

Resolution (R) is a propositional proof system for the language of unsatisfiable set of clauses.

Given two clauses $C \cup \{x\}, D \cup \{\bar{x}\}$, the **resolution rule** permits us to conclude $C \cup D$ from them.

Note that the resolution rule is **sound**: if τ satisfies $C \cup \{x\}, D \cup \{\bar{x}\}$, then τ must also satisfy $C \cup D$.

A **resolution refutation** of an unsatisfiable set of clauses \mathcal{S} is a sequence of clauses C_1, C_2, \dots, C_n , where:

1. each C_i is either in \mathcal{S} , or follows by the resolution rule from some $C_j, C_k, j, k < i$,
2. $C_n = \square$, i.e., it is the empty clause.

3

$$\text{PHP}_1^2 = \{\{P_{11}\}, \{P_{21}\}, \{\bar{P}_{11}, \bar{P}_{21}\}\}$$

Here is its resolution refutation:

$$\{P_{11}\}, \{P_{21}\}, \{\bar{P}_{11}, \bar{P}_{21}\}, \{\bar{P}_{21}\}, \{\} = \square$$

4

Resolution

A **literal** is a variable x , or its negation, \bar{x} .

A **clause** is a set of literals, $\{l_1, l_2, \dots, l_k\}$.

A truth assignment τ **satisfies** a clause C , written $\tau \models C$, if it makes at least one literal in C true.

A (finite) set of clauses \mathcal{S} is **satisfiable** if there exists a truth assignment τ that satisfies all the clauses in \mathcal{S} .

e.g. $\mathcal{S} = \{\{x, \bar{y}, \bar{z}\}, \{\bar{x}\}, \{y, z\}\}$ is satisfiable, and the truth assignment $\tau(x) = \tau(y) = 0, \tau(z) = 1$ satisfies it.

e.g. $\mathcal{S} = \{\{x\}, \{\bar{x}, y\}, \{\bar{y}, z\}, \{\bar{z}\}\}$ is unsatisfiable.

1

e.g. (The Pigeonhole Principle) Let $\text{PHP}_{n-1}^n, n > 1$, be the set of clauses given by:

1. $\{P_{i1}, P_{i2}, \dots, P_{i(n-1)}\}, 1 \leq i \leq n$
2. $\{\bar{P}_{ik}, \bar{P}_{jk}\}, 1 \leq i < j \leq n, 1 \leq k \leq (n-1)$

Then PHP_{n-1}^n is a set of $n + \binom{n}{2} \cdot (n-1) = O(n^3)$ clauses, and PHP_{n-1}^n is *unsatisfiable* for every n .

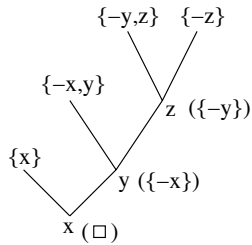
It is unsatisfiable for every n , because if it were satisfiable, a τ that satisfies it would effectively give us a 1-1 relation from $\{1, 2, \dots, n\}$ to $\{1, 2, \dots, n-1\}$, which is not possible.

2

We can transform a DPLL refutation into a **R** refutation. The leaves are already labeled with clauses. If an internal node is labelled with variable x , the idea is to now label it with the clause resulting from resolving on its two parents nodes (to which we already, inductively, assigned clauses) on x .

The resulting refutation is tree-like.

e.g.



7

R is both **sound** and **complete**.

To see that it is sound, note that the resolution rule is sound, so it can be shown inductively that if there is a τ that satisfies \mathcal{S} , then τ must also satisfy every C_i in the refutation, and in particular $C_n = \square$, which is not possible.

So if we can derive \square from \mathcal{S} , we know that \mathcal{S} cannot be satisfiable.

To show completeness, we use the Davis-Putnam algorithm. The first version, known as DPLL, gives us **tree-like** refutations; the second version, known as DP, gives us **dag-like** refutations.

A tree-like refutation is one where each clause in the proof, except for the original clauses, is used at most once as a premiss of a rule.

5

So tree-like completeness follows from the correctness of DPLL.

To show the correctness of DPLL, it is enough to observe that \mathcal{S} is unsatisfiable iff both $\mathcal{S}|_{x=0}$ and $\mathcal{S}|_{x=1}$ are unsatisfiable, for any $x \in \text{Var}(\mathcal{S})$.

$\mathcal{S}|_{x=t}$ denotes \mathcal{S} where each instance of x has been replaced by the truth value $t \in \{0, 1\}$, and the result simplified.

The simplification is obvious; if a literal l is made true by setting $x = t$, any clause containing it is eliminated altogether; if it is made false, the literal is eliminated from all the clauses containing it.

8

A dag-like refutation can be transformed into a tree-like one by re-deriving a clause each time it's used as a premiss; this gives rise, of course, to an exponential blow up.

DPLL is defined as follows: we pick an $x \in \text{var}(\mathcal{S})$, and branch out on $x = 0$ and $x = 1$. On each branch we continue selecting new variables, until one of two things happen:

1. a leaf corresponds to a (partial) truth assignment falsifying some clause, in which case we label it with such a clause, or
2. on the path from that leaf to the root we examined all the variables, in which case we get a (full) truth assignment satisfying the original set of clauses.

If at the end each leaf is labeled with a clause, we know \mathcal{S} is unsatisfiable.

6

Say l_D is some such literal, and in the original refutation we resolve on some C_i ($i > 2$) and some E to get rid of it.

Since l_D is no longer there, we simply discard E , and the subtree rooted at E , and let $C'_i := C'_{i-1}$.

We follow through all the way to \square , making such adjustments.

Claim: For all $2 \leq i \leq k$, $C'_i \subseteq C_i \cup \{l\}$, and for $i > k$, $C'_i \subseteq C_i$.

Corollary: A minimal tree-like **R** refutation is regular.

11

DP algorithm:

Repeat until $\mathcal{S} = \emptyset$ or $\square \in \mathcal{S}$

 Choose any $x \in \text{Var}(\mathcal{S})$

$\mathcal{S} := \mathcal{S} - \{C \in \mathcal{S} : \{x, \bar{x}\} \subseteq C\}$

$T := \{C \cup D : C \cup \{x\}, D \cup \{\bar{x}\} \in \mathcal{S}\}$

$\mathcal{S} := T \cup \{C \in \mathcal{S} : \{x, \bar{x}\} \cap C = \emptyset\}$

If “ \emptyset ” then \mathcal{S} is satisfiable, and if “ \square ” then \mathcal{S} is not.

Note the algorithm produces a dag-like resolution refutation.

12

We can also go efficiently in the reverse direction; from tree-like **R** to DPLL.

There is one technical difficulty: a variable in a **R** refutation might be resolved on more than once on the same path!

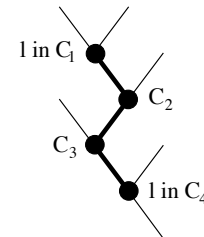
We say that a **R** refutation is **regular** if on every path from the leaves to the root (\square) each variable is resolved on at most once.

Thm. A tree-like **R** refutation can be transformed into a regular tree-like **R** refutation efficiently.

Proof. Suppose that we have a path in the refutation labelled by clauses C_1, C_2, \dots, C_k , $k > 2$, and there is a literal l in C_1 and in C_k , but l is not in any C_j , $1 < j < k$.

(Note that C_i is a premiss for C_{i+1} ; that's what being a path means here; also C_1 is *not necessarily* a leaf and C_k is *necessarily not* \square !)

9



Simply discard the first resolution on l .

We now have to modify all the C_i 's all the way down to \square . Here is how to do this.

In the above picture, discard the right-parent-subtree of C_2 , and let $C'_2 := C_1$.

Since we discarded the right-parent-subtree of C_2 , some literals will no longer be in the C'_i 's as we make our way to \square .

10

In the new monotone proof, define a clause to be **large** if it contains at least $n^2/10$ -many variables.

Suppose $S = \text{the number of large clauses} < 2^{n/20}$. We derive a contradiction, from which the theorem follows.

Claim: There is a P_{ij} contained in at least $S/10$ -many large clauses.

Proof of Claim: There are n^2 variables, and suppose that each of them is in less than $S/10$ many large clauses. Now what is the largest collection of clauses we can form where each clause is required to be large and we have $< S/10$ “copies” of each of the n^2 variables?

$$< \frac{n^2 \cdot S/10}{n^2/10} = S$$

Contradiction, since we *do* have S large clauses by assumption.

15

Choose such a P_{ij} , set it to 1, and set to 0 all P_{il} and $P_{l'j}$, where $l \neq j$ and $l' \neq i$.

Apply this restriction to the entire (monotone) refutation, to obtain a (monotone) refutation of PHP_{n-2}^{n-1} .

Technical point:

$$\begin{array}{ccc} \text{PHP}_{n-1}^n & \xrightarrow{\bar{P}_{ik} \rightsquigarrow \{P_{lk} | l \neq i\}} & \text{Monotone-PHP}_{n-1}^n \\ \downarrow & & \downarrow P_{ij}=1, \forall l \neq j, l' \neq i, P_{il}=0, P_{l'j}=0 \\ \text{PHP}_{n-2}^{n-1} & \longrightarrow & \text{Monotone-PHP}_{n-2}^{n-1} \end{array}$$

i.e., *commutative diagram*.

16

Correctness of DP

If the input \mathcal{S} is an unsatisfiable set of clauses, then the set of clauses resulting from each iteration remains unsatisfiable (*invariant*).

Let \mathcal{S}' be the result of one iteration on \mathcal{S} . Suppose $\sigma \models \mathcal{S}'$.

Then, one of the following two holds:

1. $\sigma \models C$ for all C such that $C \cup \{x\} \in \mathcal{S}$, or
2. $\sigma \models D$ for all D such that $D \cup \{\bar{x}\} \in \mathcal{S}$

Otherwise, $\sigma \not\models C_0$ and $\sigma \not\models D_0$, so $\sigma \not\models C_0 \cup D_0$, contradicting that $\sigma \models \mathcal{S}'$.

\therefore if 1., extend σ to $\hat{\sigma}$ such that $\hat{\sigma}(x) = 0$, and if 2., $\hat{\sigma}(x) = 1$.

Clearly, $\hat{\sigma} \models \mathcal{S}$.

13

Thm. Any resolution refutation of PHP_{n-1}^n requires at least $2^{n/20}$ many clauses.

Proof. A truth assignment (ta) σ is *i-critical* if σ leaves pigeon i out, and maps the remaining $(n-1)$ pigeons to holes bijectively.

Two clauses C_1, C_2 are *equivalent wrt critical ta's* if $\sigma \models C_1 \iff \sigma \models C_2$, for all critical σ .

Similarly, a resolution inference $C_1, C_2 \vdash C_3$ is *sound wrt critical ta's* if whenever a critical σ satisfies C_1, C_2 , it also satisfies C_3 .

Consider a refutation of PHP_{n-1}^n , and in every clause replace \bar{P}_{ik} by the literals $\{P_{lk} | l \neq i\}$.

All clauses in the refutation are now **monotone**, equivalent to the original wrt critical ta's, and all inferences are sound wrt critical ta's.

14

Let S be a subset of “pigeon” clauses that minimally implies C ,
 $|S| = m = \text{Complex}(C)$.

Claim: $|C| \geq (n - m)m$.

Note that $(n - m)m \geq 2n^2/9$, as it takes its min when at the
 extremes, i.e., when $m = n/3$ or $m = 2n/3$.

Proof of Claim: For $i \in S$ (short-hand for
 $\{P_{i1}, \dots, P_{i(n-1)}\} \in S$), consider an i -critical α such that $\alpha \not\models C$.

(If every i -critical α satisfied C , we would not need $i \in S$, since i is
 needed in S only if some i -critical α falsifies C .)

For each $j \notin S$ (remember $\text{Complex}(C) \leq 2n/3$), let α' be α except
 $\alpha'(i) = \alpha(j)$, and α' is j -critical.

19

$\alpha' \models C$.

Since $j \notin S$, and α' is j -critical, α' satisfies S , so it must satisfy C .

But α and α' are the same on all variables, except that if $\alpha \models P_{jl}$,
 then now $\alpha' \models P_{il}$.

Since $\alpha \not\models C$ but $\alpha' \models C$, it follows that $P_{il} \in C$.

Using the same α on all $j \notin S$ we get $(n - m)$ *distinct* variables
 $P_{il_1}, P_{il_2}, \dots, P_{il_{n-m}}$ in C .

Repeating the whole argument for each $i \in S$, gives us $(n - m)m$
 variables in C .

20

The number of large clauses in monotone PHP_{n-2}^{n-1} is at most $9S/10$.

Repeat this $\log_{10/9} S$ many times until we knock out all large
 clauses.

So we end up with a monotone refutation of $\text{PHP}_{n'-1}^{n'}$ where

$$n' \geq n - \log_{10/9} S > n(1 - (\log_{10/9} 2)/20) > 0.671n$$

and where there are no large (i.e., of size $\geq n^2/10$) clauses^a.

Lemma. Any (monotone) refutation of PHP_{n-1}^n must have a
 clause with at least $2n^2/9$ literals.

Contradiction!

$$2(n')^2/9 > n^2/10$$

^aActually, after $\log_{10/9} S$ many times you may still have one more large clause
 left, so really $n' > 0.671n - 1$. But the argument can still be made to work.

17

Proof of Lemma. For each C in the (monotone) refutation, let
 $\text{Complex}(C)$ be the minimum number of clauses in PHP_{n-1}^n that
 imply C on all critical α 's.

Note that only “pigeon” clauses $\{P_{i1}, P_{i2}, \dots, P_{i(n-1)}\}$ are included
 in a minimal set implying C (as we restrict ourselves to critical
 α 's).

The complexity of “pigeon” clauses is 1, and of \square is n .

If $C_1, C_2 \vdash C_3$, then $\text{Complex}(C_3) \leq \text{Complex}(C_1) + \text{Complex}(C_2)$.

Claim: $\therefore \exists C, n/3 < \text{Complex}(C) \leq 2n/3$.

Proof of Claim: Take C to be the class of smallest complexity
 $> n/3$ (such a class exists as $\text{Complex}(\square) = n$).

Its two premisses are therefore each of complexity $\leq n/3$.

18