$\mu \prod \lambda \forall$

Graduate Program in Logic, Algorithms and Computation

Proof Complexity: A Tableau Perspective

Master Thesis of:
Theodoros Papamakarios

July 2017

Abstract

The method of semantic tableaux (or simply tableaux) is arguably one of the most elegant proof systems. Unfortunately, it hasn't received much attention in the proof complexity literature, mainly due to early negative results, concerning the complexity of cut-free tableaux (see for example [15, 17, 18]). We bring tableaux to the fore, introducing the measures of tableau depth and width. Equipped with these, we show in an elegant, uniform way several known results spanning proof complexity, from a tableau viewpoint.

Contents

List of Figures

1

Introduction

With the rise of computer science, questions like "Can we solve a problem?" got a quantitative counterpart: "How hard is it to solve a problem?". Proof complexity deals with the quantitative version of "Can we prove a theorem?", namely, the question "How hard is it to prove a theorem?". In this thesis we study the complexity of propositional proofs in various proof systems. Despite the intrinsic interest of the problem, the main motivations come from computational complexity, automated theorem proving and bounded arithmetic. We will elaborate later on the first two. We won't touch relations with bounded arithmetic; let us just mention here that lower bounds on the complexity of propositional proofs yield independence results in weak subsystems of Peano arithmetic; we refer the interested reader to [25, 14].

1.1 What is a proof?

A proof is an object certifying the validity of a statement. Since at least Euclid's Elements, proofs are seen as rigorous arguments, formalized in an axiomatic system. But the notion is way broader. For example, if X is a propositional formula, a proof of the statement "X is satisfiable" is an assignment satisfying X. On the other hand, if after running an algorithm for satisfiability on X, it determines that X is not satisfiable, then the trace of its run is a proof of the statement "X is unsatisfiable".

An obvious requirement that a proof must meet, is to be easily verifiable, where "easily" means "in polynomial time". As formalized by Cook and Reckhow in [15, 16], proofs are encoded as finite strings, and a proof system for a class L of statements is a polynomial-time algorithm that takes a finite string y and

1. checks if y is a valid encoding of a proof in the system; if it is not, returns a prespecified statement $T \in L$;

2. returns the statement which y proves.

Definition 1.1.1. Fix an alphabet Σ. A *proof system* for a language $L \subseteq \Sigma^*$ is a function $f\colon \Sigma^* \to L$, where f is computable in polynomial time and f is onto. A proof system f is *polynomially bounded* if there is a polynomial $p(n)$ such that for all $x \in L$, there is a $y \in \Sigma^*$ such that $x = f(y)$ and $|y| \leq p(|x|)$.

The condition "f is onto" refers to *completeness*; every statement must have a proof in the system.

The class NP is by definition the class of all languages that have short (polynomial-size) proofs. We thus have Theorem 1.1.1. Let us note that this theorem is the polynomially bounded analogue of the statement in computability saying that L is recursively enumerable if and only if $L = \emptyset$ or L is the range of a recursive function.

Theorem 1.1.1 (Cook and Reckhow [15, 16])**.** *For any language $L \subseteq \Sigma^*$, $L \in \mathsf{NP}$ if and only if $L = \emptyset$ or L has a polynomially bounded proof system.*

Proof. Suppose that $L \in \mathsf{NP}$ and $L \neq \emptyset$. Since $L \neq \emptyset$, there is a string $t \in L$. Since $L \in \mathsf{NP}$, there is a predicate $R \subseteq \Sigma^* \times \Sigma^*$ computable in polynomial time and a polynomial $p(n)$ such that for each $x \in \Sigma^*$

$$x \in L \iff \exists y \in \Sigma^*(|y| \leq p(|x|) \ \ \& \ \ R(x, y)).$$

The function $f\colon \Sigma^* \to L$, where

$$f(z) = \begin{cases} x, & \text{if } z = (x, y) \ \ \& \ \ |y| \leq p(|x|) \ \ \& \ \ R(x, y), \\ t, & \text{otherwise.} \end{cases}$$

is a polynomially bounded proof system for L.

Conversely, obviously $\emptyset \in \mathsf{NP}$, and if f is a polynomially bounded proof system with the bound $p(n)$, then for each $x \in \Sigma^*$

$$x \in L \iff \exists y \in \Sigma^*(|y| \leq p(|x|) \ \ \& \ \ f(y) = x).$$

This is an NP definition of L. □

We will be particularly interested in the set TAUT of all propositional tautologies. TAUT, being the complement of an NP-complete set, is coNP-complete, so from Theorem 1.1.1, we get the following.

Corollary 1.1.1. $\mathsf{NP} = \mathsf{coNP}$ *if and only if there is a polynomially bounded proof system for the set of propositional tautologies.*

Hence, showing superpolynomial lower bounds for stronger and stronger proof systems for propositional tautologies could be seen as an approach to the conjecture $\mathsf{NP} \neq \mathsf{coNP}$. Although this approach seems infeasible (until now there is no clue on how to prove non-trivial lower bounds for "simple" axiomatic proof systems), the hope is that eventually general techniques will be unveiled.

We close this section by defining a notion for comparing proof systems over Σ. The definition is from [16]

Definition 1.1.2. If $f_1 \colon \Sigma^* \to L$ and $f_2 \colon \Sigma^* \to L$ are proof systems for L, then f_2 *p-simulates* f_1 provided there is a polynomial-time computable function $g \colon \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$

$$f_2(g(x)) = f_1(x).$$

It is easy to see that the p-simulation relation is reflexive and transitive, so that its symmetric closure is an equivalence relation, with equivalence classes within which the systems are *"polynomially equivalent"*. Also, as an immediate consequence of the definitions, we have the following proposition.

Proposition 1.1.1. *If a proof system f_2 for L p-simulates a polynomially bounded proof system f_1, then f_2 is also polynomially bounded.*

1.2 Games

Sometimes, proofs can be perceived in terms of a game played by two players. Plato, a philosopher who, perhaps more than anyone influenced western thought, used to expound his arguments through an interaction (*dialogue*) between two individuals. In his book Meno [29, parts 82b–85b], Socrates speaks with a boy. The boy claims that by doubling the side of a square, the length of which is two feet, we get a square the area of which is twice the area of the initial square, i.e., eight feet. Socrates goes on and refutes this claim, by asking questions, forcing the boy to a contradiction. Quoting from [29]:

> *Socrates.* Wouldn't this line [a side of the initial square] become double itself if we add another of the same length at this point [the end of the line]?
>
> *Boy.* Of course.
>
> *Socrates.* So from this there will be an eight foot area, if we have four such lines?
>
> *Boy.* Yes.
>
> *Socrates.* Let us draw then, using this double line, four equal lines. Is this exactly what you say the eight foot area is or not?
>
> *Boy.* This is exactly it.
>
> *Socrates.* Doesn't this picure contain four areas, each of which is equal to that four foot area?

Boy. Yes.

Socrates. How large is this area? Isn't it four times as large?

Boy. Of course it is.

Socrates. Can therefore, something four times as large, to be double?

Boy. No, by Zeus.

We are going to see several interactive protocols (or games) of this form. The general setting is the following. The two participants (or players) are called Prover and Adversary. The Adversary brags that a statement, say S, is true and the Prover tries to prove him wrong. He does this by asking the Adversary a series of questions. If at some point there is a contradiction in Adversary's answers, e.g., he has responded with a statement and its negation, then the Prover wins. Notice that a winning strategy of the Prover, i.e., a collection $\mathcal{C}$ of sequences of questions for all possible responses, such that each sequence forces the Adversary to a contradiction, is a proof of the negation of S (provided of course that $\mathcal{C}$ can be encoded as a finite string and can be easily verified, which means that there must be some structure in Prover's questions).

Generally, games offer a nice, intuitive way of seeing things in many areas, from algorithms (e.g., adversary arguments) to model theory (e.g., Ehrenfeucht–Fraïssé games). The idea is that a winning strategy for the adversary yields lower bounds. In the case of algorithms "lower bounds" means running time lower bounds; in model theory it means inexpressibility, i.e., results of the kind: a property is not definable in a certain logic (see e.g. [19]). As you may expect, in proof complexity, winning strategies for the Adversary translate to lower bounds on the complexity of proofs.

1.3 Satisfiability algorithms and resolution

Virtually all algorithms for satisfiability are based on a proof system, in a way that lower bounds for the proof system imply limitations of the corresponding algorithm. The most straightforward such connection is the connection between *resolution* and algorithms such as the *DPLL procedure*. As it is reminiscent of resolution's relation to tableaux, which we will see in Chapter 3, we shall illustrate the basic idea here.

Resolution is a proof system for refuting sets of clauses (or CNF-formulas), where a clause is a disjunction of variables and negations of variables. Suppose we begin with a set of clauses, and through sound deductive steps, we end up with the empty clause. Since no assignment can satisfy the empty clause, the initial set of clauses is unsatisfiable. Resolution encodes clauses as sets and its only way of producing new clauses from old ones is the rule:

from $C \cup \{P\}$ and $D \cup \{\neg P\}$, infer $C \cup D$.

DPLL-based algorithms are built upon the following primitive method: On input a set S of clauses, select a variable P, and select a value $v \in \{0, 1\}$. Then

call yourself recursively on input $S[P = v]$, where $S[P = v]$ is the set that results from S after applying the restriction $P = v$ (if for example $v = 1$, remove all clauses containing P and remove $\neg P$ from all clauses). If at some point a clause is falsified, backtrack.

Algorithm 1 A basic satisfiability algorithm.

```
procedure PRIMITIVE(S)
    if S = ∅ then
        return true
    if S contains the empty clause then
        return false
    choose a variable P that occurs in S
    choose a value v ∈ {0, 1}
    return PRIMITIVE(S[P = v]) or PRIMITIVE(S[P = 1 − v])
```

Now, if we run this primitive algorithm on an unsatisfiable set of clauses S, then its execution tree corresponds to a resolution refutation of S. The idea is that if α is the assignment that has created before a recursive call, then α falsifies the clause which results from a resolution derivation corresponding to the call on $S[\alpha]$. Figure 1.1 hopefully makes things clear. The boxes at the upper half contain the initial clauses falsified making the algorithm backtrack.

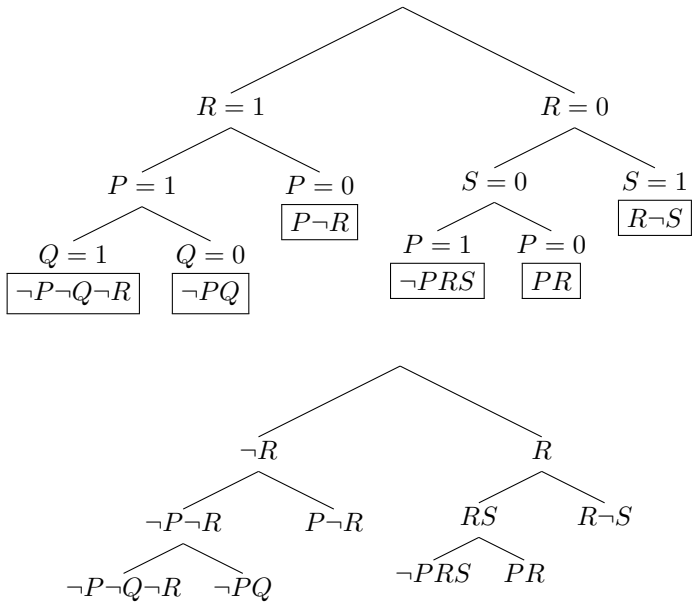

Figure 1.1: A sample run of Algorithm 1 on the set of clauses $S = \{\neg P\neg Q\neg R, \neg PQ, P\neg R, \neg PRS, PR, R\neg S\}$ (up) and the corresponding resolution refutation (down).

Notice that the above transformation tells us that resolution is a *complete* proof system, and on top of that, gives an upper bound on the size of resolution refutations.

Proposition 1.3.1. *If S is an unsatisfiable set of clauses defined over n variables, where n is a non-negative integer, then there is a resolution refutation of S in at most 2^n steps.*

Proof. Fix an order $P_1, \ldots, P_n$ of the variables which occur in S, and call Algorithm 1 with respect to that order. The execution tree, as well as the corresponding resolution tree, will have at most 2^n nodes. □

The first super-polynomial lower bound on the size of resolution proofs was obtained in the 1960's by Tseitin [35], for a subsystem of resolution called *regular resolution*. The first super-polynomial lower bound for general resolution was given by Armin Haken in 1985, showing that the CNF encoding the pigeon-hole principle's negation is intractable for resolution [23]. Many intractability results for resolution followed, based on Haken's method, the so-called "bottleneck method", most notably [36] and [12]. A crucial notion underlying the above proofs was that of *resolution width*; the width of a resolution proof is the size of the largest clause in the proof. This was first made explicit by Ben-Sasson and Wigderson [8], unifying and greatly simplifying the proofs of the aforementioned lower bounds, by showing that *short proofs are narrow*:

> If an r-CNF formula on n variables has a resolution refutation of size S, then it has a resolution refutation of width $O(\sqrt{n \log S}) + r$.

Resolution width, unlike resolution size, is a measure of semantic flavor. This, as far as we know, was first noticed by Atserias and Dalmau [3], characterizing the minimum width needed to refute a formula using a two-player game. That paper (and the subsequent [38], characterizing in a similar way the minimum resolution depth) were the main inspiration and the starting point for our work. Furthermore, from this characterization followed that

> the minimum space of refuting an r-CNF formula X is always bigger than the minimum width of refuting X minus $r - 1$.

1.4 Thesis overview

We define in the present thesis the notions of *tableau depth* and *tableau width*. These measures are generalizations of resolution depth and width, and are characterized by a generic two-player game. We identify cut-free tableau width (which we just call tableau width) as a crucial parameter for resolution, by noticing that many relations concerning resolution width, are not only continue to hold if we replace resolution width by tableau width, but also are simplified. We believe that connections between different versions of the tableau depth/width game, in particular the versions resulting by adding or restricting cuts, are fruitful and should be investigated further.

Chapter 2 contains the basic theory, introducing tableau depth and tableau width. Our exposition of the tableaux proof system closely follows [21]. Chapter 3 contains applications, notably the relation of tableaux with clause space, resolution and Frege systems. The sections of Chapter 3 are independent of each other and can be read in any order.

Remark 1.4.1. Although some of the results presented in this thesis can be extended to first-order, or other logics, the focus is on propositional logic. Traditionally proof complexity deals with only propositional logic. Perhaps one reason for this is that the question "Does there exist a tautology with no short proofs?" can be answered for undecidable logics in a very strong way. Take any proof system F for the set, let's say of all first-order valid formulas VAL. If there was a computable function f such that for any formula $X \in \mathsf{VAL}$ of size n, X has a proof in F of size at most $f(n)$, then we would have a procedure deciding VAL: On input a string x, write down all possible strings of size at most $f(|x|)$, and for each one of them check if it represents a proof of x in F. If such a string was found, return yes; otherwise return no. We know that such a decision procedure cannot exist, therefore for any proof system for VAL and any computable function f, there exists a first-order valid formula with no proof of size $f(n)$.

2

Tableaux

2.1 Uniform notation

We consider throughout the thesis only propositional logic. The formulas are built up from propositional variables $P_1, P_2, \ldots, P, Q, \ldots$ using negations, and finite conjunctions/disjunctions of unbounded arity. A propositional variable or the negation of a propositional variable is called a *literal*. If $X_1, \ldots, X_k$ are formulas, we denote by $\langle X_1, \ldots, X_k \rangle$ their conjunction and by $[X_1, \ldots, X_k]$ their disjunction. $(X \wedge Y)$ and $(X \vee Y)$ are abbreviations for $\langle X, Y \rangle$ and $[X, Y]$ respectively. $(X \to Y)$ is an abbreviation for $[\neg X, Y]$.

Making use of Smullyan's uniform notation [33], we group all formulas of the forms $\langle X_1, \ldots, X_k \rangle$, $\neg\langle X_1, \ldots, X_k \rangle$, $[X_1, \ldots, X_k]$ and $\neg[X_1, \ldots, X_k]$ into two categories, those that act conjunctively, which we call α-*formulas*, and those that act disjunctively, which we call β-*formulas*. For an α-formula we write $\alpha_1, \ldots, \alpha_k$ for its components. Similarly, we write $\beta_1, \ldots, \beta_k$ for the components of a β-formula. The situation is summarized in Table 2.1.

α	α_1	$\ldots$	α_k	β	β_1	$\ldots$	β_k
$\langle X_1, \ldots, X_k \rangle$	X_1	$\ldots$	X_k	$[X_1, \ldots, X_k]$	X_1	$\ldots$	X_k
$\neg[X_1, \ldots, X_k]$	$\neg X_1$	$\ldots$	$\neg X_k$	$\neg\langle X_1, \ldots, X_k \rangle$	$\neg X_1$	$\ldots$	$\neg X_k$

Table 2.1: Uniform notation.

The principles of structural induction and structural recursion as well as the unique parsing theorem stated in this setting are the following.

Theorem 2.1.1 (Structural Induction)**.** *Every formula has a property* **Q** *provided that:*

Base case. *Every literal has property* **Q**.

Inductive step.

If Z has property **Q***, then so does $\neg\neg Z$.*

If every component $\alpha_1, \ldots, \alpha_k$ of α has property **Q***, then so does α.*

If every component $\beta_1, \ldots, \beta_k$ of β has property **Q***, then so does β.*

Theorem 2.1.2 (Structural Recursion)**.** *There is a unique function f defined on the set of propositional formulas such that:*

Base case. *The value of f is specified explicitly on literals.*

Recursion step.

The value of f on $\neg\neg Z$ is specified in terms of the value of f on Z.

The value of f on α is specified in terms of the values of f on the components $\alpha_1, \ldots, \alpha_k$ of α.

The value of f on β is specified in terms of the values of f on the components $\beta_1, \ldots, \beta_k$ of β.

Theorem 2.1.3 (Unique Parsing)**.** *Every formula is in exactly one of the following categories: a literal, $\neg\neg Z$ for a unique formula Z, an α-formula, with unique components $\alpha_1, \ldots, \alpha_k$, or a β-formula, with unique components $\beta_1, \ldots, \beta_k$.*

2.2 What are tableaux?

Tableaux make up one of the simplest proof systems one could think of. To show that a set of formulas S_0 is unsatisfiable, we start from S_0 and try to produce a contradiction. To do this, we expand the formulas in S_0 so that inessential details of their logical structure are cleared away. Such an expansion takes the form of a tree.

We consider a tableau system with the rules depicted in Table 2.2. The

$\dfrac{\neg\neg Z}{Z}$	$\dfrac{\alpha}{\alpha_i}$	for any component α_i of α	$\dfrac{\beta}{\beta_1 \mid \ldots \mid \beta_k}$

Table 2.2: Tableau expansion rules.

rules 2.2 allow us to turn a tree with formulas as node labels into another such tree. Suppose we have a finite tree **T** with nodes labelled by formulas.

Select a branch θ of $\mathbf{T}$ and a non-literal formula occurrence X on θ. If X is $\neg\neg Z$, lengthen θ by adding a node labelled by Z to its end. If X is α, select a component α_i of α and add a node to the end of θ labelled by α_i. Finally, if X is β, add k children to the final node of θ, and label them by the components $\beta_1, \ldots, \beta_k$ of β. Call the resulting tree $\mathbf{T}'$. We say $\mathbf{T}'$ results from $\mathbf{T}$ by the application of a *tableau expansion rule*.

The notion of a tableau is given in the following recursive definition.

Definition 2.2.1. Let $\{X_1, \ldots, X_m\}$ be a set of propositional formulas.

1. The following one-branch tree is a tableau for $\{X_1, \ldots, X_m\}$:

   X_1
   $\vdots$
   X_m

2. If $\mathbf{T}$ is a tableau for $\{X_1, \ldots, X_m\}$ and $\mathbf{T}'$ results from $\mathbf{T}$ by the application of a tableau expansion rule, then $\mathbf{T}'$ is a tableau for $\{X_1, \ldots, X_m\}$.

Definition 2.2.2. A branch θ of a tableau is called *closed* if both X and $\neg X$ occur on θ for some formula X. A tableau is *closed* if every branch is closed. Tableaux form a *refutation proof system*, which means that a tableau proof of a formula X is a closed tableau for $\neg X$ (for singletons $\{X\}$ we say "tableau for X" instead of "tableau for $\{X\}$").

Example 2.2.1. Figure 2.1 shows a tableau proof of $(P \vee (Q \wedge R)) \rightarrow ((P \vee Q) \wedge (P \vee R))$. The first line is the negation of the formula to be proved; the second and third lines resulted from the first by the α rule; the two formulas in the first branch resulted from the second line by the β rule; the branch underneath the node labelled by P resulted by the β rule from the third line, and so forth.

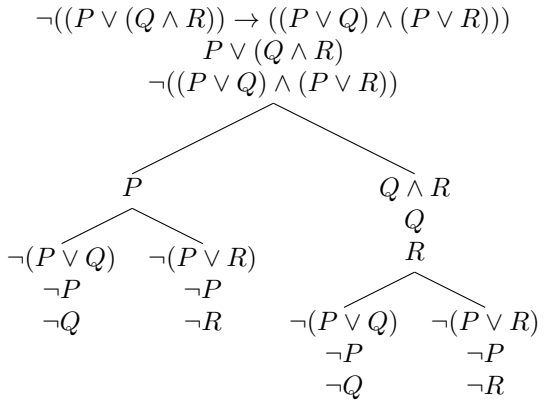

Figure 2.1: A tableau proof of $(P \vee (Q \wedge R)) \rightarrow ((P \vee Q) \wedge (P \vee R))$.

Tableaux constitute a sound and complete proof system. The soundness direction is easy; for the completeness direction see [21], and also Section 2.7.

Theorem 2.2.1. *For any set S_0 of formulas, S_0 is unsatisfiable if and only if there exists a closed tableau for S_0.*

Remark 2.2.1. A minor technical point, which will not bother us, except now. Theorem 2.2.1 does not hold if we are dealing with things such as the negation of the empty conjunction $\neg\langle\ \rangle$ or the empty disjunction $[\ ]$. Tableaux as stated, cannot produce a contradiction from neither of these. To be able to do this, so that Theorem 2.2.1 includes these cases, we may add the rules:

$$\frac{\neg\langle\ \rangle}{X} \qquad \frac{[\ ]}{X},$$

where X is an arbitrary formula, or simply say that a branch is closed if it contains $\neg\langle\ \rangle$ or $[\ ]$. We could augment all the appropriate definitions to include empty disjunctions/conjunctions. We chose, for the sake of simplicity, not to do so.

2.3 Cuts

Tableaux with the rules of Table 2.2 have a very important property: the *subformula property*. Every formula in a proof is a subformula (or the negation of one) of the formula to be proved. It is this property that makes the procedure of constructing proofs appear algorithmic. In Gentzen's words [22]:

> Perhaps we may express the essential properties of such a normal proof by saying: it is not roundabout. No concepts enter into the proof other than these contained in its final result, and their use was therefore essential to the achievement of that result.

The *cut rule*: for any formula X, at any point split a branch in two, adding X to one's end and $\neg X$ to the other's,

$$\frac{}{X \mid \neg X},$$

violates the subformula propery. Instead, it gives immense power to tableau proofs in terms of efficiency.

First, the cut rule allows the use of *intermediate lemmas*. Known tautologies could serve as lemmas for further work: Suppose that we have already proved a formula X, and now we are trying to prove a different formula, Y. In a tableau for $\neg Y$, we can split a branch at any point, adding $\neg X$ to one fork and X to the other. Underneath $\neg X$, we can simply copy the closed tableau for $\neg X$ that we already have, thus closing that fork. This leaves us the other fork to work with, and we have X available on it as a lemma, which may help us in producing a closed branch.

Secondly, consider an axiomatic proof system F with finitely many axiom schemes, say $A_1, \ldots, A_k$, and *modus ponens*

$$\frac{X \qquad X \to Y}{Y}$$

as the only rule of inference, where the proofs are in tree form. A proof in F is a tree the leaves of which are labelled with substitutional instances of the axioms, and the formulas labelling the internal nodes can be derived from the formulas in their children by modus ponens.

Proposition 2.3.1. *Tableaux with cuts p-simulate F.*

Proof. Let T be a proof in F. We show that there is a constant c such that for every node x of T, if X is the formula labelling x and T_x is the subtree of T having x as its root, then there is a closed tableau for $\neg X$ with at most $c \cdot |T_x|$ nodes.

Fix a large enough constant $c \geq 3$ so that for every axiom A_i of F, there is a closed tableau for $\neg A_i$ with c nodes. For the inductive step, suppose that y is an internal node of T labelled by Y, and say that T_1 and T_2 are the immediate subtrees of y, the roots of which are labelled by X and $X \to Y$ respectively. From the induction hypothesis, there are closed tableaux, say $\mathbf{T}_1$ and $\mathbf{T}_2$, for $\neg X$ and $\neg(X \to Y)$, with at most $c \cdot |T_1|$ and $c \cdot |T_2|$ nodes respectively. A closed tableau $\mathbf{T}$ for $\neg Y$ can be constructed as in Figure 2.2.

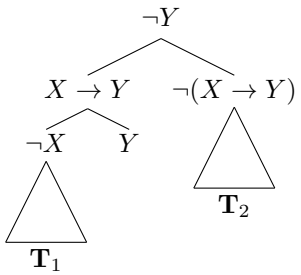

Figure 2.2: A tableau simulation of modus ponens.

$\mathbf{T}$ has at most

$$c \cdot |T_1| + c \cdot |T_2| + 3 \leq c \cdot (|T_1| + |T_2| + 1) = c \cdot |T_y|$$

nodes, and we are done.

It should be clear from the above proof how this transormation can be computed in linear time, therefore tableaux with cuts p-simulate F. □

We will see Proposition 2.2 in its full generality in Chapter 3.

Gentzen's cut elimination theorem gives an algorithm that removes the cuts from proofs, making them purely analytic. For a quite readable proof

(basically Gentzen's proof [22] in tableau terms), see [21, Section 8.9]. As we may anticipate, eliminating cuts may result in a blow-up of proof size; in fact cuts can make the difference between linear and exponential. This, as far as we know, was first made explicit for propositional logic in [34] (also see [21, Section 8.10] and Section 3.3), by giving an example of formulas having proofs of linear size when the use of cuts is allowed, but require cut-free proofs of exponential size.

Henceforth, by tableaux we mean tableaux without the cut rule; when we want to speak of tableaux with cuts, we will mention it explicitly.

Remark 2.3.1. Let us, before moving on, make one more observation about the tableau proof system. We stressed in Chapter 1 that there are two dual ways of seeing a proof system: as a bottom-up proof system, or as a top-down satisfiability algorithm. For example, we saw that resolution, when seen as a satisfiability algorithm, tries to construct a satisfying assignment for the set we are trying to refute, by querying variables. Tableaux are a system already in its algorithmic manifestation. They try to construct an assignment (or more generally a model) satisfying the set we are trying to refute, by querying, either formulas in the partial model we have already constructed, or (when the cut rule is allowed) arbitrary formulas. The bottom-up counterpart of tableaux is a proof system, first defined by Gentzen in [22] in order to prove his cut-elimination theorem, called *sequent calculus* (see [33, Chapter XI, Section 1] for details).

One difference between resolution and the DPLL procedure, or between sequent calculus and tableaux, is that in the latter the proofs are restricted to be in tree-like form, whereas in the former the proofs can be in a DAG-like form. While a representation of a proof as a DAG can be much shorter than its shortest representation as a tree, the tree-like restriction doesn't affect the depth or width of proofs. As we will see, it also does not affect the size of the proofs, if the set from where the cut-formulas are drawn is big enough.

2.4 Tableau depth

We define the length of a path as the number of its vertices. The *depth*, $\mathrm{TDepth}(\mathbf{T})$, of a tableau $\mathbf{T}$, is the length of its longest branch. The *tableau depth* of an unsatisfiable set of formulas S_0 is the minimum depth of a closed tableau for S_0:

$$\mathrm{TDepth}(S_0) \stackrel{\text{def}}{=} \min\{\mathrm{TDepth}(\mathbf{T}) : \mathbf{T} \text{ is a closed tableau for } S_0\}.$$

If S_0 is the singleton $\{X\}$, we write $\mathrm{TDepth}(X)$ instead of $\mathrm{TDepth}(\{X\})$.

The collection of all sets of formulas for which there exists a closed tableau (of unbounded depth) is, from Theorem 2.2.1, exactly the collection of the unsatisfiable sets of formulas. What about the collection of all sets of formulas for which there exists a closed tableau of depth d (for a fixed positive integer d)? We characterize those sets using the notion of a *d-consistency property*.

Definition 2.4.1. A d-consistency property is a collection $\mathcal{C}$ of sets of formulas such that for each $S \in \mathcal{C}$:

1. For each formula X, not both $X \in S$ and $\neg X \in S$.
2. $\neg\neg Z \in S \ \ \& \ \ |S| < d \implies S \cup \{Z\} \in \mathcal{C}$.
3. $\alpha \in S \ \ \& \ \ |S| < d \implies S \cup \{\alpha_i\} \in \mathcal{C}$ for every component α_i of α.
4. $\beta \in S \ \ \& \ \ |S| < d \implies S \cup \{\beta_i\} \in \mathcal{C}$ for some component β_i of β.

We say that a set S_0 of formulas has the d-consistency property if there exists a d-consistency property $\mathcal{C}$ such that $S_0 \in \mathcal{C}$.

Remark 2.4.1. A d-consistency property is a bounded version of what Fitting calls in [21] a propositional consistency property.

Lemma 2.4.1 (Soundness)**.** *If S_0 has the d-consistency property, then there does not exist a closed tableau of depth at most d for S_0.*

Proof. For a branch θ of a tableau, we denote by Γ_θ the set of formulas that appear in θ.

Let $\mathcal{C}$ be a d-consistency property such that $S_0 \in \mathcal{C}$. We will show that for every tableau $\mathbf{T}$ for S_0 of depth at most d, there exists a non-closed branch θ of $\mathbf{T}$ such that $\Gamma_\theta \in \mathcal{C}$.

> *Base case.* If $\mathbf{T}$ consists of a unique branch with the formulas of S_0 labelling its nodes, then $\mathbf{T}$ cannot be closed because of the condition 1 of Definition 2.4.1.
>
> *Inductive step.* Suppose that $\mathbf{T}$ has depth at most d and resulted from $\mathbf{T}'$ by the application of a tableau expansion rule on a branch τ of $\mathbf{T}'$. From the induction hypothesis there exists a non-closed branch θ' of $\mathbf{T}'$ such that $\Gamma_{\theta'} \in \mathcal{C}$. If $\tau \neq \theta'$ we are done, since θ' is a branch of $\mathbf{T}$. Otherwise, we have the following cases:
>
> > *Case 1.* $\neg\neg Z \in \Gamma_{\theta'}$ and the branch θ of $\mathbf{T}$ resulted adding to the end of θ' a node labelled by Z. $\mathbf{T}$ has depth at most d, thus $|\Gamma_\theta| \leq d$ and $|\Gamma_{\theta'}| < d$. Now since $\Gamma_{\theta'} \in \mathcal{C}$, from the condition 2 of Definition 2.4.1, $\Gamma_\theta \in \mathcal{C}$, and from the condition 1 θ is not closed.
> >
> > *Case 2.* $\alpha \in \Gamma_{\theta'}$ and θ resulted adding to the end of θ' a node labelled by α_i, where α_i is a component of α. As in case 1, $|\Gamma_{\theta'}| < d$, thus $\Gamma_\theta \in \mathcal{C}$ and θ is not closed.
> >
> > *Case 3.* $\beta \in \Gamma_{\theta'}$ and $\mathbf{T}$ resulted adding to the end of θ' a fork of k nodes labelled by the components $\beta_1, \ldots, \beta_k$ of β. We have that $|\Gamma_{\theta'}| < d$ and $\Gamma_{\theta'} \in \mathcal{C}$, so from the condition 4 of Definition 2.4.1, there is a β_i such that $\Gamma_{\theta'} \cup \{\beta_i\} \in \mathcal{C}$. For the branch θ of $\mathbf{T}$ which contains θ' and ends with β_i, it holds that $\Gamma_\theta \in \mathcal{C}$ and θ is not closed. □

Lemma 2.4.2 (Completeness)**.** *Let S_0 be a set of propositional formulas such that $|S_0| \leq d$. If there is no closed tableau of depth at most d for S_0, then S_0 has the d-consistency property.*

Proof. Let

$$\mathcal{C} := \{S : |S| \leq d \ \ \& \ \ \mathrm{TDepth}(S) > d\}.$$

Obviously $S_0 \in \mathcal{C}$. We will show that $\mathcal{C}$ is a d-consistency property, i.e., $\mathcal{C}$ satisfies the conditions of Definition 2.4.1. For the first condition, let S be an arbitrary member of $\mathcal{C}$. For any formula X, it cannot hold that both $X \in S$ and $\neg X \in S$, since then a single branch of length $|S|$ labelled by the formulas of S would constitute a closed tableau of depth at most d for S.

For condition 2, we work in the contrapositive direction. Let $S \in \mathcal{C}$. Suppose that $\neg\neg Z \in S$ and $|S| < d$, but $S \cup \{Z\} \notin \mathcal{C}$. We have that $|S \cup \{Z\}| \leq d$, so since $S \cup \{Z\} \notin \mathcal{C}$, $\mathrm{TDepth}(S \cup \{Z\}) \leq d$, therefore there is a closed tableau, let us call it $\mathbf{T}$, for the set $S \cup \{Z\}$ of depth at most d. But we can easily see that $\mathbf{T}$ is also a closed tableau for the diminished set S, contradicting the fact that S is in $\mathcal{C}$.

Conditions 3 and 4 are similar to condition 2. Let $S \in \mathcal{C}$. Suppose that $\alpha \in S$ and $|S| < d$ but there is a component α_i of α such that $S \cup \{\alpha_i\} \notin \mathcal{C}$. As before, there is a closed tableau $\mathbf{T}$ of depth at most d for the set $S \cup \{\alpha_i\}$ and $\mathbf{T}$ is also a closed tableau for the set S.

For condition 4, suppose that $\beta \in S$ and $|S| < d$, but $S \cup \{\beta_i\} \notin \mathcal{C}$ for every component β_i, $1 \leq i \leq k$, of β. Then there are closed tableaux $\mathbf{T}_1, \ldots, \mathbf{T}_k$ each of depth at most d for the sets $S \cup \{\beta_1\}, \ldots, S \cup \{\beta_k\}$, and is an easy task to construct from $\mathbf{T}_1, \ldots, \mathbf{T}_k$ a closed tableau of depth at most d for the set S. □

Combining Lemmas 2.4.1 and 2.4.2 we get:

Theorem 2.4.1. *Let S_0 be a set of propositional formulas such that $|S_0| \leq d$. Then there is no closed tableau for S_0 of depth at most d if and only if S_0 has the d-consistency property.*

Remark 2.4.2. We can modify accordingly Definition 2.4.1 so that Theorem 2.4.1 holds for any kind of tableau system, e.g. tableaux with cuts, first-order tableaux, tableaux for modal logics etc. For example, if we augment the tableau rules of Table 2.2 with the cut rule, where the cut formulas are from a specified set $\mathcal{A}$,

$$\frac{}{X \ \mid \ \neg X} \quad \text{for any formula } X \in \mathcal{A},$$

then the change to Definition 2.4.1 will be to add the condition

$$X \in \mathcal{A} \ \ \& \ \ |S| < d \implies S \cup \{X\} \in \mathcal{C} \ \ \text{or} \ \ S \cup \{\neg X\} \in \mathcal{C}.$$

2.5 The Prover-Adversary game

Definitions such as Definition 2.4.1 are often seen as a game between two players. We call the two players *Prover* and *Adversary* and the game, played on a set of formulas S_0, is as follows: The aim of the Prover is to show that S_0 is unsatisfiable, demonstrating a contradiction, while the Adversary tries to frustrate this intention. During a play of the game the two players construct a set of formulas S. Initially, $S := S_0$. In an arbitrary round, the Prover selects a non-literal $X \in S$ and

- if X is of the form $\neg\neg Z$, then the Prover sets $S := S \cup \{Z\}$;
- if X is of the form α, then the Prover selects a component α_i of α and sets $S := S \cup \{\alpha_i\}$;
- if X is of the form β, then *the Adversary* selects a component β_i of β and the Prover sets $S := S \cup \{\beta_i\}$.

If at any moment S contains a formula and its negation, then the Prover wins.

Due to the completeness of the tableau system, if we allow an unlimited number of rounds, the Prover can always win, provided that S_0 is unsatisfiable. This is not the case if we bound the number of rounds. Indeed, a d-consistency property $\mathcal{C}$ containing S_0 provides a winning strategy for the Adversary when the number of rounds is bounded by $d - |S_0|$, in the sense that no matter what Prover does, he can't win in the course of $d - |S_0|$ rounds. The converse of the above sentence is also true. If the Adversary has a winning strategy for $d - |S_0|$ rounds, then the responses, according to this strategy, to all the possible queries of the Prover, form a d-consistency property containing S_0.

Proposition 2.5.1. *Let S_0 be a set of formulas. Then S_0 has the d-consistency property if and only if the Adversary has a winning strategy for the Prover-Adversary game played on S_0, when the number of rounds is bounded by $d - |S_0|$.*

2.6 Tableau width

Consider the following variation of the Prover-Adversary game, which we call the *width game*. We give Prover the ability to "forget" formulas. Initially, $S := S_0$, where S_0 is the input, i.e., the set of formulas the game is played on. In an arbitrary round, the Prover either selects a non-literal formula $X \in S$ and the game proceeds as before, or forgets a formula $X \in S$, i.e., selects a formula $X \in S$ and sets $S := S - \{X\}$ as the current set. We now have an unlimited number of rounds, but we set a bound to the size of the set S the two players preserve. The question is: Can the Prover always reach a contradiction (reach a set containing a formula and its negation), maintaining that $|S| \leq w$? If he can, we say that the Prover has a winning strategy for the w-width game played on S_0; otherwise, we say that the Adversary has a winning strategy. Like in Proposition 2.5.1, we can see that the following proposition is true.

Proposition 2.6.1. *Let S_0 be a set of formulas such that $|S_0| \leq w$. Then there is a w-consistency property which is* closed under subsets *containing S_0 if and only if the Adversary has a winning strategy for the w-width game played on S_0.*

For an unsatisfiable set of formulas S_0, we define the *tableau width* of S_0, $\mathrm{TWidth}(S_0)$, as the minimum w such that the Prover has a winning strategy for the w-width game on S_0. Again, for a single formula X, we write $\mathrm{TWidth}(X)$ instead of $\mathrm{TWidth}(\{X\})$. Notice that

$$\mathrm{TWidth}(S_0) \leq \mathrm{TDepth}(S_0),$$

since a tableau of depth at most d for S_0 gives rise to a winning strategy for the Prover which lasts for at most $d - |S_0|$, and this strategy is also a winning strategy for the d-width game.

Tableau width is a space measure. We shall show now, that indeed, for any reasonable definition of tableau space, tableau width is less than or equal to tableau space.

Let S_0 be a set of formulas. Suppose that during the construction of a tableau for S_0 we are allowed to manipulate the tableau, deleting some of its nodes. Of course, we cannot do this arbitrarily; we must do it in a sound way. This means that, if $\mathbf{T}'$ and $\mathbf{T}$ are trees the nodes of which are labelled with formulas, and $\mathbf{T}'$ resulted after the deletion of some of the nodes of $\mathbf{T}$, then for every non-closed branch θ of $\mathbf{T}$, there exists a branch τ of $\mathbf{T}'$ such that the set of formulas which occur on τ is a subset of the formulas which occur on θ. We write $\mathbf{T} \models\!\!\approx \mathbf{T}'$ for two trees $\mathbf{T}$ and $\mathbf{T}'$ satisfying the above condition.

Definition 2.6.1. An *s-tableau sequence* for S_0 is a sequence $\mathbf{T}_1, \ldots, \mathbf{T}_s$ of trees, such that $\mathbf{T}_1$ is the one-branch tableau with $|S_0|$ nodes labelled by all the formulas in S_0, and for each i, $1 < i \leq s$,

1. $\mathbf{T}_i$ results from $\mathbf{T}_{i-1}$ by the application of a tableau expansion rule, or
2. $\mathbf{T}_{i-1} \models\!\!\approx \mathbf{T}_i$.

If $\mathbf{T}_s$ is closed then the sequence is called an *s-tableau refutation* of S_0.

Definition 2.6.2. The *space*, $\mathrm{TSpace}(\pi)$, of an s-tableau sequence π is the size (i.e., the number of nodes) of the biggest tree in π. The *tableau space* of an unsatisfiable set of formulas S_0 is the minimum space of a tableau refutation of S_0:

$$\mathrm{TSpace}(S_0) \stackrel{\mathrm{def}}{=} \min\{\mathrm{TSpace}(\pi) : \pi \text{ is an s-tableau refutation of } S_0\}.$$

Proposition 2.6.2. *For any unsatisfiable set of formulas S_0,*

$$\mathit{TWidth}(S_0) \leq \mathit{TSpace}(S_0).$$

Proof. The proof mimics that of Lemma 2.4.1. Suppose that $\mathrm{TWidth}(S_0) > w$. This means that there exists a w-consistency property $\mathcal{C}$ closed under subsets containing S_0. We show that there cannot be an s-tableau refutation of S_0 of space less than or equal to w. More specifically, let $\pi = \mathbf{T}_1, \ldots, \mathbf{T}_s$ an s-tableau sequence such that the size $|\mathbf{T}_i|$ of every tableau in π is less than or equal to w; we show that every $\mathbf{T}_i$ must have a non-closed branch θ such that $\Gamma_\theta \in \mathcal{C}$, where Γ_θ is the set of formulas occurring in θ.

> *Base case.* $\mathbf{T}_1$ has as its only branch a branch θ labelled by the formulas in S_0, so $\Gamma_\theta \in \mathcal{C}$ and from the condition 1 of Definition 2.4.1 θ cannot be closed.
>
> *Inductive step.* Suppose that $\mathbf{T}_i$ resulted from $\mathbf{T}_{i-1}$, and let θ' be a non-closed branch of $\mathbf{T}_{i-1}$ such that $\Gamma_{\theta'} \in \mathcal{C}$. If $\mathbf{T}_i$ is the result of the application of a tableau expansion rule on a branch of $\mathbf{T}_{i-1}$ different from θ', then θ' remains in $\mathbf{T}_i$ and we are done. So suppose (the two other cases are similar) that $\beta \in \Gamma_{\theta'}$ and $\mathbf{T}_i$ resulted from $\mathbf{T}_{i-1}$ adding to the end of θ' a fork of k nodes labelled by the components $\beta_1, \ldots, \beta_k$ of β. We have that $|\mathbf{T}_i| \leq w$, therefore $|\Gamma_{\theta'}| < w$. From the condition 4 of Definition 2.4.1, there is a β_i such that $\Gamma_{\theta'} \cup \{\beta_i\} \in \mathcal{C}$ and for the branch θ of $\mathbf{T}_i$ which contains θ' and ends with β_i, it holds that $\Gamma_\theta \in \mathcal{C}$ and θ is not closed.
>
> Next, suppose that $\mathbf{T}_{i-1} \not\approx \mathbf{T}_i$. Then $\mathbf{T}_i$ contains a branch θ such that $\Gamma_\theta \subseteq \Gamma_{\theta'}$, and since $\mathcal{C}$ is subset closed, $\Gamma_\theta \in \mathcal{C}$. □

Again, Proposition 2.6.2 extends to tableaux with cuts, first-order tableaux etc.

2.7 An upper bound

We show in this section a general upper bound on the depth (and thus also the width) of an unsatisfiable set of formulas. Doing this we also prove tableau completeness for finite sets; the infinite case follows via *compactness*. Although the upper bound is fairly straightforward, it requires some preliminary results.

Definition 2.7.1. A tableau $\mathbf{T}$ is called *saturated* if for each branch θ of $\mathbf{T}$:

1. If $\neg\neg Z$ occurs on θ, then so does Z.
2. If α occurs on θ, then so does every component α_i of α.
3. If β occurs on θ, then at least one component β_i of β occurs on θ.

Lemma 2.7.1 (Hintikka's Lemma)**.** *Let S_0 be an unsatisfiable set of formulas. Then every saturated tableau for S_0 must be closed.*

Proof. Suppose, for the sake of contradiction, that **T** is a saturated but not closed tableau for S_0. Since **T** is not closed, it must contain a non-closed branch. Let θ be such a branch and let H be the set of formulas that appear on θ. We show that there exists an assignment that satisfies all formulas in H, which, since $S_0 \subseteq H$, contradicts the assumption that S_0 is unsatisfiable.

Let f be the assignment of Boolean values to propositional variable defined as follows: For every propositional variable P

$$f(P) \coloneqq \begin{cases} \textbf{true}, & \text{if } P \in H, \\ \textbf{false}, & \text{otherwise.} \end{cases}$$

Now let v be the Boolean valuation (i.e., the assignment of Boolean values to all propositional formulas respecting the meaning of connectives) that extends f. We show that for every formula X,

$$X \in H \implies v(X) = \textbf{true}.$$

The proof is by structural induction (Theorem 2.1.1).

Base case. Suppose that X is a literal and that $X \in H$. If X is a propositional variable P, then by definition, $v(X) = \textbf{true}$. If X is the negation of a variable $\neg P$, then from the fact that θ is not closed, it cannot be the case that $P \in H$, therefore $v(P) = \textbf{false}$ and $v(X) = \textbf{true}$.

Inductive step. Suppose that $Z \in H \implies v(Z) = \textbf{true}$; we show that $\neg\neg Z \in H \implies v(\neg\neg Z) = \textbf{true}$. Suppose that $\neg\neg Z \in H$. Then because **T** is saturated, $Z \in H$, so $v(Z) = \textbf{true}$ and $v(\neg\neg Z) = \textbf{true}$.

The α and β cases are similar to the $\neg\neg Z$ case. We show only the second. Suppose that $\beta_i \in H \implies v(\beta_i) = \textbf{true}$ for every component β_i of β. If $\beta \in H$, then since **T** is saturated, $\beta_j \in H$ for some component β_j, thus $v(\beta_j) = \textbf{true}$ and $v(\beta) = \textbf{true}$. □

Theorem 2.7.1. *Let h be a function mapping every formula to an integer, such that $h(P) = h(\neg P) = 1$ for every propositional variable, and*

$$\begin{aligned} h(\neg\neg Z) &= 1 + h(Z); \\ h(\alpha) &= 1 + \sum h(\alpha_i); \\ h(\beta) &= 1 + \max h(\beta_i). \end{aligned}$$

From Theorem 2.1.2 h exists and is unique. For any finite unsatisfiable set of formulas $S_0 = \{X_1, \ldots, X_m\}$, there is a closed tableau for S_0 of depth

$$\sum_{i=1}^{m} h(X_i).$$

Proof. We show that for any formula X, there is a saturated tableau for X of depth $h(X)$. Theorem 2.7.1 then follows because a saturated tableau $\mathbf{T}$ for S_0 of depth $k-1$ can be obtained from a saturated tableau for X_{S_0} of depth k, where X_{S_0} is the conjunction of all the formulas in S_0. From Lemma 2.7.1, $\mathbf{T}$ will be closed. The proof is by structural induction.

Base case. For every literal L, the tableau with one node labelled by L, is a saturated tableau for L of depth $1 = h(L)$.

Inductive step. Suppose that $\mathbf{T}$ is a saturated tableau for Z of depth $h(Z)$. For a saturated tableau for $\neg\neg Z$ simply plug-in $\mathbf{T}$ under $\neg\neg Z$. The depth of the new tableau is $1 + h(Z) = h(\neg\neg Z)$.

For the α case, suppose that every component α_i, $1 \leq i \leq k$, of α has a saturated tableau $\mathbf{T}_i$ of depth $h(\alpha_i)$. Start with $\mathbf{T}_1$ and to the end of every branch of $\mathbf{T}_1$ add $\mathbf{T}_2$. Call the resulting tree $\mathbf{T}'$. Then do the same, adding $\mathbf{T}_3$ to the end of every branch of $\mathbf{T}'$. Keep doing this, until all the trees $\mathbf{T}_k$ have been added, and then add a new root (making the old root a child of the new root) labelled by α. Call the resulting tree $\mathbf{T}^*$. The depth of $\mathbf{T}^*$ is $1 + \sum_{i=1}^{k} h(\alpha_i) = h(\alpha)$. We claim that $\mathbf{T}^*$ is a saturated tableau for α. The fact that $\mathbf{T}^*$ is a valid tableau is easy to verify, and $\mathbf{T}^*$ is saturated since all $\mathbf{T}_i$ are saturated and every branch of $\mathbf{T}^*$ contains all the components α_i of α.

Finally, for the β case, suppose that every component β_i, $1 \leq i \leq k$, of β has a saturated tableau $\mathbf{T}_i$ of depth $h(\beta_i)$. For a saturated tableau for β, start with a node labelled by β, add $\mathbf{T}_1, \ldots, \mathbf{T}_k$ as the immediate subtrees of β. It is easy to see that the resulted tree is a saturated tableau for β, and its depth is $1 + \max\{\beta_i : 1 \leq i \leq k\} = h(\beta)$. □

Corollary 2.7.1. *For any unsatisfiable set of formulas $S_0 = \{X_1, \ldots, X_m\}$,*

$$TDepth(S_0) \leq \sum_{i=0}^{m} s(X_i),$$

where for a formula X, $s(X)$ is the number of symbols appearing in X.

Proof. We show, by structural induction, that for any formula X, $h(X) \leq s(X)$. The base case is clearly true. For the inductive step, we only show the α case. The others are similar. Suppose that $h(\alpha_i) \leq s(\alpha_i)$ for every component α_i, $1 \leq i \leq k$, of α. Then

$$\begin{aligned} h(\alpha) &= 1 + \sum_{i=1}^{k} h(\alpha_i) \\ &\leq 1 + \sum_{i=1}^{k} s(\alpha_i) \\ &= 1 + (s(\alpha) - c), \qquad \text{where } c \geq 1 \\ &\leq s(\alpha) \end{aligned}$$

□

Corollary 2.7.2. *Let S_0 be an unsatisfiable set of formulas, and c an integer bigger than the arities of the connectives occurring in S_0. If s is the total number of symbols in S_0, then there exists a closed tableau for S_0 with at most c^{s+1} nodes.*

Proof. Immediate, from the fact that a tree of depth d, every node of which has at most c children, has at most $(c^{d+1}-1)/(c-1)$ nodes. □

2.8 Examples

Example 2.8.1. Let us begin with a family of formulas for which Prover has an efficient winning strategy in the basic variation of the Prover-Adversary game. Set

$$X_n \coloneqq \langle P_1, P_1 \to P_2, P_2 \to P_3, \ldots, P_{n-1} \to P_n, \neg P_n \rangle.$$

Prover's strategy is a "binary search" on the sequence

$$P_1, P_1 \to P_2, P_2 \to P_3, \ldots, P_{n-1} \to P_n, \neg P_n.$$

He starts by selecting from X_n the middle formula $P_{\lceil n/2 \rceil} \to P_{\lceil n/2 \rceil + 1}$. The Adversary may respond with either $\neg P_{\lceil n/2 \rceil}$ or $P_{\lceil n/2 \rceil + 1}$. In the first case the Prover continues recursively with the sequence

$$P_1, P_1 \to P_2, P_2 \to P_3, \ldots, P_{\lceil n/2 \rceil - 1} \to P_{\lceil n/2 \rceil}, \neg P_{\lceil n/2 \rceil}$$

and in the second case he continues with the sequence

$$P_{\lceil n/2 \rceil + 1}, P_{\lceil n/2 \rceil + 1} \to P_{\lceil n/2 \rceil + 2}, P_{\lceil n/2 \rceil + 2} \to P_{\lceil n/2 \rceil + 3}, \ldots, P_{n-1} \to P_n, \neg P_n.$$

In at most $2\lceil \log n \rceil$ rounds, the players will reach a set containing either $\neg P_1$, or P_n, or P_i and $\neg P_i$ for some i, $1 < i < n$.

The best the Prover can do is $2 \log n$. A winning strategy of the Adversary when the game is played for at most $2 \log n - 1$ rounds is the following. Suppose that the Prover selects at the beginning the formula $P_i \to P_{i+1}$. If $i < n/2$ then the Adversary selects P_{i+1}; otherwise he selects $\neg P_i$. Suppose (the other case is similar) that $i < n/2$. Then after Adversary selected P_{i+1}, it is no use for the Prover to play on the sequence

$$P_1, P_1 \to P_2, P_2 \to P_3, \ldots, P_i \to P_{i+1}$$

and if he plays on the other half sequence (which becomes the current critical sequence), the Adversary continues as before. Every time the Adversary picks a formula, the size of the critical sequence is reduced by at most a factor of two, therefore the Adversary can be consistent for at least $2 \log n - 1$ rounds.

It follows that

$$2 \log n < \mathrm{TDepth}(X_n) \leq 2 \log n + 2.$$

Example 2.8.2. The next example is a family of formulas for which the Prover does not have a good winning strategy. Set

$$A_i \coloneqq \langle P_1, \ldots, P_i \rangle,$$

$$S_n \coloneqq \{A_1, A_1 \to P_2, A_2 \to P_3, \ldots, A_{n-1} \to P_n, \neg P_n\}$$

and let X_n be the conjunction of all the formulas in S_n. Also, for $1 < i \leq n$, let $S_n[P_i]$ be the formula which results from S_n removing the formula $A_{i-1} \to P_i$ and removing every occurrence of P_i.

Let $G(S_n)$ be the minimum number of rounds in which the Prover can win the Prover-Adversary game played on S_n. We will show, by induction on n, that for every $n \geq 2$,

$$G(S_n) \geq n.$$

From this, will also follow that $G(X_n) \geq n$ and $\mathrm{TDepth}(X_n) > n$. The base case is immediate. For the inductive step, suppose that $n > 2$, and suppose that the first move of the Prover is to select the formula $A_{i-1} \to P_i$. Adversary responds with P_i. One round has passed and now the current set is $S_n \cup \{P_i\}$. We show that $G(S_n \cup \{P_i\}) \geq n - 1$; $G(S_n) \geq n$ follows. The argument is that

$$\begin{aligned} G(S_n \cup \{P_i\}) &\geq G(S_n[P_i]) \\ &\geq G((S_{n-1}) \\ &\geq n - 1. \end{aligned}$$

The first line follows because when the game is played on $S_n \cup \{P_i\}$, the Prover accomplishes nothing by selecting the formula $A_{i-1} \to P_i$ and if he selects a formula of the form $\neg A_j$ for some $j > i$, the Adversary cannot select $\neg P_i$. The second line follows from the fact that the sets $S_n[P_i]$ and S_{n-1} are the same up to a renaming of the variables and the third line is the induction hypothesis.

From Theorem 2.7.1, $\mathrm{TDepth}(X_n) \leq 1 + 3n$, so the above lower bound is tight.

Example 2.8.3. The cut rule gives the ability to the Prover to select at any time an arbitrary formula X. Adversary's response must be either X or $\neg X$. We show now, that equipped with this ability, the Prover can refute the set S_n (and therefore also the formula X_n) of the previous example in $O(\log n)$ rounds.

Prover's strategy is the following: He starts with the set S_n and selects the "middle" formula $A_{\lceil n/2 \rceil}$. If the Adversary responds with $A_{\lceil n/2 \rceil}$, then the current set contains the set consisting of the formulas

$$A_{\lceil n/2 \rceil}, A_{\lceil n/2 \rceil} \to P_{\lceil n/2 \rceil + 1}, A_{\lceil n/2 \rceil + 1} \to P_{\lceil n/2 \rceil + 1}, \ldots, A_{n-1} \to P_n, \neg A_n$$

and the Prover may continue recursively with this set. If the Adversary responds with $\neg A_{\lceil n/2 \rceil}$ then the Prover may continue recursively with the set

$$A_1, A_1 \to P_2, A_2 \to P_3, \ldots, A_{\lceil n/2 \rceil - 1} \to P_{\lceil n/2 \rceil}, \neg A_{\lceil n/2 \rceil}.$$

In at most $\lceil \log n \rceil$ rounds, the two players will reach a set containing the formulas A_i, $A_i \to P_{i+1}$ and $\neg A_{i+1}$, and from this set the Prover can reach a contradiction in at most four rounds.

Prover's strategy in Example 2.8.3 can be stated in a more general manner, by noticing that Prover always picks an *interpolant* which splits the current unsatisfiable set of formulas into two sets of equal size.

Definition 2.8.1. Let S be a set of formulas and let (S_1, S_2) be a partition of S into two sets S_1 and S_2 (i.e., S_1 and S_2 satisfy $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$). An *interpolant* of S with respect to the partition (S_1, S_2) is a formula I such that every every propositional variable of I also occurs in both S_1 and S_2 and the sets $S_1 \cup \{\neg I\}$ and $S_2 \cup \{I\}$ are both unsatisfiable.

The Prover can always find such an interpolant, since:

Lemma 2.8.1. *For any set S of formulas and any partition (S_1, S_2) of S, if S is unsatisfiable, then there exists an interpolant of S with respect to (S_1, S_2).*

Proof. Let $\mathbf{P}_1$ be the set of variables that occur in S_1 and let $\mathbf{P}_2$ be the set of variables that occur in S_2. Set $\mathbf{Q} \coloneqq \mathbf{P}_1 \cap \mathbf{P}_2$ and define the function f as follows. For any assignment α of Boolean values to $\mathbf{Q}$,

$$f(\alpha) \coloneqq \begin{cases} \textbf{true}, & \text{if } S_2 \text{ becomes unsatisfiable under } \alpha, \\ \textbf{false}, & \text{otherwise.} \end{cases}$$

The formula I defined over the variables in $\mathbf{Q}$ that realizes f is an interpolant of S with respect to (S_1, S_2). This is easy to check, since for any assignment to $\mathbf{Q}$, either S_1 becomes unsatisfiable or S_2 becomes unsatisfiable. □

Craig's celebrated interpolation theorem says that Lemma 2.8.1 holds when S is an unsatisfiable set of first-order sentences. As a matter of fact, an interpolant of S (with respect to a partition) of size at most n can be constructed from a closed cut-free tableau for S of size n (see [21, Section 8.12]).

Example 2.8.4. Actually the formulas in both examples 2.8.1 and 2.8.2 belong to a more general family of formulas, referred to in the literature as *pebbling contradictions*.

Definition 2.8.2. Let G be a finite DAG. Let $S \subseteq V(G)$ be the set containing the vertices in G of zero in-degree and $T \subseteq V(G)$ the set containing the vertices of zero out-degree. Associate with each vertex $x \in V(G)$ a variable P_x. $\mathrm{Peb}(G)$ is defined as the conjunction of the following formulas:

1. the formulas P_x, for each $x \in S$;
2. the formulas $\neg P_x$, for each $x \in T$;
3. the formulas $\langle P_{x_1}, \ldots, P_{x_k} \rangle \to P_x$, for each vertex $x \in V(G)$ with immediate predecessors the vertices $x_1, \ldots, x_k$, where $k > 0$.

So the formulas in Example 2.8.1 correspond to directed paths, and the formulas in Example 2.8.2 to the transitive closures of directed paths.

Peb(G) is unsatisfiable, because every assignment α that satisfies the formulas of the forms 1 and 3, must make every variable P_x true, thus α falsifies the formulas of the form 2. We shall see in this example that for any finite DAG G,

$$\text{TWidth}(\text{Peb}(G)) = O(1).$$

Therefore $\{\text{Peb}(G)\}$ is a family of formulas that separates cut-free tableau width from cut-free tableau depth.

Prover's strategy is the following. He starts on a vertex $t \in T$ and walks along a path in the reverse direction to a vertex $s \in S$. He maintains the invariant that when he is on a vertex x, then $\neg P_x \in S$, and at any time $|S| = O(1)$, where S is the set the two players keep. At the start of the game, the Prover selects $\neg P_t$ and the invariant is true. Now, suppose that the Prover is on a vertex x. If $x \in S$, then he selects P_x, and the game ends, since $\{P_x, \neg P_x\} \in S$. Otherwise, he selects the formula $\langle P_{x_1}, \ldots, P_{x_k}\rangle \to P_x$, where $x_1, \ldots, x_k$ are the immediate predecessors of x. If the Adversary selects P_x, the game ends. If he selects $\neg\langle P_{x_1}, \ldots, P_{x_k}\rangle$, then the Prover asks for $\neg\langle P_{x_1}, \ldots, P_{x_k}\rangle$, and after the Adversary selects one of the literals $\neg P_{x_1}, \ldots, \neg P_{x_k}$, say $\neg P_{x_i}$ the Prover moves to the vertex x_k and deletes from S all formulas except Peb(G) and $\neg P_{x_i}$.

3

Applications

3.1 CNF-formulas and the space needed to refute them

A *clause* is a disjunction of literals. A conjuctive normal form formula (shortly, *CNF-formula*) is a conjunction of clauses. Finally an r-CNF formula is a CNF-formula the clauses of which have at most r literals.

The Prover-Adversary games have a particularly nice form when they are played on a CNF-formula X; they boil down to this: The Prover is selecting the clauses of X, and when the Prover selects a clause C, the Adversary must respond with a literal contained in C. If the Adversary selected a variable and its negation, the Prover wins. Note that the upper bound of Theorem 2.7.1 gives us $\mathrm{TDepth}(X) \leq 1 + 2m$ for any CNF formula with m clauses.

In this section we will prove the following statement:

> The space needed, in any proof system working with clauses, in order to refute a CNF-formula X, is always lower bounded by the tableau width of X minus 2.

But what we mean by "the space needed, in any proof system working with clauses, in order to refute a CNF-formula"? Consider a CNF-formula X and a refutation system with syntactic rules by which we can produce clauses from clauses. Suppose we have a set $\mathcal{M}$ viewed as the memory. Initially, $\mathcal{M} = \emptyset$. At any stage of a refutation of X, we can either add a clause C of X to $\mathcal{M}$, erase a clause from $\mathcal{M}$, or add to $\mathcal{M}$ a clause which can be inferred using the rules of the system from clauses already in $\mathcal{M}$. The smallest amount of memory we can use in order to reach a contradiction (when working with clauses, a contradiction is just the empty clause $[\,]$) is the space needed to refute X. This is made precise in the following definitions of [2].

Definition 3.1.1. A *configuration* is a set of clauses. A *derivation* π from a CNF X is a sequence of configurations $\mathcal{M}_0, \ldots, \mathcal{M}_s$ such that $\mathcal{M}_0 = \emptyset$ and, for all $i \in \{1, \ldots, s\}$, $\mathcal{M}_i$ is obtained from $\mathcal{M}_{i-1}$ by one of the following rules:

Axiom download. $\mathcal{M}_i \coloneqq \mathcal{M}_{i-1} \cup \{C\}$ for a clause C of X.

Erasure. $\mathcal{M}_i \coloneqq \mathcal{M}_{i-1} - \{C\}$ for a clause $C \in \mathcal{M}_{i-1}$.

Inference. $\mathcal{M}_i \coloneqq \mathcal{M}_{i-1} \cup \{C\}$ for a clause C such that $\mathcal{M}_{i-1} \models C$.

If $[\,] \in \mathcal{M}_s$ then the derivation is called a *refutation* of X.

Definition 3.1.2. The *clause space* of a sequence of configurations $\pi = \mathcal{M}_0$, $\ldots, \mathcal{M}_s$ is

$$\mathrm{CSpace}(\pi) \stackrel{\text{def}}{=} \max\{|\mathcal{M}_i| : i \in \{0, \ldots, s\}\}.$$

The clause space of an unsatisfiable CNF-formula X is

$$\mathrm{CSpace}(X) \stackrel{\text{def}}{=} \min\{\mathrm{CSpace}(\pi) : \pi \text{ is a refutation of } X\}.$$

Note that the inference case of Definition 3.1.1 allows us to talk about the space needed in any refutational proof system working with clauses.

Remark 3.1.1. What we defined as clause space is often called in the literature *semantic clause space*, whereas the term clause space is used to denote the resolution clause space. We chose to call the semantic clause space simply clause space, in order to avoid unnecessary terminology. As a matter of fact, this is of little importance, since resolution clause space and semantic clause space are linearly related (see [2]).

A key, immediate fact about clause space is the following lemma of [2]. For two sets of formulas S and T, we write $S \models T$ if every assignment that makes all formulas in S true, also makes all formulas in T true.

Lemma 3.1.1 (Locality Lemma)**.** *Let S be a satisfiable set of literals, and T a set of clauses. If $S \models T$, then there is a set $S' \subseteq S$ of size $|S'| \leq |T|$ such that $S' \models T$.*

Proof. For each clause $C \in T$, there is a literal $L_C \in S$ contained in C (otherwise, if there was a clause $C \in T$ containing no literals in S, we could at the same time satisfy S and make C false). Setting $S' \coloneqq \{L_C : C \in T\}$ we have the lemma. □

Finally, we have the following theorem (the proof of which is similar to the proof of theorem 3 in [3], or of theorem 3.13 in [2]).

Theorem 3.1.1. *For any unsatisfiable CNF-formula X,*

$$\mathit{TWidth}(X) \leq \mathit{CSpace}(X) + 2.$$

Proof. Suppose that $\mathrm{TWidth}(X) > w \geq 1$. This means that there exists a w-consistency property $\mathcal{C}$ closed under subsets, containing X. We show that there cannot be a refutation of X of space at most $w-2$. More specifically, let $\pi = \mathcal{M}_0, \ldots, \mathcal{M}_s$ be a derivation from X of space at most $w-2$. We show that for every $\mathcal{M}_i$ there exists a satisfiable set of literals S_i such that $S_i \cup \{X\} \in \mathcal{C}$ and $S_i \models \mathcal{M}_i$. It follows that no $\mathcal{M}_i$ can be the empty clause, hence π cannot be a refutation.

Base case. $\mathcal{M}_0 = \emptyset$, and $S_0 \coloneqq \emptyset$ is a satisfiable set of literals such that $S_0 \cup \{X\} \in \mathcal{C}$ and $S_0 \models \mathcal{M}_0$.

Inductive step. Suppose that $i > 0$. From the induction hypothesis, there is a satisfiable set S_{i-1} such that $S_{i-1} \cup \{X\} \in \mathcal{C}$ and $S_{i-1} \models \mathcal{M}_{i-1}$. We have the three cases:

Axiom Download. $\mathcal{M}_i = \mathcal{M}_{i-1} \cup \{C\}$ for a clause C of X not in $\mathcal{M}_{i-1}$. From hypothesis, $|\mathcal{M}_i| \leq w-2$ so $|\mathcal{M}_{i-1}| \leq w-3$. Since S_{i-1} is satisfiable and $S_{i-1} \models \mathcal{M}_{i-1}$, from the locality lemma there is a set $S'_{i-1} \subseteq S_{i-1}$ such that $S'_{i-1} \models \mathcal{M}_{i-1}$ and $|S'_{i-1}| \leq w-3$. Moreover, since $S_{i-1} \cup \{X\} \in \mathcal{C}$ and $\mathcal{C}$ is closed under subsets, we have that $S'_{i-1} \cup \{X\} \in \mathcal{C}$. Now $\mathcal{C}$ is a w-consistency property and $|S'_{i-1} \cup \{X\}| \leq w-2$, so we have that

$$
\begin{aligned}
S'_{i-1} \cup \{X\} \cup \{C\} &\in \mathcal{C}, && \text{from condition 3 of Definition 2.4.1} \\
S'_{i-1} \cup \{X\} \cup \{C\} \cup \{L\} &\in \mathcal{C}, && \text{from condition 4 of Definition 2.4.1} \\
S'_{i-1} \cup \{X\} \cup \{L\} &\in \mathcal{C}, && \text{because } \mathcal{C} \text{ is closed under subsets,}
\end{aligned}
$$

where L is a literal contained in C. Now from the condition 1 of Definition 2.4.1, the set $S_i \coloneqq S'_{i-1} \cup \{L\}$ is satisfiable and $S_i \cup \{X\} \in \mathcal{C}$, $S_i \models \mathcal{M}_i$.

Erasure. $\mathcal{M}_i = \mathcal{M}_{i-1} - \{C\}$ for a clause $C \in \mathcal{M}_{i-1}$. The set $S_i \coloneqq S_{i-1}$ is a set such that $S_i \cup \{X\} \in \mathcal{C}$ and $S_i \models \mathcal{M}_i$.

Inference. $\mathcal{M}_i = \mathcal{M}_{i-1} \cup \{C\}$ for a clause C such that $\mathcal{M}_{i-1} \models C$. Again, since $S_{i-1} \models \mathcal{M}_{i-1}$ and $\mathcal{M}_{i-1} \models C$, the set $S_i \coloneqq S_{i-1}$ is a set such that $S_i \cup \{X\} \in \mathcal{C}$ and $S_i \models \mathcal{M}_i$. □

3.2 Atomic cuts and resolution

Resolution is a system for refuting CNF-formulas. It can be converted into a polynomially equivalent proof system for general propositional formulas, by employing an efficient translation to CNF, first used in [35]: Any formula X can be transformed in polynomial time to a CNF X', so that X is satisfiable if and only if X' is satisfiable.[1]

[1] This can also be seen to be true from Cook's theorem [13]: Any NP set, in particular the set of all satisfiable formulas, can be reduced in polynomial time to the set of all satisfiable CNF-formulas.

Clauses are represented within the system as *sets of literals*. The only rule of inference is the *resolution rule*,

$$\frac{C \cup \{P\} \qquad D \cup \{\neg P\}}{C \cup D}.$$

We say that $C \cup D$ resulted from $C \cup \{P\}$ and $D \cup \{\neg P\}$ *resolving on* P.

Definition 3.2.1. A *resolution derivation* from a CNF X is a sequence $C_1, \ldots, C_s$ of clauses (represented as sets of literals) such that for each $i \in \{1, \ldots, s\}$, either C_i is a clause of X, or is the result of the resolution rule on two previous clauses. If C_s is the empty set, then the derivation is called a *resolution refutation* of X.

Given a resolution derivation $\pi = C_1, \ldots, C_s$, we can define a DAG $G_\pi = (V_\pi, E_\pi)$ with s vertices $V_\pi = \{x_1, \ldots, x_s\}$, and edges given by the relation E_π, where

$$E_\pi(x_i, x_j) \iff C_i \text{ is one of the two premises from which } C_j \text{ was derived.}$$

If G_π is a tree then we say that π is in *tree-like* form.

Tree-like resolution refutations can be seen as a special case of tableau refutations using only atomic cuts (i.e., cuts to propositional variables) except at the bottom levels.[2] This is demonstrated in Figure 3.1, where for convenience we omitted commas and brackets. In words, let π be a tree-like resolution refutation of X and let T be its corresponding tree. Label the root of T by X. For every node which resulted from its children resolving on, say P, label the child the clause of which contains P with $\neg P$, and the other child with P. Call the resulting tree T'. We can easily see that for every branch θ of T', the set of formulas that occur in θ falsifies a clause of X, namely the clause associated with the leaf of T corresponding to θ, so we can add that clause at the end of θ and expand it, closing θ.

Actually, tableaux with atomic cuts and tree-like resolution are two p-equivalent proof systems. This is a corollary of the fact that the cut rule can efficiently simulate the tableau expansion rules; more precisely, is a corollary (why?) of the following lemma.

Lemma 3.2.1. *For any formula X and any tableau (even with cuts) $\mathbf{T}$ for X, there is a tableau $\mathbf{T}_r$ for X such that*

1. *$\mathbf{T}_r$ is using only cuts except at the bottom levels;*
2. *$|\mathbf{T}_r| \leq 3 \cdot |\mathbf{T}|$;*
3. *there is a one-to-one mapping f between the branches of $\mathbf{T}$ and $\mathbf{T}_r$, such that $\Gamma_\theta \subseteq \Gamma_{f(\theta)}$ and for every non-closed branch τ of $\mathbf{T}_r$ there is a branch θ such that $\tau = f(\theta)$;*

[2] More accurately, on each branch every application, except the last one, of a tableau rule is the application of the cut rule on a propositional variable.

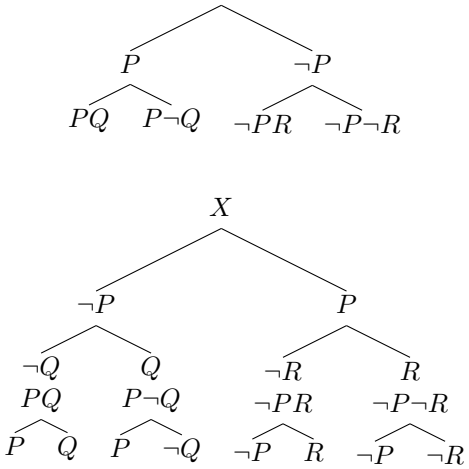

Figure 3.1: A resolution refutation of the formula $X = \langle [P, Q], [P, \neg Q], [\neg P, R], [\neg P, \neg R] \rangle$ (up) and the corresponding closed tableau (down).

4. *every cut-formula in* $\mathbf{T}_r$ *is either a formula or the negation of a formula that occurs in* $\mathbf{T}$.

Recall that Γ_θ *is the set of formulas that occur in* θ.

Proof. The proof is by induction on the construction of $\mathbf{T}$. The base case is trivial, so suppose that we have already constructed a tableau $\mathbf{T}'_r$ for $\mathbf{T}'$ satisfying the lemma's conditions, and $\mathbf{T}$ resulted from $\mathbf{T}'$ by the expansion of a β-formula (the other cases are similar, see Figure 3.2) on a branch θ' of $\mathbf{T}'$. Let τ' the branch of $\mathbf{T}'_r$, guaranteed by the induction hypothesis, such that $\Gamma_{\theta'} \subseteq \Gamma_{\tau'}$. To construct $\mathbf{T}_r$, simply append to the end of τ' the rightmost tree of Figure 3.2 (erasing the root label β) to the end of τ'. It is easy to see that the resulting tableau satisfies the conditions 1, 3 and 4 of the lemma, and its size is

$$|\mathbf{T}'_r| + 3k \leq 3 \cdot (|\mathbf{T}'| + k) = 3 \cdot |\mathbf{T}|,$$

completing the proof. □

Resolution depth

The *depth* of a resolution derivation π is the length of the longest path from a leaf to a root of G_π *minus one* (we don't want to count the root). The *resolution depth*, $\mathrm{RDepth}(X)$, of an unsatisfiable CNF-formula X is the minimum depth of a resolution refutation of X:

$$\mathrm{RDepth}(X) \stackrel{\text{def}}{=} \min\{\mathrm{RDepth}(\pi) : \pi \text{ is a resolution refutation of } X\}.$$

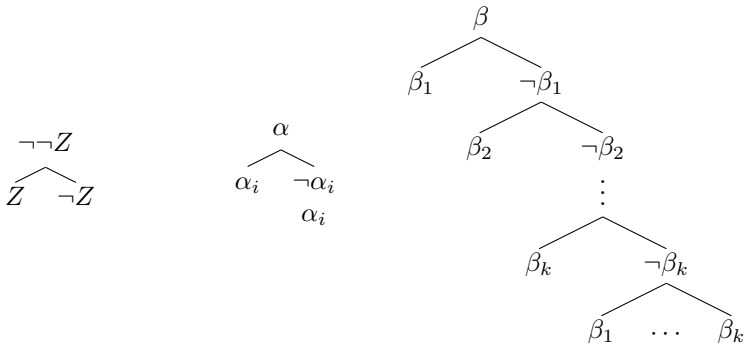

Figure 3.2: A simulation of the tableau expansion rules by the cut rule.

Since, by repeating subderivations, we can turn any resolution refutation to a tree-like one without increasing the depth, this minimum could be taken over all tree-like refutations. The transformation is described in more detail in the following proposition.

Proposition 3.2.1. *A depth d resolution derivation of a clause C from a CNF X can be transformed into a depth d tree-like resolution derivation of C from X.*

Proof. Let $C_1, \ldots, C_s$ be a resolution derivation from X. We show that for every $i \in \{1, \ldots, s\}$, C_i has a tree-like resolution derivation from X of depth d_i, where d_i is the depth of the subderivation $C_1, \ldots, C_i$.

If C_i is a clause of X, then C_i is a tree-like derivation from X and we are done. Now suppose that C_i was derived from the clauses C_j and C_k. From the induction hypothesis, there are tree-like derivations π_j and π_k of C_j and C_k with depths d_j and d_k respectively. $\pi_k \circ \pi_j \circ C_i$, where $\circ$ denotes sequence concatenation, is a depth d_i tree-like derivation of C_i from X. □

Remark 3.2.1. Even though the above construction keeps the depth invariant, it induces redundacies, increasing the proof size. Indeed, it is shown in [6] that resolution is exponentially more powerfull than tree-like resolution in terms of size, demonstrating a family of formulas (which are a variant of the pebbling contradictions of Definition 2.8.2) of size n that have $O(n)$-size resolution refutations, but require $\exp(\Omega(n/\log n))$-size tree-like resolution refutations.

We have the following version of Definition 2.4.1 for characterizing the resolution depth of an unsatisfiable CNF-formula. Theorem 3.2.1 is from [38] (presented here in slightly different terms); we omit its proof, which is, given the translation of Figure 3.1, a variation of the proof of Theorem 2.4.1. But first, some handy notation: For a propositional variable P, let $\overline{P} := \neg P$ and $\overline{\neg P} := P$. Also, for a formula X, we denote by $V(X)$ the set of variables that occur in X.

Definition 3.2.2. Let X be a CNF-formula and d a non-negative integer. We say that X has the *d-resolution consistency property* if there exists a collection $\mathcal{C}$ of sets of literals such that $\emptyset \in \mathcal{C}$ and for each $S \in \mathcal{C}$:

1. For each clause C of X, S does not falsify C, i.e., for any subset $S' \subseteq \{\overline{L} : L \in S\}$, C is not the disjunction of S'.
2. $P \in V(X)\ \ \&\ \ |S| < d \implies S \cup \{P\} \in \mathcal{C}\ \ \text{or}\ \ S \cup \{\neg P\} \in \mathcal{C}$.

Theorem 3.2.1 (Urquhart [38])**.** *For any CNF X, X does not have a resolution refutation of depth at most d if and only if X has the d-resolution consistency property.*

The version of the Prover-Adversary game corresponding to Definition 3.2.2 is as follows: Let X be the CNF-formula the game is played on, and S the set maintained by the two players. Initially, $S \coloneqq \emptyset$. In any round, the Prover selects a variable $P \in V(X)$ and the Adversary must respond with either P or $\neg P$. In the first case P is added to S, while in the second case $\neg P$ is added to S. If at any stage of the game the set of literals S falsifies a clause C of X then the Prover wins.

We have the following relationship between tableau and resolution depth. In order to separate the games characterizing the tableau depth and the resolution depth, we refer to the first as the *tableau depth game* and to the second as the *resolution depth game*. Also, we call the players of the tableau depth game Prover A and Adversary A, and the players of the resolution depth game Prover B and Adversary B.

Proposition 3.2.2. *For any unsatisfiable r-CNF formula,*

$$\mathit{RDepth}(X) \leq r \cdot \mathit{TDepth}(X).$$

Proof (sketch). Given a winning strategy of Prover A which lasts for at most d rounds, we show how Prover B can simulate this strategy in at most $r \cdot d$ rounds. Consider an arbitrary round of the tableau depth game, and say that Prover A has selected a clause C of X, waiting for Adversary A to respond with a literal contained in C. Then Prover B, selects one by one, all the variables in $V(C)$, until Adversary B selects a literal L contained in C; if Adversary B selected for each variable $P \in V(C)$ the opposite of C's literal corresponding to P, then clause C is falsified and Prover B wins. Afterwards Prover A continues provided that Adversary A selected literal L and the simulation goes on. □

Remark 3.2.2. The above construction corresponds to the translation of tableau proofs into resolution proofs of Lemma 3.2.1 (see also Theorem 5.1 of [37]).

An example of a family of formulas, due to Cook (see [15, Section V] and also [37, page 432]), for which the resolution depth is much smaller than the tableau depth is the following: Let T_n be a complete binary tree with $n+1$ levels in which the interior nodes are labelled with distinct variables. We associate a CNF-formula $\Sigma(T_n)$ with T_n, in such a way that each branch θ in T_n has a

clause C_θ of $\Sigma(T_n)$ associated with it. The variables in C_θ are those labelling the nodes in θ; if P is such a variable, then P is included in C_θ if θ branches to the left below the node labelled with P, otherwise C_θ contains $\neg P$. Figure 3.2 shows an example.

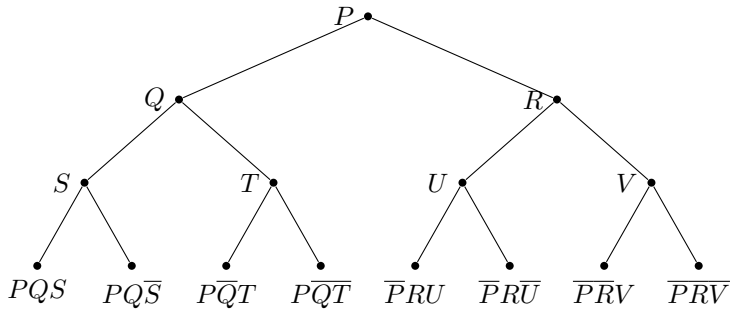

Figure 3.3: $\Sigma(T_3)$ is the formula $\langle[PQS], [PQ\overline{S}], [P\overline{Q}T], [P\overline{QT}], [\overline{P}RU], [\overline{P}R\overline{U}], [\overline{PR}V], [\overline{PRV}]\rangle$.

We can see that T_n forms a resolution refutation of $\Sigma(T_n)$, so for every $n > 0$,

$$\mathrm{RDepth}(\Sigma(T_n)) \leq n.$$

On the other hand, $\mathrm{TDepth}(\Sigma(T_n))$ is as large as it can be.

Proposition 3.2.3. *For every $n > 0$,*

$$\mathit{TDepth}(\Sigma(T_n)) > 2^n.$$

Proof. The proof is by induction on n. The base case is true, since

$$\mathrm{TDepth}(\Sigma(T_1)) > 2.$$

For the inductive step, suppose that $n > 1$. Let P be the variable labelling the root of T_n and let T^0 and T^1 be the immediate left and right respectively subtrees of T_n. Since T^0 and T^1 are binary trees with n levels, we have by the induction hypothesis that

$$\mathrm{TDepth}(\Sigma(T^0)) > 2^{n-1}$$

and

$$\mathrm{TDepth}(\Sigma(T^1)) > 2^{n-1},$$

meaning that the Adversary has winning strategies in the tableau depth game played on $\Sigma(T^0)$ and $\Sigma(T^1)$ for at most $2^{n-1} - 1$ rounds. Fix a strategy $\mathcal{C}^0$ for $\Sigma(T^0)$ and a strategy $\mathcal{C}^1$ for $\Sigma(T^1)$. We construct a winning strategy for the Adversary in the tableau depth game played on $\Sigma(T_n)$ for at most $2^n - 1$ rounds.

The Adversary selects literals different from P and $\neg P$ until he is no longer able to do this. More specifically, if the Prover selects a clause corresponding to a leaf of T^0, then the Adversary selects a literal according to the strategy $\mathcal{C}^0$; if he cannot do this, he selects P. Similarly, if the Prover selects a clause corresponding to a leaf of T^1, then the Adversary selects a literal according to the strategy $\mathcal{C}^1$, and if he cannot do this, he selects $\neg P$. Playing this way, the Prover can force the Adversary to select P only when he has selected all the clauses corresponding to the leaves of T^0 after 2^{n-1} rounds, and similarly for $\neg P$. Since the sets of variables of T^0 and T^1 are disjoint, the above strategy is a winning strategy for the Adversary in the tableau depth game played on $\Sigma(T_n)$ for $2^n - 1$ rounds. □

Resolution width

The *width* of a resolution derivation $\pi = C_1, \ldots, C_s$ is the size of the largest clause in π,

$$\mathrm{RWidth}(\pi) \stackrel{\mathrm{def}}{=} \max\{|C_i| : i \in \{1, \ldots, s\}\}.$$

The *resolution width* of an unsatisfiable CNF-formula X is the minimum width of a resolution refutation of X,

$$\mathrm{RWidth}(X) \stackrel{\mathrm{def}}{=} \min\{\mathrm{RWidth}(\pi) : \pi \text{ is a resolution refutation of } X\}.$$

A major insight of [3] was to show that the following theorem holds.

Theorem 3.2.2 (Atserias and Dalmau [3])**.** *Let X be an r-CNF formula and w an integer such that $r \leq w$. Then X does not have a resolution refutation of width at most w if and only if there exists a $(w+1)$-resolution consistency property closed under subsets for X.*

Proof. (*Soundness*) Let $\mathcal{C}$ be a $(w+1)$-resolution consistency property closed under subsets for X. Let $\pi = C_1, \ldots, C_s$ be a resolution derivation from X, of width at most w. We show that for every set $S \in \mathcal{C}$ and every clause C_i in π, S does not falsify C_i. Since $\mathcal{C}$ is non-empty, it follows that π cannot contain the empty clause, therefore cannot be a refutation.

> *Base case.* If C_i is a clause of X, then for every $S \in \mathcal{C}$, S cannot falsify C_i, from the condition 1 of Definition 3.2.2.
>
> *Inductive step.* Suppose that the clause $C_i = C \cup D$ results from the clauses $C \cup \{P\}$ and $D \cup \{\neg P\}$. Assume, for the sake of contradiction, that there is a set $S \in \mathcal{C}$ falsifying C_i. Let S' be the minimal subset of S falsifying C_i. Since $\mathcal{C}$ is closed under subsets $S' \in \mathcal{C}$. Moreover, since $|C_i| \leq w$, $|S'| \leq w$. From the condition 2 of Definition 3.2.2, either $S' \cup \{P\} \in \mathcal{C}$, or $S' \cup \{\neg P\} \in \mathcal{C}$. In the first case $S' \cup \{P\}$ falsifies $D \cup \{\neg P\}$ and in the second case $S' \cup \{\neg P\}$ falsifies $C \cup \{P\}$, contradicting the induction hypothesis.

(*Completeness*) Suppose that there is no resolution refutation of X of width at most w. Let $\mathcal{C}$ be the set of all clauses having a resolution derivation from X of width at most w. Set

$$C' \coloneqq \{S \text{ set of literals} : \text{for each } C \in \mathcal{C},\ S \text{ does not falsify } C\}.$$

We show that $\mathcal{C}'$ is a $(w+1)$-resolution consistency property closed under subsets for X. First, $\emptyset \in \mathcal{C}'$ since $\mathcal{C}$ does not contain the empty clause. Secondly, a subset of a set $S \in \mathcal{C}'$ cannot falsify a clause in $\mathcal{C}$, so $\mathcal{C}'$ is closed under subsets. For the first condition of Definition 3.2.2, X is an r-CNF and $r \leq w$, so all the clauses of X belong to $\mathcal{C}$, and for any such clause there is no set in $\mathcal{C}'$ falsifying it. For condition 2, let S be any set in $\mathcal{C}'$ such that $|S| \leq w$ and P a variable. Assume, for the sake of contradiction, that $S \cup \{P\}$ falsifies $C \in \mathcal{C}$ and $S \cup \{\neg P\}$ falsifies $D \in \mathcal{C}$. It follows that $C = E \cup \{\neg P\}$, since otherwise S would falsify C, and similarly $D = F \cup \{P\}$. But then S must falsify $E \cup F$, and since $|S| \leq w$, it must be that $|E \cup F| \leq w$. Therefore, from the fact that C and D are in $\mathcal{C}$, it follows that $E \cup F$ is in $\mathcal{C}$, which is a contradiction. □

Of course, the closure under subsets condition corresponds to the ability of the Prover to forget formulas. More specifically, consider the width game, where the Prover is selecting variables and tries to reach a set falsifying a clause of the r-CNF ($r \leq w$) X; we call this game *resolution width game*. From Theorem 3.2.2, we have that the Prover wins the $(w+1)$-width resolution game if and only if $\mathrm{RWidth}(X) \leq w$. Similar to Proposition 3.2.2, we have the following relationship with tableau width.

Proposition 3.2.4. *For any unsatisfiable r-CNF formula X,*

$$\mathit{RWidth}(X) \leq \mathit{TWidth}(X) + r - 1.$$

Proof (sketch). Given a winning strategy of Prover A in the w-width tableau game played on X, we show how Prover B can simulate this strategy, winning the $(w+r)$-width resolution game. Consider an arbitrary round of the tableau width game, and say that Prover A has selected a clause C of X, waiting for Adversary A to respond with a literal contained in C. Then Prover B, selects one by one, all the variables in $V(C)$, until Adversary B selects a literal L contained in C; if Adversary B selected for each variable $P \in V(C)$ the opposite of C's literal corresponding to P, then clause C is falsified and Prover B wins. Afterwards Prover B forgets all the unnecessary literals (those that Prover A doesn't have in his memory) corresponding to variables in $V(C)$ except L and Prover A continues provided that Adversary A selected literal L. Whenever Prover A forgets a literal L, Prover B also forgets L. □

From Theorem 3.1.1, we have the following major theorem of [3].

Corollary 3.2.1. *For any unsatisfiable r-CNF X,*

$$\mathit{RWidth}(X) \leq \mathit{CSpace}(X) + r + 1.$$

Open Problem. Does there exist a family of formulas separating resolution width from tableau width? To put it in other words, do atomic cuts shorten the width of proofs?

3.3 From large depth to large size

While for both tableaux and resolution, width lower bounds imply size lower bounds (see Chapter 4), depth lower bounds are of no help in proving size lower bounds. It may be the case that a formula has large tableau (or resolution) depth, but has a closed tableau (or a resolution refutation) of small size. The formulas X_n of Example 2.8.2 provide such an example. When the cut rule is forbidden, they require $\Omega(n)$ tableau depth, but there is a closed tableau for X_n of size $O(n^2)$. For an even simpler example take the formula

$$\underbrace{\neg \dots \neg}_{2n}(P \wedge \neg P).$$

We demonstrate in this section, how we can get a formula that requires large size from a formula that requires large depth.

First, an observation about tableau size. Suppose that the Prover-Adversary game is played on X and for every size d sequence of Prover's questions, the Adversary has at least k different winning strategies. More specifically, suppose that for at least k β-formulas, the Adversary has more than one choices. We will show that this implies that every tableau of depth d for X has size at least 2^k. Following [31], we give the ability to the Adversary to select, instead of a single component β_i, a set of components. Each time the Adversary selects such a set B with $|B| \geq 2$ he scores one point, and then the Prover selects a formula $X \in B$ and updates to $S \coloneqq S \cup \{X\}$. As usual, Prover wins if he reaches a set S containing a formula and its negation.

Proposition 3.3.1. *If the set S_0 has a closed tableau of size t, then the Prover has a strategy which forces the Adversary to score at most $\lceil \log t \rceil$ points.*

Proof. Let $\mathbf{T}$ be a closed tableau for S_0 of size t. The Prover follows a branch of $\mathbf{T}$ beginning after S_0. He keeps the invariant:

> If θ is the path already traversed, then the current set S of the game contains exactly the formulas occurring in θ. Moreover, if the Adversary has scored k points, then the subtree of $\mathbf{T}$ rooted at the current node has size at most $s/2^k$.

So every time the Adversary scores a point, the size of the current subtree is reduced by at least a factor of two. Since $\mathbf{T}$ is closed, when the Prover reaches a subtree of size 1, the set S will contain a formula and its negation, and therefore the Adversary cannot score more than $\lceil \log t \rceil$ points.

At the beginning $k = 0$ and the invariant is obviously true. Now suppose that θ is the path already traversed and the invariant is true. If θ doesn't split

at the current node, then the next node x to be visited is the result of either a double negation rule, or an α-rule. In either case, the Prover may add the formula labelling x to S, and since Adversary doesn't score any points, the invariant is maintained. Next suppose that θ splits at the current node to the components of a formula β in θ. Then Prover selects β. If the Adversary selects exactly one component β_i, then Prover goes to the node labelled by β_i. The invariant is maintained since adversary doesn't score any points. If the Adversary selects more than one components, then the Prover selects the component corresponding to the smallest subtree, and goes to the root of that tree. Adversary scores one point, and since the size of this subtree is at most half the size rooted at the previous node, the invariant is maintained. □

Remark 3.3.1. We should note here that one can get, using a similar game to the above, a characterization of the order of tableau size, along the lines of [9].

Now to our main result. The transformation turning large tableau depth to large tableau size is the following. Being so naive, this transformation manifests the inefficiency of cut-free tableaux.

Definition 3.3.1. Let f be a function mapping formulas to formulas, such that $f(L) = L \vee L$ for every literal L, and

$$\begin{aligned} f(\neg\neg Z) &= \neg\neg f(Z) \vee \neg\neg f(Z);\\ f(\alpha) &= \langle f(\alpha_1), \ldots, f(\alpha_k)\rangle \vee \langle f(\alpha_1), \ldots, f(\alpha_k)\rangle;\\ f(\beta) &= [f(\beta_1), \ldots, f(\beta_k)] \vee [f(\beta_1), \ldots, f(\beta_k)]. \end{aligned}$$

Theorem 3.3.1. *For any formula X, if TDepth$(X) > d+1$, then every tableau for $f(X)$ must have size at least 2^d.*

Proof (sketch). Suppose that $\mathrm{TDepth}(X) > d+1$. This means that the Adversary wins, when the depth game is played on X for at most d round. We show that the Adversary can always score d points when the game played on $f(X)$.

Define c as the function mapping every formula to a set of formulas such that $c(L) = \{L\}$ for every literal L, and

$$\begin{aligned} c(\neg\neg Z) &= c(Z) \cup \{\neg\neg Z\};\\ c(\alpha) &= \bigcup\{c(\alpha_i) : \alpha_i \text{ is a component of } \alpha\} \cup \{\alpha\};\\ c(\beta) &= \bigcup\{c(\beta_i) : \beta_i \text{ is a component of } \beta\} \cup \{\beta\}. \end{aligned}$$

It is easy to show, by structural induction, that

1. the formulas selected by the two players when the Prover-Adversary game is played on X, must belong to the set $c(X)$;

2. a formula in $c(f(X))$ is either of the form $f(Z)$, or one of the two components of $f(Z)$, for some $Z \in c(X)$.

Having said that, consider the game played on $f(X)$. Whenever the Prover selects a formula $f(Z)$, then the Adversary selects both its components and he scores one point. We claim that the Adversary can play so that the Prover must select at least d such formulas. This is true, because of the fact that the Prover cannot win the depth game on X in the course of d rounds. The Prover of the depth game on X can simulate a strategy of the Prover on $f(X)$ in a way that every time the second Prover selects a component of a formula $f(Z)$, then the first Prover selects Z. The details are left to the reader. □

Remark 3.3.2. Notice that Theorem 3.3.1 makes sense when the size of $f(X)$ is polynomial to the size of X.

Remark 3.3.3. The formulas of [34], showing that cuts shorten proofs, are very similar to the formulas $f(X_n)$, where X_n are the formulas of Example 2.8.2. First of all, notice that the size of $f(X_n)$ is $O(n^2)$. Now we saw that $\mathrm{TDpeth}(X_n) > n$, hence from Theorem 3.3.1 $f(X_n)$ requires tableau size at least 2^n. On the other hand, when the cut rule is allowed, $f(X_n)$ has a closed tableau of size $O(n^2)$ (see [34], [21, Section 8.10]).

In the case of resolution, the corresponding transformation, as described in [38], is the following. $P_1 \oplus P_2$ stands for the formula $(P_1 \wedge \neg P_2) \vee (\neg P_1 \wedge P_2)$. For the proof of the Theorem 3.3.2 see [38]. The idea is very similar to that of Theorem's 3.3.1 proof.

Definition 3.3.2. Let X be a CNF-formula, and $\{P_1, P_2\}$ a pair of variables associated with each distinct variable in $V(X)$. Then the *xorification* of X, written $X^{\oplus}$, is defined to be the conjunctive normal form of the formula obtained from X by substituting $P_1 \oplus P_2$ for each variable $P \in V(X)$.

Theorem 3.3.2 (Urquhart [38])**.** *If $RDepth(X) \geq k$, then any tree resolution refutation of $X^{\oplus}$ has size at least 2^k.*

3.4 General cuts and Frege systems

We consider now tableaux with cuts to arbitrary formulas. Let us repeat Definition 2.4.1 in this setting.

Definition 3.4.1. A d-Frege consistency property is a collection $\mathcal{C}$ of sets of formulas such that for each $S \in \mathcal{C}$:

1. For each formula X, not both $X \in S$ and $\neg X \in S$.
2. $\neg\neg Z \in S \ \ \& \ \ |S| < d \implies S \cup \{Z\} \in \mathcal{C}$.
3. $\alpha \in S \ \ \& \ \ |S| < d \implies S \cup \{\alpha_i\} \in \mathcal{C}$ for every component α_i of α.
4. $\beta \in S \ \ \& \ \ |S| < d \implies S \cup \{\beta_i\} \in \mathcal{C}$ for some component β_i of β.
5. $|S| < d \implies S \cup \{X\} \in \mathcal{C}$ or $S \cup \{\neg X\} \in \mathcal{C}$ for any formula X.

Of course, a formula X has the d-Frege consistency property if and only if there is no closed tableau with cuts to arbitrary formulas for X of depth at most d. In the corresponding Prover-Adversary game, which we call the *Frege game*, the Prover can at any time select an arbitrary formula Y and the Adversary must respond with Y or $\neg Y$. We will prove in this section the following statement, due to [30], phrased in the tableau setting, that for any tautology X,

> the minimum number of rounds needed for the Prover to win the Frege game on $\neg X$ (which is equal to the minimum depth of a closed tableau with cuts for $\neg X$ minus one) is at most the logarithm of the minimum number of steps in a Frege proof of X plus a constant.

But first, what is a Frege proof? Frege proof systems are the standard textbook proof systems; typically such a system consists of a couple of axioms and modus ponens as the only rule of inference.

Definition 3.4.2. An *axiomatic system* is defined as a finite set of local, sound rules of the form

$$\frac{A_1 \quad \dots \quad A_k}{B}.$$

If $k = 0$ for some rule, the rule is called an *axiom*. The rules are faced as *schemes*, meaning that when applying them we use their substitutional instances, where a substitutional instance of a rule is the result of uniformly substituting the propositional variables of the rule by arbitrary formulas.

Definition 3.4.3. Let S_0 be a set of formulas and F an axiomatic system. A proof of a formula X from S_0 in F is a sequence $X_1, \dots, X_s$ of formulas such that X_s is X and for each $i \in \{1, \dots, s\}$, X_i is either an element of S_0, or follows from a constant number (zero if the rule is an axiom) of previous formulas by an inference rule. We say that X has a proof in F, if it has a proof in F from $\emptyset$.

Definition 3.4.4. An axiomatic system F is called *implicationally complete* if there is a proof in F of X from S_0 whenever $S_0 \models X$. A *Frege system* is an implicationally complete axiomatic system.

Frege systems provide a quite robust notion of a proof system. Much like what can be computed by a Turing machine in polynomial time does not change by adding extra features to the Turing machine, such as multiple tapes or random access to the tape (see e.g. [27]), formulas having short proofs in a Frege system still have short proofs if the system is enhanced. As it is shown in [32], any two Frege proof systems (even defined over different bases of connectives) are p-equivalent. It can be shown that any Frege system can p-simulate tableaux with the cut rule, and this, combined with Proposition 2.3.1 and the fact that tree-like and DAG-like Frege systems are p-equivalent (see Corollary 3.4.1), yields that tableaux with arbitrary cuts are p-equivalent to any Frege system.

Before proving the main theorem, let us mention the following obvious lemma.

Lemma 3.4.1. *Let F be a Frege system. There is a constant c, depending only on F, such that for any substitutional instance*

$$\frac{A_1 \quad \dots \quad A_k}{B}$$

of an inference rule in F, there is a closed, cut-free tableau for the set $\{A_1, \dots, A_k, \neg B\}$ of depth at most c.

Theorem 3.4.1 (Buss and Pudlák [30])**.** *For any tautology X, the minimum number of rounds needed for the Prover to win the Frege game played on $\neg X$ is at most the logarithm of the minimum number of formulas in a Frege proof of X plus a constant.*

Proof. Let $X_1, \dots, X_t$ be a Frege proof of X. We set $Y_i := \langle X_1, \dots, X_i \rangle$. Prover's strategy is a binary search on the sequence $Y_1, \dots, Y_t$. More specifically, the Prover starts by selecting the formula $Y_{\lceil t/2 \rceil}$. If the Adversary responds with $Y_{\lceil t/2 \rceil}$, the Prover continues recursively with the sequence $Y_{\lceil t/2 \rceil}, \dots, Y_t$, and if the Adversary responds with $\neg Y_{\lceil t/2 \rceil}$, the Prover continues with the sequence $Y_1, \dots, Y_{\lceil t/2 \rceil}$. Eventually, in at most $\lceil \log t \rceil$ rounds, a configuration S will be reached containing either Y_t, or $\neg Y_1$, or both Y_i and $\neg Y_{i+1}$ for some i, $1 \leq i < t$.

> *Case 1.* If $Y_t \in S$, then since Y_t is the formula $\langle X_1, \dots, X_t \rangle$, the Prover can add to S the formula X_t and a contradiction is reached, as $\neg X_t \equiv \neg X \in S$.[3]
>
> *Case 2.* If $\neg Y_1 \in S$, then the Prover may select $\neg Y_1 \equiv \neg \langle X_1 \rangle$, and the Adversary is obliged to select $\neg X_1$. Since X_1 is the substitutional instance of an axiom, from Lemma 3.4.1, the Prover can reach a contradiction in a constant number of rounds.
>
> *Case 3.* Finally, suppose that $Y_i \equiv \langle X_1, \dots, X_i \rangle \in S$ and $\neg Y_{i+1} \equiv \neg \langle X_1, \dots, X_i, X_{i+1} \rangle \in S$ for some i. Then the Prover first selects $\neg Y_{i+1}$. If the Adversary selects a formula $\neg X_j$, for $1 \leq j \leq i$, then the Prover can select X_j from Y_i, and a contradiction is reached. So suppose that the Adversary selects $\neg X_{i+1}$. Suppose that X_{i+1} was derived from $X_{i_1}, \dots, X_{i_k}$, $i_1, \dots, i_k \leq i$. Then the Prover can select $X_{i_1}, \dots, X_{i_k}$ from Y_i and force, by Lemma 3.4.1, a contradiction in a constant number of rounds. □

We saw that a tree-like resolution proof is a "normal form" of a tableau proof, in which the top levels consist exclusively of atomic cuts, and the tableau expansion rules of Table 2.2 occur at the bottom levels, only to falsify a clause of the formula to be refuted. Theorem 3.4.1 constructs tableau proofs of a

[3] We write $\equiv$ for the syntactic equivalence of two expressions.

similar form, where the top levels consist of cuts to arbitrary formulas, and each branch contains at its end a constant number of the tableau expansion rules, used only to demonstrate a local contradiction in the branch. In fact, and this statement is also from [30],

> the minimum depth of such a closed tableau for a formula $\neg X$ (or the minimum depth of any closed tableau with arbitrary cuts for $\neg X$, provided that X has only bounded disjunctions) is proportional to the logarithm of the minimum number of steps in a Frege proof of X.

Indeed, depth-d tableaux of these forms have at most $2^{O(d)}$ nodes and such a tableau can be formulated as a Frege proof with at most a polynomial increase in the number of steps.

This also shows that, unlike resolution, where proofs are exponentially more powerfull than tree-like proofs (see Remark 3.2.1), the tree-like restriction does not harm the efficiency of Frege proofs. Our proof is from [30].

Corollary 3.4.1 (Krajíček [24])**.** *Let F_1 be a Frege proof system and F_2 be a tree-like Frege proof system. Then F_1 and F_2 are p-equivalent.*

Proof (sketch). Take an arbitrary Frege proof of size n. First, transform it into a $O(\log n)$-step winning strategy of the Prover. Transforming the later into a tableau proof, we get a tableau of size $2^{O(\log n)} = n^{O(1)}$ in the form discussed above. Then one can check that the translation of this proof into a Frege proof can be done so that the tree form is preserved. □

Regarding the width of tableau proofs with arbitrary cuts, it is easy to see that it is always bounded by a constant.

Proposition 3.4.1. *When cuts to arbitrary formulas are allowed, then for any tautology X, there is a constant c, such that the Prover can win the c-width game played on $\neg X$.*

Proof. Let $X_1, \ldots, X_t$ be a Frege proof of X. Set $Y_i \coloneqq \langle X_1, \ldots, X_i \rangle$, and let c be the constant guaranteed by Lemma 3.4.1. The Prover begins by choosing Y_1. If Adversary responds with $\neg Y_1$, then, since Y_1 is an axiom, the Prover can force a contradiction in c rounds, and we are done. Otherwise the Prover continues with Y_2. If the the Adversary responds with Y_2, then the Prover forgets Y_1, and continues in the same way. At some point, while maintaining that the size of the current set S of formulas is at most $c+1$, the Prover will either reach Y_n, or Y_i and $\neg Y_{i+1}$ for some i. In the first case, a contradiction is reached since always $\neg X \in S$. In the second case, arguing as in the proof of Theorem 3.4.1, Y_i and $\neg Y_{i+1}$ will contain a local contradiction, which the Prover can find within c rounds. □

3.5 Lower bounds

Everyone familiar with complexity theory knows that proving lower bounds is a difficult task. Regarding propositional proofs, the first exponential lower bound on the size of resolution proofs was proved in 1985, almost 20 years since the first attempts. The Adversary arguments provide a conceptually viable way for showing lower bounds. We already saw several lower bounds for tableau depth. We will now show lower bounds on the tableau width of the formulas PHP_n^{n+1} and POP_n. The first is encoding the negation of the pigeon-hole principle, and the second the negation of the fact that a finite partial order has always a minimal element.

Definition 3.5.1. The formula PHP_n^{n+1} is defined as the conjunction of the following clauses:

1. $[P_{i1}, \ldots, P_{in}], \quad i \in \{1, \ldots, n+1\}$;
2. $[\neg P_{ik}, \neg P_{jk}], \quad i, j \in \{1, \ldots, n+1\},\ i \neq j,\ k \in \{1, \ldots, n\}$.

The intended meaning of the variable P_{ij} is that pigeon i is in hole j. The clauses 1 are saying that every pigeon must go to a hole, while the clauses 2 are saying that there cannot be a hole with two pigeons. PHP_n^{n+1} is unsatisfiable, beacause an assignment sending all the pigeons in $\{1, \ldots, n+1\}$ to a hole in $\{1, \ldots, n\}$ must send, from the pigeon-hole principle, two different pigeons to the same hole. Following lemma 6 of [3] we have the following proposition.

Proposition 3.5.1. *For any positive integer n,*

$$TWidth(PHP_n^{n+1}) > n.$$

Proof. Let $\mathcal{F}$ be the set of all partial one-to-one functions from $\{1, \ldots, n+1\}$ to $\{1, \ldots, n\}$. With every function $f \in \mathcal{F}$, we associate a set S_f of literals as follows.

$$S_f \coloneqq \{P_{ij} : f(i) = j\} \cup \{\neg P_{ij} : f(i) \text{ is defined but } f(i) \neq j\}.$$

Let $\mathcal{C}$ be the collection of all the sets S_f for $f \in \mathcal{F}$, each joined with PHP_n^{n+1} and all the clauses of PHP_n^{n+1},

$$\mathcal{C} \coloneqq \{S_f \cup \{\mathrm{PHP}_n^{n+1}\} \cup \{C : C \text{ is a clause of } \mathrm{PHP}_n^{n+1}\} : f \in \mathcal{F}\},$$

and let $\mathcal{C}^+$ be the closure of the set $\mathcal{C}$ under subsets. We claim that $\mathcal{C}^+$ is a n-consistency property. Since $\mathcal{C}^+$ is closed under subsets and $\{\mathrm{PHP}_n^{n+1}\} \in \mathcal{C}^+$, the proposition follows. Indeed, by the definition of $\mathcal{C}^+$, a set $S \in \mathcal{C}^+$ cannot contain a formula and its negation. Now let $S \in \mathcal{C}^+$ such that $|S| < n$ and say that the Prover chooses a clause C of PHP_n^{n+1}. Since $|S| < n$, there is an unoccupied hole, therefore there exists an extension of S in $\mathcal{C}^+$ containing a literal of C. □

Proposition 3.5.1 is tight, since the Prover has the following strategy for refuting PHP_n^{n+1} in $O(n)$ rounds: He first chooses one by one all the clauses of the form $[P_{i1}, \ldots, P_{in}]$. The Adversary is obliged to put two different pigeons into the same hole, i.e., to select P_{ik} and P_{jk} for some i, j, k with $i \neq j$. Then the Prover can force the Adversary to a contradiction by selecting the clause $[\neg P_{ik}, \neg P_{jk}]$.

Definition 3.5.2. The formula POP_n is defined as the conjunction of the following clauses:

No-minimal. $[P_{1i}, \ldots, P_{ni}], \quad i \in \{1, \ldots, n\}$.

Transitivity. $[\neg P_{ij}, \neg P_{jk}, P_{ik}], \quad i, j, k \in \{1, \ldots, n\}$.

Irreflexivity. $\neg P_{ii}, \quad i \in \{1, \ldots, n\}$.

The intended meaning of the variable P_{ij} is that i is smaller than j, which we visualize as an edge from vertex i to vertex j. POP_n is unsatisfiable due to the fact that a finite partial order can be turned into a linear order, or in computer science jargon "you can always topologically sort a finite DAG". More specifically, take an arbitrary element i. Because of the no-minimal clauses, i must have a smaller element, say i'. In turn i' must have a smaller element i'' and so on. Since the universe of elements is finite, at some point we will reach a cycle. The transitivity clauses then would force an element to be smaller than itself, something which contradicts the irreflexivity clauses.

Proposition 3.5.2. *For any positive integer n,*

$$TWidth(POP_n) > n - 1.$$

Proof. With every directed graph G on the vertices $V(G) = \{1, \ldots, n\}$, we associate a set of literals S_G as follows.

$$S_G \coloneqq \{P_{ij} : (i, j) \in E(G)\} \cup \{\neg P_{ij} : (i, j) \notin E(G)\}.$$

Now let

$$\mathcal{G} \coloneqq \{G : G \text{ is the transitive closure of a DAG}\},$$
$$\mathcal{C} \coloneqq \{S_G \cup \{\mathrm{POP}_n\} \cup \{C : C \text{ is a clause of } \mathrm{POP}_n\} : G \in \mathcal{G}\},$$

and let $\mathcal{C}^+$ be the closure of the set $\mathcal{C}$ under subsets. We claim that $\mathcal{C}^+$ is a $(n-1)$-consistency property. As in Proposition 3.5.1, since $\mathcal{C}^+$ is closed under subsets and $\{\mathrm{POP}_n\} \in \mathcal{C}^+$, $\mathrm{TWidth}(\mathrm{POP}_n) > n - 1$ follows. By definition, $\mathcal{C}^+$ cannot contain a formula and its negation. Now take an $S \in \mathcal{C}^+$ such that $|S| < n - 1$ and suppose that the Prover chooses a clause C of POP_n. Suppose (the other cases are easy to check) that C is of the form $[P_{1i}, \ldots, P_{ni}]$. Let G be the DAG on the vertices $V(G) = \{1, \ldots, n\}$ which contains an edge (i, j) if and only if $P_{ij} \in S$. We show that there exists a DAG G' containing exactly

the edges of G plus the edge (k, i) for some k, where $k \neq i$. Then $S_{G''} \in \mathcal{C}^+$, for the transitive closure of G', and adding to $S_{G''}$ all the formulas in

$$\{\mathrm{POP}_n\} \cup \{C : C \text{ is a clause of } \mathrm{POP}_n\}$$

which appear in S, and deleting all the literals except P_{ki} which do not appear in S, we get that $S \cup \{P_{ki}\} \in \mathcal{C}^+$. Because $|S| < n - 1$, G has at most $n - 2$ edges. This means that since

$$|E(G)| = \sum_{j \in V(G)} d_{\mathrm{in}}(j),$$

where $d_{\mathrm{in}}(j)$ is the number of edges entering j, G must have a vertex different than i with no incoming edges. Let k be such a vertex and let G' be the graph which results from G by adding the edge (k, i). Since G is acyclic, G' is also acyclic, because a cycle cannot contain the vertex k and we are done. □

From Propositions 3.5.1 and 3.5.2 and Theorem 3.1.1 we get that

$$\mathrm{CSpace}(\mathrm{PHP}_n^{n+1}) > n - 2$$

and

$$\mathrm{CSpace}(\mathrm{POP}_n) > n - 3.$$

We claim, along the lines of [3], that almost all known clause space lower bounds can be derived in this way. For example, the main theorem of [4] can be seen as providing the Adversary with a winning strategy in the $\Theta(n)$-width game played on a random r-CNF with n variables and a constant number times n clauses.

Furthermore, as we will see in Chapter 4, the lower bounds 3.5.1 and 3.5.2, tell us that any closed tableau with atomic cuts for the formulas PHP_n^{n+1} and POP_n, must have size at least $c \cdot 2^n$ for some constant c.

As we add cuts to the tableaux, the task of proving non-trivial lower bounds on the depth/width of tableau proofs becomes increasingly more difficult. If we confine ourselves to atomic cuts, we have resolution depth/width, and the problem remains doable (see e.g. [8]). The situation becomes chaotic when the cuts are unrestricted. The limit of our knowledge reaches the case where the cuts are restricted to formulas of *bounded depth*. The depth of a formula X is the depth of the tree representing X. For a non-negative integer k, what is the minimum number of rounds in which the Prover can refute a given formula, using cuts of depth at most k? Theorem 3.4.1 tells us that if this number is, say d, then the size of all Frege proofs consisting of formulas of depth at most $k - 1$ is $\Omega(2^d)$. One of the most advanced results in the field of proof complexity is that for any k this minimum number for PHP_n^{n+1} (when n is large enough) is at least n^μ, where μ is a ratio depending on k (see [1, 28, 26, 5]). On the contrary, in the case of unrestricted cuts, we know that there are polynomial size Frege proofs of PHP_n^{n+1} [11], so the Prover can refute PHP_n^{n+1} using unrestricted cuts within $O(\log n)$ rounds.

4

Conclusions

We believe that tableaux provide a nice, uniform template for arguing about the complexity of proofs. Tableaux with atomic cuts is a system p-equivalent to tree-like resolution. Tableaux with bounded depth cuts form a strictly stronger system (see e.g. [24]), p-equivalent to what is known as bounded depth Frege systems. Tableaux with arbitrary cuts form an even stronger system, p-equivalent to Frege systems. A general question is what is the exact effect of adding cuts on the width, depth or size of tableaux proofs.

Concerning the relationship of cut-free tableaux and resolution, the situation is summarized in Figure 4.1.

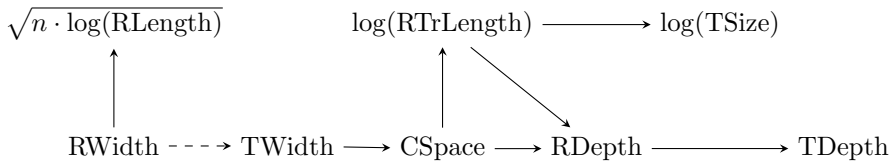

Figure 4.1: Cut-free tableaux and resolution.

$A \dashrightarrow B$ means that for any r-CNF X, where r is a constant independent of the number of variables of X, $A(X) = O(B(X))$. $A \to B$ means that $A \dashrightarrow B$ and there is a CNF X such that $A(X) = o(B(X))$. RLength(X) denotes the minimum number of clauses in a resolution refutation of X; RTreeLength(X) denotes the minimum number of clauses in a tree-like resolution refutation of X. Finally, TSize(X) is the minimum number of nodes in a closed tableau for X.

Let us elaborate on Figure 3.1. Let X be an r-CNF, where r is a constant.

$$\mathrm{RWidth}(X) = O\left(\sqrt{n \cdot \log(\mathrm{RLength}(X))}\right)$$

is the "short proofs are narrow" relation of [8]. This relation is almost tight [10] and for a formula separating RWidth from $\sqrt{\log(\mathrm{RLength})}$ take any large, minimally unsatisfiable 2-CNF. A family of formulas separating TWidth from CSpace are the XOR-pebbling contradictions. These formulas result from the formulas of Definition 2.8.2 after applying the transformation of Definition 3.3.2. We can see, as in Example 2.8.4, that any such formula has $O(1)$ tableau width, but there is a family of graphs (of constant in-degree) for which the corresponding formulas require clause space $\Omega(n/\log n)$, where n is proportional to the formula size [7]. The relation

$$\mathrm{CSpace}(X) = O\left(\log(\mathrm{TreeRLegth}(X))\right)$$

(actually, $\mathrm{CSpace}(X) \leq \log(\mathrm{TreeRLegth}(X)) + 2$) is from [20]; for a formula separating the two sides of the relation, again take any large, minimally unsatisfiable 2-CNF. A family of formulas separating log(RTreeLength) from RDepth are the pebbling contradictions (i.e., the formulas of Definition 2.8.2) [38]. Finally, a formula separating tableau size from resolution size are the formulas $\Sigma(T_n)$ of Figure 3.3 (see Theorem 5.1 of [37]). Could Figure 4.1 be made better?

Notice that we can deduce from Figure 3.1 the relation

$$\mathrm{TWidth}(X) = O\left(\log(\mathrm{TSize}(X))\right),$$

for any unsatisfiable r-CNF X, and in fact, being a little more careful,

$$\mathrm{TWidth}(X) \leq \log\left(\mathrm{TSize}(X)\right) + O(1),$$

for any unsatisfiable CNF X. That is, *"short proofs are narrow" in cut-free tableaux too*, for CNF-formulas. One could merge the different parts of this proof into a single proof, something we do right now.

Lemma 4.0.1. *Let X be a CNF-formula and L a literal. We write $X[L]$ for the formula resulting from X by removing all the clauses which contain L and removing $\overline{L}$ from all the clauses containing it. If $TWidth(X[L]) \leq w$, then the Prover has a width-w strategy on X, by which either wins, or forces the Adversary to select $\overline{L}$.*

Proof. The fact that $\mathrm{TWidth}(X[L]) \leq w$ means that the Prover has a width-w winning strategy on $X[L]$. Now if the same strategy is applied on X and the Adversary never selects $\overline{L}$, the Prover will eventually win. □

Theorem 4.0.1. *For any CNF-formula X, if there is a size t closed tableau with atomic cuts for X, then*

$$TWidth(X) \leq 3 + \log t.$$

Proof. A tableau for a CNF-formula is called *regular* if it uses only atomic cuts except at the bottom levels, where a clause is expanded. We show, by induction on t, that for any CNF X, and every regular closed tableau $\mathbf{T}$ for X, if $\mathbf{T}$ has t nodes, then

$$\mathrm{TWidth}(X) \leq \log t.$$

Theorem 4.0.1 follows, because a closed tableau with atomic cuts for a CNF-formula can be transformed, using Lemma 3.2.1, into a closed regular tableau, of size at most three times the size of the initial tableau. We assume during the proof, for convenience, that the Provers always keep the formula the game is played on in their memory, but we do not charge extra space for it.

The base case is easy to check. For the inductive step, let X be a CNF and let $\mathbf{T}$ be a regular closed tableau for X of size t. $\mathbf{T}$ must have two subtrees, say $\mathbf{T}_0$ and $\mathbf{T}_1$, the roots of which are labelled by P and $\neg P$ respectively, for some propositional variable P. Moreover, one of $\mathbf{T}_0$ and $\mathbf{T}_1$, suppose $\mathbf{T}_0$, must have size less than $t/2$. We will construct a $(\log t)$-width winning strategy for the Prover on X. The Prover begins by trying to select the formula P. We claim that he can do this using at most $\log t$ space. This is so, because by deleting all the leaves labelled by P from $\mathbf{T}_1$, and deleting all the leaves which are labelled with $\neg P$ along with their siblings, we can get a closed regular tableau for $X[\neg P]$ of size less than t. By the induction hypothesis,

$$\mathrm{TWidth}(X[\neg P]) < \log t,$$

and by Lemma 4.0.1, the Prover can force the Adversary to select P using only space $\lfloor \log t \rfloor$. Then the Prover forgets everything except P and while having P in his memory, tries to select $\neg P$ using at most $\lfloor \log t \rfloor - 1$ space. The trick is the same as before. From $\mathbf{T}_0$ we can get a closed regular tableau for $X[P]$ of size less than $|\mathbf{T}_0|$. Now since $|\mathbf{T}_0| < t/2$, this new tableau will have size less than $t/2$, and by the induction hypothesis

$$\mathrm{TWidth}(X[P]) < \log t - 1.$$

So again, by Lemma 4.0.1 the Prover can select $\neg P$ using space at most $\lfloor \log t \rfloor - 1$ and the proof is complete. □

Open Problem. Does Theorem 4.0.1 hold for non CNF-formulas? Does it hold for tableaux using only analytic cuts (i.e. tableaux where the cut formulas are subformulas, or negations of subformulas of the input formula)? Does it hold for cut-free, DAG-like sequent calculus proofs?

Bibliography

[1] Miklós Ajtai. The complexity of the pigeonhole principle. *Combinatorica*, 14:417–433, 1994.

[2] Michael Alekhnovich, Eli Ben-Sasson, Alexander Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM Journal of Computing*, 31:1184–1211, 2002.

[3] Albert Atserias and Victor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74:323 –334, 2008.

[4] Eli Ben-Sasson and Nicola Galesi. Space complexity of random formulae in resolution. *Random Structures & Algorithms*, 23:92–109, 2003.

[5] Eli Ben-Sasson and Prahladh Harsha. Lower bounds for bounded depth frege proofs via pudlák-buss games. *ACM Transactions on Computational Logic*, 11:19:1–19:17, 2010.

[6] Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of tree-like and general resolution. *Combinatorica*, 24:585–603, 2004.

[7] Eli Ben-Sasson and Jakob Nordström. Short proofs may be spacious: An optimal separation of space and length in resolution. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS '08*, pages 709–718, 2008.

[8] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow — resolution made simple. *Journal of the ACM*, 48:149–169, 2001.

[9] Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. A characterization of tree-like resolution size. *Information Processing Letters*, 113:666–671, 2013.

[10] Maria Luisa Bonet and Nicola Galesi. Optimality of size-width tradeoffs for resolution. *Computational Complexity*, 10:261–276, 2002.

[11] Samuel Buss. Polynomial size proofs of the propositional pigeonhole principle. *Journal of Symbolic Logic*, 52:916927, 1987.

[12] Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35:759–768, 1988.

[13] Stephen Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, 1971.

[14] Stephen Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, 2010.

[15] Stephen Cook and Robert Reckhow. On the lengths of proofs in the propositional calculus (preliminary version). In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 135–148, 1974.

[16] Stephen Cook and Robert Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979.

[17] Marcello D'Agostino. Are tableaux an improvement on truth-tables? *Journal of Logic, Language and Information*, 1:235–252, 1992.

[18] Marcello D'Agostino and Marco Mondadori. The taming of the cut. classical refutations with analytic cut. *Journal of Logic and Computation*, 4:285–319, 1994.

[19] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

[20] Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Information and Computation*, 171:84–97, 2001.

[21] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Graduate Texts in Computer Science. Springer, 1996.

[22] Gerhard Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, 1969.

[23] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297 – 308, 1985.

[24] Jan Krajíček. Lower bounds to the size of constant-depth propositional proofs. *Journal of Symbolic Logic*, 59:7386, 1994.

[25] Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*. Cambridge University Press, 1995.

[26] Jan Krajícek, Pavel Pudlák, and Alan Woods. An exponenetioal lower bound to the size of bounded depth frege proofs of the pigeonhole principle. *Random Structures and Algorithms*, 7:15–40, 1995.

[27] Christos Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[28] Toniann Pitassi, Paul Beame, and Russell Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational Complexity*, 3:97–140, 1993.

[29] Plato. *Meno*. Polis, 2008. in Greek.

[30] Pavel Pudlák and Samuel Buss. How to lie without being (easily) convicted and the length of proofs in propositional calculus. In *Selected Papers from the 8th International Workshop on Computer Science Logic*, CSL '94, pages 151–162, 1995.

[31] Pavel Pudlák and Russell Impagliazzo. A lower bound for DLL algorithms for k-SAT. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 128–136, 2000.

[32] Robert Reckhow. *On the lengths of proofs in the propositional calculus*. PhD thesis, Department of Computer Science, University of Toronto, 1975.

[33] Raymond Smullyan. *First-order Logic*. Dover, 1995.

[34] Richard Statman. Bounds for proof-search and speed-up in the predicate calculus. *Annals of Mathematical Logic*, 15:225–287, 1978.

[35] Grigori Tseitin. On the complexity of derivation in propositional calculus. *Studies in constructive mathematics and mathematical logic, part 2*, pages 115–125, 1968.

[36] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34:209–219, 1987.

[37] Alasdair Urquhart. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 1:425–467, 1995.

[38] Alasdair Urquhart. The depth of resolution proofs. *Studia Logica*, 99:349–364, 2011.