PROPOSITIONAL FORMULAS: THE SAT PROBLEM

VL Logic
Part I: Propositional Logic
Satisfiability Checking

Definition (Satisfiability Problem of Propositional Logic (SAT))

Given a formula $\phi$, is there an assignment $\nu$ such that $[\phi]_\nu = 1$?

- **Oldest NP-complete problem**
  - Checking a solution (assignment satisfies formula) is easy (polynomial effort)
  - Finding a solution is difficult (probably exponential in the worst case)

- **Many practical applications (used in industry)**

- **Efficient SAT solvers (solving tools) are available**

- **Other problems can be translated to SAT**:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Formulation in propositional logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi$ is valid</td>
<td>$\neg \phi$ is unsatisfiable</td>
</tr>
<tr>
<td>$\phi$ is refutable</td>
<td>to $\neg \phi$ is satisfiable</td>
</tr>
<tr>
<td>$\phi \leftrightarrow \psi$</td>
<td>to $\neg (\phi \leftrightarrow \psi)$ is unsatisfiable</td>
</tr>
<tr>
<td>$\phi_1, \ldots, \phi_n \models \psi$</td>
<td>$\phi_1 \land \ldots \land \phi_n \land \neg \psi$ is unsatisfiable</td>
</tr>
</tbody>
</table>
Reasoning with (Propositional) Calculi

- **goal**: automatically reason about (propositional) formulas
  i.e., mechanically show validity / unsatisfiability

- **basic idea**: use syntactical manipulations to prove/refute a formula

- **elements of a calculus**:
  - **axioms**: trivial truths/trivial contradictions
  - **rules**: inference of new formulas

- **approach**: construct a **proof/refutation**
  - apply the rules of the calculus until only axioms are inferred
  - if this is not possible, then the formula is not valid/unsatisfiable

- **examples of calculi**:
  - sequence calculus: shows validity (actually entailment)
  - resolution calculus: shows unsatisfiability
Logic Entailment

Let \( \phi_1, \ldots, \phi_n, \psi \) be propositional formulas. Then \( \phi_1, \ldots, \phi_n \) entail \( \psi \) (written as \( \phi_1, \ldots, \phi_n \models \psi \)) iff 
\[
[\phi_1]_\nu = 1, \ldots [\phi_n]_\nu = 1 \implies [\psi]_\nu = 1.
\]

Informal meaning: True premises derive a true conclusion.

- \( \models \) is a meta-symbol (it is not part of the language)
- \( \phi_1, \ldots, \phi_n \models \psi \) iff \( (\phi_1 \land \ldots \land \phi_n) \rightarrow \psi \) is valid,
  i.e., we can express semantics by means of syntactics.
- If \( \phi_1, \ldots, \phi_n \) do not entail \( \psi \), we write \( \phi_1, \ldots, \phi_n \not\models \psi \).

Example:

- \( a \models a \lor b \)
- \( \models a \lor \neg a \)
- \( a, a \rightarrow b \models b \)
- \( a, b \models a \land b \)
- \( \not\models a \land \neg a \)
- \( \bot \models a \land \neg a \)
Formula Strength

- formula $\phi$ is stronger than formula $\psi$ iff $\phi \models \psi$
- formula $\psi$ is weaker than formula $\phi$ iff $\phi \models \psi$
- formulas $\phi$ and $\psi$ are equally strong iff $\phi \models \psi$ and $\psi \models \phi$

Examples

- $a \oplus b$ is stronger than $a \lor b$
- $a \land b$ is stronger than $a \lor b$
- $\bot$ is the strongest formula
- $\top$ is the weakest formula
PROPOSITIONAL FORMULAS: SEQUENT CALCULUS

VL Logic
Part I: Propositional Logic
Sequents

Definition
A **sequent** is an expression of the form

\[ \phi_1, \ldots, \phi_n \vdash \psi \]

where \( \phi_1, \ldots, \phi_n, \psi \) are propositional formulas.
The formulas \( \phi_1, \ldots, \phi_n \) are called **assumptions**, \( \psi \) is called **goal**.

**remarks:**

- **intuitively** \( \phi_1, \ldots, \phi_n \vdash \psi \) means goal \( \psi \) follows from \{\( \phi_1, \ldots, \phi_n \}\)
- **special case** \( n = 0 \):
  - written as \( \vdash \psi \)
  - meaning: we have to prove that \( \psi \) is valid
- **notation**: for sequent \( \phi_1, \ldots, \phi_n \vdash \psi \), we write \( K \ldots \phi_i \vdash \psi \) if we are only interested in assumption \( \phi_i \)
- the assumptions are **orderless** not ordered
Axioms

- **axiom "goal in assumption":**
  If the goal is among the assumptions, the goal can be proved.

\[
\text{GoalAssum} \quad \frac{}{K \ldots, \psi \vdash \psi}
\]

- **axiom "contradiction in assumptions":**
  If the assumptions are contradicting, anything can be proved.

\[
\text{ContrAssum} \quad \frac{}{K \ldots, \phi, \neg \phi \vdash \psi}
\]
Negation Rules

■ rules "contradiction":

\[
\begin{align*}
A_\neg & \quad \frac{K \ldots, \neg \psi \vdash \phi}{K \ldots, \neg \phi \vdash \psi} \\
P_\neg & \quad \frac{K \ldots, \phi \vdash \bot}{K \ldots \vdash \neg \phi}
\end{align*}
\]

\(A_\neg\): We know \(\neg \phi\) and have to prove \(\psi\). Thus we may assume \(\neg \psi\) and prove \(\phi\).

\(P_\neg\): We have to prove \(\neg \phi\). Thus we may assume \(\phi\) and derive a contradiction.

■ rules "elimination of double negation":

\[
\begin{align*}
P_\neg & \quad \frac{K \ldots \vdash \psi}{K \ldots \vdash \neg \neg \psi} \\
A_\neg & \quad \frac{K \ldots, \phi \vdash \psi}{K \ldots, \neg \neg \phi \vdash \psi}
\end{align*}
\]
Binary Connective Rules

- **rules "conjunction":**

  \[
  \frac{K \ldots, \phi_1, \phi_2 \vDash \psi}{K \ldots, \phi_1 \land \phi_2 \vDash \psi}
  \]

- **rules "disjunction":**

  \[
  \frac{K \ldots, \neg \psi_1 \vDash \psi_2}{K \ldots \vDash \psi_1 \lor \psi_2}
  \]

  \[
  \frac{K \ldots, \neg \psi_2 \vDash \psi_1}{K \ldots \vDash \psi_1 \lor \psi_2}
  \]

\[P-\lor: \text{indeterministic!!!} \]

Rules for other connectives like implication “\(\rightarrow\)” and equivalence “\(\leftrightarrow\)”
are constructed accordingly.
Some Remarks on Sequent Calculus

- **premises** of a rule: sequent(s) above the line
- **conclusion** of a rule: sequent below the line
- **axiom**: rule without premises
- **non-deterministic rule**: \( P \lor \)
- **further non-determinism**: decision which rule to apply next
- **rules with case split**: \( P \land, A \lor \)
- **proof of formula** \( \psi \)
  1. start with \( \vdash \psi \)
  2. apply rules from bottom to top as long as possible, i.e., for given conclusion, find suitable premise(s)
  3. if finally all sequents are axioms then \( \psi \) is valid

- note: there are many variants of the sequent calculus
Algorithm: entails

Data: set of assumptions $\mathcal{A}$, formula $\psi$

Result: 1 iff $\mathcal{A}$ entails $\psi$, i.e., $\mathcal{A} \vdash \psi$

1. if ($\psi \in \mathcal{A}$) or ($\phi, \neg \phi \in \mathcal{A}$) then return 1;
2. if $\mathcal{A} \cup \{\psi\}$ contains only literals then return 0;
3. if $\psi = \neg \neg \psi'$ then return entails ($\mathcal{A}, \psi'$);
4. if $\neg \neg \phi \in \mathcal{A}$ then return entails ($\mathcal{A}\setminus\neg \phi \cup \{\phi\}, \psi$);
5. if $\neg \phi \in \mathcal{A}$ then return entails ($\mathcal{A}\setminus\neg \phi \cup \{\neg \psi\}, \phi$);
6. if $\phi_1 \land \phi_2 \in \mathcal{A}$ then return entails ($\mathcal{A}\setminus\{\phi_1 \land \phi_2\} \cup \{\phi_1, \phi_2\}, \psi$);
7. if $\phi_1 \lor \phi_2 \in \mathcal{A}$ then return entails ($\mathcal{A} \cup \{\neg \psi\}, \psi_1$) && entails ($\mathcal{A} \cup \{\neg \psi_2\}, \psi_1$);
8. switch $\psi$ do
   case $\neg \psi'$ do return entails ($\mathcal{A} \cup \{\psi'\}, \bot$);
   case $\psi_1 \lor \psi_2$ do return entails ($\mathcal{A} \cup \{\neg \psi_1\}, \psi_2$) || entails ($\mathcal{A} \cup \{\neg \psi_2\}, \psi_1$);
   case $\psi_1 \land \psi_2$ do return entails ($\mathcal{A}, \psi_1$) && entails ($\mathcal{A}, \psi_2$);
Proving XOR stronger than OR

proof direction

GoalAssum : \[b, (\neg a \lor \neg b), \neg a \vdash b\]
ContrAssum : \[a, (\neg a \lor \neg b), \neg a \vdash b\]

\[
\begin{align*}
A \lor & (a \lor b) \land (\neg a \lor \neg b) \vdash a \lor b \\
A \land & (a \land b) \lor (\neg a \land \neg b) \vdash a \lor b \\
P \land & \neg (a \land b) \lor (\neg a \land \neg b) \lor (a \land b) \vdash \neg ((a \lor b) \land (\neg a \land \neg b)) \lor (a \lor b)
\end{align*}
\]
Refuting XOR stronger than AND

<table>
<thead>
<tr>
<th>GAss</th>
<th>A ∨ V</th>
<th>CAss</th>
<th>A ∨ V</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, (¬a ∨ ¬b) ⊢ a</td>
<td>b, ¬b ⊢ a</td>
<td>b, ¬a ⊢ a</td>
<td>b, (¬a ∨ ¬b) ⊢ a</td>
</tr>
<tr>
<td>A ∨ V</td>
<td>P ∧ V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a ∨ b), (¬a ∨ ¬b) ⊢ a</td>
<td>(a ∨ b), (¬a ∨ ¬b) ⊢ a ∧ b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a ∨ b) ∧ (¬a ∨ ¬b) ⊢ a ∧ b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>¬((a ∨ b) ∧ (¬a ∨ ¬b)) ⊢ a ∧ b</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

counter example to validity: \( a = \bot, b = \top \)
Soundness and Completeness

For any calculus important properties are, soundness, i.e. the question “Can only valid formulas be shown as valid?” and completeness, i.e. the question “Is there a proof for every valid formula?”. 

Soundness

If a formula is shown to be valid in the Gentzen Calculus, then it is valid.

Completeness

Every valid formula can be proven to be valid in the Gentzen Calculus.
PROPOSITIONAL FORMULAS: SOLVING WITH DPLL

VL Logic
Part I: Propositional Logic
Proving Formulas in Normal Form

- In practice, formulas of arbitrary structure are quite challenging to handle
  - tree structure
  - simplifications affect only subtrees
- We have seen that CNF and DNF are able to represent every formula
  - so why not use them as input for SAT?

  **Conjunctive Normal Form**
  - refutability is easy to show
  - CNF can be efficiently calculated (polynomial)

  **Disjunctive Normal Form**
  - satisfiability is easy to show
  - complexity is in getting the DNF
- CNF and DNF can be obtained from the **truth tables**
  - exponential many assignments have to be considered
- alternative approach
  - **structural rewritings** are (satisfiability) equivalence preserving
DPLL Overview

The DPLL algorithm is ...

- **old** (invented 1962)
- **easy** (basic pseudo-code is less than 10 lines)
- **popular** (well investigated; also theoretical properties)
- usually realized for **formulas in CNF**
- using **binary constraint propagation (BCP)**
- in its modern form as **conflict drive clause learning (CDCL)**
  basis for state-of-the-art SAT solvers
Binary Constraint Propagation

**Definition (Binary Constraint Propagation (BCP))**

Let $\phi$ be a formula in CNF containing a unit clause $C$, i.e., $\phi$ has a clause $C = (l)$ which consists only of literal $l$. Then $BCP(\phi, l)$ is obtained from $\phi$ by

- removing all clauses with $l$
- removing all occurrences of $\overline{l}$

- BCP on variable $x$ can trigger application of BCP on variable $y$
- if BCP produces the empty clause, then the formula is unsatisfiable
- if BCP produces the empty CNF, then the formula is satisfiable

**Example**

$\phi = \{ (\lnot a \lor b \lor \lnot c), (a \lor b), (\lnot a \lor \lnot b), (a) \}$

1. $\phi' = BCP(\phi, a) = \{ (b \lor \lnot c), (\lnot b) \}$
2. $\phi'' = BCP(\phi', \lnot b) = \{ (\lnot c) \}$
3. $\phi''' = BCP(\phi', c) = \{ \} = \top$
DPLL Algorithm

1. **Algorithm**: evaluate
   - **Data**: formula $\phi$ in CNF
   - **Result**: 1 iff $\phi$ satisfiable

2. while 1 do
   3. $\phi = \text{BCP}(\phi)$
   4. if $\phi == \top$ then return 1;
   5. if $\phi == \bot$ then
      6. if stack.isEmpty() then return 0;
      7. $(l, \phi) = \text{stack.pop}()$
      8. $\phi = \phi \land l$
   9. else
      10. select literal $l$ occurring in $\phi$
      11. stack.push($\bar{l}, \phi$)
      12. $\phi = \phi \land l$
Some Remarks on DPLL

- DPLL is the basis for most state-of-the-art SAT solvers
  - Lingeling  http://fmv.jku.at/lingeling
  - CaDiCaL  http://fmv.jku.at/cadical
  - some more established solvers: MiniSAT, PicoSAT, Glucose, ...

- DPLL alone is not enough - powerful optimizations required for efficiency:
  - learning and non-chronological back-tracking (CDCL)
  - reset strategies and phase-saving
  - compact lazy data-structures
  - variable selection heuristics
  - usually combined with preprocessing before and inprocessing during search

- variants of DPLL are also used for other logics:
  - quantified propositional logic (QBF)
  - satisfiability modulo theories (SMT)

- challenge to parallelize
  - some successful attempts: ManySAT, Plingeling, Penelope, Treengeling, ...