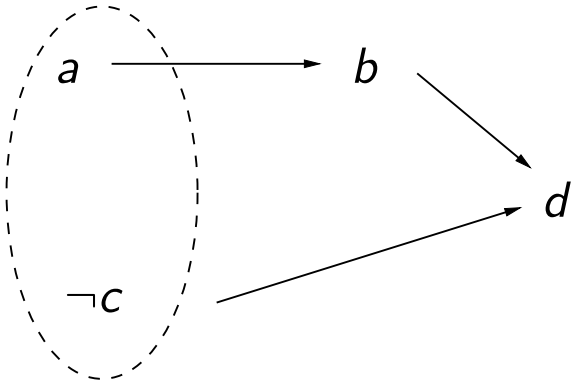
Principles: DPLL-style SAT solvers

(slides from McMillan's tutorial at CAV 2003)

- DPLL = Davis-Putnam-Loveland-Logeman (early 60's)
- CHAFF, GRASP, BERKMIN, ...
- Objective: check satisfiability of a CNF formula
  - literal: p or $\neg p$
  - clause: disjunction of literals
  - CNF: conjunction of clauses
- Method
  - Branch: make arbitrary decisions
  - Propagate implication graph
  - Use conflicts to guide inference steps

The implication graph

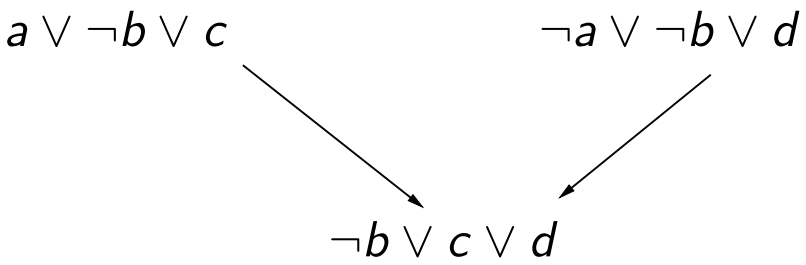
- Unit Propagation (UP) or Boolean Constraint Propagation (BCP)
- Consider CNF formula: $(\neg a \vee b) \wedge (\neg a \vee c \vee d)$



Assigment:
$a \wedge b \wedge \neg c \wedge d$

decisions

Resolution

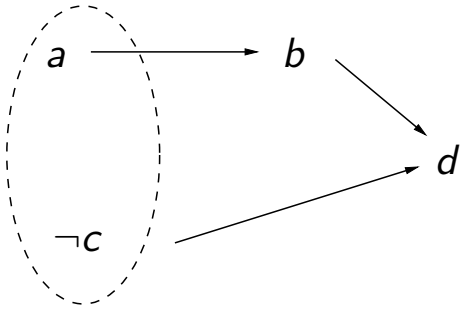


Resolution is used to solve conflicts

Implication graph used to guide resolution

Conflict clauses (1)

$$(\neg a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee \neg d)$$



decisions

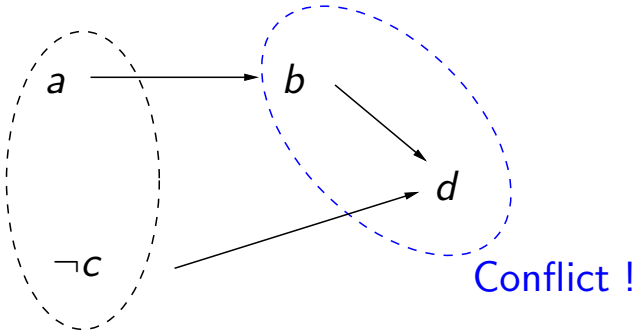
Implication of decisions with first 2 clauses: $a \wedge b \wedge \neg c \wedge d$, and then conflict with last clause

Conflict clauses (1)

$$(\neg a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee \neg d)$$

Resolve

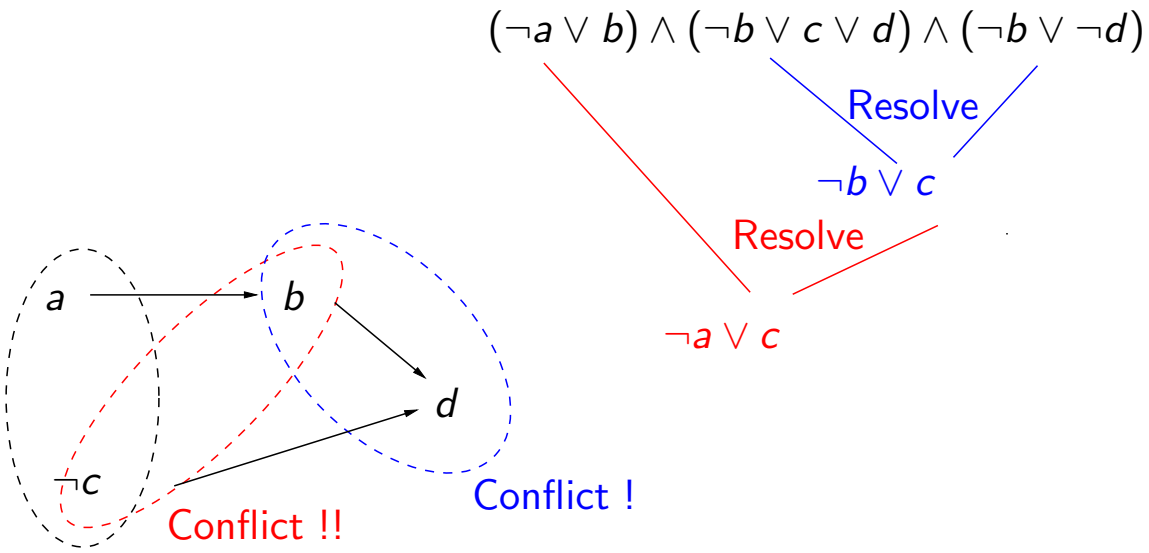
$$\neg b \vee c$$



decisions

Assigment $a \wedge b \wedge \neg c \wedge d$.
Resolution with last clause first, conflict with assignment

Conflict clauses (1)



decisions

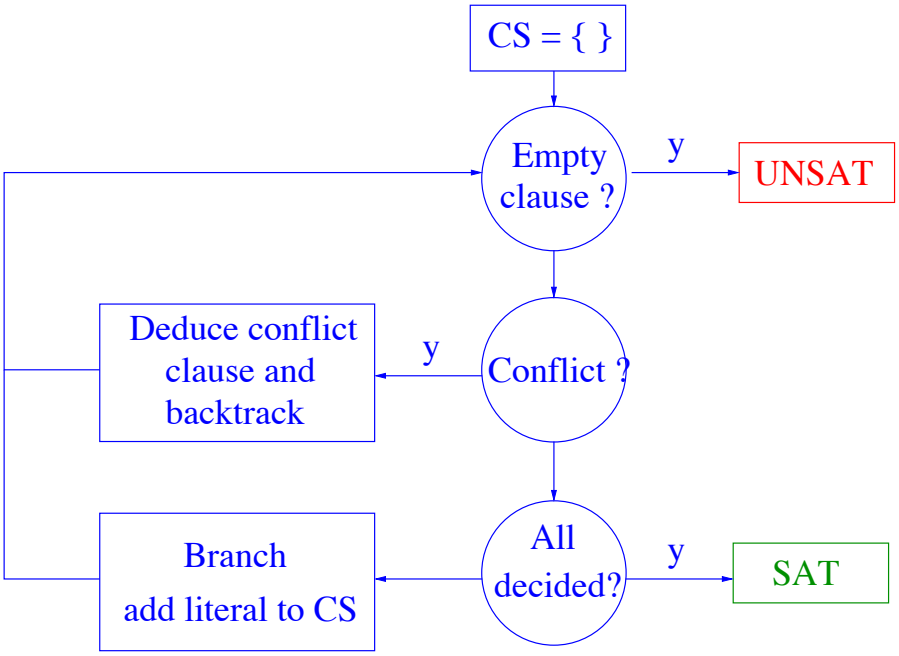
Assigment $a \wedge b \wedge \neg c \wedge d$.

Resolution continued and new decision: start with $\neg a$ and c

Conflict Clauses (2)

- Generated by resolution
- Implied by existing clauses
- In conflict with current assignment
- Safely added to the clause set
- Heuristics/Implementations
  - when to stop resolution
  - which clauses should be used for resolution

Basic SAT algorithm



Summary(1)

- SAT is an NP-hard problem
- Efficient implementations today/clever heuristics
  - which literals to consider (decisions)
  - ordering of propagation (BCP)
  - cache-aware implementations
  - pre-processing
  - learning
  - restarts
  - ...
- Every NP problem can be reduced to SAT in polynomial time
  - General solver + tuning can be effective

Summary (2)

- Industrial applications (PowerPC and Intel's Pentium 4)
  - SAT-based very good for small depth bugs in large systems
- Automated test generation
- SAT on problems with hundreds of thousands of clauses
- SAT competition