5 The Deductive Reasoning Task

In this chapter, we briefly put aside the PAC-semantics and focus on the classical semantics of knowledge; we will return to unifying the two in Chapter 6. This chapter discusses how reasoning can be mechanized through proof procedures that are sound and complete under the classical interpretation of truth for deduction. We introduce the classical entailment problem in Section 5.1 and how proof systems can help us capture variants of the problem of more tractable scope in Section 5.2; indeed, we will see in Section 5.3 how the resolution proof system, and in particular the size and structure of the proofs themselves, captures the behavior of the standard algorithms used for automated reasoning in practice. We include an example application of such algorithms to planning problems in Section 5.6. We will also discuss the complexity of sound and complete reasoning in several respects. First, we will see in Section 5.4 that in many cases, these standard algorithms must be slow because small resolution proofs do not exist. Second, we will see that the mere existence of a small resolution proof is not a sufficient condition for such algorithms, and indeed it is intractable to find small resolution proofs in Section 5.5. Thus, analogous to the learning task, we will see that there are significant limits to how sophisticated the logics may be if they are captured by computationally tractable reasoning algorithms. Following the analogy further, we will see in Section 5.8 that for the slightly more human-friendly logics used in textbook introductions, no efficient algorithm can decide whether a given query formula has a small proof in the system, even if it does not produce such a proof as output. Motivated by these limits, in Section 5.7 we introduce a family of resolution proofs of limited complexity that we can guarantee will be found by efficient algorithms. This formulation of a tractable reasoning task will serve as our basis going forward.

5.1 Knowledge bases and inference.

One of the most basic capabilities we aim to provide is to answer *queries* about the state of the world, or the state of a hypothetical world. As we will see, these queries can help guide intelligent behavior. For example, a robot may be tasked with placing a box in some room. It would be helpful for the robot to be able to determine when the box is in the room, and if it is not currently in the room, what hypothetical course of action might lead it to arrive there. If we have access to the complete description of possible

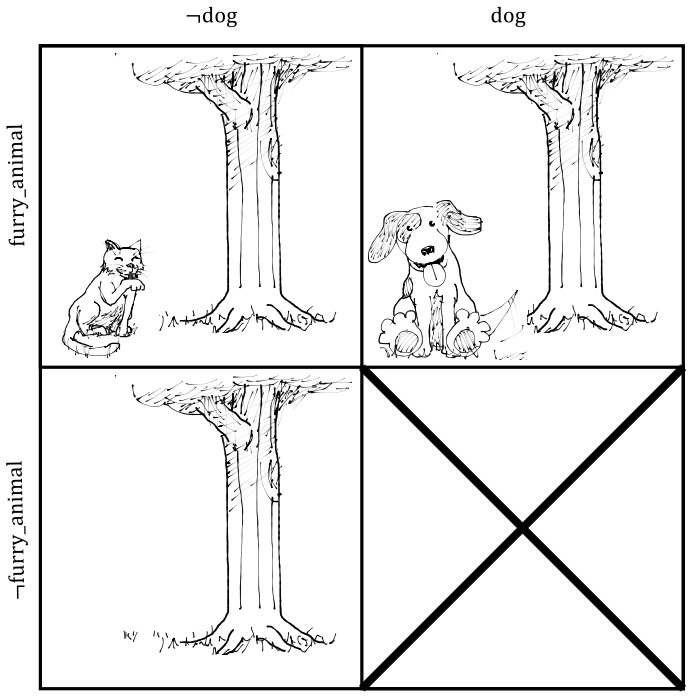


Figure 5.1 The space of possible worlds on the attributes "there is a dog" and "there is a furry animal." Only the world in which both are true remains possible given the knowledge base.

worlds corresponding to the world's current state, the state of the world after each of these hypothetical actions, and so on, then this task would correspond to evaluating a representation of the query on the descriptions of those respective possible worlds. The central problem that we face in this chapter arises when we do not have such a complete, explicit description of the world.

Instead of this full description of the world, we suppose that we possess a *knowledge base*: a set of representations in some fixed knowledge representation language that we assume to hold for a world of interest. Our task, now, is that when we are given a further query representation q, we wish to know if q holds in the world described by our knowledge base. This is, more precisely, an *entailment query*:

Definition 5.1 The *entailment query* task for knowledge representation families $\mathcal{H}$ and $\mathcal{Q}$ is, given a *knowledge base* KB consisting of $h_1, \ldots, h_k \in \mathcal{H}$ for some k, and a *query* representation $q \in \mathcal{Q}$, does $q(\vec{x})$ hold for all possible worlds x for which $h_1(\vec{x}), h_2(\vec{x}), \ldots, h_k(\vec{x})$ simultaneously hold, i.e., for the corresponding sets of possible worlds, is q is *valid* for $\{h_1, \ldots, h_k\}$? If so, we denote this by $\{h_1, \ldots, h_k\} \models q$. When the knowledge base is empty, this corresponds to simply asking of q is valid for all possible worlds, i.e., if it is a *tautology*.

Entailment queries capture reasoning as it is usually understood.

Example 5.1: As a concrete illustration, let's consider a domain in which there are two propositional attributes, "there is a dog" and "there is a furry animal." Our knowledge base might contain the representation, h_1 = ["there is a dog" $\supset$ "there is a furry animal"], expressing that we know dogs are furry animals. Now, if we consider a hypothetical world in which the knowledge base contains h_1 and, additionally, h_2 = "there is a dog". If we are now given the query q = "there is a furry animal" for the knowledge base $\{h_1, h_2\}$, we should be able to determine that q is indeed valid. Formally, there are four possible worlds, given by the two truth values that these two propositional attributes may take. But h_1 rules out the world in which "there is a dog" is true and "there is a furry animal" is false. h_2 then rules out the two possible worlds in which "there is a dog" is false, corresponding to the two truth values "there is a furry animal" may take. (See Figure 5.1.) We are thus left with only the world in which "there is a dog" is true *and* "there is a furry animal" is true, so the query q must be true.

Entailment queries describe idealized reasoning. Unfortunately, it turns out that such idealized reasoning is generally unattainable. To see this clearly, we first observe that the entailment query task can be reformulated in a few ways. First:

Proposition 5.2 *q is valid for $\{h_1, \ldots, h_k\}$ if and only if $\models [h_1 \wedge \cdots \wedge h_k] \supset q$.*

Proof Indeed, if q is valid for $\{h_1, \ldots, h_k\}$, then for all possible worlds either some h_i is false or q is true, so $\models [h_1 \wedge \cdots \wedge h_k] \supset q$. Conversely, if $\models [h_1 \wedge \cdots \wedge h_k] \supset q$, it must be that for all of the possible worlds $\vec{x}$ where $h_1(\vec{x}), \ldots, h_k(\vec{x})$ are true, $q(\vec{x})$ must also be true. □

It therefore suffices to be able to decide which representations are tautologies; equivalently, whether the representations corresponding to their negations are satisfiable. Formally,

Proposition 5.3 *q is valid for $\{h_1, \ldots, h_k\}$ if and only if $h_1 \wedge \cdots \wedge h_k \wedge \neg q$ is not satisfiable.*

Proof Follows immediately by De Morgan's law on Proposition 5.2. □

Recall that 3SAT is the problem of deciding whether an arbitrary 3-CNF formula – an AND of ORs of at most three literals – is NP-complete. Proposition 5.3 shows that this NP-complete problem is essentially equivalent to evaluating entailment queries for some very simple families of representations. The corresponding NP search problem is to find a counterexample to $[h_1 \wedge \cdots \wedge h_k] \supset q$, which is a satisfying assignment to $h_1 \wedge \cdots \wedge h_k \wedge \neg q$. Thus:

Theorem 5.4 *If $\mathcal{H}$ contains all clauses of at most three literals and $\mathcal{Q}$ contains a representation of the empty property, then entailment queries for knowledge bases from $\mathcal{H}$ and queries from $\mathcal{Q}$ is an NP-hard decision problem.*

We thus cannot hope to find efficient algorithms for answering entailment queries against even modestly expressive knowledge base representations. Once again, the classical view of knowledge is too demanding to be actionable by algorithms.

5.2 Proof systems: chaining and resolution.

While the formulation of entailment queries is elegant, it does not present a means to distinguish between queries that are hard to answer and queries that are easy to answer. Theorem 5.4 tells us that some queries must be hard to answer; intuitively, we should not be surprised if our algorithms cannot quickly determine the validity of deep mathematical theorems for us. But, if the query representation is among the representations in the knowledge base, this should be an easy query; our algorithms should be able to determine that the query is entailed. In general, we should hope for algorithms that are guaranteed to be able to answer queries efficiently when they are sufficiently "easy" in some sense, and to perhaps only answer them slowly if at all when the queries are not so easy. We will be able to capture the "easiness" of queries in terms of *proofs* in a *proof system*.

Definition 5.5 A *(Frege) proof system* is given by a class of knowledge representations $\mathcal{F}$ and a relation $R \subseteq \mathcal{F} \times \mathcal{F}^*$, where $\mathcal{F}^*$ denotes all finite length sequences (of length zero or more) of representations from $\mathcal{F}$ describing the *rules of inference*. A *proof* of a representation $q \in \mathcal{F}$ from *hypotheses* $h_1, \ldots, h_k \in \mathcal{F}$ is given by a sequence $\Pi = \pi_1, \pi_2, \ldots, \pi_\ell$ such that $\pi_\ell = q$ and for each π_i, either $\pi_i = h_j$ for some j or for some tuple $\pi'_1, \ldots, \pi'_r \in \{\pi_1, \ldots, \pi_{i-1}\}$, $(\pi_i, \pi'_1, \ldots, \pi'_r) \in R$. If $r = 0$, i.e., $(\pi_i) \in R$, we say π_i is an *axiom* of the proof system, and otherwise we say that π_i is a *consequence* of a rule of inference with *justification* $\pi'_1, \ldots, \pi'_r$.

If $\{\pi'_1, \ldots, \pi'_r\} \models \pi$ for all tuples $(\pi, \pi'_1, \ldots, \pi'_r) \in R$, we say that the proof system is *sound*. If for all q and $h_1, \ldots, h_k$ such that $\{h_1, \ldots, h_k\} \models q$ there is a proof of q from $h_1, \ldots, h_k$ in the proof system, then we say that the proof system is *complete*. If there is a polynomial-time algorithm deciding membership in the inference rule relation R, we say that the proof system is *efficiently verifiable*.

We note that sound proof systems can only have tautologies as axioms, and indeed a proof of q from $h_1, \ldots, h_k$ in a sound proof system ensures $\{h_1, \ldots, h_k\} \models q$. Because our motivations here are primarily computational – we want to identify reasoning problems that can be solved reasonably well by algorithms – we will be focused on proof systems for which the rules of inference are particularly simple. The first, simplest example is the (forward) chaining system:

Definition 5.6 In the *(forward) chaining* proof system, the representations are either literals (sometimes called *facts*) or *rules* of the form $[\ell_1 \wedge \cdots \wedge \ell_k] \supset \ell^*$ for some number k, where $\ell_1, \ldots, \ell_k$ and ℓ^* are literals. We call $\ell_1 \wedge \cdots \wedge \ell_k$ the *body* of the rule and ℓ^* the *head*. The rules of inference are of the form $(\ell^*, [\ell_1 \wedge \cdots \wedge \ell_k] \supset \ell^*, \ell_1, \ldots, \ell_k)$: that is, we may infer ℓ^* given a rule $[\ell_1 \wedge \cdots \wedge \ell_k] \supset \ell^*$ and the literals $\ell_1, \ldots, \ell_k$.

Along the lines of Example 5.1, it is clear that chaining is a sound proof system and that it is efficiently verifiable.

Example 5.2: Suppose we have a knowledge base containing the literals "dog" and "young", and the rules ["dog" ∧ "young"] ⊃ "puppy" and "puppy" ⊃ "mischief". We can prove this knowledge base entails "mischief" with the following chaining proof:

Inference	Justification
"dog"	knowledge base
"young"	knowledge base
["dog" ∧ "young"] ⊃ "puppy"	knowledge base
"puppy"	["dog" ∧ "young"] ⊃ "puppy", "dog", "young"
"puppy" ⊃ "mischief"	knowledge base
"mischief"	"puppy" ⊃ "mischief", "puppy"

As presented here, chaining is not complete.

Example 5.3: In chaining proofs, inference may only proceed from the body to the head of a rule. Suppose our knowledge base consists only of "dog" ⊃ "furry animal" and ¬"furry animal". Even though ¬"dog" is entailed by this knowledge base, there is no chaining proof of this fact. Indeed, the only literal provable by chaining here is ¬"furry animal".

If we consider a variant of chaining in which the literals may not use negation – i.e., instead of literals we simply have attributes – then chaining is complete for facts, see Exercise 5.1.

We next introduce a more powerful proof system that turns out to play a central role in the study of automated reasoning, *resolution*.

Definition 5.7 In the *resolution* proof system, the representations consist of *clauses* which, recall, are ORs of any number of literals. The rules of inference are of the form $(C \vee D, C', D')$ where C and D are ORs of any number of literals, C' is a clause containing an attribute X and the literals of C, and D' is a clause containing the literal $\neg X$ and the literals of D; that is, we may infer a clause $C \vee D$ from clauses $X \vee C$ and $\neg X \vee D$, i.e., containing an attribute and its negation respectively. This is called the *resolution* or *cut* rule.

Let $\perp$ denote the *empty clause* which we take to be the all-false property. A *resolution refutation* of a CNF formula $C_1 \wedge \cdots \wedge C_k$ is a resolution proof of $\perp$ from $\{C_1, \ldots, C_k\}$.

It is not hard to see that resolution is sound

Proposition 5.8 *The resolution rule is sound.*

Proof We show that for any inference $(C \vee D, C', D')$ allowed by resolution, for any possible world $\vec{x}$ that satisfies both C' and D', $\vec{x}$ satisfies $C \vee D$. Indeed, C contains all of the literals of C' except some attribute X and D contains all of the literals of D' except $\neg X$. We are supposing that both C' and D' have a literal that is true in $\vec{x}$, but note that one of X and $\neg X$ must be false, so these cannot be the respective literals in C' and D'. Thus at least one literal in $C \vee D$ is satisfied in $\vec{x}$, i.e., $\{C', D'\} \models C \vee D$. □

Although we presented resolution as a proof system for proving clauses, it is used primarily to prove DNF queries from CNF knowledge bases by refutation of the negation of the query, along the lines of Proposition 5.3.

Example 5.4: We revisit Example 5.2 to show how resolution can be used to obtain a proof of "mischief" from an equivalent knowledge base with a proof by refutation. We first note that the chaining rules can be represented as clauses, ¬"dog" ∨ ¬"young" ∨ "puppy" and ¬"puppy" ∨ "mischief". Individual literals are likewise clauses. Now, to prove a query "mischief" by refutation, we refute ¬"mischief" together with the knowledge base containing the clauses "dog", "young", ¬"dog" ∨ ¬"young" ∨ "puppy", and ¬"puppy" ∨ "mischief":

Inference	Justification
"dog"	knowledge base
"young"	knowledge base
¬"dog" ∨ ¬"young" ∨ "puppy"	knowledge base
¬"young" ∨ "puppy"	"dog", ¬"dog" ∨ ¬"young" ∨ "puppy"
"puppy"	"young", ¬"young" ∨ "puppy"
¬"puppy" ∨ "mischief"	knowledge base
"mischief"	"puppy", ¬"puppy" ∨ "mischief"
¬"mischief"	knowledge base (query)
⊥	"mischief", ¬"mischief"

We may thus conclude the original query, "mischief", follows from the knowledge base.

Indeed, along the lines of Example 5.4, we observe that for any chaining proof, there is a resolution proof of the same conclusion in which each chaining step is obtained by a sequence of at most n resolution steps when there are n attributes; see Exercise 5.7. That is, resolution is at least as expressive as chaining. We can capture this relationship with the following definition:

Definition 5.9 We say that a proof system P *p-simulates* a proof system Q if there exists a polynomial $p(s, n)$ such that whenever there is a proof of some representation q from $h_1, \ldots, h_k$ in Q of size s in an n attribute domain, there is also a proof in P of a representation q' equivalent to q (in the sense of representing the same property)

from representations $h'_1, \ldots, h'_k$, equivalent to $h_1, \ldots, h_k$ respectively, of size at most $p(s, n)$.

Thus, just as with knowledge representations, we can compare the expressive power of proof systems. We again allow for a small polynomial overhead in the notion of simulation, since this is necessary to allow intuitively reasonable kinds of simulation as with chaining by resolution in the example above.

Resolution can furthermore draw inferences that chaining cannot:

Example 5.5: We return to Example 5.3: suppose our knowledge base contains "dog" $\supset$ "furry animal" encoded as the clause ¬"dog" $\vee$ "furry animal" and ¬"furry animal". Resolution can prove ¬"dog": supposing "dog" for contradiction,

Inference	Justification
"dog"	knowledge base (query)
¬"furry animal"	knowledge base
¬"dog" $\vee$ "furry animal"	knowledge base
"furry animal"	"dog", ¬"dog" $\vee$ "furry animal"
$\perp$	¬"furry animal", "furry animal"

So resolution can draw contrapositive inferences. Indeed, resolution does not use any notion of order among the literals appearing in clauses, treating the clauses merely as an unordered set of literals.

Thus, resolution is *strictly* more expressive than chaining. Indeed, we will see in the next section that resolution, when used for refutation, is a *complete* proof system.

5.3 SAT-solvers and resolution.

We know that the satisfiability problem for CNF formulas is NP-complete, and therefore we do not hope to give a polynomial-time algorithm for this problem in general. Nevertheless, if we accept that we will be unable to find solutions to some instances, it is possible to design algorithms that solve many instances of CNF satisfiability in reasonable time. We will present some such algorithms in this section. The question that arises is then, if the algorithms cannot solve all instances in reasonable time, *which* instances will be solved quickly? We'll see how resolution proofs can help shed light on this question.

The basic approach underlying most SAT solvers is a *backtracking search* for a satisfying assignment. We can understand these algorithms as iteratively adding to a *partial assignment* to the attributes, assigning a truth value to one variable at a time and seeing if any clauses become violated.

Definition 5.10 A *partial assignment* to a set of attributes $X_1, \ldots, X_n$ is an assignment of a Boolean value to a subset of those attributes; the rest of the attributes are

said to be *free*. A partial assignment ρ is said to be *consistent* with a possible world $\vec{x}$ if whenever ρ assigns the attribute X_i a Boolean value, that value is x_i (i.e., the same value X_i takes in $\vec{x}$).

For a partial assignment ρ, the *partial evaluation* or *restriction* of a De Morgan formula φ, denoted $\varphi|_\rho$, is a formula using only the free attributes of ρ, inductively defined as follows

- For a single attribute formula X_i, if ρ fixes X_i to 1, $X_i|_\rho$ is the all-true formula $\top$; if ρ fixes X_i to 0, $X_i|_\rho$ is the all-false formula $\bot$; otherwise, if X_i is free, $X_i|_\rho$ is the formula X_i.
- For a formula $\neg\varphi$, if $\varphi|_\rho$ is a constant $b \in \{\bot, \top\}$, $(\neg\varphi)|_\rho$ is the constant $\bar{b} \in \{\bot, \top\}$ where $\bar{b}$ is $\bot$ if b is $\top$, and $\bar{b}$ is $\top$ if b is $\bot$. Otherwise, $(\neg\varphi)|_\rho$ is the formula $\neg(\varphi|_\rho)$.
- For a formula $[\varphi \wedge \psi]$
  - If $\varphi|_\rho$ or $\psi|_\rho$ are $\bot$, then $[\varphi \wedge \psi]|_\rho$ is $\bot$.
  - If both $\varphi|_\rho$ and $\psi|_\rho$ are $\top$, then $[\varphi \wedge \psi]|_\rho$ is $\top$.
  - If $\varphi|_\rho$ is $\top$ and $\psi|_\rho$ is a nonconstant formula, $[\varphi \wedge \psi]|_\rho$ is the formula $\psi|_\rho$. (Similarly, if $\psi|_\rho$ is $\top$ but $\varphi|_\rho$ is nonconstant, $[\varphi \wedge \psi]|_\rho$ is $\varphi|_\rho$.)
- Similarly, for a formula $[\varphi \vee \psi]$
  - If $\varphi|_\rho$ or $\psi|_\rho$ is $\top$, then $[\varphi \vee \psi]|_\rho$ is $\top$.
  - If both $\varphi|_\rho$ and $\psi|_\rho$ are $\bot$, then $[\varphi \vee \psi]|_\rho$ is $\bot$.
  - If $\varphi|_\rho$ is $\bot$ and $\psi|_\rho$ is nonconstant, $[\varphi \wedge \psi]|_\rho$ is the formula $\psi|_\rho$; likewise, if $\psi|_\rho$ is $\top$ but $\varphi|_\rho$ is nonconstant, $[\varphi \wedge \psi]|_\rho$ is $\varphi|_\rho$.

It is essentially immediate that whenever a partial assignment ρ is consistent with a possible world $\vec{x}$, for the assignment $\vec{x}'$ to the attributes left free in ρ, $\varphi(\vec{x})$ is the same truth value as $\varphi|_\rho(\vec{x}')$.

We can now easily describe a basic backtracking search algorithm for CNF satisfiability, Naïve-SAT, given in Algorithm 5. Its correctness is immediate by induction on the number of attributes. A simple modification to this algorithm enables it to return a resolution refutation instead of simply "FAIL" when the formula is unsatisfiable. To state this extension, we first extend the partial evaluation operation from formulas to proofs as follows.

Definition 5.11 For a proof π, we let $\pi|_\rho$ denote the sequence obtained by applying partial evaluation by ρ to each step of π.

If we extend resolution to include a rule called *weakening* that permits adding literals to clauses, the partial evaluation by ρ of a resolution proof from a CNF φ is a resolution proof from $\varphi|_\rho$—see exercise 5.6. Weakening itself does not add any power to resolution, and indeed any resolution proof that uses weakening can be rewritten to omit it, see exercise 5.3. We can now present the extension of Naïve-SAT to Naïve-RES, see Algorithm 6.

Theorem 5.12 *Naïve-RES returns a resolution refutation of an unsatisfiable φ and returns a satisfying partial assignment ρ when φ is satisfiable.*

Input: Conjunctive Normal Form formula φ on n attributes $X_1, \ldots, X_n$.
Output: Partial assignment ρ such that for all possible worlds $\vec{x} \in \{0,1\}^n$ consistent with ρ, $\varphi(\vec{x})$ is true, or FAIL if none exist.

```
begin
    if φ = ⊤ then return the empty assignment
    else if φ contains the empty clause ⊥ then return FAIL
    else
        Let X_1 be the first attribute of φ.
        if Naïve-SAT(φ|_{X_1←1}) returns ρ then return ρ′ extending ρ by setting
          X_1 ← 1
        else if Naïve-SAT(φ|_{X_1←0}) returns ρ then return ρ′ extending ρ by
          setting X_1 ← 0
        else return FAIL
end
```

Algorithm 5: Naïve-SAT: a basic algorithm for CNF formula satisfiability

Proof By induction on the number of attributes in $\varphi|_\rho$. Observe that a formula with no attributes must either be $\top$ or contain $\bot$; in the former case, ρ is a satisfying assignment, and in the latter the derivation π that simply states the clause C of φ such that $C|_\rho$ is $\bot$ indeed gives a refutation of $\varphi|_\rho$.

Now, for the induction step, suppose first that $\varphi|_\rho$ is satisfiable. Consider a setting b of the first free attribute X_i in a satisfying extension of ρ. Note that when we consider extending ρ by $X_i \leftarrow b$, and call Naïve-RES on the partial assignment $\rho' = \rho, X_i \leftarrow b$, since it has one less free variable, by the induction hypothesis it must return a satisfying partial assignment ρ'' extending ρ', hence also extending ρ. Indeed, furthermore by the induction hypothesis, if Naïve-RES returns a partial assignment ρ'' for either of these calls, it must extend ρ with the corresponding setting of X_i and some further assignments, and satisfy φ. Thus in this case, the algorithm correctly returns some such partial assignment ρ''.

It remains to consider the case when $\varphi|_\rho$ is unsatisfiable. Since $\varphi|_\rho$ would be satisfiable if extending it with either setting of X_i yielded a satisfiable formula, it must be that extending ρ with either setting of X_i gives an unsatisfiable formula. Thus, for each call Naïve-RES on a partial assignment $\rho_b = \rho, X_i \leftarrow b$, since it has one less free variable, by the induction hypothesis it must return a resolution proof π_b such that $\pi_b|_{\rho_b}$ is a refutation of $\varphi|_{\rho_b}$. In particular, now $\pi_0|_\rho$ is either a proof ending in $\bot$ or X_i and $\pi|_1|_\rho$ is either a proof ending in $\bot$ or $\neg X_i$. If either of these end in $\bot$, the respective π_b is as needed, a proof such that $\pi_b|_\rho$ is a refutation of $\varphi|_\rho$. Otherwise, the respective final clauses C_0 and C_1, respectively contain X_i and $\neg X_i$, with all other literals set to false by ρ. Thus, the proof π that starts with π_1, continues with π_0, and concludes with a resolution step on X_i on C_0 and C_1 satisfies that $\pi|_\rho$ is a refutation of $\varphi|_\rho$. □

In particular, since whenever a CNF formula φ is unsatisfiable, there is a resolution refutation of φ for Naïve-RES to return, we have found that resolution is *complete*:

Input: Conjunctive Normal Form formula φ on n attributes $X_1, \ldots, X_n$, current partial assignment ρ.
Output: Partial assignment ρ' extending ρ such that for all possible worlds $\vec{x} \in \{0,1\}^n$ consistent with ρ', $(\varphi|_\rho)(\vec{x})$ is true, or a resolution proof π such that $\pi|_\rho$ is a refutation of $\varphi|_\rho$ if none exist.

begin
  if $\varphi|_\rho = \top$ **then return** ρ
  else if $\varphi|_\rho$ *contains an empty clause* $\bot$ **then return** a clause C of φ such that $C|_\rho$ is $\bot$
  else
    Let X_i be the first free attribute of $\varphi|_\rho$.
    if *Naïve-RES*$(\varphi, (\rho, X_i \leftarrow 1))$ *returns* ρ'' **then return** ρ' extending ρ'' by setting $X_i \leftarrow 1$
    else if *Naïve-RES*$(\varphi, (\rho, X_i \leftarrow 0))$ *returns* ρ'' **then return** ρ' extending ρ'' by setting $X_i \leftarrow 0$
    else
      Let π_1 be the resolution refutation returned by Naïve-RES$(\varphi, (\rho, X_i \leftarrow 1))$ and let π_0 be the refutation returned by Naïve-RES$(\varphi, (\rho, X_i \leftarrow 0))$.
      if π_1 *ends with* C *such that* $C|_\rho$ *is* $\bot$ **then return** π_1
      else if π_0 *ends with* C *such that* $C|_\rho$ *is* $\bot$ **then return** π_0
      else
        // π_1 and π_0 end with clauses $C_1 = \neg X_i \vee C_1'$ and $C_0 = X_i \vee C_0'$, respectively, such that $C_1|_\rho$ is $\neg X_i$ and $C_0|_\rho$ is X_i
        return the resolution refutation of $\varphi|_\rho$:

$$\begin{array}{ll} & \pi_1 \\ & \pi_0 \\ C_1' \vee C_0' & C_1, C_0 \end{array}$$

end

Algorithm 6: Naïve-RES: an algorithm for CNF formula satisfiability with resolution proofs of unsatisfiability

Corollary 5.13 *Resolution is complete for CNF refutation.*

Moreover, we observe that each step of the returned proof π corresponds to one recursive call made by Naïve-RES. It follows that the *size* of the resolution refutation is a lower bound on the *running time* of the algorithm.

Unfortunately, these simplistic algorithms are obviously slow: even the simple formula $(\neg X_1 \vee X_n) \wedge (\neg X_1 \vee \neg X_n) \wedge (\neg X_2 \vee X_n) \wedge (\neg X_2 \vee \neg X_n) \wedge \cdots \wedge (\neg X_{n-1} \vee X_n) \wedge (\neg X_{n-1} \vee \neg X_n)$ takes time $\Theta(2^n)$ for it to solve. What is surprising is that only

a small handful of modifications distinguish this algorithm from algorithms that are highly successful in practice.

5.3.1 Pure variable rule.

At the start of the algorithm, scan the formula for literals that do not appear. Suppose the literal $\neg X_i$ for some attribute X_i does not appear in the formula. Then continue on $\varphi|_{X_i \leftarrow 1}$ and add $X_i \leftarrow 1$ to ρ if a satisfying partial assignment ρ is found. Similarly, if X_i does not appear, continue on $\varphi|_{X_i \leftarrow 0}$ and add $X_i \leftarrow 0$ to ρ if a satisfying partial assignment ρ is found.

Observe that if $\varphi|_\rho$ was satisfiable, it remains so with the extended ρ. The effect of this rule is then to drop the clauses containing X_i from φ. Indeed, moreover, note that the clauses including X_i, if they exist, cannot be used in any resolution refutation since the complementary literal does not exist in the formula, and thus an empty clause cannot be obtained from a clause using a literal containing X_i. Thus we can still find a resolution refutation if the formula was unsatisfiable.

5.3.2 Unit propagation.

At the start of the algorithm, following the pure variable rule, run the *unit propagation* algorithm described in Algorithm 7.

Input: Nonconstant Conjunctive Normal Form formula φ.
Output: Partial assignment ρ such that if there exists a possible world $\vec{x} \in \{0,1\}^n$ such that $\varphi(\vec{x})$ is true, $\vec{x}$ is consistent with ρ, and such that $\varphi|_\rho$ does not contain any clauses consisting of a single literal, or if φ is unsatisfiable, may return FAIL.

```
begin
    Initialize ρ to the empty partial assignment.
    while φ contains some clause consisting of a single literal ℓ do
        if ℓ is X_i then add X_i ← 1 to ρ
        else add X_i ← 0 to ρ        (ℓ is ¬X_i)
        φ ← φ|_ρ
        if φ contains the empty clause ⊥ then return FAIL
    end
    return ρ
end
```

Algorithm 7: Unit Propagation

The correctness of unit propagation follows immediately from induction on the number of iterations of the loop: if a clause X_i (resp. $\neg X_i$) appears in a formula φ, any possible world satisfying φ must set X_i to 1 (resp. 0). The same holds for any such clauses appearing in $\varphi|_\rho$ since all satisfying possible worlds to φ must be consistent

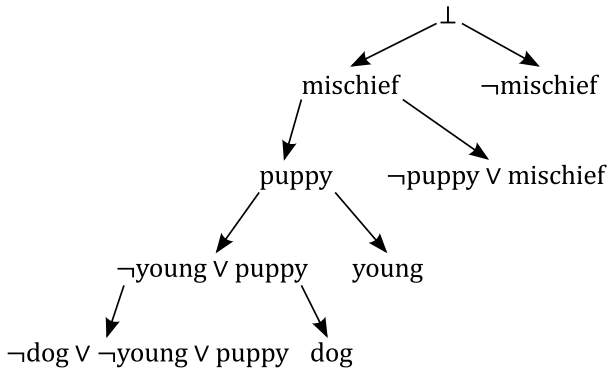


Figure 5.2 The graph of the proof from Example 5.4. Observe that it is a (rooted) tree.

with ρ. Thus, if $\varphi|_\rho$ contains an empty clause, it is indeed unsatisfiable. The loop indeed otherwise terminates only when $\varphi|_\rho$ no longer has any such clauses.

It turns out that unit propagation can be simulated by resolution, see Exercise 5.7, so we can still return a resolution refutation of size no greater than the running time of the algorithm. At the same time, unit propagation also subsumes chaining (Exercise 5.7 again) and in turn directly simulates the evaluation of circuits constructed given the encodings constructed by the reduction from circuit satisfiability to 3SAT (Theorem 3.29). The appeal of unit propagation is that it has a linear-time implementation (see Exercise 5.8) which is very fast in practice, and handles these kinds of structures created by common reductions.

When the pure variable rule and unit propagation are added to Naïve-SAT, the resulting algorithm is known as *DPLL* (for the authors Davis, Putnam, Logemann, and Loveland, who first studied this algorithm). A weakness of this algorithm is reflected in the fact that the resolution proofs that it finds do not reuse any intermediate derivations; such proofs are said to be *treelike*:

Definition 5.14 Given a proof π in some proof system, we can associate a graph G_π with the proof as follows: for each line π_i of the proof there is a corresponding vertex v_i, and for each of the vertices v_j corresponding to each of the previous lines π_j justifying a line π_i of the proof, there is a directed edge from the vertex v_j to the vertex v_i for that line π_i. (Since $j < i$, the graph is acyclic.) If this graph is a tree – i.e., if each line π_j is used as a justification for at most one other line π_i – we say that the proof π is *treelike*.

Example 5.6: Our example resolution proofs from the previous section, e.g., Example 5.4, are treelike proofs; see Figure 5.2 for the graph.

It is immediate from inspection that Naïve-RES returns treelike resolution refutations; it is less obvious, but follows from Exercise 5.7, that DPLL also returns treelike resolution refutations. Thus we find:

Proposition 5.15 *The size of the smallest treelike resolution refutation of an unsatisfiable formula gives a lower bound on the running time of the DPLL algorithm on that formula.*

Treelike proofs are known to be particularly restrictive, as we'll discuss further later.

5.3.3 Clause learning.

In *clause learning*, when the search backtracks (identifying an unsatisfiable formula), we add a clause C to φ such that C is falsified by the partial assignment chosen by the algorithm at that point – we call the literals satisfied by this partial assignment *decision* literals – and *not* including assignments made by unit propagation, which we refer to as *implied* literals, and $\varphi \wedge C$ has the same set of satisfying assignments as φ. The intuitive benefit is that unit propagation when C is included would prevent the algorithm from revisiting such branches. Moreover, the use of these "learned" clauses will turn out to free the algorithm from the limits of treelike proofs.

As a simple example, we could take the negation of all decision literals as such a clause—observe that this is equivalent to the negation of the conjunction corresponding to that partial assignment. Although such clauses meet our criteria, it is often possible to identify more useful clauses. It is generally considered desirable to identify a smaller conflict clause, for example, as this helps eliminate a larger set of canidate partial assignments. These clauses are identified by constructing *conflict graphs* capturing the inferences made during unit propagation:

Definition 5.16 A *implication graph* is a directed acyclic graph in which the vertices are labeled either with literals or $\perp$ and the edges are labeled with sets of clauses; at a given point in the backtracking search, it is constructed as follows (e.g., during unit propagation):

- The decision literals are source vertices.
- For each clause in the current φ, C_i consisting of the literals $\{\neg\ell_1, \neg\ell_2, \ldots, \neg\ell_k, \ell^*\}$, such that vertices labeled by $\ell_1, \ldots, \ell_k$ exist in the graph, add a vertex ℓ^* (if not already present), and for each ℓ_j, add an edge from ℓ_j to ℓ^* if not already present, and add the clause C_i to the set labeling its edge.
- Add a sink vertex $\perp$ with edges from any pair of literals X_i and $\neg X_i$ present. These literals are said to be *conflict literals*, and the graph is said to *contain a conflict* if any such literals exist.

Now, a *conflict graph* is a subgraph G of the implication graph with the following properties:

- G contains $\perp$ and exactly one pair of conflict literals.
- All nodes in G contain a path to $\perp$.
- All nodes ℓ in G are either decision literals or have edges from precisely the set of literals $\ell_1, \ldots, \ell_k$ such that a clause consisting of $\{\neg\ell_1, \neg\ell_2, \ldots, \neg\ell_k, \ell^*\}$ appears in the current φ.

For any subset S of the vertices of a conflict graph G that contains all of the decision literals, containing at most one of the conflict literals, and omitting $\perp$, the corresponding *conflict clause* consists of $\neg\ell$ for each literal $\ell \in S$ with an edge to a literal $\ell' \notin S$.

Intuitively, these conflict clauses summarize the contribution of S to the unit propagation derivation of a contradiction: if the clause is falsified, then following $G - S$ gives a unit propagation refutation from φ. These conflict clauses preserve the set of satisfying assignments because they may be derived from existing clauses by resolution proofs:

Theorem 5.17 *A conflict clause may be proved by resolution from the formula φ with a resolution proof that repeatedly introduces one new clause of φ and applies resolution to it and the previous clause of the proof, and only uses resolution at most once on each variable.*

Proof We first note that if the set S corresponding to the conflict clause contains all of the literals but one conflict literal ℓ^*, then by definition, there is a clause consisting of $\{\neg\ell_1, \neg\ell_2, \ldots, \neg\ell_k, \ell^*\}$ in φ, and this is precisely the conflict clause; it thus has a trivial resolution proof, simply consisting of that clause.

We now proceed by induction on the number of vertices *not* in S. Let the conflict clause consist of literals $\{\neg\ell_1, \neg\ell_2, \ldots, \neg\ell_k\}$. Because the graph is acyclic, there must exist some vertex ℓ^* of G that is not in S, but such that all edges to ℓ^* are from vertices ℓ_i of S (such that $\neg\ell_i$ is in the conflict clause). Observe therefore that by definition, for some subclause $\{\neg\ell'_1, \ldots, \neg\ell'_p\}$ of the conflict clause, there must exist a clause consisting of $\{\neg\ell'_1, \ldots, \neg\ell'_p, \ell^*\}$ in the current φ. Now, if we consider the strictly larger set $S \cup \{\ell^*\}$, we obtain a conflict clause containing $\neg\ell^*$ along with $\{\neg\ell_1, \neg\ell_2, \ldots, \neg\ell_k\} - \{\neg\ell'_1, \ldots, \neg\ell'_p\}$ (and possibly some of the literals from $\{\neg\ell'_1, \ldots, \neg\ell'_p\}$ if they appeared in other clauses). By induction, this conflict clause has a resolution proof from φ of the promised form. If we extend this proof by applying a cut on ℓ^* using the clause $\{\neg\ell'_1, \ldots, \neg\ell'_p, \ell^*\}$ in the current φ, we indeed obtain the conflict clause, consisting of the literals $\{\neg\ell_1, \neg\ell_2, \ldots, \neg\ell_k\}$; since vertices are only added to S and each literal appears only once in the conflict graph, ℓ^* was not used in an earlier step of the proof. We thus obtain a resolution proof of the conflict clause of the claimed form, completing the induction step. □

Note that since the conflict graph is constructed during unit propagation – each implied literal is created by a step of unit propagation – the size of the resolution proof of the conflict clause is also no greater than the number of steps of unit propagation. Thus, when the formula is unsatisfiable, we can still modify the algorithm to return a resolution proof (now, in general, no longer treelike), the size of which is at most the running time of the algorithm. Although many different strategies are used to select the conflict graph and subset for the conflict clause, our analysis here does not depend on which strategy is used, and there is currently no clear theoretical reason to prefer one to another.

Typically, the algorithm is further modified so that when a conflict clause is added, the algorithm *"backjumps"* to the level of the recursion where unit propagation applied to the conflict clause would have implied its final unset literal to be satisfied. Algorithms that extend DPLL with clause learning and backjumping are known as *Conflict-Directed Clause Learning (CDCL)* algorithms.

5.3.4 Random restarts.

A final modification that is important in practice is that CDCL algorithms may be modified to, with some small probability at each recursive call, instead return to the original call – clearing the partial assignment constructed so far – and restart the search while maintaining the set of additional clauses obtained from clause learning. Again, since these clauses had resolution proofs from the input formula, both this modification and backjumping still allow the algorithm to return a resolution proof with size no greater than the algorithm's running time. Thus, we can make the following observation:

Proposition 5.18 *The size of the smallest resolution refutation of an unsatisfiable formula gives a lower bound on the running time of a CDCL algorithm on that formula.*

In summary, we first note that the methods developed to solve satisfiability for CNF formulas can produce resolution proofs for many instances that arise in practice reasonably efficiently. Second, conversely, in order for the methods to run in a reasonable amount of time, a reasonably small resolution refutation must exist. But, as we will see, resolution is a relatively weak system, so the requirement that small proofs exist poses a significant constraint.

5.4 Resolution is weak: introducing proof complexity.

We saw that most algorithms used for reasoning, such as CDCL SAT solvers, have simple, low-overhead methods for producing resolution refutations for unsatisfiable inputs. The size of the resolution refutation therefore bounds the running time of the algorithm. We would therefore like to better understand which formulas have small resolution refutations, and which do not. Are these formulas merely pathological, with these SAT solvers capable of solving every formula we might encounter in practice? In this section we will see that actually, some very simple, natural formulas are hard for these basic SAT solvers.

In particular, we will show that formulas capturing the *"pigeonhole principle"* in combinatorics are hard for resolution: if we are trying to place *"pigeons"* in *"holes"* and the pigeons outnumber the holes, then the pigeons cannot all be placed in their own hole. Intuitively, the difficulty presented by the pigeonhole principle is that it requires a kind of quantitative reasoning that resolution cannot capture concisely. It is possible to augment these algorithms with some quantitative reasoning ability to solve such

instances, but this power generally comes at a cost in running time on other instances. As in learning, representational power comes at a price.

So, we are primarily interested in one measure of the difficulty of a reasoning problem: the length of the shortest resolution refutation. It turns out to be easier to study another quantity, the width of the largest clauses that need to be considered for such resolution refutations. We'll use the following notation.

Definition 5.19 The *length* of a resolution refutation π, denoted $L(\pi)$, is the number of formulas in the sequence. The *(resolution) length complexity* of a CNF formula φ, denoted $L(\varphi \vdash \bot)$, is the minimum $L(\pi)$ over all refutations π of φ.

The *width* of a resolution refutation π, denoted $w(\pi)$, is the largest width (number of literals) of a clause π_i in π. We similarly denote the width of a CNF formula φ by $w(\varphi)$. The *(resolution) width complexity* of a CNF formula φ, denoted $w(\varphi \vdash \bot)$, is the minimum $w(\pi)$ over all refutations π of φ.

The connection between length and width is that for formulas of low width, high width complexity implies high length complexity:

Theorem 5.20 *For any unsatisfiable CNF formula φ,*

$$L(\varphi \vdash \bot) \geq 2^{\Omega((w(\varphi \vdash \bot) - w(\varphi))^2/n)}.$$

Proof We'll actually prove that $w(\varphi \vdash \bot) \leq w(\varphi) + O(\sqrt{n \log L(\varphi \vdash \bot)})$; the theorem then follows by simple algebra. Let π be a refutation of φ of length $L = L(\varphi \vdash \bot)$. We will call clauses *heavy* if they have width at least $h = \lceil \sqrt{2n \log L} \rceil$. We will prove, by induction on the number of attributes n, that if the number of heavy clauses is less than $\left(\frac{1}{1-h/2n}\right)^t$ for an integer t, then

$$w(\varphi \vdash \bot) \leq w(\varphi) + h + t.$$

Note that since the number of heavy clauses in π is at most L, this yields the final bound $w(\varphi \vdash \bot) \leq w(\varphi) + O(\sqrt{n \log L(\varphi \vdash \bot)})$ by considering the minimum t^* satisfying $\left(\frac{1}{1-h/2n}\right)^{t^*} \geq L$, i.e., $t^* = \lceil \frac{\log L}{-\log(1-h/2n)} \rceil$. Since $\ln(1 - h/2n) \leq -h/2n$, $\frac{1}{-\ln(1-h/2n)} \leq 2n/h = O(\sqrt{\frac{n}{\log L}})$. Thus, $t^* = O(\sqrt{n \log L})$ (and similarly for h by definition).

Now, if $n = 0$, φ can only contain $\bot$, so $w(\varphi \vdash \bot) = 0$. Thus, suppose that the claim is proved for $n - 1$ attributes, and we wish to establish it for n. If $t = 0$, there cannot be any heavy clauses, and then $w(\varphi \vdash \bot) \leq h$ and we are done. Thus, consider $t \geq 1$. Since the heavy clauses collectively contain at least $h\left(\frac{1}{1-h/2n}\right)^t$ literals, at least one of the $2n$ literals must appear in at least $\frac{h}{2n}\left(\frac{1}{1-h/2n}\right)^t$ of the heavy clauses. Let ℓ^* be such a literal, and consider the partial assignment that sets $\ell^* \leftarrow 1$. Then in the partial evaluation of π under this assignment, $\pi|_{\ell^* \leftarrow 1}$, all of the clauses containing ℓ^* are replaced by $\top$, and we obtain a refutation of $\varphi|_{\ell^* \leftarrow 1}$ (with weakening) on $n - 1$ attributes (cf. exercise 5.6) and with at most $\left(\frac{1}{1-h/2n}\right)^t - \frac{h}{2n}\left(\frac{1}{1-h/2n}\right)^t = \left(\frac{1}{1-h/2n}\right)^{t-1}$ heavy clauses. Thus, by the induction hypothesis,

$$w(\varphi|_{\ell^* \leftarrow 1} \vdash \bot) \leq w(\varphi|_{\ell^* \leftarrow 1}) + h + t - 1 \leq w(\varphi) + h + t - 1.$$

Let π' be any resolution proof from φ such that $\pi'|_{\ell^* \leftarrow 1}$ is a refutation of $\varphi|_{\ell^* \leftarrow 1}$; observe that π' is either a refutation of φ or a proof of $\neg\ell^*$. Thus, in particular, for the refutation π^* such that $w(\pi^*|_{\ell^* \leftarrow 1}) \leq w(\varphi) + h + t - 1$ that is guaranteed to exist by the induction hypothesis, π^* is either a refutation of φ of width $w(\varphi) + h + t$, in which case we are done, or a proof of $\neg\ell^*$ of width $w(\varphi) + h + t$. In the latter case, we can use $\neg\ell^*$ to eliminate ℓ^* from the remaining clauses, to obtain $\varphi|_{\ell^* \leftarrow 0}$, which has $n - 1$ attributes.

Furthermore, observe that the partial evaluation of π in which we set $\ell^* \leftarrow 0$ still contains fewer than $\left(\frac{1}{1-h/2n}\right)^t$ heavy clauses. Thus, we can apply the induction hypothesis again and obtain that $\varphi|_{\ell^* \leftarrow 0}$ can then be refuted in width at most $w(\varphi)+h+t$. Taking such a refutation π^{**} of $\varphi|_{\ell^* \leftarrow 0}$ thus completes our refutation of φ while maintaining width at most $w(\varphi)+h+t$ as needed. This completes the induction step. □

Theorem 5.20 only provides nontrivial bounds for formulas of low width. One such formulation of the pigeonhole principle is as follows: we divide the n attributes into p blocks of size q (so $n = p \cdot q$). We interpret the q Boolean attributes $X_{i,1}, \ldots, X_{i,q}$ in each ith block as representing the number of a "hole" in binary, to which the corresponding ith pigeon is assigned. Thus, there are p pigeons and 2^q holes, and as long as $2^q < p$ – i.e., $q < \log_2 p$ – it is impossible for the pigeons to be assigned distinct holes. We represent this with clauses asserting that for each of the 2^q holes, each pair of distinct pigeons i_1 and i_2 are not simultaneously placed in that hole. If we use the notation $X_{i,j}^1$ for the literal $X_{i,j}$ and $X_{i,j}^0$ for $\neg X_{i,j}$, observe that for the hole with index $b_1, \ldots, b_q$, the conjunction $X_{i,1}^{b_1} \wedge \cdots \wedge X_{i,q}^{b_q}$ is true iff pigeon i is assigned that hole. Thus, by De Morgan's law, our formula is:

$$\mathrm{PHP}_q^p = \bigwedge_{\vec{b}, i_1 < i_2} X_{i_1,1}^{1-b_1} \vee \cdots \vee X_{i_1,q}^{1-b_q} \vee X_{i_2,1}^{1-b_1} \vee \cdots \vee X_{i_2,q}^{1-b_q}$$

which is a CNF formula of $2^q \binom{p}{2} < O(p^3)$ clauses of width $2q < 2 \log p$.

We show PHP_q^p requires width at least 2^q by considering the following game:

Definition 5.21 In the *formula game*, there are two players, the "Prover" and the "Liar". The Liar claims that the formula φ has a satisfying assignment, whereas the prover aims to show that the formula is unsatisfiable. The game is played in rounds. On each round, the Prover selects an attribute and challenges the Liar to reveal an assignment to that attribute. The Prover remembers the Liar's answers from previous rounds, but is permitted to "forget" these answers if it chooses. We assume the Prover's choice of attribute depends only on the round and the contents of its memory. If, at any point, the partial assignment ρ the Prover remembers falsifies a clause of φ, the Prover wins. If the Liar has a counter-strategy that never reaches such a partial assignment, the Liar wins.

The number of attributes the Prover needs to remember in a winning strategy captures the width complexity:

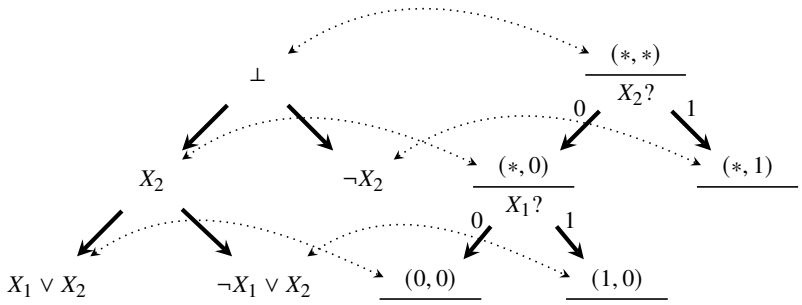


Figure 5.3 Example of the correspondence between the Prover's strategy and a resolution refutation for the clauses $X_1 \vee X_2$, $\neg X_1 \vee X_2$, and $\neg X_2$. The clause at each step in the refutation contains precisely the set of literals falsified by the contents of the prover's memory in the corresponding state of the strategy.

Theorem 5.22 *If there is a resolution refutation of φ of width w the Prover has a winning strategy in the formula game on φ while remembering at most $w+1$ attributes simultaneously. Conversely, if the Prover has a winning strategy while remembering at most w attributes simultaneously, there is a resolution refutation of width w.*

Proof Let π be a resolution refutation of φ of width w. The Prover can obtain a winning strategy from π remembering at most $w+1$ attributes by following the graph of π as follows: let the "current vertex" v initially correspond to the final step of π, containing $\bot$. We will maintain the invariant that the partial assignment ρ in the Prover's memory falsifies the clause in the current vertex; observe that for the initial empty assignment, $\bot$ is indeed falsified. At the vertex corresponding to any step π_i, the step was justified by the resolution rule on some attribute X on some prior steps j and k. The Prover queries the Liar about the value of X. Since we are guaranteed that ρ falsifies all of the literals of π_i, one of π_j and π_k contained X and the other contained $\neg X$, and the rest of the literals of π_j and π_k appear in π_i, no matter which answer the Liar gives, the resulting partial assignment ρ' extended with an assignment to X must falsify either π_j or π_k. The Prover then updates the current vertex to the falsified one, and forgets the values assigned to literals not contained in that clause. If the clause was one of the clauses of φ, the Liar loses, and since the proof has finite length, the game must eventually reach one of the clauses of φ, so this is indeed a winning strategy for the Prover. Moreover, since each clause in π has width at most w and the Prover only needs to remember the w literals plus the one used in the resolution step, this strategy only involves $w+1$ literals at a time, as claimed.

Conversely, suppose that the Prover has such a strategy. Consider the following graph associated with the strategy: for each step t of the game and state ρ of the Prover's memory, we have a vertex. Since we assume that the Prover wins while remembering at most w attributes, the game lasts for a finite number of rounds and we only need to consider partial assignments on up to w attributes. We will include an edge in the

graph from ρ in layer t to the vertices in layer $t+1$ corresponding to the two partial assignments that the Prover will use, depending on the Liar's answers to the next chosen attribute. Furthermore, if the Prover forgets the value of the queried attribute in one of the vertices, we omit the edge to the other vertex. We only consider the portion of the graph reachable from the initial, empty assignment in the first layer. The graph is finite and acyclic. We now obtain a resolution proof with weakening as follows: first, for each sink of the graph, if the corresponding partial assignment does not correspond to precisely the assignment falsifying a clause of φ, then we add a vertex with that assignment with a single edge. Now, consider an ordering of the vertices in which the sinks obtain the lowest numbers and the unique source has the highest number. If we write down the unique clause falsified by the partial assignment at each vertex, we claim this is a resolution refutation using weakening.

Indeed, we have ensured that the sinks are all clauses of φ, and inductively supposing that the first $i-1$ steps are a valid resolution proof, step i either has an edge to a single vertex corresponding to an earlier step, or two edges to two such vertices. In the former case, either it was one of our added vertices or a vertex that simply consists of a subset of the previous partial assignment (since the Prover did not record the Liar's answer), and thus may be obtained by weakening. Otherwise, the two vertices both recorded the Liar's two possible answers and thus correspond to clauses with a complementary pair of literals, and otherwise containing only a subset of the literals of the clause for step i. Thus, step i may be obtained by resolution on the queried attribute, possibly followed by weakening. We therefore see that inductively, we obtain a valid resolution proof with weakening, in which each step is a clause of width at most w. Since the empty partial assignment corresponds to the empty clause, it is indeed a refutation of φ. Finally, we can drop the weakening steps to obtain a refutation of width w without weakening—cf. exercise 5.3. □

Finally, we obtain the width lower bound by showing how the Liar can win the game for PHP_q^p if the Prover remembers fewer than 2^q attributes:

Theorem 5.23 *The resolution width complexity of PHP_q^p (for $q < \log_2 p$) is at least $2^q - 1$.*

Proof Fix a strategy for the Prover in which the Prover remembers fewer than 2^q attributes in any state. The Liar can now win using the following strategy. We say that the partial assignment *mentions* a pigeon i if it includes some attribute $X_{i,j}$ for some j. Inductively, the Liar will remember a set of distinct hole assignments for all of the pigeons mentioned in the Prover's current memory; when the Prover forgets all of the attributes for a given pigeon, the Liar will also forget the assignment chosen for that pigeon. This will allow the Liar to continue to maintain this assignment and answer the Prover's queries:

Initially, the Prover's memory is empty and so the empty set of assignments suffices. Whenever the Prover queries a new attribute, if it is for a pigeon that is mentioned in the Prover's current memory, the Liar answers with the corresponding bit of the assignment currently chosen for that pigeon. Otherwise, it is for a new pigeon; since the Prover's

state includes at most $2^q - 1$ attributes, the Liar currently has assigned at most $2^q - 1$ pigeons holes, so the Liar can choose an index distinct from the indices chosen for the currently mentioned pigeons for this new pigeon. Thus the invariant is maintained.

Now, since at any point the Prover's memory contains assignments consistent with the Liar's chosen set of distinct holes for the mentioned pigeons, none of the clauses of PHP_q^p are falsified. Thus, the Liar wins. Since by Theorem 5.22, a width complexity of $2^q - 2$ would give the Prover a winning strategy remembering fewer than 2^q attributes, the width complexity of PHP_q^p is at least $2^q - 1$. □

Theorem 5.20 now shows that PHP_q^p requires a long refutation:

Corollary 5.24 *For sufficiently large n and q satisfying $\log_2(n/q) - 1 \leq q < \log_2(n/q)$, $PHP_q^{n/q}$ requires refutations of length at least $2^{\Omega(n/\log^2 n)}$.*

Proof For such n and q, $\mathrm{PHP}_q^{n/q}$ has width $2q < 2\log_2(n/q)$. By Theorem 5.23 also requires refutations of width $2^q - 1 \geq \frac{n}{2q} - 1$. Noting that $\frac{(\frac{n}{2q} - 1 - 2q)^2}{n} \geq \Omega(n/\log^2 n)$, the claim follows from Theorem 5.20. □

And finally, by Proposition 5.18, CDCL algorithms therefore also run for at least $2^{\Omega(n/\log^2 n)}$ steps on these formulas, which have only n attributes and $O(n^3)$ clauses.

5.5 Resolution is hard.

In Section 5.3, we saw that the size of resolution proofs poses a barrier to standard algorithms for deductive reasoning based on CDCL SAT-solvers: the running time on an unsatisfiable formula is at least the size of the smallest resolution refutation (Proposition 5.18). We did not see a converse to this claim, that some CDCL SAT-solver has its running time bounded by, say, a polynomial in the size of the smallest resolution refutation. There are many possible clause learning methods and many possible heuristics one might use to select a variable ordering for the decisions; it is natural to ask if there could be a choice that yields an algorithm that efficiently determines that formulas are unsatisfiable when they have a small resolution refutation. Formally, we refer to such proof systems as *automatable*:

Definition 5.25 We say that a proof system is $t(n, S)$-*automatable* if there is an *automating* algorithm that, when given a formula φ and knowledge base KB containing n attributes, such that φ is provable from KB in the proof system by a proof of total size S, outputs a proof of φ in the system in time $t(n, S)$. If $t(n, S)$ is a polynomial in n and S, we will say that the proof system is *efficiently automatable*.

Unfortunately, as we will see, resolution is *not* efficiently automatable unless P = NP. Thus, in particular, no efficiently computable strategy for clause learning and variable ordering can efficiently discover resolution proofs.

Theorem 5.26 *If* P ≠ NP*, then resolution is not efficiently automatable.*

We will prove that resolution is not efficiently automatable by a reduction showing how an automating algorithm for resolution can be used to decide CNF satisfiability. The reduction will map an input CNF formula φ to a new CNF formula $f(\varphi)$ that, informally, asserts that φ can be refuted in length L. When φ has n attributes and m clauses, for $L \geq n^2 + mn$, we will be able to show that these formulas exhibit the following behavior:

- If φ is satisfiable: $f(\varphi)$ has a polynomial-size resolution refutation.
- If φ is unsatisfiable: $f(\varphi)$ cannot have resolution refutations of less than exponential length.

So, if resolution were efficiently automatable, we could decide CNF satisfiability by running the automating algorithm on $f(\varphi)$ for a polynomial number of steps and accepting if a refutation is returned, rejecting otherwise.

We first formally describe the formula $f(\varphi)$:

Definition 5.27 Let φ be a CNF formula on attributes $X_1, \ldots, X_n$ with m clauses. For an integer L, the formula $\mathrm{REF}_L(\varphi)$ is the following CNF. There are $(2n+2+\lceil\log_2 m\rceil+\lceil\log_2 n\rceil + 2\lceil\log_2 L\rceil) \cdot L$ attributes that encode the L steps of a refutation as follows. For each ith step, there are

- $2n$ attributes $Y_{i,j,b}$, where $Y_{i,j,0} = 1$ represents that the clause contains the literal $\neg X_j$ and $Y_{i,j,1} = 1$ represents that the clause π_i contains the literal X_j.
- Attributes $R_{i,F}$ and $R_{i,R}$ representing, respectively, whether the clause π_i is a weakening of a clause of φ or obtained by resolution on some attribute.
- Attributes $H_{i,1}, \ldots, H_{i,\lceil\log_2 m\rceil}$ representing an index h of a clause of φ between 1 and m in binary, when π_i was obtained from φ by weakening (i.e., if $R_{i,F} = 1$).
- Attributes $J_{i,1}, \ldots, J_{i,\lceil\log_2 L\rceil}$ and $K_{i,1}, \ldots, K_{i,\lceil\log_2 L\rceil}$ representing two prior indices between 1 and $i-1$ in binary, and $A_{i,1}, \ldots, A_{i,\lceil\log_2 n\rceil}$ representing the index a of an attribute between 1 and n in binary, when π_i was obtained by resolution on X_a from these two prior steps (i.e., $R_{i,R} = 1$)

$\mathrm{REF}_L(\varphi)$ then consists of the following clauses.

1. *Formula steps and resolution steps are mutually exclusive.* For $i = 1, \ldots, L$, $R_{i,F} \vee R_{i,R}$ and $\neg R_{i,F} \vee \neg R_{i,R}$.
2. *Formula steps contain the literals of the corresponding clause.* For $i = 1, \ldots, L$, $c = 1, \ldots, n$, $b \in \{0,1\}$, and $h = 1, \ldots, m$, such that X_c^b is in clause h of φ and $h_1, \ldots, h_{\lceil\log_2 m\rceil}$ is the binary representation of h,
$$\neg R_{i,F} \vee H_{i,1}^{1-h_1} \vee \cdots \vee H_{i,\lceil\log_2 m\rceil}^{1-h_{\lceil\log_2 m\rceil}} \vee Y_{i,c,b}$$
3. *Resolution steps do not contain the eliminated attribute.* For $i = 1, \ldots, L$, $j = 1, \ldots, i-1$, $k = 1, \ldots, i-1$, $a = 1, \ldots, n$, and $b \in \{0,1\}$
$$\neg R_{i,R} \vee J_{i,1}^{1-j_1} \vee \cdots \vee J_{i,\lceil\log_2 L\rceil}^{1-j_{\lceil\log_2 L\rceil}} \vee K_{i,1}^{1-k_1} \vee \cdots \vee K_{i,\lceil\log_2 L\rceil}^{1-k_{\lceil\log_2 L\rceil}} \vee A_{i,1}^{1-a_1} \vee \cdots \vee A_{i,\lceil\log_2 n\rceil}^{1-a_{\lceil\log_2 n\rceil}} \vee \neg Y_{i,a,b}$$

4. *Resolution steps contain all other literals of π_j.* For $i = 1, \ldots, L$, $j = 1, \ldots, i-1$, $a = 1, \ldots, n$, $c \neq a$ in $\{1, \ldots, n\}$, and $b \in \{0, 1\}$,
$$\neg R_{i,R} \vee J_{i,1}^{1-j_1} \vee \cdots \vee J_{i,\lceil \log_2 L \rceil}^{1-j_{\lceil \log_2 L \rceil}} \vee A_{i,1}^{1-a_1} \vee \cdots \vee A_{i,\lceil \log_2 n \rceil}^{1-a_{\lceil \log_2 n \rceil}} \vee \neg Y_{j,c,b} \vee Y_{i,c,b}$$
5. *Resolution steps contain all other literals of π_k.* For $i = 1, \ldots, L$, $k = 1, \ldots, i-1$, $a = 1, \ldots, n$, $c \neq a$ in $\{1, \ldots, n\}$, and $b \in \{0, 1\}$,
$$\neg R_{i,R} \vee K_{i,1}^{1-k_1} \vee \cdots \vee K_{i,\lceil \log_2 L \rceil}^{1-k_{\lceil \log_2 L \rceil}} \vee A_{i,1}^{1-a_1} \vee \cdots \vee A_{i,\lceil \log_2 n \rceil}^{1-a_{\lceil \log_2 n \rceil}} \vee \neg Y_{k,c,b} \vee Y_{i,c,b}$$
6. *In resolution steps, π_j contains X_a* For $i = 1, \ldots, L$, $j = 1, \ldots, i-1$, $a = 1, \ldots, n$
$$\neg R_{i,R} \vee J_{i,1}^{1-j_1} \vee \cdots \vee J_{i,\lceil \log_2 L \rceil}^{1-j_{\lceil \log_2 L \rceil}} \vee A_{i,1}^{1-a_1} \vee \cdots \vee A_{i,\lceil \log_2 n \rceil}^{1-a_{\lceil \log_2 n \rceil}} \vee Y_{j,a,1}$$
7. *In resolution steps, π_k contains $\neg X_a$* For $i = 1, \ldots, L$, $k = 1, \ldots, i-1$, $a = 1, \ldots, n$
$$\neg R_{i,R} \vee K_{i,1}^{1-k_1} \vee \cdots \vee K_{i,\lceil \log_2 L \rceil}^{1-k_{\lceil \log_2 L \rceil}} \vee A_{i,1}^{1-a_1} \vee \cdots \vee A_{i,\lceil \log_2 n \rceil}^{1-a_{\lceil \log_2 n \rceil}} \vee Y_{k,a,0}$$
8. *In resolution steps, $j \in \{1, \ldots, i-1\}$* For $i = 1, \ldots, L$, $j \notin \{1, \ldots, i-1\}$, $\neg R_{i,R} \vee J_{i,1}^{1-j_1} \vee \cdots \vee J_{i,\lceil \log_2 L \rceil}^{1-j_{\lceil \log_2 L \rceil}}$
9. *In resolution steps, $k \in \{1, \ldots, i-1\}$* For $i = 1, \ldots, L$, $k \notin \{1, \ldots, i-1\}$, $\neg R_{i,R} \vee K_{i,1}^{1-k_1} \vee \cdots \vee K_{i,\lceil \log_2 L \rceil}^{1-k_{\lceil \log_2 L \rceil}}$
10. *Clauses are not trivial.* For $i = 1, \ldots, L$ and $a = 1, \ldots, n$, $\neg Y_{i,a,0} \vee \neg Y_{i,a,1}$
11. *π_L is the empty clause.* For $c = 1, \ldots, n$, $b \in \{0, 1\}$, $\neg Y_{L,c,b}$

Now, if φ is in fact satisfiable, we know that the soundness of resolution ensures that there does not exist a refutation of φ of any length. Thus, in this case $\mathrm{REF}_L(\varphi)$ is unsatisfiable; indeed, we show it has a polynomial-length refutation, which completes the first case of the reduction. The resolution proof here formally follows the proof of soundness of resolution (Proposition 5.8) using a fixed satisfying assignment for φ.

Theorem 5.28 *For any satisfiable CNF formula φ on n attributes with m clauses and any integer L, there is a polynomial-length refutation of* $\mathrm{REF}_L(\varphi)$ *(in L, m, and n).*

Proof Let $x_1, \ldots, x_n$ be a satisfying assignment to the attributes $X_1, \ldots, X_n$ of φ. We'll show, by induction on i, how to prove that each ith step encoded by $\mathrm{REF}_L(\varphi)$ contains a satisfied literal, corresponding to the clause
$$Y_{i,1,x_1} \vee \cdots \vee Y_{i,n,x_n}.$$
Once we obtain $Y_{L,1,x_1} \vee \cdots \vee Y_{L,n,x_n}$, we'll be able to use the clauses asserting the final clause is empty ($\neg Y_{L,c,b}$ for all c and b) to complete the refutation in n steps.

For $i = 1$, observe that since no smaller j and k can be chosen, clauses of type 8 (or 9) can be used to derive $\neg R_{1,R}$, by resolving on all of the attributes $J_{i,1}, \ldots, J_{i,\lceil \log_2 L \rceil}$ in $2L - 1$ steps. Specifically, we pair up the clauses with indices that agree in all but the final bit, and resolve those clauses on $J_{i,\lceil \log_2 L \rceil}$; we then resolve the pairs obtained that have identical literals except on $J_{i,\lceil \log_2 L \rceil - 1}$ and so on, until all of these attributes are eliminated. We can resolve this with the clauses of type 1 to obtain $R_{1,F}$ – that is, π_1 is a clause of φ. Since we have assumed that $x_1, \ldots, x_n$ is a satisfying assignment to

φ, each clause $h = 1, \ldots, m$ has at least one satisfied literal. Thus, choosing one literal satisfied by the assignment for each clause and resolving the clauses of type 2 for the respective chosen satisfied literal for each clause h on the variables $H_{i,1}, \ldots, H_{i,\lceil \log_2 m \rceil}$ (as we did with the $J_{i,j}$ clauses earlier) in $2m-1$ steps, and then using $R_{1,F}$ to eliminate $\neg R_{1,F}$ from the clause, gives a subclause of $Y_{1,1,x_1} \vee \cdots \vee Y_{1,n,x_n}$ in $2L + 2m - 1$ steps, as needed. (We can weaken this clause to $Y_{1,1,x_1} \vee \cdots \vee Y_{1,n,x_n}$ for convenience.)

So, we suppose that we have obtained clauses $Y_{i',1,x_1} \vee \cdots \vee Y_{i',n,x_n}$ for $i' = 1, \ldots, i-1$. Observe that just as we did for $i = 1$, we can use clauses corresponding to the respective chosen satisfied literal for each clause $h = 1, \ldots, m$ of type 2, and eliminate the variables $H_{i,1}, \ldots, H_{i,\lceil \log_2 m \rceil}$ from these clauses in $2m-1$ steps to obtain $\neg R_{1,F} \vee Y_{i,1,x_1} \vee \cdots \vee Y_{i,n,x_n}$.

Next, we'll derive $\neg R_{1,R} \vee Y_{i,1,x_1} \vee \cdots \vee Y_{i,n,x_n}$ as follows. Here it is helpful to recall that we are formalizing the argument that the eliminated attribute x_a must have been falsified in one of the clauses π_j or π_k, and so one of the other satisfied literals must have been in that clause. So, first supposing $x_a = 0$, the falsified literal is the positive literal in π_j (by the type 6 clause). For each $j = 1, \ldots, i-1$ and $a \in \{1, \ldots, n\}$ such that $x_a = 0$, we can first take the previously obtained clause $Y_{j,1,x_1} \vee \cdots \vee Y_{j,n,x_n}$ and use the clauses of type 4 corresponding to $b = x_c$ for each $c \neq a$ to sequentially replace $Y_{j,1,x_1}, Y_{j,2,x_2}, \ldots, Y_{j,n,x_n}$ (except Y_{j,a,x_a}) from the last obtained clause with $Y_{i,1,x_1} \ldots Y_{i,n,x_n}$ respectively by resolution. We thus obtain clauses of the form

$$\neg R_{i,R} \vee J_{i,1}^{1-j_1} \vee \cdots \vee J_{i,\lceil \log_2 L \rceil}^{1-j_{\lceil \log_2 L \rceil}} \vee A_{i,1}^{1-a_1} \vee \cdots \vee A_{i,\lceil \log_2 n \rceil}^{1-a_{\lceil \log_2 n \rceil}} \vee Y_{j,a,x_a} \vee Y_{i,1,x_1} \vee \cdots \vee Y_{i,n,x_n}$$

for each $j = 1, \ldots, i-1$ and $a \in \{1, \ldots, n\}$ such that $x_a = 0$, in $(i-1)|\{a : x_a = 0\}|n$ steps total. Resolution with a clause of type 10 replaces $Y_{j,a,0}$ with $\neg Y_{j,a,1}$ in each of these, and then we can eliminate $\neg Y_{j,a,1}$ using the clauses of type 6 in each, for another $(i-1)|\{a : x_a = 0\}| \cdot 2$ steps. Now, we can use these together with the clauses of type 8 for $j \notin \{1, \ldots, i-1\}$ to eliminate the variables $J_{i,1}, \ldots, J_{i,\lceil \log_2 L \rceil}$ in $2L-1$ steps for each a, obtaining $\neg R_{i,R} \vee A_{i,1}^{1-a_1} \vee \cdots \vee A_{i,\lceil \log_2 n \rceil}^{1-a_{\lceil \log_2 n \rceil}} \vee Y_{i,1,x_1} \vee \cdots \vee Y_{i,n,x_n}$ for each a such that $x_a = 0$ using $[(i-1)(n+2) + (2L-1)]|\{a : x_a = 0\}|$ steps.

When $x_a = 1$, the negative literal $\neg x_a$ in π_k (per the type 7 clause) is falsified, so π_k must contain one of the other satisfied literals. We obtain the analogous clauses

$$\neg R_{i,R} \vee K_{i,1}^{1-k_1} \vee \cdots \vee K_{i,\lceil \log_2 L \rceil}^{1-k_{\lceil \log_2 L \rceil}} \vee A_{i,1}^{1-a_1} \vee \cdots \vee A_{i,\lceil \log_2 n \rceil}^{1-a_{\lceil \log_2 n \rceil}} \vee Y_{k,a,x_a} \vee Y_{i,1,x_1} \vee \cdots \vee Y_{i,n,x_n}$$

for $k = 1, \ldots, i-1$ and $a \in \{1, \ldots, n\}$ such that $x_a = 1$ in another $(i-1)|\{a : x_a = 1\}|n$ steps. We then eliminate Y_{k,a,x_a} using the clauses of type 7 and type 10 in another $(i-1)|\{a : x_a = 1\}| \cdot 2$ steps. Now, using the clauses of type 9 for $k \notin \{1, \ldots, i-1\}$, we eliminate the variables $K_{i,1}, \ldots, K_{i,\lceil \log_2 L \rceil}$ in $2L-1$ steps for each a to obtain $\neg R_{i,R} \vee A_{i,1}^{1-a_1} \vee \cdots \vee A_{i,\lceil \log_2 n \rceil}^{1-a_{\lceil \log_2 n \rceil}} \vee Y_{i,1,x_1} \vee \cdots \vee Y_{i,n,x_n}$ for each a such that $x_a = 1$ in another $[(i-1)(n+2) + (2L-1)]|\{a : x_a = 1\}|$ steps.

Using these clauses, we next eliminate the attributes $A_{i,1}, \ldots, A_{i,\lceil \log_2 n \rceil}$ in $2n-1$ steps to finally obtain $\neg R_{i,R} \vee Y_{i,1,x_1} \vee \cdots \vee Y_{i,n,x_n}$ as needed; overall, this took $(2n-1) + (i-1)(n+2)n + (2L-1)n$ steps.

We now use the clause $R_{i,F} \vee R_{i,R}$ with the above two clauses to obtain $Y_{i,1,x_1} \vee \cdots \vee Y_{i,n,x_n}$ in two more steps, using a total of $O(m + i \cdot n^2 + Ln)$ steps. Since we repeat this for $i = 2, \ldots, L$, the entire proof takes $O(Lm + L^2n^2)$ steps. □

On the other hand, if φ is unsatisfiable, depending on the length L, $\mathrm{REF}_L(\varphi)$ may or may not be satisfiable. If it is satisfiable, then the soundness of resolution ensures that the automating algorithm cannot output a proof and we are done. Otherwise, we can argue that $\mathrm{REF}_L(\varphi)$ does not have a *short* refutation when L is sufficiently large. We will show this using the length-width connection of Theorem 5.20, and in particular by giving a strategy for the Liar in the Prover-Liar formula game and invoking Theorem 5.22 to establish that any refutation must use wide clauses, much as we did for the pigeonhole principle (Theorem 5.23). In this case, the Liar's strategy will be guided by a naïve exponential-size resolution refutation, similar to the one constructed by Naïve-RES (Algorithm 6). Formally:

Definition 5.29 The *complete* formula on n attributes $X_1, \ldots, X_n$, κ_n, is a formula that contains all clauses of width n. The *naïve refutation* of an unsatisfiable formula φ on n attributes of length $2^{n+1} - 1$ is obtained as follows: for each $\vec{x} \in \{0,1\}^n$, some clause of φ that is falsified by $\vec{x}$ is weakened to the clause of width n that is falsified by $\vec{x}$, thus obtaining κ_n. Then the proof derives $\kappa_{n-1}, \kappa_{n-2}, \ldots, \kappa_1, \bot$ in order by pairing up clauses of κ_i that contain the same literals on $X_1, \ldots, X_{i-1}$ and resolving them on X_i, to obtain the clauses of κ_{i-1}.

Theorem 5.30 *For all sufficiently large n, for any unsatisfiable CNF formula φ with m clauses on n attributes and $L \geq n^2 + mn$, the resolution width complexity of* $\mathrm{REF}_L(\varphi)$ *is at least* $\Omega(L/n)$.

Proof Consider any family of Prover strategies that remember at most $o(L/n)$ attributes at a time. We'll show that for sufficiently large n, the Liar has a winning strategy as follows. We first partition the L steps of an L-step instance of $\mathrm{REF}_L(\varphi)$ as follows: we say the Lth step has *level 0*, and (nearly) evenly partition the steps $1, \ldots, L-1$ into blocks of size $(L-1)/n$, referring to the jth block as having *level j*. Level j steps will contain clauses from κ_j, so the final step contains the encoding of the empty clause (as required by the type 11 clause of $\mathrm{REF}_L(\varphi)$).

We'll say that the Prover's partial assignment *mentions* a step i of an L-step instance of $\mathrm{REF}_L(\varphi)$ if it includes one of the attributes with step index i. Let $\mathcal{M}$ denote the set of these mentioned steps, together with step L. For all such mentioned steps, the Liar will select settings for all of the associated variables: each step i will encode a clause of the associated level j, derived by resolution on the attribute X_{j+1} from steps associated with level $j+1$ if $j < n$; if $j = n$, then the variables will indicate that the clause is derived by weakening a clause of the input φ. Let $\partial\mathcal{M}$ denote the *boundary* of $\mathcal{M}$ maintained by the Liar, which includes the indices of steps of level $j+1$ mentioned by steps of level $j < n$ in $\mathcal{M}$. The Liar will also select consistent settings of the clause variables for each of these—in particular, each specifying a clause of level $j+1$ that extends the clause of level j by including the literal X_{j+1} or $\neg X_{j+1}$ as appropriate to satisfy the step in $\mathcal{M}$ referencing that step of $\partial\mathcal{M}$.

When the Prover asks for an attribute of a step in $\mathcal{M}$, the Liar returns the previously chosen value of that attribute. If the Prover asks for an attribute of a step in $\partial\mathcal{M}$, the step is added to $\mathcal{M}$ and if $j < n$, the Liar chooses settings for the variables referencing steps associated with level $j+1$ that were neither previously mentioned in $\mathcal{M}$ nor in $\partial\mathcal{M}$, and adds them to $\partial\mathcal{M}$, choosing an appropriate extension of the clause with a literal using X_{j+1} for each of their associated clauses. Thus, the local assignment will satisfy the clauses of $\mathrm{REF}_L(\varphi)$ of type 1 and 3–10. Observe that for sufficiently large n, the $o(L/n)$ function bounding the number of attributes remembered by the Prover is less than $\frac{L-1}{3n}$, where each such attribute may result in one step being added to $\mathcal{M}$ and two others to $\partial\mathcal{M}$; thus, since there are $\frac{L-1}{n}$ steps allocated to each level, there must be two steps for that level that did not previously belong to $\mathcal{M}$ or $\partial\mathcal{M}$. On the other hand, if the step is of level n, then the Liar chooses a clause of φ that can be weakened to the clause chosen for the step. Such a clause must exist because φ was assumed to be unsatisfiable, and hence the unique assignment that falsifies this level n clause must also falsify some clause of φ; the falsified clause of φ therefore must either be equal to this level n clause or contain a subset of its literals. Either way, we can indeed obtain the desired level n clause from some clause of φ, so the clauses of $\mathrm{REF}_L(\varphi)$ of type 2 will be satisfied. If the Prover asks for an attribute of a step that is neither in $\mathcal{M}$ nor in $\partial\mathcal{M}$, the Liar chooses an arbitrary clause of the appropriate level (e.g., always the clause $X_1 \vee \cdots \vee X_j$) for that step and two indices of level $j+1$ with associated clauses if $j < n$ or a suitable clause of φ if $j = n$, similar to if the step were in $\partial\mathcal{M}$. Such clauses and steps may be chosen by the same argument, and the partial assignments chosen by the Liar will thus continue to satisfy the clauses of $\mathrm{REF}_L(\varphi)$.

If the Prover chooses to forget some attributes, let $\mathcal{M}' \subset \mathcal{M}$ be the new set of steps mentioned by the attributes remembered by the Prover. If there are steps in $\mathcal{M} - \mathcal{M}'$ that are still in $\partial\mathcal{M}'$, the Liar can retain the clause associated with that step and merely forget the steps of the next level it was claimed to be obtained from. Otherwise, everything about the step is forgotten by the Liar as well. Either way, the Liar also forgets everything about the steps in $\partial\mathcal{M} - \partial\mathcal{M}'$. The Liar will be able to subsequently re-use these forgotten steps with new clauses if necessary. Thus, we see that the Prover will never discover a violated clause of $\mathrm{REF}_L(\varphi)$. Hence, Theorem 5.22 yields that the resolution width complexity of $\mathrm{REF}_L(\varphi)$ is at least $\Omega(L/n)$ as claimed. □

We can finally complete the proof of Theorem 5.26.

Proof of Theorem 5.26. Suppose that resolution were efficiently automatable. We first observe that given φ, $\mathrm{REF}_L(\varphi)$ for $L \geq n^2 + mn$ can indeed be constructed in polynomial time. It has $O(Lnm + L^3n + L^2n^2)$ clauses, each of width $O(\log m + \log L + \log n)$. If φ is satisfiable, then by Theorem 5.28 $\mathrm{REF}_L(\varphi)$ has a polynomial-length refutation in L, m, and n. Then, the automating algorithm for resolution is guaranteed to return a refutation of $\mathrm{REF}_L(\varphi)$ in polynomial time, in terms of m and n. Let our reduction be as follows: on input φ, construct $\mathrm{REF}_L(\varphi)$ for $L = n^2 + mn$, and run the automating algorithm for this polynomial number of steps; if it outputs a correct refutation of $\mathrm{REF}_L(\varphi)$, return that φ is satisfiable, and otherwise return that φ is unsatisfiable. We thus see already that if φ is satisfiable, the reduction is correct.

On the other hand, if φ is unsatisfiable, first suppose $\mathrm{REF}_L(\varphi)$ is satisfiable. Then by the soundness of resolution, the automating algorithm cannot return a correct refutation, so the reduction will correctly report that φ is unsatisfiable. Finally, if $\mathrm{REF}_L(\varphi)$ is unsatisfiable, Theorem 5.30 guarantees that its width complexity is $\Omega(L/n) \geq \Omega(n+m)$. So, since the width of $\mathrm{REF}_L(\varphi)$ is $O(\log m + \log L + \log n) = O(\log m + \log n)$, for sufficiently large n, Theorem 5.20 gives that any refutation of $\mathrm{REF}_L(\varphi)$ must have length at least

$$2^{\Omega((w(\mathrm{REF}_L(\varphi)\vdash\bot)-w(\mathrm{REF}_L(\varphi)))^2/n)} \geq 2^{(C(n+m-\log n-\log m))^2/n} \geq 2^{\Omega(n+\frac{m^2}{n})}.$$

Hence, for sufficiently large n, the automating algorithm will not be run for long enough to return a refutation of $\mathrm{REF}_L(\varphi)$, and again the algorithm will correctly report that φ is unsatisfiable. We can finally complete the reduction by hard-coding in the answers for a finite number of instances where n is not sufficiently large. Thus, we would obtain a polynomial-time algorithm for deciding CNF satisfiability and thereby obtain P = NP since this problem is NP-complete. □

5.6 Satisfiability for planning: reasoning in action.

To illustrate the use of deductive reasoning, we'll show how a family of basic task-planning problems can be efficiently reduced to CNF satisfiability. Moreover, we'll observe that unit propagation is quite powerful on the resulting instances, which helps explain the effectiveness of these methods in practice. Even though there are some simple formulas that SAT solvers fail to solve in reasonable time, these algorithms are still effective for many of the instances we obtain when reducing other problems to satisfiability. Hence, such reductions can sometimes be an effective strategy for algorithm design.

We actually already presented an example of the reduction earlier, in Example 3.3, using the representation of the environment in attributes described in Section 2.1. In this section, we'll be generalizing these examples, to find a method for solving such problems in general.

In the setting we consider here, an environment has a *state* that is described by some finite number of propositional attributes, n, that may take different values over time. We refer to these as *fluents*; the state of a fluent at any point in time t, given by an integer, is a propositional variable. Supposing that our n fluents are $X_1, \ldots, X_n$, we thus associate each X_i with a family of propositional variables $X_{i,0}, X_{i,1}, \ldots, X_{i,t}, \ldots$ where the state of the environment at time t is thus defined by the values of the propositional attributes $X_{1,t}, X_{2,t}, \ldots, X_{n,t}$. (Again, a more concrete example appeared in in Section 2.1.)

We will consider the problems faced by an *agent* that has at its disposal a set of *actions*. Each action is associated with a *precondition* property and a collection of *effects*. In the simple environments we consider, the preconditions will be given by a conjunction over the fluents, and the effects will be a set of literals over the fluents. The interpretation is that the agent may only use the action at a time t if the precondition

holds on the attributes $X_{1,t}, X_{2,t}, \ldots, X_{n,t}$. Then, the environment state changes so that the effect literals are satisfied in the environment at time $t + 1$. Attributes that do not appear in an effect take the same value that they had at time t. Thus, the state at time $t + 1$ is completely determined by the action together with the state at time t.

We will consider problems in which the agent seeks to bring the environment into a state satisfying a given *goal* property.

Definition 5.31 A *classical planning problem* is given by parameters indicating the number of fluents and actions; a (possibly empty) precondition conjunction and list of effect literals for each action; and a pair for the instance, consisting of an *initial state*, i.e., values for the attributes at time $t = 0$, and a *goal* given by a conjunction over the fluents. A *plan* is a sequence of actions indexed by time $t = 1, 2, \ldots, T$ such that the precondition of each tth action is satisfied by the state at time $t - 1$. The plan is a solution to the classical planning problem if the goal is true of the state at time T.

A *bounded-horizon classical planning problem* is given by a classical planning problem together with a *horizon* bound h on the length of a plan. A plan is a solution to this problem if it is a plan solving the planning problem given by a sequence of actions of length at most h. We will represent the bound on the horizon in unary, i.e., by a string $\overbrace{11\cdots1}^{h}0$.

The use of a unary representation of the horizon ensures that the search problem is polynomially balanced – solutions are given by sequences bounded by the length of the input – and thus formally an NP-search problem. Without this restriction, there may exist problems for which plans exist, but only of exponential length. With the unary horizon bound, it is immediate that the bounded-horizon classical planning problem reduces to any NP-complete problem in polynomial time. Actually, it is also easy to show that the family of bounded-horizon classical planning problems is NP-complete. (We leave this as an exercise.) We'll now show that this problem has in particular a rather direct reduction to CNF satisfiability that is solved well by CDCL SAT-solvers.

For our reduction to satisfiability, we'll have an attribute $X_{i,t}$ for each fluent X_i at each time index t from 0 to the horizon bound h. For each jth action and time index $t = 1, \ldots, h$, we will have a corresponding attribute $A_{j,t}$ that we interpret as indicating if the jth action is the action taken at step t of a plan. We will have a "no-op" action that has an empty precondition and no effects to the given set of actions, that will correspond to all $A_{j,t} = 0$. Note that we will be able to omit any occurrence of the no-op action from a plan solving the planning problem and still obtain a plan that solves the problem. (It permits solutions of length less than h to be "padded" to have length exactly h.)

Now, the instance will have the following clauses.

- Initial state: for each setting x_i of the fluent X_i in the initial state, we include a clause consisting of a single literal ℓ such that $\ell(x_i)$ is true: $X_{i,0}$ if $x_i = 1$ and $\neg X_{i,0}$ otherwise.

- Mutually exclusive action: for each time index $t = 1, \ldots, h$ and each pair of distinct actions j and k, include the clause $\neg A_{j,t} \vee \neg A_{k,t}$.
- Preconditions: for each time index $t = 1, \ldots, h$, for each jth action, if a literal ℓ is in the precondition of the jth action, we include the clause $\neg A_{j,t} \vee \ell_{t-1}$.
- Effects: for each time index $t = 1, \ldots, h$, and each jth action, if a literal ℓ is in the effects of the jth action, we include the clause $\neg A_{j,t} \vee \ell_t$.
- Frame axiom: for each literal on a fluent ℓ, let $\mathcal{A}_\ell$ denote the set of actions that have ℓ as an effect. For each time index $t = 1, \ldots, h$ and literal ℓ, we include the clause $\bigvee_{j \in \mathcal{A}_\ell} A_{j,t} \vee \ell_{t-1} \vee \neg \ell_t$.
- Goal: for each literal ℓ in the goal conjunction, include the clause consisting of the literal ℓ_h.

A concrete example of such a clausal encoding was given previously, in Example 3.3. Observe that it is easy to construct this clausal encoding from any reasonable representation of the planning problem where there are n fluents and a actions in time $O(a^2 + nah)$.

Theorem 5.32 *For a given bounded-horizon planning problem, for each h-step plan, the partial assignment that sets $A_{j,t} = 1$ for the action j taken on step t in the plan, and $A_{j',t} = 0$ for all other j' is consistent with a unique satisfying assignment to the clausal encoding with the Goal clauses omitted. In particular, unit propagation from this partial assignment yields this satisfying assignment. Conversely, each satisfying assignment to the clausal encoding with the Goal clauses omitted is consistent with the partial assignment obtained from some plan.*

Proof Since unit propagation is a sound rule of inference, it will suffice to show that unit propagation completes the satisfying assignment. Thus, let the partial assignment corresponding to a plan be given. We will show by induction on t that unit propagation obtains the assignment to $X_{1,t}, \ldots, X_{n,t}$ that is identical to the state of the environment at time t. For time $t = 0$, we observe that unit propagation applied to the Initial state clauses sets each $X_{i,0}$ to the initial state value x_i. Thus, suppose that the claim holds for $t - 1$. Then for each literal ℓ that is an effect of the action j for which $A_{j,t} = 1$, the effect axioms simplify to ℓ_t under the given partial assignment, so unit propagation will set the corresponding fluent so that $\ell_t = 1$. For all literals ℓ that are not effects of this action, since $j \notin \mathcal{A}_\ell$ and all other $A_{k,t}$ (for $k \neq j$) are false, if ℓ was false prior at time $t - 1$, by the induction hypothesis the frame axiom will simplify to the unit clause $\neg \ell_t$, so ℓ_t will be set false by unit propagation. (Observe, if ℓ_{t-1} was true, the literal $\neg \ell_{t-1}$ was false, and unit propagation in this case sets $\neg \ell_t$ false, i.e., sets $\ell_t = 1$.) This completes the induction step.

Finally, to see that this is indeed a satisfying assignment, we note that since at most one action j has $A_{j,t} = 1$, the mutual exclusion clauses are satisfied, and since the plan was valid, the preconditions of the action j taken at step t were satisfied. Thus the precondition clauses for this action are satisfied in this unique assignment, and unit propagation has ensured that the effect clauses are satisfied for this action, and the

frame axioms are satisfied. For all other actions k, the corresponding precondition and effects clauses are satisfied since $A_{k,t} = 0$.

For the converse, we again argue by induction on t that if we let the assignment of all $A_{j,t} = 0$ correspond to our no-op action, the assignment to the $A_{j,t}$ attributes corresponds to a valid plan with the assignment to $X_{1,t}, \ldots, X_{n,t}$ identical to the state of the environment at time t. First observe that in any satisfying assignment to the clausal encoding, only the setting $(X_{1,0}, \ldots, X_{n,0}) = (x_1, \ldots, x_n)$ given as the initial state satisfies the initial state clauses. So, suppose the claim holds up to step $t - 1$. Next, observe that by the mutual exclusion clauses, we cannot have for any pair of distinct actions $j \neq k$ $A_{j,t} = 1$ and $A_{k,t} = 1$ for any t. If all $A_{j,t} = 0$, then (as in our analysis of unit propagation) the frame axiom clauses are only satisfied if for all literals ℓ, $\ell_t = \ell_{t-1}$, which indeed agrees with the effect of the no-op. Otherwise, for the unique j such that $A_{j,t} = 1$, since the assignment to the fluents at time index $t - 1$ contains the state at time $t - 1$ by induction hypothesis and the precondition clauses are satisfied, it must be that all of the literals in the precondition of this action are satisfied. Furthermore, since the effect clauses are satisfied and $A_{j,t} = 1$, the effects of the action are true in the assignment to the fluents at time t. Finally, since the frame axioms are satisfied for the literals that are not effects of this action, the $A_{k,t}$ occurring in the frame axiom clause for these literals are all false, and hence the literals satisfy $\ell_t = \ell_{t-1}$. This completes the induction step. □

Corollary 5.33 *For a given bounded-horizon planning problem, for each H-step plan solving the problem, there is a a unique satisfying assignment to the clausal encoding consistent with the partial assignment that sets $A_{j,t} = 1$ for the action j taken on step t in the plan, and $A_{j',t} = 0$ for all other j'. In particular, unit propagation from this partial assignment yields this satisfying assignment. Conversely, each satisfying assignment to the clausal encoding is consistent with the partial assignment obtained from some plan solving the problem.*

Proof If the given plan solves the given problem, then the literals of the goal conjunction are satisfied at step $t = h$. Since unit propagation infers that the attributes $X_{1,h}, \ldots, X_{n,h}$ take the value corresponding to the state at time $t = h$, the Goal clauses are indeed satisfied by this assignment as well. And, conversely, in any satisfying assignment, since the Goal clauses are satisfied and we have argued that the action attributes contain a valid plan consistent with the fluents taking that final value, the satisfying assignment indeed corresponds to a plan solving the problem. □

We stress that unit propagation thus effectively reduces the search for a satisfying assignment to the search for a sequence of actions. Note that unit propagation also infers $A_{k,t} = 0$ from the mutual exclusion clauses once some other $A_{j,t} = 1$. Although one does not obtain a state-of-the-art planner by combining the above reduction with a generic CDCL SAT-solver – it is still useful to have heuristic guidance as to *which* actions are more promising to consider – such a generic approach is still reasonably effective in practice.

5.7 Tractable fragments of resolution.

Although algorithms such as CDCL SAT solvers are complete, we do not have any guarantees on their running time. We know that they can only determine unsatisfiability when the input has a small resolution proof (Proposition 5.18), but that this is only a necessary condition, not a guarantee that they will find such refutations quickly. We know by Theorem 5.26 moreover, that unless P = NP, they indeed must fail at discovering some such refutations in any polynomial time bound. We now turn to considering syntactic restrictions of resolution proofs, generally referred to as *fragments* of the proof system, for which we can obtain some meaningful guarantees.

We already have seen one such fragment: the *treelike* proofs introduced in Definition 5.14, that capture the kinds of resolution proofs returned by DPLL (Proposition 5.15). DPLL, like CDCL SAT solvers, also does not provide any guarantee that it finds small treelike resolution proofs when they exist. But, unlike general resolution proofs, we will see that it is possible to obtain nontrivial running time guarantees for automating treelike resolution, via a slightly modified algorithm.

Towards the development and analysis of this algorithm for automating treelike resolution, we will need to first introduce a new complexity measure for resolution proofs, capturing the amount of memory space needed to verify the proof. To describe this complexity measure, we will first need to revisit our definition of a proof.

Definition 5.34 For a proof system given by the class of representations $\mathcal{F}$ and inference relation R, a *configurational proof* of $q \in \mathcal{F}$ from hypotheses $h_1, \ldots, h_k \in \mathcal{F}$ is given by a sequence of representations from $\mathcal{F}$ $\Pi = \pi_1, \ldots, \pi_\ell$ together with a sequence of sets of representations from $\mathcal{F}$ $C_0, C_1, C_2, \ldots, C_{\ell-1}$ with $C_0 \subseteq \{h_1, \ldots, h_k\}$, called *configurations*, and such that for each $i = 1, \ldots, \ell$,

- for some $\pi'_1, \ldots, \pi'_r \in C_{i-1}$, the tuple $(\pi_i, \pi'_1, \ldots, \pi'_r) \in R$ and
- $C_i \subseteq C_{i-1} \cup \{h_1, \ldots, h_k, \pi_i\}$.

It may be helpful to think of the configurations as the contents of a "blackboard," on which a collection of formulas we have inferred are written. The rule constraining the sequence of configurations essentially asserts that we are allowed to add as many formulas from the hypotheses (in addition to the line we just derived) or erase as many formulas from the blackboard as we wish in between steps of the proof. But, each inference of the proof can only use the formulas currently written on the blackboard. This is in contrast to the original definition, that permitted the use of any prior line of the proof in an inference, effectively taking the configurations to consist of the entire sequence of previous steps of the proof.

Example 5.7: We revisit Example 5.4 using the new language of configurations. Recall that we had a set of hypotheses dog, young, ¬dog ∨ ¬young ∨ puppy, and ¬puppy ∨ mischief and we wished to prove mischief by refutation, i.e., prove the

query $\bot$ from the set of hypotheses with ¬mischief added.

Configuration	Inference	Justification
{dog, ¬dog ∨ ¬young ∨ puppy}		(Initial configuration)
{young, ¬young ∨ puppy}	¬young ∨ puppy	dog, ¬dog ∨ ¬young ∨ puppy
{¬puppy ∨ mischief, puppy}	puppy	young, ¬young ∨ puppy
{¬mischief, mischief}	mischief	puppy, ¬puppy ∨ mischief
	$\bot$	mischief, ¬mischief

We note that configuration proofs yield a convenient alternative characterization of treelike proofs that we will use:

Proposition 5.35 *Suppose that a configuration proof $C_0, \pi_1, C_1, \ldots, C_{\ell-1}, \pi_\ell$ from hypotheses $h_1, \ldots, h_k$ satisfies for each $i = 1, \ldots, \ell - 1$*

$$C_i \subseteq \{h_1, \ldots, h_k, \pi_i\} \cup C_{i-1} - \{\pi'_1, \ldots, \pi'_r\}$$

where $\pi'_1, \ldots, \pi'_r$ is the justification of π_i. Then the proof is treelike. Conversely, any treelike proof has a configuration proof of this form.

Proof Recall that a proof is treelike if each line is used as a justification for at most one other line. Since the constraint on the configurations of the proof guarantees that the representations $\pi'_1, \ldots, \pi'_r$ used in the justification of line i are removed from C_i, the proof must be treelike. Conversely, if a proof is treelike, the representations used in the justification of a line are not used as the justification in any subsequent line, and hence can be removed from C_i. □

Our main purpose in introducing the configurations was to characterize the memory space complexity of a resolution proof as follows:

Definition 5.36 The *clause space complexity* of a configuration resolution proof is the maximum number of clauses in any of the configurations in the proof.

Example 5.8: The configuration proof in Example 5.7 has clause space complexity 2, since each configuration contains at most two clauses.

Indeed, for every chaining proof, there is a treelike clause-space two resolution refutation. It turns out, more specifically, that treelike clause space two is equivalent to unit propagation: see Exercise 5.7.

The clause space of a treelike resolution proof has an important alternative combinatorial characterization:

Definition 5.37 The *rank* or *Strahler number* of a rooted binary tree is inductively defined as follows:

- Leaves have rank 0.
- A node with two child subtrees of rank r has rank $r + 1$.

- Otherwise, the rank of a node is equal to the maximum rank of its child subtrees.

Note that rank was previously considered in Exercise 4.3, which related the rank of a tree to a bound on its size. We also leave the following characterization as an exercise:

Lemma 5.38 *A treelike resolution proof has a configuration proof of clause space s if and only if its graph is a tree of rank $s-1$.*

In particular, treelike proofs with a fixed clause space bound s have polynomial size, $n^{O(s)}$.

Now, we will show that the treelike bounded-clause-space fragment of resolution is efficiently automatable. The algorithm, Space-RES, is obtained from our previous backtracking search algorithms, in particular Naïve RES (Algorithm 6) or DPLL, by limiting the depth of assignments explored by the algorithm. At the same time, we extend the algorithm to iterate over all possible assignments to free variables in a recursive call. Crucially, once some assignment is found that yields a proof with a smaller depth bound, the algorithm stops searching over assignments to the free variables, and only searches for a refutation with the complementary assignment to that attribute using the original depth bound. Complete pseudocode for Space-RES is given in Algorithm 8.

Theorem 5.39 *Space-RES, run with the empty partial assignment, automates treelike clause-space s resolution, $s \geq 1$, in time $O(|\varphi|^{2s-1})$ where $|\varphi|$ is the representation size of φ in bits. If no treelike clause-space s resolution refutations of the input clauses exist, it returns $FAIL$ in time $O((|\varphi|)^{2s-1})$.*

Proof We show by induction on the number of free attributes in the input partial assignment that Space-RES returns a resolution proof Π such that $\Pi|_\rho$ is a treelike clause-space s refutation of $\varphi|_\rho$ in time $O((2n+|\varphi|)^{2s-1})$. If there are no free variables in the partial assignment, then the input clauses are evaluated to $\top$ or $\bot$. If all are $\top$, $\varphi|_\rho = \top$, and indeed the algorithm correctly reports $FAIL$ since $\varphi|_\rho$ is satisfied. Conversely, if some clause is $\bot$, then there is a trivial refutation consisting of that clause. In either case, the algorithm runs in time $O(|\varphi|)$. Also note, the only refutation of space $s < 2$ is the trivial refutation, and without loss of generality we may assume each attribute appears at least once in φ so $|\varphi| \geq n$.

So, we suppose that we have shown the claim when $n-1$ free variables remain. Suppose that there is a treelike clause-space $s \geq 2$ resolution proof π^* such that $\pi^*|_\rho$ is a refutation of $\varphi|_\rho$. Then by Lemma 5.38, since the graph of such a proof is a tree of rank at most $s-1$, it must be that one of its child subtrees has rank at most $s-2$, corresponding to a treelike proof π_1^* of clause space at most $s-1$. Let π_0^* denote its sibling subtree, which has rank at most $s-1$/clause space at most s. In particular, since $\pi^*|_\rho$ is a refutation, $\pi_1^*|_\rho$ must end with some literal ℓ and $\pi_0^*|_\rho$ must end with $\neg\ell$, so that $\pi^*|_\rho$ ends with $\bot$ after a resolution step on these clauses.

Now, we see that if Space-RES is run with the partial assignment ρ extended by $\ell \leftarrow 1$ and bound $s-1$, by the induction hypothesis, since there is a treelike clause-space $s-1$ refutation, π_1^*, and the partial assignment leaves at most $n-1$ free variables,

Input: Conjunctive Normal Form formula φ on n attributes $X_1, \ldots, X_n$, partial assignment ρ, clause space bound s.
Output: A clause space-s treelike resolution proof π such that $\pi|_\rho$ is a refutation of $\varphi|_\rho$ or $FAIL$ if none exist.

begin
  if $\varphi|_\rho = \top$ **then return** $FAIL$
  else if $\varphi|_\rho$ *contains an empty clause* $\bot$ **then return** a clause C of φ such that $C|_\rho$ is $\bot$
  else if $s > 1$ **then foreach** *literal* ℓ *on an attribute* X_i *free in* ρ **do**
    if *Space-RES*$(\varphi, (\rho, \ell \leftarrow 1), s-1)$ *returns* π_1 **then**
      if π_1 *ends with* C *such that* $C|_\rho$ *is* $\bot$ **then return** π_1
      else if *Space-RES*$(\varphi, (\rho, \ell \leftarrow 0), s)$ *returns* π_0 **then**
        if π_0 *ends with* C *such that* $C|_\rho$ *is* $\bot$ **then return** π_0
        else
          // π_1 and π_0 end with clauses $C_1 = \neg\ell \vee C_1'$ and $C_0 = \ell \vee C_0'$, respectively, such that $C_1|_\rho$ is $\neg\ell$ and $C_0|_\rho$ is ℓ
          return π such that $\pi|_\rho$ is a space s refutation of $\varphi|_\rho$:

$$\begin{array}{lll} & \pi_0 & \\ \{C_0\} \cup \ldots & \pi_1 & \\ & C_1' \vee C_0' & C_1, C_0 \end{array}$$

      end
      else return $FAIL$
  end
  return $FAIL$
end

Algorithm 8: Space-RES: an automating algorithm for clause space-s treelike resolution

Space-RES will return some treelike clause-space $s-1$ resolution proof π_1 such that $\pi_1|_\rho$ either ends with $\neg\ell$ or $\bot$. If it ends with $\bot$, Space-RES returns π_1 which is as needed. Otherwise, it recursively calls itself on the partial assignment extended by $\ell \leftarrow 0$ and space bound s. Again, since this partial assignment has at most $n-1$ free variables and there exists a treelike clause-space s proof, π_0^*, such that $\pi_0^*|_{\rho,\ell\leftarrow 0}$ is a refutation, by the induction hypothesis Space-RES again returns π_0 that is a treelike clause-space s resolution proof such that $\pi_0|_\rho$ either ends with ℓ or $\bot$. Again, if it ends with $\bot$, Space-RES correctly returns π_0, and otherwise it returns the following treelike clause-space s resolution proof: it first follows the clause-space s proof π_0, which ends with $C_0 = \ell \vee C_0'$. Then it follows π_1 with C_0 included in each of the configurations. π_1 ended with $C_1 = \neg\ell \vee C_1'$, so the final configuration is $\{C_0, C_1\}$. Thus, the proof may

conclude by resolution on ℓ using C_0 and C_1, and remove these from the configuration. Since we have that $C_0|_\rho = \ell$ and $C_1|_\rho = \neg\ell$, $C_0' \vee C_1'|_\rho = \bot$, this proof is a refutation under ρ as needed.

As for the running time, let c be the constant such that $c(n-1+|\varphi|)^{2s-1}$ is an upper bound on the running time for $n-1$ attributes and all s. Observe that Space-RES when there are n free attributes makes at most $2n$ calls on $n-1$ attributes with space $s-1$ (for each of the $2n$ assignments) and then makes at most one call with space s. It also performs the initial check for the value of the clauses of φ (base cases) in time $c'|\varphi|$. Thus, the overall total running time is at most

$$2n \cdot c(n-1+|\varphi|)^{2(s-1)-1} + c(n-1+|\varphi|)^{2s-1} + c|\varphi|.$$

Since we only consider $s \geq 2$, $2(s-1)-1 = 2s-3 \geq 1$ and $2s-1 \geq 3$. Thus, we can introduce/pull out a factor of $(n-1+|\varphi|)$ for the first two terms and see that we are seeking to bound an expression that is at most

$$\begin{aligned}
&c\left(\frac{2n}{n-1+|\varphi|} + n - 1 + |\varphi| + \frac{|\varphi|}{(n-1+|\varphi|)^2}\right)(n-1+|\varphi|)^{2s-2} \\
&\leq c\left(\frac{2n(n-1+|\varphi|)+|\varphi|}{(n-1+|\varphi|)^2} + n - 1 + |\varphi|\right)\left(\frac{n-1+|\varphi|}{n+|\varphi|}\right)^2 (n+|\varphi|)^{2s-2} \quad (s \geq 2) \\
&\leq c\left(\underbrace{\frac{2n(n-1+|\varphi|)+|\varphi|}{(n+|\varphi|)^2}}_{} + n - 1 + |\varphi|\right)(n+|\varphi|)^{2s-2} \\
&\leq c\,(\qquad 1 \qquad + n - 1 + |\varphi|)\,(n+|\varphi|)^{2s-2} \qquad\qquad (n \leq |\varphi|) \\
&\leq c(n+|\varphi|)^{2s-1}.
\end{aligned}$$

This completes the induction step.

We similarly observe that if Space-RES returns a resolution proof, it is a treelike clause-space s refutation. Thus, if no such proof exists, the algorithm returns $FAIL$ as claimed. □

Furthermore, on account of the relationship between tree size and rank, we also immediately obtain *quasipolynomial*-time automation of treelike resolution:

Corollary 5.40 *Treelike resolution of length L refutations of a CNF φ is automatable in time $O(|\varphi|^{2\log_2 L+1})$.*

Proof Recall that a rank r tree must have at least 2^r nodes. Since each node of the graph corresponding to a resolution proof is a proof step, we see that a length L treelike refutation has rank at most $\log_2 L$, hence clause space at most $\log_2 L+1$ by Lemma 5.38. Thus, we obtain the claimed bound from Theorem 5.39 using $s = \log_2 L + 1$. □

We also remark that bounded-width resolution is another efficiently automatable fragment, see Exercise 5.9.

5.8 Feasible interpolation and the hardness of weak automation.

Although we have essentially ruled out the possibility of automating resolution, there remains a possibility that there may exist an efficient algorithm that, for the purposes of deciding entailment, is as good as automating resolution—just as long as the algorithm cannot be used to reliably estimate the lengths of resolution proofs. The situation is similar to what we encountered in learning: although finding a 3-term DNF formula fitting the data was intractable, efficient algorithms could always find 3-CNF formulas. We referred to this as "improper" learning of 3-term DNF formulas. A direct analogue of improper learning for automation is known as *weak automation*:

Definition 5.41 We say that a proof system P is *weakly automatable* by a proof system Q if there is an *automating* algorithm that, when given a formula φ and knowledge base KB containing n attributes, such that φ is provable from KB in the proof system P by a proof of total size S, outputs a proof of φ in the system Q in time $t(n, S)$ where t is a polynomial in n and S.

As with improper learning, weak automation is particularly useful in establishing the limits of efficient reasoning: if a proof system cannot be weakly automated, then it captures consequences that are truly beyond the reach of efficient algorithms. And, just as with improper learning, the examples of such difficult consequences are found in cryptography.

The hard statements capture the "binding" guarantee of *commitment schemes*. Here, one party wishes to commit to a bit – say, the outcome of a coin toss – that may be revealed later, which we call "opening" the commitment. In this way, another party can call the coin toss, and the two parties can be confident that the outcome is random, even if they are only communicating electronically. Formally:

Definition 5.42 A *commitment scheme* is given by a pair of polynomial-time computable functions $Open$ and $Commit$. $Commit$ takes as input a *secret key* S and a *message* $B \in \{0, 1\}$, and produces as output a *commitment* C. $Open$ takes as input a purported secret key S, message B, and commitment C, and outputs *accept* or *reject*, satisfying the following properties.

- *Completeness*: For $C = Commit(S, B)$, $Open(S, B, C)$ outputs accept.
- *Perfect binding*: for $C = Commit(S, B)$, there does not exist an S' such that $Open(S', \neg B, C)$ outputs accept.
- *Computational hiding*: For any polynomial-time algorithm Alg, if S is chosen uniformly at random from $\{0, 1\}^n$, the advantage of Alg at guessing B is negligible in n, i.e., for all positive polynomials $p(n)$,

$$\left| \Pr_{S \in \{0,1\}^n} [Alg(Commit(S, 0)) = 1] - \Pr_{S \in \{0,1\}^n} [Alg(Commit(S, 1)) = 1] \right| < \frac{1}{p(n)}.$$

Other varieties of commitment schemes exist, but this is the one that is relevant for us: The hard theorems correspond to the perfect binding property of a commitment scheme, namely proving that it is impossible to commit to both 0 and 1. Computational

hiding will translate directly to a guarantee that the proof system cannot be automated as we will see. This is because automatable proof systems have a property known as *feasible interpolation*.

Definition 5.43 We say that a proof system has *feasible interpolation* if there is a polynomial p such that for any pair of formulas $\varphi_0(\vec{A},\vec{C})$ and $\varphi_1(\vec{B},\vec{C})$ on n attributes $\vec{A},\vec{B},\vec{C}$ such that $\varphi_0(\vec{A},\vec{C})\wedge\varphi_1(\vec{B},\vec{C})$ has a refutation π of size S, there is a algorithm $I(\vec{C})$ running in time $p(n,S)$ called an *interpolant* such that for any assignment $\vec{a},\vec{b},\vec{c}$, if $I(\vec{c})=0$ $\phi_0(\vec{a},\vec{c})$ is false and if $I(\vec{c})=1$, $\varphi_1(\vec{b},\vec{c})$ is false

In particular, if $\varphi_0(\vec{S},\vec{C})$ formalizes "$\vec{C}$ is a commitment to 0 with secret key $\vec{S}$" and if $\varphi_1(\vec{S},\vec{C})$ formalizes "$\vec{C}$ is a commitment to 1 with secret key $\vec{S}$" then the perfect binding property corresponds to $\varphi_0(\vec{S},\vec{C})\wedge\varphi_1(\vec{S}',\vec{C})$ being unsatisfiable, but a polynomial-time interpolant would break the hiding property of the commitment. Feasible interpolation is possible whenever a proof system is weakly automatable:

Theorem 5.44 *Let P be a weakly automatable proof system that is closed under restrictions. Then P has feasible interpolation.*

Proof Since we suppose P is weakly automatable, let Q be the system produced by an automating algorithm Alg with polynomial time bound $t(S,n)$. Then $p(S,n) = t(S,n)+O(S+n)$ will be our polynomial bound for the running time of the interpolating algorithm given in Algorithm 9:

Input: Setting $\vec{c}$ for the common variables.
Output: $b\in\{0,1\}$ such that φ_b is unsatisfiable on the partial assignment $\vec{C}=\vec{c}$
Parameters: Automating algorithm Alg for Q, size s^*, total number of attributes n
begin
  if $Alg(\varphi_0|_{\vec{C}=\vec{c}},\bot)$ *run for* $t(s^*,n)$ *steps returns a valid refutation* π **then**
    return 0
  else return 1
end

Algorithm 9: Interpolation from weak automation

We suppose that $\varphi_0(\vec{A},\vec{C})\wedge\varphi_1(\vec{B},\vec{C})$ has a refutation π in P of size s^*. First observe that since we can compute $\varphi_0|_{\vec{C}=\vec{c}}$ in time $O(s^*+n)$, the algorithm runs in time $p(s^*,n)$.

Next, observe that whenever the algorithm outputs 0, there is indeed a refutation of $\varphi_0|_{\vec{C}=\vec{c}}$, so $\varphi_0(\vec{A},\vec{c})$ is indeed unsatisfiable and the algorithm is correct in this case.

Now suppose that for some $\vec{c}$, $\varphi_1(\vec{B},\vec{c})$ is actually satisfiable, by $\vec{B}=\vec{b}$. Then since $\varphi_0(\vec{A},\vec{C})\wedge\varphi_1(\vec{B},\vec{C})$ is unsatisfiable, it must be that $\varphi_0(\vec{A},\vec{c})$ is actually unsatisfiable. Moreover, since P is closed under restrictions, $\pi|_{(\vec{B},\vec{C})=(\vec{b},\vec{c})}$ is a refutation of $\varphi_0(\vec{A},\vec{C})\wedge\varphi_1(\vec{B},\vec{C})|_{(\vec{B},\vec{C})=(\vec{b},\vec{c})}$. But, since $\vec{b},\vec{c}$ is a satisfying assignment for φ_1,

$$\varphi_0(\vec{A},\vec{C})\wedge\varphi_1(\vec{B},\vec{C})|_{(\vec{B},\vec{C})=(\vec{b},\vec{c})}=\varphi_0(\vec{A},\vec{C})|_{(\vec{B},\vec{C})=(\vec{b},\vec{c})}$$

and moreover, since φ_0 does not use any attributes of $\vec{B}$, $\varphi_0(\vec{A},\vec{C})|_{(\vec{B},\vec{C})=(\vec{b},\vec{c})} = \varphi_0(\vec{A},\vec{C})|_{\vec{C}=\vec{c}}$. Thus, $\pi|_{(\vec{B},\vec{C})=(\vec{b},\vec{c})}$ is in particular a refutation of $\varphi_0|_{\vec{C}=\vec{c}}$, and since restriction does not increase the size of the proof, it has size at most s^*. Thus, in this case, on input $\varphi_0|_{\vec{C}=\vec{c}}$ Alg returns a refutation in Q in $t(s^*,n)$ steps. Observe, therefore, in this case the algorithm correctly returns 0 and not 1. Hence, contrapositively, if the algorithm outputs 1 it must be that $\varphi_1(\vec{B},\vec{c})$ was unsatisfiable. We thus see that Algorithm 9 is indeed an interpolant. □

Summarizing the above discussion, we now obtain that commitment schemes prevent automation of a proof system P if the perfect binding property can be proved in P by a short proof:

Theorem 5.45 *For a commitment scheme with perfect binding and computational hiding, let $\varphi_0(\vec{S},\vec{C})$ represent the property "$\vec{C}$ is a commitment to 0 with secret key $\vec{S}$" and let $\varphi_1(\vec{S},\vec{C})$ represent the property "$\vec{C}$ is a commitment to 1 with secret key $\vec{S}$." If the proof system P has a refutation of $\varphi_0(\vec{S},\vec{C}) \wedge \varphi_1(\vec{S}',\vec{C})$ of size polynomial in n, then P is not weakly automatable.*

Proof Suppose for contradiction that for a weakly automatable P, there were a refutation of $\varphi_0(\vec{S},\vec{C}) \wedge \varphi_1(\vec{S}',\vec{C})$ of size $s(n)$ for some polynomial s. Then by Theorem 5.44, there is an interpolant I running in time polynomial in n such that if $I(\vec{c}) = 0$, φ_0 is false, and if $I(\vec{c}) = 1$, φ_1 is false. Consider the algorithm Alg that runs the interpolant I on the input and if it outputs b, outputs $\neg b$ instead. Alg thus now runs in polynomial time and outputs 1 precisely when $\vec{c}$ is a commitment to 1 and 0 when $\vec{c}$ is a commitment to 0. Hence, the commitment scheme does not satisfy computational hiding. □

We briefly note that many simple systems, such as resolution, actually do have feasible interpolation. Hence, such systems cannot prove the perfect binding property of any commitment scheme, and Theorem 5.45 actually cannot be used to rule out weak automation of these systems. On the other hand, this also means that we can conclude that these statements require long proofs in systems with feasible interpolation. Thus, feasible interpolation plays two roles: for weak systems where feasible interpolation is possible, it is a means of showing statements require long proofs; and for strong systems, the lack of feasible interpolation is a means to show that the system cannot be weakly automated by *any* other system. The strong systems for which Theorem 5.45 may be used to rule out weak automation are generally fragments of the following proof system:

Definition 5.46 *Frege* or *sequent calculus* is the following proof system. Its lines are given by *sequents* which have the form

$$\Gamma \to \Delta$$

where Γ and Δ are finite (possibly empty) sequences of De Morgan formulas. It has the following rules of inference:

- *Axioms*: $\bot \to$, $\to \top$, and $X \to X$ where X is a propositional attribute.

- *Weakening left*: From $\Gamma \to \Delta$ infer $\varphi, \Gamma \to \Delta$ for any De Morgan formula φ.
- *Weakening right*: From $\Gamma \to \Delta$ infer $\Gamma \to \Delta, \varphi$ for any De Morgan formula φ.
- *Exchange left*: For any sequences Γ_1 and Γ_2 and De Morgan formulas φ and ψ, from $\Gamma_1, \varphi, \psi, \Gamma_2 \to \Delta$ infer $\Gamma_1, \psi, \varphi, \Gamma_2 \to \Delta$.
- *Exchange right*: For any sequences Δ_1 and Δ_2 and De Morgan formulas φ and ψ, from $\Gamma \to \Delta_1, \varphi, \psi, \Delta_2$ infer $\Gamma \to \Delta_1, \psi, \varphi, \Delta_2$.
- $\neg$ *introduction left*: For any De Morgan formula φ, from $\Gamma \to \Delta, \varphi$ infer $\neg\varphi, \Gamma \to \Delta$
- $\neg$ *introduction right*: For any De Morgan formula φ, from $\varphi, \Gamma \to \Delta$ infer $\Gamma \to \Delta, \neg\varphi$.
- $\wedge$ *weakening*: For any De Morgan formulas φ and ψ, from $\varphi, \Gamma \to \Delta$ infer $\varphi \wedge \psi, \Gamma \to \Delta$ or $\psi \wedge \varphi, \Gamma \to \Delta$.
- $\wedge$ *combination*: For any De Morgan formulas φ and ψ, from $\Gamma \to \Delta, \varphi$ and $\Gamma \to \Delta, \psi$, infer $\Gamma \to \Delta, \varphi \wedge \psi$.
- $\vee$ *weakening*: For any De Morgan formulas φ and ψ, from $\Gamma \to \Delta, \varphi$ infer $\Gamma \to \Delta, \varphi \vee \psi$ or $\Gamma \to \Delta, \psi \vee \varphi$.
- $\vee$ *combination*: For any De Morgan formulas φ and ψ, from $\varphi, \Gamma \to \Delta$ and $\psi, \Gamma \to \Delta$ infer $\varphi \vee \psi, \Gamma \to \Delta$.
- *Cut*: For any De Morgan formula φ, from $\varphi, \Gamma \to \Delta$ and $\Gamma \to \Delta, \varphi$ infer $\Gamma \to \Delta$.

This sequent calculus has many equivalent formulations (w.r.t. p-simulation), but the version presented above is standard in the literature. Indeed, it has been used as the definition of propositional logic in certain classical textbooks.

Moreover, it is natural to define fragments of this system by restricting the form of the De Morgan formulas to those representable by "constant-depth" circuits when we allow unbounded fan-in to the connectives besides negation. For example, conjunctions and disjunctions may be represented, respectively, by a single AND or OR gate, and we consider these to be *depth-1* formulas—we do not count NOT gates toward the depth. DNF and CNF formulas may be represented, respectively, as a layer of AND gates feeding to a single OR gate, and a layer of OR gates feeding to a single AND gate; we consider these to be *depth-2* formulas. Since DNFs and CNFs are universal classes, they are equivalent in absolute expressive power to general De Morgan formulas, but we know they may need to be exponentially larger. In general,

Definition 5.47 AC_0 is the class of families of Boolean circuits $\{C_n\}_{n=1}^{\infty}$ with unbounded fan-in AND and OR gates and unary NOT gates, such that there is a constant $d > 0$ for which at most d AND or OR gates occur on any path from an output to an input in any circuit C_n in the family. We say that these circuits have *depth* d. *AC_0-Frege* of depth d is the fragment of sequent calculus in which the De Morgan formulas occurring in the lines of a proof may be written as depth-d AC_0 circuits.

Frege systems are expressive enough to efficiently represent the rules of basic arithmetic on binary encodings of integers with the usual operations of addition, subtraction, multiplication, division, and taking remainders. The bounded-depth fragment is strong enough to capture small expressions with limited fan-in addition or multi-

plication of small numbers (with a polylogarithmic number of bits), or equivalently, using "subexponential" size proofs (of size 2^{n^ϵ} for some $\epsilon > 0$). These are not difficult to establish, but the development is lengthy and we do not repeat it here. As a consequence, these systems are able to prove the binding property for some simple commitment schemes. For example, the *Diffie-Hellman key exchange protocol* may be used as such a commitment scheme:

Definition 5.48 The *Diffie-Hellman key exchange protocol* involves two parties that we call Alice and Bob, and proceeds as follows: Alice chooses a modulus m and a *generator* g such that for all integers i that share no nontrivial factors with m, $i = g^k \bmod m$ for some integer k. Let $\phi(m)$ be the number of such integers i between 1 and $m-1$. Alice then chooses an integer a between 1 and $\phi(m)$ uniformly at random, and sends m, g, and $g^a \bmod m$ to Bob. Bob then chooses b between 1 and $\phi(m)$ uniformly at random, and sends $g^b \bmod m$ to Alice. Alice and Bob now agree on $g^{ab} \bmod m$: Alice obtains $g^{ab} = (g^b)^a \bmod m$ and Bob obtains $g^{ab} = (g^a)^b \bmod m$.

We can use this as a commitment scheme, in which the commitment is $(m, g, g^a \bmod m, g^b \bmod m, i)$, where the opening also receives a and b, and checks that the ith bit of $g^{ab} \bmod m$ is the claimed bit. Sequent calculus can compute $g^A \bmod m$ from A and g, $g^B \bmod m$ from B and g, and $g^{AB} \bmod m$ from these values, and thus refute the possibility that an unknown A and B could be such that the relevant bit of $g^{AB} \bmod m$ takes two distinct values. The hiding property is a consequence of the *Diffie-Hellman assumption*:

Definition 5.49 The *Decisional Diffie-Hellman assumption* is that for any polynomial-time algorithm Alg and polynomial bound $p(n)$, for all sufficiently large n-bit m and generators g,

$$\left| \Pr_{a,b}[Alg(m, g, g^a \bmod m, g^b \bmod m, g^{ab} \bmod m) = 1] - \Pr_{a,b,u}[Alg(m, g, g^a \bmod m, g^b \bmod m, u) = 1] \right| < \frac{1}{p(n)}$$

where a and b are chosen uniformly at random from $1, \ldots, \phi(m)$ and u is chosen uniformly at random from the integers sharing no nontrivial factors with m. The *subexponential-time Diffie-Hellman assumption* is the same, except that we assume there exists some $\epsilon > 0$ such that Alg may run in time 2^{n^ϵ} (rather than merely polynomial time).

Thus, from Theorem 5.45 we obtain

Theorem 5.50 *Given the Decisional Diffie-Hellman assumption, sequent calculus is not weakly automatable.*

In fact, the same conclusion holds for a fragment of sequent calculus that is constant depth, but includes connectives for modular addition and linear inequality constraints, known as TC_0-Frege; see the chapter notes for references.

AC_0-Frege can establish the binding property of a Diffie-Hellman commitment, but only using proofs of size 2^{n^ϵ} for some $\epsilon > 0$. Nevertheless, under the stronger, subexponential-time Diffie-Hellman assumption, it is also not automatable:

Theorem 5.51 *Given the subexponential-time Diffie-Hellman assumption, AC_0-Frege is not weakly automatable.*

Theorems 5.50 and 5.51 establish the known hard limits to the power of automated reasoning. We cannot hope to reliably produce the kind of reasoning captured by these basic textbook systems. Thus, our guarantees will necessarily be limited to less expressive systems such as resolution or its fragments.

Chapter notes

As we noted previously, the modern view of logic as the study of sound reasoning in formal languages originates with De Morgan (1847). Again, both Cook (1971) and Levin (1973) noted that propositional reasoning is NP-complete in general, and this was one of the main motivations for the development of the theory of NP completeness.

The first example of a Frege proof system is due to Frege (1879), and Reckhow (1976) shows that many such proof systems (concretely, such as Definition 5.46 from Maciel and Pitassi (1998), based on a system by Gentzen (1936)) are essentially equivalent in expressive power. In the mathematical literature, these systems are sometimes called "Hilbert-style". The forward chaining rule (Modus Ponens) is ancient, prominently appearing in Aristotle's first work on logic. However, the focus on a proof system consisting only of clauses using this rule was motivated by the need for practical efficiency in "logic programming" (1974). It seems that resolution was first proposed by Charles Dodgson (a.k.a. Lewis Carroll) (1897), rediscovered by Blake (1937), and subsequently by many other authors.

The naïve algorithm for satisfiability was essentially introduced by Samson and Mueller (1955), and the DPLL algorithm, that introduced the pure variable and unit propagation rules, was developed in a series of papers by Davis and Putnam (1960) and Davis, Logemann, and Loveland (1962). The connection to treelike resolution is folklore; lower bounds on the running time of such algorithms on some satisfiable instances have also been established by Alekhnovich, Hirsch, and Itskyson (2005). Dowling and Gallier (1984) were the first to observe that inference with *Horn rules* (implications with only positive literals; equiv., clauses with all but one literal negated) could be performed in linear time in the size of the formulas. Dalal and Etherington (1992) were the first to observe that this could be extended to unit propagation. Exercise 5.8 is adapted from Brachman and Levesque (2004). The first clause-learning method was proposed by Stallman and Sussman (1977). The method of clause learning from analysis of unit propagation conflicts was introduced by Marques-Silva and Sakallah (1999) and independently by Bayardo and Schrag (1997). Random restarts were proposed by Gomes, Selman, and Crato (1997). These ideas, together with new heuristics for selecting assignments and faster data structures, were engineered into

the first practical SAT solver, Chaff, by Zhang et al. (2001). The observation that all of these clause learning methods still resulted in resolution proofs is due to Beame, Kautz, and Sabharwal (2004). We remark that Beame, Kautz, and Sabharwal also showed that unless a clause-learning method captures general resolution (not possible for tractable methods unless P = NP by Theorem 5.26) then the resulting fragments of resolution obtained by clause learning are *not* closed under restrictions.

The first nontrivial resolution lower bound is due to Tseitin (1968), under the syntactic restriction that on any path in the proof, variables are eliminated at most once (as we observe holds without loss of generality for treelike resolution). Tseitin's lower bound was for formulas essentially expressing the fact that graphs have an even number of odd degree nodes (indeed: summing the degrees counts each edge twice, so the number must be even). The first superpolynomial lower bound for general resolution was for a different formalization of the pigeonhole principle, by Haken (1985). The binary variant we consider here was introduced by Filmus et al. (2015). Theorem 5.20 was established by Ben-Sasson and Wigderson (2001); Exercise 5.10 is also from that work. The framework of prover-liar games was introduced by Pudlák and Buss (1995) and Theorem 5.22 is from Pudlák (2000).

Questions about efficient automation of proof systems were first formalized by Bonet, Pitassi, and Raz (2000), though they credit the notion to Impagliazzo. Theorem 5.26 was first established by Atserias and Müller (2020). The proof was later simplified by de Rezende et al. (2021) (who proved that analogously, an algebraic logic known as "polynomial calculus" is NP-hard to automate), and we follow a presentation of the proof by Robere that incorporates these improvements.

Although McCarthy (1959) originally suggested that planning could be cast as a reasoning problem, the first thorough treatment of planning using resolution was by Green (1969). The STRIPS model was due to Fikes and Nilsson (1971). The more modern approach using propositional SAT-solvers was suggested by Kautz and Selman (1992; 1996).

The clause space complexity of proofs was first defined by Esteban and Torán (2001). The rank/Strahler number, in the context of trees, was first considered by Horton (1945) and Strahler (1952) in geology. Kullmann (1999) was the first to prove Theorem 5.39, but an essentially similar algorithm for properly learning low-rank decision trees (see Exercise 4.3) was proposed by Ehrenfeucht and Haussler (1989): the difference is simply whether the recursive search terminates when an input clause is found or when only examples with a common label remain consistent with the partial assignment. The connection of the rank parameter to clause space (Exercise 5.5) was made explicit by Ansótegui et al. (2008), and Exercise 5.7 is also adapted from that work. Beame and Pitassi (1996) were the first to show that treelike resolution can be automated in quasipolynomial time, but their analysis was based on the width rather than the clause space of the proof. Exercise 5.9 is based on work by Galil (1977).

The notion of weak automation was introduced by Atserias and Bonet (2004), but the hardness of automating reasoning based on cryptographic constructions was studied earlier. The first was for Frege systems with constant-depth threshold formulas ("TC_0-Frege", cf. Theorem 5.50) by Bonet, Pitassi, and Raz (2000) based on Diffie-

Hellman key exchange (Diffie and Hellman, 1976). The extension to simply constant depth formulas ("AC_0-Frege", Theorem 5.51) is due to Bonet et al. (2004). The existence of interpolants was first proved by Craig (1957); the feasible version was introduced by Krajíček (1994). The first work connecting cryptographic hardness to automated reasoning, for a strong system by Krajíček and Pudlák (1995), actually considered feasible interpolation, not automation. Bonet, Pitassi, and Raz (2000) connected interpolation with automation (Theorem 5.44). They credit M. Naor with the observation that hardness may be based on bit commitment (Theorem 5.45). Cryptographic bit commitment was first considered by M. Blum (1982).

As with the previous chapters, this chapter only provides an introduction to the essentials of automated reasoning with a focus on polynomial-time algorithms. We recommend a reader interested in propositional reasoning to the *Handbook of Satisfiability* (Biere et al., 2021). For more on the quantitative aspects of propositional proof systems, especially lower bounds, see the book by Krajíček (2019).

Exercises

5.1 *Chaining without negation is complete for positive facts.* Suppose that we have a knowledge base KB for chaining in which the rules only contain positive literals, i.e., are of the form $[X_{i_1} \wedge \cdots \wedge X_{i_k}] \supset X_j$ for some $i_1, \ldots, i_k$ and j. Prove that for any attribute X_{i^*}, if $KB \models X_{i^*}$, then there is a chaining proof of X_{i^*} from KB.

5.2 *Writing a resolution proof.* Consider the following knowledge base over Boolean attributes

Q = "the request is quarantined", R = "the request is marked high-risk",
S = "the request is sent to security review", G = "the system grants access", and
N = "the request is from a known network".

The knowledge base contains the following facts and rules:

$$G, \qquad N, \qquad Q \supset (R \vee S), \qquad R \supset \neg G, \qquad S \supset \neg N.$$

(a) Rewrite the implication rules as clauses.
(b) Give a resolution refutation proving that the knowledge base entails $\neg Q$. That is, add the clause Q and derive the empty clause $\bot$.
(c) Explain why the proof cannot be reproduced by a simple chaining proof system that only derives heads of rules from their bodies.

5.3 *Weakening does not add power to resolution.* We defined resolution as a proof system with a single rule of inference, the *resolution* or *cut* rule. Sometimes, the system is considered to have a second rule of inference known as *weakening*: given any clause $C = \ell_1 \vee \cdots \vee \ell_k$, the weakening rule allows us to infer the clause $C' = \ell_1 \vee \cdots \vee \ell_k \vee \ell_{k+1} \vee \cdots \vee \ell_r$ for any set of literals $\ell_{k+1}, \ldots, \ell_r$ we wish (Note that C' is a "weaker" clause than C from the standpoint of seeking an unsatisfiable formula, since any assignment that satisfies C also satisfies C'.) In this exercise, we show that the weakening rule does not help us find refutations.

(a) Prove that if there is a resolution refutation of a CNF formula φ of length L when the weakening rule is allowed, there is also a resolution refutation of φ of length L that only uses the resolution rule, not weakening.
(b) Prove that if there is a clause space s treelike resolution refutation of a CNF φ that uses the weakening rule, then there is also a clause space s treelike resolution refutation of φ that does not use weakening.
(c) Prove that if there is a width w resolution refutation of a CNF φ using the weakening rule, then there is also a width w refutation of φ that does not use weakening.

5.4 *Clausal completeness of resolution with weakening.* Exercise 5.3 shows that weakening does not change the power of resolution when used for refutations. In this exercise, we will show that this is not true when we use resolution to prove clauses.
(a) Give an example of a set of clauses $KB = \{C_1, \ldots, C_m\}$ and clause C^* such that $KB \models C^*$ but there is no resolution proof of C^* from KB without weakening.
(b) Prove that for every set of clauses $KB = \{C_1, \ldots, C_m\}$ and clause C^* such that $KB \models C^*$, there is a proof of C^* from KB using resolution with weakening.
(c) A *prime implicate* of a set of clauses $KB = \{C_1, \ldots, C_m\}$ is any clause C' such that $KB \models C'$, but for any proper subclause C'' of C', $KB \not\models C''$. Prove that for any prime implicate of KB that is not a tautology, C', there is a resolution proof of C' from KB that does not use weakening.

5.5 *Treelike clause space s resolution proofs are those with a proof graph that is a tree of rank s-1.* The clause space of a treelike resolution refutation can be characterized cleanly in terms of the *rank* of the tree underlying the proof.
(a) Prove that the graph of a treelike resolution refutation using clause space s is a tree of rank $s - 1$.
(b) Prove that if a treelike resolution refutation has a graph that is of rank k, then the treelike refutation can be carried out (in an appropriate order) using clause space $k + 1$.

5.6 *Closure under restrictions.* In this exercise, we'll establish the key property that the families of resolution proofs we consider are closed under restrictions.
(a) Prove that if there is a clause space s treelike resolution refutation of a CNF φ, then for any partial assignment ρ, there is also a clause space s treelike resolution refutation of $\varphi|_\rho$ (that does not use the weakening rule). You may use the result of Exercise 5.3(b).
(b) Prove that if there is a width-w refutation of a CNF φ, then for any partial assignment ρ, there is also a width-w refutation of $\varphi|_\rho$ (that does not use the weakening rule). You may use the result of Exercise 5.3(c).

5.7 *Relating chaining, unit propagation, and clause-space-2 resolution.*
(a) *Unit propagation simulates chaining.* Prove that if there is a (forward) chaining derivation of a literal ℓ from KB (and not of $\neg\ell$), then repeated application of unit propagation starting from the empty partial assignment $(*, \ldots, *)$ applied to KB eventually obtains a partial assignment that satisfies ℓ.
(b) *Treelike clause space* 2 *resolution simulates unit propagation (and hence also chaining).* A *unit propagation refutation* occurs if unit propagation obtains a partial

assignment ρ such that for some C_i in KB, $C_i|_\rho$ is an empty clause. Prove that if there is a unit propagation refutation of KB from the empty partial assignment, then there is a treelike resolution refutation of KB using clause space 2.
(c) *Unit propagation simulates treelike clause-space-2 resolution.* Prove that if there is a treelike resolution refutation of KB using clause space 2, then unit propagation starting from the empty partial assignment will lead to a refutation.

5.8 *Linear time algorithm for unit propagation.* In this exercise, you will design data structures to enable a linear-time implementation of unit propagation (Algorithm 7): given as input a CNF given as a list of clauses $C_1, \ldots, C_m$ each represented by a list of literals, return the partial assignment ρ obtained by unit propagation or FAIL if there is a unit propagation refutation. In particular, you should design data structures for the clauses that allow you to detect whether a new assignment reduces it to a single literal (or empty clause) in constant time per assignment, and data structures for the literals that allow you to process the result of setting a literal ℓ to true in constant time per occurrence of ℓ in the input. (You'll need to initially process the input to initialize these data structures prior to the main loop of Algorithm 7.) Prove that your implementation is correct and runs in linear time.

5.9 *Efficient algorithms for bounded width resolution (with applications).* In Theorem 5.39 we saw an efficient algorithm for searching for space-bounded treelike resolution refutations. This is not the only family of proofs for which proof search is efficient. In this exercise, you will show that for fixed w, there is an efficient algorithm for searching for width-w refutations. Crucially, we *do not* assume that these proofs are treelike.
(a) More precisely, prove that there is an algorithm that, on input a CNF φ, runs in time $n^{O(w)}$, and decides whether or not there is a width-w refutation of φ. ***(Hint:** use dynamic programming)*
(b) Prove that such an algorithm can be used to solve 2SAT (i.e., decides whether or not an input 2CNF has a satisfying assignment) in polynomial time. ***(Hint**: You will want to use Corollary 5.13.)*

5.10 *Exponential lower bound for treelike resolution.* In Theorem 5.20, we saw that lower bounds on the width required to refute a formula may be used to give a subexponential lower bound on the size of a resolution refutation. Here, we will find that the width gives a truly exponential-size bound on the size of a treelike refutation: Indeed, prove that if the CNF formula φ requires width $w(\varphi \vdash \bot)$ to refute and has width $w(\varphi)$, that any treelike refutation of φ must have size at least $2^{w(\varphi \vdash \bot) - w(\varphi)}$. (In particular, we conclude DPLL takes $\Omega(2^{w(\varphi \vdash \bot) - w(\varphi)})$ time to determine φ is unsatisfiable.)

5.11 *Feasible interpolation for resolution.* We saw in Theorem 5.44 that (weakly) automatable proof systems have feasible interpolation (Definition 5.43). Although Theorem 5.26 suggests that resolution is not (strongly) automatable, and it is not known whether or not it is weakly automatable, it nevertheless has feasible interpolation: Prove that resolution has feasible interpolation.