[*DPLL*](http://homes.dsi.unimi.it/~ghilardi/ls/SlidesDPLL.pdf)

**Introduction**

1950: first attempt to solve SAT problems, <10 variables

1960: Davis Putnam

1962: DPLL was invented

Now DPLL can work with 10’000 variables

To use DPLL need to use CNF (conjunctive normal form?) SAT

Euristic: method that has no effect in the worst possible case but in practical case is very efficient

**Preprocessing**

DPLL work with clauses, need a preprocessing process, need to use it from passing NNF (negation only in front of propositional letters) and CNF (conjunctions of clauses)

**Procedures:**

Pairs with a set of clauses (V,C)

C = set of clauses

V= partial assignment

ø and C0 you get either the empty clauses (the branch fail) or the set of clauses is empty (no clauses at all, it is a satisfiable assignment)

If all branch fail than it is unsat

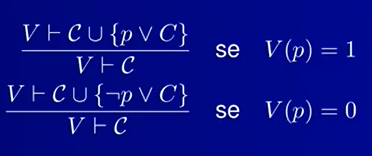
**DPLL RULES**

**Subsumption**

**Remove clauses**

If I have assign the value true (1) to the propositional letter P all clauses containing P can be removed

If P is true than P v Something is also true, remove a clause when I know it is satisfied

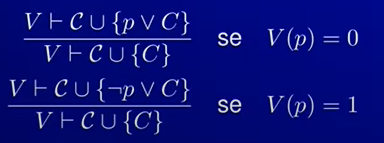


If I have assign the value false (0) to the propositional letter P all clauses containing ¬ P it is satisfied, can be removed

**Unit resolution**

**Remove literals**

If I have assigned the value 0 to p and I have PvC, in order to satisfy it I need to satisfy C, if P is false PvC is true only if C is true.



If I have assigned the value 1 to p I will remove ¬ p. If you have ¬ p v C and p is true, then for having the clause true C must be true

I remove the literals the opposite literals

If i have remove all the literal: empty clause

If i have a clause and remove a internal I get the empty clause

Empty clauses ≠ empty set of clauses

It is different to have the

* empty clause → faile branch
* empty set of clauses → SAT
* Unit resolution remove literals
* Subsumption remove clauses

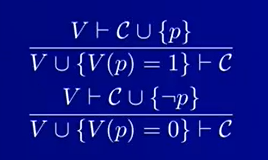
If I have a clause P and I remove the literal P I get the empty clause

If I have a clause P and I remove the clause P I get the remaining set of clauses. If no closes remain, I get the empty set of clauses

**Assertion**

Assert something

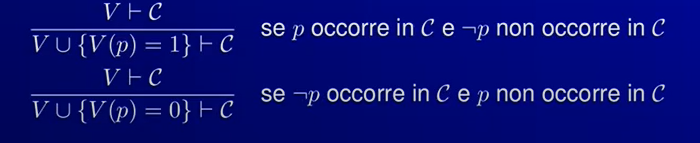
remove the unit clause



If I have clause P for it to be true I put P = 1

if I have ¬ P for it to be true I put P = 0

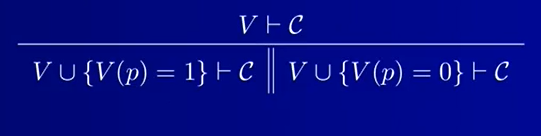
**Pure literal**



If P occurs in my clause and ¬ P never occurs it is convenient to put P = 1

If ¬ P occurs in my clauses and P never occurs it is convenient to put P = 0

**Splitting**

****

In case that no other rules apply

Two branches

This rule apply

**Example 1**

C0 = {p1 v p2 , p1 v ¬p2 , ¬p1 v p2 , ¬p1 v ¬p2}

| ø, C0 (start with the empty assignment) | split on p1 |
| --- | --- |

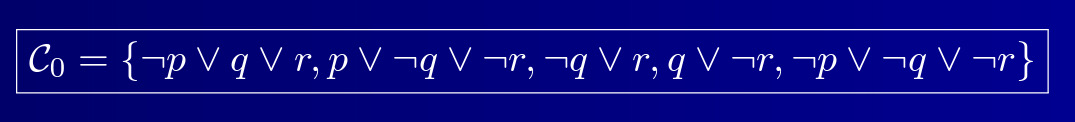
This is the first branch

| p1 = 1, C0  ↓ | subsumption (remove clauses with p1) |
| --- | --- |
| p1 = 1 , { ¬p1 v p2 , ¬p1 v ¬p2} | unit resolution (since p1= 1 I will remove ¬p1) |
| p1 = 1, { p2 ,¬p2} | assertion |
| p1= 1, p2 = 1, { ¬ p2} | Unit resolution  (since p2= 1 I will remove ¬p2) |
| p1= 1, p2 = 1 ⬜ (empty clause) | **failed branch** |

This is the second branch

| p1 = 0, C0 | subsumption (remove the clauses with ¬p1) |
| --- | --- |
| p1 = 0, {p1 v p2 , p1 v ¬p2} | unit resolution (since p1 = 0 remove all clauses contain p1) |
| p1 = 0, {p2 , ¬p2} | assertion |
| p1 = 0, p2= 1 {¬p2} | unit resolution (since p2 = 1 I will remove all the clauses containing ¬ p2) |
| p1 = 0, p2= 1 ⬜ (empty clause) | **failed branch** |

**Example 2**

****

C0 = { ¬p v q v r, p v ¬q v ¬r, ¬q v r, q v ¬r, ¬p v ¬q v ¬r}

This is the first branch

| ø, C0 | splitting on p |
| --- | --- |
| p = 1, { ¬p v q v r, p v ¬q v ¬r, ¬q v r, q v ¬r, ¬p v ¬q v ¬r} | subsumption, remove clauses containing p |
| p = 1, { ¬p v q v r, ¬q v r, q v ¬r, ¬p v ¬q v ¬r} | unit resolution, since p = 1 remove ¬p |
| p = 1, { q v r, ¬q v r, q v ¬r, ¬q v ¬r} | subsumption, remove clauses containing r |
| p = 1, r = 1, {q v ¬r, ¬q v ¬r} | unit resolution, since r = 1 remove all literals containing ¬ r |
| p = 1, r = 1, {q , ¬q} | subsumption, remove all clauses containing q |
| p = 1, r = 1, q = 1 { ¬ q} | unit resolution, since q = 1 remove all ¬q |
| p = 1, r = 1, q = 1 ⬜ (empty clause) |  |

This is the second branch

| ø C0 | splitting on p |
| --- | --- |
| p = 0, { ¬p v q v r, p v ¬q v ¬r, ¬q v r, q v ¬r, ¬p v ¬q v ¬r} | subsumption remove all clauses contain ¬p |
| p = 0, {¬q v ¬r, ¬q v r, q v ¬r} | unit resolution, if p = 0 remove all literals p |
| p = 0, q= 0 {¬q v ¬r, ¬q v r, q v ¬r} | subsumption, remove all clauses containing ¬ q |
| p = 0, q= 0 {q v ¬r} | unit resolution q=0. remove it |
| p = 0, q= 0 {¬r} | assertion, r = 0. remove the clause so get nothing |
| p = 0, q= 0 r= 0 ø | SAT empty set of clauses |

If I take p = 0, q = 0, r = 0 everything in the clauses become true

**Example 2 revisited**

C0 = { ¬p v q v r, p v ¬q v ¬r, ¬q v r, q v ¬r, ¬p v ¬q v ¬r}

This is the first branch

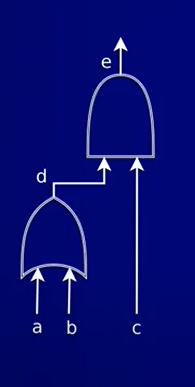
| ø ⊢ C0 | splitting on q |
| --- | --- |
| q = 1⊢ { ¬p v q v r, p v ¬q v ¬r, ¬q v r, q v ¬r, ¬p v ¬q v ¬r} | sumbumption. Since q = 1 i can remove all the clauses containing q = 1 |
| q = 1⊢ {p v ¬q v ¬r, ¬q v r, ¬p v ¬q v ¬r} | unit resolution, since q=1 ¬q=0 I can remove the literals |
| q = 1⊢ {p v ¬r, r, ¬pv ¬r} | subsumption. r = 0, remove all the clauses containing ¬ r . remove the clauses |
| q = 1 , r = 0 ⊢ {r} | unit resolution, since r = 0 I can remove the literal |
| q = 1 , r = 0 ⊢ ⬜ (empty clause) | failed branch |

This is the second branch

| ø ⊢ C0 | splitting on q |
| --- | --- |
| q = 0 ⊢ { ¬p v q v r, p v ¬q v ¬r, ¬q v r, q v ¬r, ¬p v ¬q v ¬r} | subsumption. remove all clauses contain ¬ q |
| q = 0 ⊢ { ¬p v q v r, q v ¬r} | unit resolution. q = 0. Remove literal |
| q = 0 ⊢ { ¬p v r, ¬r} | subsumption r = 1, remove clauses containing r |
| q = 0, r = 1 ⊢ {¬r} | unit resolution r = 1 so ¬ r = 0, remove literal |
| q = 0, r = 1 ⊢⬜ (empty clause) | failed branch |

**example of application**

Main use of SAT solver is for hardware verification



this is a circuit with AND and OR gates

gate d is a or b

d ↔ (a v b)

gate e is c and d

e ↔ (c ^ d)

SAT solver to check if everything is correct in circuits