Logic and Proof Hilary 2026

Resolution and Interpolation

James Worrell

In this lecture we introduce the resolution proof calculus for propositional logic and we show how to efficiently extract interpolants from resolution proofs. Interpolants have many uses in program verification and automatic theorem proving. In the next lecture we show how interpolation can be used to give exponential lower bounds on the length of resolution proofs.

1 Resolution

It is convenient to regard a CNF formula as a set of clauses and a clause as a set of literals. Since the elements of a set do not have order or multiplicity, this representation accounts for the associativity, commutativity, and idempotence of conjunction and disjunction. For example, the following three CNF formulas are all represented by the set $\{\{p_3\},\{p_1,\neg p_2\}\}$:

$$(p_3 \wedge (p_1 \vee p_1 \vee \neg p_2) \wedge p_3), \quad ((\neg p_2 \vee p_1 \vee \neg p_2) \wedge (p_3 \vee p_3)), \quad (p_3 \wedge (\neg p_2 \vee p_1)).$$

In the set representation of CNF formulas, the empty clause, denoted $\Box$, is equivalent to **false** since it is the unit of the disjunction operation. Therefore, if a CNF formula contains the empty clause, then it is equivalent to **false**. By contrast if a CNF formula *is* the empty set of clauses then it is equivalent to **true** (the unit of the conjunction operation).

Let p be a propositional variable. Recall that the complement of p is $\neg p$, and the complement of $\neg p$ is p. Let C_1 and C_2 be clauses. A clause R is called a *resolvent* of C_1 and C_2 if there are complementary literals $L \in C_1$ and $\overline{L} \in C_2$ such that $R = (C_1 \setminus \{L\}) \cup (C_2 \setminus \{\overline{L}\})$.

A *derivation* (or proof) of a clause C from a set of clauses F is a sequence $C_1, C_2, \ldots, C_m$ of clauses, where $C_m = C$ and for each $i = 1, 2, \ldots, m$ either $C_i \in F$ or C_i is a resolvent of C_j, C_k for some $j, k < i$. A derivation of the empty clause $\Box$ from a formula F is called a *refutation* of F.

Example 1. A refutation of the CNF formula $\{\{p,\neg q\},\{q,r\},\{\neg p,\neg q,r\},\{\neg r\}\}$ is as follows:

1.	$\{p,\neg q\}$	Assumption	5.	$\{\neg p, r\}$	2,4 Resolution
2.	$\{q,r\}$	Assumption	6.	$\{\neg r\}$	Assumption
3.	$\{p,r\}$	1,2 Resolution	7.	$\{r\}$	3,5 Resolution
4.	$\{\neg p,\neg q,r\}$	Assumption	8.	$\Box$	6,7 Resolution

The same derivation can also be represented by the following *proof tree*:

$$\dfrac{\dfrac{\dfrac{\{p,\neg q\} \quad \{q,r\}}{\{p,r\}} \quad \dfrac{\{q,r\} \quad \{\neg p,\neg q,r\}}{\{\neg p,r\}}}{\{r\}} \quad \{\neg r\}}{\Box}$$

2 Soundness and Completeness

The proof of the following is left as an exercise.

Lemma 2. Let F be a CNF formula represented as a set of clauses. Suppose R is a resolvent of two clauses C_1 and C_2 of F then $F \equiv F \cup \{R\}$.

Soundness says that we only derive a contradiction from an unsatisfiable set of clauses.

Theorem 3 (Soundness)**.** If there is a derivation of $\Box$ from F then F is unsatisfiable.

Proof. Suppose that $C_1, C_2, \ldots, C_m = \Box$ is a proof of $\Box$ from F. By repeated applications of Lemma 2 we have that F is equivalent to $F \cup \{C_1, C_2, \ldots, C_m\}$. But the latter set of clauses includes the empty clause and thus is unsatisfiable. $\Box$

Completeness is the converse of soundness: it says that if a CNF formula F is unsatisfiable then we can derive the empty clause from F by resolution. Before giving the proof, we specialise the notion of substitution of propositional formulas, introduced in an earlier lecture.

Given a set of clauses F involving a propositional variable p, let $F[\mathbf{false}/p]$ (read "F with **false** for p") denote that clause set obtained by replacing p everywhere in F with **false**, replacing $\neg p$ with **true**, and then simplifying by applying the unit laws. That is, $F[\mathbf{false}/p]$ arises by deleting p from all clauses in F that contain p, and removing from F all clauses that contain $\neg p$ (since they become true). Likewise we define $F[\mathbf{true}/p]$ by deleting all clauses containing p and removing $\neg p$ from all clauses containing $\neg p$.

Example 4. If $F = \{\{p_1, p_3\}, \{\neg p_1, p_2\}, \{\neg p_2, p_3\}, \{\neg p_3\}\}$, then $F[\mathbf{false}/p_3] = \{\{p_1\}, \{\neg p_1, p_2\}, \{\neg p_2\}\}$ and $F[\mathbf{true}/p_3] = \{\{\neg p_1, p_2\}, \Box\}$.

Theorem 5 (Completeness)**.** If F is unsatisfiable, then there is a derivation of $\Box$ from F.

Proof. The proof is by induction on the number n of different propositional variables in F.

Base case. The base case is that $n = 0$, i.e., F does not mention any propositional variables. Then F either contains no clauses or it contains only the empty clause. In the former case F is equivalent to **true**. But F is assumed to be unsatisfiable, so it must be that $F = \{\Box\}$ and there is a one-line resolution refutation of F.

Induction step. Suppose that F is an unsatisfiable set of clauses that mentions variables $p_1, p_2, \ldots, p_n$ and let $F_0 := F[\mathbf{false}/p_n]$ and $F_1 := F[\mathbf{true}/p_n]$.

Since F is unsatisfiable, F_0 is also unsatisfiable. Moreover F_0 mentions one fewer variable than F so by the induction hypothesis there is a resolution proof $C_0, C_1, \ldots, C_m = \Box$ that derives the empty clause from F_0. For each clause C_i in the above proof that comes from F_0, either C_i is already in F or $C_i \cup \{p_n\}$ is in F. By replacing the deleted copies of p_n from the latter type of clauses and propagating p_n through the proof we obtain a new proof $C'_0, C'_1, \ldots, C'_m$ from F, where either $C'_m = \Box$ or $C'_m = \{p_n\}$ (see Example 6). In the first case there is nothing more left to prove, so we consider the second case: $C'_m = \{p_n\}$. Applying similar reasoning to F_1 we can assume that there is a proof of $\{\neg p_n\}$ from F. Gluing together these two proofs and applying one more resolution step to $\{p_n\}$ and $\{\neg p_n\}$ gives a proof of $\Box$ from F. $\Box$

Example 6. We illustrate the transformation on proofs underlying the induction step in the proof of the Completeness Theorem. Consider the formula $F = \{\{p, r\}, \{\neg p, q\}, \{\neg q, r\}\}$. We transform the following derivation of $\Box$ from $F[\mathbf{false}/r]$

$$\dfrac{\dfrac{\{p\} \qquad \{\neg p, q\}}{\{q\}} \qquad \{\neg q\}}{\Box}$$

to the following derivation of $\{r\}$ from F:

$$\dfrac{\dfrac{\{p,r\} \qquad \{\neg p,q\}}{\{q,r\}} \qquad \{\neg q,r\}}{\{r\}}$$

3 Interpolation

Let $F \wedge G$ be an unsatisfiable CNF formula and denote by $\boldsymbol{z}$ the set of propositional variables that occur in both F and G. An *interpolant* of F and G is a formula I over $\boldsymbol{z}$ such that $F \to I$ and $I \to \neg G$ are both valid. In other words, for each valuation $\boldsymbol{\alpha}$ of $\boldsymbol{z}$, if $I[\boldsymbol{\alpha}/\boldsymbol{z}] = 0$ then $F[\boldsymbol{\alpha}/\boldsymbol{z}]$ is unsatisfiable and if $I[\boldsymbol{\alpha}/\boldsymbol{z}] = 1$ then $G[\boldsymbol{\alpha}/\boldsymbol{z}]$ is unsatisfiable. In this section we show how to efficiently construct an interpolant of F and G from a resolution refutation of $F \wedge G$. As we shortly explain, we represent this interpolant as a straight-line program (or circuit) rather than a formula.

A *straight-line program (or boolean circuit)* over a set of propositional variables $\{z_1, z_2, \ldots, z_n\}$ consists of a finite sequence of definitions $e_1 := E_1, \ldots, e_m := E_m$, where each expression E_i has the form either $\mathbf{false}, \mathbf{true}, A \wedge B, A \vee B$, or $\neg A$, where A and B are drawn from the set of *input variables* $z_1, \ldots, z_n$ and *program variables* e_j for $j < i$. A valuation $\boldsymbol{\alpha} \in \{0,1\}^n$ of the input variables is extended sequentially to the program variables in the obvious way. Such a program thus denotes a function $f : \{0,1\}^n \to \{0,1\}$ such that $f(\boldsymbol{\alpha})$ is the value of e_m when the input is $\boldsymbol{\alpha}$. We can think of formulas as a special case of straight-line programs in which each program variable appears at most once on the right-hand side of an assignment. Every straight-line program can be unfolded to a formula by duplicating variables, although this may cause an exponential blow-up in the number of variables. We say that a straight-line program is *monotone* if it does not use negation.

Example 7. The following is a monotone straight-line program for computing whether a strict majority of its inputs $z_1, \ldots, z_3$ is equal to 1.

$$\begin{array}{lcllcl} e_1 & := & z_1 \wedge z_2 & e_4 & := & e_1 \vee e_2 \\ e_2 & := & z_1 \wedge z_3 & e_5 & := & e_4 \vee e_3 \\ e_3 & := & z_2 \wedge z_3 & & & \end{array}$$

Proposition 8. There is a polynomial-time algorithm that, given as input a resolution refutation of a CNF formula $F \wedge G$ and a Boolean valuation $\boldsymbol{\alpha}$ of the shared variables $\boldsymbol{z}$ of F and G, outputs a bit $b \in \{0,1\}$ such that: if $b = 0$, then $F[\boldsymbol{\alpha}/\boldsymbol{z}]$ is unsatisfiable; if $b = 1$, then $G[\boldsymbol{\alpha}/\boldsymbol{z}]$ is unsatisfiable.

Proof. Let $C_1, \ldots, C_m$ be the given resolution refutation of $F \wedge G$. We compute in polynomial time a sequence $\widetilde{C}_1, \ldots, \widetilde{C}_m$, where either $\widetilde{C}_i \subseteq C_i$ is a clause or $\widetilde{C}_i = \mathbf{true}$, and a function origin : $\{1, \ldots, m\} \to \{F, G\}$ with the following two properties for all $i \in \{1, \ldots, m\}$:

(i) $\widetilde{C}_i \models C_i[\boldsymbol{\alpha}/\boldsymbol{z}]$;

(ii) if origin$(i) = F$, then $F[\boldsymbol{\alpha}/\boldsymbol{z}] \models \widetilde{C}_i$; otherwise, $G[\boldsymbol{\alpha}/\boldsymbol{z}] \models \widetilde{C}_i$.

For $i = 1, \ldots, m$, we iteratively define the clause $\widetilde{C}_i$ and the value of origin$(i) \in \{F, G\}$ to satisfy Items (i) and (ii), above. The definition is by cases, according to how C_i is derived.

1. If C_i is a clause from F, then define $\widetilde{C}_i := C_i[\boldsymbol{\alpha}/\boldsymbol{z}]$ and origin$(i) := F$.
2. If C_i is a clause from G, then define $\widetilde{C}_i := C_i[\boldsymbol{\alpha}/\boldsymbol{z}]$ and origin$(i) := G$.

3. Suppose that C_i follows from C_j, C_k for $j, k < i$ by resolution on $x \in C_j$ and $\neg x \in C_k$, where variable x only occurs in F. There are three sub-cases. If origin$(j) = G$, then define origin$(i) :=$ G and $\widetilde{C}_i := \widetilde{C}_j$; otherwise, if origin$(k) = G$ then define origin$(i) := G$ and $\widetilde{C}_i := \widetilde{C}_k$; otherwise, define origin$(i) = F$ and

$$\widetilde{C}_i := \begin{cases} \mathbf{true} & \text{if } \widetilde{C}_j \equiv \mathbf{true} \text{ or } \widetilde{C}_k \equiv \mathbf{true}, \\ \widetilde{C}_j & \text{otherwise, if } x \notin \widetilde{C}_j, \\ \widetilde{C}_k & \text{otherwise, if } \neg x \notin \widetilde{C}_k, \\ (\widetilde{C}_j \setminus \{x\}) \cup (\widetilde{C}_k \setminus \{\neg x\}) & \text{otherwise.} \end{cases}$$

4. Suppose that C_i follows from C_j, C_k for $j, k < i$ by resolution on $y \in C_j$ and $\neg y \in C_k$, where variable y only occurs in G. There are three sub-cases. If origin$(j) = F$, then define origin$(i) := F$ and $\widetilde{C}_i := \widetilde{C}_j$; otherwise, if origin$(k) = F$ then define origin$(i) := F$ and $\widetilde{C}_i := \widetilde{C}_k$; otherwise, define origin$(i) = G$ and

$$\widetilde{C}_i := \begin{cases} \mathbf{true} & \text{if } \widetilde{C}_j \equiv \mathbf{true} \text{ or } \widetilde{C}_k \equiv \mathbf{true}, \\ \widetilde{C}_j & \text{otherwise, if } y \notin \widetilde{C}_j, \\ \widetilde{C}_k & \text{otherwise, if } \neg y \notin \widetilde{C}_k, \\ (\widetilde{C}_j \setminus \{y\}) \cup (\widetilde{C}_k \setminus \{y\}) & \text{otherwise.} \end{cases}$$

5. Suppose C_i follows from C_j, C_k for j, k by resolution on $z \in C_j$ and $\neg z \in C_k$, where z is a common variable. There are two sub-cases. If z false under $\boldsymbol{\alpha}$, then $\widetilde{C}_i := \widetilde{C}_j$ and origin$(i) :=$ origin(j); otherwise, if z is true under $\boldsymbol{\alpha}$ then define $\widetilde{C}_i := \widetilde{C}_k$ and origin$(i) :=$ origin(k).

To prove Items (i) and (ii), we show by strong induction on $i \in \{1, \ldots, m\}$ that: (IH1) if $\widetilde{C}_i \equiv \mathbf{true}$, then $C_i[\boldsymbol{\alpha}/\boldsymbol{z}] \equiv \mathbf{true}$; (IH2) if $\widetilde{C}_i \not\equiv \mathbf{true}$, then $\widetilde{C}_i \subseteq C_i$; (IH3) the subsequence of clauses $\widetilde{C}_1, \ldots, \widetilde{C}_i$ with origin F is a resolution proof from $F[\boldsymbol{\alpha}/\boldsymbol{z}]$, and the subsequence of clauses with origin G is a resolution proof from $G[\boldsymbol{\alpha}/\boldsymbol{z}]$. We omit the details.

Having defined $\widetilde{C}_1, \ldots, \widetilde{C}_m$, we can conclude the proof. Since $C_m = \square$, by Item (i) we have $\widetilde{C}_m = \square$. Next, by Item (ii), if origin$(m) = F$, then $F[\boldsymbol{\alpha}/\boldsymbol{z}]$ is unsatisfiable otherwise, if origin$(m) = G$, then $G[\boldsymbol{\alpha}/\boldsymbol{z}]$ is unsatisfiable. Thus our algorithm outputs 0 if origin$(m) = F$, and it outputs 1 if origin$(m) = G$. □

In the following result we show how to extract from the proof of Proposition 8 the description of a straight-line program that computes the interpolant of two formulas.

Theorem 9. There is a polynomial-time construction that transforms a resolution refutation of a CNF formula $F \wedge G$ into a straight-line program on the set of common variables $\boldsymbol{z}$ that represents an interpolant of F and G. Moreover, if the variables $\boldsymbol{z}$ appear only positively in F then the above straight-line program is monotone.

Proof. Let $C_1, \ldots, C_m$ be a refutation of $F \wedge G$. We define a straight-line program $e_1 := E_1, \ldots, e_m := E_m$, with one assignment for each clause in the refutation. Referring to the proof of Proposition 8, the program is designed so that when given as input a valuation $\boldsymbol{\alpha} \in \{0,1\}^n$ of the variables $\boldsymbol{z}$ the program variable e_i takes value 0 just in case origin$(i) = F$ and e_i takes value 1 when origin$(i) = G$.

For $i \in \{1, \ldots, m\}$ we define the assignment $e_i := E_i$ according to the following cases:

1. If C_i is a clause of F, then the assignment is $e_i := 0$.

2. If C_i is a clause of G, then the assignment is $e_i := 1$.

3. If C_i is the resolvent of clauses C_j and C_k around a variable x that occurs only in F, then the assignment is $e_i := e_j \vee e_k$.

4. If C_i is the resolvent of clauses C_j and C_k around a variable that occurs only in G, then the assignment is $e_i := e_j \wedge e_k$.

5. If C_i is the resolvent of clauses $C_j = C'_j \cup \{z\}$ and $C_k = C'_k \cup \{\neg z\}$ over a common variable z, then the assignment is $e_i := (e_j \wedge \neg z) \vee (e_k \wedge z)$.

We leave to the reader to verify that the program correctly computes the origin function. The proof of the final clause of the theorem (the monotone case) is the subject of the following exercise. □

Exercise 10. Assume that the shared variables $\boldsymbol{z}$ occur only positively in F.

(a) Suppose that we change Item 5 of the proof of Proposition 8 to the following:

5′ Suppose C_i follows from C_j and C_k, for $j, k < i$, by resolution on $z \in C_j$ and $\neg z \in C_k$. Then

- if z is true under $\boldsymbol{\alpha}$, then define $\widetilde{C}_i := \widetilde{C}_k$ and $\text{origin}(i) := \text{origin}(k)$;
- if z is false under $\boldsymbol{\alpha}$ and $\text{origin}(k) = G$, then define $\widetilde{C}_i := \widetilde{C}_j$ and $\text{origin}(i) := \text{origin}(j)$;
- if z is false under $\boldsymbol{\alpha}$ and $\text{origin}(k) = F$, then define $\widetilde{C}_i := \widetilde{C}_k$ and $\text{origin}(i) := F$;

Adapt the proof of Proposition 8 to show that, if we replace Item 5 with Item 5′, it still holds that $\widetilde{C}_i \models C_i[\boldsymbol{\alpha}/\boldsymbol{z}]$ for all i.

Hint: Strengthen the induction hypothesis in the proof by altering (IH1) to to the following: (IH1′) If $\widetilde{C}_i \equiv \mathbf{true}$, then C_i contains a literal that is made true by $\boldsymbol{\alpha}$ and if, moreover, $\text{owner}(i) = F$, then the aforementioned literal is positive.

(b) Explain why, in the description of the monotone circuit in the proof of Theorem 9, we can change Item 5 to be $e_i := (e_j \vee z) \wedge e_k$.