Solvers for the Problem of Boolean Satisfiability (SAT)

Will Klieber

15-414

Aug 31, 2011

Why study SAT solvers?

- Many problems reduce to SAT.
  - Formal verification
  - CAD, VLSI
  - Optimization
  - AI, planning, automated deduction
- Modern SAT solvers are often fast.
- Other solvers (QBF, SMT, etc.) borrow techniques from SAT solvers.
- SAT solvers and related solvers are still active areas of research.

Negation-Normal Form (NNF)

- A formula is in negation-normal form iff:
  - all negations are directly in front of variables, and
  - the only logical connectives are: "$\wedge$", "$\vee$", "$\neg$".
- A **literal** is a variable or its negation.
- Convert to NNF by pushing negations inward:

$$\neg(P \wedge Q) \Leftrightarrow (\neg P \vee \neg Q)$$
$$\neg(P \vee Q) \Leftrightarrow (\neg P \wedge \neg Q)$$

(De Morgan's Laws)

Disjunctive Normal Form (DNF)

- Recall: A *literal* is a variable or its negation.
- A formula is in DNF iff:
  - it is a disjunction of conjunctions of literals.

$$\underbrace{(\ell_{11} \wedge \ell_{12} \wedge \ell_{13})}_{\text{conjunction 1}} \vee \underbrace{(\ell_{21} \wedge \ell_{22} \wedge \ell_{23})}_{\text{conjunction 2}} \vee \underbrace{(\ell_{31} \wedge \ell_{32} \wedge \ell_{33})}_{\text{conjunction 3}}$$

- Every formula in DNF is also in NNF.
- A simple (but inefficient) way convert to DNF:
  - Make a truth table for the formula φ.
  - Each row where φ is true corresponds to a conjunct.

Conjunctive Normal Form (CNF)

- A formula is in CNF iff:
  - it is a conjunction of disjunctions of literals.

$$\underbrace{(\ell_{11} \vee \ell_{12} \vee \ell_{13})}_{\text{clause 1}} \wedge \underbrace{(\ell_{21} \vee \ell_{22} \vee \ell_{23})}_{\text{clause 2}} \wedge \underbrace{(\ell_{31} \vee \ell_{32} \vee \ell_{33})}_{\text{clause 3}}$$

- Modern SAT solvers use CNF.
- Any formula can be converted to CNF.
  - Equivalent CNF can be exponentially larger.
- Equi-satisfiable CNF (Tseitin encoding):
  - Only linearly larger than original formula.

Tseitin transformation to CNF

- Introduce new variables to represent subformulas.

$$\text{Original: } \exists\vec{x}.\,\phi(\vec{x})$$

$$\text{Transformed: } \exists\vec{x}.\exists\vec{g}.\,\psi(\vec{x},\vec{g})$$

- E.g, to convert (A ∨ (B ∧ C)):
  - Replace (B ∧ C) with a new variable g_1.
  - Add clauses to equate g_1 with (B ∧ C).
  - $(A \vee g_1) \wedge \underbrace{(B \vee \neg g_1)}_{(\neg B \rightarrow \neg g_1)} \wedge \underbrace{(C \vee \neg g_1)}_{(\neg C \rightarrow \neg g_1)} \wedge \underbrace{(\neg B \vee \neg C \vee g_1)}_{((B \wedge C) \rightarrow g_1)}$
  - Gives value of g_1 for all 4 possible assignments to {B, C}.

Tseitin transformation to CNF

Convert (A ∨ (B ∧ C)) to CNF by introducing new variable g_1 for (B ∧ C).

$$(A \vee g_1) \wedge \underbrace{\underbrace{(\neg g_1 \vee B)}_{(g_1 \rightarrow B)} \wedge \underbrace{(\neg g_1 \vee C)}_{(g_1 \rightarrow C)}}_{(g_1 \rightarrow (B \wedge C))} \wedge \underbrace{\underbrace{(\neg B \vee \neg C \vee g_1)}_{((B \wedge C) \rightarrow g_1)}}_{((B \wedge C) \rightarrow g_1)}$$

$$\underbrace{(g_1 \rightarrow (B \wedge C)) \quad \wedge \quad ((B \wedge C) \rightarrow g_1)}_{(g_1 \Leftrightarrow (B \wedge C))}$$

SAT Solvers -- Representation

- A CNF formula is represented by a set of clauses.
  - Empty set represents a true formula.
- A clause is represented by a set of literals
  - Empty set represents a false clause.
- A variable is represented by a positive integer.
- The logical negation of a variable is represented by the arithmetic negation of its number.
- E.g., $((x1 \vee x2) \wedge (\neg x1 \vee \neg x2))$ is represented by $\{\{1, 2\}, \{-1, -2\}\}$

Naïve Approach

- SAT problem: Given a boolean formula φ, does there exist an assignment that satisfies φ?
- Naïve approach: Search all assignments!
  - n variables $\rightarrow$ 2^n possible assignments
  - Explosion!
- SAT is NP-complete:
  - Worst case is likely $O(2^n)$, unless P=NP.
  - But for many cases that arise in practice, we can do much better.

Unit Propagation

- Davis-Putnam-Logemann-Loveland (DPLL)
- Unit Clause: Clause with exactly one literal.
- Algorithm:
  - If a clause has exactly one literal, then assign it true.
  - Repeat until there are no more unit clauses.
- Example:
  - $((x1 \vee x2) \wedge (\neg x1 \vee \neg x2) \wedge (x1))$
  - $((T \vee x2) \wedge (F \vee \neg x2) \wedge (T))$
  - $((T) \wedge (\neg x2))$
  - T

Helper function

```
from copy import copy, deepcopy
def AssignLit(ClauseList, lit):
    ClauseList = deepcopy(ClauseList)
    for clause in copy(ClauseList):
        if lit in clause: ClauseList.remove(clause)
        if -lit in clause: clause.remove(-lit)
    return ClauseList
```

```
>>> AssignLit([[1, 2, -3], [-1, -2, 4], [3, 4]], 1)
[[-2, 4], [3, 4]]

>>> AssignLit([[1, 2, -3], [-1, -2, 4], [3, 4]], -1)
[[2, -3], [3, 4]]
```

Assumption: No clause contains both a variable and its negation.

Naïve Solver

```
def AssignLit(ClauseList, lit):
    ClauseList = deepcopy(ClauseList)
    for clause in copy(ClauseList):
        if lit in clause: ClauseList.remove(clause)
        if -lit in clause: clause.remove(-lit)
    return ClauseList

def IsSatisfiable(ClauseList):
    # Test if no unsatisfied clauses remain
    if len(ClauseList) == 0: return True

    # Test for presense of empty clause
    if [] in ClauseList: return False

    # Split on an arbitrarily decided literal
    DecLit = ClauseList[0][0]
    return (IsSatisfiable(AssignLit(ClauseList, DecLit)) or
            IsSatisfiable(AssignLit(ClauseList, -DecLit)))
```

DPLL Solver

```
def IsSatisfiable(ClauseList):
    # Unit propagation
    repeat until fixed point:
        for each unit clause UC in ClauseList:
            ForcedLit = UC[0]
            ClauseList = AssignLit(ClauseList, ForcedLit)
```

unchanged

```
    # Test if no unsatisfied clauses remain
    if len(ClauseList) == 0: return True
    # Test for presense of empty clause
    if [] in ClauseList: return False
    # Split on an arbitrarily decided literal
    DecLit = (choose a variable occuring in ClauseList)
    return (IsSatisfiable(AssignLit(ClauseList, DecLit)) or
            IsSatisfiable(AssignLit(ClauseList, -DecLit)))
```

GRASP: an efficient SAT solver

Original Slides by Pankaj Chauhan
Modified by Will Klieber

Please interrupt me if anything is not clear!

Terminology

- CNF formula φ
  - $x_1,\ldots, x_n$: n variables
  - $\omega_1,\ldots, \omega_m$: m clauses

$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3$

$\omega_1 = (x_2 \vee x_3)$

$\omega_2 = (\neg x_1 \vee \neg x_4)$

$\omega_3 = (\neg x_2 \vee x_4)$

$A = \{x_1{=}0, x_2{=}1, x_3{=}0, x_4{=}1\}$

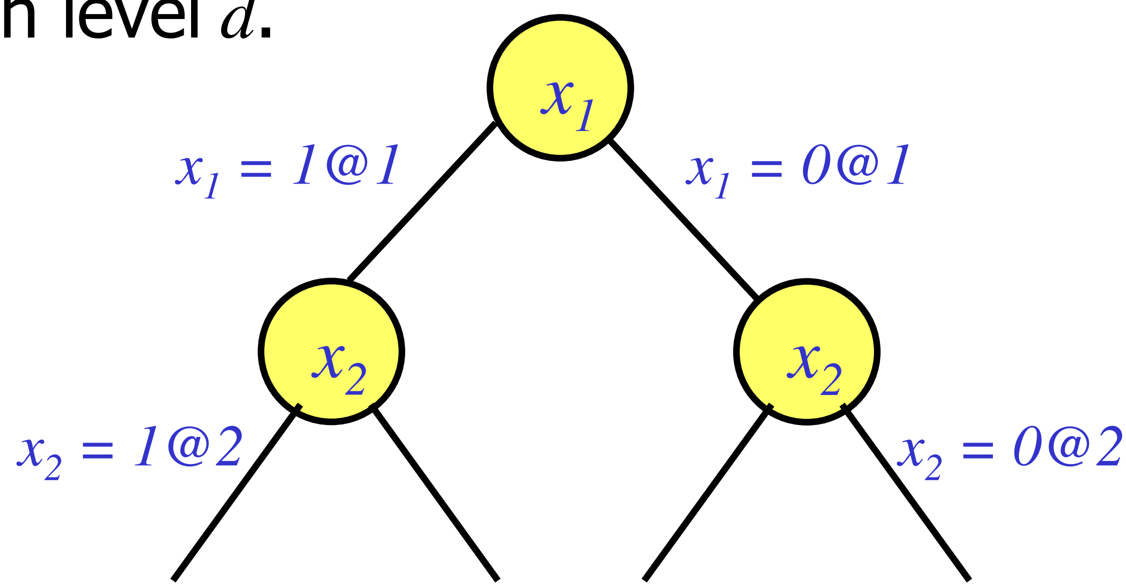
- Assignment A
  - Set of *(variable, value)* pairs.
  - Notation: $\{(x_1,1), (x_2,0)\}$, $\{x_1{:}1, x_2{:}0\}$, $\{x_1=1, x_2=0\}$, $\{x_1, \neg x_2\}$
  - $|A| < n \rightarrow$ partial assignment $\{x_1{=}0, x_2{=}1, x_4{=}1\}$
  - $|A| = n \rightarrow$ complete assignment $\{x_1{=}0, x_2{=}1, x_3{=}0, x_4{=}1\}$
  - $\varphi|_A = 0 \rightarrow$ falsifying assignment $\{x_1{=}1, x_4{=}1\}$
  - $\varphi|_A = 1 \rightarrow$ satisfying assignment $\{x_1{=}0, x_2{=}1, x_4{=}1\}$
  - $\varphi|_A = X \rightarrow$ unresolved asgnment $\{x_1{=}0, x_2{=}0, x_4{=}1\}$

Terminology

- An assignment partitions the clause database into three classes:
  - Satisfied, falsified, unresolved
- Free literal: an unassigned literal
- Unit clause: has exactly one free literal

Basic Backtracking Search

- Organize the search in the form of a decision tree.
  - Each node is a decision variable.
  - Outgoing edges: assignment to the decision variable.
  - Depth of node in decision tree is decision level $\delta(x)$.
  - " $x=v$ @ d " means variable x is assigned value v at decision level d.



Basic Backtracking Search

1. Make new decision assignments.
2. Infer implied assignments by a deduction process (unit propagation).
   - May lead to falsifying clauses, **conflict!**
   - The assignment is called “conflicting assignment”.
3. Conflicting assignments leads to backtrack.

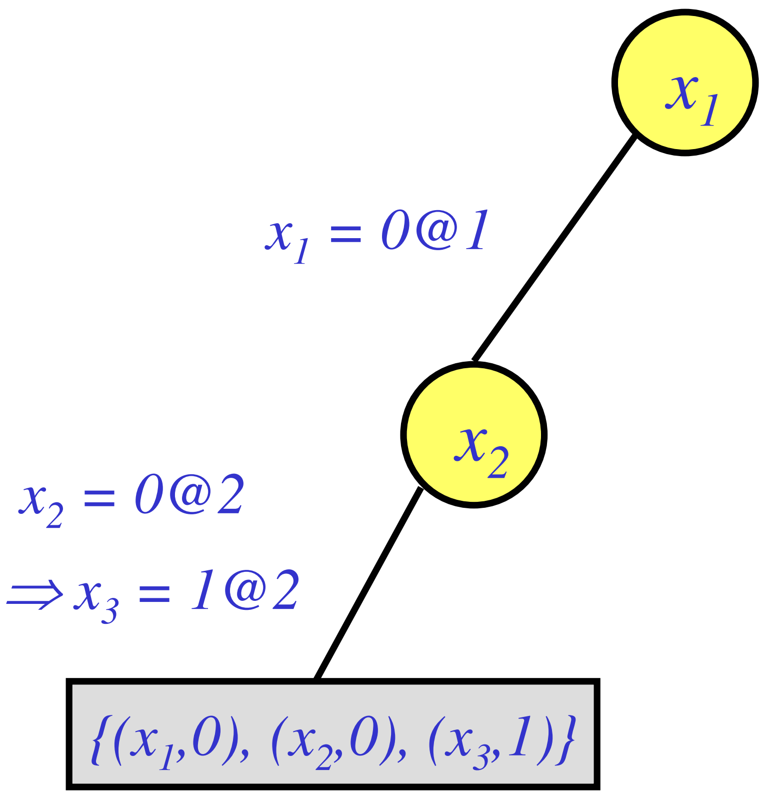
Backtracking Search in Action

Example 1

$$\omega_1 = (x_2 \vee x_3)$$

$$\omega_2 = (\neg x_1 \vee \neg x_4)$$

$$\omega_3 = (\neg x_2 \vee x_4)$$



No backtrack in this example!

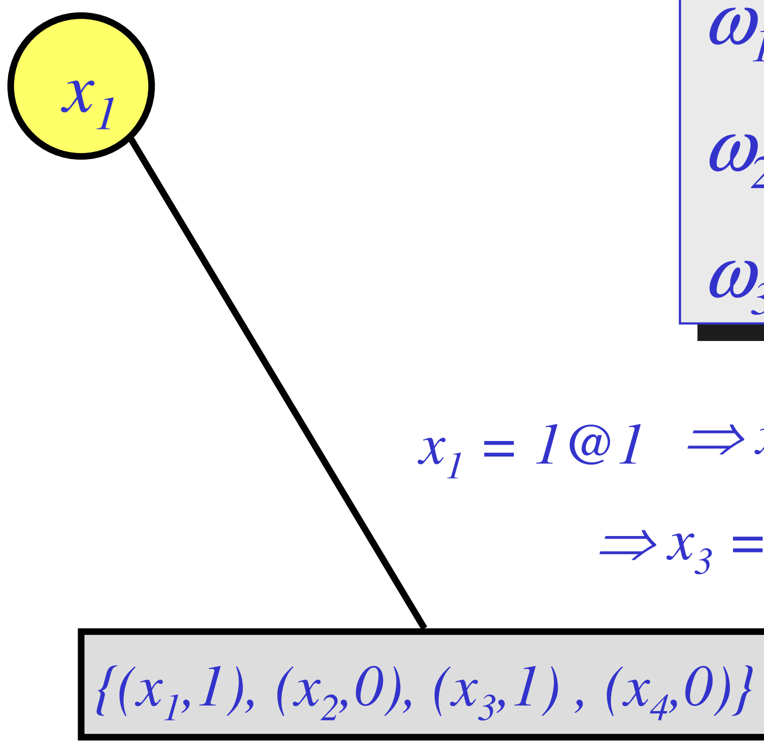
Backtracking Search in Action

Example 2

$$\omega_1 = (x_2 \vee x_3)$$

$$\omega_2 = (\neg x_1 \vee \neg x_4)$$

$$\omega_3 = (\neg x_2 \vee x_4)$$



$$x_1 = 1@1 \Rightarrow x_4 = 0@1 \Rightarrow x_2 = 0@1$$

$$\Rightarrow x_3 = 1@1$$

No backtrack in this example!

Backtracking Search in Action

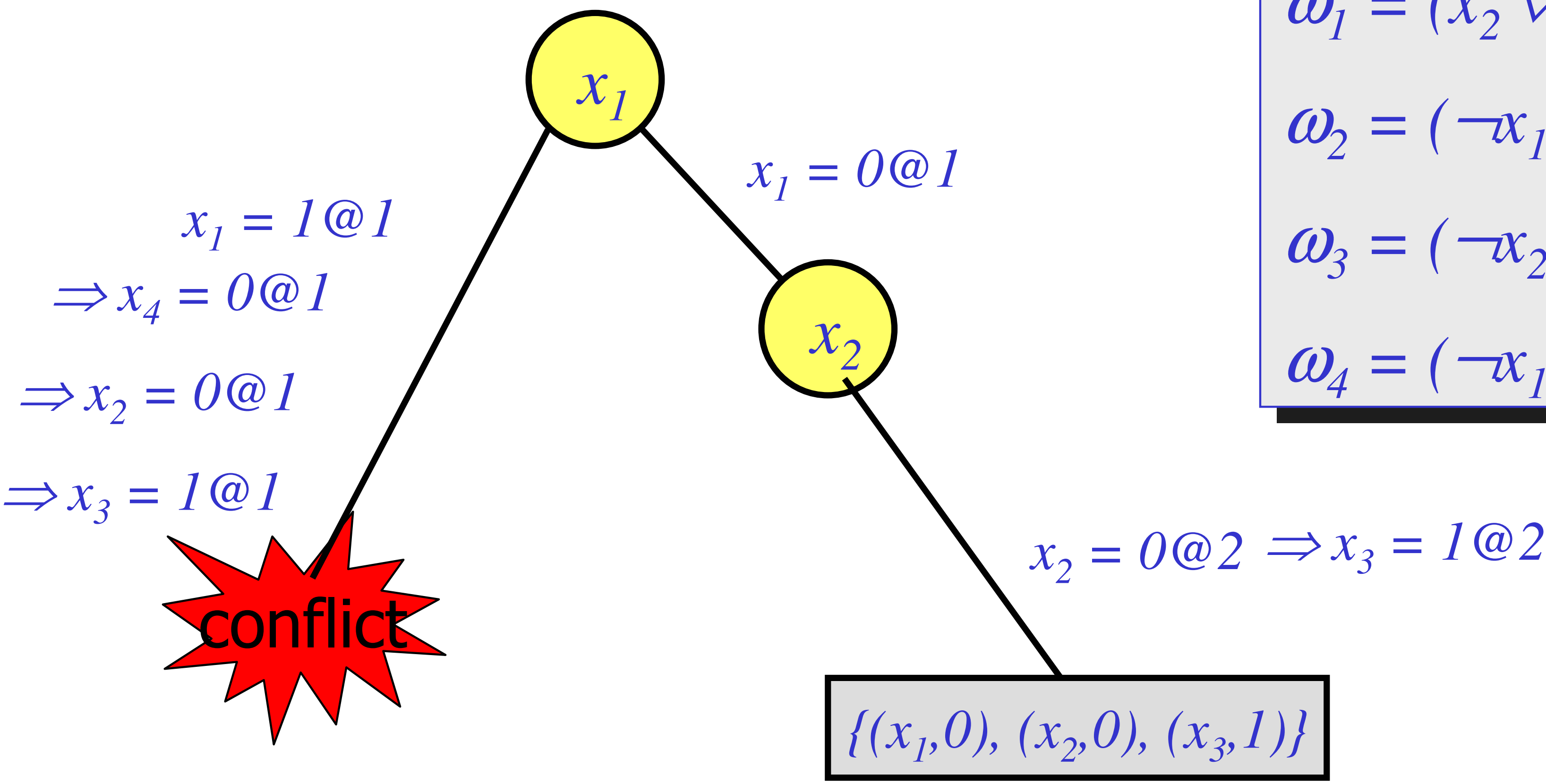
Example 3

$$\omega_1 = (x_2 \vee x_3)$$

$$\omega_2 = (\neg x_1 \vee \neg x_4)$$

$$\omega_3 = (\neg x_2 \vee x_4)$$

$$\omega_4 = (\neg x_1 \vee x_2 \vee \neg x_3)$$



GRASP

- GRASP is Generalized seaRch Algorithm for the Satisfiability Problem (Silva, Sakallah, '96).
- Features:
  - Implication graphs for Unit Propagation and conflict analysis.
  - Learning of new clauses.
  - **Non-chronological** backtracking!

Learning

- GRASP can learn new clauses that are logically implied by the original formula.
- Goal is to allow Unit Prop to deduce more forced literals, pruning the search space.
- Example:
  - φ contains clauses $(x \vee y \vee z)$ and $(x \vee y \vee \neg z)$.
  - **Resolving** on z yields a new clause $(x \vee y)$.
  - If y is false, then x must be true for φ to be true.
    - But not discoverable by simple Unit Prop w/o resolvent clause.
  - Clause $(x \vee y)$ allows Unit Prop to force $x=1$ when $y=0$.
- New clauses learned from conflicting assignments.

Resolution

From

$$(x_1 \vee \cdots \vee x_n \vee r) \wedge (\neg r \vee y_1 \vee \cdots \vee y_m)$$

deduce

$$(x_1 \vee \cdots \vee x_n \ \vee \ y_1 \vee \cdots \vee y_m)$$

Top-level of GRASP-like solver

```
CurAsgn = {};
while (true) {
    while (value of φ under CurAsgn is unknown) {
        DecideLit();   // Add decision literal to CurAsgn.
        Propagate();  // Add forced literals to CurAsgn.
    }
    if (CurAsgn satisifies φ) {return true;}
    Analyze conflict and learn a new clause;
    if (the learned clause is empty) {return false;}
    Backtrack();
    Propagate();  // Learned clause will force a literal
}
```

GRASP Decision Heuristics

- Procedure `DecideLit()`
- Choose the variable that satisfies the most clauses
- Other possibilities exist

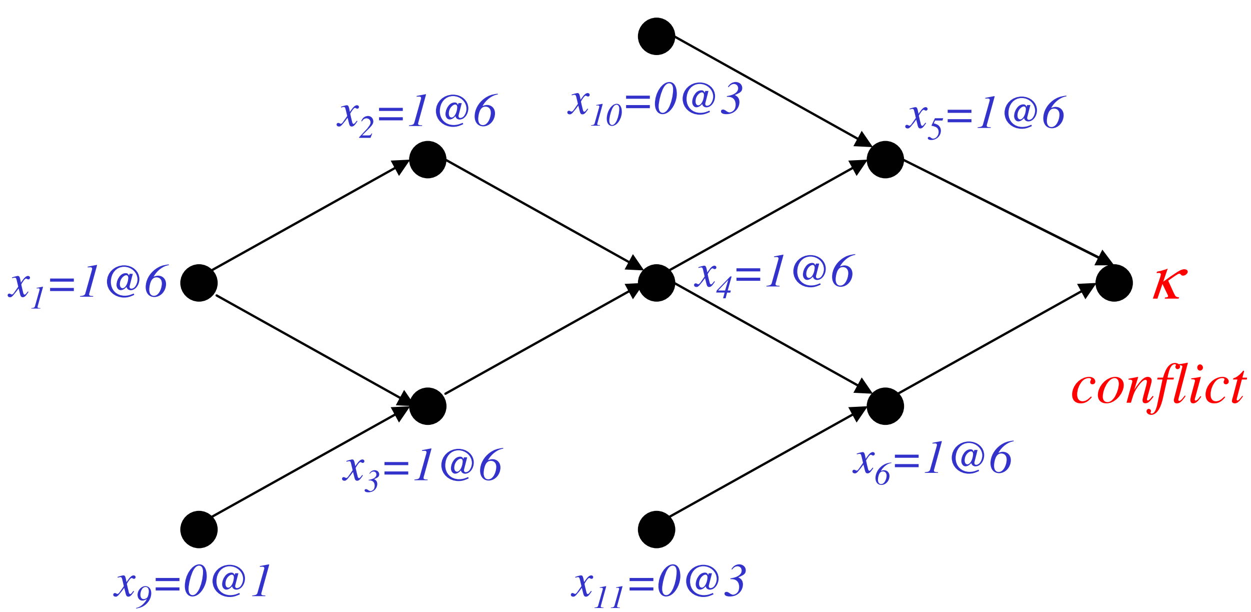
GRASP Deduction

- Unit Propagation is a type of Boolean Constraint Propagation (BCP).
- Grasp does Unit Prop using implication graphs:
  E.g., for the clause $\omega = (x \vee \neg y)$,
  if $y=1$, then $x=1$ is forced; the antecedent of x is $\{y=1\}$.
- If a variable x is forced by a clause during BCP, then assignment of 0 to all other literals in the clause is called the antecedent assignment $A(x)$.
  - E.g., for $\omega = (x \vee y \vee \neg z)$,

    $$A(x) = \{y{:}0, z{:}1\},\ A(y) = \{x{:}0, z{:}1\},\ A(z) = \{x{:}0, y{:}0\}$$
  - Variables directly responsible for forcing the value of x.
  - Antecedent assignment of a decision variable is empty.

Implication Graphs

- Depicts the antecedents of assigned variables.
- A node is an assignment to a variable.
  - (decision or implied)
- Predecessors of x correspond to antecedent $A(x)$.
  - No predecessors for decision assignments!
- For special conflict vertex κ, antecedent $A(\kappa)$ is assignment to vars in the falsified clause.



Example Implication Graph

Current truth assignment: $\{x_9=0@1, x_{12}=1@2, x_{13}=1@2, x_{10}=0@3, x_{11}=0@3\}$

Current decision assignment: $\{x_1=1@6\}$

$\omega_1 = (\neg x_1 \vee x_2)$

$\omega_2 = (\neg x_1 \vee x_3 \vee x_9)$

$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$

$\omega_4 = (\neg x_4 \vee x_5 \vee x_{10})$

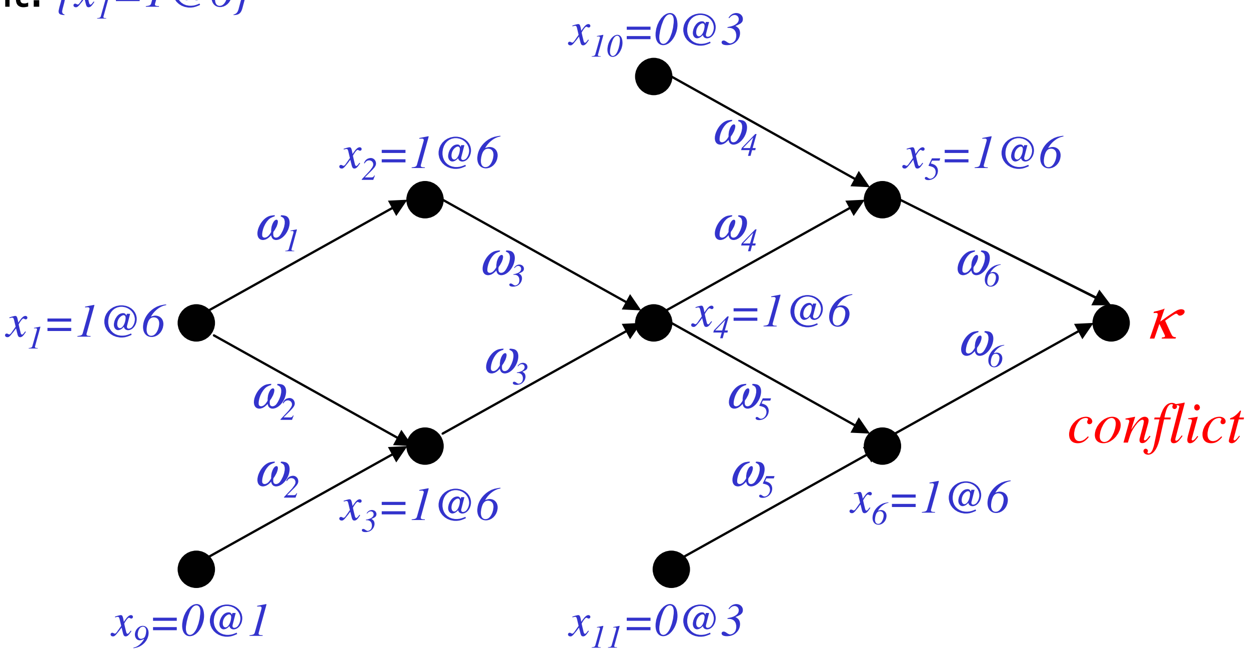
$\omega_5 = (\neg x_4 \vee x_6 \vee x_{11})$

$\omega_6 = (\neg x_5 \vee \neg x_6)$

$\omega_7 = (x_1 \vee x_7 \vee \neg x_{12})$

$\omega_8 = (x_1 \vee x_8)$

$\omega_9 = (\neg x_7 \vee \neg x_8 \vee \neg x_{13})$



GRASP Conflict Analysis

- After a conflict arises, analyze the implication graph.
- Add new clause that would prevent the occurrence of the same conflict in the future.
  $\Rightarrow$ Learning
- Determine decision level to backtrack to; this might not be the immediate one.
  $\Rightarrow$ Non-chronological backtrack

Learning Algorithm

1. Let CA be the assignment of False to all literals in the falsified clause. ("CA" is short for "conflict assignment".)
   - Example: $CA = \{x_5 = 1@6,\ x_6 = 1@6\}$
2. A literal $l \in$ CA is a unique implication point (UIP) iff every other literal in CA has an earlier decision level than l.
3. **loop**:
   - Remove the most recently assigned literal from CA and replace it by its antecedent.
   - **if** (CA is empty or has a UIP): **break**;
4. Let $\{L_1, ..., L_n\}$ = CA; learn clause $(\neg L_1 \vee ... \vee \neg L_n)$.
5. Backtrack to the earliest decision level at which the learned clause will force the UIP to be false.
   - Why is this guaranteed to be possible?

Example Implication Graph

Current truth assignment: $\{x_9=0@1, x_{12}=1@2, x_{13}=1@2, x_{10}=0@3, x_{11}=0@3\}$

Current decision assignment: $\{x_1=1@6\}$

$\omega_1 = (\neg x_1 \vee x_2)$

$\omega_2 = (\neg x_1 \vee x_3 \vee x_9)$

$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$

$\omega_4 = (\neg x_4 \vee x_5 \vee x_{10})$

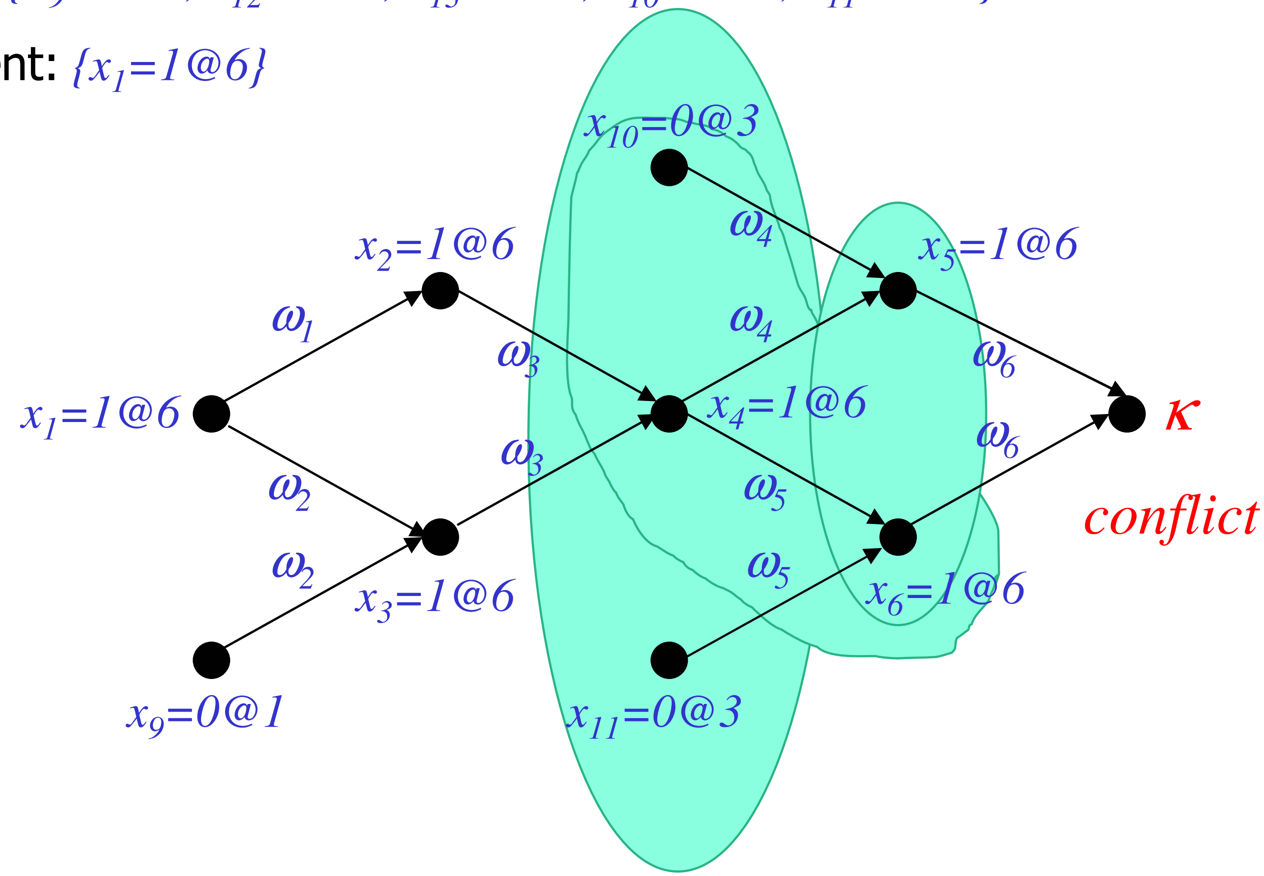
$\omega_5 = (\neg x_4 \vee x_6 \vee x_{11})$

$\omega_6 = (\neg x_5 \vee \neg x_6)$

$\omega_7 = (x_1 \vee x_7 \vee \neg x_{12})$

$\omega_8 = (x_1 \vee x_8)$

$\omega_9 = (\neg x_7 \vee \neg x_8 \vee \neg x_{13})$



Example

$$\omega_1 = (\neg x_1 \vee x_8 \vee x_9)$$

$$\omega_2 = (\neg x_1 \vee x_8 \vee \neg x_9)$$

$$\omega_3 = (\neg x_1 \vee \neg x_8 \vee x_9)$$

$$\omega_4 = (\neg x_1 \vee \neg x_8 \vee \neg x_9)$$

$$\omega_5 = (x_1 \vee x_3)$$

$$\omega_6 = (x_1 \vee \neg x_3)$$

Is that all?

- Huge overhead for boolean constraint propagation (BCP)
- Better decision heuristics
- Better learning, problem specific
- **Better engineering!**

Chaff