Example: Find an Equisatisfiable Set of Formulas in CNF

$$\{\boxed{c \land a} \lor (\neg c \land b)\}$$

Example: Find an Equisatisfiable Set of Formulas in CNF

$$\{\boxed{c \land a} \lor (\neg c \land b)\}$$

$$\{x_1 \lor \boxed{\neg c \land b},\ x_1 \longleftrightarrow (c \land a)\}$$

Example: Find an Equisatisfiable Set of Formulas in CNF

$$\{\boxed{c \wedge a} \vee (\neg c \wedge b)\}$$

$$\{x_1 \vee \boxed{\neg c \wedge b},\ x_1 \longleftrightarrow (c \wedge a)\}$$

$$\begin{aligned} \{x_1 \vee x_2,\ \ & x_2 \longleftrightarrow (\neg c \wedge b), \\ & x_1 \longleftrightarrow (c \wedge a)\} \end{aligned}$$

Example: Find an Equisatisfiable Set of Formulas in CNF

$$\{\boxed{c \wedge a} \vee (\neg c \wedge b)\}$$

$$\{x_1 \vee \boxed{\neg c \wedge b},\ x_1 \longleftrightarrow (c \wedge a)\}$$

$$\begin{aligned}\{x_1 \vee x_2,\ & x_2 \longleftrightarrow (\neg c \wedge b),\\ & x_1 \longleftrightarrow (c \wedge a)\}\end{aligned}$$

$$\begin{aligned}\{x_1 \vee x_2,\ & x_2 \rightarrow (\neg c \wedge b),\ (\neg c \wedge b) \rightarrow x_2,\\ & x_1 \rightarrow (c \wedge a),\ (c \wedge a) \rightarrow x_1\}\end{aligned}$$

$$\begin{aligned}\{x_1 \vee x_2,\ & \neg x_2 \vee \neg c,\ \neg x_2 \vee b,\ c \vee \neg b \vee x_2,\\ & \neg x_1 \vee c,\ \neg x_1 \vee a,\ \neg c \vee \neg a \vee x_1\}\end{aligned}$$

When representing clauses as sets:

$$\begin{aligned}\{\{x_1, x_2\},\ & \{\neg x_2, \neg c\},\ \{\neg x_2, b\},\ \{c, \neg b, x_2\},\\ & \{\neg x_1, c\},\ \{\neg x_1, a\},\ \{\neg c, \neg a, x_1\}\}\end{aligned}$$

Exercise: Find an Equisatisfiable CNF

$$\neg(((c \land a) \lor (\neg c \land b)) \longleftrightarrow ((c \rightarrow b) \land (\neg c \rightarrow b)))$$

Exercise: Find an Equisatisfiable CNF

$$\neg(((c \land a) \lor (\neg c \land b)) \longleftrightarrow ((c \to b) \land (\neg c \to b)))$$

$$(\ (\neg c \lor \neg a)\ \land\ (c \lor \neg b)\ )\ \ \oplus\ \ (\ (c \land \neg b)\ \lor\ (\neg c \land \neg b)\ )$$

Exercise: Derive $\emptyset$ From These Clauses Using Resolution

Davis-Putnam-Logemann-Loveland (DPLL) Algorithm Sketch

```
def DPLL(S: Set[Clause]) : Bool =
  val S' = subsumption(UnitProp(S))
  if ∅ ∈ S' then false // unsat
  else if S' has only unit clauses then true // unit clauses give e
  else
    val L = a literal from a clause of S' where {L} ∉ S'
    DPLL(S' ∪ {{L}}) || DPLL(S' ∪ {{complement(L)}})

def UnitProp(S: Set[Clause]): Set[Clause] = // Unit Propagation (BCP)
  if C ∈ S, unit U ∈ S, resolve(U,C) ∉ S
  then UnitProp((S - {C}) ∪ {resolve(U,C)}) else S

def subsumption(S: Set[Clause]): Set[Clause] =
  if C1,C2 ∈ S such that C1 ⊆ C2
  then subsumption(S - {C2}) else S
```

Why are answers correct? Why does it terminate?

Data Structures in a SAT Solver

Previous algorithm

- generates new clauses in `UnitProp`
- deletes clauses in `UnitProp` and `subsumption`

This is very inefficient. SAT solvers use more efficient data structures:

- all unit clauses are represented as current assignment, a candidate environment e, a *partial map* from some of the variables to truth values (starts as empty map)
  - unit clause $\{\neg a\}$ becomes $e(a) = 0$, unit clause $\{a\}$ becomes $e(a) = 1$
- whenever a new literal L becomes true, we check if e assigns its value in the contradictory way and, if so, we detect a *conflict*, corresponding to $\emptyset$
- instead of resolving $\{L_1, L_2, \ldots, L_n\}$ with a unit literal $\{\overline{L_1}\}$: interpret each clause in the context of current e: once $[\![L_1]\!]_e = 0$, we interpret clause as $\{L_2, \ldots, L_n\}$
- when all except for one literal in a clause are 0, the remaining literal gives a new variable in e (or a conflict)
- instead of subsumption: mark and ignore clauses that are true in current e

Generating Simple Proofs from SAT Solver Runs

With CDCL (conflict-driven clause learning), the solver maintains the progress in exploring the space using learned clauses.
Each learned clause is derived by resolution from existing ones.
If we find a top-level conflict, we have a derivation of $\emptyset$

Given the length of proofs and subtlety of SAT solvers, there exist very compact formats that can be checked independenty using simple and efficient proof checkers, which operate in polynomial time.

Running time of the solver is at least as long as the size of the proof.

There exists combinatorial statements (e.g. Pigenhole principle) that generate an infinite family of unsat formulas $F_1, F_2, \ldots$ such that the shortest resolution proof $F_i \vdash \emptyset$ is exponential in the size of F_i.
Alasdair Urquhart: Hard examples for resolution. J. ACM 34, 1 (Jan. 1987), 209-219.

Propositional Interpolants from Proofs

Bounded model checking precisely unfolds transition relation. In the worst case, we may need to copy the transition relation as many steps as the number of states (2^n for sequential circuit with n states).

In *interpolation-based model checking* algorithms, we try to discover inductive invariant while doing bounded model checking.

Definition

Let F, G be propositional formulas. An **interpolant** for (F, G) is a formula H such that the following three conditions hold:

- $F \models H$
- $H \models G$
- $FV(H) \subseteq FV(F) \cap FV(G)$

Note that these conditions imply that $F \models G$. Formulas H is between F and G and serves as an *explanation* why F implies G, without referring to variables that are specific to F or specific to G.

Example of Interpolant

Does there exist an interpolant H for these two formulas:

- $F : (c \rightarrow a) \wedge (\neg c \rightarrow b)$
- $G : d \rightarrow a \vee b$

Which variables may an interpolant H contain?

Existence and Lattice of Interpolants

Theorem

Let F, G be propositional formulas such that $F \models G$ and let S be the set of interpolants (F, G). Then the following hold:

- *If $H_1, H_2 \in S$ then $H_1 \wedge H_2 \in S$ and $H_1 \vee H_2 \in S$*
- *S is non-empty*
- *There exists $H_{min} \in S$ such that, for all $H \in S$, $H_{min} \models H$*
- *There exists $H_{max} \in S$ such that, for all $H \in S$, $H \models H_{min}$*