**CHAPTER 3**

Defined version of DPLL

CDCL: CONFLICT DRIVEN CLAUSE LEARNING

Save computation

When DPLL

Backjumping: not always possible, also called *non chronological backtracking* not coming back to the last split but earlier. A chronological backtracking will be to come back to the last split

Learning: some clauses are logical consequences of the initial clauses, the machine is trying to learn, at some point it realizes that something is a consequence.

If ¬ p is a logical consequence it is oblivious that there will not be any more split

In some cases during the computation the information arrives, keep track of the information

C0 initial set of clauses

(V | C | α )

V partial assignment

C set of clauses (include C0 and maybe some “learned” clauses)

an algorithm can be in 2 states

α can be

\* search state

C “conflict state” (C is conflict clause)

Conflict clause is a logical consequence of C0

The algorithm starts with a search state, then conflicts, then searches… ends with UNSAT if it has the empty clauses as conflict clause empty clause can be satisfiable so it is unsat

The algorithm ends with SAT if it is in search state and no rule applies. There are 6 rules:

* 3 rules for search state
* 3 rules for conflict state

**Initialisation**

( ø | C0 | \* )

ø empty assignment

C0 input set of clauses

\* search state

The partial assignment is represented as a list of literals,

P ¬ Q R

V(P) = 1

V(Q) = 0

V(R) = 1

We agree to use the “bar” notation for complementary literal

If L is P than is ¬ P

if L is ¬ P than is P

The bar on top mean the other one

Each literal in the partial assignment is marked as:

* “**decided**” - the literal is there because of a splitting rule
* “**propagated**” - if it is there by another rule

(V | C | α )

When l is propagated there is a clause l v D € L such that l v D = l v P v ¬ Q

V(p) = 0

V(q) = 1

All literals have been assignment in the wrong way

( ø | C0 | \* )

empty assignment

initial set of clauses

search state

**Propagate (search rule)**

(V | C, D v l | \* )

if V(d) = 0, all literals in D are assignment in a way that D is false

in this case add the literal l, l is a propagated literal

↓

(V , lDvl | C, Dvl | \* )

The literal l is been assigned because if I want Dvl to be true I am forced to make this literal true.

**Conflict enter (search rule)**

If you are in a situation

(V | C, D | \* )

Supposed V(D) = 0 all literal in D have been assigned in the wrong way. there is something wrong, need to backtrack

↓

Enter conflict state

(V | L, D | D )

When entering in conflict there is need of some revision, if possible, need some review

**Decide or splitting (search rule)**

This apply if no other rule applies

(V | C | \* )

Add certain literal market as decided if L is not yet assigned and L occurs in C

(V, Ld | C | \* )

I some are deceived some propagated

(Pd, ¬ QC, ¬ RD… | C | \* )

P value, Q is false, R is false, some are decided (marked with d) some are propagated (marked with the clause that cause the propagation)

If I go on like that without finding a conflict I get an assignment, if I enter a conflict state need to review and recover

**Failure (conflict rule)**

(V | C| ⬜️) UNSAT

if the conflict clause is the empty clause there is nothing to do

The empty clause is always a logical consequences of the original clause

**Explain rule (conflict rule)**

(V | C D v | LvC)

↓

propagated D v

When entering a conflict clause there is certainly a propagated literal.

L v C

V(C) = 0 i propagated L

Using resolution

D v L v C

—————————-

C v D

Replace

(V | C D v | C v D)

**Backjumping (conflict rule)**

(V1, ld V2 | L | C v L’)

L’ is decided

Supposed V1(C) = 0 then come back and add L’ and CvL’ as a learned clauses and back to search state:

(V1 L’ C v l’ | C, CvL’ | \* )

L’ is propagated

In search state I can

* propagate
* go to conflict
* make a split if there is no other possibility

In conflict clause if I get the empty clause I exit or I can make a resolution step for a propagated literal or for a decided literal I can backtrack

**Example**

*1 2 3 4*

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

Start with empty assignment, C0 and search state

( ø | C0 | \* ) → take p1 decided ( p1 | C0 | \* )

start propagating. Since p1 is true and *2* need to be true, we must have p2 true

→ ( p1d, p2*2* | C0 | \* )

we are getting in a conflict because *4* is evaluated false because p1 and p2 are both true. Go to conflict case

→ ( p1d, p2*2* | C0 | ¬ p1 v ¬ p2 )

In this conflict clause p2 comes from a propagated literal. Try to make a resolution step between the conflict clause (*4*) and the clause that forced the propagation of p2 (*2*)

¬ p1 v ¬ p2 ¬ p1 v p2

--------------------------------- remove complementary literal (p2)

¬ p1

Replace the conflict clause with the resolvent

→ ( p1d, p2*2* | C0 | ¬ p1 )

Use the backjumping rule, remove everything after the decision

*5* *5*

→ ( ¬ p1 | C0, ¬ p1 | \* )

Go back to search state, consider clause (*3*)

→ ( ¬ p1*5* , ¬p2*3* | C0, ¬ p1 | \* )

Get to conflict state because (*1*)

→ ( ¬ p1*5* , ¬p2*3* | C0, ¬ p1*5* | p1 v p2 )

resolution step between conflict clause and the clause used to propagate ¬ p2

p1 v p2 p1 v ¬ p2

-------------------------------

p1

Replace the conflict clause with the resolvent

→ ( ¬ p1*5* , ¬p2*3* | C0, ¬ p1*5* | p1 )

p1 is propagated, resolution between conflict clause p1 and *5*

p1 ¬ p1

------------

⬜️

final step:

→ ( ¬ p1*5* , ¬p2*3* | C0, ¬ p1*5* | ⬜️ ) UNSAT

**EXERCISE**

*1 2 3 4 5 6 7 8*

C0 = {a v r, b v c, ¬a v ¬p, ¬a v ¬q, ¬b v p, ¬b v ¬r, ¬c v q, ¬c v ¬r } UNSAT

Start with empty assignment C0 and search state

( ø | C0 | \* )

take a decided

→ (a | C0 | \* )

start propagating. since a is true ¬ p is true ( *3*)

→ (ad , ¬p*3* | C0 | \* )

start propagating. since a is true ¬ q is true ( *4*)

(ad , ¬p*3* , ¬q*4* | C0 | \* )

start propagating. Since p is false ¬b is true ( *5*)

(ad , ¬p*3* , ¬q*4* , ¬b*5*| C0 | \* )

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

start an empty assignment, propagate a new variable, conflict, enter a conflict clause, if you are in search and no possibility of conflict or propagation make a split case and a decision. In conflict state make resolution with propagated literals unity empty clause or in the conflict clause you see a literal that has been decided, revert the decision and come back to the earòies possibile decision point

If you take natural numbers there are not infinitely many. star from 100 that 99, 98 at certain time you will stop

if you take pairs you can do **lexicographic comparison**

(n,m) > (n’, m’) iff either n>n’ or n=n’ and m>m’

also the lexicographic comparison has no infinite decreasing chains.

(p1 p2 p3) p1d p2p p3unassigned

1 0 2

whenever you apply a search rule or whenever you come back to search from conflict this vector decreases lexicographic

* you can not stay in conflict forever
* you can not go out and back from search infinitely many time, whenever you apply a search rule or come back to search state the vector decreases, can go on forever