Artificial Intelligence

Blai Bonet

Universidad Simón Bolívar, Caracas, Venezuela

Satisfiability (SAT)

Goals for the lecture

- Motivation for using propositional languages
- Syntax and semantics of propositional logic
- Inference problem and its solvability over restricted classes of formulas
- Solving inference problem for CNF formulas by either pure search, pure inference, or combination of search with limited forms of inference

Propositional logic

In many applications knowledge can be expressed with simple formulas in propositional logic

Answering queries about the system or making decisions can be cast as inference problems over propositional formulas

We present results and algorithms for making such inferences

(Following slides based on material from A. Darwiche)

Syntax of propositional logic

Logical formulas build from propositional symbols (atoms) belonging to a finite set $\mathcal{P}$ of propositions in a recursive manner:

– p is a formula (called atom) for every propositional symbol p

– if φ is a formula, then $\neg\varphi$ is also a formula

– if φ and ψ are formulas, then $\varphi \wedge \psi$, $\varphi \vee \psi$ and $\varphi \rightarrow \psi$ are also formulas

– if φ is a formula, then (φ) is also a formula

Example: $p \vee (q \wedge \neg r) \rightarrow \neg p \vee \neg q$

A **positive literal** is atom (e.g. p) while a **negative literal** is the negation of atom (e.g. $\neg q$)

Propositional valuations

For defining the semantics (meaning) of formulas, we consider **valuations** (aka models) over symbols in $\mathcal{P}$

A **valuation** ν is a function that maps symbols in $\mathcal{P}$ to truth values denoted by $\{0, 1\}$

If $\nu(p) = 1$ (resp. $\nu(p) = 0$), we say that ν makes p **true** (resp. **false**)

Semantics of propositional logic

Let φ be a formula over $\mathcal{P}$ and $\nu : \mathcal{P} \rightarrow \{0, 1\}$ be a valuation for $\mathcal{P}$

We define when ν makes φ true, written $\nu \vDash \varphi$, inductively:

– $\nu \vDash p$ iff $\nu(p) = 1$

– $\nu \vDash \neg\varphi$ iff $\nu \nvDash \varphi$

– $\nu \vDash \psi \wedge \varphi$ iff $\nu \vDash \psi$ and $\nu \vDash \varphi$

– $\nu \vDash \psi \vee \varphi$ iff $\nu \vDash \psi$ or $\nu \vDash \varphi$

– $\nu \vDash \psi \rightarrow \varphi$ iff $\nu \nvDash \psi$ or $\nu \vDash \varphi$

A formula φ over $\mathcal{P}$ is **valid** iff $\nu \vDash \varphi$ for **every** valuation ν over $\mathcal{P}$, while φ is **satisfiable** iff $\nu \vDash \varphi$ for **at least one** valuation ν over $\mathcal{P}$

Logical consequence (entailment)

Given two formulas φ and ψ over $\mathcal{P}$, we say that ψ is a **logical consequence** of φ (or that φ **entails** ψ, or ψ is entailed by φ) iff

$$\nu \vDash \varphi \quad \Longrightarrow \quad \nu \vDash \psi$$

for every valuation ν over $\mathcal{P}$

If φ entails ψ, we write $\varphi \vDash \psi$

Basic inference problem

Given two formulas φ and ψ over a propositional language $\mathcal{P}$, the **basic inference problem** is to determine whether φ entails ψ (i.e. check whether $\varphi \vDash \psi$)

Fundamental result: $\varphi \vDash \psi$ iff $\varphi \wedge \neg\psi$ is **unsatisfiable**

Proof: ($\Rightarrow$) $\varphi \vDash \psi$ iff for every valuation ν that makes φ true, then ν makes ψ true as well. Therefore, there is no valuation ν such that $\nu \vDash \varphi \wedge \neg\psi$ (i.e. $\varphi \wedge \neg\psi$ is unsatisfiable)

($\Leftarrow$) $\varphi \wedge \neg\psi$ is unsatisfiable iff there is no valuation ν that makes $\varphi \wedge \neg\psi$ true. Therefore, for every valuation ν over $\mathcal{P}$, if ν makes φ true, then it must make ψ true as well

Representing knowledge as propositional formulas

Given knowledge base (logical formula), queries (questions) on the knowledge base often correspond to basic inference problems

The inference problem can be solved using a **satisfiability solver** for propositional logic

Other queries may be more difficult to answer and require more elaborated solvers (e.g. model counting, enumeration of models, etc.)

General satisfiability problem

The satisfiability problem (SAT) is:

Given formula φ over propositional language $\mathcal{P}$, determine whether φ is satisfiable

The satisfiability problem is **NP-hard**: there is no known efficient algorithm for it (and we believe such algorithm doesn't exist)

SAT is a **fundamental problem** in CS and central to complexity theory

Propositional languages

We can restrict the SAT problem by restricting the **form** of the formulas φ considered

Important cases:

– **Conjunctive normal form (CNF):** formula φ is in CNF iff it is a conjunction of disjunctions of literals (a disjunction of literals is called **clause**)

  Example: $(p \vee \neg q) \wedge r \wedge (\neg p \vee q \vee \neg r)$

– **Disjunctive normal form (DNF):** formula φ is in CNF iff it is a disjunction of conjunctions of literals (a conjunction of literals is called **term**)

  Example: $(p \wedge \neg q \wedge r) \vee (p \wedge \neg r) \vee \neg p$

Propositional languages

CNF/DNF are **universal languages** meaning that for every formula φ, there is formula ψ in CNF (resp. DNF) that is **equivalent** to φ: for every valuation $\nu : \mathcal{P} \to \{0,1\}$, $\nu \vDash \varphi$ iff $\nu \vDash \psi$

Further restrictions:

– k-CNF: formula φ is in k-CNF iff it is in CNF and very clause in φ has **exactly k literals**; k-CNF is **universal** for $k \geq 3$

– k-DNF: formula φ is in k-DNF iff it is in DNF and very term in φ has **exactly k literals**

– Horn theory: formula φ is a Horn theory iff it is a conjunction of **Horn clauses**, where a Horn clause is a clause with at **most one positive literal**

– ...

Satisfiability over restricted languages

– 1-CNF and 2-CNF: solvable in polynomial time

– k-CNF for $k \geq 3$: NP-hard

– DNF: solvable in polynomial time

– Horn theories: solvable in polynomial time

Satisfiability for 1-CNF

Let $\varphi = \ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n$ be a formula in 1-CNF where each ℓ_i is a literal (either atom or negation of atom)

φ is SAT iff there is no i and j such that $\ell_i \equiv \neg\ell_j$ (i.e. $\ell_i = p$ and $\ell_j = \neg p$ for some proposition p)

Observe that 1-CNF formulas correspond to terms and thus to DNF formulas with a single term

Satisfiability for DNF

Let $\varphi = t_1 \vee t_2 \vee \cdots \vee t_n$ be formula in DNF where each t_i is a term (conjunction of literals)

φ is SAT iff there is term t_i containing no complemented literals (i.e. term t_i does not contain p and $\neg p$ for some proposition p)

Indeed, if $t_i = \ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_m$ is such a term, define the valuation ν as follows

$$\nu(p) \;=\; \begin{cases} 1 & \text{if } \ell_i = p \text{ for some } 1 \leq i \leq m \\ 0 & \text{otherwise} \end{cases}$$

It is not hard to check that $\nu \vDash t_i$ and thus $\nu \vDash \varphi$

Conversely, if $\nu \vDash \varphi$, then $\nu \vDash t_i$ for some $1 \leq i \leq n$ and thus t_i cannot contain two complemented literals

Satisfiability for 2-CNF

A 2-CNF formula φ is a CNF formula made of clauses $\ell \vee \ell'$

Each clause $\ell \vee \ell'$ is equivalent to the implications $\neg\ell \to \ell'$ and $\neg\ell' \to \ell$; in symbols

$$\ell \vee \ell' \;\equiv\; \neg\ell \to \ell' \;\equiv\; \neg\ell' \to \ell$$

Given 2-CNF formula φ over propositions $\mathcal{P}$, we construct the **implication graph** for φ:

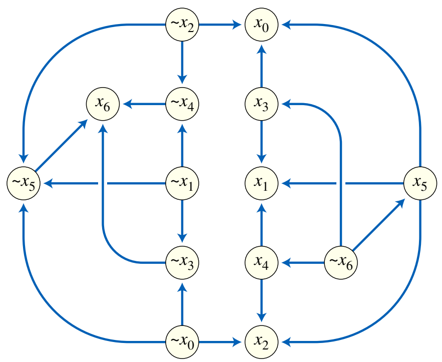
– set of vertices are all the literals over $\mathcal{P}$

– there is edge between $\neg\ell$ and ℓ' iff φ contains the clause $\ell \vee \ell'$

Example of implication graph

Consider the 2-CNF formula φ given by

$$(x_0 \vee x_2) \wedge (x_0 \vee \neg x_3) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (x_2 \vee \neg x_4) \wedge$$
$$(x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$



[Image from Wikipedia]

Solving satisfiability for 2-CNF in linear time

A formula φ in 2-CNF is satisfiable iff no **strongly connected component (SCC)** in its implication graph contains two complemented literals

Observe that $\nu \vDash \varphi$ iff all literals in a SCC receive the same truth value from ν. Therefore,

– If φ is SAT then no SCC contains two complemented literals

– Conversely, if no SCC has two complemented literals, we can construct valuation ν such that $\nu \vDash \varphi$ (by "top-down" traversal of SCC DAG)

The SCCs of a directed graph can be computed in linear time using **Tarjan's algorithm** (alternatively, **Kosaraju's algorithm** [CLRS])

In the example, the implication graph is **acyclic** and thus each vertex appears in its own (singleton) SCC. The formula is thus satisfiable

Satisfiability of Horn theories

A Horn clause is a clause with at most one positive literal, while a formula is a Horn theory if it is a conjunction of Horn clauses

If φ is a Horn theory **without unit clauses** (clauses of size 1), then every clause contains at least one negative literal

Therefore, the valuation $\nu : \mathcal{P} \to \{0,1\}$ that makes false every proposition $p \in \mathcal{P}$ is a valuation that makes φ true

If φ has one or more unit clauses, we run **unit propagation (UP)** (in linear time) to remove all unit clauses and output formula ψ where

– ψ is a Horn theory without unit clauses and thus satisfiable

– φ is satisfiable iff UP doesn't derive a contradiction

Satisfiability of CNF formulas (SAT-CNF)

SAT problem for general CNF is **NP-hard**

We present algorithms for solving SAT that either

– perform pure search

– perform pure inference

– combine search with limited forms of inference

From now on, SAT refers to SAT-CNF

Clausal form for CNF

It is convenient to represent CNF formulas in **clausal form**

CNF formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$ where each C_i is a clause of form

$$\ell_{i1} \ \vee \ \ell_{i2} \ \vee \ \cdots \ \vee \ \ell_{im_i}$$

is represented by the collection of subsets

$$\{\,\{\ell_{11}, \ell_{12}, \ldots, \ell_{1m_1}\},\ \ldots,\ \{\ell_{i1}, \ell_{i2}, \ldots, \ell_{1m_i}\},\ \ldots\}$$

For example,

$$\begin{aligned}\Delta \ &= (A \vee B) \wedge (B \vee C) \wedge (\neg A \vee \neg X \vee Y) \wedge (\neg A \vee X \vee Z) \wedge \\ &\quad (\neg A \vee \neg Y \vee Z) \wedge (\neg A \vee X \vee \neg Z) \wedge (\neg A \vee \neg Y \vee \neg Z)\end{aligned}$$

is represented as the collection

$$\begin{aligned}\Delta \ &= \ \{\,\{A, B\}, \{B, C\}, \{\neg A, \neg X, Y\}, \{\neg A, X, Z\}, \{\neg A, \neg Y, Z\}, \\ &\quad \{\neg A, X, \neg Z\}, \{\neg A, \neg Y, \neg Z\}\,\}\end{aligned}$$

Conditioning

Let φ be a formula over propositions $\mathcal{P}$

$\varphi|X$ denotes the formula φ where each **occurrence** of X is replaced by **true** and the formula is simplified accordingly

For example,

$$\Delta|X = (A \vee B) \wedge (B \vee C) \wedge (\neg A \vee Y) \wedge (\neg A \vee \neg Y \vee Z) \wedge (\neg A \vee \neg Y \vee \neg Z)$$

Likewise, $\varphi|\neg X$ denotes the formula φ where each **occurrence** of X is replaced by **false** and the formula is simplified accordingly

For example,

$$\Delta|\neg Z = (A \vee B) \wedge (B \vee C) \wedge (\neg A \vee \neg X \vee Y) \wedge (\neg A \vee X) \wedge (\neg A \vee \neg Y)$$

Conditioning in clausal form

If φ is in clausal form, conditioning can be implemented easily:

– $\varphi|X = \{S \setminus \{\neg X\} : S \in \varphi, \neg X \in S\} \cup \{S \in \varphi : \{X, \neg X\} \cap S = \emptyset\}$

– $\varphi|\neg X = \{S \setminus \{X\} : S \in \varphi, X \in S\} \cup \{S \in \varphi : \{X, \neg X\} \cap S = \emptyset\}$

For example, for Δ given by

$$\begin{aligned}\Delta \ &= \ \{\,\{A, B\}, \{B, C\}, \{\neg A, \neg X, Y\}, \{\neg A, X, Z\}, \{\neg A, \neg Y, Z\}, \\ &\quad \{\neg A, X, \neg Z\}, \{\neg A, \neg Y, \neg Z\}\,\}\end{aligned}$$

$$\Delta|X \ = \ \{\,\{A, B\}, \{B, C\}, \{\neg A, Y\}, \{\neg A, \neg Y, Z\}, \{\neg A, \neg Y, \neg Z\}\,\}$$

$$\Delta|\neg Z \ = \ \{\,\{A, B\}, \{B, C\}, \{\neg A, \neg X, Y\}, \{\neg A, X\}, \{\neg A, \neg Y\}\,\}$$

Simple backtracking

Let Δ be a CNF formula in clausal form over $\mathcal{P} = \{X_1, X_2, \ldots, X_n\}$

```
SAT-I(theory Δ, int i)
    if i == n + 1
        if Δ == ∅
            return ∅
        else
            return FAIL

    M := SAT-I(Δ|X_i, i + 1)
    if M != FAIL then return M ∪ { X_i }

    M := SAT-I(Δ|¬X_i, i + 1)
    if M != FAIL then return M ∪ { ¬X_i }

    return FAIL
```


Simple backtracking with early detection

Let Δ be a CNF formula in clausal form over $\mathcal{P} = \{X_1, X_2, \ldots, X_n\}$

```
SAT-II(theory Δ, int i)
    if Δ == ∅ then return ∅
    if Δ contains ∅ then return FAIL

    M := SAT-II(Δ|X_i, i + 1)
    if M != FAIL then return M ∪ { X_i }

    M := SAT-II(Δ|¬X_i, i + 1)
    if M != FAIL then return M ∪ { ¬X_i }

    return FAIL
```


Simple backtracking with literal selection

Let Δ be a CNF formula in clausal form over $\mathcal{P} = \{X_1, X_2, \ldots, X_n\}$

```
SAT-III(theory Δ)
    if Δ == ∅ then return ∅
    if Δ contains ∅ then return FAIL

    L := choose literal in Δ

    M := SAT-III(Δ|L)
    if M != FAIL then return M ∪ { L }

    M := SAT-III(Δ|∼L)
    if M != FAIL then return M ∪ { ¬L }

    return FAIL
```


Unit propagation

Limited but efficient form of inference that reasons with **unit clauses**

If Δ contains **unit clause** $\{\ell\}$, then $\Delta \equiv \ell \wedge \Delta'$ where $\Delta' = \Delta|\ell$

Reduction can be applied recursively until Δ' contains **no unit clause**

We obtain the equivalence $\Delta \equiv \ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_k \wedge \Delta'$ where $\Delta' = \Delta|\ell_1, \ell_2, \ldots, \ell_k$ contains no unit clause

Pair $(\{\ell_1, \ell_2, \ldots, \ell_k\}, \Delta')$ is result of **unit propagation (UP)** over Δ (there is a unique such pair (result) of UP)

UP can be implemented in **linear time** with the right data structures

Example of unit propagation

For Δ given by

$$\begin{aligned}\Delta = \{ & \{A,B\},\{B,C\},\{\neg A,\neg X,Y\},\{\neg A,X,Z\},\{\neg A,\neg Y,Z\}, \\ & \{\neg A,X,\neg Z\},\{\neg A,\neg Y,\neg Z\}\}\end{aligned}$$

$$\begin{aligned}
(\emptyset,\Delta) &= \text{Unit-Propagation}(\Delta) \\
(\{\neg A,B\},\emptyset) &= \text{Unit-Propagation}(\Delta\cup\{\{\neg A\}\}) \\
(\{A\},\Gamma) &= \text{Unit-Propagation}(\Delta\cup\{\{A\}\}) \text{ where} \\
\Gamma &= \{\{B,C\},\{\neg X,Y\},\{X,Z\},\{\neg Y,Z\},\{X,\neg Z\},\{\neg Y,\neg Z\}\} \\
(I,\emptyset) &= \text{Unit-Propagation}(\Delta\cup\{\{\neg X\},\{\neg Z\}\}) \text{ where} \\
I &= \{\neg A,B,\neg X,\neg Z\}
\end{aligned}$$

DPLL algorithm: Search + inference

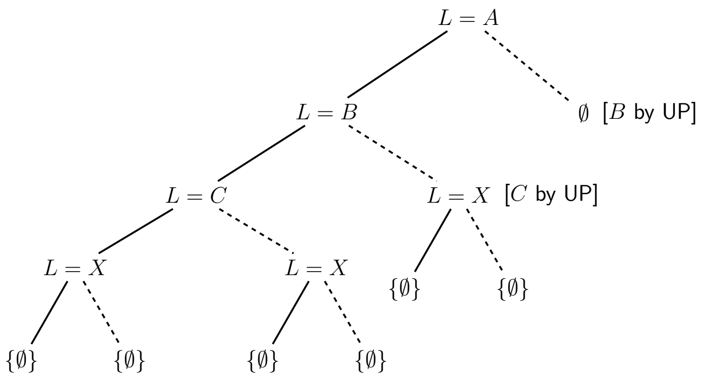
Let Δ be a CNF formula in clausal form over $\mathcal{P}=\{X_1,X_2,\ldots,X_n\}$

The following implements **DPLL (Davis, Putnam, Longemann, Loveland, 1962)** algorithm without conditioning by exploiting UP

```
DPLL(theory Δ)
    (I,Γ) := Unit-Propagation(Δ)
    if Γ == ∅ then return I
    if Γ contains ∅ then return FAIL

    L := choose literal in Δ

    M := DPLL(Γ ∪ {{L}})
    if M != FAIL then return M ∪ I

    M := DPLL(Γ ∪ {{∼L}})
    if M != FAIL then return M ∪ I

    return FAIL
```


Example of DPLL

$$\begin{aligned}\Delta = \{ & \{A,B\},\{B,C\},\{\neg A,\neg X,Y\},\{\neg A,X,Z\},\{\neg A,\neg Y,Z\}, \\ & \{\neg A,X,\neg Z\},\{\neg A,\neg Y,\neg Z\}\}\end{aligned}$$



Motivation for implication graph

Formula $\Delta|A,B,C,X$ is false because the form $\{\{\neg X,Y\},\{X,Z\},\{\neg Y,Z\},\{X,\neg Z\},\{\neg Y,\neg Z\}\}$ is unsatisfiable

DPLL doesn't see this fact and must **exhaust subtree** below the assignment $A=false$

Implication graphs are used to find **causes of failure** after the search below a node fails

Implication graph for DPLL

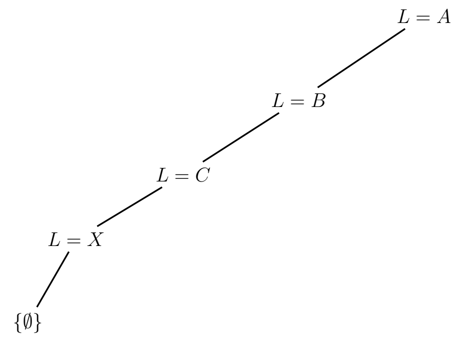
The implication graph has nodes of the form d/ℓ, where d is an integer, that means literal ℓ is set to true at **decision level** d

Assignment ℓ is by either a **decision** or **implication** obtained by UP

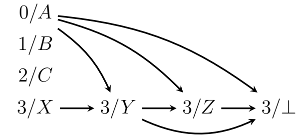
When clause $\{\ell_1, \ldots, \ell_m\}$ becomes unit $\{\ell_m\}$, we add edges $d_i/\ell_i \to d/\ell_m$ to implication graph (where d is current decision level)

When UP derives a contradiction at level d (i.e. null clause), the node $d/\bot$ is added to the graph together with edges $d_i/\ell_i \to d/\bot$ for the literals ℓ_i that belong to **original clause**

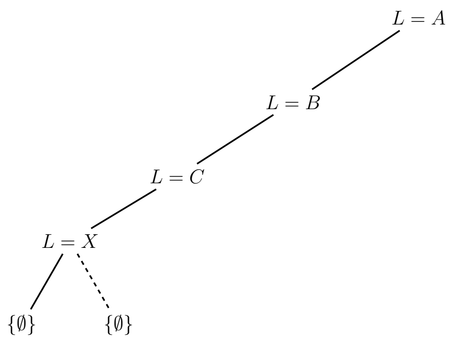
Example of implication graphs



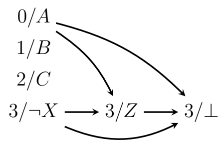
$$\Delta|A,B,C,X \;=\; \{\{Y\},\{\neg Y, Z\},\{\neg Y,\neg Z\}\} \overset{\text{UP}}{\Longrightarrow} \{\emptyset\}$$



Example of implication graphs



$$\Delta|A,B,C,\neg X \;=\; \{\{Z\},\{\neg Y, Z\},\{\neg Z\},\{\neg Y,\neg Z\}\} \overset{\text{UP}}{\Longrightarrow} \{\emptyset\}$$



Analyzing reasons for failure

Every **cut** in the implication graph that **separates** the decisions from the contradiction makes up a **conflict set**

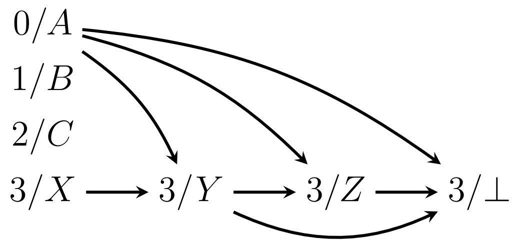
Conflict sets are used to:

– analyze **reasons for failure**

– compute **backtrack level**

– obtain clasues to **learn**

Example of conflict sets



A cut that separates decisions from conflict makes up a conflict set

$$\{0/A, 1/B, 2/C, 3/X, 3/Y, 3/Z\},\ \{3/\bot\} \implies \{0/A, 3/Z, 3/Y\} \mapsto A \wedge Z \wedge Y$$

$$\{0/A, 1/B, 2/C, 3/X, 3/Y\},\ \{3/Z, 3/\bot\} \implies \{0/A, 3/Y\} \mapsto A \wedge Y$$

$$\{0/A, 1/B, 2/C, 3/X\},\ \{3/Y, 3/Z, 3/\bot\} \implies \{0/A, 3/X\} \mapsto A \wedge X$$

Unit implication point

Which conflict set to use?

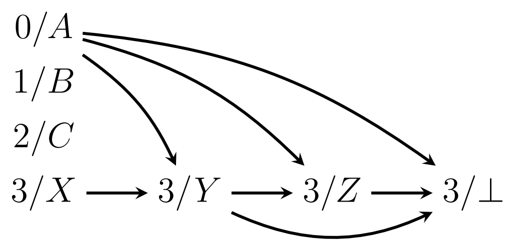
Choose cut where all nodes at current decision level, except the decision node at current level, are on one side of the cut, and all other nodes are on the other side

Formally, we define $C(n)$ for node n in implication graph:

$$C(n) = \begin{cases} \{n\} & \text{if } Pa(n) = ePa(n) = \emptyset \\ ePa(n) \cup \bigcup_{n' \in Pa(n)} C(n') & \text{otherwise} \end{cases}$$

where $Pa(n)$ are the parent of node n that are at same level as n, and $ePa(n)$ are the parents of n at earlier levels

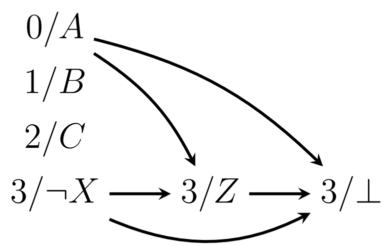
Example UIP conflict set



$$C(n) = \begin{cases} \{n\} & \text{if } Pa(n) = ePa(n) = \emptyset \\ ePa(n) \cup \bigcup_{n' \in Pa(n)} C(n') & \text{otherwise} \end{cases}$$

$$\begin{aligned} C(3/\bot) &= ePa(3/\bot) \cup C(3/Z) \cup C(3/Y) \\ &= \{0/A\} \cup [ePa(3/Z) \cup C(3/Y)] \cup [ePa(3/Y) \cup C(3/X)] \\ &= \{0/A\} \cup [\{0/A\} \cup ePa(3/Y) \cup C(3/X)] \cup [\{0/A\} \cup \{3/X\}] \\ &= \{0/A, 3/X\} \end{aligned}$$

Example UIP conflict set



$$C(n) = \begin{cases} \{n\} & \text{if } Pa(n) = ePa(n) = \emptyset \\ ePa(n) \cup \bigcup_{n' \in Pa(n)} C(n') & \text{otherwise} \end{cases}$$

$$\begin{aligned} C(3/\bot) &= ePa(3/\bot) \cup C(3/Z) \cup C(3/\neg X) \\ &= \{0/A\} \cup [ePa(3/Z) \cup C(3/\neg X)] \cup \{3/\neg X\} \\ &= \{0/A\} \cup [\{0/A\} \cup \{3/\neg X\}] \cup \{3/\neg X\} \\ &= \{0/A, 3/\neg X\} \end{aligned}$$

From conflict sets to implied clauses

A conflict set corresponds to a **term** t that is inconsistent with the theory Δ; i.e. $\Delta \wedge t$ in UNSAT

Therefore, by fundamental result about inference, $\Delta \vDash \neg t$

Since $\neg t$ is a clause, clause $\neg t$ is **implied by** Δ and $\Delta \wedge \neg t \equiv \Delta$

In example, the conflict set $\{0/A, 3/X\}$ corresponds to clause $\{\neg A, \neg X\}$ while $\{0/A, 3/\neg X\}$ corresponds to clause $\{\neg A, X\}$

[Remark: both clauses implies the unit clause $\{\neg A\}$ which says A must be false in every model of Δ]

From conflict sets to backtrack level

Conflict set also defines backtrack level. For conflict set C, define:

– **Backtrack level (bl)** is highest decision level of any literal in C

– **Assertion level (al)** is second highest decision level of any literal in C (-1 if C is singleton conflict set)

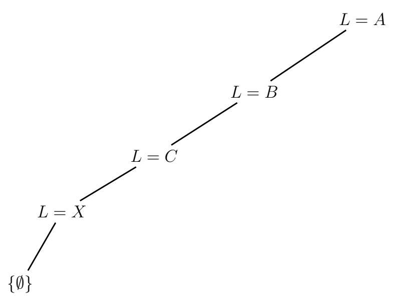
For conflict set $C = \{0/A, 3/X\}$, $bl = 3$ and $al = 0$

Backtracking to level bl corresponds to **chronological bracktracking** since bl is always equal to current decision level

Search can be improved by backtracking to level $al + 1$, undoing all decisions at levels $al + 1, al + 2, \ldots$, but **adding** (learning) the clause that corresponds to the conflict set C

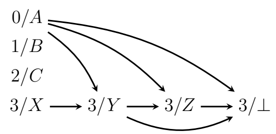
This type of backtrack is not the regular backtrack as the search starts again at level $al + 1$ by running UP on the extended theory and then choosing a new literal to branch upon

Conflict-driven clause learning (CDCL)



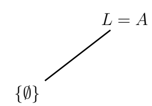
$$\Delta = \{\{A, B\}, \{B, C\}, \{\neg A, \neg X, Y\}, \{\neg A, X, Z\}, \{\neg A, \neg Y, Z\}, \{\neg A, X, \neg Z\}, \{\neg A, \neg Y, \neg Z\}\}$$

$$\Delta|A, B, C, X = \{\{Y\}, \{\neg Y, Z\}, \{\neg Y, \neg Z\}\} \stackrel{\text{UP}}{\Longrightarrow} (\{A, B, C, X, Y, Z, \neg Z\}, \{\emptyset\})$$



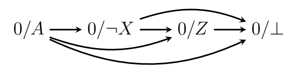
$$C(3/\bot) = \{0/A, 3/X\}, \quad cl = \{\neg A, \neg X\}, \quad al = 0$$

Conflict-driven clause learning (CDCL)



$$\Delta' = \{\{A, B\}, \{B, C\}, \{\neg A, \neg X, Y\}, \{\neg A, X, Z\}, \{\neg A, \neg Y, Z\}, \{\neg A, X, \neg Z\}, \{\neg A, \neg Y, \neg Z\}, \{\neg A, \neg X\}\}$$

$$\Delta'|A = \{\{\neg X\}, \{B, C\}, \{\neg X, Y\}, \{X, Z\}, \{\neg Y, Z\}, \{X, \neg Z\}, \{\neg Y, \neg Z\}\} \stackrel{\text{UP}}{\Longrightarrow} (\{A, \neg X, Z, \neg Z\}, \{\emptyset\})$$



$$C(0/\bot) = \{0/A\}, \quad cl = \{\neg A\}, \quad al = -1$$

Conflict-driven clause learning (CDCL)

$\emptyset$

$\Delta'' = \{\{A,B\},\{B,C\},\{\neg A,\neg X,Y\},\{\neg A,X,Z\},\{\neg A,\neg Y,Z\},\{\neg A,X,\neg Z\},\{\neg A,\neg Y,\neg Z\},\{\neg A,\neg X\},\{\neg A\}\}$

$\Delta'' \overset{\text{UP}}{\Longrightarrow} (\{\neg A,\neg B\},\emptyset)$

$-1/\neg A \longrightarrow -1/B$ [decision level -1 indicates all inferences done by UP from new unit clause]

Random re-starts

Each learned clause prunes at least one branch from search tree, the current branch, but may prune more branches

If a clause is learned each time a conflict is reached, the set of new clauses prune the set of visited branches

Therefore, if search **re-starts from scratch** with new theory, it skips the already visited branches and continue a complete exploration over remaining search tree

Random re-starts is a technique in which the search algorithm is re-started from scratch from time to time in order to **diversify** the search without **loosing completeness** and without getting **trapped in infinite loop**

Random re-starts are **key component of modern SAT solvers**

Implementing UP

Modern SAT solvers maintain **global** partial assignment and implication graph that are updated as search makes decisions and performs backtracks

Every time a decision L is performed (i.e. setting L to true), UP is run to see whether other literals are implied or conflict is detected

UP accounts for ~90% or more of running time

A good implementation of UP is **crucial** for a effective SAT solver

General scheme for UP

```
1  Propagate(L)
2
3      C := clauses where ~L appears
4
5      foreach clause c in C
6          if c is unit clause              % given current assignment
7              L' := unique literal in c    % one with no value assigned
8
9              if Propagate(L') == false
10                 return false             % conflict detected
11
12         else if c is violated
13             return false                 % conflict detected
14
15     return true
```

Bottlenecks: lines 3, 6, 7, and 12

Using indices and counters

For each literal L: keep list of clauses that contain L (lists must be updated when new clauses are added)

For each clause c: counters for clause size, #positive literals in c, and #negative literals in c denoted by $S[c]$, $P[c]$, and $N[c]$ respectively

- Initially, $P[c] = N[c] = 0$ for each clause c
- Clause c is **satisfied** when $P[c] > 0$
- Clause c is **violated** when $N[c] = S[c]$
- Clause c is **unit** when $N[c] = S[c] - 1$

Counters are **updated** when literals receive value, when performing backtracks, and when performing re-starts

UP with counters

```
1  Propagate(L)
2      CP := clauses where L appears
3      CN := clauses where ∼L appears
4      no-conflict := true
5
6      foreach clause c in CN
7          N[c] := N[c] + 1                       % inc #lits false
8          if P[c] == 0 and N[c] == S[c] - 1      % c is unit
9              L' := unique literal in c          % one with no value
10             if Propagate(L') == false
11                 no-conflict := false
12
13         else if P[c] == 0 and N[c] == S[c]     % c is violated
14             no-conflict := false
15
16     foreach clause c in CP
17         P[c] := P[c] + 1                       % inc #lits true
18
19     return no-conflict
```

Why don't we directly return in lines 11 and 14?

Ideal UP algorithm and approximation

Ideal algorithm:

- Inspect a clause only when all literals except one are assigned false
- Nothing to do when clause is satisfied or non-unit

Best known approximation to ideal (introduced in zChaff in 2001):

- Associate each clause to two of its own unassigned literals
- Inspect clause when one of the two literals is assigned false

Lazy data structure: Watched literals

- **For non-satisfied clause c:** "watch" two non-false literals in c
- **For literal L:** keep list of clauses where L is watched

Maintain invariant:

- If watched literal L becomes false, find another to watch
- If there is no other unassigned literal, the clause is unit

Advantages:

- Visit fewer clauses when literal is assigned
- No need to do anything when backtrack or re-starts!

Example: Watched literals

t/f	t/f	t/f	t/f
$\neg X$	Y	W	$\neg Z$

Clause: $\neg X \vee Y \vee W \vee \neg Z$

Assignment:

Example: Watched literals

f	t/f	t/f	t/f
$\neg X$	Y	W	$\neg Z$

Clause: $\neg X \vee Y \vee W \vee \neg Z$

Assignment: $X = true$

Example: Watched literals

t/f	t/f	t/f	t/f
$\neg X$	Y	W	$\neg Z$

Clause: $\neg X \vee Y \vee W \vee \neg Z$

Assignment: (backtrack)

Example: Watched literals

t/f	t	t/f	t/f
$\neg X$	Y	W	$\neg Z$

Clause: $\neg X \vee Y \vee W \vee \neg Z$

Assignment: $Y = true$

Example: Watched literals

t/f	t/f	t/f	t/f
$\neg X$	Y	W	$\neg Z$
	$\uparrow$	$\uparrow$	

Clause: $\neg X \vee Y \vee W \vee \neg Z$

Assignment: (backtrack)

Example: Watched literals

t/f	t/f	t/f	f
$\neg X$	Y	W	$\neg Z$
	$\uparrow$	$\uparrow$	

Clause: $\neg X \vee Y \vee W \vee \neg Z$

Assignment: $Z = true$

Example: Watched literals

t/f	f	t/f	f
$\neg X$	Y	W	$\neg Z$
$\uparrow$		$\uparrow$	

Clause: $\neg X \vee Y \vee W \vee \neg Z$

Assignment: $Z = true$, $Y = false$

Example: Watched literals

t/f	f	f	f
$\neg X$	Y	W	$\neg Z$
$\uparrow$		$\uparrow$	

Clause: $\neg X \vee Y \vee W \vee \neg Z$

Assignment: $Z = true$, $Y = false$, $W = false$

Unit!

Example: Watched literals

t/f	t/f	t/f	t/f
$\neg X$	Y	W	$\neg Z$

Clause: $\neg X \vee Y \vee W \vee \neg Z$

Assignment: (re-start)

UP with watched literals

```
Propagate(L)
    WC := clauses where ~L is watched

    foreach clause c in WC
        w1 := watched literal for ~L in c
        w2 := another watched literal in c
        Replace w1 with another unassigned literal in c

        if w1 cannot be replaced and w2 is unknown
            if Propagate(w2) == false
                return false

        else if w1 cannot be replaced and w2 is false
            return false

    return true
```


Literal selection: VSIDS heuristic

Most used heuristic for literal selection during search is VSIDS or *variable state independent decaying sum*

- Keep counters for each literal
- Counter for L initialized to number of clauses containing L
- Variable with **highest combined count** is chosen with value given by **highest count**
- When clause is added, increment counter of each literal in clause
- From time to time, all counters are halved (decaying)
- Variables are ordered using heap
- "Cheap" to implement (accounts for small % of running time)

Solving SAT by pure inference

SAT problems can be solved by pure deductive methods

We use **resolution** as unique rule of inference for CNF theories

We will see:

- Resolution is a **correct** rule of inference
- Resolution is a **complete** rule of inference with respect to **refutation**

Propositional resolution

Let C and C' be clauses with complemented literals L and $\sim L$; i.e.

$$C = L \vee \ell_1 \vee \ell_2 \vee \cdots \vee \ell_n$$
$$C' = \sim L \vee \ell'_1 \vee \ell'_2 \vee \cdots \vee \ell'_{n'}$$

Let ν be valuation such that $\nu \vDash C \wedge C'$. There are two mutually exclusive cases whether $\nu \vDash L$ or $\nu \nvDash L$:

- if $\nu \vDash L$, then $\nu \vDash \ell'_1 \vee \ell'_2 \vee \cdots \vee \ell'_{n'}$
- if $\nu \nvDash L$, then $\nu \vDash \ell_1 \vee \ell_2 \vee \cdots \vee \ell_n$

Then, $\nu \vDash \ell'_1 \vee \cdots \vee \ell'_{n'} \vee \ell_1 \vee \cdots \vee \ell_n$. In clausal form, $\nu \vDash (C \cup C') \setminus \{L, \sim L\}$

Rule of inference for resolution infers clause $(C \cup C') \setminus \{L, \sim L\}$ from clauses C and C'. Inferred clause is called **resolution of C and C' upon literal L**, denoted by $Res_L(C, C')$

Resolution closure

Let Δ be a CNF theory

Δ is **resolution free** iff for any two clauses C and C' in Δ containing a complemented literal L, there is clause $C'' \in \Delta$ such that $C'' \subseteq Res_L(C, C')$ (i.e. no new knowledge is obtained by resolution)

For any CNF Δ, there is CNF Δ' such that $\Delta \equiv \Delta'$ and Δ' is resolution free: Δ' is called a **resolution closure** for Δ

A resolution closure for Δ can be computed iteratively by applying resolution until no new clause is generated (i.e. **fix point computation**)

Resolution is refutation complete

Let Δ be CNF theory and Δ' be resolution closure for Δ. Then, Δ is UNSAT iff Δ' contains the empty clause

Proof: ($\Leftarrow$) if $\emptyset \in \Delta'$, $\Delta \vDash \emptyset$ since $\Delta \equiv \Delta'$. As no valuation makes the empty clause true, there is no valuation for Δ and Δ is UNSAT

($\Rightarrow$) For forward direction, we show **contrapositive of the implication:** if Δ' doesn't contain the empty clause, then Δ is SAT

Assume $\emptyset \notin \Delta'$. We construct valuation ν for Δ iteratively:

- Choose literal for X_1 consistent with Δ'. It can be done since Δ' doesn't contain both $\{X_1\}$ and $\{\neg X_1\}$ (otherwise $\emptyset \in \Delta'$)
- After choosing literals for $X_1, \ldots, X_{i-1}$ consistent with Δ', choose literal for X_i consistent with Δ' and chosen literals $\ell_1, \ldots, \ell_{i-1}$ (i.e. violating no clause)

  Indeed, if $\ell_1 \wedge \cdots \wedge \ell_{i-1} \wedge X_i$ violates clause C, and $\ell_1 \wedge \cdots \wedge \ell_{i-1} \wedge \neg X_i$ violates clause C', then $\ell_1 \wedge \cdots \wedge \ell_{i-1}$ violates $Res_{X_i}(C, C')$ which is in Δ' (contradiction). Therefore, either X_i or $\neg X_i$ (or both) is consistent with current valuation and Δ'

Solving 2-CNF in polynomial time using resolution

Let Δ be a 2-CNF theory. We can test for satisfiability by computing a resolution closure Δ' of Δ and checking whether $\emptyset \in \Delta'$

We show that such closure can be computed in polynomial time for 2-CNF

If $\{L, \ell\}$ and $\{\neg L, \ell'\}$ are two clauses in Δ containing complemented literal L, then $Res_L(C, C') = \{\ell, \ell'\}$ is again a clause of size ≤ 2

By applying resolution iteratively over Δ, no clause of size > 2 is generated. The number of clauses of size ≤ 2 over n variables is $2n + 4\binom{n}{2} = O(n^2)$

Therefore, a naive algorithm for computing the closure performs $O(n^2)$ iterations, where each iteration takes time $O(n^4)$ (two nested loops that scan over clauses), for a total running time of $O(n^6)$

Other inference methods for SAT

- All the methods for CSP together with their guarantees apply for SAT as SAT is a special case of CSP
- There are other inference algorithm for SAT whose complexity is exponential in certain measures of **width** for CNF theories

Summary

- SAT is a fundamental problem in AI and CS
- SAT problem is intractable in general but for some special cases SAT is tractable
- CNF is a good representation language for expressing succinctly many interesting problems
- SAT-CNF is intractable in general
- SAT can be solved by either pure search, pure inference, or combination of search with limited forms of inference like UP
- State-of-the-art solvers combine search with unit propagation, non-chronological backtracking via clause learning, and random re-starts