$.*

Proof sketch. Assume the nodes of G are labelled by propositional variables from $\{p_i\}_{i=1}^n$. For one direction, if $m(f)(\beta) = 1$ then by Definition 2.2.4, we know there exists some assignment $\alpha \leqslant \beta$ such that $f(\alpha) = 1$. Hence G accepts on α and consequently G^+ accepts on α (since the same accepting run can be followed). Furthermore, similar to the argument given in Proposition 2.2.2, since for every 0-edge in G^+ there is a parallel 1-edge, it is clear that G^+ also accepts on β. For the other direction, for any assignment β such that $G^+(\beta) = 1$, we will show that $m(f)(\beta) = 1$. If it holds $G(\beta) = 1$ then also $f(\beta) = 1$ and by Definition 2.2.4 we obtain $m(f)(\beta) = 1$. Hence, the only case to consider is that for some assignment β, G^+ has an accepting run r^+ , but G does not. Since for any run of G there is an identical run in G^+, it must be that r^+ follows some of the 1-edges in G^+ that do not occur in G. Since every such 1-edge is parallel to a 0-edge, there is an accepting run r in G corresponding to some assignment $\alpha \leqslant \beta$. Hence $f(\alpha) = 1$ and by Definition 2.2.4 $m(f)(\beta) = 1$.

□

A minimal counterexample of a BP G in which a repetition of a variable in a path causes G^+ to fail to compute $m(f)$ is illustrated in Figure 2.5.

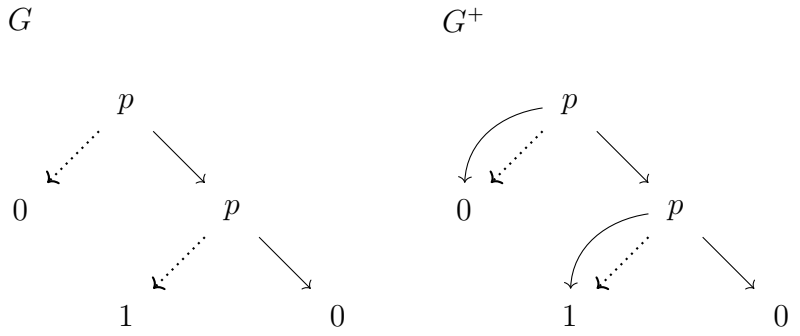


Figure 2.5: Why reading once matters. G computes the zero function but its positive closure G^+, does not since $G^+(p \mapsto 1)$ accepts.

In particular, Proposition 2.2.6 holds when G is an **ordered branching program** (usually abbreviated as 'OBDD') from **ordered binary decision diagram**, i.e. a BP in which propositional variables appear in the same relative order in each run through G. An example of this which will be useful later involves OBDDs calculating the exact counting functions in Chapter 3.

Example 2.2.7 (Monotone closure of Exact)**.** A BP and its positive closure are given in Figure 2.6. The upper graph G is an OBDD computing f, the Exact 2-out-of-4 function (returns 1 iff exactly two input bits are 1) and below is G^+, its positive closure. Since G is read-once by Proposition 2.2.6 we know that G^+ is a PBP computing $m(f)$, the Threshold 2-out-of-4 function (returns 1 iff at least two input bits are 1).

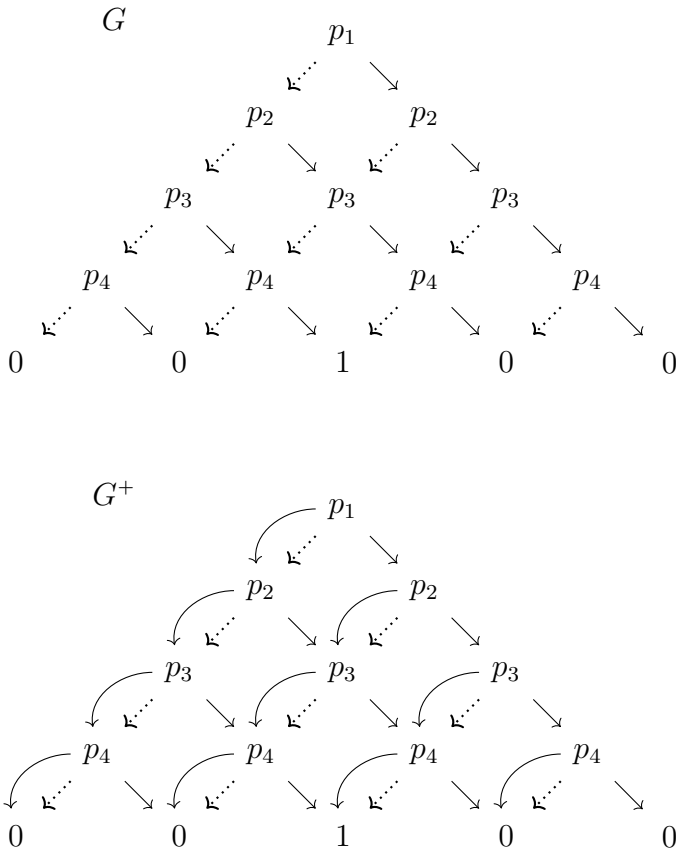


Figure 2.6: Upper: OBDD for 2-out-of-4 Exact, lower: its positive closure computing 2-out-of-4 Threshold.

In [33, 32], Grigni and Sipser proved separation results for the monotone counterparts of some well known complexity classes. Among others, they show that the monotone version of the Immerman-Szelepcsényi theorem (**NL** = *co***NL**, [37, 58]) does not hold: **mNL** is not closed under complementation.[6] Their method starts by stating that reachability between two nodes in a graph is in **mNL** (there is an **L**-computable family of polynomial-size PBPs deciding it) and later show that the only monotone co-non-deterministic BPs computing reachability will have exponential size, concluding that reachability is not in *co*-**mNL** implying **mNL** $\neq$ *co*-**mNL**. In light of the Immerman-Szelepcsényi theorem: **NL** = *co*-**NL**, [37, 58], they showed that **mNL** is not the monotone fragment of **NL**. Since all monotone functions can be computed

[6]They defined **positive monotone** functions as we defined monotone functions in Subsection 2.2.1 and called their complements **negative monotone** functions. The **set of monotone functions** is the union of positive and negative monotone functions. Only in this manner, considering the complements of monotone complexity classes makes sense. See also Section 2.1 of [32]

via PBPs the above can be understood as: there is no feasible notion of constructing the positive closure of NBPs (i.e. maintaining polynomial-size) that succeeds in always computing the monotone closure of boolean functions.

A corollary of their results is:

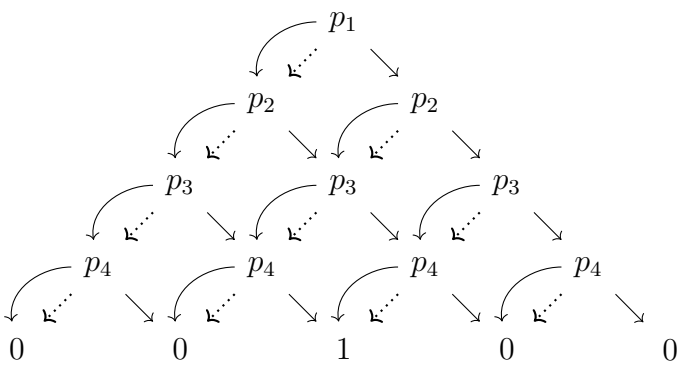
Theorem 2.2.8 ([33])**.** *There are monotone functions computed by polynomial-size families of NBPs, but only super-polynomial-size families of PBPs.*

2.2.3 Representations of positive branching programs

We now revisit representing NBPs by extended formulas but specifically tailored to PBPs. In this section and until further notice we use the grammar for m-eNDT formulas from (2.8) and the appropriate semantics from Definition 2.1.17.

Definition 2.2.9 (Positive formulas)**. Positive extended non-deterministic decision tree** formulas (eNDT$^+$) are eNDT formulas such that for each of their subformulas of the form ApB, it holds $B = A \vee C$ for some C. A set of extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$ is **positive** if each E_i is positive.

eNDT$^+$ formulas, under positive extension axioms are representations of PBPs, the extension axioms given in Figure 2.7 represent a PBP computing the Threshold 2-out-of-4 function.



$e_{11} \leftrightarrow e_{21}p_1(e_{21} \vee e_{22})$
$e_{21} \leftrightarrow e_{31}p_2(e_{31} \vee e_{32})$
$e_{22} \leftrightarrow e_{32}p_2(e_{32} \vee e_{33})$
$e_{31} \leftrightarrow e_{41}p_3(e_{41} \vee e_{42})$
$e_{32} \leftrightarrow e_{42}p_3(e_{42} \vee e_{43})$
$e_{33} \leftrightarrow e_{43}p_3(e_{43} \vee e_{44})$
$e_{41} \leftrightarrow 0p_4(0 \vee 0)$
$e_{42} \leftrightarrow 0p_4(0 \vee 1)$
$e_{43} \leftrightarrow 1p_4(1 \vee 0)$
$e_{44} \leftrightarrow 0p_4(0 \vee 0)$

Figure 2.7: Positive BP computing 2-out-of-4 Threshold, and representation by positive extension axioms.

In particular a **positive decision**[7] $Ap(A \vee B)$ is logically equivalent to the monotone boolean formula $A \vee (p \wedge B)$. Thus, under the restriction that all decisions are positive, eNDT$^+$ formulas over positive sets of extension axioms duly compute monotone Boolean functions:

Proposition 2.2.10. *Suppose $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$ is a positive set of extension axioms. Each eNDT$^+$ formula A over $e_0, \ldots, e_{n-1}$ computes a monotone Boolean function with respect to $\mathcal{E}$.*

[7]From now on we may use the term 'general decision' when referring to the decisions introduced in Section 2.1.2 to distinguish them from positive ones.

Proof. The proof proceeds by $\mathcal{E}$-induction on formulas. It is immediate that propositional variables and constants compute monotone Boolean functions. Assume that by induction hypothesis we have obtained that A, B compute monotone Boolean functions. Given an eNDT$^+$ formula $Ap(A \vee B)$, we recall it is logically equivalent to $A \vee (p \wedge B)$ which, since we do not permit negative literals, is a monotone boolean formula. Hence, $Ap(A \vee B)$ must also compute a monotone Boolean function. An eNDT$^+$ formula of the form $A \vee B$, for A, B computing monotone boolean functions, also immediately computes a monotone Boolean function since $\vee$ is monotone. The case of an extension variable e_i such that $e_i \leftrightarrow E_i$ is an extension axiom, is covered by the logical equivalence of e_i with E_i and invoking induction hypothesis on E_i since $E_i <_{\mathcal{E}} e_i$. □

2.2.4 The positive fragment of eLNDT

Our goal is to define the proof system eLNDT$^+$ reasoning about formula representations of positive BPs. For this, we need to design new decision rules which are sound, invertible, satisfy the subformula property and also have no 'side changing'. That is, apart from id and cut, in all rules of eLNDT$^+$ distinguished formulas are typeset on only one side of the respective sequents. As we will see later in Proposition 2.2.15, the invertibility property together with the subformula property imply completeness of the positive decision rules and the system eLNDT$^+$.

The semantic equivalence of $Ap(A \vee B)$ and $A \vee (p \wedge B)$ motivates the following 'positive decision' rules:

$$p^+\text{-}l\,\frac{\Gamma, A \longrightarrow \Delta \quad \Gamma, p, B \longrightarrow \Delta}{\Gamma, Ap(A \vee B) \longrightarrow \Delta} \qquad p^+\text{-}r\,\frac{\Gamma \longrightarrow \Delta, A, p \quad \Gamma \longrightarrow \Delta, A, B}{\Gamma \longrightarrow \Delta, Ap(A \vee B)} \tag{2.9}$$

These rules can be further justified by showing they are derivable by general decisions, e.g. p^+-l can be derived by only using p-l in combination with $\vee$-l and w-l.

Theorem 2.2.11. *There are polynomial-size* eLNDT *derivations for the above rules (2.9).*

Proof. We can obtain the following derivation:

$$p\text{-}l\,\frac{\mathsf{w}\text{-}l\,\dfrac{\Gamma, A \longrightarrow \Delta}{\Gamma, p, A \longrightarrow \Delta} \qquad \vee\text{-}l\,\dfrac{\Gamma, p, B \longrightarrow \Delta \qquad \mathsf{w}\text{-}l\,\dfrac{\Gamma, A \longrightarrow \Delta}{\Gamma, p, A \longrightarrow \Delta}}{\Gamma, p, A \vee B \longrightarrow \Delta}}{\Gamma, Ap(A \vee B) \longrightarrow \Delta}$$

This derivation concludes with $\Gamma, Ap(A \vee B) \longrightarrow \Delta$ with $\Gamma, A \longrightarrow \Delta$ and $\Gamma, p, B \longrightarrow \Delta$ as premises, the same ones as in the p^+-l. This shows that p^+-l can be derived by only using p-l, $\vee$-l and w-l.

Similarly, we can obtain the derivation:

$$p\text{-}r\,\frac{\Gamma \longrightarrow \Delta, p, A \qquad \mathsf{w}\text{-}l, \vee\text{-}r\,\dfrac{\Gamma \longrightarrow \Delta, A, B}{\Gamma, p \longrightarrow \Delta, A \vee B}}{\Gamma \longrightarrow \Delta, Ap(A \vee B)}$$

This derivation shows that that p^+-r can be derived by only using p-r, $\vee$-r and w-l. □

By construction, none of the formulas A, p or B 'change sides' in the rules above, making positive decisions indeed monotone. This is unlike general decisions, for which p may change sides.

Remark 2.2.12. One may observe that contrary to the case of general decisions (which are self-dual), the rules for positive decisions are not dual (for instance A is 'duplicated' only in the positive decision right rule). This can be seen in the process of constructing them: in the case of the left positive decision rule, the three valid hypotheses occur because of a left disjunction step which, is not dual to a right disjunction step. Accordingly, in the case of the positive decision right rule there are only two valid hypotheses.

This happens because we use existential nondeterminism but not universal. If we defined a new type of decision of the form: $(A \wedge B)pA$ we could, similarly to the case above, define new left and right decision rules such that they would be the dual of the previous positive decision rules.

We are now ready to define the positive fragment of eLNDT:

Definition 2.2.13 (eLNDT$^+$)**.** The system eLNDT$^+$ is defined like eLNDT, using the rules for the system m-LNDT from Figure 2.4, except with replacing the p-l and p-r rules by their positive counterparts from Equation (2.9). It is also required that all extension axioms and formulas (in particular formulas introduced by a cut) in a proof are positive.

A **positive sequent** contains only positive formulas under positive extension axioms.

It was shown in [22] for LDT (and in extension LNDT, eLDT, eLNDT) that the set of valid sequents is expressive enough i.e. **coNP**-complete which makes their investigation meaningful proof-complexity wise [30]. Their approach employs a well known result by S.Cook and L. Levin [26, 45] that shows the validity problem for Boolean formulas in disjunctive normal form (DNF) is **coNP**-complete. They then used decisions to define DT formulas that simulated conjunctions and disjunctions, showing that lines in LDT proofs can express DNF properties and thus obtaining the result. We provide a proof of the equivalent statement for valid (positive) sequents in eLNDT$^+$.

Proposition 2.2.14. *The set of valid positive sequents (without extension variables) is* **coNP***-complete.*

Proof. By [26, 45] we know validity of DNFs is **coNP**-complete. We start by expressing conjunctions of propositional variables as eNDT$^+$ formulas. This is simple, observe that a positive decision $0p(0 \vee A)$ is logically equivalent to $p \wedge A$. We can recursively apply this m-times to express an m-ary conjunction of variables as nested positive decisions:

$$\bigwedge_{i=1}^{m} p_i \iff 0p_1(0 \vee 0p_2(0 \vee (\cdots 0p_{m-1}(0 \vee p_m)\cdots))) \tag{2.10}$$

Using the same notation as in [22], we write $\mathrm{Conj}(p_1, \ldots, p_m)$ for the eNDT$^+$ formula on the right-hand side of 2.10.

Now, fix a Boolean formula $A = \bigvee_{i=1}^{n} \bigwedge \mathbf{l}_i$ in DNF where $\mathbf{l}_i$ are lists of literals ranging over propositional variables $p_1, \ldots, p_k$ and their negations. Then, obtain the lists $\mathbf{p}_i$ by substituting each negated literal $l_j = \overline{p}_j$ in $\mathbf{l}_i$, with a fresh one p'_j and construct the following positive sequent that is valid if and only if A is:

$$p_1 \vee p'_1, \ldots, p_k \vee p'_k \longrightarrow \mathrm{Conj}(\mathbf{p}_1), \ldots, \mathrm{Conj}(\mathbf{p}_n)$$

Intuitively, the left hand side of the sequent dictates the semantic behavior of p'_j for $j = 1, \ldots k$ i.e. it forces the clauses to act as cases of 'excluded middle'. Then, any assignment satisfying the formula A can be extended to an assignment satisfying the sequent and vice versa.

More formally: assume that A is valid. Consider any assignment α' on $p_1, p'_1, \ldots, p_k, p'_k$ that satisfies the LHS of the above sequent i.e. $\alpha'(p_j \vee p'_j) = 1$ for all $1 \leqslant j \leqslant k$. We need to prove that $\alpha'(\mathrm{Conj}(\mathbf{p}_i)) = 1$ for some $1 \leqslant i \leqslant n$. Since A is valid, if α is the restriction of α' on $p_1, \ldots, p_k$, it will satisfy $\alpha(\bigwedge \mathbf{l}_i) = 1$ for some i and we claim that $\alpha'(\mathrm{Conj}(\mathbf{p}_i)) = 1$ as well. This is easy to see: for all p_j in $\mathbf{l}_i$, $\alpha(p_j) = \alpha'(p_j) = 1$ since α is the restriction of α' on $p_1, \ldots, p_k$. Furthermore, for all $\overline{p}_j$ in $\mathbf{l}_i$, it holds $\alpha(\overline{p}_j) = 1$ hence, $\alpha(p_j) = \alpha'(p_j) = 0$. Since it must be that $\alpha'(p_j \vee p'_j) = 1$, we obtain that $\alpha(p'_j) = 1$ which concludes that $\alpha'(\mathrm{Conj}(\mathbf{p}_i)) = 1$. For the other direction, assume the sequent above is valid. Then, any assignment α on $p_1, \ldots, p_k$, can be extended to an assignment α' on $p_1, p'_1, \ldots p_k, p'_k$ choosing $\alpha'(p'_j) = 1 - \alpha(p_j)$. By definition, for some i, $\alpha'(\mathrm{Conj}(\mathbf{p}_i)) = 1$. Hence, $\alpha(\bigwedge \mathbf{l}_i) = 1$ and thus $\alpha(A) = 1$. □

We next prove the 'positive' counterpart to Proposition 2.1.14 that is, soundness and completeness for eLNDT^+.

Proposition 2.2.15 (Soundness and completeness)**.** *Given a positive extension free sequent* $\Gamma \longrightarrow \Delta$*,* eLNDT^+ *proves* $\Gamma \longrightarrow \Delta$ *if and only if* $\bigwedge \Gamma \supset \bigvee \Delta$ *is valid.*

Proof sketch. **Soundness**: soundness follows immediately by observing that for all rules, validity (more so truth) of premise(s) implies validity of the conclusion and since the initial sequents are all valid.

Completeness: We proceed by bottom-up cut-free proof search and without using any extension variables. First note that all logical rules are invertible: validity of the conclusion implies validity of premises. Moreover, due to the subformula property, the premises must always be less complex than the conclusion: going bottom-up the number of connectives is strictly reduced. In this way, by applying logical rules as necessary we eventually reach sequents only containing propositional variables and constants. Validity was preserved at each step so the sequents we have obtained must either have a 0 on the left hand side, a 1 on the right hand side or the same variable p on both sides. All such cases can be obtained via initial sequents with some additional weakening steps. □

Remark 2.2.16 (Extended soundness and completeness)**.** Conventionally, proofs using extension conclude with extension-free theorems. It is possible however to extend Proposition 2.2.15 to proofs in which extension variables may appear as formulas in the lines. Given a set of (positive) extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$ for an assignment

α, recall we defined $\alpha \models_{\mathcal{E}} e_i$ if $\alpha \models_{\mathcal{E}} E_i$. An eNDT$^+$ formula A over $e_1, \ldots, e_{n-1}$ is **$\mathcal{E}$-valid** if, for every assignment α, we have $\alpha \models_{\mathcal{E}} A$ (see Definition 2.1.8).

In proving the completeness of eLNDT^+ w.r.t. positive sequents containing extension, the only difference with the completeness proof in Proposition 2.2.15 is that we have to accordingly deal with extension variables: upon reaching (bottom-up) a connective-free sequent with at least one extension variable e_i, we use the corresponding extension axiom $e_i \leftrightarrow E_i$ to unfold e_i into the formula E_i in an 'equivalence' ($\leftrightarrow$) step. The proof then continues in the same manner and termination is guaranteed by the complexity argument from Proposition 2.2.15 and also $\mathcal{E}$-induction, since $E_i <_{\mathcal{E}} e_i$.

2.2.5 Some basic theorems

This section presents basic results of eLNDT^+ that will be necessary to carry out future arguments while exemplifying tools and techniques we will frequently use.

It is natural to first show the positive version of Proposition 2.1.15 that is, a polynomial-size proof of general identity for positive sequents.

Proposition 2.2.17 (General positive identity)**.** *Let $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$ be a positive set of extension axioms. There are polynomial-size eLNDT^+ proofs of $A \longrightarrow A$, for positive formulas A containing only extension variables among $e_0, \ldots, e_{n-1}$.*

Proof. We proceed almost identically to Proposition 2.1.15, only difference being that we are in a positive context. Given eNDT$^+$ formula A, we construct a (dag-like) polynomial-size proof of $A \longrightarrow A$ by $\mathcal{E}$-induction. More precisely, for each such A, we construct a polynomial-size proof containing sequents $B \longrightarrow B$ for each $B \leqslant_{\mathcal{E}} A$ by $\mathcal{E}$-induction on A. We only mention the cases that differ from Proposition 2.1.15.

- If $A = 0$ then we have:

$$\mathsf{w}\text{-}r\,\dfrac{0\,\dfrac{}{0 \longrightarrow}}{0 \longrightarrow 0}$$

- If $A = 1$ then we have:

$$\mathsf{w}\text{-}l\,\dfrac{1\,\dfrac{}{\longrightarrow 1}}{1 \longrightarrow 1}$$

- If A is an extension variable $= e_i$ for some $i < n$, we extend the proof obtained by the inductive hypothesis by adding the step:

$$\mathsf{cut}\,\dfrac{\dfrac{}{e_i \longrightarrow E_i} \quad \dfrac{}{E_i \longrightarrow e_i}}{e_i \longrightarrow e_i}$$

  where the premises are extension axioms from $\mathcal{E}$.

- If $A = Bp(B \vee C)$ then we extend the proof obtained by the inductive hypothesis by adding the step:

$$\dfrac{\dfrac{\dfrac{\dfrac{IH}{B \longrightarrow B}}{B \longrightarrow p, B}\,\text{w-}r \quad \dfrac{\dfrac{IH}{B \longrightarrow B}}{B \longrightarrow B, C}\,\text{w-}r}{B \longrightarrow Bp(B \vee C)}\,p^+\text{-}r \quad \dfrac{\dfrac{\dfrac{\dfrac{}{p \longrightarrow p}\,\text{id}}{p, C \longrightarrow B, p}\,\text{w-}l,\text{w-}r \quad \dfrac{\dfrac{IH}{C \longrightarrow C}}{p, C \longrightarrow B, C}\,\text{w-}l,\text{w-}r}{p, C \longrightarrow Bp(B \vee C)}\,p^+\text{-}r}{Bp(B \vee C) \longrightarrow Bp(B \vee C)}\,p^+\text{-}l$$

In all cases sequents marked *IH* are obtained through the inductive hypothesis.

The same complexity analysis as in Proposition 2.1.15 takes place: at each step of the argument above, we add a constant number of lines of polynomial size in $|A|$ and $|\mathcal{E}|$, see Proposition 2.1.10. The proof must be dag-like to avoid exponential explosion in size due to the possible repetition of sequents. □

We next provide explicit proofs of 'truth conditions' of positive decisions as we will make use of them in many simulation results.

Proposition 2.2.18 (Truth conditions)**.** *Let $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$ be a positive set of extension axioms and let A and B be eNDT$^+$ formulas over $e_0, \ldots, e_{n-1}$. There are polynomial-size* eLNDT$^+$ *proofs of the following sequents with respect to $\mathcal{E}$:*

1. $Ap(A \vee B) \longrightarrow A, p$
2. $Ap(A \vee B) \longrightarrow A, B$
3. $A \longrightarrow Ap(A \vee B)$
4. $p, B \longrightarrow Ap(A \vee B)$

Proof. We provide proofs for all four sequents:

$$1: \dfrac{\dfrac{\dfrac{\dfrac{}{A \longrightarrow A}\,\text{id}}{A \longrightarrow A, p}\,\text{w-}r \quad \dfrac{\dfrac{}{p \longrightarrow p}\,\text{id}}{p, B \longrightarrow A, p}\,\text{w-}l,\text{w-}r}{Ap(A \vee B) \longrightarrow A, p}\,p^+\text{-}l \qquad 2: \dfrac{\dfrac{\dfrac{\dfrac{}{A \longrightarrow A}\,\text{id}}{A \longrightarrow A, B}\,\text{w-}r \quad \dfrac{\dfrac{}{B \longrightarrow B}\,\text{id}}{p, B \longrightarrow A, B}\,\text{w-}l,\text{w-}r}{Ap(A \vee B) \longrightarrow A, B}\,p^+\text{-}l$$

$$3: \dfrac{\dfrac{\dfrac{\dfrac{}{A \longrightarrow A}\,\text{id}}{A \longrightarrow A, p}\,\text{w-}r \quad \dfrac{\dfrac{}{A \longrightarrow A}\,\text{id}}{A \longrightarrow A, B}\,\text{w-}r}{A \longrightarrow Ap(A \vee B)}\,p^+\text{-}r \qquad 4: \dfrac{\dfrac{\dfrac{\dfrac{}{p \longrightarrow p}\,\text{id}}{p, B \longrightarrow A, p}\,\text{w-}l,\text{w-}r \quad \dfrac{\dfrac{}{B \longrightarrow B}\,\text{id}}{p, B \longrightarrow A, B}\,\text{w-}l,\text{w-}r}{p, B \longrightarrow Ap(A \vee B)}\,p^+\text{-}r$$

where steps marked id follow from Proposition 2.2.17. □

Remark 2.2.19. We note that Proposition 2.2.18 could be proven in a more 'high-level' way, without providing explicit proofs for each sequent. Proposition 2.2.17 provides us with polynomial-size proofs of general identity in eLNDT^+ and by Proposition 2.2.15 we know that eLNDT^+ is complete w.r.t. valid positive sequents. We then observe that the sequents 1-4 are valid and consider their 'atomic' instances, i.e. A is replaced by an atom a and B by an atom b. It is clear that these would have constant-size proofs ending in identity steps. Substituting each a and b in these constant-size proofs with the eNDT$^+$ formulas A, B respectively, would provide us with proofs ending in general identity steps, each of which has size polynomial in A, B.

Example 2.2.20 (A positive 'medial')**.** A positive version of Example 2.1.16, can be obtained by taking the positive closures of the BPs by adding 1-edge parallel to every 0-edge. This would result in the following pair of sequents:

$$\begin{array}{rl} & (Aq(A \vee B))p((Aq(A \vee B)) \vee (Cq(C \vee D))) \\ \leftrightarrow & (Ap(A \vee C))q((Ap(A \vee C)) \vee (Bp(B \vee D))) \end{array} \tag{2.11}$$

The validity of this equivalence can be seen by noticing that each side is semantically equivalent to $A \vee (p \wedge C) \vee (q \wedge B) \vee (p \wedge q \wedge D)$ and the LHS and RHS are visualised in Figure 2.8.

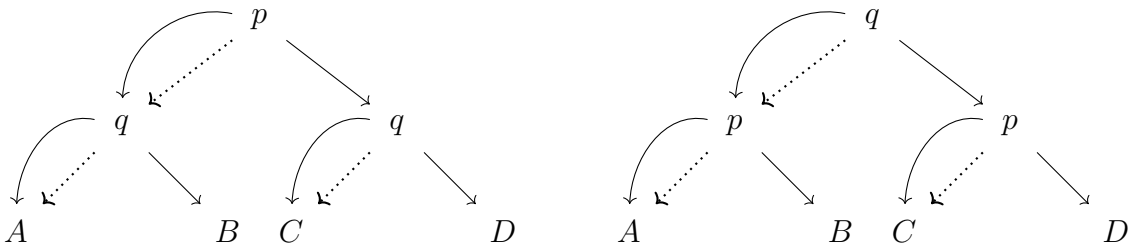


Figure 2.8: The positive 'medial' visualised. The two PBPs compute the same monotone Boolean function despite the change on the order of nodes.

By completeness Proposition 2.2.15 and the substitution argument from Remark 2.2.19, we have polynomial-size proofs of (2.11).

Finally, we will use the truth conditions from Proposition 2.2.18 to show that we can substitute provably-equivalent formulas in a decision in a proof.

Corollary 2.2.21. Given hypotheses $\Gamma, A \longrightarrow \Delta, A'$ and $\Gamma, B \longrightarrow \Delta, B'$, there are polynomial size eLNDT^+ derivations of

$$\Gamma, Ap(A \vee B) \longrightarrow \Delta, A'p(A' \vee B')$$

over any positive set of extension axioms that contains all extension variables occurring in A, A' and B, B'.

Proof. We explicitly provide the derivation:

$$\dfrac{\dfrac{\Gamma, A \longrightarrow \Delta, A' \quad \dfrac{}{A' \longrightarrow A'p(A' \vee B')}\,\text{Proposition 2.2.18.3}}{\Gamma, A \longrightarrow \Delta, A'p(A' \vee B')}\,\mathsf{cut} \quad \dfrac{\Gamma, B \longrightarrow \Delta, B' \quad \dfrac{}{p, B' \longrightarrow A'p(A' \vee B')}\,\text{Proposition 2.2.18.4}}{\Gamma, p, B \longrightarrow \Delta, A'p(A' \vee B')}\,\mathsf{cut}}{\Gamma, Ap(A \vee B) \longrightarrow \Delta, A'p(A' \vee B')}\,p^+\text{-}l$$

Some structural rules have been omitted for simplicity, namely $\mathsf{w}\text{-}l, \mathsf{w}\text{-}r$ on the right premise above cut-steps necessary to match contexts. It will be typical for us to omit similar steps and freely use 'context splitting' and 'context sharing' behaviour, under structural rules. □

Chapter 3

Counting

This chapter formally introduces the Boolean counting functions that appeared in our earlier Example 2.2.7, provides canonical BPs computing them and presents the necessary framework to represent said BPs in our proof systems, m-eLDT, m-eLNDT via formulas with extension.

The **Exact** functions $\mathrm{Ex}_k^n : \{0,1\}^n \to \{0,1\}$ are defined by:

$$\mathrm{Ex}_k^n(b_1, \ldots, b_n) = 1 \quad \Longleftrightarrow \quad \sum_{i=1}^{n} b_i \;=\; k$$

I.e. $\mathrm{Ex}_k^n(b_1, \ldots, b_n) = 1$ if and only if **exactly** k of $b_1, \ldots, b_n$ are 1.

The monotone closures of these functions (see Definition 2.2.4), are the **Threshold** functions $\mathrm{Th}_k^n : \{0,1\}^n \to \{0,1\}$ defined by:

$$\mathrm{Th}_k^n(b_1, \ldots, b_n) = 1 \quad \Longleftrightarrow \quad \sum_{i=1}^{n} b_i \;\geqslant\; k$$

I.e. $\mathrm{Th}_k^n(b_1, \ldots, b_n) = 1$ if and only if *at least* k of $b_1, \ldots, b_n$ are 1.

Staying consistent with the previous exposition from Chapter 2, given a list $\mathbf{p} = p_1, \ldots, p_n$ of propositional variables, we may construe $\mathrm{Ex}_k^n(\mathbf{p}), \mathrm{Th}_k^n(\mathbf{p})$ as Boolean functions from $\mathsf{Ass}^{\mathsf{P}}$ to $\{0,1\}$, and writing $\mathrm{Ex}_k^n(\mathbf{p})(\alpha), \mathrm{Th}_k^n(\mathbf{p})(\alpha)$ for their respective (Boolean) outputs.

3.1 OBDDs for Exact and their representations as eDT formulas

It is well-known that counting functions including those above, are computable by OBDDs (see [63]).

We present a more general version of the OBDD computing the exact 2 out of 4 function seen in Example 2.2.7. That is, in Figure 3.1 we give an OBDD for the exact k out of n function, $\mathrm{Ex}_k^n(p_1, \ldots, p_n)$. Since OBDDs are by definition deterministic, they can be formally represented by m-eDT formulas (i.e. $\vee$-free) by defining an infinite set of extension axioms as follows:

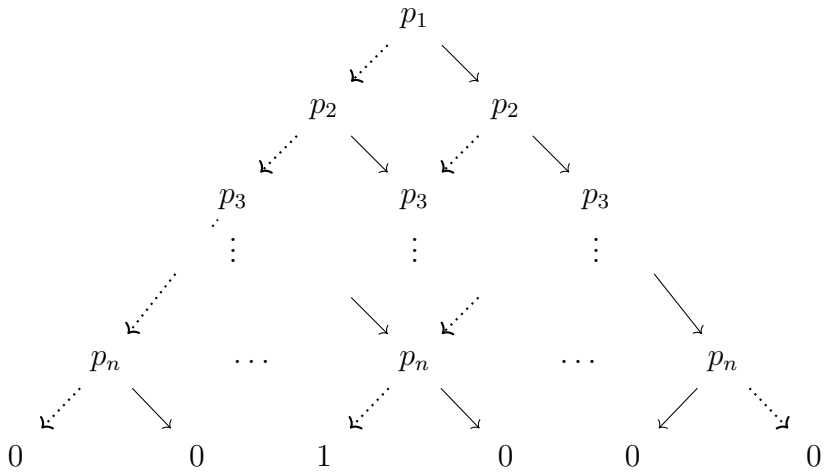


Figure 3.1: An 'ordered' BP/ binary decision diagram (OBDD) for the 'standard' order of propositional variables. It computes the $\mathrm{Ex}_k^n(p_1, \ldots, p_n)$, where only the $k+1$th leaf is a 1 and rest are 0.

Definition 3.1.1 (m-eDTs for Exact)**.** For each list $\mathbf{p}$ of propositional variables , and each integer k, we introduce an extension variable $e_k^{\mathbf{p}}$ and write $\mathcal{X}$ for the set of all extension axioms of the form (i.e. for all choices of p, $\mathbf{p}$ and k),

$$\begin{array}{lcll} e_0^{\varepsilon} & \leftrightarrow & 1 & \\ e_k^{\varepsilon} & \leftrightarrow & 0 & \text{if } k \neq 0 \\ e_k^{p\mathbf{p}} & \leftrightarrow & e_k^{\mathbf{p}} p e_{k-1}^{\mathbf{p}} & \end{array} \tag{3.1}$$

where ε is used for the empty list.

While the extension variables (and the extension axioms) from Definition 3.1.1 do not strictly follow the subscripting conditions from Definition 2.1.7, we may understand them to be 'names' for the appropriate subscripting. It suffices to establish the well-foundedness of $<_{\mathcal{X}}$ as done in Remark 2.1.9, which is clear by induction on the length of the superscript of the extension variables. That is, a variable $e_k^{\mathbf{p}}$ is 'smaller' than a variable $e_l^{\mathbf{q}}$ w.r.t. $<_{\mathcal{X}}$ if $\mathbf{p}$ is a suffix of $\mathbf{q}$. Furthermore, one may extend this well-founded partial order to a well-founded total order by setting $e_k^{\mathbf{p}} <_{\mathcal{X}} e_l^{\mathbf{p}}$ when $k \leqslant l$.

Remark 3.1.2. Since proofs are parametrised by sets of extension axioms, we must be careful about the amount of extra data we consider in particular, it should not be too large (or infinite for that matter). However, $\mathcal{X}$ being an infinite set does not present us with issues regarding proof complexity. In the case we use it as the underlying set of extension axioms in proofs, only finitely many of the extension axioms will actually ever be used as hypotheses in a particular proof. It is implicitly assumed here, that the subscripts of the extension variables are appropriately small (to not contribute significantly to proof complexity) and such that the subscripting condition from Definition 2.1.7 is satisfied. Same applies to all other 'well-founded' sets of extension axioms we may define later. For example, in Section 6.6 we introduce extension variables with seemingly large indices. This does not pose a problem in our setting though. That is because we understand that a proof P in which extension variables have large indices could be

'substituted' by a proof P' that is identical to P with only difference that the (finite) set of extension variables/axioms it uses is considered under appropriate renaming so that extension variables only have indices from $\{1, \ldots, S\}$ for some $S \in \mathbb{N}$.

Proposition 3.1.3. *Let $\mathbf{p} = (p_1, \ldots, p_n)$ be a list of propositional variables. Then, $e_k^{\mathbf{p}}$ computes $\mathrm{Ex}_k^n(\mathbf{p})$, with respect to $\mathcal{X}$.*

Proof. We show that $\alpha \models_{\mathcal{X}} e_k^{\mathbf{p}} \iff \mathrm{Ex}_k^n(\mathbf{p})(\alpha) = 1$ by induction on the length n of the list $\mathbf{p}$. If $n = 0$ then $\mathrm{Ex}_k^0(\varepsilon)$ returns 1 just if $k = 0$, so the result is immediate from the first two axioms of (3.1). For the inductive step we have:

$$
\begin{array}{lll}
\alpha \models_{\mathcal{X}} e_k^{p\mathbf{p}} & \iff & \alpha \models_{\mathcal{X}} e_k^{\mathbf{p}} p e_{k-1}^{\mathbf{p}} \qquad \text{by (3.1) and Definition 2.1.17} \\
& \iff & \begin{cases} \alpha \models_{\mathcal{X}} e_k^{\mathbf{p}} & \alpha(p) = 0 \\ \alpha \models_{\mathcal{X}} e_{k-1}^{\mathbf{p}} & \alpha(p) = 1 \end{cases} \\
& \iff & \begin{cases} \mathrm{Ex}_k^n(\mathbf{p})(\alpha) = 1 & \alpha(p) = 0 \\ \mathrm{Ex}_{k-1}^n(\mathbf{p})(\alpha) = 1 & \alpha(p) = 1 \end{cases} \qquad \text{by inductive hypothesis} \\
& \iff & \mathrm{Ex}_k^{n+1}(p, \mathbf{p})(\alpha) = 1 \qquad \square
\end{array}
$$

3.1.1 Programs for Threshold via positive closure

Notice that since the Exact programs we considered were OBDDs, which are in particular read-once, the semantic characterisation of positive closure by monotone closure from Proposition 2.2.6 applies. Looking back to Figure 3.1, the positive closures we are after (as NBPs) are given in Figure 3.2.

Realising this directly as *m-eNDT* formulas and extension axioms we obtain the following:

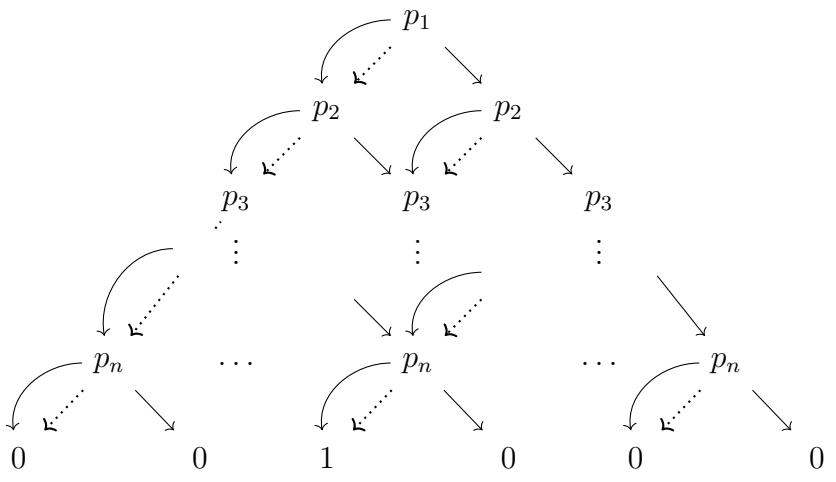


Figure 3.2: The positive closure of the OBDD for Exact from Figure 3.1, computing $\mathrm{Th}_k^n(p_1, \ldots, p_n)$. Again, there are k 0s to the left of the 1, and $n - k$ to the right.

Definition 3.1.4 (Positive *m-eNDT*s for Threshold)**.** For each list $\mathbf{p}$ of propositional variables, and each integer k, we introduce an extension variable $t_k^{\mathbf{p}}$ and write $\mathcal{T}$ for the set of all extension axioms of the form (i.e. for all choices of p, $\mathbf{p}$ and k):

$$\begin{array}{lcll} t_0^{\varepsilon} & \leftrightarrow & 1 & \\ t_k^{\varepsilon} & \leftrightarrow & 0 & \text{if } k \neq 0 \\ t_k^{p\mathbf{p}} & \leftrightarrow & t_k^{\mathbf{p}} p(t_k^{\mathbf{p}} \vee t_{k-1}^{\mathbf{p}}) & \end{array} \tag{3.2}$$

Just as in the case of $\mathcal{X}$ in Remark 3.1.2, even though $\mathcal{T}$ is an infinite set, we shall typically write eLNDT^+ proofs with respect to this set of extension axioms, understanding that only finitely many are ever used in any particular proof. This finite subset will not be explicitly computed , but such a consideration will be subsumed by our analysis of proof complexity.

Note that the relation between the OBDD from Figure 3.1 and the PBP in Figure 3.2 is reflected by the respective extension axioms from $\mathcal{X}, \mathcal{T}$ representing their graphs. That is, the extension variables $t_k^{\mathbf{p}}$ and set of extension axioms $\mathcal{T}$ above, are just the positive closures of $e_k^{\mathbf{p}}$ and $\mathcal{X}$ earlier, within the *m-eNDT* setting. Thus, an immediate consequence of Proposition 2.2.6 is that for each non-negative k, $t_k^{\mathbf{p}}$ computes exactly the threshold function $\mathrm{Th}_k^n(\mathbf{p})$ with respect to $\mathcal{T}$:

Corollary 3.1.5. If $k \geqslant 0$, then $t_k^{\mathbf{p}}$ computes $\mathrm{Th}_k^n(\mathbf{p})$, with respect to $\mathcal{T}$.

We note that there are alternative ways of phrasing Definition 3.1.4. For k negative, we could have alternatively set t_k^{ϵ} to be 1. We could have also simply set $t_0^{\mathbf{p}}$ to be 1 for arbitrary $\mathbf{p}$. Instead, we have chosen to systematically take the positive closure of the aforementioned Exact programs, to make our exposition more uniform.

3.1.2 Small proofs of basic counting properties

The main results we present later heavily rely on having small proofs of characteristic properties of counting formulae. We provide said proofs in this section.

We start with a needed basic monotonicity property for our PBPs computing threshold:

Proposition 3.1.6 ($t_k^{\mathbf{p}}$ is decreasing in k)**.** *There are polynomial-size* eLNDT^+ *proofs of the following sequents over extension axioms* $\mathcal{T}$*:*

1. $\longrightarrow t_0^{\mathbf{p}}$
2. $t_{k+1}^{\mathbf{p}} \longrightarrow t_k^{\mathbf{p}}$
3. $t_k^{\mathbf{p}} \longrightarrow$ *whenever* $k > |\mathbf{p}|$

Proof. We proceed by induction on the length of $\mathbf{p}$ (and sub-induction on k). In the base case, for $\mathbf{p} = \epsilon$, all three sequents follow easily from $\mathcal{T}$ initial sequents, weakening and cuts. For the inductive steps, we construct polynomial-size proofs as follows:

$$
\begin{array}{lll}
 & \longrightarrow t_0^{\mathbf{p}} & \text{by the inductive hypothesis} \\
1: & \longrightarrow t_0^{\mathbf{p}} p(t_0^{\mathbf{p}} \vee t_{-1}^{\mathbf{p}}) & \text{by Proposition 2.2.18.(3)} \\
 & \longrightarrow t_0^{p\mathbf{p}} & \text{by extension axioms } \mathcal{T}
\end{array}
$$

$$
\begin{array}{lll}
t_{k+1}^{p\mathbf{p}} & \longrightarrow t_{k+1}^{\mathbf{p}} x(t_{k+1}^{\mathbf{p}} \vee t_k^{\mathbf{p}}) & \text{by extension axioms } \mathcal{T} \\
2: & \longrightarrow t_k^{\mathbf{p}} p(t_k^{\mathbf{p}} \vee t_{k-1}^{\mathbf{p}}) & \text{by inductive hypotheses and Corollary 2.2.21} \\
 & \longrightarrow t_k^{p\mathbf{p}} & \text{by extension axioms } \mathcal{T}
\end{array}
$$

$$
\begin{array}{lll}
t_k^{p\mathbf{p}} & \longrightarrow t_k^{\mathbf{p}} p(t_k^{\mathbf{p}} \vee t_{k-1}^{\mathbf{p}}) & \text{by extension axioms } \mathcal{T} \\
 & \longrightarrow t_k^{\mathbf{p}}, t_{k-1}^{\mathbf{p}} & \text{by Proposition 2.2.18.(2)} \\
3: & \longrightarrow t_{k-1}^{\mathbf{p}} & \text{by 2 and contraction} \\
 & \longrightarrow & \text{by inductive hypothesis}
\end{array}
$$

The proofs have size polynomial in the length of the list $\mathbf{p}$ and the number k: we perform induction on the length of $\mathbf{p}$ and each time, we require $k+1$ instantiations of all the sequents above, one for each choice of threshold $i \leqslant k$. The same proof complexity analysis will usually suffice for later arguments, in which case we shall suppress it unless further justification is required.

□

Convention 3.1.7. The above proof is presented in 'linear' style instead of the usual tree-like form. We consider the reasoning easier to follow when presented this way and the proof is meant to be read by obtaining each sequent by the justification provided in the right with maybe additional structural steps.

Convention 3.1.8. The second sequent requires using Corollary 2.2.21, which allows us to apply previously proven implications or ($\leftrightarrow$)-equivalences 'deeply' within a decision formula. This 'technique' shall be used throughout this work without further mentioning Corollary 2.2.21 to lighten the exposition.

A Boolean function is called **symmetric** if its value does not depend on the order of its input bits, but only on the number of ones (or zeros) in the input i.e. the output of f remains unchanged under a permutation of its input bits. Equivalently, if two assignments α, β on a finite list of propositional variables send the same number of p_is to 1, then a Boolean n-ary function f is symmetric if and only if $f(\alpha) = f(\beta)$.

Counting functions are known to be symmetric [62]. In particular, the provable symmetry of the formulas $t_k^{\mathbf{p}}$ computing threshold functions, will be essential later on. We shall establish this property through a series of results, beginning by a 'case analysis' on a propositional variable changing places in a list. The proof serves as a prime example within the eLNDT^+ framework as it involves many of previously established results, general positive identity Proposition 2.2.17, positive medial Example 2.2.20 and Corollary 2.2.21.

Lemma 3.1.9 (Case analysis)**.** *There are polynomial-size* eLNDT^+ *proofs of,*

$$t_k^{\mathbf{p}q\mathbf{q}} \leftrightarrow t_k^{q\mathbf{p}\mathbf{q}}$$

over the extension axioms $\mathcal{T}$*.*

Proof. We proceed by induction on the length of $\mathbf{p}$. The base case, when $\mathbf{p}$ is empty, follows immediately by general identity, Proposition 2.2.17.

For the inductive step we construct polynomial-size proofs as follows:

$$
\begin{array}{lll}
 & t_k^{p\mathbf{p}q\mathbf{q}} & \\
\leftrightarrow & t_k^{\mathbf{p}q\mathbf{q}}p(t_k^{\mathbf{p}q\mathbf{q}} \vee t_{k-1}^{\mathbf{p}q\mathbf{q}}) & \text{by } \mathcal{T} \\
\leftrightarrow & t_k^{q\mathbf{p}\mathbf{q}}p(t_k^{q\mathbf{p}\mathbf{q}} \vee t_{k-1}^{q\mathbf{p}\mathbf{q}}) & \text{by } \textit{IH} \text{ and 2.2.21} \\
\leftrightarrow & t_k^{\mathbf{p}\mathbf{q}}q(t_k^{\mathbf{p}\mathbf{q}} \vee t_{k-1}^{\mathbf{p}\mathbf{q}})p(t_k^{\mathbf{p}\mathbf{q}}q(t_k^{\mathbf{p}\mathbf{q}} \vee t_{k-1}^{\mathbf{p}\mathbf{q}}) \vee t_{k-1}^{\mathbf{p}\mathbf{q}}q(t_{k-1}^{\mathbf{p}\mathbf{q}} \vee t_{k-2}^{\mathbf{p}\mathbf{q}})) & \text{by } \mathcal{T} \\
\leftrightarrow & t_k^{\mathbf{p}\mathbf{q}}p(t_k^{\mathbf{p}\mathbf{q}} \vee t_{k-1}^{\mathbf{p}\mathbf{q}})q(t_k^{\mathbf{p}\mathbf{q}}p(t_k^{\mathbf{p}\mathbf{q}} \vee t_{k-1}^{\mathbf{p}\mathbf{q}}) \vee t_{k-1}^{\mathbf{p}\mathbf{q}}p(t_{k-1}^{\mathbf{p}\mathbf{q}} \vee t_{k-2}^{\mathbf{p}\mathbf{q}})) & \text{by Example 2.2.20} \\
\leftrightarrow & t_k^{p\mathbf{p}\mathbf{q}}q(t_k^{p\mathbf{p}\mathbf{q}} \vee t_{k-1}^{p\mathbf{p}\mathbf{q}}) & \text{by } \mathcal{T} \\
\leftrightarrow & t_k^{qp\mathbf{p}\mathbf{q}} & \text{by } \mathcal{T} \quad \square
\end{array}
$$

Similarly to the proof of Proposition 3.1.6, the above argument should be read as providing polynomial-size proofs 'in both directions', by the justifications given on the right. Our use of $\leftrightarrow$-equivalences necessitates that we restrict cedents to contain only a single formula to avoid ambiguity of the comma delimiter. Polynomial proof size on the length of $\mathbf{p}, \mathbf{q}$ and k is, again, immediate by inspection on the number of lines.

The main result of this section is presented next. It states the naturally expected symmetry property of our extension variables in $\mathcal{T}$:

Theorem 3.1.10 (Symmetry)**.** *Let π be a permutation of $\mathbf{p}$. Then there are polynomial-size* eLNDT$^+$ *proofs over the extension axioms $\mathcal{T}$ of:*

$$t_k^{\mathbf{p}} \leftrightarrow t_k^{\pi(\mathbf{p})}$$

Proof. Write $\mathbf{p} = p_1 \cdots p_n$, $\pi(\mathbf{p}) = q_1 \cdots q_n$ and write $\mathbf{p}^{i,\ldots,n}$ for the list $\mathbf{p}$ with the elements $q_i, \ldots, q_n$ removed, otherwise preserving relative order of the propositional variables. We construct polynomial-size proofs by repeatedly applying Lemma 3.1.9 as follows:

$$
\begin{array}{lll}
t_k^{\mathbf{p}} \leftrightarrow & t_k^{q_n\mathbf{p}_n} & \text{by Lemma 3.1.9} \\
\leftrightarrow & t_k^{q_{n-1}q_n\mathbf{p}^{n-1,n}} & \text{by Lemma 3.1.9} \\
\vdots & & \\
\leftrightarrow & t_k^{q_1\cdots q_n} & \text{by Lemma 3.1.9} \\
\leftrightarrow & t_k^{\pi(\mathbf{p})} & \text{by definition of } \mathbf{q}
\end{array}
$$

$\square$

Chapter 4

Pigeons

4.1 Pigeons, holes and their use in proof complexity

The rudimentary combinatorial principle that there is no one to one mapping from a set of finite cardinality m into a set of cardinality n where $m > n$, is known as the *pigeonhole principle*. A common way to state it involves a number of m pigeons which is to be put into n holes for $m > n$, concluding that at least two pigeons will be put together. The version of the pigeonhole principle we are going to work with involves $n + 1$ pigeons and n holes. It is usually encoded in propositional logic by a family (parametrized by n) of $\neg$-free sequents of the following form:

$$\bigwedge_{i=1}^{n+1} \bigvee_{j=1}^{n} p_{ij} \rightarrow \bigvee_{j=1}^{n} \bigvee_{i=1}^{n} \bigvee_{i'=i+1}^{n+1} (p_{ij} \wedge p_{i'j}) \tag{4.1}$$

Here it is useful to think of the propositional variables p_{ij} as expressing "pigeon i nests in hole j". In this way the left-hand side (LHS) above expresses that each pigeon $i \in \{1, \ldots, n+1\}$, nests in some hole $j \in \{1, \ldots, n\}$, and the right-hand side (RHS) expresses that there is some hole j occupied by two (distinct) pigeons i, i'. We remark that this encoding allows the mapping from pigeons to holes to be 'multi-functional', i.e. the LHS allows for a (dismembered[1]) pigeon to nest in multiple holes.

Propositional encodings of the pigeonhole principle have been utilized as 'hard' to prove tautologies to examine the relative strength of different proof systems. Starting from [30], Cook and Reckhow introduced propositional tautologies encoding the pigeonhole principle and showed polynomial-size proofs in extended Hilbert-Frege systems while noting that the only then known non-extended Hilbert-Frege proofs were exponential in size. In [34], it was shown that for sufficiently large n, resolution proofs of the pigeonhole principle have exponential (on n) size lower bounds. [2]Following this, [20] improved the result of [30] by finding polynomial-size proofs for the pigeonhole principle in Hilbert-Frege systems without utilizing extension. Later on in [3], Ajtai showed superpolynomial lowers bounds for proofs of the pigeonhole principle in bounded depth Hilbert-Frege systems. His method uses combinatorial/probabilistic elements and draws

[1]No pigeons were harmed in the making of this thesis.

[2]They use different tautologies encoding that n pigeons put into $n + 1$ holes leaves a hole empty. For a survey on the proof complexity of variations of the PHP see [55].

ideas from Cohen's forcing [25]. Ajtai first proved the result for a non-standard model of Peano Arithmetic and then obtains it for standard values of natural numbers.

The same result was shown in [13], where Bellantoni et al. introduced the notion of an *approximate proof*: inferences of such a proof may only be sound on a subset of the possible truth assignments. Their use of approximate proofs 'simplifies' Ajtai's result, by eliminating the need of using a non-standard model of Peano Arithmetic.

This result was improved in [12], where the authors showed exponential lower bounds on the size of bounded Hilbert-Frege proofs for the pigeonhole principle via a more sophisticated version of Håstad's switching lemma [36]. More recently, in [7], the authors obtained quasi-polynomial upper bounds for the propositional pigeonhole principle, expressed as a monotone sequent in MLK. This bound was improved to polynomial in [8], where the authors considered two well-known variants of the pigeonhole principle: OPHP_n^{n+1} and FPHP_n^{n+1}. In these, the mapping from pigeons to holes is onto and each pigeon is nesting in exactly one hole i.e. the mapping is not multifunctional, respectively. They achieve polynomial size tree-like MLK proofs for OPHP_n^{n+1}, FPHP_n^{n+1} and a perfect matching principle PM_n, concluding that the pigeonhole principle from [7] has polynomial size MLK proofs.

We draw a lot of inspiration from this last result, and adapt some of their techniques (adjusted for our setting) to obtain the material in the following sections. At a high level, what we develop next can be viewed as a case study of the more general simulation result in Chapter 5. That is, we know eLDT polynomially simulates LK, see [22], and thus there are polynomial size eLDT proofs of PHP_n. We are going to exemplify the expressive strength of eLNDT^+ by finding explicit polynomial size dag-like eLNDT^+ proofs of PHP_n.

4.2 Case study: Pigeonhole principle via positive branching programs

The grammar for eNDT$^+$ formulas does not natively express conjunctions, so we adopt a slightly different encoding to 4.1 that fits our setting. Since 4.1 is a sequent, the outermost conjunctions on the LHS can simply be replaced by commas. When it comes to the subformulas $p_{ij} \wedge p_{i'j}$ on the RHS, we observe they may be encoded as $0p_{ij}(0 \vee p_{i'j})$ since they are logically equivalent. This can be justified within eLNDT^+ by the following 'truth conditions' :

Corollary 4.2.1 (Truth conditions for conjunction encoding)**.** The following sequents have polynomial-size eLNDT^+ proofs:

- $p, B \longrightarrow 0p(0 \vee B)$
- $0p(0 \vee B) \longrightarrow p$
- $0p(0 \vee B) \longrightarrow B$

Under these changes, we shall work with the following encoding of the pigeonhole principle throughout this section:

Definition 4.2.2 (Pigeonhole principle)**.** PHP_n is the following positive sequent:

$$\left\{\bigvee_{j=1}^{n} p_{ij}\right\}_{i=1}^{n+1} \longrightarrow \bigvee_{j=1}^{n}\bigvee_{i=1}^{n}\bigvee_{i'=i+1}^{n+1} 0p_{ij}(0 \vee p_{i'j})$$

We write LPHP_n and RPHP_n for the LHS and RHS, respectively, of PHP_n.

The results of this section culminate to proving polynomial-size proofs for the pigeonhole principle in eLNDT^+:

Theorem 4.2.3. *There are polynomial-size* eLNDT^+ *proofs of* PHP_n.

4.3 Summary of proof structure

Before providing an overview of our method, let us introduce some necessary notation:

Notation 4.3.1. We fix $n \in \mathbb{N}$ throughout this section and write:

- $\mathbf{p}_i$ for the list $p_{i1}, \ldots, p_{in}$, and just $\mathbf{p}$ for the list $\mathbf{p}_1, \ldots, \mathbf{p}_{n+1}$.
- $\mathbf{p}^\intercal_j$ for the list $p_{1j}, \ldots, p_{n+1j}$ and just $\mathbf{p}^\intercal$ for the list $\mathbf{p}^\intercal_1, \ldots, \mathbf{p}^\intercal_n$.

The notation $\mathbf{p}^\intercal$ is chosen to be suggestive since, construing $\mathbf{p}$ as an $(n+1) \times n$ matrix of propositional variables, $\mathbf{p}^\intercal$ is just the transpose $n \times (n+1)$ matrix.

At this point it might be helpful to recall the definitions of the extension variables and axiom set $\mathcal{T}$ from Definition 3.1.4 since they will be useful next. Our approach towards finding polynomial-size eLNDT^+ proofs for PHP_n will be broken up into the three smaller steps, proving the following sequents respectively:

1. $\mathsf{LPHP}_n \longrightarrow t^{\mathbf{p}}_{n+1}$
2. $t^{\mathbf{p}}_{n+1} \longrightarrow t^{\mathbf{p}^\intercal}_{n+1}$
3. $t^{\mathbf{p}^\intercal}_{n+1} \longrightarrow \mathsf{RPHP}_n$

Notice that, since $\mathbf{p}^\intercal$ is just a permutation of $\mathbf{p}$, we already have small proofs of sequent 2 from Theorem 3.1.10. In the next two subsections we shall focus on sequents 1 and 3. There, being able to 'merge' and 'split' threshold formulas will be quite useful, hence we provide the following lemma:

Lemma 4.3.2 (Manipulating threshold arguments)**.** *There are polynomial-size* eLNDT^+ *proofs, over extension axioms* $\mathcal{T}$ *of the following sequents:*

1. $t^{\mathbf{p}}_k, t^{\mathbf{q}}_l \longrightarrow t^{\mathbf{pq}}_{k+l}$ *(merging)*
2. $t^{\mathbf{pq}}_{k+l} \longrightarrow t^{\mathbf{p}}_{k+1}, t^{\mathbf{q}}_l$ *(splitting)*

Proof. We proceed by induction on the length of the list $\mathbf{p}$. In the base case, when $\mathbf{p}$ is the empty list ε, we have two cases for 1 for which we provide short derivations:

- if $k = 0$ then $t^{\varepsilon}_0, t^{\mathbf{q}}_l \longrightarrow t^{\mathbf{q}}_l$ follows by id and w-l:

$$\mathsf{w}\text{-}l\,\dfrac{\mathsf{id}\,\dfrac{}{t^{\mathbf{q}}_l \longrightarrow t^{\mathbf{q}}_l}}{t^{\varepsilon}_0, t^{\mathbf{q}}_l \longrightarrow t^{\mathbf{q}}_l}$$

- if $k \neq 0$ then we have an axiom $t^{\varepsilon}_k \longrightarrow 0$ from $\mathcal{T}$, whence $t^{\varepsilon}_k, t^{\mathbf{q}}_l \longrightarrow t^{\mathbf{q}}_{k+l}$ follows by initial step for 0, a cut and a w-l:

$$\mathsf{w}\text{-}l\,\dfrac{\mathsf{cut}\,\dfrac{0\,\dfrac{}{0 \longrightarrow t^{\mathbf{q}}_{k+l}} \quad \dfrac{}{t^{\varepsilon}_k \longrightarrow 0}}{t^{\varepsilon}_k \longrightarrow t^{\mathbf{q}}_{k+l}}}{t^{\varepsilon}_k, t^{\mathbf{q}}_l \longrightarrow t^{\mathbf{q}}_{k+l}}$$

For 2, we have polynomial-size proofs of $t^{\mathbf{q}}_{k+l} \longrightarrow t^{\mathbf{q}}_l$ already from Proposition 3.1.6, whence we obtain $t^{\mathbf{q}}_{k+l} \longrightarrow t^{\varepsilon}_{k+1}, t^{\mathbf{q}}_l$ by w-r.

For the inductive step we will find polynomial-size proofs for the sequents $t^{p\mathbf{p}}_k, t^{\mathbf{q}}_l \longrightarrow t^{p\mathbf{pq}}_{k+l}$ and $t^{p\mathbf{pq}}_{k+l} \longrightarrow t^{p\mathbf{p}}_{k+1}, t^{\mathbf{q}}_l$ for which we shall appeal to Corollary 2.2.21. First, for 1, by the inductive hypothesis we already have a polynomial-size proof of the sequents:

$$\begin{array}{rcl} t^{\mathbf{p}}_k, t^{\mathbf{q}}_l & \longrightarrow & t^{\mathbf{pq}}_{k+l} \\ t^{\mathbf{p}}_{k-1}, t^{\mathbf{q}}_l & \longrightarrow & t^{\mathbf{pq}}_{k+l-1} \end{array}$$

Thus, we can derive,

$$t^{\mathbf{p}}_k p(t^{\mathbf{p}}_k \vee t^{\mathbf{p}}_{k-1}), t^{\mathbf{q}}_l \longrightarrow t^{\mathbf{pq}}_{k+l} p(t^{\mathbf{pq}}_{k+l} \vee t^{\mathbf{pq}}_{k+l-1})$$

from which we obtain the sequent $t^{p\mathbf{p}}_k, t^{\mathbf{q}}_l \longrightarrow t^{p\mathbf{pq}}_{k+l}$ by the extension axioms $\mathcal{T}$.

For 2, by the inductive hypothesis we have a polynomial-size proof of:

$$\begin{array}{rcl} t^{\mathbf{pq}}_{k+l} & \longrightarrow & t^{\mathbf{p}}_{k+1}, t^{\mathbf{q}}_l \\ t^{\mathbf{pq}}_{k+l-1} & \longrightarrow & t^{\mathbf{p}}_k, t^{\mathbf{q}}_l \end{array}$$

Thus, we can derive,

$$t^{\mathbf{pq}}_{k+l} p(t^{\mathbf{pq}}_{k+l} \vee t^{\mathbf{pq}}_{k+l-1}) \longrightarrow t^{\mathbf{p}}_{k+1} p(t^{\mathbf{p}}_{k+1} \vee t^{\mathbf{p}}_k), t^{\mathbf{q}}_l$$

from which we obtain the sequent $t^{p\mathbf{pq}}_{k+l} \longrightarrow t^{p\mathbf{p}}_{k+1}, t^{\mathbf{q}}_l$ by the extension axioms $\mathcal{T}$. □

4.4 From LPHP_n to $(n+1)$-threshold

In this subsection we will give small proofs of the sequent 1 from Section 4.3.

Lemma 4.4.1. *Let* $\mathbf{q} = q_0, \ldots, q_{k-1}$. *For all* $j < k$, *there are polynomial-size* eLNDT^+ *proofs over extension axioms* $\mathcal{T}$ *of:*

$$q_j \longrightarrow t^{\mathbf{q}}_1$$

Proof. First we derive,

$$q_j \leftrightarrow t_1^{q_j} \tag{4.2}$$

as follows:

$$\begin{array}{rcll} q_j & \leftrightarrow & 0q_j(0 \vee 1) & \text{by Proposition 2.2.18, axioms and } \mathsf{cut} \\ & \leftrightarrow & t_1^{\varepsilon} q_j(t_1^{\varepsilon} \vee t_0^{\varepsilon}) & \text{by extension axioms } \mathcal{T} \text{ and Corollary 2.2.21} \\ & \leftrightarrow & t_1^{q_j} & \text{by extension axioms } \mathcal{T} \end{array}$$

By repeatedly applying Lemma 4.3.2.1 we obtain polynomial-size proofs of,

$$t_0^{q_0}, \ldots, t_0^{q_{j-1}}, t_1^{q_j}, t_0^{q_{j+1}}, \ldots, t_0^{q_{k-1}} \longrightarrow t_1^{\mathbf{q}}$$

However, we also have small proofs of $\longrightarrow t_0^{q_j}$ by Proposition 3.1.6.1, and so applying $k-1$ cuts we obtain a polynomial-size proof of:

$$t_1^{q_j} \longrightarrow t_1^{\mathbf{q}} \tag{4.3}$$

The required sequent now follows by simply cutting 4.2 against 4.3. □

Intuitively, the above lemma can be explained as following: a threshold formula $t_1^{\mathbf{q}}$, computes the threshold function with input the list $\mathbf{q}$ and outputs 1 when at least one of the elements of the list evaluates to 1. Hence for any choice of $q_j \in \mathbf{q}$, the sequent $q_j \longrightarrow t_1^{\mathbf{q}}$ is valid. Note that an explicit choice does not need to be made as long as at least one of the variables is evaluated to 1. This is formally proven in the next lemma:

Lemma 4.4.2. *There are polynomial size* eLNDT^+ *proofs of* $\bigvee \mathbf{p}_i \longrightarrow t_1^{\mathbf{p}_i}$.

Proof. Let $i \in \{1, \ldots, n+1\}$. By 4.4.1 above, we have small proofs of,

$$p_{ij} \longrightarrow t_1^{\mathbf{p}_i}$$

for each $j = 1, \ldots, n$. By applying $n-1$ left $\vee$ steps we derive:

$$\bigvee \mathbf{p}_i \longrightarrow t_1^{\mathbf{p}_i} \tag{4.4}$$

□

Proposition 4.4.3. *There are polynomial-size* eLNDT^+ *proofs of 1, i.e.,*

$$\mathsf{LPHP}_n \longrightarrow t_{n+1}^{\mathbf{p}}$$

over extension axioms $\mathcal{T}$.

Proof. Applying Lemma 4.3.2.1 n times (and using cuts), we obtain small proofs of:

$$t_1^{\mathbf{p}_1}, \ldots, t_1^{\mathbf{p}_{n+1}} \longrightarrow t_{n+1}^{\mathbf{p}} \tag{4.5}$$

Finally, by instantiating Equation (4.4) for each $i = 1, \ldots, n+1$ and applying $n+1$ cut steps against Equation (4.5) we derive the required sequent:

$$\bigvee \mathbf{p}_1, \ldots, \bigvee \mathbf{p}_{n+1} \longrightarrow t_{n+1}^{\mathbf{p}}$$

□

4.5 From $(n+1)$-threshold to RPHP_n

Before deriving the final sequent 3 for our proof of PHP_n, we will need some lemmas. In particular the next one establishes expected 'truth conditions' between our threshold formulas and the positive decisions $0q(0 \vee q)_i$ that we used to encode the conjunction $q \wedge q_i$. In the following, it may be convenient to write $A \longrightarrow \{B_i\}_{i<k}$ instead of $A \longrightarrow B_1, \ldots, B_{k-1}$.

Lemma 4.5.1. *Let* $\mathbf{q} = q_0, \ldots, q_{k-1}$. *There are polynomial-size* eLNDT^+ *proofs of,*

$$q, t_1^{\mathbf{q}} \longrightarrow \{0q(0 \vee q_i)\}_{i<k}$$

over extension axioms $\mathcal{T}$.

Proof. For each $i < k$, by Proposition 2.2.18.4 we have a (constant-size) proof of,

$$q, q_i \longrightarrow 0q(0 \vee q_i)$$

and so by cutting against appropriate instances of 4.2 we obtain:

$$q, t_1^{q_i} \longrightarrow 0q(0 \vee q_i)$$

Instantiating the above for each $i < k$ and applying several w-r and $\vee$-l steps we obtain:

$$q, \bigvee_{i<k} t_1^{q_i} \longrightarrow \{0q(0 \vee q_i)\}_{i<k} \tag{4.6}$$

Now, by repeatedly applying Lemma 4.3.2.2 (under cuts) and $\vee$-r steps we obtain polynomial-size proofs of:

$$t_1^{\mathbf{q}} \longrightarrow \bigvee_{i<k} t_1^{q_i} \tag{4.7}$$

Finally, we conclude by cutting 4.7 above against 4.6. □

Lemma 4.5.2. *Let* $\mathbf{q} = q_0, \ldots, q_{k-1}$. *There are polynomial-size* eLNDT^+ *proofs of,*

$$t_2^{\mathbf{p}} \longrightarrow \{0q_i(0 \vee q_{i'})\}_{i<i'<k}$$

over extension axioms $\mathcal{T}$.

Proof. We proceed by induction on the length k of $\mathbf{q}$. In the base case, when $\mathbf{q} = \varepsilon$, we have an axiom $t_2^{\varepsilon} \longrightarrow 0$ from $\mathcal{T}$, whence we conclude by a cut against the 0-axiom and w-r.

For the inductive step, we obtain by two applications of Lemma 4.3.2.2 the following sequents:

$$\begin{array}{rcl} t_2^{q\mathbf{q}} & \longrightarrow & t_1^q, t_2^{\mathbf{q}} \\ t_2^{q\mathbf{q}} & \longrightarrow & t_2^q, t_1^{\mathbf{q}} \end{array}$$

Now we already have small proofs of $t_1^q \longrightarrow q$ from 4.2 and of $t_2^q \longrightarrow$ from Proposition 3.1.6.3, and so cutting against the respective sequents above we obtain:

$$t_2^{q\mathbf{q}} \longrightarrow q, t_2^{\mathbf{q}} \tag{4.8}$$

$$t_2^{q\mathbf{q}} \longrightarrow t_1^{\mathbf{q}} \tag{4.9}$$

Finally, we combine these sequents using cuts as follows:

$$\mathsf{2cut}\,\dfrac{\dfrac{\overset{4.8}{\overline{t_2^{q\mathbf{q}} \longrightarrow q, t_2^{\mathbf{q}}}} \quad \overset{4.9}{\overline{t_2^{q\mathbf{q}} \longrightarrow t_1^{\mathbf{q}}}} \quad \overset{4.5.1}{\overline{q, t_1^{\mathbf{q}} \longrightarrow \{0q(0 \vee q_i)\}_{i<k}}}}{\mathsf{cut}\,\dfrac{t_2^{q\mathbf{q}} \longrightarrow \{0q(0 \vee q_i)\}_{i<k} \qquad \overset{IH}{\overline{t_2^{\mathbf{q}} \longrightarrow \{0q_i(0 \vee q_{i'})\}_{i<i'<k}}}}{t_2^{q\mathbf{q}} \longrightarrow \{0q_i(0 \vee q_{i'})\}_{i<i'<k}}}$$

where the proof marked *IH* is obtained from the inductive hypothesis. □

Proposition 4.5.3. *There are polynomial-size* eLNDT^+ *proofs over* $\mathcal{T}$ *of 3, i.e. of:*

$$t_{n+1}^{\mathbf{p}^\intercal} \longrightarrow \mathsf{RPHP}_n$$

Proof. Recall that $\mathbf{p}^\intercal = \mathbf{p}_1^\intercal, \ldots, \mathbf{p}_n^\intercal$, so by $n-1$ applications of Lemma 4.3.2.2 (and cuts) we have small proofs of:

$$t_{n+1}^{\mathbf{p}^\intercal} \longrightarrow t_2^{\mathbf{p}_1^\intercal}, \ldots, t_2^{\mathbf{p}_n^\intercal} \tag{4.10}$$

Now, instantiating Lemma 4.5.2 with $\mathbf{q} = \mathbf{p}_j^\intercal$ we also have small proofs of,

$$t_2^{\mathbf{p}_j^\intercal} \longrightarrow \{0p_{ij}(0 \vee p_{i'j})\}_{1 \leqslant i < i' \leqslant n} \tag{4.11}$$

for each $j = 1, \ldots, n$. Finally, we may apply n cut steps on 4.10 against each instance of 4.11 (for $j = 1, \ldots, n$) and apply $\vee$-r steps to obtain the required sequent. □

Remark 4.5.4. The reason we consider 'threshold-2' formulas like $t_2^{\mathbf{p}_j^\intercal}$ is connected to the form RPHP_n has. Being on the right side of the sequent arrow, it can be equivalently rewritten as:

$$\bigvee_{i=1}^{n} \bigvee_{i'=i+1}^{n+1} (0p_{i1}(0 \vee p_{i'1})), \ldots, \bigvee_{i=1}^{n} \bigvee_{i'=i+1}^{n+1} (0p_{in}(0 \vee p_{i'n}))$$

or, alternatively:

$$\{0p_{i1}(0 \vee p_{i'1})\}_{1 \leqslant i < i' \leqslant n}, \ldots, \{0p_{in}(0 \vee p_{i'n})\}_{1 \leqslant i < i' \leqslant n}$$

Then, each bracket $\{0p_{ij}(0 \vee p_{i'j})\}_{1 \leqslant i < i' \leqslant n}$ corresponds to the formula $t_2^{\mathbf{p}_j^\intercal}$ as seen in Lemma 4.5.2; intuitively, each bracket 'states' that at least one pair of variables $p_{ij}, p_{i'j}$ for $1 \leqslant j \leqslant n$ must be true.

4.6 Putting it all together

We are now ready to assemble our proofs for PHP_n.

Proof of Theorem 4.2.3. We simply cut together the proofs of 1, 2 and 3 that we have so far constructed:

$$\mathsf{2cut}\,\dfrac{\dfrac{\text{Proposition 4.4.3}}{\mathsf{LPHP}_n \longrightarrow t_{n+1}^{\mathbf{p}}} \quad \dfrac{\text{Theorem 3.1.10}}{t_{n+1}^{\mathbf{p}} \longrightarrow t_{n+1}^{\mathbf{p}^\intercal}} \quad \dfrac{\text{Proposition 4.5.3}}{t_{n+1}^{\mathbf{p}^\intercal} \longrightarrow \mathsf{RPHP}_n}}{\mathsf{LPHP}_n \longrightarrow \mathsf{RPHP}_n}$$

□

Remark 4.6.1. One observes that differently to [7], we achieved polynomial size eLNDT^+ proofs (w.r.t. number of variables) of the pigeonhole principle. The difference comes from the threshold formulas we considered: we used extension to represent the dag-structure of our NBPs which, in Definition 3.1.4 permitted us to define eNDT^+ formulas of size polynomial and hence also provide proofs of size polynomial. On the contrary, the Boolean formulas they used in [7] are quasi-polynomial in size which translated in quasipolynomial sized proofs. They are however able to get a size/depth tradeoff, with their proofs being only logarithmic in depth.

Chapter 5

Simulation of eLNDT by eLNDT$^+$

5.1 Positive simulation of non-positive proofs

In this chapter we consider the system eLNDT over the alternative grammar for m-$eNDT$ formulas, (2.7). Hence, whenever we mention eLNDT one should keep in my that we work with the system m-eLNDT. This poses no issues in light of the results in Section 2.1.4.

In Chapter 4 we formalised counting arguments within the system eLNDT$^+$, which were then used to give polynomial-size proofs for the pigeonhole principle. The same counting arguments will be of use in the next sections, where we provide the necessary steps for a polynomial simulation (over positive sequents) of the system eLNDT by its positive fragment eLNDT$^+$. The main result of this chapter is:

Theorem 5.1.1. eLNDT$^+$ *polynomially simulates* eLNDT *over positive sequents.*

While the high-level structure of the argument is similar to that of [8], we must make several specialisations to the current setting due to the peculiarities of decision formulas and extension axioms.

5.1.1 Summary of proof structure

Before providing technical details, let us go over our strategy towards proving Theorem 5.1.1. Our approach is divided into three main parts, which mimic the analogous proof structure from [8].

In the first part, Section 5.2, we deal with the non-positive formulas occurring in a eLNDT proof. The intuition is similar to what is done in [8] where they first reduced all negations to the variables using De Morgan duality. In our setting formulas are no longer closed under duality (general decisions are self-dual while disjunctions have no dual connective). Nonetheless, we are able to devise for each formula A an appropriate 'positive normal form' A^- which, may contain negative literals (in particular as decision variables), but all decisions themselves are positive.

We duly consider an extension eLNDT$^+_-$ of eLNDT$^+$ which admits negative literals $\overline{p}$ and has two extra initial sequents: $p, \overline{p} \longrightarrow$ and $\longrightarrow p, \overline{p}$. The main result of the first part is that eLNDT$^+_-$ polynomially simulates eLNDT over positive sequents (Corollary 5.2.4). This is achieved by first proving Theorem 5.2.2 where being given a eLNDT

proof P for a sequent $\Gamma \longrightarrow \Delta$ with Γ, Δ multisets of positive formulas, we construct an eLNDT^+_- proof P^- for $\Gamma^- \longrightarrow \Delta^-$ where if $\Gamma = \{A_1, A_2, \ldots\}$ then $\Gamma^- = \{A_1^-, A_2^-, \ldots\}$. Then, employing Lemma 5.2.3 we derive $\Gamma \longrightarrow \Delta$ from $\Gamma^- \longrightarrow \Delta^-$ by finding small proofs for $A \leftrightarrow A^-$, for each A appearing in Γ and Δ. Finally, the wanted result is obtained by several cut-steps.

In the second part the aim is to 'replace' negative literals in an eLNDT^+_- proof by certain (eLNDT^+) formulas computing threshold from Definition 3.1.4. This is the same idea as in [8], but in our setting we must deal with some technicalities encountered when substituting extended formulas in eLNDT^+_- and eLNDT^+. In particular, if a literal occurs as a decision variable, then we cannot directly substitute it for an extension variable (e.g. a threshold formula $t_k^{\mathbf{p}}$), since the syntax of eLNDT (and its fragments) does not allow for this. To handle this issue appropriately, we introduce in Section 5.3 a refinement of our previous threshold extension variables and axioms, defined mutually inductively with eNDT formulas themselves, that accounts for all such substitution situations (Definition 5.3.1).

For the remainder of the argument, in Section 5.4 we fix an eLNDT^+_- proof P of $\Gamma \longrightarrow \Delta$ over extension axioms $\mathcal{E}$ and propositional variables $\mathbf{p} = p_0, \ldots, p_{m-1}$. We define, for each $k \geqslant 0$, systems $\mathsf{eLNDT}^+_k(P)$ that each have polynomial-size proofs P^k of $\Gamma \longrightarrow \Delta$ (Lemma 5.4.5). Morally speaking, this simulation is by 'substituting' negative literals by threshold formulas and the consequent new axioms required in $\mathsf{eLNDT}^+_k(P)$ are parametrised by the threshold k. The number k functions as a lower bound on the number of variables in $\mathbf{p}$ that are required to be true in each case.

We point out that $\mathsf{eLNDT}^+_k(P)$ itself is tailored to the specific set of extension axioms $\mathcal{E}$ and propositional variables $\mathbf{p}$ to facilitate the choice of threshold formulas and required extension variables/axioms.

The final part, Section 5.5, essentially stitches together proofs obtained in each $\mathsf{eLNDT}^+_k(P)$ for $0 \leqslant k \leqslant m+1$. More precisely, using basic properties of threshold formulas, we show that each $\mathsf{eLNDT}^+_k(P)$ proof of a positive sequent $\Gamma \longrightarrow \Delta$ can be polynomially transformed into a eLNDT^+ proof of $t_k^{\mathbf{p}}, \Gamma \longrightarrow \Delta, t_{k+1}^{\mathbf{p}}$, over appropriate extension axioms (Lemma 5.5.2). We conclude the argument for our main result Theorem 5.1.1 by simply cutting these together and appealing to Proposition 3.1.6.

5.2 Positive normal form of eLNDT proofs

In this section we will define the system eLNDT^+_-, an extension of eLNDT^+ that further admits *negative* literals and permits positive decisions on negative literals.

For this reason for each propositional variable p, we consider its **dual** $\overline{p}$, which we shall also refer to as 'negative literal'. The formulas in the eLNDT^+_- system are given by the following grammar:

$$A, B ::= \ 1 \ \mid \ 0 \ \mid \ p \ \mid \ \overline{p} \mid \ Ap(A \vee B) \mid \ A\overline{p}(A \vee B) \ \mid \ A \vee B \ \mid e \qquad (5.1)$$

The system eLNDT^+_- is defined similarly to eLNDT^+: all syntactic positivity constraints on decisions and extension axioms remain.

The rules of eLNDT$^+_-$ include two additional initial sequents:

$$\neg\text{-}l\,\frac{}{p, \overline{p} \longrightarrow} \qquad \neg\text{-}r\,\frac{}{\longrightarrow p, \overline{p}}$$

and two additional positive decision rules on negated atoms accordingly.

$$\overline{p}^+\text{-}l\,\frac{\Gamma, A \longrightarrow \Delta \quad \Gamma, \overline{p}, B \longrightarrow \Delta}{\Gamma, A\overline{p}(A \vee B) \longrightarrow \Delta} \qquad \overline{p}^+\text{-}r\,\frac{\Gamma \longrightarrow \Delta, A, \overline{p} \quad \Gamma \longrightarrow \Delta, A, B}{\Gamma \longrightarrow \Delta, A\overline{p}(A \vee B)}$$

Essentially, the system eLNDT$^+_-$ admits a 'normal form' of eLNDT proofs:

Definition 5.2.1. We define a (polynomial-time) translation from an eNDT formula A to an eNDT$^+_-$ formula A^- as follows:

$$\begin{array}{ll} 0^- := 0 & \\ 1^- := 1 & (A \vee B)^- := A^- \vee B^- \\ p^- := p & (ApB)^- := 0\overline{p}(0 \vee A^-) \vee 0p(0 \vee B^-) \\ \overline{p}^- := \overline{p} & \end{array}$$

For a multiset of formulas $\Gamma = A_1, \ldots, A_n$ we write $\Gamma^- := A_1^-, \ldots, A_n^-$. For a set of extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$, we write $\mathcal{E}^-$ for $\{e_i^- \leftrightarrow E_i^-\}_{i<n}$ where $\{e_i^-\}_{i<n}$ is a fresh set of extension variables.

This section focuses on proving the following:

Theorem 5.2.2. *Let P be a* eLNDT *proof of $\Gamma \longrightarrow \Delta$ over extension axioms $\mathcal{E}$. There is an* eLNDT$^+_-$ *proof P^- of $\Gamma^- \longrightarrow \Delta^-$ over $\mathcal{E}^-$ of size polynomial in $|P|$.*

Notice that, since the translation $\_^-$ commutes with all connectives except for decisions, to prove the above theorem it suffices to just derive the translation of decision steps. For this it will be useful to have another lemma giving us the appropriate 'truth conditions':

Lemma 5.2.3 (Truth conditions for $\_^-$-translation)**.** *Let $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$ be a set of positive extension axioms and let A and B be formulas over $e_0, \ldots, e_{n-1}$. There are polynomial-size* eLNDT$^+_-$ *proofs of the following sequents over $\mathcal{E}^-$:*

1. $(ApB)^- \longrightarrow A^-, p$
2. $(ApB)^-, p \longrightarrow B^-$
3. $A^- \longrightarrow (ApB)^-, p$
4. $p, B^- \longrightarrow (ApB)^-$

Proof. We give the proofs explicitly below:

$$\vee\text{-}l\,\frac{\overline{p}^+\text{-}l\,\frac{\text{2w-}r\,\frac{0\,\frac{}{0 \longrightarrow}}{0 \longrightarrow A^-, p} \quad \text{w-}l,\text{w-}r\,\frac{\text{id}\,\frac{}{A^- \longrightarrow A^-}}{\overline{p}, A^- \longrightarrow A^-, p}}{0\overline{p}(0 \vee A^-) \longrightarrow A^-, p} \quad p^+\text{-}l\,\frac{\text{2w-}r\,\frac{0\,\frac{}{0 \longrightarrow}}{0 \longrightarrow A^-, p} \quad \text{w-}l,\text{w-}r\,\frac{\text{id}\,\frac{}{p \longrightarrow p}}{p, B^- \longrightarrow A^-, p}}{0p(0 \vee B^-) \longrightarrow A^-, p}}{(ApB)^- \longrightarrow A^-, p}$$

$$\dfrac{\dfrac{\dfrac{0\dfrac{}{0 \longrightarrow}}{\text{w-}l,\text{w-}r\dfrac{}{0, p \longrightarrow B^-}}}{\overline{p}^+\text{-}l\dfrac{}{0\overline{p}(0 \vee A^-), p \longrightarrow B^-}} \quad \dfrac{\dfrac{\neg\text{-}l\dfrac{}{\overline{p}, p \longrightarrow}}{\text{w-}l,\text{w-}r\dfrac{}{\overline{p}, A^-, p \longrightarrow B^-}}}{}}{\vee\text{-}l\dfrac{}{(ApB)^-, p \longrightarrow B^-}} \quad \dfrac{\dfrac{\dfrac{0\dfrac{}{0 \longrightarrow}}{\text{w-}l,\text{w-}r\dfrac{}{0, p \longrightarrow B^-}}}{p^+\text{-}l\dfrac{}{0p(0 \vee B^-), p \longrightarrow B}} \quad \dfrac{\dfrac{\mathsf{id}\dfrac{}{B^- \longrightarrow B^-}}{2\text{w-}l\dfrac{}{p, B^-, p \longrightarrow B^-}}}{}}{}$$

$$\dfrac{\dfrac{\dfrac{\neg\text{-}r\dfrac{}{\longrightarrow \overline{p}, p}}{\text{w-}l,\text{w-}r\dfrac{}{A^- \longrightarrow 0\overline{p}, p}} \quad \dfrac{\mathsf{id}\dfrac{}{A^- \longrightarrow A^-}}{2\text{w-}r\dfrac{}{A^- \longrightarrow 0, A^-, p}}}{\overline{p}^+\text{-}r\dfrac{}{A^- \longrightarrow 0\overline{p}(0 \vee A^-), p}}}{\text{w-}r,\vee\text{-}r\dfrac{}{A^- \longrightarrow (ApB)^-, p}} \quad \dfrac{\dfrac{\dfrac{\mathsf{id}\dfrac{}{p \longrightarrow p}}{\text{w-}l,\text{w-}r\dfrac{}{p, B^- \longrightarrow 0, p}} \quad \dfrac{\mathsf{id}\dfrac{}{B^- \longrightarrow B^-}}{\text{w-}l,\text{w-}r\dfrac{}{p, B^- \longrightarrow 0, B^-}}}{\overline{p}^+\text{-}r\dfrac{}{p, B^- \longrightarrow 0p(0 \vee B^-)}}}{\text{w-}r,\vee\text{-}r\dfrac{}{p, B^- \longrightarrow (ApB)^-}}$$

The premises marked by id, are understood to follow from polynomial-size proofs by a generalised identity result for eLNDT$^+_-$. We do not explicitly provide proofs of this since they would be similar to that of Proposition 2.1.15 and Proposition 2.2.17 □

We are now ready to prove our polynomial-size interpretation of eLNDT within eLNDT$^+_-$:

Proof of Theorem 5.2.2. We proceed by a straightforward induction on the length of P. The critical cases are when P ends with decision steps, which we translate as follows. A left decision step,

$$p\text{-}l\dfrac{\Gamma, A \longrightarrow \Delta, p \quad \Gamma, p, B \longrightarrow \Delta}{\Gamma, ApB \longrightarrow \Delta}$$

will be simulated by the following derivation:

$$\mathsf{cut}\dfrac{\mathsf{cut}\dfrac{\dfrac{\text{Lemma 5.2.3.1}}{(ApB)^- \longrightarrow A^-, p} \quad \dfrac{IH}{\Gamma, A^- \longrightarrow \Delta, p}}{\Gamma, (ApB)^- \longrightarrow \Delta, p} \quad \mathsf{cut}\dfrac{\dfrac{\text{Lemma 5.2.3.2}}{(ApB)^-, p \longrightarrow B^-} \quad \dfrac{IH}{\Gamma, p, B^- \longrightarrow \Delta}}{\Gamma, (ApB)^-, p \longrightarrow \Delta}}{\Gamma, (ApB)^- \longrightarrow \Delta}$$

A right decision step,

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

will be simulated by the following derivation:

$$\mathsf{cut}\dfrac{\mathsf{cut}\dfrac{\dfrac{IH}{\Gamma \longrightarrow \Delta, A^-, p} \quad \dfrac{\text{Lemma 5.2.3.3}}{A^- \longrightarrow (ApB)^-, p}}{\Gamma \longrightarrow \Delta, (ApB)^-, p} \quad \mathsf{cut}\dfrac{\dfrac{IH}{\Gamma, p \longrightarrow \Delta, B^-} \quad \dfrac{\text{Lemma 5.2.3.4}}{p, B^- \longrightarrow (ApB)^-}}{\Gamma, x \longrightarrow \Delta, (ApB)^-}}{\Gamma \longrightarrow \Delta, (ApB)^-}$$ □

As a corollary of the above, the $\_^-$ translation gives rise to a bona fide polynomial simulation of eLNDT by eLNDT$^+_-$ over positive sequents:

Corollary 5.2.4. eLNDT$^+_-$ polynomially simulates eLNDT, over positive sequents.

Proof. From Theorem 5.2.2 above, it suffices to derive $\Gamma \longrightarrow \Delta$ from $\Gamma^- \longrightarrow \Delta^-$. For this we shall give short eLNDT$^+_-$ proofs of,

$$A \leftrightarrow A^- \tag{5.2}$$

over $\mathcal{E} \cup \mathcal{E}^-$ when A is positive and free of extension variables. From this, we obtain that $\Gamma \longrightarrow \Delta$ follows from $\Gamma^- \longrightarrow \Delta^-$ by several cuts. We proceed by structural induction on A, for which the critical case is when A is a decision formula (since $\_^-$ commutes on other connectives). The two directions are proven separately.

First, note that we have polynomial-size proofs of the following sequents:

$$Ap(A \vee B) \longrightarrow A, p \qquad \text{by Proposition 2.2.18.1} \tag{5.3}$$

$$Ap(A \vee B) \longrightarrow A \vee B \qquad \text{by Proposition 2.2.18.2 and } \vee\text{-}r \tag{5.4}$$

$$A \longrightarrow (Ap(A \vee B))^-, p \qquad \text{by Lemma 5.2.3.3, } IH \text{ and cut} \tag{5.5}$$

$$p, A \vee B \longrightarrow (Ap(A \vee B))^- \qquad \text{by Lemma 5.2.3.4, } IH \text{ and cut} \tag{5.6}$$

We arrange these into a proof of the left-right direction as follows:

$$\dfrac{\dfrac{5.3 \qquad 5.5}{Ap(A \vee B) \longrightarrow (Ap(A \vee B))^-, p}\,A\text{-cut} \qquad \dfrac{5.4 \qquad 5.6}{Ap(A \vee B), p \longrightarrow (Ap(A \vee B))^-}\,A \vee B\text{-cut}}{Ap(A \vee B) \longrightarrow (Ap(A \vee B))^-}\,p\text{-cut}$$

Next, note that we have small proofs of the following sequents:

$$A \longrightarrow Ap(A \vee B) \qquad \text{by Proposition 2.2.18.3} \tag{5.7}$$

$$p, B \longrightarrow Ap(A \vee B) \qquad \text{by Proposition 2.2.18.4} \tag{5.8}$$

$$(Ap(A \vee B))^- \longrightarrow A, p \qquad \text{by Lemma 5.2.3.1, } IH \text{ and cut} \tag{5.9}$$

$$(Ap(A \vee B))^- \longrightarrow A \vee B \qquad \text{by Lemma 5.2.3.2, } IH \text{ and cut} \tag{5.10}$$

We arrange these into a proof of the left-right direction as follows:

$$\dfrac{\dfrac{5.9 \qquad 5.7}{Ap(A \vee B)^- \longrightarrow Ap(A \vee B), p}\,A\text{-cut} \qquad \dfrac{5.10 \qquad 5.7}{Ap(A \vee B)^-, p \longrightarrow Ap(A \vee B)}\,A \vee B\text{-cut}}{Ap(A \vee B)^- \longrightarrow Ap(A \vee B)}\,p\text{-cut} \qquad \square$$

5.3 Generalised counting formulas

In [8], after 'pushing' negations to variables, they substitute negated literals occuring in proofs of LK by explicitly given monotone formulas that compute threshold functions. While this is entirely unproblematic in their setting (based on Boolean formulas), for us it is more challenging. That is because we are dealing with extension variables that

represent NBPs via extension axioms, and thus handling substitutions is much more subtle and notationally heavy.

We do not treat this uniformly but instead specialise to substituting negated literals for counting formulas. For this we start by carefully giving an appropriate mutually recursive definition of formulas and extension variables.

Definition 5.3.1 (Decisions on thresholds)**.** We introduce extension variables $[At^{\mathbf{p}}_k(A \vee B)]$ for each list $\mathbf{p}$ of propositional variables, integer k, and formulas A, B that may contain extension variables e which will be 'smaller' w.r.t. a partial order defined shortly in in Remark 5.3.2.

We extend $\mathcal{T}$ to include all extension axioms of the following form, with $\mathbf{p}, k, A, B$ ranging as just described:

$$\begin{array}{lcll} [At^{\epsilon}_0(A \vee B)] & \leftrightarrow & A \vee B & \\ [At^{\epsilon}_k(A \vee B)] & \leftrightarrow & A & k \neq 0 \\ [At^{p\mathbf{p}}_k(A \vee B)] & \leftrightarrow & [At^{\mathbf{p}}_k(A \vee B)]p([At^{\mathbf{p}}_k(A \vee B)] \vee [At^{\mathbf{p}}_{k-1}(A \vee B)]) & \end{array} \tag{5.11}$$

Note that, the notation for $[At^{\mathbf{p}}_k(A \vee B)]$ is chosen to be suggestive but, formally speaking, it is a single extension variable, not a decision on the extension variable $t^{\mathbf{p}}_k$ which is not permitted in our setting. The square brackets are used to distinguish it from other formulas, though we shall justify this notation by providing appropriate 'truth conditions' shortly.

One should view the extension variables above and the notion of an eLNDT$^+$ formula as being *mutually* defined, to avoid foundational issues. For instance, we allow an extension variable $[Ct^{\mathbf{p}}_k(C \vee D)]$, where C or D may themselves contain extension variables of the form $[At^{\mathbf{q}}_k(A \vee B)]$. By building up formulas and extension variables by mutual induction we ensure that such constructions are well-founded. This is made formal in the following remark:

Remark 5.3.2 (Well-foundedness of $\mathcal{T}$)**.** We may define the extension variables $[At^{\mathbf{p}}_k(A \vee B)]$ and eLNDT$^+$ formulas 'in stages' as follows:

- Write F_0 for the set of all eLNDT$^+$ formulas over some base set of extension variables $E_0 = \{e_{00}, e_{01}, \dots\}$.
- Write T_n for the set of all extension variables $t^{\mathbf{p}}_k$ and of the form $[At^{\mathbf{p}}_k(A \vee B)]$ with $A, B \in F_n$.
- Write F_{n+1} for the set of all formulas built from propositional variables, disjunctions, positive decisions and extension variables from T_n, E_n and a fresh set of new extension variables $E_{n+1} = \{e_{(n+1)0}, e_{(n+1)1}, \dots\}$.

Within each T_n we (partially well-)order extension variables by the length of the superscript $\mathbf{p}$, and within each E_n we (well-)order the extension variables e_{ni} by the subscript i. We set $E_n < T_n < E_{n+1}$ (i.e. if $e \in E_n$, $t \in T_n$ and $e' \in E_{n+1}$ then $e < t < e'$). In this way $\mathcal{T} = \bigcup_n T_n$ and the extension axioms from 5.11 (and 3.2) indeed satisfy the required well-foundedness criterion. We may also admit further extension axioms allowing elements of E_n to abbreviate formulas in F_n (satisfying the indexing condition internal to E_n), preserving well-foundedness. We shall indeed do this later.

Note, however, that the required order type on indices of these extension variables, a priori, exceeds the ordinal ω. This causes no issue for us since, in any finite proof, we will only use finitely many extension variables, and so may construe each index as a (relatively small) natural number while preserving the aforementioned order. We shall gloss over this issue in what follows.

As previously mentioned, our notation $[At^{\mathbf{p}}_k(A \vee B)]$ is designed to be suggestive and it is justified by the following counterpart of the truth conditions from Proposition 2.2.18:

Proposition 5.3.3 (Truth conditions for threshold decisions)**.** *There are polynomial size* eLNDT$^+$ *proofs over* $\mathcal{T}$ *of:*

1. $[At^{\mathbf{p}}_k(A \vee B)] \longrightarrow A, t^{\mathbf{p}}_k$
2. $[At^{\mathbf{p}}_k(A \vee B)] \longrightarrow A, B$
3. $A \longrightarrow [At^{\mathbf{p}}_k(A \vee B)]$
4. $t^{\mathbf{p}}_k, B \longrightarrow [At^{\mathbf{p}}_k(A \vee B)]$.

Proof. We proceed by induction on the length of $\mathbf{p}$ and make use of previous results including extension axioms from Equation (3.2). For the base case when $\mathbf{p} = \varepsilon$, we have the following proofs:

$$\mathsf{w}\text{-}l,\mathsf{w}\text{-}r\ \dfrac{\mathcal{T},\mathsf{cut}\ \dfrac{1\ \dfrac{}{\longrightarrow 1}}{\longrightarrow t^{\varepsilon}_0}}{[At^{\varepsilon}_0(A \vee B)] \longrightarrow A, t^{\varepsilon}_0} \qquad \mathsf{id},\vee,\mathsf{cut}\ \dfrac{\mathcal{T}\ \dfrac{}{[At^{\varepsilon}_0(A \vee B)] \longrightarrow A \vee B}}{[At^{\varepsilon}_0(A \vee B)] \longrightarrow A, B}$$

$$\mathsf{w}\text{-}r,\vee\text{-}r\ \dfrac{\mathsf{id}\ \dfrac{}{A \longrightarrow A}}{\mathcal{T},\mathsf{cut}\ \dfrac{A \longrightarrow A \vee B}{A \longrightarrow [At^{\varepsilon}_0(A \vee B)]}} \qquad \mathsf{w}\text{-}l,\mathsf{w}\text{-}r,\vee\text{-}r\ \dfrac{\mathsf{id}\ \dfrac{}{B \longrightarrow B}}{\mathcal{T},\mathsf{cut}\ \dfrac{t^{\varepsilon}_0, B \longrightarrow A \vee B}{t^{\varepsilon}_0, B \longrightarrow [At^{\varepsilon}_0(A \vee B)]}}$$

For the inductive step for 1 we derive the following sequents:

$$\begin{array}{rcll}
[At^{\mathbf{p}}_k(A \vee B)] & \longrightarrow & A, t^{\mathbf{p}}_k & \text{by } \mathit{IH} \\
[At^{\mathbf{p}}_{k-1}(A \vee B)] & \longrightarrow & A, t^{\mathbf{p}}_{k-1} & \text{by } \mathit{IH} \\
[At^{\mathbf{p}}_k(A \vee B)]p([At^{\mathbf{p}}_k(A \vee B)] \vee [At^{\mathbf{p}}_{k-1}(A \vee B)]) & \longrightarrow & A, t^{\mathbf{p}}_k p(t^{\mathbf{p}}_k \vee t^{\mathbf{p}}_{k-1}) & \text{by Corollary 2.2.21} \\
[At^{\mathbf{p}}_k(A \vee B)] & \longrightarrow & A, t^{p\mathbf{p}}_k & \text{by } \mathcal{T} \text{ and } 2\mathsf{cut}
\end{array}$$

For the inductive step for 2 we derive the following sequents:

$$\begin{array}{rcll}
[At^{\mathbf{p}}_k(A \vee B)] & \longrightarrow & A, B & \text{by } \mathit{IH} \\
p, [At^{\mathbf{p}}_{k-1}(A \vee B)] & \longrightarrow & A, B & \text{by } \mathit{IH} \text{ and } \mathsf{w}\text{-}l \\
[At^{\mathbf{p}}_k(A \vee B)]p([At^{\mathbf{p}}_k(A \vee B)] \vee [At^{\mathbf{p}}_{k-1}(A \vee B)]) & \longrightarrow & A, B & \text{by } p^+\text{-}l \\
[At^{p\mathbf{p}}_k(A \vee B)] & \longrightarrow & A, B & \text{by } \mathcal{T} \text{ and } \mathsf{cut}
\end{array}$$

For the inductive step for 3 we derive the following sequents:

$$\begin{array}{rcll} A & \longrightarrow & [At_k^{\mathbf{p}}{}_{(A\vee B)}], p & \text{by } IH \text{ and } \mathsf{w}\text{-}r \\ A & \longrightarrow & [At_k^{\mathbf{p}}{}_{(A\vee B)}], [At_{k-1}^{\mathbf{p}}{}_{(A\vee B)}] & \text{by } IH \text{ and } \mathsf{w}\text{-}r \\ A & \longrightarrow & [At_k^{\mathbf{p}}{}_{(A\vee B)}]p([At_k^{\mathbf{p}}{}_{(A\vee B)}] \vee [At_{k-1}^{\mathbf{p}}{}_{(A\vee B)}]) & \text{by } p^+\text{-}r \\ A & \longrightarrow & [At_k^{p\mathbf{p}}{}_{(A\vee B)}] & \text{by } \mathcal{T} \text{ and } \mathsf{cut} \end{array}$$

For the inductive step for 4 we derive the following sequents:

$$\begin{array}{rcll} t_k^{\mathbf{p}}, B & \longrightarrow & [At_k^{\mathbf{p}}{}_{(A\vee B)}] & \text{by } IH \\ t_{k-1}^{\mathbf{p}}, B & \longrightarrow & [At_{k-1}^{\mathbf{p}}{}_{(A\vee B)}] & \text{by } IH \\ t_k^{\mathbf{p}}p(t_k^{\mathbf{p}} \vee t_{k-1}^{\mathbf{p}}), B & \longrightarrow & [At_k^{\mathbf{p}}{}_{(A\vee B)}]p([At_k^{\mathbf{p}}{}_{(A\vee B)}] \vee [At_{k-1}^{\mathbf{p}}{}_{(A\vee B)}]) & \text{by Corollary 2.2.21} \\ t_k^{p\mathbf{p}}, B & \longrightarrow & [At_k^{p\mathbf{p}}{}_{(A\vee B)}] & \text{by } \mathcal{T} \text{ and } 2\mathsf{cut} \quad \square \end{array}$$

5.4 'Substituting' thresholds for negative literals

For the rest of this chapter, let us work with a fixed poly-size eLNDT$^+_-$ proof P, over extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i(e_0, \ldots, e_{i-1})\}_{i<n}$, of a positive sequent $\Gamma \longrightarrow \Delta$ containing propositional variables among $\mathbf{p} = p_0, \ldots, p_{m-1}$ and extension variables among $\mathbf{e} = e_0, \ldots, e_{n-1}$.

Recall that we have already obtained polynomial simulation of eLNDT by eLNDT$^+_-$ in Corollary 5.2.4. Hence, since we are aiming to give a polynomial simulation of eLNDT by eLNDT$^+$ over positive sequents, our consideration of eLNDT$^+_-$ here suffices. We shall also work with the extension axioms $\mathcal{T}$ from Definition 5.3.1 and will soon explain its interaction with $\mathcal{E}$ from P.

Throughout this section, we shall write $\mathbf{p}[x/p_i]$ for $p_0, \ldots, p_{i-1}, x, p_{i+1}, \ldots, p_{m-1}$, i.e. for the list $\mathbf{p}$ but with p_i substituted by $x \in \{0, 1\}$. We also write $\mathbf{p}_i$ for $p_0, \ldots, p_{i-1}, p_{i+1}, \ldots, p_{m-1}$, i.e. for the list $\mathbf{p}$ with the variable p_i removed. We will define an intermediary family of systems eLNDT$^+_k(P)$, one for each $k \geqslant 0$ and prove simulation results between eLNDT$^+_-$ and each such eLNDT$^+_k(P)$. Before that, we need to introduce the following translation of formulas.

Definition 5.4.1 ('Substituting' thresholds)**.** We define a (polynomial-time) translation from an eNDT$^+_-$ formula A (over $\mathbf{p}$, $\overline{\mathbf{p}}$ and $\mathbf{e}$) to an eNDT$^+$ formula A^k (over $\mathbf{p}$, some extension variables $\mathbf{e}^k$ and extension variables from $\mathcal{T}$) as follows:

$$\begin{array}{ll} 0^k := 0 & e_i^k \text{ is a fresh extension variable} \\ 1^k := 1 & (A \vee B)^k := A^k \vee B^k \\ p_i^k := p_i & (Ap_i(A \vee B))^k := A^k p_i(A^k \vee B^k) \\ \overline{p}_i^k := t_k^{\mathbf{p}_i} & (A\overline{p}_i(A \vee B))^k := [A^k t_k^{\mathbf{p}[0/p_i]}{}_{(A^k \vee B^k)}] \end{array}$$

We also define $\mathcal{E}^k := \{e_i^k \leftrightarrow E_i^k(e_0^k, \ldots, e_{i-1}^k)\}_{i<n}$, and for a multiset $\Gamma = B_1, \ldots, B_l$ we write Γ^k for $B_1^k, \ldots, B_l^k$.

Remark 5.4.2. The idea here is replacing the 'unwanted' negated literals with specific eNDT$^+$ formulas, similarly to [8]. The reason replacing a literal $\overline{p_i}$, by a threshold formula $t_k^{\mathbf{p}_i}$ for $k \geqslant 0$ works is given later in Proposition 5.5.1. There, we prove the necessary 'truth conditions' that show that in the context of exactly k of the literals in $\mathbf{p}_i$ being true (i.e. having $t_k^{\mathbf{p}_i}$ on one side and $t_{k+1}^{\mathbf{p}}$ or $t_k^{\mathbf{p}}$ accordingly on the other side of a sequent), we are able to simulate $\overline{p}_i$.

Both this translation and the threshold decisions themselves may seem notationally heavy. However, the underlying idea is quite simple at the level of branching programs: to obtain the NBP represented by A^k, substitute each node labelled by $\overline{p}_i$ in the NBP represented by A, by the NBP represented by $t_k^{\mathbf{p}_i}$. This is better visualised in Section 5.4.

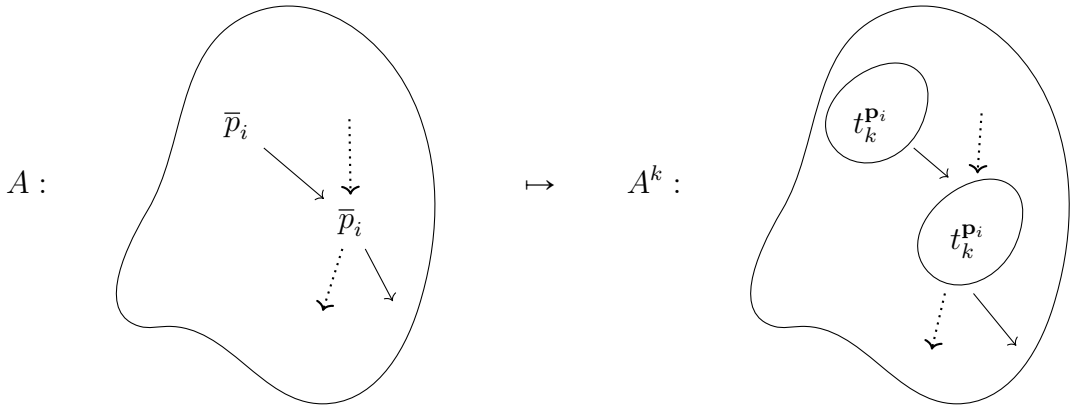


Figure 5.1: Visualising the $\_^k$-translation: the NBP represented by A^k is obtained by substituting $t_k^{\mathbf{p}_i}$ for $\overline{p}_i$ in the NBP represented by A.

Since in what follows, we shall work with the set of extension axioms $\mathcal{T} \cup \mathcal{E}^k$, we must justify its well-foundedness:

Remark 5.4.3 (Well-foundedness of $\mathcal{T} \cup \mathcal{E}^k$)**.** Following on from Remark 5.3.2, well-foundedness of $\mathcal{T} \cup \mathcal{E}^k$ follows from a suitable indexing of the extension variables the union contains. For this, we assign 'stages' to each formula A^k by $\mathcal{E}$-induction on A, using the notation of Remark 5.3.2:

- $p_i^k, 0^k, 1^k \in F_0$.
- $\overline{p}_i^k \in T_0$.
- $e_i^k \in E_m \subseteq F_m$ if $E_i^k(e_0^k, \ldots, e_{i-1}^k) \in F_m$, with index i (i.e., e_i^k is e_{mi}).
- $(A \vee B)^k \in F_m$ if $A^k, B^k \in F_m$.
- $(Ap_i(A \vee B))^k \in F_m$ if $A^k, B^k \in F_m$.
- $(A\overline{p}_i(A \vee B))^k \in T_m \subseteq F_{m+1}$ if $A^k, B^k \in F_m$.

Note in particular that stages can grow even for formulas free of e_i^k since we may have nested decisions on negated variables $\overline{p}_i$.

Once again, in terms of proof complexity, we will gloss over this subtlety and simply count the number of propositional and extension variable occurrences in a proof, assuming that each variable can be equipped with a 'small' index.

We are now ready to define our intermediary systems.

Definition 5.4.4. For $k \geqslant 0$, the system $\mathsf{eLNDT}^+_k(P)$ is defined just like eLNDT^+, but includes additional initial sequents,

$$t\text{-}l\ \frac{}{p_i, t_k^{\mathbf{p}_i} \longrightarrow} \qquad t\text{-}r\ \frac{}{\longrightarrow p_i, t_k^{\mathbf{p}_i}} \tag{5.12}$$

and may only use the extension axioms $\mathcal{T} \cup \mathcal{E}^k$.

Lemma 5.4.5. *There is an* $\mathsf{eLNDT}^+_k(P)$ *proof of* $\Gamma \longrightarrow \Delta$ *of size polynomial in* $|P|$.

Proof. We construct the required proof P^k by replacing every formula occurrence A in the eLNDT^+_- proof P by A^k. Note that all structural steps, identities and cuts remain correct. An extension axiom for e_i from $\mathcal{E}$ is just translated to the corresponding extension axiom for e_i^k from $\mathcal{E}^k$, and the initial sequents $\neg$-l and $\neg$-r from eLNDT^+_- are translated to the two new initial sequents t-l and t-r, respectively, from 5.12 above. It remains to simulate the logical steps.

The simulation of $\vee$ steps is immediate, since the $\_^k$-translation commutes with $\vee$. Similarly for positive decisions on p_i. A left positive decision step on $\overline{p}_i$,

$$\overline{p}_i^+\text{-}l\ \frac{\Gamma, A \longrightarrow \Delta \quad \Gamma, \overline{p}_i, B \longrightarrow \Delta}{\Gamma, A\overline{p}_i(A \vee B) \longrightarrow \Delta}$$

will be simulated by the following derivation:

$$\mathsf{cut}\ \frac{\mathsf{cut}\ \dfrac{\dfrac{}{(A\overline{p}_i(A \vee B))^k \longrightarrow A^k, \overline{p}_i^k}\ \text{Proposition 5.3.3.1} \quad \mathsf{cut}\ \dfrac{\dfrac{}{(A\overline{p}_i(A \vee B))^k \longrightarrow A^k, B^k}\ \text{Proposition 5.3.3.2} \quad \Gamma^k, \overline{p}_i^k, B^k \longrightarrow \Delta}{\Gamma^k, (A\overline{p}_i(A \vee B))^k, \overline{p}_i^k \longrightarrow \Delta^k, A^k}}{\Gamma^k, (A\overline{p}_i(A \vee B))^k \longrightarrow \Delta^k, A^k} \qquad \Gamma^k, A^k \longrightarrow \Delta^k}{\Gamma^k, (A\overline{p}_i(A \vee B))^k \longrightarrow \Delta^k}$$

A right positive decision rule on $\overline{p}_i$,

$$\overline{p}_i^+\text{-}r\ \frac{\Gamma \longrightarrow \Delta, A, \overline{p}_i \quad \Gamma \longrightarrow \Delta, A, B}{\Gamma \longrightarrow \Delta, A\overline{p}_i(A \vee B)}$$

will be simulated by the following derivation:

$$\mathsf{cut}\ \frac{\Gamma^k \longrightarrow \Delta^k, A^k, \overline{p}_i^k \qquad \mathsf{cut}\ \dfrac{\Gamma^k \longrightarrow \Delta^k, A^k, B^k \quad \dfrac{}{\overline{p}_i^k, B^k \longrightarrow (A\overline{p}_i(A \vee B))^k}\ \text{Proposition 5.3.3.4}}{\Gamma^k, \overline{p}_i^k \longrightarrow \Delta^k, A^k, (A\overline{p}_i(A \vee B))^k}}{\Gamma^k \longrightarrow \Delta^k, A^k, (A\overline{p}_i(A \vee B))^k} \qquad \frac{}{A^k \longrightarrow (A\overline{p}_i(A \vee B))^k}\ \text{Proposition 5.3.3.3}$$
$$\mathsf{cut}\ \frac{\Gamma^k \longrightarrow \Delta^k, A^k, (A\overline{p}_i(A \vee B))^k}{\Gamma^k \longrightarrow \Delta^k, (A\overline{p}_i(A \vee B))^k}$$

□

5.5 Putting it all together

We are now ready to assemble the proof for our main simulation result, Theorem 5.1.1. Recall that we are still working with the fixed eLNDT^+_- proof P of a positive sequent $\Gamma \longrightarrow \Delta$ from Section 5.4, over extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$ and propositional variables $\mathbf{p} = p_0, \ldots, p_{m-1}$. We continue to write $\mathbf{p}_i$ for $p_0, \ldots, p_{i-1}, p_{i+1}, \ldots, p_{m-1}$, i.e. $\mathbf{p}$ with p_i removed.

Proposition 5.5.1. *For $k \geqslant 0$, there are polynomial size eLNDT^+ proofs of,*

$$p_i, t_k^{\mathbf{p}_i} \longrightarrow t_{k+1}^{\mathbf{p}} \tag{5.13}$$

$$t_k^{\mathbf{p}} \longrightarrow p_i, t_k^{\mathbf{p}_i} \tag{5.14}$$

over extension axioms $\mathcal{T}$.

Proof. We derive 5.13 as follows:

$$\begin{array}{rcll} t_1^{p_i}, t_{k+1}^{\mathbf{p}_i} & \longrightarrow & t_{k+1}^{p_i \mathbf{p}_i} & \text{by Lemma 4.3.2.1} \\ p_i, t_k^{\mathbf{p}_i} & \longrightarrow & t_{k+1}^{p_i \mathbf{p}_i} & \text{by 4.2 and } \mathsf{cut} \\ & \longrightarrow & t_{k+1}^{\mathbf{p}} & \text{by Lemma 3.1.9 and } \mathsf{cut} \end{array}$$

We derive 5.14 as follows:

$$\begin{array}{rcll} t_k^{\mathbf{p}} & \longrightarrow & t_k^{p_i \mathbf{p}_i} & \text{by Lemma 3.1.9} \\ & \longrightarrow & t_1^{p_i}, t_k^{\mathbf{p}_i} & \text{by Lemma 4.3.2.2 and } \mathsf{cut} \\ & \longrightarrow & p_i, t_k^{\mathbf{p}_i} & \text{by 4.2 and } \mathsf{cut} \quad \square \end{array}$$

Lemma 5.5.2. *For $k \geqslant 0$, there are polynomial size eLNDT^+ proofs of,*

$$t_k^{\mathbf{p}}, \Gamma \longrightarrow \Delta, t_{k+1}^{\mathbf{p}}$$

over extension axioms $\mathcal{T} \cup \mathcal{E}^k$.

Proof. By Lemma 5.4.5, we already have a polynomial-size proof $\mathsf{eLNDT}^+_k(P)$ proof P^k of $\Gamma \longrightarrow \Delta$. Recalling the definition of $\mathsf{eLNDT}^+_k(P)$ (Definition 5.4.4), we may construe P^k as an eLNDT^+ derivation of $\Gamma \longrightarrow \Delta$ over extension axioms $\mathcal{T} \cup \mathcal{E}^k$ from hypotheses:

$$p_i, t_k^{\mathbf{p}_i} \longrightarrow \tag{5.15}$$

$$\longrightarrow p_i, t_k^{\mathbf{p}_i} \tag{5.16}$$

We obtain the required proof by adding $t_k^{\mathbf{p}}$ to the LHS of each sequent and $t_{k+1}^{\mathbf{p}}$ to the RHS of each sequent in P^k. Each local inference step remains correct, except that applying some weakening steps may be required to 'repair' identity steps.[1] Finally we replace occurrences of the hypotheses 5.15 and 5.16 above by the proofs of 5.13 and 5.14 respectively from Proposition 5.5.1. $\square$

[1] An id step $p \longrightarrow p$ in P^k will become $p, t_k^{\mathbf{p}} \longrightarrow p, t_{k+1}^{\mathbf{p}}$ which requires a w-l and a w-r to reach (upwards) the desired id step $p \longrightarrow p$ in the new eLNDT^+ proof.

We are now ready to prove our main result, that eLNDT$^+$ polynomially simulates eLNDT over positive sequents:

Proof of Theorem 5.1.1 . By Corollary 5.2.4, without loss of generality let P be an eLNDT$^+_-$ proof of a positive sequent $\Gamma \longrightarrow \Delta$ over extension axioms $\mathcal{E}$. By Lemma 5.5.2 we construct, for each $k \leqslant m+1$, polynomial-size proofs of $t^{\mathbf{p}}_k, \Gamma \longrightarrow \Delta, t^{\mathbf{p}}_{k+1}$, over $\mathcal{T} \cup \mathcal{E}^k$, and we simply 'cut' them all together as follows:

$$\dfrac{\dfrac{}{\longrightarrow t^{\mathbf{p}}_0}\text{Proposition 3.1.6.1} \quad \dfrac{}{t^{\mathbf{p}}_0, \Gamma \longrightarrow \Delta, t^{\mathbf{p}}_1}\text{Lemma 5.5.2} \quad \cdots \quad \dfrac{}{t^{\mathbf{p}}_m, \Gamma \longrightarrow \Delta, t^{\mathbf{p}}_{m+1}}\text{Lemma 5.5.2} \quad \dfrac{}{t^{\mathbf{p}}_{m+1} \longrightarrow}\text{Proposition 3.1.6.3}}{\Gamma \longrightarrow \Delta}\,(m+2)\mathsf{cut}$$

The resulting proof is an eLNDT$^+$ proof of the required sequent, over extension axioms $\mathcal{T} \cup \mathcal{E}^0 \cup \mathcal{E}^1 \cup \cdots \cup \mathcal{E}^{m+1}$. Note that this set of extension axioms is indeed well-founded, since each $\mathcal{E}^k$ is only used in distinct subproofs. □

Chapter 6

Prover-Adversary games for NBPs

Our goal in this chapter is to present Prover-Adversary games for the systems eLDT, eLNDT similarly to how the Boolean formula game of Pudlák and Buss corresponds to Frege [52]. While we work within the grammar for 0/1-eLNDT (2.6) from Section 2.1.3, we will mostly write eL(N)DT for simplicity, as stated in Remark 2.1.27.

The chapter starts with a preliminary section where the original games from [52] are discussed and a more 'general' framework is introduced. In Sections 6.2 and 6.3, we set up the required technical tools for defining the respective sets of queries and simple contradictions. Then, in Sections 6.4, 6.5 we define the games DB and NB for the systems eLDT and eLNDT respectively and show that comparably to [52], proofs in these systems can be translated to logarithmic depth strategies in the corresponding games (see Theorem 6.5.1). Section 6.6 contains the main technical work of this chapter, our non-uniform (partial) formalisation of the Immerman-Szelepcsényi theorem, $co\mathbf{NL} = \mathbf{NL}$ [37, 58]. Lastly, in Section 6.7 we use intermediate translations to obtain the converse of Theorem 6.5.1 i.e. that from a strategy in DB, NB we can respectively obtain an eLDT, eLNDT proof with exponential increase in the number of symbols as a function of the depth of the strategy.

In what follows, we may write eL(N)DT, (N)BP etc. to mean we are ranging over both deterministic and non-deterministic setting and whatever differences in the treatment of each shall be properly discussed when necessary.

6.1 Preliminaries

6.1.1 Prover-Adversary Games

Prover-Adversary games are a well known class of games inspired by lower bound techniques in complexity theory (adversary arguments) and certain game-theoretical characterizations of circuit complexity measures [51],[64]. The typical version was first formally presented in work by Pudlák and Buss [52]. One of the main reasons they introduce these games is to try prove lower bounds for Frege systems and their restricted versions by investigating the relation of the number of rounds in the game to the number of steps in Frege proofs. They showed that the minimal number of rounds in a 'winning Prover-strategy' is proportional to the logarithm of the minimal number of steps in a Frege proof.

The game from [52] admits Boolean formulas over the basis $B = \{\wedge, \vee, \neg\}$ as **queries** and involves two players, the Prover and the Adversary. Prover's goal is to prove a formula Φ and the aim of Adversary is to pretend (for as long as possible) there exists an assignment of variables so that Φ is false. The game starts with Prover asking a formula Φ and Adversary answering 0. They then alternate turns, Prover asking other formulas and the Adversary assigning values to them, 1 as true and 0 as false. A **simple contradiction** bespoke to B, means that the Adversary gave answers contradicting the truth table for some connective $\diamond \in B$. For example, in case the Adversary answered both ϕ_1 and ϕ_2 by 1 (true) but $\phi_1 \wedge \phi_2$ by 0 (false). For ϕ a query we write $\phi \mapsto 0$ if the Adversary has answered 0 to ϕ and $\phi \mapsto 1$ if 1.

A binary tree is called **almost full** if all non-leaf nodes have two outgoing edges besides the root which has only one. A **Prover-strategy** for Φ can be visualized as an ordered, almost full binary tree rooted at Φ. Its nodes are the queries of the Prover and edges are possible answers of the Adversary. The only edge coming out of the root Φ is labeled by 0. All other non-leaf nodes have two outgoing edges, one labeled by 0 and the other by 1, corresponding to the possible answers the Adversary can give. A possible **play** in a Prover-strategy is a branch of the tree starting at the root. The **size** of a particular (finite) play is the number of rounds in it i.e. number of nodes in that branch. The **depth** of a strategy is naturally the depth of its corresponding binary tree (which may be infinite).

The game is won by the Prover if there is a simple contradiction in the statements of the Adversary. Else, because of the determinacy of the game, Adversary wins. A **winning** Prover-strategy is a finite strategy whose plays lead to leaves that are simple contradictions.

Remark 6.1.1. There is a winning Prover-strategy for Φ if and only if Φ is a tautology.

In general, if the queries of the game are over a functionally complete basis of connectives, the above remark allows viewing the Prover-Adversary game as a propositional proof system and strategies as proofs. With that in mind the authors continue to show:

Proposition 6.1.2 ([52])**.** *The Prover-Adversary game is a complete (and sound) proof system.*

Proof sketch. **Completeness**: Assume Φ is a tautology over B. Starting from the immediate subformulas of Φ, all the Prover has to do is to ask progressively less complex subformulas of Φ until eventually for some connective $\diamond \in B$, the values assigned to $A \diamond B$, A, B contradict the truth table of $\diamond$. We know Φ's truth table has its rightmost column populated only by 1's and Adversary has answered Φ with 0. Hence, by the time all the data to complete the truth table of Φ has been gathered, there must have been a simple contradiction in the statements of Adversary.

Soundness: We will use contrapositive reasoning. If Φ is not a tautology it means there is a falsifying assignment α for Φ. All the Adversary has to do is answer all queries according to α and a simple contradiction cannot be reached.

□

Example 6.1.3. A winning strategy for the formula $\neg\neg(Q \vee \neg Q)$:

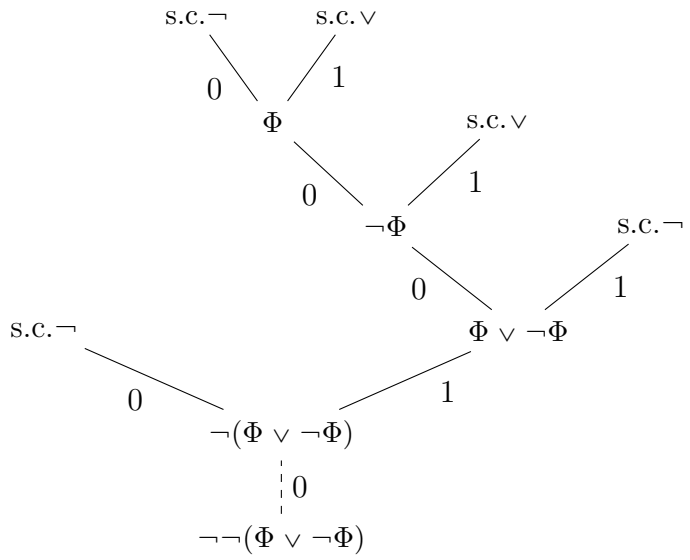


Figure 6.1: Leaves are simple contradictions (s.c.) w.r.t. the noted connective.

The authors of [52] proceed to establish a correspondence between Prover-Adversary games and well known proof systems. For example, winning Prover-strategies in the typical game, correspond to LK proofs with *cut* as the only rule besides the leaves which will be proofs of simple contradictions with a constant number of steps.

Under this correspondence between winning strategies and proofs, a careful survey on the relation between the minimal size of a possible play in a winning strategy and the respective proofs results in:

Proposition 6.1.4 ([52])**.** *The minimal number of rounds in the game needed to prove a formula* Φ *is proportional to the logarithm of the minimal number of steps in a Frege proof of* Φ*.*

In other words, given a Frege proof $\phi_1, \ldots, \phi_k$, a winning Prover-strategy for ϕ_k such that any possible play is of at most $O(\log k)$ rounds can be constructed and vice versa. Given a winning Prover-strategy for Φ, the proof constructed from it will be tree like. By Proposition 6.1.4 one can then obtain:

Corollary 6.1.5 ([41])**.** A Frege proof can be transformed into a tree-like Frege proof with at most polynomial increase in size.

It was observed that restrictions on the form of the queries can impose restrictions on the formulas in the respective proof. For example, in the typical game, allowing only queries of bounded depth corresponds to bounded depth LK proofs.

Another restriction they think interesting is queries to be *monotone* formulas i.e. only using $\wedge, \vee$ connectives. Monotone formulas were considered due to results in circuit complexity. In [53, 54], Razborov restricted circuits computing clique functions to be monotone and proved superpolynomial[1] lower bounds on their size while circuits using negation had only linear lower bounds on size. As a parallel to that, in [52]

[1]Multiple papers have since improved these to exponential lower bounds, [6, 5, 35]

they question if restricting queries in the typical game to be monotone, corresponds to MLK proofs (monotone fragment of LK) and whether they could be used to show polynomial simulation of LK by MLK. We note that the relationship of negation-free games and MLK was unclear at the time and the result: MLK polynomially simulates LK on monotone sequents, was yet to be proven, for more see Section 1.1.

6.1.2 Abstract Pudlák-Buss games

This section formalises a more general version of Pudlák and Buss' games with arbitrary queries and simple contradiction sets. In this manner both Pudlák and Buss' original presentation for Boolean formulas and the games of this work may be admitted in the same abstract framework.

Definition 6.1.6. A **game** G is a pair $(\mathcal{Q}, \mathcal{C})$ where $\mathcal{Q}$ is a set of **queries** and $\mathcal{C}$ is a set of **(simple) contradictions**. A simple contradiction $C \in \mathcal{C}$ is a (finite) set of **assignments** to queries: expressions of the form $Q \mapsto 0$ or $Q \mapsto 1$ for $Q \in \mathcal{Q}$. The **size** of a query is the number of connectives in it (bespoke to the grammar used to define the query). The **depth** of query is the maximum number of nested connectives in it.

Gameplay: The game involves two players, Prover and Adversary. A **state** or **specification** S is a (finite) set of assignments to queries $Q \in \mathcal{Q}$ and the game initializes at a state I. A **play** from I proceeds with a sequence of rounds. In each round Prover first asks a query Q and then Adversary answers with an assignment $Q \mapsto 0$ or $Q \mapsto 1$.

Winning (for Prover). Over the course of a play from I we construct the set $S \supseteq I$ extending I by all the assignments made by Adversary. If at some point of the game there is a simple contradiction C in $\mathcal{C}$ and $C \subseteq S$, the game ends and Prover **wins**[2].

Remark 6.1.7 (Winning for Adversary)**.** An interesting consideration is whether and how it makes sense for the Adversary to win. While out of the scope of this work, we include a definition for completeness sake: Adversary wins if and only if Prover does not, i.e. the play does not end but rather continues forever with the set of responses never containing a simple contradiction.

Definition 6.1.8. Strategies (for Prover). A **Prover-strategy** from S is a rooted almost full binary tree T such that:

- Each internal node of T is labelled by a query;
- The root node has one outgoing edge labelled by 0. Each internal node of T has exactly two outgoing edges, labelled respectively by 0 and 1;
- The leaves of T are labelled by simple contradictions each of whose elements are assignments to queries matching the labels of nodes and edges in the path to the leaf.

[2]Hence, starting at a state I that contains a contradiction $C \in \mathcal{C}$ means the game finishes immediately with Prover winning.

If every maximal path of T is finite, i.e. ends at a leaf, then T is a **winning** Prover-strategy.

Remark 6.1.9 (Winning every play)**.** Observe that the notion of 'winning strategy' for Prover above is consistent with how we defined 'winning play' for Prover and Adversary: Prover wins if (after finitely many rounds) some simple contradiction C is found in the set of assignments to queries given by the Adversary. In contrast, Adversary wins just if the play never ends. That winning Prover-strategies are required to be finite is essentially justified by contraposition on (weak) König's Lemma: if Prover wins on every play of a strategy T, then every play is finite, which means T is itself finite since it is a binary tree.

Over the course of this work we shall (almost always) only concern ourselves with finite (thus winning) Prover-strategies. On the same note, while strategies for Adversary may be defined as (countably branching) trees in a standard way, we shall not concern ourselves with them. Hence, when we say "winning strategy" we leave it implicit that it is a finite Prover strategy.

Games as proof systems

Our games can be seen as proof systems by construing winning strategies as proofs. Formally, given a game G, a **winning strategy** from S is a G**-proof** of the **sequent** $\Gamma \longrightarrow \Delta$ where:

- $\Gamma = \{Q : Q \mapsto 1 \in S\}$
- $\Delta = \{Q : Q \mapsto 0 \in S\}$.

A G-proof of a query Q is just a G-winning strategy from the state $\{Q \mapsto 0\}$.

It is implicit here, in light of the Cook-Reckhow definition of propositional proof system, that the sets $\mathcal{Q}$ and $\mathcal{C}$ we consider throughout this work will be polynomial-time decidable.

Example 6.1.10 (Boolean formula game)**.** We can now view the original game presented by Pudlák and Buss' in[52] in light of our more general definition. They presented a game $\mathsf{B} = (\mathcal{Q}, \mathcal{C})$ with $\mathcal{Q}$ the set of Boolean formulas, and $\mathcal{C}$ the set of assignments inconsistent with a row of the truth table of some connective, i.e. all sets of the form:

- $\{Q \mapsto b, \neg Q \mapsto b\}$ for $b \in \{0, 1\}$.
- $\{P \mapsto b, R \mapsto c, P \vee Q \mapsto d\}$ for $b, c, d \in \{0, 1\}$ with $d \neq \max(b, c)$.
- $\{P \mapsto b, R \mapsto c, P \wedge Q \mapsto d\}$ for $b, c, d \in \{0, 1\}$ with $d \neq \min(b, c)$.

What Pudlák and Buss showed in Proposition 6.1.4 [52] can be slightly rephrased to fit the abstract framework: for a Boolean formula Q, there is a winning strategy from initial state $I = \{Q \mapsto 0\}$ of height $O(\log N)$ if and only if Q has a Frege proof of $O(N)$-many steps. This immediately reduces the completeness of the game $\mathcal{B}$, as a proof system, to that of Frege and also establishes a strong form of polynomial equivalence between the two systems. In this way we say that B **corresponds** to the Frege system.

6.2 'Similar' representations of branching programs

It is our intention to define Prover-Adversary games that reason about (N)BPs in the same way the original game from [52] reasons about Boolean formulas. We represent (N)BPs by e(N)DT formulas which use extension to express the underlying dag structure of (N)BPs. Hence, it is important that we address the matter of equivalent representations of (N)BPs by syntactically different e(N)DT formulas. For example one could use different 'names' for the extension variables but define isomorphic sets of extension axioms so that they represent the same (N)BP. This problem is particular to our setting and it did not occur in the original work by Pudlák and Buss, [52] where queries are Boolean formulas. This can be seen by representing Boolean formulas graphically: they are binary trees and exhibit no dag-like behavior thus requiring no extension.

The games we will define are bootstrapped with native 'winning conditions' that is a notion of equivalence, namely *simulation* of transition systems (refer to [10]) specialised to our syntax.[3] It is in this way that we can see our games as operating *directly* on (N)BPs rather than their representations, equating branching programs simulating each other.

A straightforward way to decide whether two (N)BPs are bisimilar is equality of their unfolding as decision trees. While deterministic and non-deterministic BPs require different treatments (which we will examine soon) the following definition applies to general eNDT formulas.

Definition 6.2.1 (Unfolding)**.** For an eNDT formula A over a set of extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i(e_j)_{j<i}\}_{i<n}$ we define the eNDT formula $\mathtt{Unf}_{\mathcal{E}}(A)$ by :

- $\mathtt{Unf}_{\mathcal{E}}(0) = 0$, $\mathtt{Unf}_{\mathcal{E}}(1) = 1$.
- $\mathtt{Unf}_{\mathcal{E}}(A \vee B) = \mathtt{Unf}_{\mathcal{E}}(A) \vee \mathtt{Unf}_{\mathcal{E}}(B)$.
- $\mathtt{Unf}_{\mathcal{E}}(ApB) = \mathtt{Unf}_{\mathcal{E}}(A)p\mathtt{Unf}_{\mathcal{E}}(B)$.
- $\mathtt{Unf}_{\mathcal{E}}(e_i) = \mathtt{Unf}_{\mathcal{E}}(E_i)$ for all $i < n$.

Remark 6.2.2. Restricting the above to be $\vee$-free i.e. only on eDT formulas, makes the notion of unfolding canonical: there is only one way for a deterministic BP to be unfolded into a deterministic decision tree. On the contrary, the presence of disjunction allows unfolding an NBP into a non-deterministic decision tree using different bracketings of $\vee$ under associativity and commutativity. We demonstrate this in Figure 6.2 where an NBP can be represented by either of the eNDT formulas: $1p(1 \vee 0)$, $1p(0 \vee 1)$.

There are further important distinctions between the deterministic and non-deterministic setting: for BPs , equality of the unfolding of their respective eDT formulas implies bisimilation and vice versa. On the contrary, two eNDT formulas representing NBPs could 'unfold' into different trees while they are still accepting and rejecting on the same assignments. That is because of the existential nature of non-determinism we consider. Hence, a more general definition is necessary to capture the idea of equivalence we want to include in our sets of simple contradictions:

[3]Intuitively, (N)BP G will simulate (N)BP F if every child-node of the root node of F is simulated by a child-node of the root node of G and so on. A sink node labelled by a Boolean b simulates a sink node labelled by b' if and only if $b = b'$

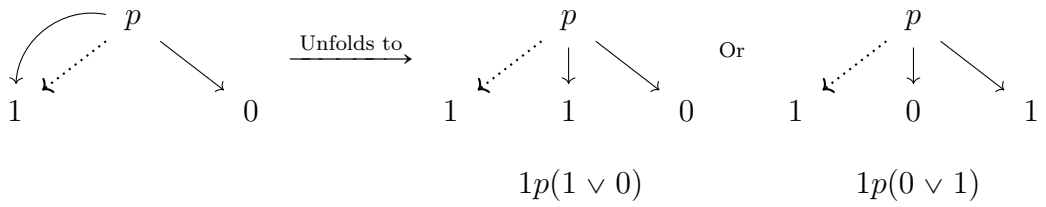


Figure 6.2: Under associativity and commutativity of $\vee$ an NBP may be unfolded into a non-deterministic decision tree in multiple ways.

Definition 6.2.3 (Simulation)**.** Let $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$ be a set of extension axioms. We define the judgement $A \gtrsim_{\mathcal{E}} B$, '$A$ **$\mathcal{E}$-simulates** B', by:

$$\frac{}{A \gtrsim_{\mathcal{E}} A} \quad \frac{A \gtrsim_{\mathcal{E}} C_0 \quad A \gtrsim_{\mathcal{E}} C_1}{A \gtrsim_{\mathcal{E}} C_0 \vee C_1} \quad \frac{A \gtrsim_{\mathcal{E}} E_i}{A \gtrsim_{\mathcal{E}} e_i} \quad \frac{A_j \gtrsim_{\mathcal{E}} B}{A_0 \vee A_1 \gtrsim_{\mathcal{E}} B} \quad \frac{E_i \gtrsim_{\mathcal{E}} B}{e_i \gtrsim_{\mathcal{E}} B} \quad \frac{A \gtrsim_{\mathcal{E}} C \quad B \gtrsim_{\mathcal{E}} D}{ApB \gtrsim_{\mathcal{E}} CpD}$$

where j is either 0 or 1 and $A \gtrsim_{\mathcal{E}} B$ means that the (N)BP represented by A (wrt $\mathcal{E}$) *simulates* the (N)BP represented by B (wrt $\mathcal{E}$), as transition systems. With the above in mind, we plan to include simple contradictions of the form: $[B \mapsto 1, A \mapsto 0]$ for $A \gtrsim B$ in our games.

In the following, we establish the connection between our notions of simulation for formulas A, B with the proof theoretic notion $A \leftrightarrow B$ and elaborate further on the differences between the deterministic vs non-deterministic setting.

Deterministic branching programs

In the deterministic setting simulation between BPs is fairly simple. It reduces to equivalence between BPs with the same unfolding as decision trees.

Proposition 6.2.4. *Let A, B be eDT formulas over a $\vee$-free set of extension axioms $\mathcal{E}$. $A \gtrsim_{\mathcal{E}} B$ if and only if $B \gtrsim_{\mathcal{E}} A$ if and only if $\mathtt{Unf}_{\mathcal{E}}(A) = \mathtt{Unf}_{\mathcal{E}}(B)$.*

Proof. It is enough to show $A \gtrsim_{\mathcal{E}} B$ if and only if $\mathtt{Unf}_{\mathcal{E}}(A) = \mathtt{Unf}_{\mathcal{E}}(B)$. Both directions follow by $\mathcal{E}$ induction on A and B as needed. We only present the $\Rightarrow$ direction according to Definition 6.2.3, ignoring disjunction cases.

- if $A, B \in \{0, 1\}$ the result is obvious since it must be $A = B$.
- for $A \gtrsim_{\mathcal{E}} e_i$ we use $\mathtt{Unf}_{\mathcal{E}}(e_i) = \mathtt{Unf}_{\mathcal{E}}(E_i)$ and $\mathtt{Unf}_{\mathcal{E}}(E_i) = \mathtt{Unf}_{\mathcal{E}}(A)$ by inductive hypothesis.
- for $e_i \gtrsim_{\mathcal{E}} CqD$ we use $\mathtt{Unf}_{\mathcal{E}}(e_i) = \mathtt{Unf}_{\mathcal{E}}(E_i) = \mathtt{Unf}_{\mathcal{E}}(CqD)$ by inductive hypothesis.
- for $ApB \gtrsim_{\mathcal{E}} CpD$, by inductive hypothesis we have: $\mathtt{Unf}_{\mathcal{E}}(A) = \mathtt{Unf}_{\mathcal{E}}(C)$ and $\mathtt{Unf}_{\mathcal{E}}(B) = \mathtt{Unf}_{\mathcal{E}}(D)$. Then, $\mathtt{Unf}_{\mathcal{E}}(ApB) = \mathtt{Unf}_{\mathcal{E}}(A)p\mathtt{Unf}_{\mathcal{E}}(B) = \mathtt{Unf}_{\mathcal{E}}(C)p\mathtt{Unf}_{\mathcal{E}}(D)$ $= \mathtt{Unf}_{\mathcal{E}}(CpD)$. □

Note that for general eNDT formulas A, B over $\mathcal{E}$, only one direction from Proposition 6.2.4 holds, namely: if $\mathtt{Unf}_{\mathcal{E}}(A) = \mathtt{Unf}_{\mathcal{E}}(B)$ then $A \gtrsim_{\mathcal{E}} B$ and $B \gtrsim_{\mathcal{E}} A$.

Proposition 6.2.5 (Cost of Unfolding)**.** *If A, B are two e(N)DT formulas over a set of extension axioms $\mathcal{E} = \{e_i \leftrightarrow E_i(e_j)_{j<i}\}_{i<n}$ and $\mathtt{Unf}_{\mathcal{E}}(A) = \mathtt{Unf}_{\mathcal{E}}(B)$ then there are polynomial size (on $|A|, |B|, |\mathcal{E}|$) $\mathsf{eL(N)DT}$ proofs of the sequents $A \longrightarrow B$ and $B \longrightarrow A$ with hypotheses from $\mathcal{E}$.*

Proof. We start by observing that for two $\mathsf{eL(N)DT}$ formulas A, B if $\mathtt{Unf}_{\mathcal{E}}(A) = \mathtt{Unf}_{\mathcal{E}}(B)$ then either at least one of them is an extension variable or otherwise, they share their outermost connective. For example, if $A = CpD$ then B would be of the form GpF which would in turn require $\mathtt{Unf}(C) = \mathtt{Unf}(G)$ and $\mathtt{Unf}(D) = \mathtt{Unf}(F)$. To prove the result, it is enough to construct (dag-like) proofs for the sequent $A \longrightarrow B$ by $\mathcal{E}$-induction on A and B simultaneously.

- Regarding constants, the only cases are when A, B are both 0 or 1:
- If $A = B = 0$ then we have:

$$\mathsf{w}\text{-}r\,\dfrac{0\,\dfrac{}{0 \longrightarrow}}{0 \longrightarrow 0}$$

- If $A = B = 1$ then we have:

$$\mathsf{w}\text{-}l\,\dfrac{1\,\dfrac{}{\longrightarrow 1}}{1 \longrightarrow 1}$$

- If $A = e_i$ for some $i < n$, then we extend the proof obtained by the inductive hypothesis as follows,

$$\mathsf{cut}\,\dfrac{\mathcal{E}\,\dfrac{}{e_i \longrightarrow E_i} \quad \dfrac{IH}{E_i \longrightarrow B}}{e_i \longrightarrow B}$$

- If $B = e_i$ for some $i < n$, the proof is extended in a similar manner to the above,
- If $A = C \vee D$, $B = G \vee F$ and $\mathtt{Unf}(C) = \mathtt{Unf}(G), \mathtt{Unf}(D) = \mathtt{Unf}(F)$ then we extend the proof obtained by the inductive hypothesis as follows,

$$\vee\text{-}r\,\dfrac{\vee\text{-}l\,\dfrac{\mathsf{w}\text{-}r\,\dfrac{\dfrac{IH}{C \longrightarrow G}}{C \longrightarrow G, F} \quad \mathsf{w}\text{-}r\,\dfrac{\dfrac{IH}{D \longrightarrow F}}{D \longrightarrow G, F}}{C \vee D \longrightarrow G, F}}{C \vee D \longrightarrow G \vee F}$$

- If $A = CpD$, $B = GpF$ and $\mathtt{Unf}(C) = \mathtt{Unf}(G), \mathtt{Unf}(D) = \mathtt{Unf}(F)$ then we extend the proof obtained by the inductive hypothesis as follows:

$$p\text{-}l\,\dfrac{p\text{-}r\,\dfrac{\mathsf{w}\text{-}r\,\dfrac{\dfrac{IH}{C \longrightarrow G}}{C \longrightarrow G, p} \quad \mathsf{w}\text{-}r,\mathsf{w}\text{-}l\,\dfrac{\mathsf{id}\,\dfrac{}{p \longrightarrow p}}{C, p \longrightarrow p, F}}{C \longrightarrow GpF, p} \quad p\text{-}r\,\dfrac{\mathsf{w}\text{-}l,\mathsf{w}\text{-}r\,\dfrac{\mathsf{id}\,\dfrac{}{p \longrightarrow p}}{p, D \longrightarrow G, p} \quad \mathsf{w}\text{-}l\,\dfrac{\dfrac{IH}{D \longrightarrow F}}{p, D \longrightarrow F}}{p, D \longrightarrow GpF}}{CpD \longrightarrow GpF}$$

Where subderivations *IH* are obtained by the inductive hypothesis, and premises marked by $\mathcal{E}$ are extension axioms from $\mathcal{E}$. □

Remark 6.2.6. It is important to note that in the case where a sequent appears multiple times in the proof, we could end up with an exponential explosion in size. Like previously, we treat this by considering only the number of symbols appearing in distinct sequents in the proof, making our proofs dag-like.

Corollary 6.2.7 (Simulation in eLDT)**.** Let A, B be eDT formulas over a $\vee$-free set of extension axioms $\mathcal{E}$. If $A \gtrsim_{\mathcal{E}} B$, then there are polynomial size eLDT proofs of the sequents $A \leftrightarrow B$.

Proof. Since A, B are eDT formulas over $\mathcal{E}$ which is $\vee$-free, the result follows by Proposition 6.2.5 and Proposition 6.2.4. □

Non-deterministic branching programs

NBPs can simulate each other in the sense of Definition 6.2.3 but without having the same unfolding as decision trees i.e. we may have $A \gtrsim_{\mathcal{E}} B$ but $\mathtt{Unf}_{\mathcal{E}}(A) \neq \mathtt{Unf}_{\mathcal{E}}(B)$. Furthermore it may be that $A \gtrsim_{\mathcal{E}} B$ but **not** $B \gtrsim_{\mathcal{E}} A$.[4] For example, consider the graphs in Figures 6.3 and 6.4.

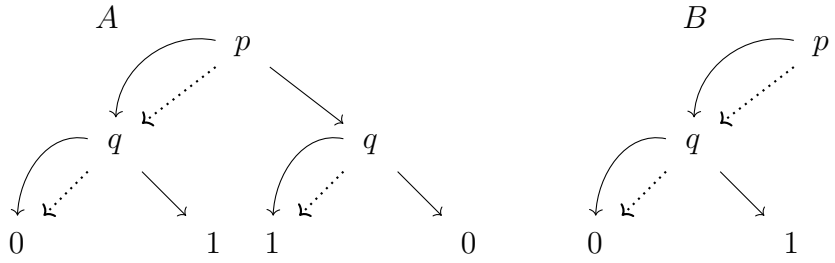


Figure 6.3: In the above, $A \gtrsim_{\mathcal{E}} B$ but not $B \gtrsim_{\mathcal{E}} A$

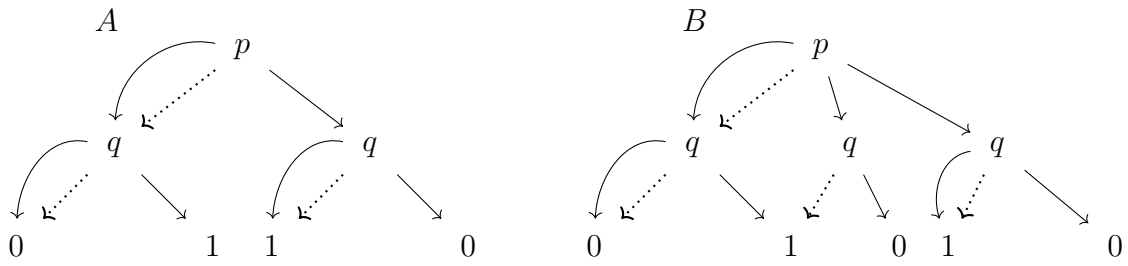


Figure 6.4: In the above, $A \gtrsim_{\mathcal{E}} B$ and $B \gtrsim_{\mathcal{E}} A$ but $\mathtt{Unf}_{\mathcal{E}}(A) \neq \mathtt{Unf}_{\mathcal{E}}(B)$

[4] In particular this happens due to the rule: $\dfrac{A_j \gtrsim_{\mathcal{E}} B}{A_0 \vee A_1 \gtrsim_{\mathcal{E}} B}$.

Thus, we need a non-deterministic analogue to Corollary 6.2.7 for eNDT formulas:

Proposition 6.2.8 (Simulation in eLNDT)**.** *Let $\mathcal{E}$ be a set of eNDT extension axioms and suppose $A \gtrsim_{\mathcal{E}} B$. Then, we can construct polynomial-size* eLNDT *proofs of $B \longrightarrow A$ over $\mathcal{E}$.*

Proof. We proceed by $\mathcal{E}$-induction according to the definition of simulation. The proof is similar to that of Proposition 6.2.5 hence we omit most details.

- if $b \in \{0, 1\}$, for $b \gtrsim_{\mathcal{E}} b$ the corresponding proofs have constant size.
- for $A \gtrsim_{\mathcal{E}} C \vee D$ we just use a $\vee$-left step from derivations of $C \longrightarrow A$ and $D \longrightarrow A$ obtained by inductive hypothesis.
- for $A \gtrsim_{\mathcal{E}} e_i$ we cut the extension axiom $e_i \longrightarrow E_i$ against a derivation of $E_i \longrightarrow A$ obtained by the inductive hypothesis.
- for $A_0 \vee A_1 \gtrsim_{\mathcal{E}} CqD$ we apply a weakening and $\vee$-right step to the derivation of $CqD \gtrsim_{\mathcal{E}} A_i$ obtained by the inductive hypothesis.
- for $e_i \gtrsim_{\mathcal{E}} CqD$ we cut the extension axiom $E_i \longrightarrow e_i$ against the derivation of $CqD \longrightarrow E_i$ obtained by the inductive hypothesis.
- for $ApB \gtrsim_{\mathcal{E}} CpD$ we construct a derivation just like the one from Proposition 6.2.5:

$$\dfrac{\dfrac{\dfrac{\dfrac{IH}{C \longrightarrow A}}{C \longrightarrow A, p}\,\text{w-}r \qquad \dfrac{\dfrac{}{p \longrightarrow p}\,\text{id}}{C, p \longrightarrow p, B}\,\text{w-}r,\text{w-}l}{C \longrightarrow ApB, p}\,p\text{-}r \qquad \dfrac{\dfrac{\dfrac{}{p \longrightarrow p}\,\text{id}}{p, D \longrightarrow A, p}\,\text{w-}l,\text{w-}r \qquad \dfrac{\dfrac{IH}{D \longrightarrow B}}{p, D \longrightarrow B}\,\text{w-}l}{p, D \longrightarrow ApB}\,p\text{-}r}{CpD \longrightarrow ApB}\,p\text{-}l$$

  where the subderivations *IH* are obtained by the inductive hypothesis. □

6.3 Boolean combinations and negating NBPs

The next matter we must address is to some extent a design choice. To prove equivalence of our games and eL(N)DT, first recall from Section 1.2 that games are essentially tree-like versions of the inference system they correspond to, so we shall need to prove some sort of closure under Boolean combinations, as in [41]. There, they work with a specifically formulated grammar for LK for technical reasons. The main feature of this grammar is admitting 'big' disjunctions $\bigvee$ and conjunctions $\bigwedge$ of unbounded arity and boolean combinations of thereof[5]. The reason we will need a similar convention (adjusted for our connectives) comes from a technique used in showing polynomial equivalence between LK and tree-LK (generally, polynomially simulating a dag-structure by a tree-structure)

[5]Regardless of these differences this formulation of LK is polynomially equivalent to the usual LK

Briefly explained, for an $\mathsf{eL(N)DT}$ proof P with conclusion $\Gamma \longrightarrow \Delta$, we construe each line of P as a single query in the corresponding game. For a sequent $\Gamma_i \longrightarrow \Delta_i$, this involves defining the conjunction of all formulas in Γ_i, $\bigwedge \Gamma_i$, the disjunction of all formulas in Δ_i, $\bigvee \Delta_i$ and then considering the implication $\bigwedge \Gamma_i \supset \bigvee \Delta_i$. Under De Morgan duality, this can be rewritten as $\neg \bigwedge \Gamma_i \vee \bigvee \Delta_i$. We then find a strategy for $\neg \bigwedge \Gamma \vee \bigvee \Delta$ by applying a divide-and-conquer method on conjunctions of lines in P. This way, from a proof of N steps, we can construct a corresponding winning Prover-strategy of at most $O(\log N)$ rounds whose queries are boolean combinations of e(N)DT formulas.

Doing the converse i.e. translating strategies in Prover-Adversary games to $\mathsf{eL(N)DT}$ proofs crucially requires defining negation of (N)BPs. This is not hard for the system eLDT, mostly because the negation of a deterministic BP G is trivial: it is the same BP but with all its leaves flipped (0 to 1 and 1 to 0). For eLNDT, due to non-determinism, finding the negation of a formula A is much more involved, in particular requiring the formalisation of a (nonuniform) version of the Immerman-Szelepcsényi Theorem: $\mathbf{NL} = co\mathbf{NL}$. This could be carried out either within a game system or within an inference system. However, doing so within games requires us to again be resource conscious of the number of rounds, which must be logarithmic. It is not clear (to us) that this can be duly carried out without further bootstrapping. Coupled with our intention that an appropriate game system should serve as a feature-rich way of *reasoning about* $\mathsf{eL(N)DT}$, we choose to expand the queries of our game to be as expressive as possible, closing them under Boolean combinations. This has the effect of simplifying the translation from $\mathsf{eL(N)DT}$-proofs to strategies, but rendering the converse more cumbersome.

Definition 6.3.1 (Boolean combinations)**.** We write P, Q, R for **Boolean combinations** of e(N)DT formulas (Bool(e(N)DT)-formulas), generated by:

$$P, Q, R, \ldots \quad ::= \quad A \mid \neg Q \mid (Q \vee R) \mid (Q \wedge R)$$

where A is an eNDT formula. We use other Boolean connectives to denote their usual abbreviations, for example $P \supset Q := \neg P \vee Q$.

The intended semantics of this extended class of formulas is again parametrised by a set of extension axioms in order to interpret the extension variables in an e(N)DT-subformula. Note that the syntax here is 'two-tiered': Boolean connectives may not alternate with decisions and extensions (except $\vee$ in the case of Bool(eNDT)-formulas). For example our grammar does **not** admit decisions of the form $(P_0 \vee P_1)q(R_0 \vee R_1)$ for P_j, R_j Bool(e(N)DT)-formulas. This is reflected by the use of different metavariables (P, Q, R etc.) for Boolean combinations of branching programs. Later we consider intermediate systems that extend $\mathsf{eL(N)DT}$ by (positive) Boolean combinations, interpolating the translation from strategies to $\mathsf{eL(N)DT}$.

6.4 Games for (non-deterministic) branching programs

We briefly recall some of the material presented in Section 6.1.2. A **game** is given by a set $\mathcal{Q}$ of **queries** and a set $\mathcal{C}$ of **(simple) contradictions** which are sets of Boolean

assignments to queries. A **state** is a finite set of assignments to queries which we usually denote by S and a **strategy** is binary tree whose inner nodes are labelled by queries and leaves are simple contradictions. We usually denote a strategy by T. The mechanics of the Prover-Adversary game are defined as follows: the game initialises at a state I, then Prover asks queries, and Adversary assigns them a value, 0 or 1. During the game additional assignments (Adversary's answers) are added to I extending it to a state S. If during a play the set S of accumulated assignments contains a simple contradiction then Prover wins.[6] A **strategy** (for Prover) from S is represented as an almost full binary tree in the natural way. Furthermore, by viewing games as proof systems, we call a winning strategy from state $\{A_1 \mapsto 1, \ldots, A_m \mapsto 1, B_1 \mapsto 0, \ldots, B_n \mapsto 0\}$ a **proof** of the sequent $A_1, \ldots, A_m \longrightarrow B_1, \ldots, B_n$.

Let us now present our games corresponding to the system $\mathsf{eL(N)DT}$:

Definition 6.4.1. [(N)BP games] The game NB (wrt $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$) is defined in the following way:

- Queries are Bool(eNDT)-formulas.
- Simple contradictions consist of just:
  - Decisions: $\{A_0 \mapsto b_0, A_1 \mapsto b_1, p \mapsto i, A_0 p A_1 \mapsto c : c \neq b_i\}$ for $b_0, b_1, i, c \in \{0, 1\}$.
  - Boolean connectives: all sets contradicting the row of a truth table for $\neg, \vee, \wedge$.
  - Contradictions for extension: $\{e_i \mapsto b, E_i \mapsto 1 - b\}$, for $b \in \{0, 1\}$.
  - Contradictions for similarity: $\{A \mapsto 0, B \mapsto 1 : A \gtrsim_{\mathcal{E}} B\}$

The game DB (wrt $\mathcal{E}$) is defined the same way, only wrt Bool(eDT) instead of Bool(eNDT).

6.5 From proofs to strategies

This section focuses on proving the following result:

Theorem 6.5.1. *If* $\mathsf{eL(N)DT}$ *has a proof of* N*-many steps for a sequent* $\Gamma \longrightarrow \Delta$*, there is a* $O(\log N)$*-round winning strategy for* $\Gamma \longrightarrow \Delta$ *in* NB *(or* DB*, resp.).*

Thanks to the Boolean combinations we just defined, this direction of the correspondence between NB and eLNDT , follows by essentially the same proof structure as Pudlák and Buss in [52]. The idea, as explained before, is similar to the translation of dag-like Frege proofs into tree-form [41]: each line is construed as a single query by Boolean combinations, and the winning strategy searches for the first false sequent by a divide-and-conquer search on conjunctions of lines. This process will eventually contradict the soundness of some rule, which is again won in logarithmically many rounds.

For the rest of this subsection we fix a set of extension axioms $\mathcal{E}$ over which all proofs and strategies are formulated. We also may state definitions and results without specifying the system eLDT or eLNDT since the exposition works for both.

[6] In this work, we do not concern ourselves with situations where Adversary wins.

Long combinations: forcing and finding

For a list of formulas Γ we write $\bigwedge \Gamma$ for the conjunction of its formulas bracketed as a (nearly) balanced binary tree. Formally, if $\Gamma = A_1, \ldots, A_k$ (and $k \geqslant 2$) then:

$$\bigwedge \Gamma := \bigwedge_{i=1}^{\lfloor k/2 \rfloor} A_i \ \wedge \bigwedge_{i=\lfloor k/2 \rfloor + 1}^{k} A_i$$

Similarly for long disjunctions, $\bigvee \Gamma$. For the sake of legibility, we shall almost always assume that such lists have length a power of 2, e.g. by repeating elements, at the cost of at most doubling the length.

When describing strategies, we shall use some suggestive terminology:

- "**force** $Q \mapsto b$ (or win) from S in r rounds" is a (partial) strategy initialised at state S of depth $\leqslant r$ where each leaf is either a simple contradiction or it has incoming edge labelled $Q \mapsto b$;
- "**find** $\alpha \in T$ (or win) from S in r rounds" is a (partial) strategy initialised at state S of depth $\leqslant r$ where each leaf is either a simple contradiction or it has incoming edge labelled according to some assignment α in T.

We almost always omit 'or win' when using these phrases. We shall often expand out '$\alpha \in T$' according to the context, e.g. saying 'find $Q \in \Gamma$ with $Q \mapsto b$'.

For instance, from $\{Q_0 \vee Q_1 \mapsto 1\}$ we can find $Q_i \mapsto 1$ in constantly many rounds by:

- ask Q_0; if $Q_0 \mapsto 1$ we are done; else,
- ask Q_1; if $Q_1 \mapsto 1$ we are done; else,
- we have a simple contradiction $\{Q_0 \vee Q_1 \mapsto 1, Q_0 \mapsto 0, Q_1 \mapsto 0\}$.

Also from $\{Q \supset R \mapsto 0\}$ we can force $Q \mapsto 1$ and $R \mapsto 0$ in constantly many rounds:

- ask Q and R; if both $Q \mapsto 1$ and $R \mapsto 0$ we are done; else,
- recalling that $Q \supset R := \neg Q \vee R$, if $R \mapsto 1$ we have a simple contradiction against $Q \supset R \mapsto 0$; else,
- we have $Q \mapsto 0$, so ask $\neg Q$;
  - if $\neg Q \mapsto 1$ we have a simple contradiction against $Q \supset R \mapsto 0$; else,
  - if $\neg Q \mapsto 0$ we have a simple contradiction against $Q \mapsto 0$.

The above two examples have corresponding strategies given in Figure 6.5:

A more interesting and useful example is:

Proposition 6.5.2. *Let Γ be a list of queries of length n. From $\{\bigwedge \Gamma \mapsto b\}$ we can:*

- *($b = 0$) find $A \in \Gamma$ such that $A \mapsto 0$ in $O(\log n)$ rounds.*

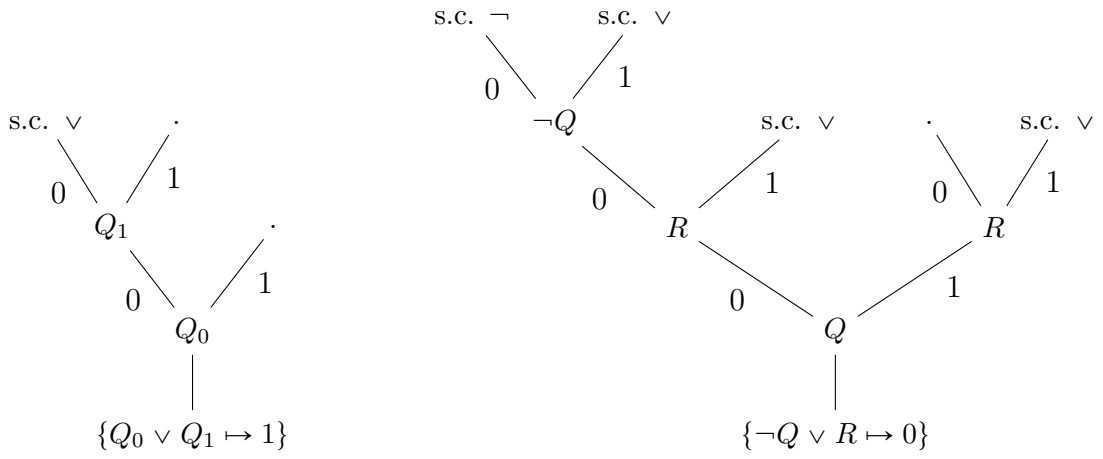


Figure 6.5: Simple strategies. Leaves labelled by " . " indicate we successfully forced a desirable answer by the Adversary on the respective branch.

- *($b = 1$) force $A \mapsto 1$ for any $A \in \Gamma$ in $O(\log n)$ rounds.*

Dually from $\{\bigvee \Gamma \mapsto b\}$, we can:

- *($b = 1$) find $A \in \Gamma$ such that $A \mapsto 1$ in $O(\log n)$ rounds.*
- *($b = 0$) force $A \mapsto 0$ for any $A \in \Gamma$ in $O(\log n)$ rounds.*

Proof. We will only prove the first two bullet points. The proof proceeds by divide-and-conquer induction on n. Note that when $n = 0, 1$ we are done in a constant number of steps. Otherwise let Γ_0 and Γ_1 be the first and second halves of Γ, respectively.

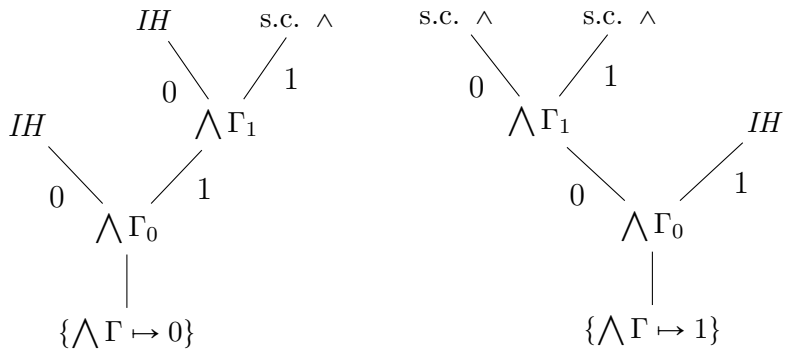
For the case $b = 0$:

- ask $\bigwedge \Gamma_0$ and $\bigwedge \Gamma_1$;
- if $\bigwedge \Gamma_0 \mapsto 0$ find $A \in \Gamma_0$ with $A \mapsto 0$, by inductive hypothesis; else,
- if $\bigwedge \Gamma_1 \mapsto 0$ find $A \in \Gamma_1$ with $A \mapsto 0$, by inductive hypothesis; else,
- we have a simple contradiction $\{\bigwedge \Gamma_0 \mapsto 1, \bigwedge \Gamma_1 \mapsto 1, \bigwedge \Gamma \mapsto 0\}$.

For the case $b = 1$ and w.l.o.g. let $A \in \Gamma_0$:

- ask $\bigwedge \Gamma_0$;
- if $\bigwedge \Gamma_0 \mapsto 1$ force $A \mapsto 1$ by inductive hypothesis; else,
- ask $\bigwedge \Gamma_1$; any response b_1 yields a simple contradiction including $\{\bigwedge \Gamma_0 \mapsto 0, \bigwedge \Gamma_1 \mapsto b_1 \bigwedge \Gamma \mapsto 1\}$.

The respective strategies are the following where *IH* indicates 'Induction Hypothesis':



□

Querying inferences

Before polynomially simulating eLNDT, let us first state a relatively simple consequence of the earlier forcing and finding strategies, similar to what appears in [52]:

Lemma 6.5.3 (Local soundness)**.** *Fix an inference step:*

$$\mathsf{r}\,\frac{\Gamma_1 \longrightarrow \Delta_1 \quad \Gamma_2 \longrightarrow \Delta_2}{\Gamma \longrightarrow \Delta}$$

where all $\Gamma, \Delta, \Gamma_i, \Delta_i$ *have length* $\leqslant n$*. Consider the following notation:*

- $L := \bigwedge \Gamma$ *and* $L_i := \bigwedge \Gamma_i$*;*
- $R := \bigvee \Delta$ *and* $R_i := \bigvee \Delta_i$*;*
- $Q := L \supset R$ *and* $Q_i := L_i \supset R_i$*;*

for $i = 1, 2$*.*

From $\{Q \mapsto 0, Q_1 \mapsto 1, Q_2 \mapsto 1\}$ *there is a strategy winning in* $O(\log n)$ *rounds.*

Proof sketch. The proof is relatively simple and proceeds by case analysis on r. We present some representative cases, the rest follow accordingly.

From $Q \mapsto 0$ force $L \mapsto 1$ and $R \mapsto 0$ in (three) constant number of steps just like in Figure 6.5.

Suppose r is an initial rule i.e. a 0-step a 1-step or an identity step. The premises of this rule are empty and either a simple contradiction is immediate or we obtain one by similarity from Definition 6.4.1. In the case it is an extension step, it will have only one premise: a hypothesis from $\mathcal{E}$. Then, we will again obtain a simple contradiction by similarity from Definition 6.4.1.

Otherwise, if r has two premises, for $i = 1, 2$ find either $A_i \in \Gamma_i$ for which we can force $A_i \mapsto 0$ or find $B_i \in \Delta_i$ with $B_i \mapsto 1$ in $O(\log n)$ rounds. If any such A_i occurs in Γ or B_i occurs in Δ, then we can obtain a simple contradiction by forcing $A_i \mapsto 1$ or

$B_i \mapsto 0$ respectively since we have forced $L \mapsto 1, R \mapsto 0$. Else, we either found $A_i \mapsto 0$ and $A_i \notin \Gamma$ or $B_i \mapsto 1$ and $B_i \notin \Delta$. Let us check the case for decision left:

$$p\text{-}l\,\frac{\Gamma, A \longrightarrow \Delta, p \quad \Gamma, p, B \longrightarrow \Delta}{\Gamma, ApB \longrightarrow \Delta}$$

The scenarios to consider are for $A_1 = A$ or $B_1 = p$ and $A_2 \in \{p, B\}$. If we forced $B_1 \mapsto 1$ and $A_2 \mapsto 0$ for $A_2 = p$, we obtain a simple contradiction by similarity. Else, for the three following cases: $B_1 \mapsto 1$ and $A_2 \mapsto 0$ for $A_2 = B$, or $A_1 \mapsto 0$ and $A_2 \mapsto 0$ for $A_2 = B$, or $A_1 \mapsto 0$ and $A_2 \mapsto 0$ for $A_2 = p$ we obtain a simple contradiction for decision since we can force $ApB \mapsto 1$.

□

Referring to Proposition 6.5.2, let us note that the proof of the first bullet point i.e. the 'finding' strategy, actually establishes more than stated;

Proposition 6.5.4. *From* $\left\{\bigwedge_{i<k} A_i \mapsto 0\right\}$ *we can find the* least A_i *with* $A_i \mapsto 0$ *in* $O(\log k)$ *rounds, i.e. such that, for any* $j < i$*, we can force* $A_j \mapsto 1$ *after an extra* $O(\log k)$ *rounds.*

We are now ready to prove the main result of this subsection.

Proof of Theorem 6.5.1. Fix a (dag-like) $\mathsf{eL(N)DT}$-proof $\pi : (\Gamma_i \longrightarrow \Delta_i)_{i<n}$. For $i < n$ we write:

- $L_i := \bigwedge \Gamma_i$;
- $R_i := \bigwedge \Delta_i$;
- $Q_i := L_i \supset R_i$
- $Q^i := \bigwedge_{j<i} Q_i$

We first construct a winning strategy from $\{Q^{n-1} \mapsto 0\}$ as follows:

- find least Q_i such that $Q_i \mapsto 0$ by Proposition 6.5.4;
- if $\Gamma_i \longrightarrow \Delta_i$ is the conclusion of an inference step with premisses (among) $\Gamma_j \longrightarrow \Delta_j$ and $\Gamma_k \longrightarrow \Delta_k$, then we have $j, k < i$ so we can force $Q_j \mapsto 1$ and $Q_k \mapsto 1$ by leastness;
- we can now reach a simple contradiction in logarithmically many rounds by Lemma 6.5.3.

Now, from $\{A \mapsto 1 : A \in \Gamma_{n-1}\} \cup \{B \mapsto 0 : B \in \Delta_{n-1}\}$ we have a winning strategy:

- ask Q^{n-1}; if $\{Q^{n-1} \mapsto 0\}$ proceed as above; else,
- $Q^{n-1} \mapsto 1$ so force $Q_{n-1} \mapsto 1$ and find either some $A \in \Gamma_{n-1}$ with $A \mapsto 0$ or $B \in \Delta_{n-1}$ with $B \mapsto 1$; either way we have a simple contradiction against the initial state. □

6.6 Non-uniform version of Immerman-Szelepcsényi

We aim to provide a translation from DB, NB strategies to $\mathsf{eL(N)DT}$ proofs for which we require simulation of negation of (N)BPs. This, while simple for the deterministic case (flipping values of leaves of BPs), it is much more involved for NBPs. In this section we will define and develop the necessary tools for our setting i.e. the system eLNDT, to present a non-uniform (partial) formalisation of the Immerman-Szelepcsényi theorem, $co\mathbf{NL} = \mathbf{NL}$, that suffices for simulating non-deterministic negation of eNDT formulas. There are two main differences in our result: first, the inductive counting within the original proof [37, 58] is devolved to the level of eLNDT proofs simplifying the overall argument. Second, our formalization of the Immerman-Szelepcsényi theorem is partial in the sense that we present explicit constructions that (only) simulate the negation of a non-deterministic branching program under the condition that *exactly* k formulas of a given list are true. Hence, given an eNDT formula, we do not provide a single eNDT formula computing its negation.

6.6.1 Working with positive decisions

It will be convenient in our construction to 'stitch' together branching programs using positive decisions i.e. decisions of the form $Ap(A \vee B)$, introduced in Section 2.2.3. We recall that this imposed restriction on decisions induces what we called positive (non-deterministic) branching programs (PBPs): NBPs where, for every 0-edge, there is a parallel 1-edge.

In what follows, we shall 'bootstrap' our language with the capacity to make (certain) positive decisions on *complex* formulas (over a fixed set of extension variables), and even recursively so. Until now however, the syntax of our systems only permits decisions on literals and more specifically the grammars for m-$\mathsf{eLNDT}, 0/1$-eLNDT only allow forming decisions on non-negated literals. In any case, let us informally assume that for eNDT formulas A, B, C we could express both ABC and $AB(A \vee C)$ as eNDT formulas over some appropriate set of extension axioms. A good way to visualise ABC is to consider the NBP consisted of A, B, C such that all 0-sinks of B are connected to the root of A and all 1-sinks of B connect to the root of C. Similarly, one could visualise $AB(A \vee C)$ as the NBP where all the 0-sinks of B connect to the root of A and each 1-sink of B is copied with one copy connecting to the root of A and the other to the root of C. Note that $AB(A \vee C)$ it would compute exactly 'if B then $A \vee C$ else A'. On the contrary, due to non-determinism general decisions are not as 'well behaved' i.e. if we could express ABC as an eNDT formula, it would not correctly compute 'if B then C else A'. As a counterexample consider the case where for some assignment of variables a path in B evaluates to 1 but there is also a path to 0, C evaluates to 0 and A evaluates to 1. It follows that ABC evaluates (wrongly) to 1. To avoid situations like the above we will restrict ourselves to a semi-positive setting in this section: in what follows we are in eLNDT but all decisions we define on complex formulas will be positive.

More formally:

Definition 6.6.1 (Complex positive decisions)**.** Let $\mathcal{E} = \{e_i \leftrightarrow E_i\}_{i<n}$ be a set of extension axioms. For a finite list of formulas $\mathbf{B}$ over $\mathcal{E}$ and any $B \in \mathbf{B}$, we inductively

generate extension variables $AB(A \vee C)$ for each formula A and C.[7] We define $\mathcal{E}^+_{\mathbf{B}}$ to extend $\mathcal{E}$ by all extension axioms of the form:

$$\begin{array}{l} A0(A \vee C) \leftrightarrow A \\ A1(A \vee C) \leftrightarrow A \vee C \\ Ae_i(A \vee C) \leftrightarrow AE_i(A \vee C) \end{array} \qquad \begin{array}{r} A(B_0 \vee B_1)(A \vee C) \leftrightarrow (AB_0(A \vee C)) \vee (AB_1(A \vee C)) \\ A(B_0 p B_1)(A \vee C) \leftrightarrow (AB_0(A \vee C)) p (AB_1(A \vee C)) \end{array}$$

The notation we have used is intentionally suggestive under interpretation by $\mathcal{E}^+_{\mathbf{B}}$, perhaps at the danger of ambiguity: we insist, for foundational reasons, that $AB(A \vee C)$ are *not* complex formulas, but rather bona fide extension variables.

A result for the appropriate 'truth conditions' may be established by $\mathcal{E}$-induction:

Lemma 6.6.2 (Truth for complex positive decisions)**.** *Fix a set of extension axioms $\mathcal{E}$ and formula B over only those extension variables. There are polynomial size proofs over $\mathcal{E}^+_{\mathbf{B}}$ of:*

$$\begin{array}{rcl} AB(A \vee C) & \longrightarrow & A, B \\ AB(A \vee C) & \longrightarrow & A, C \end{array} \qquad \begin{array}{rcl} A & \longrightarrow & AB(A \vee C) \\ B, C & \longrightarrow & AB(A \vee C) \end{array}$$

Proof. The argument is standard $\mathcal{E}$-induction on B. We will only present proofs for $B, C \longrightarrow AB(A \vee C)$ as the other sequents follow in a similar fashion.

Base cases for $B = 0, B = 1$ are a 0-step and an identity step with weakening steps:

$$\mathsf{w}\text{-}l,\mathsf{w}\text{-}r\ \frac{0 \longrightarrow}{0, C \longrightarrow B} \qquad \mathsf{w}\text{-}l,\mathsf{w}\text{-}r\ \frac{C \longrightarrow C}{1, C \longrightarrow A, C}$$

Induction step: The case where $B = D \vee E$ with D, E eNDT formulas over $\mathcal{E}$ is as follows :

$$\mathcal{E},\vee\text{-}r\ \frac{\vee\text{-}l\ \dfrac{\dfrac{IH}{D, C \longrightarrow AD(A \vee C)} \quad \dfrac{IH}{E, C \longrightarrow AE(A \vee C)}}{D \vee E, C \longrightarrow AD(A \vee C), AE(A \vee C)}}{D \vee E, C \longrightarrow A(D \vee E)(A \vee C)}$$

The case where $B = DpE$ with D, E eNDT formulas over $\mathcal{E}^+_{\mathbf{B}}$ is as follows:

$$p\text{-}l\ \frac{\mathcal{E}^+,p\text{-}r\ \dfrac{\mathsf{w}\text{-}r\ \dfrac{IH}{C, D \longrightarrow p, AD(A \vee C)}\ w \quad \mathsf{w}\text{-}l,\mathsf{w}\text{-}r\ \dfrac{p \longrightarrow p}{C, D, p \longrightarrow p, AE(A \vee C)}}{C, D \longrightarrow p, A(DpE)(A \vee C)} \quad \mathcal{E}^+,p\text{-}r\ \dfrac{\mathsf{w}\text{-}l,\mathsf{w}\text{-}r\ \dfrac{p \longrightarrow p}{C, p, E \longrightarrow p, AD(A \vee C)} \quad \mathsf{w}\text{-}l\ \dfrac{IH}{C, p, E \longrightarrow AE(A \vee C)}}{C, p, E \longrightarrow A(DpE)(A \vee C)}}{DpE, C \longrightarrow A(DpE)(A \vee C)}$$

In the case that B is an extension variable e_i such that $e_i \leftrightarrow E_i$ is an extension axiom, we call the inductive hypothesis for $AE_i(A \vee C)$. □

Remark 6.6.3. We may instantiate the above result to define connectives/formulas we are interested in but are not natively available to our setting. Take conjunction $\wedge$

[7] This is a closure property: we also have extension variables of the form $AB(A \vee (CB'(C \vee D)))$ and so on. Note that both B and B' must be over the original set of extension axioms $\{e_i\}_{i<n}$ to ensure well-foundedness.

for instance, writing $A \wedge B := 0A(0 \vee B)$ and using Lemma 6.6.2, it is simple to obtain polynomial-size proofs of the expected 'truth conditions' $A \wedge B \longrightarrow A$, $A \wedge B \longrightarrow B$ and $A, B \longrightarrow A \wedge B$.

Another interesting example that we shall use later is:

Definition 6.6.4 (Thresholds)**.** Fix a list of eLNDT formulas $\mathbf{B} = B_0, \ldots, B_{N-1}$ over a set of extension axioms $\mathcal{B}$. After generating $\mathcal{B}^+_{\mathbf{B}}$ as in Definition 6.6.1 above, we may introduce further new extension variables $t_k^{\mathbf{B}^n_s}$ for each segment $\mathbf{B}^n_s = B_s, \ldots, B_n$ of $\mathbf{B}$ for $s \leqslant n \leqslant N$ and $k \leqslant N$ and write $\mathcal{T}^{\mathbf{B}}$ for the extension of $\mathcal{B}^+_{\mathbf{B}}$ by all extension axioms of the form:

$$\begin{array}{rcl} t_0^{\mathbf{B}^n_s} & \leftrightarrow & 1 \\ t_{k+1}^{\epsilon} & \leftrightarrow & 0 \\ t_{k+1}^{\mathbf{B}^n_s} & \leftrightarrow & t_{k+1}^{\mathbf{B}^n_{s+1}} B_s(t_{k+1}^{\mathbf{B}^n_{s+1}} \vee t_k^{\mathbf{B}^n_{s+1}}) \quad (\text{for } s < N) \end{array}$$

It is not hard to see, by induction on k that, over $\mathcal{T}^{\mathbf{B}}$, $t_k^{\mathbf{B}^n_s}$ is true just when at least k of $\mathbf{B}^n_s$ are true. In fact we showed in Section 3.1.2, that this is the case when $\mathbf{B}^n_s$ is a list of variables in eLNDT : there are polynomial-size proofs of the basic truth conditions of threshold functions, in particular: $\longrightarrow t_0^{\mathbf{B}^n_s}$ and $t_{n-s+1}^{\mathbf{B}^n_s} \longrightarrow$.

Remark 6.6.5 (Differences with previous definition.)**.** Definition 6.6.4 is slightly different to Definition 3.1.4: before k was any integer while now $0 \leqslant k \leqslant N$ and the extension axioms had slightly different form. These changes are mostly for convenience and it is easy to show (via polynomial size eLNDT proofs) that the two choices are equivalent. Hence, eLNDT proofs of basic properties will proceed in the same fashion as before.

We restate some results of Section 3.1.2 and Chapter 4 for our current setting:

Proposition 6.6.6 ($t_k^{\mathbf{B}^n_s}$ is decreasing in k)**.** *There are polynomial-size* eLNDT^+ *proofs of the following sequents over extension axioms* $\mathcal{T}^{\mathbf{B}}$*:*

1. $\longrightarrow t_0^{\mathbf{B}^n_s}$
2. $t_{k+1}^{\mathbf{B}^n_s} \longrightarrow t_k^{\mathbf{B}^n_s}$
3. $t_k^{\mathbf{B}^n_s} \longrightarrow$ *whenever* $k > |\mathbf{B}^n_s|$

We want to be able to manipulate threshold arguments much like Lemma 4.3.2. Due to our setting being slightly different we will consider 'neighboring' segments $\mathbf{B_0}, \mathbf{B_1}$ of the list of formulas $\mathbf{B}$ (i.e. the concatenation $\mathbf{B_0B_1}$ is a segment of $\mathbf{B}$).

Lemma 6.6.7 (Merging and splitting)**.** *For a segment* $\mathbf{B}_0\mathbf{B}_1$ *of the list* $\mathbf{B}$ *there are polynomial-size* eLNDT^+ *proofs, over extension axioms* $\mathcal{T}^{\mathbf{B}}$*, of the following sequents:*

1. $t_k^{\mathbf{B}_0}, t_l^{\mathbf{B}_1} \longrightarrow t_{k+l}^{\mathbf{B}_0\mathbf{B}_1}$ *(merging)*
2. $t_{k+l}^{\mathbf{B}_0\mathbf{B}_1} \longrightarrow t_{k+1}^{\mathbf{B}_0}, t_l^{\mathbf{B}_1}$ *(splitting)*

Lemma 6.6.8. *For any element* B_i *of the list* $\mathbf{B}$*, there are polynomial size* eLNDT *proofs over extension axioms* $\mathcal{T}^{\mathbf{B}}$ *of the following sequents:*

$$B_i \leftrightarrow t_1^{B_i}$$

The proofs of the above results are very similar to the ones from Section 3.1.2 and Chapter 4 but always w.r.t. the new set of extension axioms $\mathcal{T}^{\mathbf{B}}$. From now on we will mostly work with suffixes $\mathbf{B}_s = B_s, \ldots, B_{N-1}$ of the list $\mathbf{B}$ for $s \leqslant N$.

6.6.2 Decider Construction

In what follows, we construct the 'k-decider', an NBP utilizing complex positive decisions. Under the condition that k-many of the elements of $\mathbf{B}$ are true, it will compute a complex general decision on eNDT formulas over an appropriate set of extension axioms. For the rest of this section, we fix formulas $\mathbf{B} = B_0, \ldots, B_{N-1}$ over a set of extension axioms $\mathcal{B}$. As before, for $1 \leqslant s \leqslant N-1$, we write $\mathbf{B}_s$ for the list $B_s, \ldots, B_{N-1}$. In our definitions and statements, we feature the parameter $k \leqslant N$ which intuitively serves as a meta-level 'counter' of the number of formulas among $\mathbf{B}$ that are true. Recalling the threshold programs from Definition 6.6.4 over $\mathcal{T}^{\mathbf{B}}$, the main result of this section is:

Theorem 6.6.9 (Immerman-Szelepcsényi, formalised)**.** *For all $k \leqslant N$ and formulas A, C there are extension variables $AB_i^k C$ and an appropriate set of extension axioms extending $\mathcal{T}^{\mathbf{B}}$ such that the following sequents have polynomial-size* eLNDT *proofs:*

$$t_k^{\mathbf{B}}, AB_i^k C \longrightarrow A, B_i, t_{k+1}^{\mathbf{B}} \qquad t_k^{\mathbf{B}}, B_i, C \longrightarrow AB_i^k C, t_{k+1}^{\mathbf{B}}$$
$$t_k^{\mathbf{B}}, AB_i^k C, B_i \longrightarrow C, t_{k+1}^{\mathbf{B}} \qquad t_k^{\mathbf{B}}, A \longrightarrow B_i, AB_i^k C, t_{k+1}^{\mathbf{B}}$$

The idea behind the sequents above is that, under the condition that *exactly* k of the formulas in $\mathbf{B}$ are true, the 'decider' $AB_i^k C$ correctly computes a decision on B_i, returning the value of A if B_i is false, and C if true. The appropriate set of extension axioms extending $\mathcal{T}^{\mathbf{B}}$ will be explicitly constructed in the proof of Proposition 6.6.14 from which Theorem 6.6.9 follows as a special case. It is worth noting that expressing the condition 'exactly k- many of the elements in $\mathbf{B}$ are true' could theoretically be done by having fresh extension variables computing the exact-k function w.r.t. the list $\mathbf{B}$ on the LHS, akin to Definition 3.1.1). This however, was done by using general decisions which is not available to us since we want our k-decider to simulate general decisions on complex formulas instead of decision literals. Instead, we opt for using the extension variables $t_k^{\mathbf{B}}, t_{k+1}^{\mathbf{B}}$ computing threshold functions for k. Note that $t_{k+1}^{\mathbf{B}}$ on the RHS is semantically equivalent to having its negation on the LHS, and of course $t_k^{\mathbf{B}} \wedge \neg t_{k+1}^{\mathbf{B}}$ is true just when *exactly* k of $\mathbf{B}$ are true.

Our choice of notation $AB_i^k C$ is again suggestive, and we still insist, for foundational reasons, that these expressions are *not* complex formulas, but rather fresh extension variables. We call $AB_i^k C$ the **k-decider** of B_i (for A, C). It is visualised (over its corresponding extension axioms) in Figure 6.6 as an NBP, in fact as a positive combination of the programs $\mathbf{B}$ we started with and some leaves substituted by A, C accordingly.

Let us now define the k-decider formally and provide some results that will be required later.

Definition 6.6.10 (Decision programs)**.** We introduce extension variables $d_{m,l,b}^{i,\mathbf{B}_s,k}$ for $m \geqslant s, i \geqslant s, N > m$, $l \leqslant m - s$ and $b \in \{0,1\}$ and then construct $\mathcal{B}^+$ as in Definition 6.6.1 and $\mathcal{T}^{\mathbf{B}}$ as in Definition 6.6.4. From here we set $\mathcal{D}^{\mathbf{B}}$ to extend $\mathcal{T}^{\mathbf{B}}$ by all axioms:

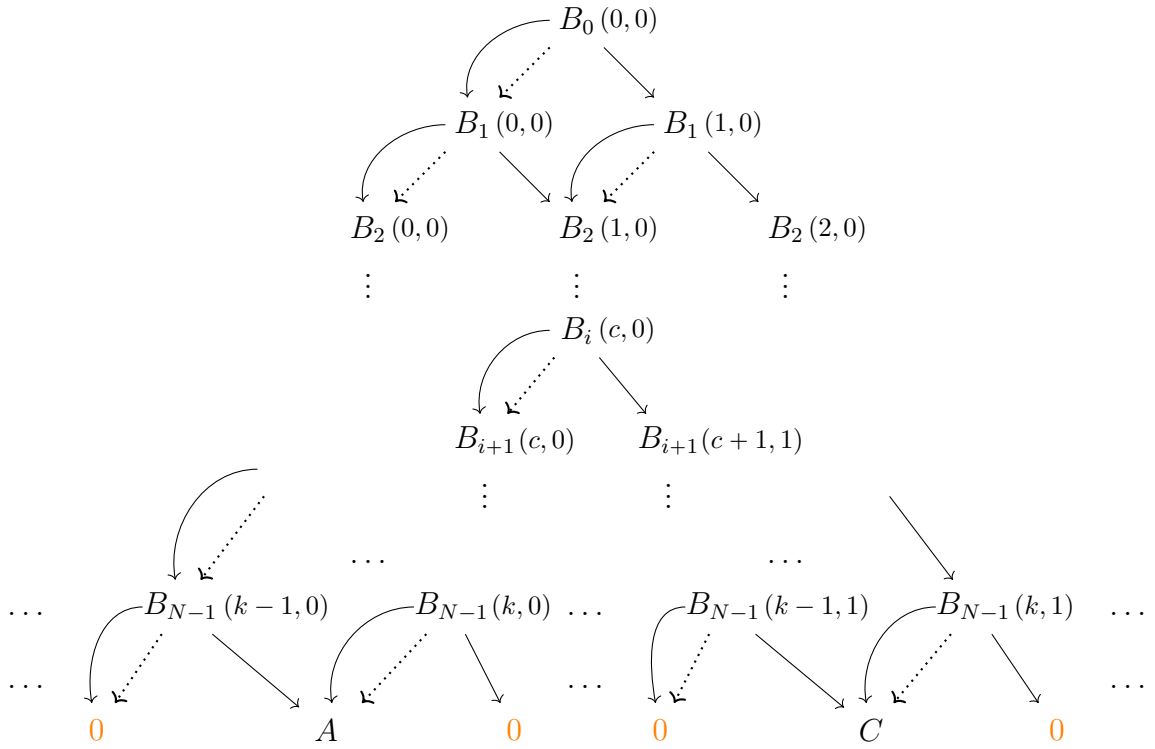


Figure 6.6: The k-decider AB_i^kC, visualised as an NBP. 0-edges are dotted and 1-edges are solid.

$$
\begin{aligned}
d^{i,\mathbf{B}_s,k}_{N-1,l,0} &\leftrightarrow 0B_{N-1}(0 \vee 0) && \text{for } l \in \{0,\dots,k-2,k+1,\dots,N-3\}\\
d^{i,\mathbf{B}_s,k}_{N-1,l,1} &\leftrightarrow 0B_{N-1}(0 \vee 0) && \text{for } l \in \{1,\dots,k-2,k+1,\dots,N-2\}\\
d^{i,\mathbf{B}_s,k}_{N-1,k,0} &\leftrightarrow AB_{N-1}(A \vee 0)\\
d^{i,\mathbf{B}_s,k}_{N-1,k,1} &\leftrightarrow CB_{N-1}(C \vee 0)\\
d^{i,\mathbf{B}_s,k}_{N-1,k-1,0} &\leftrightarrow 0B_{N-1}(0 \vee A)\\
d^{i,\mathbf{B}_s,k}_{N-1,k-1,1} &\leftrightarrow 0B_{N-1}(0 \vee C)\\
d^{i,\mathbf{B}_s,k}_{m,l,0} &\leftrightarrow d^{i,\mathbf{B}_s,k}_{m+1,l,0}B_m(d^{i,\mathbf{B}_s,k}_{m+1,l,0} \vee d^{i,\mathbf{B}_s,k}_{m+1,l+1,0}) && \text{for } m \neq i\\
d^{i,\mathbf{B}_s,k}_{m,l,1} &\leftrightarrow d^{i,\mathbf{B}_s,k}_{m+1,l,1}B_m(d^{i,\mathbf{B}_s,k}_{m+1,l,1} \vee d^{i,\mathbf{B}_s,k}_{m+1,l+1,1}) && \text{for } 1 \leqslant l, m > i\\
d^{i,\mathbf{B}_s,k}_{i,l,0} &\leftrightarrow d^{i,\mathbf{B}_s,k}_{i+1,l,0}B_i(d^{i,\mathbf{B}_s,k}_{i+1,l,0} \vee d^{i,\mathbf{B}_s,k}_{i+1,l+1,1})
\end{aligned}
$$

From here we set $AB_i^0C := A$ and, for $0 < k \leqslant N$, the k-decider w.r.t. the list $\mathbf{B}$ is $AB_i^kC := d^{i,\mathbf{B},k}_{0,0,0}$.

Remark 6.6.11. Looking back to Figure 6.6, note that each 'node' is actually a copy of an NBP $B_m \in \mathbf{B}$, indexed by a pair (l,b). Each variable $d^{i,\mathbf{B}_s,k}_{m,l,b}$ computes the subprogram rooted at the node $B_m(l,b)$. Intuitively, the indices of an extension variable $d^{i,\mathbf{B}_s,k}_{m,l,b}$ have the following purpose: i indicates the B_i we wish our construction to 'decide' upon, $\mathbf{B}_s$ is the list of the formulas involved in our construction, k denotes the exact number of formulas $B_j \in \mathbf{B}_s$ true in the environment, m is the index of the formula labelling the current node, l keeps count of the number of 'right' edges we have traversed at the level of the graph (number of true B_js along the path thusfar), while b is a Boolean flag that flips from 0 to 1 if B_i is true along the path. In paths where exactly k of the $\mathbf{B}$ are true,

this program correctly 'decides' B_i, calling A if it is false and C if it is true. Otherwise the program may return erroneously, indicated by the orange 0 leaves.

In order to prove Theorem 6.6.9, we first need some intermediate results. The first gives us relationships between nodes with differing values of k and l while fixing all other parameters.

Lemma 6.6.12 (Monotonicity)**.** *There are polynomial size* eLNDT *proofs over* $\mathcal{D}^{\mathbf{B}}$ *of*

$$\begin{array}{lcll} d^{i,\mathbf{B}_s,k}_{m,l,b} & \leftrightarrow & d^{i,\mathbf{B}_s,k-1}_{m,l-1,b} & \\ d^{i,\mathbf{B}_s,k}_{m,l,b} & \longrightarrow & d^{i,\mathbf{B}_s,k}_{m,l+1,b} & \text{for } l \neq k \end{array}$$

Proof. Here the first line is a consequence of our simulation result, Proposition 6.2.8 since it is easy to check that $d^{i,\mathbf{B}_s,k}_{m,l,b} \gtrsim_{\mathcal{D}^{\mathbf{B}}} d^{i,\mathbf{B}_s,k-1}_{m,l-1,b}$ and $d^{i,\mathbf{B}_s,k-1}_{m,l-1,b} \gtrsim_{\mathcal{D}^{\mathbf{B}}} d^{i,\mathbf{B}_s,k}_{m,l,b}$.
The sequent $d^{i,\mathbf{B}_s,k}_{m,l,b} \longrightarrow d^{i,\mathbf{B}_s,k}_{m,l+1,b}$ is proven by induction on $N-1-m$ and examining the case for $l = k-1$ separately.

Base case: $m = N-1$.

By definition for $l \notin \{k-1, k\}$ the following is an extension axiom of $\mathcal{D}^{\mathbf{B}}$:

$$d^{i,\mathbf{B}_s,k}_{N-1,l,b} \leftrightarrow 0B_{N-1}(0 \vee 0)$$

So for $l \notin \{k-1, k\}$ after a constant number of decision steps and cuts, one obtains a polynomial size proof for

$$d^{i,\mathbf{B}_s,k}_{N-1,l,b} \longrightarrow d^{i,\mathbf{B}_s,k}_{N-1,l+1,b}$$

For $l = k-1$ by definition the following are extension axioms of $\mathcal{D}^{\mathbf{B}}$:

$$\begin{array}{lcl} d^{i,\mathbf{B}_s,k}_{N-1,k-1,0} & \leftrightarrow & 0B_{N-1}(0 \vee A) \\ d^{i,\mathbf{B}_s,k}_{N-1,k,0} & \leftrightarrow & AB_{N-1}(A \vee 0) \\ d^{i,\mathbf{B}_s,k}_{N-1,k-1,1} & \leftrightarrow & 0B_{N-1}(0 \vee C) \\ d^{i,\mathbf{B}_s,k}_{N-1,k,1} & \leftrightarrow & CB_{N-1}(C \vee 0) \end{array}$$

Applying constant many decision and cut steps we obtain polynomial size proofs for

$$d^{i,\mathbf{B}_s,k}_{N-1,k-1,0} \longrightarrow d^{i,\mathbf{B}_s,k}_{N-1,k,0} \quad \text{and} \quad d^{i,\mathbf{B}_s,k}_{N-1,k-1,1} \longrightarrow d^{i,\mathbf{B}_s,k}_{N-1,k,1}$$

Induction step: The following are axioms of $\mathcal{D}^{\mathbf{B}}$:

$$\begin{array}{lcll} d^{i,\mathbf{B}_s,k}_{m,l,0} & \leftrightarrow & d^{i,\mathbf{B}_s,k}_{m+1,l,0}B_m(d^{i,\mathbf{B}_s,k}_{m+1,l,0} \vee d^{i,\mathbf{B}_s,k}_{m+1,l+1,0}) & \text{for } m \neq i \\ d^{i,\mathbf{B}_s,k}_{m,l,1} & \leftrightarrow & d^{i,\mathbf{B}_s,k}_{m+1,l,1}B_m(d^{i,\mathbf{B}_s,k}_{m+1,l,1} \vee d^{i,\mathbf{B}_s,k}_{m+1,l+1,1}) & \text{for } 1 \leqslant l, m > i \\ d^{i,\mathbf{B}_s,k}_{i,l,0} & \leftrightarrow & d^{i,\mathbf{B}_s,k}_{i+1,l,0}B_i(d^{i,\mathbf{B}_s,k}_{i+1,l,0} \vee d^{i,\mathbf{B}_s,k}_{i+1,l+1,1}) & \end{array}$$

The sequent $d^{i,\mathbf{B}_s,k}_{m,l,b} \longrightarrow d^{i,\mathbf{B}_s,k}_{m,l+1,b}$ follows after a constant number of extension steps (for each respective case from above), decision steps, disjunction steps and cuts before ending the proof branch by invoking induction hypothesis or general identity. □

The next result gives us relationships between nodes with different values of s in terms of k and l (in a way adding extra elements in the list), fixing all other parameters:

Lemma 6.6.13. *For $i \geqslant s$, there are polynomial size* eLNDT *proofs over* $\mathcal{D}^{\mathbf{B}}$ *of:*

$$\begin{array}{lcl} d^{i,\mathbf{B}_{s-1},k}_{m,l,b} & \leftrightarrow & d^{i,\mathbf{B}_s,k}_{m,l,b} \\ d^{i,\mathbf{B}_{s-1},k}_{m,l,b} & \leftrightarrow & d^{i,\mathbf{B}_s,k-1}_{m,l-1,b} \end{array}$$

Proof sketch. Here the first line again follows by appealing to our simulation result, Proposition 6.2.8 since $d^{i,\mathbf{B}_{s-1},k}_{m,l,b} \gtrsim_{\mathcal{D}^{\mathbf{B}}} d^{i,\mathbf{B}_s,k}_{m,l,b}$ and $d^{i,\mathbf{B}_s,k}_{m,l,b} \gtrsim_{\mathcal{D}^{\mathbf{B}}} d^{i,\mathbf{B}_{s-1},k}_{m,l,b}$. The second follows from the first line along with the first line of Lemma 6.6.12 above. □

Now we obtain the following 'truth conditions', by induction on the length of $\mathbf{B}_s$ and subinduction on k:

Proposition 6.6.14. *The following sequents have polynomial size* eLNDT *proofs over an extension axiom set extending* $\mathcal{D}^{\mathbf{B}}$*:*

$$\begin{array}{rcl@{\qquad}rcl} t^{\mathbf{B}_s}_k, d^{i,\mathbf{B}_s,k}_{s,0,0} & \longrightarrow & B_i, A, t^{\mathbf{B}_s}_{k+1} & t^{\mathbf{B}_s}_k, B_i, C & \longrightarrow & d^{i,\mathbf{B}_s,k}_{s,0,0}, t^{\mathbf{B}_s}_{k+1} \\ t^{\mathbf{B}_s}_k, d^{i,\mathbf{B}_s,k}_{s,0,0}, B_i & \longrightarrow & C, t^{\mathbf{B}_s}_{k+1} & t^{\mathbf{B}_s}_k, A & \longrightarrow & B_i, d^{i,\mathbf{B}_s,k}_{s,0,0}, t^{\mathbf{B}_s}_{k+1} \end{array}$$

Proof. We will perform induction on the length of the list $\mathbf{B}_s$ (i.e. $N-1-s$) and subinduction on k. We present a proof of the sequent $t^{\mathbf{B}_s}_k, B_i, C \longrightarrow d^{i,\mathbf{B}_s,k}_{s,0,0}, t^{\mathbf{B}_s}_{k+1}$, the rest follow similarly.

Base case for $\mathbf{B}_{N-1} = \{B_{N-1}\}$: For $k > 1$ there are polynomial size proofs of $t^{B_{N-1}}_k \leftrightarrow 0$ by applying extension steps and positive decisions on the axioms given in Definition 6.6.4. Hence, after a cut step, the sequent: $t^{B_{N-1}}_k, B_{N-1}, C \longrightarrow d^{s,B_{N-1},k}_{s,0,0}, t^{B_{N-1}}_{k+1}$ follows by a 0-step.
Else for $k = 0$:

1. $1, B_{N-1}, C \longrightarrow AB_{N-1}(A \vee 0), B_{N-1}$ (id on B_{N-1} and w-steps)
2. $t^{B_{N-1}}_0, B_{N-1}, C \longrightarrow d^{s,B_{N-1},0}_{s,0,0}, t^{B_{N-1}}_1$ ($\leftrightarrow$ steps over $\mathcal{D}^{\mathbf{B}}$ on 1. and Lemma 6.6.8)

Else for $k = 1$:

1. $B_{N-1}, C \longrightarrow B_{N-1}, 0$ (id on B_{N-1})
2. $B_{N-1}, C \longrightarrow 0, C$ (id on C)
3. $B_{N-1}, B_{N-1}, C \longrightarrow 0B_{N-1}(0 \vee C), 0$ (posdec-r on 1.+2.)
4. $t^{B_{N-1}}_1, B_{N-1}, C \longrightarrow d^{s,B_{N-1},1}_{s,0,0}, t^{B_{N-1}}_2$ ($\leftrightarrow$ steps over $\mathcal{D}^{\mathbf{B}}$ on 3., Lemmas 6.6.6, 6.6.8)

Induction step: Assume by induction hypothesis we have short proofs of the sequents $t^{\mathbf{B}_s}_k, B_i, C \longrightarrow d^{i,\mathbf{B}_s,k}_{s,0,0}, t^{\mathbf{B}_s}_{k+1}$ for all $k \geqslant 0$. For $\mathbf{B}_{s-1} = \{B_{s-1}, B_s, \ldots, B_{N-1}\}$ we let i first range over $\{s, \ldots, N-1\}$, the case for $i = s-1$ is later covered individually. Since in the induction step for $\mathbf{B}_{s-1}, k$ we will employ induction hypothesis for both $\mathbf{B}_s, k$ and $\mathbf{B}_s, k-1$, the case for $k = 0$ is not covered and must be done explicitly. We observe that by an application of Lemma 6.6.8, multiple applications of Lemma 6.6.7 and cuts, we can obtain polynomial size eLNDT proofs of $B_i \longrightarrow t^{\mathbf{B}_{s-1}}_1$. The sequent $t^{\mathbf{B}_{s-1}}_0, B_i, C \longrightarrow d^{i,\mathbf{B}_{s-1},k}_{s-1,0,0}, t^{\mathbf{B}_{s-1}}_1$ then follows by constant number of weakening steps. Now suppose $k > 0$ and $s < N-1$. We present the reasoning below where lines 1. to 5. conclude with $t^{\mathbf{B}_{s-1}}_k, B_i, C \longrightarrow d^{i,\mathbf{B}_{s-1},k}_{s,0,0}, t^{\mathbf{B}_{s-1}}_{k+1}, B_{s-1}$ and lines 6. to 15. conclude with $t^{\mathbf{B}_{s-1}}_k, B_i, C \longrightarrow d^{i,\mathbf{B}_{s-1},k}_{s,0,0}, d^{i,\mathbf{B}_{s-1},k-1}_{s,1,0}, t^{\mathbf{B}_{s-1}}_{k+1}$. Then, a last positive decision right step on lines 5. and 15. obtains the conclusion: $t^{\mathbf{B}_{s-1}}_k, B_i, C \longrightarrow d^{i,\mathbf{B}_{s-1},k}_{s-1,0,0}, t^{\mathbf{B}_{s-1}}_{k+1}$. Some structural rules may be omitted for brevity.

1. $t_k^{\mathbf{B}_s}, B_i, C \longrightarrow d_{s,0,0}^{i,\mathbf{B}_s,k}, t_{k+1}^{\mathbf{B}_s}$ (IH on $\mathbf{B}_s$)
2. $t_k^{\mathbf{B}_s}, B_i, C \longrightarrow d_{s,0,0}^{i,\mathbf{B}_{s-1},k}, t_{k+1}^{\mathbf{B}_s}$ (by Lemma 6.6.13 on 1.)
3. $t_k^{\mathbf{B}_s}, B_i, C \longrightarrow d_{s,0,0}^{i,\mathbf{B}_{s-1},k}, t_{k+1}^{\mathbf{B}_{s-1}}, B_{s-1}$ (by Lemma 6.6.7 on 2. and w-r)
4. $B_{s-1}, t_{k-1}^{\mathbf{B}_s} \longrightarrow B_{s-1}$ (id on B_{s-1})
5. $t_k^{\mathbf{B}_{s-1}}, B_i, C \longrightarrow d_{s,0,0}^{i,\mathbf{B}_{s-1},k}, t_{k+1}^{\mathbf{B}_{s-1}}, B_{s-1}$ (posdec-l on 3.+4.)

6. $t_{k-1}^{\mathbf{B}_s}, B_i, C \longrightarrow d_{s,0,0}^{i,\mathbf{B}_s,k-1}, t_k^{\mathbf{B}_s}$ (IH on $\mathbf{B}_s$)
7. $t_{k-1}^{\mathbf{B}_s}, B_i, C \longrightarrow d_{s,0,0}^{i,\mathbf{B}_{s-1},k-1}, t_k^{\mathbf{B}_s}$ (by Lemma 6.6.13 on 6.)
8. $d_{s,0,0}^{i,\mathbf{B}_{s-1},k-1} \longrightarrow d_{s,1,0}^{i,\mathbf{B}_{s-1},k-1}$ (by Lemma 6.6.12)
9. $t_{k-1}^{\mathbf{B}_s}, B_i, C \longrightarrow d_{s,1,0}^{i,\mathbf{B}_{s-1},k-1}, t_k^{\mathbf{B}_s}, t_{k+1}^{\mathbf{B}_s}$ (cut on 7.,8. and w-r)
10. $B_{s-1} \longrightarrow t_{k+1}^{\mathbf{B}_s}, B_{s-1}$ (id on B_{s-1})
11. $B_{s-1}, t_{k-1}^{\mathbf{B}_s}, B_i, C \longrightarrow d_{s,1,0}^{i,\mathbf{B}_{s-1},k-1}, t_{k+1}^{\mathbf{B}_{s-1}}$ (posdec-r on 9.+10.)

12. $t_k^{\mathbf{B}_s}, B_i, C \longrightarrow d_{s,0,0}^{i,\mathbf{B}_s,k}, t_{k+1}^{\mathbf{B}_s}$ (IH on $\mathbf{B}_s$, same as 1.)
13. $t_k^{\mathbf{B}_s}, B_i, C \longrightarrow d_{s,0,0}^{i,\mathbf{B}_{s-1},k}, t_{k+1}^{\mathbf{B}_s}$ (by Lemma 6.6.13 on 12.)
14. $t_k^{\mathbf{B}_s}, B_i, C \longrightarrow d_{s,0,0}^{i,\mathbf{B}_{s-1},k}, t_{k+1}^{\mathbf{B}_{s-1}}$ (by Lemma 6.6.7 on 13.)
15. $t_k^{\mathbf{B}_{s-1}}, B_i, C \longrightarrow d_{s,0,0}^{i,\mathbf{B}_{s-1},k}, d_{s,1,0}^{i,\mathbf{B}_{s-1},k-1}, t_{k+1}^{\mathbf{B}_{s-1}}$ (posdec-l on 11.+14.)

16. $t_k^{\mathbf{B}_{s-1}}, B_i, C \longrightarrow d_{s-1,0,0}^{i,\mathbf{B}_{s-1},k}, t_{k+1}^{\mathbf{B}_{s-1}}$ (conclusion, posdec-r on 5.+15.)

Where we write IH when invoking induction hypothesis.

The case for $i = s-1$ i.e. deciding upon B_{s-1} is different. We recall what was mentioned in Remark 6.6.3: while conjunction is not native in our systems, there are ways to concisely express the conjunction of two formulas. By using Definition 6.6.1 , the conjunction $B_i \wedge B_j$ can be expressed as the complex positive decision $0B_i(0 \vee B_j)$ over the set of extension axioms $\mathcal{B}^+_{\mathbf{B}}$. We want to make use of this here but first consider the complex positive decision $F = (0t_k^{\mathbf{B}_s}(0 \vee A))B_{s-1}((0t_k^{\mathbf{B}_s}(0 \vee A)) \vee 0t_{k-1}^{\mathbf{B}_s}(0 \vee C))$, defined over the appropriate extension axiom set extending $\mathcal{D}^{\mathbf{B}}$. Akin to what was done in Definition 6.6.1, we can consider the extension axiom set $(\mathcal{D}^{\mathbf{B}})^+_C$ where $\mathbf{C} = \mathbf{B} \cup \{t_l^{\mathbf{B}_s}\}_{l \leqslant k}$ which is sufficient for defining the complex positive decisions we need. Now, we make the following observation: the graph of F, is isomorphic to the graph of $d_{s-1,0,0}^{s-1,\mathbf{B}_{s-1},k}$. This follows by careful inspection of the extension axioms/variables involved (essentially 1-1 correspondence of nodes and their immediate descendants between the graphs).

Then, by Proposition 6.2.8, it is enough to prove the result for the above complex positive decision F. In addition, whenever we refer to Lemma 6.6.2 we understand the result (truth conditions) to be ranging over $(\mathcal{D}^{\mathbf{B}})^+_C$. The proof proceeds as follows:

1. $B_{s-1} \longrightarrow 0t_k^{\mathbf{B}_s}(0 \vee A), B_{s-1}$ (id on B_{s-1})
2. $t_k^{\mathbf{B}_s}, B_{s-1} \longrightarrow t_{k+1}^{\mathbf{B}_{s-1}}$ (Lemmas 6.6.7, 6.6.8)
3. $t_{k-1}^{\mathbf{B}_s}, B_{s-1}, C \longrightarrow 0t_{k-1}^{\mathbf{B}_s}(0 \vee C)$ (Lemma 6.6.2, w-l)
4. $t_k^{\mathbf{B}_{s-1}}, B_{s-1}, C \longrightarrow 0t_{k-1}^{\mathbf{B}_s}(0 \vee C), t_{k+1}^{\mathbf{B}_{s-1}}$ (posdec-l on 2.+3.)
5. $t_k^{\mathbf{B}_{s-1}}, B_{s-1}, C \longrightarrow 0t_{k-1}^{\mathbf{B}_s}(0 \vee C), t_{k+1}^{\mathbf{B}_{s-1}}, 0t_k^{\mathbf{B}_s}(0 \vee A)$ (w-r on 4.)
6. $t_k^{\mathbf{B}_{s-1}}, B_{s-1}, C \longrightarrow 0t_k^{\mathbf{B}_s}(0 \vee A)B_{s-1}(0t_k^{\mathbf{B}_s}(0 \vee A) \vee 0t_{k-1}^{\mathbf{B}_s}(0 \vee C)), t_{k+1}^{\mathbf{B}_{s-1}}$ (posdec-r on 1.+5.) □

From here, Theorem 6.6.9 follows immediately as a special case of Proposition 6.6.14 above:

Proof. Simply consider Proposition 6.6.14 for for $s = 0$. □

6.7 From strategies to proofs

The main result of this section is the converse of Theorem 6.5.1:

Theorem 6.7.1. eLNDT *(*eLDT*) poylnomially simulates* NB *(*DB*, resp.).*

We remind that in the following sections (unless explicitly stated) we work within the grammar of 0/1-eLNDT but for simplicity write eLNDT.

Remark 6.7.2. We only deal with the case of eLNDT and NB since the deterministic case (eLDT and DB) is simpler and does not require the results of the previous section i.e. Theorem 6.6.9. We remind that this is because the negation of a deterministic BP A is straightforward to define: $\neg A$ is simply the BP for A but with each leaf flipped from 1 to 0 and vice versa. Hence, the result would follow by simply defining conjunctions and disjunctions of BPs using decisions in eLDT. For example, for A, B eDT formulas $A \wedge B$ could be defined as a fresh extension variable alongside the extension axiom: $A \wedge B \leftrightarrow 0AB$ and so on. The natural 'truth conditions' of such constructions are easy to verify, similarly to Lemma 6.6.2.

In what follows we present a series of intermediate polynomial simulations which, when composed, yield Theorem 6.7.1.

6.7.1 De Morgan normal form of strategies

Before translating strategies to proofs, it will be useful to work with strategies in 'De Morgan' form: each query can only have negation ($\neg$) in front of eNDT formulas. This means that there cannot be conjunction ($\wedge$) in the scope of a negation. We will write $\mathsf{NB}^{\mathrm{DM}}$ for the fragment of NB according to this syntactic restriction on queries. Formally:

Definition 6.7.3 ($\mathsf{NB}^{\mathrm{DM}}$)**.** The fragment $\mathsf{NB}^{\mathrm{DM}}$ of NB only admits queries given by the following grammar:

$$\mathcal{L} ::= A \mid \neg A \mid Q \wedge R \mid Q \vee R$$

where A is an eNDT formula.

Since it will be necessary to translate an NB strategy to an $\mathsf{NB}^{\mathrm{DM}}$ strategy, we need to define a translation of queries by performing induction on the size of queries:

Definition 6.7.4 (DM-translation)**.** For each NB-query Q we define an $\mathsf{NB}^{\mathrm{DM}}$-query Q^{DM} by:

$$\begin{aligned} A^{\mathrm{DM}} &:= A \\ (\neg A)^{\mathrm{DM}} &:= \neg A \\ (Q \wedge R)^{\mathrm{DM}} &:= Q^{\mathrm{DM}} \wedge R^{\mathrm{DM}} \\ (Q \vee R)^{\mathrm{DM}} &:= Q^{\mathrm{DM}} \vee R^{\mathrm{DM}} \end{aligned} \qquad \begin{aligned} (\neg\neg Q)^{\mathrm{DM}} &:= Q^{\mathrm{DM}} \\ (\neg(Q \vee R))^{\mathrm{DM}} &:= (\neg Q)^{\mathrm{DM}} \wedge (\neg R)^{\mathrm{DM}} \\ (\neg(Q \wedge R))^{\mathrm{DM}} &:= (\neg Q)^{\mathrm{DM}} \vee (\neg R)^{\mathrm{DM}} \end{aligned}$$

Remark 6.7.5 (Structure of formulas preserved)**.** Notice that the structure of the binary trees of queries is preserved under the DM-translation. This can be made formal by defining an isomorphism f between the nodes in the binary-tree T_Q of Q not labelled by $\neg$ and the nodes in its respective translation $T_{Q^{\text{DM}}}$. The domain of f will be $N(T_Q)^{\neg}$, the set of nodes of T_Q not labelled by $\neg$ and the codomain will be $N(T_{Q^{\text{DM}}})$, the set of nodes in $T_{Q^{\text{DM}}}$.

We define:

$$\begin{aligned} f: \ N(T_Q)^{\neg} &\to N(T_{Q^{\text{DM}}}) \\ u &\mapsto u^{\text{DM}} \end{aligned}$$

where the subformula Q_u^{DM} of Q^{DM} rooted at node u^{DM} is the translation of the subformula Q_u of Q rooted at node u if there is an even number of nodes labelled by $\neg$ in the path of T_Q from the root to u. If otherwise there is an odd number of nodes labelled by $\neg$ in that path, Q_u^{DM} is the translation of $\neg Q_u$. Similarly, a leaf B_i of T_Q is either mapped to B_i if there an even number of nodes labelled by $\neg$ on the path leading to B_i or to $\neg B_i$ if the number is odd.

Furthermore, a DM-translated query Q^{DM} and its 'dual' $\neg Q^{\text{DM}}$ will have isomorphic trees:

We construct an isomorphism:

$$\begin{aligned} N(T_{Q^{\text{DM}}}) &\to N(T_{\neg Q^{\text{DM}}}) \\ u &\mapsto u' \end{aligned}$$

such that the label of u' is the dual of u, each leaf B_i is mapped to $\neg B_i$ and the subformula Q_u^{DM} of Q^{DM} rooted at u and the subformula $\neg Q_{u'}^{\text{DM}}$ of $\neg Q^{\text{DM}}$ rooted at u' have the same size $|Q_u^{\text{DM}}| = |\neg Q_{u'}^{\text{DM}}|$.

A well known result in Boolean-circuit complexity is Spira's lemma. There have been multiple results that either improve or generalize Spira's lemma, for example in [15] they provided sharper bounds and in [14] they presented a simplified version of previous proofs. It will be useful to us and thus present and prove a version of it:

Lemma 6.7.6 (Spira's lemma [57])**.** *Let T be a binary tree. Then, there is a 'mid-way' node r such that the subtree T_r of T rooted at r has size $|\mathsf{T}|\frac{1}{3} - 1 \leqslant |\mathsf{T}_r| \leqslant |\mathsf{T}|\frac{2}{3}$.*

Proof sketch. To find r we start at the root of T and traverse downwards towards the leaves. At each node v with edges e_1, e_2 leading to nodes v_1, v_2, we go along the edge leading to the v_1 if the subtree T_{v_1} of T, rooted at v_1, has size larger than the subtree rooted at v_2 (and vice versa). Assuming we went towards v_1, we observe that T_{v_1} will have size (number of nodes) $|\mathsf{T}_{v_1}| \leqslant |\mathsf{T}_v| - 1$ and $|\mathsf{T}_{v_1}| \geqslant (\frac{1}{2} \cdot |\mathsf{T}_v| - 1)$. We stop when we have for the first time reached a node u such that the subtree T_u has size $|\mathsf{T}_u| < \frac{1}{3} \cdot |\mathsf{T}|$. The node r we were looking for is the parent node of u. □

The DM-translation will actually be lifted to strategies themselves but before achieving that, we need to prove some intermediate results. The first constructs a winning strategy from a specification where the assignment is inconsistent with some substitution of queries:

Lemma 6.7.7 (Substitution)**.** *Let Q, Q', S be NB or NB^{DM} queries with Q' a subquery of S and S a subquery of Q, and let $l \in \{0, 1\}$. From $\{Q \mapsto b, Q[Q'/l] \mapsto 1 - b, S \mapsto t, S[Q'/l] \mapsto t\}$ there is a winning strategy with $O(\log(\mathit{depth}(Q)))$ rounds.*

Proof. In a divide and conquer fashion, we perform induction on the length of the path π_1 starting at the root of Q and reaching the root of Q' in Q and the identical path π_2 in $Q[Q'/l]$. We write 'S.C.' for simple contradictions, and '*IH* ' for inductive hypotheses.

Base cases: The length of π_1 is 0. That means $Q = S = Q'$. From the specification $[Q \mapsto b, l \mapsto 1-b, Q \mapsto l, l \mapsto l]$ if $b = l$ we obtain a S.C. from $[l \mapsto 1-l]$ else if $b = 1-l$ we obtain a S.C. from $[Q \mapsto 1-l, Q \mapsto l]$.

The length of π_1 is 1. For some formula G and $\circ \in \{\wedge, \vee\}$, we have that Q is $\circ(Q', G)$ and $Q[Q'/l]$ is $\circ(l, G)$. S could either be Q or Q'.
Assume $S = Q'$. The specification $[Q \mapsto b, Q[Q'/l] \mapsto 1-b, Q' \mapsto l, l \mapsto l]$ obtains a S.C for the truth table of $\circ$.
Assume $S = Q$. Given the specification $[Q \mapsto b, Q[Q'/l] \mapsto 1-b, Q \mapsto l, Q[Q'/l] \mapsto l]$ if $b = l$ we obtain a S.C from $[Q[Q'/l] \mapsto 1-l, Q[Q'/l] \mapsto l]$. Else if $b = 1-l$ we obtain a S.C. from $[Q \mapsto 1-l, Q \mapsto l]$.

Induction step: ask the formulas $R_1, R_1[Q'/l]$ respectively rooted at the nodes serving as mid points of the paths π_1, π_2. If $R_1 \mapsto d, R_1[Q'/l] \mapsto d$ we invoke *IH* for $[Q \mapsto b, Q[Q'/l] \mapsto 1-b, R_1 \mapsto d, R_1[Q'/l] \mapsto d]$. Else, we invoke *IH* for $[R_1 \mapsto d, R_1[Q'/l] \mapsto 1-d, Q' \mapsto l, l \mapsto l]$.

□

In the above, we bounded the rounds of the winning strategy by referring to the depth of the query Q which, we construe as is the maximum length of a path from the root of Q to a (maximal) eNDT formula. Next, we use the fact that each binary tree has a subtree of roughly half the size from Lemma 6.7.6, to construct a divide-and-conquer strategy for De Morgan duality:

Lemma 6.7.8 (Duality)**.** *Given an* NB*-query Q there is a winning strategy in* $\mathsf{NB}^{\mathrm{DM}}$ *from $\{Q^{\mathrm{DM}} \mapsto b, (\neg Q)^{\mathrm{DM}} \mapsto b\}$ with $O(\log(|Q|)$ rounds.*

Proof . We will perform (divide and conquer) induction on N, the size of Q^{DM} and $\neg Q^{\mathrm{DM}}$.

Base case: $Q^{\mathrm{DM}}, (\neg Q^{\mathrm{DM}})$ are $B, \neg B$ for B an eNDT formula. From the specification $[B \mapsto b, \neg B \mapsto b]$ we obtain a S.C..

Induction step: by Lemma 6.7.6 on the trees of Q^{DM} and $\neg Q^{\mathrm{DM}}$ we find the respective 'mid-way' nodes r and r'. To be suggestive we name the subtrees rooted at r, r': $\frac{Q^{\mathrm{DM}}}{2}, \frac{\neg Q^{\mathrm{DM}}}{2}$ and we know by Remark 6.7.5 that $\frac{Q^{\mathrm{DM}}}{2}, \frac{\neg Q^{\mathrm{DM}}}{2}$ will be isomorphic.

We start by asking $\frac{Q^{\mathrm{DM}}}{2}$ and $\frac{\neg Q^{\mathrm{DM}}}{2}$.

Case 1: If $\frac{Q^{\mathrm{DM}}}{2} \mapsto l, \frac{\neg Q^{\mathrm{DM}}}{2} \mapsto l$ we invoke *IH* for N', the size of $\frac{Q^{\mathrm{DM}}}{2}$ and $\frac{\neg Q^{\mathrm{DM}}}{2}$, where $\frac{2N}{3} \geqslant N' \geqslant \frac{N}{3}$.

Case 2: If $\frac{Q^{\mathrm{DM}}}{2} \mapsto l, \frac{\neg Q^{\mathrm{DM}}}{2} \mapsto 1-l$ we ask $Q^{\mathrm{DM}}[\frac{Q^{\mathrm{DM}}}{2}/l]$ and $\neg Q^{\mathrm{DM}}[\frac{\neg Q^{\mathrm{DM}}}{2}/1-l]$.

If the Adversary answers: $Q^{\mathrm{DM}}[\frac{Q^{\mathrm{DM}}}{2}/l] \mapsto d, \neg Q^{\mathrm{DM}}[\frac{\neg Q^{\mathrm{DM}}}{2}/1-l] \mapsto d$, we invoke *IH* for $N - N'$ the size of $Q^{\mathrm{DM}}[\frac{Q^{\mathrm{DM}}}{2}/l]$ and $\neg Q^{\mathrm{DM}}[\frac{\neg Q^{\mathrm{DM}}}{2}/1-l]$, where $\frac{2N}{3} \geqslant N - N' \geqslant \frac{N}{3}$.

If instead the Adversary answered: $Q^{\mathrm{DM}}[\frac{Q^{\mathrm{DM}}}{2}/l] \mapsto 1-d, \neg Q^{\mathrm{DM}}[\frac{\neg Q^{\mathrm{DM}}}{2}/1-l] \mapsto d$ we know that depending on whether $b = 1-d$ or $b = d$, one of the specifications:

- $[Q^{\mathrm{DM}}[\frac{Q^{\mathrm{DM}}}{2}/l] \mapsto 1-d, \frac{Q^{\mathrm{DM}}}{2} \mapsto l, Q^{\mathrm{DM}} \mapsto b]$
- $[\neg Q^{\mathrm{DM}}[\frac{\neg Q^{\mathrm{DM}}}{2}/1-l] \mapsto d, \frac{\neg Q^{\mathrm{DM}}}{2} \mapsto 1-l, \neg Q^{\mathrm{DM}} \mapsto b]$

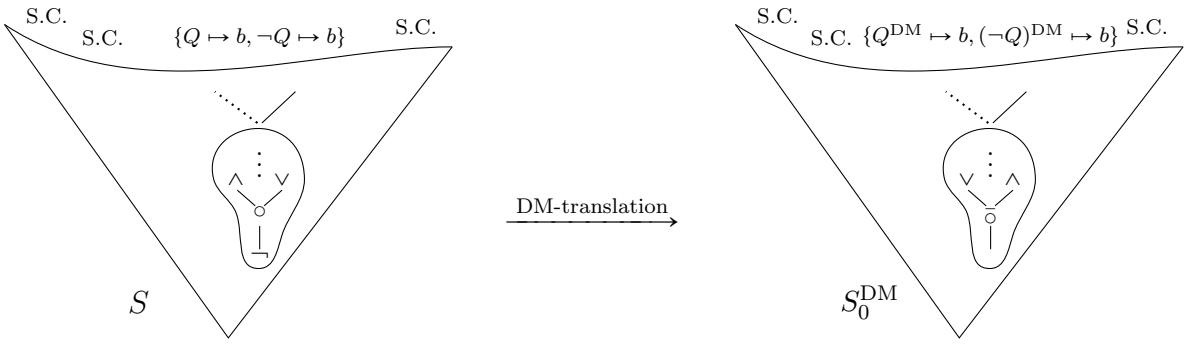


Figure 6.7: Visualisation of an NB strategy (left) and its corresponding $\mathsf{NB}^{\mathrm{DM}}$ 'pre-strategy' (right). 'Leaves' of the form $\{Q^{\mathrm{DM}} \mapsto b, (\neg Q)^{\mathrm{DM}} \mapsto b\}$ are not simple contradictions of $\mathsf{NB}^{\mathrm{DM}}$.

is inconsistent respectively. In either case we finish by applying Lemma 6.7.7. □

Finally, by replacing each query of an NB-strategy by its De Morgan translation, we can simulate simple contradictions for $\neg$ using the Lemma above, yielding:

Proposition 6.7.9. *Given an* NB *strategy S over $\mathcal{E}$ from an eNDT sequent winning in $\leqslant d$ rounds there is a* $\mathsf{NB}^{\mathrm{DM}}$ *strategy S^{DM} over $\mathcal{E}$ for the same sequent winning in $O(d + \log(|R|))$ rounds, for R the largest query in S.*

Proof. From an NB strategy S for an eLNDT sequent, we can obtain an $\mathsf{NB}^{\mathrm{DM}}$ 'pre-strategy' S_0^{DM} for the same sequent by replacing each query Q by Q^{DM}. S_0^{DM} has $\mathsf{NB}^{\mathrm{DM}}$ queries but not all leaves are simple contradictions yet in $\mathsf{NB}^{\mathrm{DM}}$. This is done by constructing a copy of the tree of S and replacing each query by its DM-translation. Observe that there is only one schema of simple contradictions in the NB game not present in $\mathsf{NB}^{\mathrm{DM}}$: $\{Q \mapsto b, \neg Q \mapsto b\}$ where $b \in \{0, 1\}$ and Q is a (non-trivial) Boolean combination of eNDT formulas (i.e. not an eNDT formula).

Finding $O(\log(\mathrm{depth}(Q)))$ strategies in $\mathsf{NB}^{\mathrm{DM}}$ for each leaf of S_0^{DM} of the form: $[Q^{\mathrm{DM}} \mapsto b, (\neg Q)^{\mathrm{DM}} \mapsto b]$ provides us with a strategy S^{DM} in $\mathsf{NB}^{\mathrm{DM}}$ which will inherit the tree structure of S_0^{DM}. For these we appeal to Lemma 6.7.8, yielding the required bounds.

The transformation is visualised in Figures 6.7 and 6.8.

□

6.7.2 From $\mathsf{NB}^{\mathrm{DM}}$ to $\mathrm{Bool}^{+}(\mathsf{eLNDT})$

We now use our formalisation of Immerman-Szelepcsényi in order to translate De Morgan strategies to proofs without negation. It will be useful to first translate to a version of eLNDT with formulas closed under *positive* Boolean combinations.

Definition 6.7.10. Write $\mathrm{Bool}^{+}(\mathsf{eLNDT})$ for the extension of eLNDT allowing $\{\vee, \wedge\}$-combinations of eNDT formulas i.e. using the following grammar:

$$C, D, \ldots \quad ::= \quad A \mid (C \vee D) \mid (C \wedge C)$$

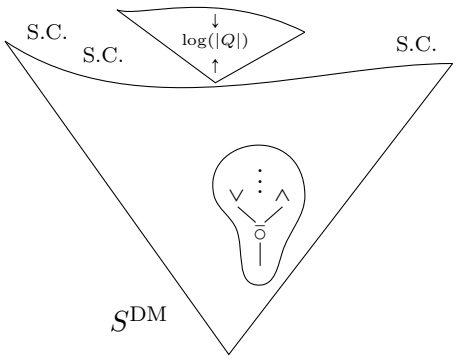


Figure 6.8: Translated strategy S^{DM}. The 'leaf-strategies' are instances of $O(\log(|Q|))$-depth strategies from $[Q^{\mathrm{DM}} \mapsto b, (\neg Q^{\mathrm{DM}}) \mapsto b]$ by Lemma 6.7.8

where A is an eNDT formula and admiting the following corresponding rules for the connectives $\vee, \wedge$:

$$\begin{array}{cc} \wedge\text{-}l\,\dfrac{\Gamma, P, Q \longrightarrow \Delta}{\Gamma, P \wedge Q \longrightarrow \Delta} & \wedge\text{-}r\,\dfrac{\Gamma \longrightarrow \Delta, P \quad \Gamma \longrightarrow \Delta, Q}{\Gamma \longrightarrow \Delta, P \wedge Q} \\ \vee\text{-}r\,\dfrac{\Gamma \longrightarrow \Delta, P, Q}{\Gamma \longrightarrow \Delta, P \vee Q} & \vee\text{-}l\,\dfrac{\Gamma, P \longrightarrow \Delta \quad \Gamma, Q \longrightarrow \Delta}{\Gamma, P \vee Q \longrightarrow \Delta} \end{array} \tag{6.1}$$

By appealing to the k-deciders from Section 6.6, in particular using the truth conditions in Theorem 6.6.9, we can simulate negations in eLNDT and $\mathrm{Bool}^+(\mathsf{eLNDT})$. As a result we have:

Proposition 6.7.11. *Let S be an $\mathsf{NB}^{\mathrm{DM}}$ strategy over $\mathcal{B}$ for some NDT formula B and let $\mathbf{B}$ enumerate all the eNDT formulas occurring in S. For each $k \leqslant |\mathbf{B}|$ there are $\mathrm{Bool}^+(\mathsf{eLNDT})$ proofs of $t_k^{\mathbf{B}} \longrightarrow B, t_k^{\mathbf{B}}$ of size polynomial in $|S|$, over a set of extension axioms extending $\mathcal{D}^{\mathbf{B}}$ cf. Section 6.6.2.*

Proof. **Idea:** We proceed inductively on the structure of S. For each query Q write Q^k for the result of replacing each subquery $\neg B_i$ by the respective decider $1B_i^k0$. The proof will mostly contain cuts on the respective queries of the Prover: whenever S concludes with a query Q we conduct a cut on Q^k, while always maintaining the property that formulas answered 0 by the adversary appear on the RHS of a sequent, and formulas answered 1 appear on the LHS. We always keep $t_k^{\mathbf{B}}$ on the LHS and $t_{k+1}^{\mathbf{B}}$ on the RHS. In this way, any simple contradictions of S for $\neg$ correspond to sequents:

$$t_k^{\mathbf{B}}, \Gamma, B_i, 1B_i^k0 \longrightarrow \Delta, t_{k+1}^{\mathbf{B}}, \qquad t_k^{\mathbf{B}}, \Gamma \longrightarrow B_i, 1B_i^k0, \Delta, t_{k+1}^{\mathbf{B}}$$

for $\{B_i \mapsto 1, \neg B_i \mapsto 1\}$ or for $\{B_i \mapsto 0, \neg B_i \mapsto 0\}$ respectively. Both of these sequents have polynomial-size proofs by Theorem 6.6.9. Any other simple contradictions also translate to polynomial-size proofs, in particular of decision truth conditions (using the decision rules), positive truth conditions (using the positive Boolean rules of (6.1)), extension axioms or similarity (using Proposition 6.2.8). The transformation is visualised in Figure 6.9.

Formally: We perform induction on the number of queries n of S. As mentioned before, for each query Q of S we construe Q^k as the result of replacing each subquery $\neg B_i$ (where B_i is an eNDT formula) in Q by the respective decider $1B_i^k0$ defined over the list of formulas $\mathbf{B}$.

Assume that for a subtree T of S of size d rooted at B we have constructed an Bool$^+$(eLNDT) derivation P_k^{T} that has the sequent $t_k^{\mathbf{B}} \longrightarrow B, t_{k+1}^{\mathbf{B}}$ as conclusion and the sequents $\{t_k^{\mathbf{B}}, \Sigma_i \longrightarrow \Delta_i, t_{k+1}^{\mathbf{B}}\}_{i\in I}$ as assumptions where I is a set of indices enumerating the maximal paths in T and $\Sigma_i = \{Q \in i : Q \mapsto 1\}$, $\Delta_i = \{Q \in i : Q \mapsto 0\}$.
Increasing the size of T by one, would w.l.o.g. mean adding a new non-leaf node $Q_\theta, \theta \in \{0,1\}^*$ with outwards edges labeled with 0 to the left and 1 to the right. If Q_θ is added to the path i of T we add to P_k^{T} a cut step introducing Q_θ^k:

$$cut\frac{t_k^{\mathbf{B}}, Q_\theta^k, \Sigma_i \longrightarrow \Delta_i, t_{k+1}^{\mathbf{B}} \quad t_k^{\mathbf{B}}, \Sigma_i \longrightarrow \Delta_i, Q_\theta^k, t_{k+1}^{\mathbf{B}}}{t_k^{\mathbf{B}}, \Sigma_i \longrightarrow \Delta_i, t_{k+1}^{\mathbf{B}}}$$

with conclusion the sequent $t_k^{\mathbf{B}}, \Sigma_i \longrightarrow \Delta_i, t_{k+1}^{\mathbf{B}}$ which was a hypothesis for P_k^{T}. Our new derivation $P_k^{\mathsf{T}'}$ corresponding to the subtree $\mathsf{T}' = \mathsf{T} \cup Q_\theta$ has $t_k^{\mathbf{B}} \longrightarrow B, t_{k+1}^{\mathbf{B}}$ as conclusion and the same hypotheses as P_k^{T} besides the ones at path θ. There, the difference is that the sequent $t_k^{\mathbf{B}}, \Sigma_i \longrightarrow \Delta_i, t_{k+1}^{\mathbf{B}}$ is proven by the following new sequents:

$$t_k^{\mathbf{B}}, Q_\theta^k, \Sigma_i \longrightarrow \Delta_i, t_{k+1}^{\mathbf{B}}, \qquad t_k^{\mathbf{B}}, \Sigma_i \longrightarrow \Delta_i, Q_\theta^k, t_{k+1}^{\mathbf{B}}$$

which serve as new hypotheses. The derivation $P_k^{\mathsf{T}'}$ is one step larger than P_k^{T}.

By induction hypothesis, from an $\mathsf{NB}^{\mathrm{DM}}$-strategy rooted at B, we can construct a family of Bool$^+$(eLNDT) derivations each proving $t_k^{\mathbf{B}} \longrightarrow B, t_{k+1}^{\mathbf{B}}$ for k bounded by l. This construction goes bottom-up until reaching the leaves of S and we observe that each derivation is tree-like up to that point.
The leaves of S are simple contradictions in $\mathsf{NB}^{\mathrm{DM}}$ which in the derivation P_k^S may correspond to constant sized proofs of axioms, polynomial size dag-like proofs of sequents $A_1 \longrightarrow B_1$ where A_1, B_1 have the same unfolding by Proposition 6.2.4 or polynomial size proofs of the truth conditions in Proposition 6.6.14.

Complexity: From S, a $\mathsf{NB}^{\mathrm{DM}}$-strategy for B of size n (number of queries) we obtain a family of $l = \mathrm{length}(\mathbf{B})$ many derivations, each with conclusion $t_k^{\mathbf{B}} \longrightarrow B, t_{k+1}^{\mathbf{B}}$ for some $k \leqslant l$. Each derivation has up to $n + O(1)$ steps and every step contains a sequent with $O(n)$ many formulas. Since the leaves of S also correspond to polynomial size proofs, the size of each derivation is polynomial in n. □

By simply cutting together $|\mathbf{B}| + 1$ many proofs, and appealing to short proofs of $\longrightarrow t_0^{\mathbf{B}}$ and $t_{|\mathbf{B}|+1}^{\mathbf{B}} \longrightarrow$ (cf. Definition 6.6.4), we have:

Corollary 6.7.12. Bool$^+$(eLNDT) polynomially simulates $\mathsf{NB}^{\mathrm{DM}}$ over NDT formulas.

6.7.3 From Bool$^+$(eLNDT) to eLNDT.

Recalling Section 6.6.1, note that we can already interpret positive Boolean combinations (over a fixed set $\mathbf{B}$ of eNDT formulas and axioms) within eLNDT. In particular note that $A \wedge B$, over a set of extension axioms $\mathcal{E}$, is equivalent to the extension variable $0A(0 \vee B)$, over $\mathcal{E}^+_{\mathbf{B}}$, cf. Definition 6.6.1. Thus we immediately obtain:

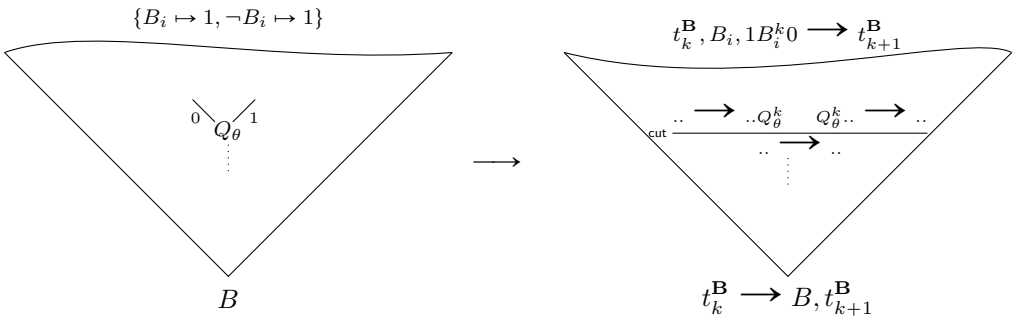


Figure 6.9: Visualisation of the translation from a $\mathsf{NB}^{\mathrm{DM}}$ strategy (left) to a $\mathrm{Bool}^+(\mathsf{eLNDT})$ proof (right). 'Leaves' of the form $t_k^{\mathbf{B}}, B_i, 1B_i^k 0 \longrightarrow t_{k+1}^{\mathbf{B}}$ are not axioms, but are rather justified by the formalisation of Immerman-Szelepcsényi, Theorem 6.6.9. The dual situation, for simple contradictions $\{B_i \mapsto 0, \neg B_i \mapsto 0\}$ is handled similarly.

Proposition 6.7.13. eLNDT *polynomially simulates* $\mathrm{Bool}^+(\mathsf{eLNDT})$ *over NDT formulas.*

Now, our main result Theorem 6.7.1, follows by Propositions 6.7.9 and 6.7.13 and Corollary 6.7.12.

Chapter 7

Systems for OBDDs

Ordered binary decision diagrams (OBDDs) are deterministic branching programs in which variables labeling nodes in a path occur w.r.t. some fixed order. The notion was first introduced in works of Randal E. Bryant including [16] and later [17]. They enjoy many interesting properties, for example they contain no 'inconsistent' paths (containing both a 0-edge and a 1-edge coming out of nodes labelled by the same variable) since each variable may occur only once in each path, see for example Figure 3.1. Furthermore for a fixed ordering r of propositional variables $p_1, p_2 \ldots, p_n$ and for each Boolean function f with inputs from $p_1, p_2 \ldots, p_n$ there is a unique (up to isomorphism) r-OBDD (OBDD where occurrence of variables in each path is w.r.t. r) computing f with minimal size. This canonical way of representing Boolean functions via minimal r-OBDDs is achieved ([63] Section 3.3) by introducing two reduction rules, *elimination* and *merging* that essentially omit 'redundant' nodes, they are illustrated in Figure 7.1. This makes studying properties and relations between OBDDs efficient: checking for satisfiability, tautological behavior (the OBDD has only 1-leaves) and the equivalence between OBDDs is solvable in polynomial time [47]. Because of their many desirable features, the study of OBDDs has seen widespread interest and applications in recent years.

Studying OBDDs from a proof theoretic perspective was initiated in [9]. There, one of the main goals of Atserias, Kolaitis, and Vardi was to present a natural way of defining a proof system corresponding to each constraint satisfaction problem and as a case study, they defined *resolution-style* proof systems based on OBDDs. More specifically, for an unsatisfiable set of clauses $\mathcal{C}$ (alternatively an unsatisfiable Boolean formula in CNF), they consider refutations as sequences of OBDDs that either compute the Boolean function defined by a clause $C \in \mathcal{C}$ or are obtained from OBDDs by using the *join* ($\wedge$) or the *weakening* (w) rules (system $\mathsf{OBDD}(\wedge, \mathsf{w})$). The join rule conjugates two previously obtained OBDDs while the weakening rule introduces an OBDD that is implied by a previously obtained one. They continued to compare the strength of their systems with resolution and the Gaussian calculus concluding that these systems are exponentially weaker than $\mathsf{OBDD}(\wedge, \mathsf{w})$ which can polynomially simulate cutting planes. Buss, Itsykson, Knop, and Sokolov expanded on these results in [23], where they showed that $\mathsf{OBDD}(\wedge, \mathsf{w})$ can give exponentially shorter proofs than (dag-like) cutting planes. Proving that involved fixing a specific order for variables and showing that Clique-Coloring tautologies have polynomial size proofs in $\mathsf{OBDD}(\wedge, \mathsf{w})$ under said

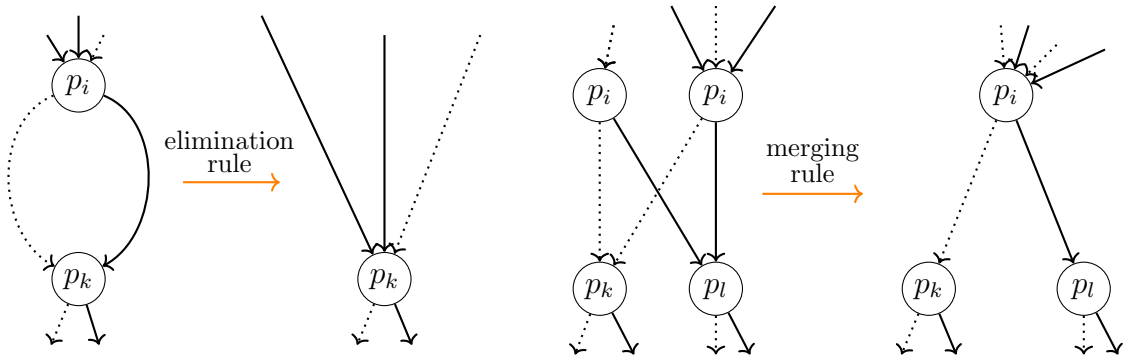


Figure 7.1: The reduction rules shown schematically cf. [63]. The direction of the orange arrow indicates locally replacing the structure on its left with the one on its right in the graph while maintaining incoming and outgoing edges.

order. Since it was already known from [50] that cutting planes has only exponential size proofs of Clique-Coloring, the separation follows. They further considered the rule 'reordering' (ord) (introduced in [38]), which allows changing the variable order for OBDDs and proved that the extension $\mathsf{OBDD}(\wedge, \mathsf{w}, \mathsf{ord})$ allowing for reordering, is strictly stronger than $\mathsf{OBDD}(\wedge, \mathsf{w})$. This was achieved by first showing that generally, given a CNF A that has polynomial size $\mathsf{OBDD}(\wedge, \mathsf{w})$ proofs under an order r and superpolynomial size $\mathsf{OBDD}(\wedge, \mathsf{w})$ proofs under a different order r', they can define a CNF $\mathcal{T}(A)$ which has polynomial size $\mathsf{OBDD}(\wedge, \mathsf{w}, \mathsf{ord})$ proofs but only exponential size $\mathsf{OBDD}(\wedge, \mathsf{w})$ proofs (that is, irrespective to the order used). More specifically, they considered a family of CNFs that encoded specific instances of Clique-Coloring under an order of variables which had poly-size $\mathsf{OBDD}(\wedge, \mathsf{w}, \mathsf{ord})$ proofs while only exponential size $\mathsf{OBDD}(\wedge, \mathsf{w})$ proofs.

Summary of our contribution

In this chapter we define a proof system for OBDDs (**o-eLDT**) as a fragment of eLDT. We will use specifically tailored extension variables/axioms to account for our formulas being 'ordered' so that they represent OBDDs. The system $\mathsf{OBDD}(\wedge, \mathsf{w})$ is formally given in Definition 7.2.1.[1] Intuitively, given a CNF $\mathcal{C}$, a derivation from $\mathcal{C}$ will be a sequence of OBDDs either representing a clause of $\mathcal{C}$ or obtained by the 'join' ($\wedge$) of two previous OBDDs or the 'weakening' of a previous OBDD. One big difference in our work will be the focus on syntax: the rules 'join' and 'weakening' in the system $\mathsf{OBDD}(\wedge, \mathsf{w})$ from [9] and [23] are semantically defined while we, provide syntactic ways of defining the corresponding rules in our system and later provide polynomial size proofs of their natural 'truth conditions'. More concretely, we introduce the judgement '$\models^i_{\mathcal{E}}$' parametrised by a 'level' i and a 'leveled' set of extension axioms $\mathcal{E}$ that, together with the appropriate 'truth conditions', will function as the weakening rule in our

[1] Definition is adjusted to better fit our setting.

setting. We will also define new sets of extension variables/axioms $\mathcal{E}^\wedge$ extending $\mathcal{E}$ that will permit admitting the join of OBDDs in our system i.e. $A \wedge B$ for A, B **o-**eDT formulas. Finally, we compare the expressive strength of **o-**eLDT to OBDD$(\wedge, \mathsf{w})$ from [23] and [9], showing that **o-**eLDT polynomially simulates OBDD$(\wedge, \mathsf{w})$.

7.1 The ordered fragment of eLDT

In this section we are going to define a proof system that canonically reasons about ordered binary decision diagrams just like eLDT reasons (more generally) about deterministic branching programs and Hilbert-Frege/LK reason about Boolean formulas. Naturally, we intend this system to be a fragment of eLDT and want it to be consistent with our previous exposition: lines in this system will be sequents of OBDDs all of which abide by a certain fixed ordering r of propositional variables. This additional constraint, that paths of OBDDs must display variables according to r, requires we restrict ourselves to eDT formulas that, together with their subformulas, represent r-OBDDs. A further complication arises, when 'combining' two r-OBDDs A, B i.e. when considering their join. While the naive approach works, that is considering all possible joins $u \wedge v$ as nodes in $A \wedge B$ for u, v nodes in A, B respectively, we must be careful about node-labelling. The issue is that $A \wedge B$ has to also be an r-OBDD and assigning the 'correct' propositional label to $u \wedge v$ is not obvious; for that we must take into account differences in the way the variables $p_1, p_2, \ldots, p_n$ occur in paths in A, B. For instance, even though both A, B are r-OBDDs, some paths in A or B may skip multiple variables (i.e. A, B may not be complete [63]). This issue could be taken care of in multiple ways, for example defining the extension axiom sets to be 'leveled' a notion which would make sure that extension variables adhere to the ordering r. One way to do that would be to construct the extension axiom set $\mathcal{E}$ so it only contains axioms of the form: $e_i \leftrightarrow bp_id$ for $b, d \in \{0, 1\} \cup \{e_j\}_j$ where $p_j < p_i$ w.r.t. the order r and $e_j \in \mathcal{E}$. We could then consider the formulas of our language to be given by the grammar

$$A, B ::= \ 1 \ \mid \ 0 \ \mid \ e \ \mid \ bp_id$$

for $b, d \in \{0, 1\} \cup \{e_j\}_j$ where $p_j < p_i$ w.r.t. r and $e \in \mathcal{E}$. In this way, it is clear that any formula defined over $\mathcal{E}$ we might consider, expresses an OBDD w.r.t. r. This however, would necessitate our proofs to conclude with sequents containing extension variables contrary to our previous convention where conclusions of proofs were extension free. Hence, our preferred way is to more generally define a notion of 'level' for eDT formulas.

Without loss of generality, let us fix r, the inverse standard ordering of propositional atoms: $p_1, p_2, \ldots, p_n$ i.e. $p_i < p_j$ when $i > j$. We choose our grammar to be the same as (2.5) for 0/1-eDT formulas that is,

$$A, B ::= \ 1 \ \mid \ 0 \ \mid \ ApB \ \mid \ e \tag{7.1}$$

As before, due to Remark 2.1.27 from now on we may only write eLDT, eDT etc. without causing any confusion. As mentioned, ensuring that our soon to be defined system will correctly correspond to the fragment of eLDT reasoning only about OBDDs requires some extra bookkeeping. To start with, all extension variables e will come

equipped with a notion of 'level' denoted by $lev(e)$, a natural number i that we may write as a superscript e^i that is meant to be an upper bound on the relative 'depth' (in the graph) of the root of the OBDD e^i represents. Then:

Definition 7.1.1. We define the **level** i of an eDT formula A ($lev(A)$) recursively:

- $lev(0) = lev(1) = 0$
- if $lev(A) = i, lev(B) = j$ and $i, j < k$ then Ap_kB has level k

We call a formula A **leveled** if it has a level. A set of extension axioms $\mathcal{E} = \{e_l \leftrightarrow E_l\}_{l<m}$ is **leveled** if for all $l < m$ it holds $lev(E_l) = lev(e_l)$.

Definition 7.1.2. The fragment of eLDT with only leveled eDT formulas over leveled sets of extension axioms is denoted by **o-**eLDT (**ordered** eLDT). The subset of eDT formulas that are leveled is denoted by **o-**eDT (**ordered eDT formulas**).

The system **o-**eLDT adequately reasons about OBDDs in the following sense: for each OBDD G there is an **o-**eDT formula A over a leveled set of extension axioms $\mathcal{E}$ that represents G and vice versa.

Remark 7.1.3. The choice of the grammar from (2.5) was simply to avoid having to define the level of individual propositional variables and simply reduce it to their index i.e. $0p_k1$ has level k. In addition note that this definition of level goes against our previous convention: here OBDDs we represent have their root labeled by some variable p_n and the rest of the nodes labeled by variables p_i for $i < n$ and so on. On the contrary previously e.g. Figure 2.1, Figure 2.6 and Figure 3.2 among others, we started drawing our BPs from the variable with the least natural number as index, usually p_1, and label the rest of the nodes with some p_i for $i > 1$. To some degree this is stylistic, and our different choice in this chapter is meant to better accommodate the intuition of 'level' where the closer you traverse to the leaves in a graph, the lower your 'level' should be matched by the index of the variable labeling the current node. More importantly, this convention allows us to concretely bound the size of OBDDs for a given set of propositional variables ordered by r, avoiding the need to relativize by size, allowing us to concisely define the notion of level[2], see for example Figure 7.2. In any case, a simple change in the fixed order of variables we consider can make BPs in the current setting look exactly like the ones from before e.g. graph G from Figure 2.6.

Remark 7.1.4. Like our previous systems, the proof complexity theoretic analysis of **o-**eLDT is meaningful since the set of valid extension-free **o-**eLDT sequents is **coNP**-complete. This is easy to see, each DNF can be represented as an **o-**eLDT sequent that is free of extension variables. Equivalently (and more convenient for what will come later) we consider proofs for sequents $F \longrightarrow$ where F is a set of extension free **o-**eDT formulas computing clauses (and joins/weakenings of clauses) of a CNF (essentially we want to 'refute' CNFs). To canonically represent CNFs by **o-**eDT sequents, we first set

[2]Think off considering a new propositional variable p_{n+1}. If a decision Ap_nB had level 1, we would need to change all levels to make it so all decisions on p_{n+1} have (instead) level 1. In our convention though, adding p_{n+1} can be done seamlessly as a decision on it will have level $n + 1$.

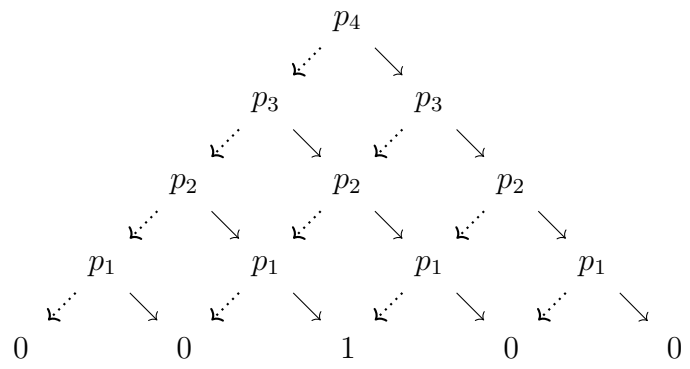


Figure 7.2: An OBDD computing the 2-out-of-4 Exact w.r.t. the inverse standard order r. The corresponding extension axioms can be defined analogously to the ones for the OBDD G from Figure 2.6

the clauses in a CNF $\mathcal{C} = \bigwedge_{s\in S} C_s$ to be ordered w.r.t. r, the inverse standard ordering of propositional variables $p_1, p_2, \ldots$. This way to each clause $C_s = q_{s_1} \vee q_{s_2} \vee \ldots \vee q_{s_k}$ for $s_k < \ldots < s_2 < s_1 \in S$ and each q_{s_j} a literal in $\{p_{s_j}, \bar{p_{s_j}}\}$ corresponds a unique extension-free **o-**eDT formula C_s^o, computing the respective Boolean function. C_s^o will be of the form $(\ldots(0q_{s_k}1\ldots)q_{s_2}1)q_{s_1}1$ which, since we can simulate a decision on a negated literal $A\bar{p}B$ by the decision BpA on the corresponding atom, will be expressed as a decision on the atoms p_{s_j} accordingly. In this way, CNF $\mathcal{C}$ can be represented by a sequent of the form $C_1^o, \ldots, C_s^o \longrightarrow$.

7.1.1 Entailment of OBDDs

The following definition is our syntactic approach to define a notion that corresponds to the weakening rule from [9]. We first introduce the judgement '$\models^i_{\mathcal{E}}$' and, since we want it to 'behave' as a logical entailment between OBDDs, provide polynomial size proofs of the sequent $A \longrightarrow B$ whenever $A \models^i_{\mathcal{E}} B$.

Definition 7.1.5. Given a leveled set of extension axioms $\mathcal{E} = \{e_l \leftrightarrow E_l\}_{l<m}$ define the judgement $A \models^i_{\mathcal{E}} B$ read as 'A **i-entails** B' when $lev(A), lev(B) \leqslant i$ as follows:

$$\frac{}{0 \models^i_{\mathcal{E}} A} \quad \frac{}{A \models^i_{\mathcal{E}} 1} \quad \frac{Ap_jB \models^{i-1}_{\mathcal{E}} Cp_kD}{Ap_jB \models^i_{\mathcal{E}} Cp_kD}\, {\scriptstyle j,k<i} \quad \frac{A \models^{i-1}_{\mathcal{E}} Cp_kD \quad B \models^{i-1}_{\mathcal{E}} Cp_kD}{Ap_iB \models^i_{\mathcal{E}} Cp_kD}\, {\scriptstyle k<i}$$

$$\frac{A \models^i_{\mathcal{E}} E_l}{A \models^i_{\mathcal{E}} e_l} \quad \frac{E_l \models^i_{\mathcal{E}} A}{e_l \models^i_{\mathcal{E}} A} \quad \frac{Ap_jB \models^{i-1}_{\mathcal{E}} C \quad Ap_jB \models^{i-1}_{\mathcal{E}} D}{Ap_jB \models^i_{\mathcal{E}} Cp_iD}\, {\scriptstyle j<i} \quad \frac{A \models^{i-1}_{\mathcal{E}} C \quad B \models^{i-1}_{\mathcal{E}} D}{Ap_iB \models^i_{\mathcal{E}} Cp_iD}$$

Thinking of the above as inference rules and given $A \models^i_{\mathcal{E}} B$, one can obtain a polynomial size dag-like derivation of $A \models^i_{\mathcal{E}} B$. We may more generally, write $A \models_{\mathcal{E}} B$ if $A \models^i_{\mathcal{E}} B$ for some $i \geqslant lev(A), lev(B)$ and make the observation that $A \models_{\mathcal{E}} B$ is essentially satisfaction adjusted for leveled eDT formulas, similar to Definition 2.1.8, which justifies using the same notation.

Proposition 7.1.6. *If A, B are leveled formulas over a leveled set of extension axioms $\mathcal{E}$ and $A \models_{\mathcal{E}} B$, there are polynomial size* ***o***-eLDT *proofs of $A \longrightarrow B$ over $\mathcal{E}$.*

Proof. The proof proceeds by induction on the judgement $\models^i_{\mathcal{E}}$. [3]

Base case: if either

$$\frac{}{0 \models^i_{\mathcal{E}} B} \qquad \text{or} \qquad \frac{}{A \models^i_{\mathcal{E}} 1}$$

the proof immediately follows by either

$$\frac{}{0 \longrightarrow}\,{\scriptstyle 0} \qquad \text{or} \qquad \frac{}{\longrightarrow 1}\,{\scriptstyle 1}$$

and a weakening right or left step respectively.

Induction step:

- if $Ap_iB \models^i_{\mathcal{E}} Cp_kD$ for $k < i$ the proof proceeds as follows:

$$\text{p-l}\,\frac{\text{w-r}\,\dfrac{IH}{A \longrightarrow p_i, Cp_kD} \qquad \text{w-l}\,\dfrac{IH}{p_i, B \longrightarrow Cp_kD}}{Ap_iB \longrightarrow Cp_kD}$$

- if $Ap_jB \models^i_{\mathcal{E}} Cp_iD$ for $j < i$ the proof proceeds as follows:

$$\text{p-r}\,\frac{\text{w-r}\,\dfrac{IH}{Ap_jB \longrightarrow p_i, C} \qquad \text{w-l}\,\dfrac{IH}{p_i, Ap_jB \longrightarrow D}}{Ap_jB \longrightarrow Cp_iD}$$

- if $Ap_iB \models^i_{\mathcal{E}} Cp_iD$ the proof proceeds as follows:

$$\text{p-r}\,\frac{\text{p-l}\,\dfrac{\dfrac{IH}{A \longrightarrow , p_i, p_i, C} \qquad \text{id}\,\dfrac{}{p_i, B \longrightarrow p_i, C}}{Ap_iB \longrightarrow p_i, C} \qquad \text{p-l}\,\dfrac{\text{id}\,\dfrac{}{p_i, A \longrightarrow p_i, D} \qquad \dfrac{IH}{\longrightarrow p_i, p_i, B \longrightarrow D}}{p_i, Ap_iB \longrightarrow D}}{Ap_iB \longrightarrow Cp_iD}$$

The cases for $A \models^i_{\mathcal{E}} e_l$ and $e_l \models^i_{\mathcal{E}} A$ immediately follow by the extension axioms $e_l \leftrightarrow E_l$ and induction hypothesis:

$$\mathcal{E}\,\frac{IH}{\dfrac{A \longrightarrow E_l}{A \longrightarrow e_l}} \qquad \mathcal{E}\,\frac{IH}{\dfrac{E_l \longrightarrow A}{e_l \longrightarrow A}}$$

□

[3] Similarly to how one may perform induction on the structure of a proof.

7.1.2 Joins of OBDDs

Another important feature our systems must have present, is the ability to admit the **join** i.e. the conjunction, of two OBDDs with the same order. Given two OBDDs G, F with the same order r, their join $G \wedge F$, must also have order r. While this is not difficult to define, we need to take into account that we do not demand our OBDDs go through all variables in each path and thus some variables in paths of G, F may be skipped. Hence, we have to be careful to adjust our definition for the OBDD that 'lags' behind. This is done via defining new extension variables/axioms that correspond to the join and considering several cases according to the relation between the levels of each pair of extension variables. We formalise it in the following definition:

Definition 7.1.7. Given a leveled set of extension axioms $\mathcal{E} = \{e_l \leftrightarrow E_l\}_{l<m}$ for all **o-**eDT formulas A, B over $\mathcal{E}$ of levels i, j respectively, we introduce fresh extension variables $A \wedge B$ of level $\max\{i, j\}$ and the following extension axioms:

$$\begin{array}{rll}
A \wedge 0 & \leftrightarrow 0 \quad 0 \wedge A \leftrightarrow 0 & \text{for } A \neq 0 \\
A \wedge 1 & \leftrightarrow A \quad 1 \wedge A \leftrightarrow A & \text{for } A \neq 1 \\
(Ap_iB) \wedge (Cp_iD) & \leftrightarrow (A \wedge C)p_i(B \wedge D) & \\
(Ap_iB) \wedge (Cp_jD) & \leftrightarrow (A \wedge (Cp_jD))p_i(B \wedge (Cp_jD)) & \text{for } i > j \\
(Ap_iB) \wedge (Cp_jD) & \leftrightarrow ((Ap_iB) \wedge C)p_j((Ap_iB) \wedge D) & \text{for } i < j \\
e_l \wedge A & \leftrightarrow E_l \wedge A & \\
A \wedge e_l & \leftrightarrow A \wedge E_l \quad \text{for } A \text{ not an extension variable} &
\end{array}$$

We denote by $\mathcal{E}^\wedge$ the set $\mathcal{E}$ extended by all instances of the above axioms (always for leveled A, B).

To show that our definition indeed computes the join of two OBDDs we shall prove the following 'truth conditions':

Lemma 7.1.8. *Given a leveled set of extension axioms $\mathcal{E}$ and A, B **o-**eDT formulas over $\mathcal{E}$, there are polynomial size **o-**eLDT proofs over $\mathcal{E}^\wedge$ of the following sequents:*

$$\begin{array}{lcl}
A \wedge B & \longrightarrow & A \\
A \wedge B & \longrightarrow & B \\
A, B & \longrightarrow & A \wedge B
\end{array}$$

Proof. Assume A has level i and B has level j. The proof will proceed by $\mathcal{E}$-induction on A and B where at each step we must consider the different cases for $i = j$, $i > j$ and $i < j$. For simplicity, we may omit some structural steps. The first two sequents have similar proofs hence we only present the proof for $A \wedge B \longrightarrow A$.

Base case: A or B is a Boolean.

$$\mathcal{E}^\wedge,\mathsf{w}\text{-}r\,\dfrac{0\,\dfrac{}{0 \longrightarrow}}{0 \wedge B \longrightarrow 0} \qquad \mathcal{E}^\wedge,\mathsf{w}\text{-}r\,\dfrac{0\,\dfrac{}{0 \longrightarrow}}{A \wedge 0 \longrightarrow A} \qquad \mathsf{w}\text{-}l\,\dfrac{1\,\dfrac{}{\longrightarrow 1}}{1 \wedge B \longrightarrow 1} \qquad \mathcal{E}^\wedge\,\dfrac{\mathsf{id}\,\dfrac{}{A \longrightarrow A}}{A \wedge 1 \longrightarrow A}$$

Induction step: for $i = j$ the proof is as follows

$$\mathcal{E}^{\wedge},\,\text{p-}l\;\dfrac{\text{p-}r\;\dfrac{IH}{A\wedge C\longrightarrow p_i,p_i,A}\quad \text{id}\;\dfrac{}{A\wedge C,p_i\longrightarrow p_i,B}}{A\wedge C\longrightarrow p_i,Ap_iB}\qquad \text{p-}l\;\dfrac{\text{id}\;\dfrac{}{p_i,B\wedge D\longrightarrow p_i,A}\quad \text{p-}r\;\dfrac{IH}{p_i,p_i,B\wedge D\longrightarrow B}}{p_i,B\wedge D\longrightarrow Ap_iB}}{(Ap_iB)\wedge(Cp_iD)\longrightarrow Ap_iB}$$

For $i>j$ the proof is as follows

$$\mathcal{E}^{\wedge},\,\text{p-}l\;\dfrac{\text{p-}r\;\dfrac{IH}{A\wedge(Cp_jD)\longrightarrow p_i,p_i,A}\quad \text{id}\;\dfrac{}{A\wedge(Cp_jD),p_i\longrightarrow p_i,B}}{A\wedge(Cp_jD)\longrightarrow p_i,Ap_iB}\qquad \text{p-}l\;\dfrac{\text{id}\;\dfrac{}{p_i,B\wedge(Cp_jD)\longrightarrow p_i,A}\quad \text{p-}r\;\dfrac{IH}{p_i,p_i,B\wedge(Cp_jD)\longrightarrow B}}{p_i,B\wedge(Cp_jD)\longrightarrow Ap_iB}}{(Ap_iB)\wedge(Cp_jD)\longrightarrow Ap_iB}$$

for $i<j$ the proof is as follows

$$\mathcal{E}^{\wedge},\,\text{p-}l\;\dfrac{IH}{(Ap_iB)\wedge C\longrightarrow p_j,Ap_iB}\qquad \dfrac{IH}{p_j,(Ap_iB)\wedge D\longrightarrow Ap_iB}}{(Ap_iB)\wedge(Cp_jD)\longrightarrow Ap_iB}$$

The cases $e_l\wedge A$ or $A\wedge e_l$ for e_l an extension variable follow by the extension axioms from $\mathcal{E}^{\wedge}$ and invoking induction hypothesis:

$$\mathcal{E}^{\wedge}\;\dfrac{IH}{E_l\wedge A\longrightarrow E_l}{e_l\wedge A\longrightarrow e_l}\qquad \mathcal{E}^{\wedge}\;\dfrac{IH}{A\wedge E_l\longrightarrow A}{A\wedge e_l\longrightarrow A}$$

The proofs for $A,B\longrightarrow A\wedge B$ are also routine and follow by $\mathcal{E}$-induction on A,B. Hence we only include the base case and one representative case of the induction step (for $i=j$).
Base case:

$$\text{w-}l,\text{w-}r\;\dfrac{0\;\dfrac{}{0\longrightarrow}}{0,B\longrightarrow 0\wedge B}\qquad \text{w-}l,\text{w-}r\;\dfrac{0\;\dfrac{}{0\longrightarrow}}{A,0\longrightarrow A\wedge 0}\qquad \text{w-}l,\mathcal{E}^{\wedge}\;\dfrac{\text{id}\;\dfrac{}{B\longrightarrow B}}{1,B\longrightarrow 1\wedge B}\qquad \text{w-}l,\mathcal{E}^{\wedge}\;\dfrac{\text{id}\;\dfrac{}{A\longrightarrow A}}{A,1\longrightarrow A\wedge 1}$$

Induction step: for $i=j$ the proof is as follows (presented linearly for space reasons):

1. $C,A\longrightarrow p_i,A\wedge C$ (IH, w-r)
2. $p_i,D\longrightarrow p_i,A\wedge C$ (id, w-l, w-r)
3. $Cp_iD,A\longrightarrow p_i,A\wedge C$ (p_i-l on 1.+2.)
4. $Cp_iD,p_i,B\longrightarrow p_i,A\wedge C$ (id, w-l, w-r)
5. $Ap_iB,Cp_iD\longrightarrow p_i,A\wedge C$ (p_i-l on 3.+4.)
6. $p_i,A,Cp_iD\longrightarrow p_i,B\wedge D$ (id, w-l, w-r)
7. $p_i,p_i,B,C\longrightarrow p_i,B\wedge D$ (id, w-l, w-r)
8. $p_i,p_i,B,p_i,D\longrightarrow B\wedge D$ (IH, w-l)
9. $p_i,p_i,B,Cp_iD\longrightarrow B\wedge D$ (p_i-l on 7.+8. and w-r)
10. $p_i,Ap_iB,Cp_iD\longrightarrow B\wedge D$ (p_i-l on 9.+6.)
11. $Ap_iB,Cp_iD\longrightarrow(Ap_iB)\wedge(Cp_iD)$ (p_i-$r,\mathcal{E}^{\wedge}$ on 10.+5.)

for $i > j$ and $i < j$ the proofs are simpler and they are similar in nature so we only include the case for $i > j$:

1. $A, Cp_jD \longrightarrow p_i, p_i, A \wedge (Cp_jD)$ $(IH, \mathsf{w}\text{-}r)$
2. $p_i, B, Cp_jD \longrightarrow p_i, A \wedge (Cp_jD)$ $(\mathsf{id}, \mathsf{w}\text{-}l, \mathsf{w}\text{-}r)$
3. $Ap_iB, Cp_jD \longrightarrow p_i, A \wedge Cp_jD$ $(p_i\text{-}l$ on 1.+2.$)$
4. $A, Cp_jD, p_i \longrightarrow p_i, B \wedge (Cp_jD)$ $(\mathsf{id}, \mathsf{w}\text{-}l, \mathsf{w}\text{-}r)$
5. $p_i, B, Cp_jD, p_i, \longrightarrow B \wedge (Cp_jD)$ $(IH, \mathsf{w}\text{-}l)$
6. $Ap_iB, Cp_jD, p_i \longrightarrow B \wedge (Cp_jD)$ $(p_i\text{-}l$ on 4.+5.$)$
7. $Ap_iB, Cp_jD \longrightarrow (Ap_iB) \wedge (Cp_jD)$ $(p_i\text{-}r, \mathcal{E}^\wedge$ on 3.+6.$)$

The cases $e_l \wedge A$ or $A \wedge e_l$ for e_l an extension variable follow by using the extension axioms from $\mathcal{E}^\wedge$ and invoking induction hypothesis:

$$\mathcal{E}^\wedge \dfrac{\dfrac{IH}{E_l, A \longrightarrow E_l \wedge A}}{e_l, A \longrightarrow e_l \wedge A} \qquad \mathcal{E}^\wedge \dfrac{\dfrac{IH}{A, E_l \longrightarrow A \wedge E_l}}{A, e_l \longrightarrow A \wedge e_l}$$

□

Remark 7.1.9. We note that the construction given in Definition 7.1.7 is general enough to define a multitude of connectives. This is done by simply changing the axioms that determine the interactions with leaves and letting connectives be distributive over decisions as before. For example, for an arbitrary connective $\star$, one could set:

$$\begin{array}{rll}
A \star 0 & \leftrightarrow A \quad 0 \star A \leftrightarrow A & \text{for } A \neq 0 \\
A \star 1 & \leftrightarrow 1 \quad 1 \star A \leftrightarrow 1 & \text{for } A \neq 1 \\
(Ap_iB) \star (Cp_iD) & \leftrightarrow (A \star C)p_i(B \star D) & \\
(Ap_iB) \star (Cp_jD) & \leftrightarrow (A \star (Cp_jD))p_i(B \star (Cp_jD)) & \text{for } i > j \\
(Ap_iB) \star (Cp_jD) & \leftrightarrow ((Ap_iB) \star C)p_j((Ap_iB) \star D) & \text{for } i < j \\
e_l \star A & \leftrightarrow E_l \star A & \\
A \star e_l & \leftrightarrow A \star E_l & \text{for } A \text{ not an extension variable}
\end{array}$$

Then, proving the appropriate 'truth conditions' for $\star$ would show it behaves as the disjunction ($\vee$) of two OBDDs. In a similar manner, one could define extension axioms for other connectives, for example $\star$ being 'exclusive-Or' ($\oplus$).

7.2 Comparison to OBDD($\wedge$, w)

One of the main goals of this chapter is to compare the strength of our system **o-**eLDT to OBDD($\wedge$, w) from [9]. Since this is a simple refutation system, we will need to accordingly translate refutations in OBDD($\wedge$, w) into proofs in **o-**eLDT. Let us now take a slightly informal look into bridging the syntactic differences between OBDD($\wedge$, w) refutations and **o-**eLDT proofs.

In [9] and [23] they rely on the unique representation of Boolean functions by minimal size OBDDs computing them which, can be achieved by appealing to the reduction

rules from Figure 7.1. Their semantic approach means that they can freely speak about canonically representing clauses in a CNF $\mathcal{C}$ by OBDDs without any further syntactic technicalities. We on the other hand, have to impose additional constraints on our formulas to make sure they represent clauses canonically. If we aimed for minimality of size of our **o-**eDT formulas, we could require our leveled sets of extension axioms are structured accordingly to the reduction rules. Consider for example the elimination rule, then for a leveled set of extension axioms $\mathcal{E}$ we could eliminate all extension axioms of the form $e^i_j \leftrightarrow e^{i'}_{j'} p_i e^{i'}_{j'}$ for $j' < i$ and then substitute e^i_j by $e^{i'}_{j'}$ everywhere in $\mathcal{E}$. Restricting $\mathcal{E}$ also w.r.t. the merging rule, would ensure we only express reduced OBDDs with **o-**eDT formulas over $\mathcal{E}$. This, while feasible, would unnecessarily introduce additional complexity in our definitions and arguments. Instead, we opt for achieving uniqueness of representation of Boolean functions via **o-**eDT formulas without minimality of size. This is done by the simple trick seen in Remark 7.1.4: we consider literals in a clause of $\mathcal{C}$ to be ordered according to r and then construct the appropriate **o-**eDT formulas representing the respective functions computed by said clause. In what comes next, we show that our system **o-**eLDT polynomially simulates $\mathsf{OBDD}(\wedge, \mathsf{w})$.

Let us first provide a formal definition of the system $\mathsf{OBDD}(\wedge, \mathsf{w})$, adapted into our setting[4]. The difference is that, we represent OBDDs in the refutation by using **o-**eDT formulas and the join of two OBDDs is given by the construction of $\wedge$ over $\mathcal{E}^\wedge$ (Definition 7.1.7).

Definition 7.2.1 ($\mathsf{OBDD}(\wedge, \mathsf{w})$ [9])**.** An $\mathsf{OBDD}(\wedge, \mathsf{w})$ derivation D from a CNF $\mathcal{C} = \bigwedge_s C_s$ is a sequence of **o-**eDT formulas $R_1, \ldots, R_t$ such that: $C^o_1, \ldots, C^o_s$ (the **o-**eDT formulas corresponding to the clauses $C_1, \ldots, C_s$) are defined over a leveled set $\mathcal{E}$, each R_j is defined over the leveled set of extension axioms $\mathcal{E}^\wedge$ extending $\mathcal{E}$ and each R_j either represents an OBDD computing some clause C_j of $\mathcal{C}$ (i.e. R_j is C^o_j) or is obtained by the following rules:

- **Join**: R_j is the extension variable $R_k \wedge R_l$ (in $\mathcal{E}^\wedge$) for $k, l < j$ i.e. R_k, R_l appear earlier in the refutation.
- **Weakening**: there exists some $k < j$ such that $R_k \models_{\mathcal{E}} R_j$.

The derivation concludes with R_t, the length of D is t and the size $|D|$ is the number of the symbols in it, i.e. the sum: $|R_1| + \ldots + |R_t|$. Naturally, the set $\mathcal{E}^\wedge$ is adequate to reason about joins since it contains all axioms of the form given in Definition 7.1.7, appropriately extending the leveled set of extension axioms $\mathcal{E}$. An $\mathsf{OBDD}(\wedge, \mathsf{w})$ **refutation** R from $\mathcal{C}$ is an $\mathsf{OBDD}(\wedge, \mathsf{w})$ derivation where R_t is 0.

Remark 7.2.2. In [23] they also consider an extension of $\mathsf{OBDD}(\wedge, \mathsf{w})$, the system $\mathsf{OBDD}(\wedge, \mathsf{w}, \mathsf{ord})$ where a further rule 'reordering' is permitted that exponentially increases the expressive strength of the system.

- **Reordering**: R_i is an $\mathsf{r_i}$-OBDD that is semantically equivalent to an $\mathsf{r_j}$-OBDD R_j with $j < i$.

[4]Additionally, akin to [38], we do not consider the rule 'projection' i.e. elimination of a variable but treat it as a special case of the join $(C \vee p) \wedge (D \vee \neg p)$ applied before a 'weakening' rule deriving $C \vee D$.

We will however not concern ourselves with systems permitting reordering for the time being and solely focus on simulating $\mathsf{OBDD}(\wedge, \mathsf{w})$.

The main result of this section, polynomial simulation of $\mathsf{OBDD}(\wedge, \mathsf{w})$ by **o-**eLDT then follows as a corollary of:

Theorem 7.2.3. *Given an* $\mathsf{OBDD}(\wedge, \mathsf{w})$ *derivation* D *over a leveled set of extension axioms* $\mathcal{E}^\wedge$ *of some formula* F *from some CNF* $\mathcal{C} = \bigwedge_{s\in S} C_s$*, there is an (dag-like)* ***o-***eLDT *proof* P_D *of size polynomial in* $|D|$ *over* $\mathcal{E}^\wedge$ *for the sequent* $\{C_s^o\}_{s\in S} \longrightarrow F$*.*

Proof. The proof follows by induction on the structure of D.

- If $F = F_1 \wedge F_2$ and it is obtained by the join rule:

$$\wedge \dfrac{\dfrac{\vdots}{F_1} \quad \dfrac{\vdots}{F_2}}{F_1 \wedge F_2}$$

  this will be simulated by two cut steps and an application of Lemma 7.1.8:

$$\mathsf{2cut}\ \dfrac{\dfrac{IH}{\{C_s^o\}_{s\in S} \longrightarrow F_1} \quad \dfrac{IH}{\{C_s^o\}_{s\in S} \longrightarrow F_2} \quad \dfrac{\text{Lemma 7.1.8}}{F_1, F_2 \longrightarrow F_1 \wedge F_2}}{\{C_s^o\}_{s\in S} \longrightarrow F_1 \wedge F_2}$$

- If instead F is implied by some previous formula, i.e. $F' \models_{\mathcal{E}^\wedge} F$ then the weakening rule:

$$\models_{\mathcal{E}^\wedge} \dfrac{\dfrac{\vdots}{F'}}{F}$$

  will be simulated by a cut step and an application of Proposition 7.1.6:

$$\mathsf{cut}\ \dfrac{\dfrac{IH}{\{C_s^o\}_{s\in S} \longrightarrow F'} \quad \dfrac{\text{Proposition 7.1.6}}{F' \longrightarrow F}}{\{C_s^o\}_{s\in S} \longrightarrow F}$$

- Lastly, if F is in $\{C_s^o\}_{s\in S}$, we simply end the proof by a left weakening step and an identity step:

$$\mathsf{w}\text{-}l\ \dfrac{\mathsf{id}\ \dfrac{}{F \longrightarrow F}}{\{C_s^o\}_{s\in S} \longrightarrow F}$$

□

Corollary 7.2.4. Given an $\mathsf{OBDD}(\wedge, \mathsf{w})$ refutation $R = R_1, \ldots, R_t$ of size n over a leveled set of extension axioms $\mathcal{E}^\wedge$ for some unsatisfiable CNF $\mathcal{C} = \bigwedge_{s\in S} C_s$, there is an (dag-like) **o-**eLDT proof P_R of size polynomial in n over $\mathcal{E}^\wedge$ for the sequent $\{C_s^o\} \longrightarrow$.

Proof. Follows by Theorem 7.2.3 for $F = 0$ and a cut step. □

Chapter 8

Summary and Future Work

8.1 Summary

We will now summarize the results presented in this thesis. There are three contributions: first, the introduction of eLNDT^+, the positive fragment of eLNDT that reasons about positive branching programs and the polynomial simulation of eLNDT by eLNDT^+ on positive sequents. Second, the construction of Prover-Adversary games in the style of Pudlák and Buss [52] that, similarly to how the original game corresponds to Hilbert-Frege/LK systems, correspond to systems of branching programs (and fragments thereof). Lastly, the explicit definition of a fragment of eLDT that reasons about OBDDs called **o-**eLDT and the polynomial simulation of a previously introduced system for OBDDs, $\mathsf{OBDD}(\wedge, \mathsf{w})$ [9] by **o-**eLDT.

8.1.1 Simulation of eLNDT by eLNDT^+.

The first part of this thesis (Chapters 3-5) culminated in showing that eLNDT is polynomially simulated by its positive fragment eLNDT^+ on positive sequents. Based on work of Buss, Das, Knop [22], Chapter 2 (re)introduced the necessary preliminary material. Starting with Section 2.1, we defined the systems $\mathsf{eLDT}, \mathsf{eLNDT}$ reasoning about deterministic and non-deterministic branching programs respectively. Next, Section 2.2 focused on monotone computation and reasoning, first monotone functions and monotone branching programs are defined and later the system eLNDT^+, the positive fragment of eLNDT is presented. eLNDT^+ reasons about positive branching programs, NBPs which at the level of the graph, for each 0-edge have a 1-edge parallel to it. This section finished with basic results for eLNDT^+ like generalised identity and truth conditions for the positive decision connective. Next is Chapter 3 where the Exact and Threshold Boolean functions are introduced alongside deterministic branching programs and their positive closures computing them respectively and later, we provided their representations as eDT, eNDT formulas w.r.t. appropriate sets of extension axioms. Chapter 4 contains the first main result of this work: a case study of the expressive power of eLNDT^+ by finding polynomial size proofs of the propositional pigeonhole principle. The statement is first adjusted to fit our setting i.e. it is expressed as a sequent where conjunctions are substituted by specific (logically equivalent) positive decisions. We continued by introducing important technical tools in the form of lemmas and propositions that allow

us to manipulate eNDT formulas computing threshold functions. The 'proof' is split into three separate parts which when 'cut' together result in the proof of the pigeonhole principle. In Chapter 5 we provided the polynomial simulation of eLNDT by eLNDT^+ on positive sequents. While the high-level structure of our argument is similar to [8], we had to tackle specific issues arising due to the peculiarities (using extension) of our setting and for that, we split our strategy into three parts. First, in Section 5.2, we defined eLNDT^+_-, a system intermediating eLNDT and eLNDT^+ where, while all decisions must be in positive form, negative literals are permitted (also as decision literals). We provided a polynomial-time translation from eNDT to eNDT^+_- formulas and a polynomial simulation of eLNDT by eLNDT^+_-. Afterwards, in Sections 5.3 and 5.4, we showed how to substitute negated literals in an eLNDT^+_- proof for our positive threshold formulas. Since making decisions on complex formulas is not permitted in our setting, we had to carefully craft new sets of extension variables and axioms for which we provided the necessary 'truth conditions' that guarantee they behave appropriately. Then, we defined $\{\mathsf{eLNDT}^+_k(P)\}_{k\geqslant 0}$ a family of proof systems, each of which extends eLNDT^+ by two initial sequents and show polynomial equivalence of eLNDT^+_- by $\mathsf{eLNDT}^+_k(P)$ for each $k \geqslant 0$. This chapter ends with Section 5.5 where the proof of the polynomial simulation of eLNDT by eLNDT^+ is put together by assembling results from the previous sections.

8.1.2 Games for Non-deterministic Branching Programs

Inspired by the work of Pudlák and Buss [52], where they defined Prover-Adversary games corresponding to Hilbert-Frege/LK, the second part of this thesis (Chapter 6) introduced specifically tailored Prover-Adversary games for $\mathsf{eLDT}, \mathsf{eLNDT}$ and fragments thereof. In Section 6.1 we presented preliminary material: definitions, basic results and examples to familiarise the reader with Prover-Adversary games. Besides the original games where queries are Boolean formulas from [52], we also provided a more general version admitting arbitrary queries and simple contradiction sets. In this way, the original game can be seen as an instance of the more abstract framework. In Section 6.2 we discussed and solved the issue of equivalent representation of (N)BPs via formulas with extension. In the deterministic setting, two BPs are bisimilar if and only if their respective eDT formulas have the same unfolding into binary decision trees. This is not the same in the non-deterministic case though. There, we needed to define a notion of simulation between eNDT formulas parametrised by the set of extension axioms they are defined over. We proceeded to show that our notions of equivalence between two e(N)DT formulas A, B translate to short proofs of the sequents $A \leftrightarrow B$. We later used this fact to justify our sets of simple contradictions when we formally presented our games. In Section 6.3 we discussed a design choice we made. That is, our games admit queries that are Boolean combinations of e(N)DT formulas making them expressively powerful tools to reason about $\mathsf{eL(N)DT}$. The only difficulty arose in the non-deterministic setting because negations of NBPs are non-trivial to define. Next, in Section 6.4 we defined the games DB and NB that correspond to eLDT and eLNDT in the same manner the original game from [52] corresponds to LK: for every $\mathsf{eLDT}, \mathsf{eLNDT}$ proof of a sequent $\Gamma \longrightarrow \Delta$ of size N there is a $O(\log N)$-round strategy for $\Gamma \longrightarrow \Delta$ in DB, NB respectively and vice versa. This is the main result of this chapter

and the 'forward' direction is shown in Section 6.5. Besides some technicalities, this is easier than the converse and it mostly follows the same idea from [41] where dag-like Hilbert-Frege proofs are transformed into tree-form with only polynomial increase in size. The other direction i.e. from a given strategy obtain the corresponding proof, is more involved and requires a non-uniform formalisation of the Immerman-Szelepcsényi theorem, that $\mathbf{NL} = co\mathbf{NL}$, to be able to simulate negations of eNDT formulas. This is detailed in Section 6.6 where we first defined a framework that allows us to work with positive decisions on complex formulas. As mentioned before, this is not allowed in our setting and thus we had to carefully craft appropriate sets of extension variables and axioms (much like in Section 5.4). The idea is that positive decisions on complex formulas can be used to construct the 'decider', an NBP (parametrised by multiple indices) that can simulate a general decision on complex formulas. After proving some natural properties of the decider, we moved on to Section 6.7 which started by defining the game $\mathsf{NB}^{\mathrm{DM}}$. This is a fragment of NB whose queries are in De Morgan normal form (negations only in front of eNDT formulas) and after some more technical results we showed that $\mathsf{NB}^{\mathrm{DM}}$ simulates NB with only a logarithmic increase in the number of rounds in a strategy. Next, we showed that from a strategy of $\mathsf{NB}^{\mathrm{DM}}$ of size N we can obtain a proof of $\mathrm{Bool}^{+}(\mathsf{eLNDT})$ (an extension of eLNDT allowing $\neg$-free Boolean combinations of eNDT formulas) with polynomial increase in size. We ended with providing a polynomial simulation of $\mathrm{Bool}^{+}(\mathsf{eLNDT})$ by eLNDT which ends the 'chain' of simulations and provides the converse direction of the main result, Theorem 6.7.1, of this Chapter.

8.1.3 Systems for OBDDs

Ordered binary decision diagrams (OBDDs) are deterministic BPs whose paths contain variables ordered w.r.t. some fixed order. Under the restriction of a fixed order, OBDDs of minimal size can canonically represent Boolean functions, a desirable feature that further makes for easy checking of some of properties and relations between OBDDs. They were given a proof theoretic treatment in [9] as a case study of proof systems corresponding to constraint satisfaction problems. Multiple papers expanded on their results including [38] and [23] introducing new systems extending the ones described in [9] and comparing their relative strengths w.r.t. proof complexity.

It was a natural direction for this project to define **o-**eLDT, the fragment of eLDT that canonically reasons about OBDDs similar to how Hilbert-Frege and LK reason about Boolean formulas, and compare its expressive strength to the system $\mathsf{OBDD}(\wedge, \mathsf{w})$ from [9]. While not very hard, there were certain technicalities we needed to address to restrict eLDT appropriately and obtain its OBDD fragment, **o-**eLDT. We started Section 7.1 by defining the notion of 'level' ensuring that our sets of extension axioms and formulas defined over them only represent OBDDs under a fixed order r. We proceeded to define the system **o-**eLDT ensuring that for each OBDD there is an **o-**eDT formula over an appropriate set of extension axioms representing it. The system $\mathsf{OBDD}(\wedge, \mathsf{w})$ uses the following rules: the 'join' $A \wedge B$ of two OBDDs and the 'weakening' where an OBDD B can be obtained by a previous one A, if A logically implies B. We adapted the definition of $\mathsf{OBDD}(\wedge, \mathsf{w})$ to fit our setting and laid out the technical framework needed to simulate the rules 'join' and 'weakening' in **o-**eLDT. More specifically, in

Section 7.1.1 we introduced a judgement $\models_{\mathcal{E}}$ between **o-**eDT formulas that, together with its 'truth conditions', is meant to simulate the weakening rule. Similarly, in Section 7.1.2, we designed appropriate extension axiom sets which will be used to simulate the join rule by essentially introducing a fresh extension variable $A \wedge B$ that behaves like the conjunction of the **o-**eDT formulas A, B. Finally, in Section 7.2, we proved that **o-**eLDT polynomially simulates OBDD$(\wedge, \mathsf{w})$: given a refutation R in OBDD$(\wedge, \mathsf{w})$ of size n of some CNF $\mathcal{C}$, we can find an **o-**eLDT proof of $\mathcal{C} \longrightarrow$ of size polynomial in n, Theorem 7.2.3.

8.2 Future Work

There are multiple future directions one could follow with respect to this project. Literature pertaining to branching programs is rich and many of their variants (beside OBDDs) have seen research interest and had their properties studied. For example, we could investigate read-once branching programs (ROBPs) which were introduced by Masek in [46], a decade before OBDDs . They are non-deterministic branching programs whose paths only contains uniquely labelled nodes that is, for each i, a variable p_i may label at most one node in each path. Defining the fragment of eLNDT that reasons about ROBPs would require constructing appropriate sets of extension axioms that only represent ROBPs, akin to what we do in Definitions 7.1.1 and 7.1.2 for OBDDs.

A direct expansion of current results from *Chapter* 7, would be to strengthen Theorem 7.2.3 by checking whether **o-**eLDT is a strictly stronger system than OBDD$(\wedge, \mathsf{w})$. On that end, it would be interesting to see if we can follow the approach of [23] and study Clique-Coloring tautologies expressed via **o-**eDT formulas. Proving that there are polynomial-size **o-**eLDT proofs of Clique-Coloring tautologies but only superpolynomial-size proofs of the same tautologies in OBDD$(\wedge, \mathsf{w})$ (the slightly altered version from Definition 7.2.1) would show that **o-**eLDT is separated from OBDD$(\wedge, \mathsf{w})$.

An alternative and viable future-research direction would be to reason about branching programs by, instead of finding proofs in the calculus-like systems eLDT, eLNDT, drawing winning strategies in the Prover-Adversary games defined in Chapter 6. The purpose of working with games is to have an expressively rich way to talk about eLDT, eLNDT and their fragments while avoiding the complexity of the syntax based on extension. This thesis develops the necessary framework for games for NBPs, but does not include a complete account of results being obtained through the game systems DB, NB. For example, all results in Chapters 3, 4 and 5 could be rewritten in the context of games and furthermore, systems reasoning about different fragments of eLNDT could be fully expressed via games.

Bibliography

[1] https://plato.stanford.edu/entries/proof-theory/.

[2] http://alessio.guglielmi.name/res/cos/.

[3] Miklós Ajtai. The complexity of the pigeonhole principle. *Combinatorica*, 14:417–433, 1994.

[4] Miklós Ajtai, János Komlós, and Endre Szemerédi. An o(n log n) sorting network. In *15th Annual ACM 25-27 April, 1983*, pages 1–9. ACM, 1983. doi:10.1145/800061.808726.

[5] Noga Alon and Ravi B Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7:1–22, 1987.

[6] Aleksandr Egorovich Andreev. A method for obtaining lower bounds on the complexity of individual monotone functions. In *Doklady Akademii Nauk*, volume 282, pages 1033–1037. Russian Academy of Sciences, 1985.

[7] Albert Atserias, Nicola Galesi, and Ricard Gavaldá. Monotone proofs of the pigeon hole principle. *Mathematical Logic Quarterly: Mathematical Logic Quarterly*, 47(4):461–474, 2001.

[8] Albert Atserias, Nicola Galesi, and Pavel Pudlák. Monotone simulations of non-monotone proofs+. *J. Comput. Syst. Sci.*, 65(4):626–638, 2002.

[9] Albert Atserias, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint propagation as a proof system. In Mark Wallace, editor, *10th CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2004. doi:10.1007/978-3-540-30201-8_9.

[10] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[11] Chris Barrett and Alessio Guglielmi. A subatomic proof system for decision trees. *ACM Transactions on Computational Logic*, 23(4):1–25, 2022.

[12] Paul Beame, Russell Impagliazzo, Jan Krajícek, Toniann Pitassi, Pavel Pudlák, and Alan R. Woods. Exponential lower bounds for the pigeonhole principle. In *24th ACM Symposium*, pages 200–220. ACM, 1992. doi:10.1145/129712.129733.

[13] Stephen Bellantoni, Toniann Pitassi, and Alasdair Urquhart. Approximation and small-depth frege proofs. *SIAM Journal on Computing*, 21(6):1161–1179, 1992.

[14] Maria Luisa Bonet and Samuel R Buss. Size-depth tradeoffs for boolean formulae. *Information Processing Letters*, 49(3):151–155, 1994.

[15] Richard P Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM (JACM)*, 21(2):201–206, 1974.

[16] Randal E Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[17] Randal E Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

[18] Samuel R Buss. *Bounded arithmetic*. Princeton University, 1985.

[19] Samuel R. Buss. The boolean formula value problem is in ALOGTIME. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 123–131. ACM, 1987. doi:10.1145/28395.28409.

[20] Samuel R Buss. Polynomial size proofs of the propositional pigeonhole principle. *The Journal of Symbolic Logic*, 52(4):916–927, 1987.

[21] Samuel R Buss. *Handbook of proof theory*. Elsevier, 1998.

[22] Samuel R. Buss, Anupam Das, and Alexander Knop. Proof complexity of systems of (non-deterministic) decision trees and branching programs. In *Proceedings of CSL*, 2020.

[23] Samuel R. Buss, Dmitry Itsykson, Alexander Knop, and Dmitry Sokolov. Reordering rule makes OBDD proof systems stronger. In Rocco A. Servedio, editor, *33rd CCC 2018, June 22-24*, volume 102 of *LIPIcs*, pages 16:1–16:24, 2018. doi:10.4230/LIPIcs.CCC.2018.16.

[24] Samuel R. Buss, Valentine Kabanets, Antonina Kolokolova, and Michal Koucký. Expander construction in VNC1. In *8th ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 31:1–31:26, 2017. doi:10.4230/LIPIcs.ITCS.2017.31.

[25] Paul J Cohen. The independence of the continuum hypothesis. *Proceedings of the National Academy of Sciences*, 50(6):1143–1148, 1963.

[26] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[27] Stephen A. Cook. A survey of complexity classes and their associated propositional proof systems and theories and a proof system for log space, 2001. Slides for Edinburgh talk.http://www.cs.toronto.edu/~sacook/.

[28] Stephen A. Cook and Pierre McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8(3):385–394, 1987.

[29] Stephen A. Cook and Phuong Nguyen. *Logical foundations of proof complexity*, volume 11. Cambridge University Press Cambridge, 2010.

[30] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *J. Symb. Log.*, 44(1):36–50, 1979.

[31] Anupam Das and Avgerinos Delkos. Proof complexity of monotone branching programs. In Ulrich Berger, Johanna N. Y. Franklin, Florin Manea, and Arno Pauly, editors, *Revolutions and Revelations in Computability - 18th Conference on Computability in Europe, CiE 2022, Swansea, UK, July 11-15, 2022, Proceedings*, volume 13359 of *Lecture Notes in Computer Science*, pages 74–87. Springer, 2022. doi:10.1007/978-3-031-08740-0_7.

[32] Michelangelo Grigni. *Structure in monotone complexity*. PhD thesis, 1991. URL: http://www.mathcs.emory.edu/~mic/papers/thesis.ps.gz.

[33] Michelangelo Grigni and Michael Sipser. Monotone complexity. In *Proceedings of the London Mathematical Society Symposium on Boolean Function Complexity*, page 57–75, USA, 1992. Cambridge University Press.

[34] Armin Haken. The intractability of resolution. *Theoretical computer science*, 39:297–308, 1985.

[35] Armin Haken and Stephen A. Cook. An exponential lower bound for the size of monotone real circuits. *Journal of Computer and System Sciences*, 58(2):326–335, 1999.

[36] Johan Håstad. *Computational limitations for small depth circuits*. PhD thesis, Massachusetts Institute of Technology, 1986.

[37] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on computing*, 17(5):935–938, 1988.

[38] Dmitry Itsykson, Alexander Knop, Andrey Romashchenko, and Dmitry 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)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[39] Emil Jerábek. A sorting network in bounded arithmetic. *Ann. Pure Appl. Logic*, 162(4):341–355, 2011. doi:10.1016/j.apal.2010.10.002.

[40] M. Karchmer and A. Wigderson. On span programs. In *Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 102–111, 1993. doi:10.1109/SCT.1993.336536.

[41] Jan Krajíček. Lower bounds to the size of constant-depth propositional proofs. *The Journal of Symbolic Logic*, 59(1):73–86, 1994. URL: https://doi.org/10.2307/2275250.

[42] Jan Krajíček. *Proof complexity*, volume 170. Cambridge University Press, 2019.

[43] Jan Krajicek et al. *Bounded arithmetic, propositional logic and complexity theory*, volume 60. Cambridge University Press, 1995.

[44] Jan Krajíček. *Proof Complexity*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2019. doi:10.1017/9781108242066.

[45] L. Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9:115–116, 1973.

[46] William Joseph Masek. *A fast algorithm for the string editing problem and decision graph complexity*. PhD thesis, Massachusetts Institute of Technology, 1976.

[47] Christoph Meinel and Anna Slobodova. On the complexity of constructing optimal ordered binary decision diagrams. In *International Symposium on Mathematical Foundations of Computer Science*, pages 515–524. Springer, 1994.

[48] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[49] Michael S Paterson. *An introduction to Boolean function complexity*. Computer Science Department, School of Humanities and Sciences, Stanford ..., 1976.

[50] Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. 62(3):981–998, 1997.

[51] Pavel Pudlak. Proofs as games. *The American Mathematical Monthly*, 107(6):541–550, 2000.

[52] Pavel Pudlák and Samuel R. Buss. How to lie without being (easily) convicted and the length of proofs in propositional calculus. In *Proceedings of CSL*, 1994. doi:10.1007/BFb0022253.

[53] Alexander A Razborov. Lower bounds for the monotone complexity of some boolean functions. In *Soviet Math. Dokl.*, volume 31, pages 354–357, 1985.

[54] Alexander A Razborov. Lower bounds on monotone complexity of the logical permanent. *Mathematical Notes of the Academy of Sciences of the USSR*, 37(6):485–493, 1985.

[55] Alexander A Razborov. Proof complexity of pigeonhole principles. In *Developments in Language Theory: 5th International Conference, DLT 2001 Wien, Austria, July 16–21, 2001 Revised Papers 5*, pages 100–116. Springer, 2002.

[56] Michael Sipser. Introduction to the theory of computation. 2021.

[57] Philip Spira. On time-hardware complexity tradeoffs for boolean functions. In *Proceedings of the 4th Hawaii Symposium on System Sciences, 1971.*

[58] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988. doi:10.1007/BF00299636.

[59] Grigori S Tseitin. On the complexity of derivation in propositional calculus. *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pages 466–483, 1983.

[60] Andrea Aler Tubella and Alessio Guglielmi. Subatomic proof systems: Splittable systems. *ACM Transactions on Computational Logic (TOCL)*, 19(1):1–33, 2018.

[61] Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer Science & Business Media, 1999.

[62] Ingo Wegener. *The complexity of Boolean functions*. John Wiley & Sons, Inc., 1987.

[63] Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000. URL: `http://ls2-www.cs.uni-dortmund.de/monographs/bdd/`.

[64] Avi Wigderson. *Mathematics and computation: A theory revolutionizing technology and science*. Princeton University Press, 2019.