**Dimacs**

SATsolvers: Zchaff mimisat picosat…

If you want to download a SATsolver picosat, easier to install

First line:

P CNF <number of letters> <number of clauses>

C <comment>

Z3 accept any format

P1v ¬ P2 will be 1 -2 0

propositional letters are named with numbers, if they are negated there is the minus, at the end of the line there is a zero

It is complicated to write program in this way (dimax formula with picosat)

**DPLL**

Davis-Putnam-Logemann-Loveland

Basic version

More refined: CDCL

The resolution is memory consuming: a saturation basic method apply the resolution until is possible, can generate exponentially many clauses, for this point of view it is even worth than the truth table (2^n rows)

For looking at the saturation you examine and if you don’t get a satisfaction you delete it, you reduce the number of rows.

Truth table are not memory consuming but are terrible, want something more practical in practical cases than truth table and not memory consuming

Basic idea of DPLL: is Boolean propagation

P v Q v R

If you know that P and Q are zero (partial assignment) you know that R is 1, because the clause must be true.

Maximise to have a truth sentence, there’s no point in trying the alternatives.

Trying to eliminate the duplication that is not needed, when we have seen unit resolution in the horn clauses it was similar. unit resolution if you have a literal that is already assigned that you can use it for resolution (because it is more convenient)

When it is not possible to solve the problem with Boolean propagation only you take linear time

In case that Boolean propagation does not apply

if you can’t make any chose you can make a case split

Make the case split only when propagation does not apply!

If you apply P 1 0 first explore 1 if this is not ok go to 0, like the truth table. AMount of memory occupied is linear

**Backjumping mechanics**: may happened to done more than one case split, if you fail instead that considering the last case split I can in certain circumstances jump to earlier case split

Input is a set of clauses C0

Need to CNF conversion

The algorithm maintains a partial assignment V and a set of clauses C

(V, C)

Initially V is empty and I have the clauses C0

(ø , C0) is the initial state

than I apply rules and generate a tree

I use boolean propagation rules or splitting rules, ti will generate a tree, in the leaves you can have a certain assignment and an empty set of clauses (V, ø ) or you can have an assigment an the empty clause (V, ⬜️)

empty clauses have three different notation

⬜️

⟹

┴

If there is a leaf like(V, ø ) than SAT

If all leafs are like (V, ⬜️) than UNSAT

Number of leaves is exponential, can’t be avoided but the tree is explored in depth truth, until I found a leaf that works or all leaves are unsat.

In order to keep the occupation of the memory small I explore one branch of the tree at time, otherwise the memory occupation will be exponentially large

start with the empty assignment and initial set of clauses

apply rules

make case split or not

at certain point you will get an assignment and no closes (V, ø ) or an assignment and the empty clause (V, ⬜️)

**Rules**

Subsumption rule

I have an assignment V, a set of clauses C which include a clause like *p or c* and in my assignment I have that p is true

V, C u {p v c} V(p)=1

——————-

V, C

I remove p v c because if my assignment assign to p is true the clause is satisfied and can be removed

V, C u { ¬ p v c } V(p) =0

—————-

V, C

¬ p v c is satisfied and it is removed

I will remove the clauses that are already satisfied

**Unit resolution rule**

V, C u {p V c} V(p) = 0

——————

V, C u {C}

I remove the p

If p is 0 in ordered to satisfy *p or c* i need to satisfy c because p is 0

V, C u { ¬ p V c} V(p) = 1

—————— WHY

V, C

**Assertion rule**

If you have V and a unit clauses {p}

V, C u {p}

——

V u {V(p) = 1}, C

V, Cu{ ¬p]

——

V u {V(p) =0], C

**Pure literal role**

V, C p occurs C but ¬ p never occurs

——-

V u {V(p) = 1], C

evaluate p to 1 it is convenient to do that because ¬ p never occurs

V, C ¬ p occurs in C but p never occurs

——-

V u {V(0) = 0], C

**Only if no other rule applies**

Then SPLITTING rule

V, C

———-

/ \

V u {V(p) = 1 ], C V u {V(p) =0 ], C

Empty closet {p1 vp2, ¬ p1 v p2, p1 v ¬ p2, ¬ p1 v ¬ p2}

subsumption: does not apply, we don’t have any evaluation

unit resolution: don’t have assign anything

assertion: no, we don’t have unit clauses here

The only possibile is the split

Empty closets, {p1 vp2, ¬ p1 v p2, p1 v¬ p2, ¬ p1 v ¬ p2}

/ \

Split left : V(p1) = 1 {p1 v p2, ~~¬ p1~~ v p2, p1 v ¬ p2, ~~¬ p1~~ v ¬ p2}

Remove ¬ p1

V(p1) = 1 {~~p1 v p2~~, p2, ~~p1 v ¬ p2~~,¬ p2}

Now you can apply subsumption, all clauses containing P1 can be removed

(**Subsumption: remove clauses!!**

**Unit resolution: remove literals!!**

If you remove a literal if nothing remains is the empty clause

If you remove the clauses which have only one literal I have removed the clause, you removed one and nothing happen)

V(p1) = 1. {~~p2~~,¬ p2}

Then I can apply the assertion rule

Remove p2

V(p1) = 1 V(p2)= 1 {¬ p2}

Use unit resolution, remove negated literal

Now you have empty clause (you removed a literal not a clause)

V(p1) = 1 V(p2)= 1 ⬜️ (empty clause)

It fails

Split right:

V(p1)= 0 {~~p1~~ v p2, ¬ p1 v p2, ~~p1~~ v ¬ p2, ¬ p1 v ¬ p2}

Remove all literals p1 (unit resolution) because they are false

V(p1)= 0 { p2, ~~¬ p1 v p2~~,¬ p2, ~~¬ p1 v ¬ p2~~}

Using subsumption, remove the clause not contain ¬ p1

V(p1)= 0 { ¬ p2, p2}

V(p1) = 0 V(p2) = 1 { ¬ p2}

V(p1) = 0 V(p2) = 1 ⬜️ (empty clause)

The final answer is both empty clauses so it is **unsat**

Pages 25-26 of notes