SAT Solving Basics
Topics

Last lecture
• Going pro with solver-aided programming

Today
• Review of propositional logic
• Normal forms
• A basic SAT solver
Review of propositional logic

• Syntax
• Semantics
• Satisfiability and validity
• Proof methods
• Semantic judgments
Syntax of propositional logic

\((\neg p \land \top) \lor (q \rightarrow \bot)\)
Syntax of propositional logic

Atom  

**truth symbols:**  \( \top \) (“true”),  \( \bot \) (“false”)
**propositional variables:**  \( p, q, r, \ldots \)

Literal  

an atom \( \alpha \) or its negation \( \neg \alpha \)

Formula  

an atom or the application of a **logical connective** to formulas \( F_1, F_2 \):

\[
\neg F_1 \quad \text{“not” (negation)} \\
F_1 \land F_2 \quad \text{“and” (conjunction)} \\
F_1 \lor F_2 \quad \text{“or” (disjunction)} \\
F_1 \rightarrow F_2 \quad \text{“implies” (implication)} \\
F_1 \leftrightarrow F_2 \quad \text{“if and only if” (iff)}
\]
Semantics of propositional logic: interpretations

An **interpretation** $I$ for a propositional formula $F$ maps every variable in $F$ to a truth value:

$$I : \{ p \mapsto \text{true}, \; q \mapsto \text{false}, \; \ldots \}$$

$I$ is a **satisfying interpretation** of $F$, written as $I \models F$, if $F$ evaluates to true under $I$.

$I$ is a **falsifying interpretation** of $F$, written as $I \not\models F$, if $F$ evaluates to false under $I$.

A satisfying interpretation is also called a **model**.
Semantics of propositional logic: definition

**Base cases:**
- $I \models \top$
- $I \not\models \bot$
- $I \models p$ iff $I[p] = \text{true}$
- $I \not\models p$ iff $I[p] = \text{false}$

**Inductive cases:**
- $I \models \neg F$ iff $I \not\models F$
- $I \models F_1 \land F_2$ iff $I \models F_1$ and $I \models F_2$
- $I \models F_1 \lor F_2$ iff $I \not\models F_1$ or $I \models F_2$
- $I \models F_1 \rightarrow F_2$ iff $I \not\models F_1$ or $I \models F_2$
- $I \models F_1 \leftrightarrow F_2$ iff $I \models F_1$ and $I \models F_2$, or $I \not\models F_1$ and $I \not\models F_2$
Semantics of propositional logic: example

\[ F: (p \land q) \rightarrow (p \lor \neg q) \]
\[ I: \{ p \mapsto \text{true}, q \mapsto \text{false}\} \]

\[ I \models F \]
Satisfiability & validity of propositional formulas

- F is **satisfiable** iff $I \models F$ for some $I$.
- F is **valid** iff $I \models F$ for all $I$.

**Duality** of satisfiability and validity:

$F$ is valid iff $\neg F$ is unsatisfiable.

If we have a procedure for checking satisfiability, we can also check validity of propositional formulas, and vice versa.
Techniques for deciding satisfiability & validity

Search

Deduction

SAT solver
Techniques for deciding satisfiability & validity

**Search**
Enumerate all interpretations (i.e., build a truth table), and check that they satisfy the formula.

**Deduction**
Assume the formula is invalid, apply proof rules, and check for contradiction in every branch of the proof tree.

**SAT solver**
**Proof by search: enumerating interpretations**

\[
F: (p \land q) \rightarrow (p \lor \neg q)
\]

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Valid.
Proof by deduction: semantic arguments

A proof rule consists of

- **premise**: facts that have to hold to apply the rule.
- **conclusion**: facts derived from applying the rule.

Commas indicate derivation of multiple facts; pipes indicate alternative facts (branches in the proof).

\[
\begin{align*}
I &\models \neg F \\
I &\not\models F
\end{align*}
\]

\[
\begin{align*}
I &\models F_1 \land F_2 \\
I &\not\models F_1, I \models F_2
\end{align*}
\]

\[
\begin{align*}
I &\models F_1 \lor F_2 \\
I &\not\models F_1 \land F_2
\end{align*}
\]

\[
\begin{align*}
I &\models F_1 \rightarrow F_2 \\
I &\not\models F_1 \land F_2
\end{align*}
\]

\[
\begin{align*}
I &\models F_1 \iff F_2 \\
I &\not\models F_1 \lor F_2
\end{align*}
\]

\[
\begin{align*}
I &\not\models F_1 \leftrightarrow F_2
\end{align*}
\]

\[
\begin{align*}
I &\models F_1 \land \neg F_2 \\
I &\not\models \neg F_1 \land F_2
\end{align*}
\]
Proof by deduction: another example

Prove \( p \land \neg q \) or find a falsifying interpretation.

1. \( I \not\models p \land \neg q \) (assumed)
   a. \( I \not\models p \) (I, \( \land \))
   b. \( I \not\models \neg q \) (I, \( \land \))
      i. \( I \models q \) (Ib, \( \neg \))

The formula is invalid, and \( I = \{ p \mapsto \text{false}, q \mapsto \text{true} \} \) is a falsifying interpretation.
Proof by deduction: another example

\[
\begin{align*}
&\quad I ⊬ F \\
&\quad I ⊭ F
\end{align*}
\]

\[
\begin{align*}
&\quad I ⊫ F_1, I ⊩ F_2 \\
&\quad I ⊩ F_1 \land F_2
\end{align*}
\]

\[
\begin{align*}
&\quad I ⊩ F_1 \lor F_2 \\
&\quad I ⊭ F_1, I ⊩ F_2
\end{align*}
\]

\[
\begin{align*}
&\quad I ⊩ F_1 \rightarrow F_2 \\
&\quad I ⊭ F_1 \land F_2
\end{align*}
\]

\[
\begin{align*}
&\quad I ⊩ F_1 \leftrightarrow F_2 \\
&\quad I ⊩ F_1, I ⊩ F_2
\end{align*}
\]

\[
\begin{align*}
&\quad I ⊩ F_1 \land F_2 \\
&\quad I ⊩ F_1 \lor F_2
\end{align*}
\]

1. \( I ⊭ (p \land (p \rightarrow q)) \rightarrow q \)
2. \( I ⊭ q \quad (1, \rightarrow) \)
3. \( I ⊩ (p \land (p \rightarrow q)) \quad (1, \land) \)
4. \( I ⊩ p \quad (3, \land) \)
5. \( I ⊩ p \rightarrow q \quad (3, \land) \)
   a. \( I ⊭ p \quad (5, \rightarrow) \)
   b. \( I ⊩ q \quad (5, \rightarrow) \)

We have reached a contradiction in every branch of the proof, so the formula is valid.
Formulas $F_1$ and $F_2$ are **equivalent**, written $F_1 \iff F_2$, iff $F_1 \leftrightarrow F_2$ is valid.

Formula $F_1$ implies $F_2$, written $F_1 \implies F_2$, iff $F_1 \rightarrow F_2$ is valid.

$F_1 \iff F_2$ and $F_1 \implies F_2$ are **not** propositional formulas (not part of syntax). They are properties of formulas, just like validity or satisfiability.

What do these definitions tell us in the context of this course?
Normal Forms (NNF, DNF, CNF)
A **normal form** for a logic is a syntactic restriction such that every formula in the logic has an equivalent formula in the normal form.

Assembly language for a logic.

Three important normal forms for propositional logic:
- Negation Normal Form (NNF)
- Disjunctive Normal Form (DNF)
- Conjunctive Normal Form (CNF)
Negation Normal Form (NNF)

Atom := Variable | ⊤ | ⊥
Literal := Atom | ¬Atom
Formula := Literal | Formula op Formula
op := ∧ | ∨

The only allowed connectives are ∧, ∨, and ¬. ¬ can appear only in literals.

Conversion to NNF performed using DeMorgan’s Laws:
¬(F ∧ G) ⇔ ¬F ∨ ¬G
¬(F ∨ G) ⇔ ¬F ∧ ¬G
Disjunctive Normal Form (DNF)

Atom := Variable | ⊤ | ⊥
Literal := Atom | ¬Atom
Formula := Clause ∨ Formula
Clause := Literal | Literal ∧ Clause

• Disjunction of conjunction of literals.
• Deciding satisfiability of a DNF formula is trivial.
• Why not SAT solve by conversion to DNF?

To convert to DNF, convert to NNF and distribute ∧ over ∨:

\[(F \land (G \lor H)) \iff (F \land G) \lor (F \land H)\]
\[((G \lor H) \land F) \iff (G \land F) \lor (H \land F)\]
Conjunctive Normal Form (CNF)

Why CNF? Doesn't the conversion explode just as badly as DNF?

Atom := Variable | ⊤ | ⊥
Literal := Atom | ¬Atom
Formula := Clause ∧ Formula
Clause := Literal | Literal ∨ Clause

To convert to CNF, convert to NNF and distribute ∨ over ∧
(F ∨ (G ∧ H)) ⇔ (F ∨ G) ∧ (F ∨ H)
((G ∧ H) ∨ F) ⇔ (G ∨ F) ∧ (H ∨ F)

- Conjunction of disjunction of literals.
- Deciding the satisfiability of a CNF formula is hard.
- SAT solvers use CNF as their input language.
Equisatisfiability and Tseitin’s transformation

Formulas $F$ and $G$ are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula $F$ into an equisatisfiable CNF formula that is **linear** in the size of $F$.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

\[
\begin{align*}
x & \rightarrow (y \land z) \\
a_1 & \leftrightarrow (x \rightarrow a_2) \\
a_2 & \leftrightarrow (y \land z)
\end{align*}
\]
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

\[
\begin{align*}
x & \rightarrow (y \land z) \\
a_1 & \rightarrow (x \rightarrow a_2) \\
(x \rightarrow a_2) & \rightarrow a_1 \\
a_2 & \leftrightarrow (y \land z)
\end{align*}
\]
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

\[
x \rightarrow (y \land z)
\]

\[
a1 \\
\neg a1 \lor \neg x \lor a2 \\
(x \land \neg a2) \lor a1 \\
a2 \leftrightarrow (y \land z)
\]
Equisatisfiability and Tseitin’s transformation

Formulas $F$ and $G$ are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula $F$ into an equisatisfiable CNF formula that is **linear** in the size of $F$.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

$x \rightarrow (y \land z)$

\[ a_1, \neg a_1 \lor \neg x \lor a_2 \\
\neg a_2 \lor a_1 \\
a_2 \leftrightarrow (y \land z) \]
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

\[ x \rightarrow (y \land z) \]

- \( \neg a_1 \lor x \lor a_2 \)
- \( x \lor a_1 \)
- \( \neg a_2 \lor y \)
- \( \neg a_2 \lor z \)
- \( y \lor \neg z \lor a_2 \)
Another key feature of CNF: proof by resolution

Resolution rule
\[ a_1 \lor \ldots \lor a_n \lor \beta \lor b_1 \lor \ldots \lor b_m \lor \neg \beta \]
\[ \Rightarrow \quad a_1 \lor \ldots \lor a_n \lor b_1 \lor \ldots \lor b_m \]

Proving that a CNF formula is valid can be done using just this one proof rule!

Apply the rule until a contradiction (empty clause) is derived, or no more applications are possible.

This procedure is sound and complete: it always produces a correct answer.
Another key feature of CNF: unit resolution

Resolution rule

\[ a_1 \lor \ldots \lor a_n \lor \beta \quad b_1 \lor \ldots \lor b_m \lor \neg \beta \]
\[ a_1 \lor \ldots \lor a_n \lor b_1 \lor \ldots \lor b_m \]

Unit resolution rule

\[ \beta \quad b_1 \lor \ldots \lor b_m \lor \neg \beta \]
\[ b_1 \lor \ldots \lor b_m \]

Unit resolution specializes the resolution rule to the case where one of the clauses is unit (a single literal).

SAT solvers use unit resolution in combination with backtracking search to implement a sound and complete procedure for deciding CNF formulas.

Unit resolution is a sound but incomplete rule of deduction, which is why we need search!
A basic SAT solver
// Returns true if the CNF formula F is satisfiable; otherwise returns false.

DPLL(F)
G ← BCP(F)
if G = ⊤ then return true
if G = ⊥ then return false
p ← choose(vars(G))
return DPLL(G{p ↦ ⊤}) ||
      DPLL(G{p ↦ ⊥})

Boolean constraint propagation applies unit resolution until fixed point.
If BCP cannot reduce F to a constant, we choose an unassigned variable and recurse assuming that the variable is either true or false.
If the formula is satisfiable under either assumption, then we know that it has a satisfying assignment (expressed in the assumptions). Otherwise, the formula is unsatisfiable.
Summary

Today

• Review of propositional logic
• Normal forms
• A basic SAT solver

Next Lecture

• A modern SAT solver