CS 228 tut 4

10.8 CDCL Run

@1 $\neg p_1 \vee p_2$

@2 ~~$\neg p_1$~~ $\vee$ ~~p_3~~ $\vee p_5$

@1 ~~$\neg p_2$~~ $\vee p_4$

@1 $\neg p_3 \vee \neg p_4$

@2 $p_1 \vee p_5 \vee$ ~~$\neg p_2$~~

@1 $p_2 \vee$ ~~p_3~~

@1 $p_2 \vee \neg p_3 \vee p_7$

@2 $p_6 \vee$ ~~$\neg p_5$~~

$\neg p_3$ @1 → p_2 @1 → p_4 @1

p_1 @2 → p_5 @2 → p_6 @2

$\neg p_7$ @2

@3 ~~$\neg p_1$~~ $\vee p_2$

@3 ~~$\neg p_1$~~ $\vee p_3 \vee$ ~~p_5~~

~~$\neg p_2$~~ $\vee p_4$

~~$\neg p_3$~~ $\vee \neg p_4$

@3 $p_1 \vee$ ~~p_5~~ $\vee \neg p_2$

@3 $p_2 \vee p_3$

@3 $p_2 \vee \neg p_3 \vee$ ~~p_7~~

@1 ~~p_6~~ $\vee \neg p_5$

p_1 @3 → p_2 @3 → p_3 @3 → conflict

$\neg p_6$ @1 → $\neg p_5$ @1

learn $\neg p_1$

can backtrack to any decision level, depending on heuristic

Correspondence with algo pseudocode

$m := \emptyset$ incrementally build model

$dl := 0$ how many decisions have been made

$dstack := 0$ assign 0 everything to

$dstack(n) = l$

after taking n decisions in this run, the size of the model was l.

```
do while false                    ← arrived at a REFUTATION
   dl = 0                         ← no backtracking possible
;  (C, dl) = analyseConflict()
;  m.resize(dstack(dl)), F = F ∪ {C}
   unitprop
while unassigned or false

if unassigned
   dstack(dl) = m.size()
   decide,
   unitprop
```

dl returned means go back to the state just after these many decisions were made

before making $(n+1)^{st}$ decision record how big

DISCLAIMER careful about

error!

model was,
so you can
backtrac

10.9 Proof that CDCL terminates

Key: Identify something that "progresses" in the sense, can't repeat, and is bounded

Eg. some integer qty that is non-negati but keeps decreasing

GENERAL way to prove termination

Idea: (m, dl) whenever a run finds a model/ conflict

Suppose CDCL doesn't terminate in some case

- Either SAT model — but this terminates.
- * - We keep getting conflict with $dl > 0$

Key: There are finitely many partial assignments

m

CLAUSE LEARNING: if (m, dl) is seen at a conflict, then if m is seen again (m, dl') $dl' < dl$

dl can only take values $0 < dl \leq$ no. of vars in F

If we can prove this then we have contradiction due to assuming infinite runs

Run R_1

m, dl : CONFLICT : Learn a clause $l_{i_1} \vee l_{i_2} .. \vee l_{i_k}$

later m, dl' CONFLICT

consider the run that

Run R_2 led to it

$m \supseteq \{ \bar{l}_{i_1}, \bar{l}_{i_2}, \bar{l}_{i_3} \ldots \}$

WLOG, $\bar{l}_{i_k}$ was assigned LAST

due to the LEARNT clause, l_{i_k} could have been assigned by unitProp

thus l_{i_k} CANNOT have been a decision variable.

decision variables of R_2 $\subset$ decision variables of R_1

$dl' < dl$

Formalising the notion that a conflicting partial assignment will never be tried again, from the analysis of the algo.

11.11 n queens, SAT encoding.

BTW for the record

Queens move in straight lines.

Horizontally, vertically, diagonally

Strategy

q_{ij}

queen on square (i,j)

1) No two on the same straight line

$q_{ij} \Rightarrow \neg q_{i'j'}$ $i'j'$ collinear with ij

i, ij'

for every pair of squares that are on the same line

$\neg q_{ij} \vee \neg q_{i'j'}$

Can be a little systematic: define a total order on squares

How many of these 2-clauses

$O(n^3)$

2) We need n queens

At least one queen in each row.

Suffices

n clauses of width n

$O(n^2)$

Total complexity $O(n^3)$

$n = \exp(\text{bitsize})$

11.12 Peaceable queens.

SAT is a decision problem
i.e. Answer is YES/NO

Optimization is computation.

Make a guess of the optimum, and
DECIDE if it is correct

N is it possible to put N queens?

Important to adapt the encoding
easily for different N.

1) White and black queens do not attack [same as before]

w_{ij} : white queen stands on i,j
b_{ij} : black queen " " "

2) There are at least/exactly N queens of each colour.

My favorite:
cardinality constraints via merge sort

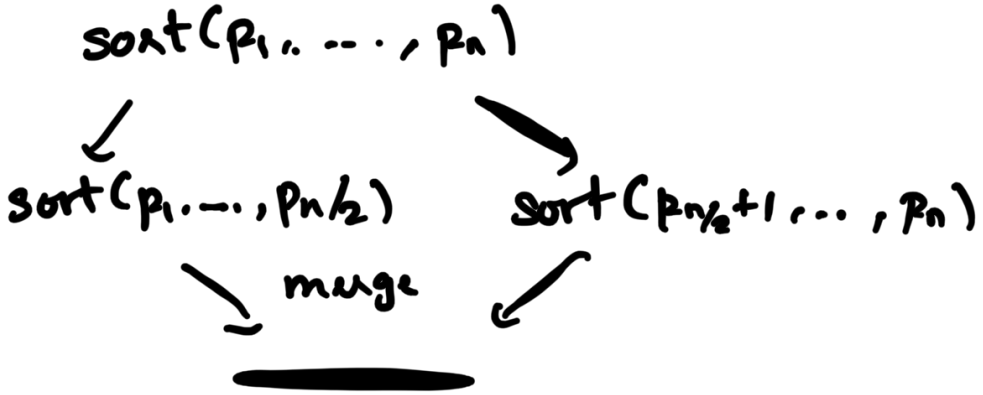
$p_1, \ldots, p_n$

{ related by clauses

$y_1, \ldots, y_n$ [permutes $p_1, \ldots, p_n$]

$y_1 \geq y_2 \geq \ldots \geq y_n$

$p_1, p_2 \longrightarrow p_1 \vee p_2,\ p_1 \wedge p_2$



Difference: this is enforced through clauses

Base Case

p_1, p_2

$y_1 \Leftrightarrow p_1 \vee p_2$

$y_2 \Leftrightarrow p_1 \wedge p_2$

Merge:

$x_1 x_2 \ldots x_n$

$y_1 y_2 \ldots y_n$

n decisions to make while INTERLEAVING

There is a mechanism to get sorted booleans

$y_1, \ldots, y_n$

$> k$: check if y_{k+1} is true

$= k$: $y_k \wedge \neg y_{k+1}$

$< k$: $\neg y_k$

To parametrize N, number of peaceful queens, you just modify this small final constraint.

Z3 practice problems.

3-bit state machine follows some update rules

find a cycle!

vs: variables

ups: FORMULAE

p, q, r ups[0] ups[1] ups[2]

Have to introduce new booleans to express the cycle!

p2, q2, r2 } Invoke get_fresh_vec

p3, q3, r3 }

ENCODE the transitions, and the fact they form a cycle.

p2 == ups[0]
q2 == ups[1]
r2 == ups[2]

Key is to realise the

p3, q3, r3 : p2, q2, r2 :: p2, q2, r2 : p, q, r

ups, except for the fact that we plug p2, q2, r2, in for p, q, r

ups

substitute_vars

p3 == ups2[0] and so on.

CYCLE COMPLETION

p == ups3[0]

vars : p3 q3 r3

p, q, r : T, F, F

At most one: Brute force: quadratic.

Fresh variables (my solution)

as in the slides

Subtle: slides has at most one, we need exactly one.

$s_i \Rightarrow \neg p_{i+1}$

$s_{i-1} \vee p_i \Leftrightarrow s_i$

My convoluted code. check if this

Common sink

Another way: simply add disjunction of all vars. ↳ works too

Use this as a subroutine in sudoku

+ Each square has exactly one number
- By k, I mean an integer 1-9

2. Each row has exactly one k
3 column
4 3×3 box

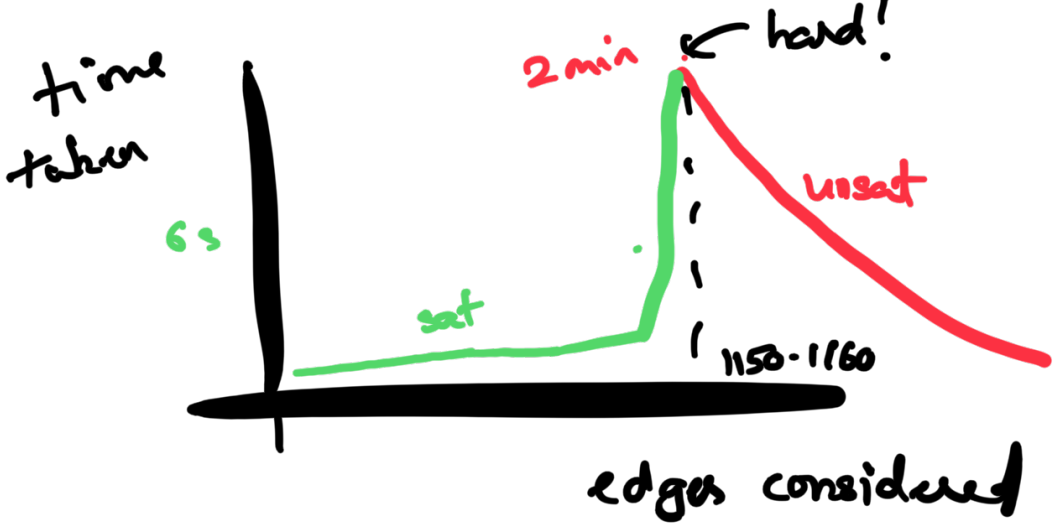
beware of some cumbersome book-keeping.

Strategy: var[i][j][k]
k+1 is entered in cell i,j

Graph 3-colouring

Run the code, see what happens!
Takes 5-6 minutes

Encoding straightforward, discussed in lectures too.



Will provide my code for reference,
but best for you to try it YOURSELF,
and look only when in doubt.