**SAT PROBLEM**

Given a proportional formula

A (p1, … , pn) checks whether A is satisfiable (if there is a V such that V(A) =1?)

The truth table has 2^n rows, too big to be calculated, if you have to check every row it will take a lot of time

If there is an assignment (sudoku solution) if you get it to the first row you are lucky, but if you are not lucky and the row is very low you might not ever get there.

This is a problem still open, there is no machine able to do this truth table.

**Heuristic**: can help find the results. Can be sufficient in many cases. The general solutions still no exist today

**Heuristics**: are methodologies that do not bring theoretical benefits in the general case, but that can have a significant impact of success in the **practical cases of interest**.

2 methods:

1. **Saturation-based.**

Algorithm used: Resolution calculus. Good for many reasons. But it need a lot of memory storage

1. **Efficient control of backtracking.**

algorithm used: DPLL procedure. Now it is considered the best method

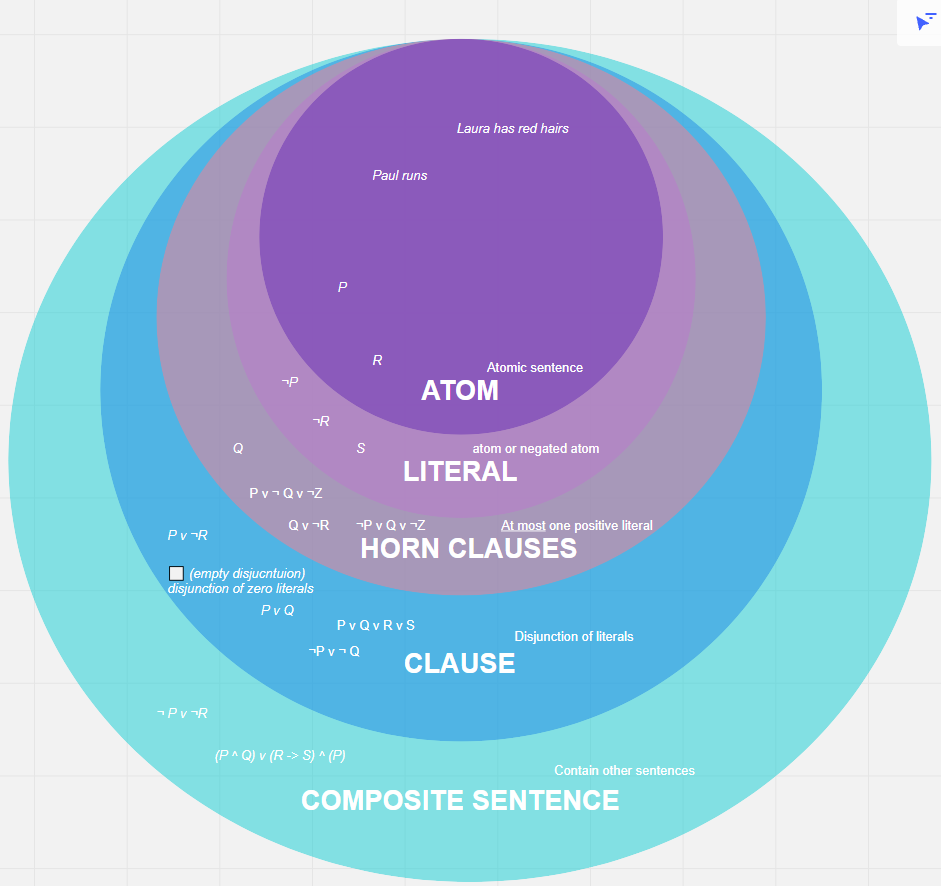
Resolution calculus and DPLL procedure need a preprocessing phase

Preprocessing → clause reduction

preprocessing: both resoluts don’t work with arbitrary formula, need a clause reduction

the set of clauses is satisfiable iff the original formula is satisfiable

| **notion** | **definition** | **examples** |
| --- | --- | --- |
| atom | atomic sentence | p, q, r, … |
| literal | atom or negated atom | p, q, ¬ p, ¬ q, ¬ r, … |
| clause | disjunction of literals | p, q, ¬ p, ¬ q, p v q, ¬ p v ¬ q…  ⬜ (empty disjunction, disjunction of 0 literals is p ^ ¬ p so something always false, also use ⟂ as symbol) |



A |-> a set of clauses set C = {C1, … Cn}

Equisatisfiabilty: there is a valuation (V) such that V(A) = 1 if and only iff there is a valuation such that V(c1 ^, … , ^ cn) = 1

Logical equivalence (stronger) (do not confuse with equisatisfiability)

For every V

V(A) = V(C1 ^, … Cn) = 1

If you have logical equivalence then you have equitisfiabilty, but not vice versa. Logical equivalence means that the truth table, every row of the truth table give the same result

Two formulas A, B are said to be **logically equivalent** iff A ↔ B is a tautology, i.e. (that is) iff (if and only if) we have V (A) = V (B) for every assignment V (this means that A and B have the same truth table).

Two formulas A, B are said to be **equisatisfiable** iff (A is satisfiable iff B is satisfiable).

P P v Q are both satisfiable (so they are equisatisfiable)

¬ P ^ Q ¬ ¬ (P ^ ¬ P) both unsatisfiable (so they are equisatisfiable)

¬ P ^ P Q are not equisaisiable (because ¬ P ^ P is unsatisfiable)

**Example1**

Check:

¬ (P ^ Q) ¬ P v ¬ Q

| ***P*** | ***Q*** | ***P ^ Q*** | ***¬ (P ^ Q)*** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

| ***P*** | ***Q*** | ***¬ P*** | ***¬ Q*** | ***¬ P v ¬ Q*** |
| --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |

¬ (P ^ Q) and ¬ P v ¬ Q have the same results, they are logically equivalent

**Example2**

Check:

¬ ( P ^ Q ) ¬ P ^ ¬ Q

| ***P*** | ***Q*** | ***P ^ Q*** | ***¬ ( P ^ Q )*** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

| ***P*** | ***Q*** | ***¬ P*** | ***¬ Q*** | ***¬ P ^ ¬ Q*** |
| --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

¬ ( P ^ Q ) ¬ P ^ ¬ Q aren’t logically equivalent but they are equisatisfiable

Logical equivalence they have the same meaning. Two formulas that have equisatisfaibilty can be very far from each other

NNF (Negation Normal Form)

CNF (Conjunctive Normal Form)

CNF: is a conjunction of clauses

(PvQ) **^** (Pv¬ Q) is a CNF

NNF: there are no implications and all negations are in front of atoms

((¬ P ^ (QvR) ) v S) is a NNF but not CNF

**NNF**: iff it does not contain implications ( → ) and all negations occurring in it are in front of atomic formulas.

( ¬ P ^ Q) **v** ( ¬ P)

**CNF**: iff it is a conjunction of clauses

( ¬ P v R) **^** ( ¬ Q v ¬ R)

1. **A is transformed into some A’** which is in negation normal form (NNF) and is logically equivalent to A
2. **A’ is transformed into some A’’** which is in conjunctive normal form (CNF) and is equisatisfiable to A’
3. Resolution (or, alternatively DPLL) operates on the set C of clauses whose conjunction is A’’: if an assignment satisfying C is eventually found, then the original formula A is satisfiable, otherwise it is not.

**LEMMA** (replacement)

Suppose I have a formula A and a reprice inside A some subformula B with a formula B’ which is logically equivalent to B. Then the result is logically equivalent to A

P ^ ¬ ( Q v R) → S (this is formula A)

¬ (Q v R) (is formula B) is logically equivalent to ( ¬ Q ^ ¬ R) (is formula B’), so I can substitute it

B = B’

P ^ ( ¬ Q ^ ¬ R) → S (this is equivalent to formula A)

If you take a piece of a formula and I replace this peace with another formula which is logically equivalent, than the global result is still logically equivalent

We know that:

A → B is logically equivalent to ¬ A v B

¬ ¬ A is logically equivalent to A

¬ (A ^ B) is logically equivalent to ¬ A v ¬ B

¬ (A v B) is logically equivalent to ¬ A ^ ¬ B

¬ ( A → B) is logically equivalent to A ^ ¬ B

If you have a formula A and you apply those transformations (whenever you find a subformula inside A which matches the part on the left of this table, you replace it with the part on the right of the table), in the end after many replacements you get A’ which is NNF

**Example:**

(orange means logically equivalent)

¬ (P → Q) → (P ^ Q → R)

(P ^ ¬ Q) → (P ^ Q → R)

(consider (P^ ¬ Q) as A and (P ^ Q → R) as B, so you have A→ B)

¬ (P ^ ¬ Q) v (P ^ Q → R)

(consider (P^Q) as A and R as B, so you have A → B)

¬ (P ^ ¬ Q) v ( ¬ (P v Q) v R)

( ¬ P v ¬ ¬ Q) v ( ¬ (P ^ Q) v R )

( ¬ P v Q ) v ( ¬ ( P ^ Q) v R)

( ¬ P v Q) v ( ¬ P v Q v R) now there is nothing to do anymore, and this is our final result

You can go in any order, doesn't matter

**Horn clauses**

Clauses with at most one positive literal

¬ P v ¬ Q and ¬ P v Q v ¬ R are Horn Clauses

P v Q v R is NOT a Horn Clauses

A → A’ → A’’

NNF CNF

How to do the transformation from NNF to CNF

CNF is the input for SAT solver

How to do it: there is a cheap way

A v (B ^ C) is logically equivalent to (A v B) ^ (A v C)

(B ^ C) v A is logically equivalent to (B v A) ^ (C v A)

This eliminates bad alternation and preserves logical equivalence. This method is expensive and the formula can become exponentially large.

**Example**

(P1 ^ Q1) v (P2 ^ Q2) v (P3 ^ Q3)

In the end you will get

(P1 V P2 V Q3) ^ (P1 v P2 v Q3) ^ (P1 v Q2 v P3)… you will get a huge formula. Have eight conjunctions in CNF and it started with 3. It is not a good idea to use this system. Can’t be the way to implementing

It is a non structural transformation

——

The method works correctly ONLY if you do the NNF and then CNF.

A v (B ^ C)

Step1: pick a new propositional letter *a*

(A v *a* ) ^ ( ¬ *a* v B) ^ ( ¬ *a* v C)

**General method**

Consider a subformula of the kind (D1 ^ D2) v C [bad alternation subformula]

Replace D1 ^ D2 with new propositional letter *a* and add clauses (¬ a v D1) ^ (¬ a v D2)

**Example**

((P1 ^ P2) v Q) ^ (R v S)

Replace P1^ P2 with a

↓

(a v Q) ^ (R v S) ^ ( ¬ A v P1) ^ ( ¬ A v P2)

* SAT problems: formula **A**
* NNF transformations up to logical equivalence **A’**
* CNN transformations **A’’** can be done in 2 ways
  + Distribution laws: up to logical equivalence but expensive, the formal is long and not convenient
  + Structure transformations: (use the new a) up to equisatiabilty and is linear, not expensive and cheap

**Resolution calculus**

¬ P1 v… v ¬ Pn v Q1 v… v Qm is a clause and it is equivalent to P1 ^ … , Pn ⟹ Q1 v … Qm

There are negative and positive literals

P1 , … , Pn → Q1, … , Qm conventional way of writing

Γ , p ⟹ Δ Γ ‘ ⟹ Δ ‘p

I can conclude that:

Γ ,Γ ‘ => Δ Δ ‘

Γ and Δ are sets of literals

If you have one literal on the left and the same literal on the right I can remove all of them and put all together

If I obtain the empty clause than UNSAT (unsatisfiable)

Otherwise SAT (satisfiable)

**Example**

{ ¬ P v ¬ Q, P v Q, ¬ P v Q, P v ¬ Q} unsat

(negative literal on the left of the arrow, positive literals on the right of the arrow)

P, Q ⟹ ⟹ P, Q P ⟹ Q Q ⟹ P

then I start applying the rule

⟹ P, Q P ⟹ Q remove p (one is positive one is negative)

=>Q, Q (it is redundant, remove one)

⟹ Q .

⟹ Q Q⟹ P

⟹ P (is the result)

P, Q ⟹ ⟹ Q

P ⟹ (is the result)

last one:

⟹ P P ⟹

⟹ (this is unsat)

1. Rewrite clauses in sequent notation
2. Apply resolution in all possible ways
3. There are two possibilities
   1. Get empty close ⟹ unsat
   2. Don’t get empty close ⟹ sat

{P v Q v S ¬ S v R ¬ R v P}

**Left negative and right positive**

⟹ P, Q, S. S⟹ R. R ⟹ P

S⟹ R R⟹ P

———————————

S ⟹ P

⟹ P, Q, S S ⟹ P

————————————

⟹ P, Q (remove one P because it is redundant)

⟹ P, Q, S S⟹ R

—————————-

⟹ P, Q, R

It is a saturation method: you add what you get until there's nothing to add. Try all possible ways. Or you get the empty close or you find the assignment (it is not easy) and so it is satisfiable