Deadlines

• Jan 29 (11:59pm)
  • Presentation choice submission (submission on course webpage)

• Jan 31 (11:59pm)
  • First paper review (submission through Canvas assignments)

• Feb 17 (in class)
  • Proposal presentation
Topics

• Propositional logic review
• Boolean satisfiability problem (SAT)
• Satisfiability Modulo Theories (SMT)
Topics

- Propositional logic review
- Boolean satisfiability problem (SAT)
- Satisfiability Modulo Theories (SMT)
Syntax of propositional logic

\[(\neg p \land \top) \lor (q \rightarrow \bot)\]

**Atom**
- truth symbols: \( \top \) ("true"), \( \bot \) ("false")
- propositional variables: \( p, q, r \)

**Literal**
- an atom \( \alpha \) or its negation \( \neg \alpha \)

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

<table>
<thead>
<tr>
<th>( \neg F_1 )</th>
<th>&quot;not&quot;</th>
<th>(negation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_1 \land F_2 )</td>
<td>&quot;and&quot;</td>
<td>(conjunction)</td>
</tr>
<tr>
<td>( F_1 \lor F_2 )</td>
<td>&quot;or&quot;</td>
<td>(disjunction)</td>
</tr>
<tr>
<td>( F_1 \rightarrow F_2 )</td>
<td>&quot;implies&quot;</td>
<td>(implication)</td>
</tr>
<tr>
<td>( F_1 \leftrightarrow F_2 )</td>
<td>&quot;if and only if&quot;</td>
<td>(iff)</td>
</tr>
</tbody>
</table>
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 $I$
  • A satisfying interpretation is also called a **model**
• $I$ is a **falsifying interpretation** of $F$, written as $I \not\models F$, if $F$ evaluates to false under $I$
Semantics of propositional logic: definition

• **Base cases:**

  \[ \models T \]
  \[ \models \bot \]
  \[ \models p \quad \text{iff } \models [p] = \text{true} \]
  \[ \models \neg p \quad \text{iff } \models [p] = \text{false} \]

• **Inductive cases:**

  \[ \models \neg F \quad \text{iff } \models \neg F \]
  \[ \models F_1 \land F_2 \quad \text{iff } \models F_1 \text{ and } \models F_2 \]
  \[ \models F_1 \lor F_2 \quad \text{iff } \models F_1 \text{ or } \models F_2 \]
  \[ \models F_1 \rightarrow F_2 \quad \text{iff } \models \neg F_1 \text{ or } \models F_2 \]
  \[ \models F_1 \leftrightarrow F_2 \quad \text{iff } \models F_1 \text{ and } \models F_2 \text{, OR } \models \neg F_1 \text{ and } \models \neg F_2 \]
Semantics of propositional logic: example

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

\[ I \models F \]
Topics

• Propositional logic review
• Boolean satisfiability problem (SAT)
• Satisfiability Modulo Theories (SMT)
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.
Techniques for deciding satisfiability & validity

**SAT Solver**

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

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

• Where
  • **Commas** indicate derivation of multiple facts
  • **Pipes** indicate alternative facts (branches in the proof)
Proof by deduction: semantic arguments

\[
I \models \neg F \\
\therefore I \not\models F
\]

\[
I \models F_1 \land F_2 \\
\therefore I \models F_1, I \models F_2
\]

\[
I \models F_1 \lor F_2 \\
\therefore I \models F_1 \mid I \models F_2
\]

\[
I \models F_1 \rightarrow F_2 \\
\therefore I \not\models F_1 \mid I \models F_2
\]

\[
I \models F_1 \leftrightarrow F_2 \\
\therefore I \not\models F_1 \lor F_2 \mid I \models F_1 \land F_2
\]

\[
I \not\models \neg F \\
\therefore I \models F
\]

\[
I \not\models F_1 \land F_2 \\
\therefore I \not\models F_1 \mid I \not\models F_2
\]

\[
I \not\models F_1 \lor F_2 \\
\therefore I \not\models F_1, I \not\models F_2
\]

\[
I \not\models F_1 \rightarrow F_2 \\
\therefore I \not\models F_1 \mid I \not\models F_2
\]

\[
I \not\models F_1 \leftrightarrow F_2 \\
\therefore I \not\models F_1 \lor F_2 \mid I \models F_1 \land \neg F_2
\]

Proving \((p \land (p \rightarrow q)) \rightarrow q\):

\[
I \not\models (p \land (p \rightarrow q)) \rightarrow q
\]

\[
I \models (p \land (p \rightarrow q)) , I \not\models q
\]

\[
I \models p , I \models (p \rightarrow q)
\]

\[
I \not\models p \mid I \models q
\]

**Contradiction!**

So the formula is valid.
Getting ready for SAT solving with normal forms

• Arbitrary formula can be hard to solve!

• **Normal form**: a syntactic restriction such that every formula in the logic has an equivalent formula in the normal form

• 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 | T | F
Literal := Atom | ¬Atom
op := ∧ | ∨
Formula := Literal | Formula op Formula

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

Conversion to NNF performed using **De Morgan’s Laws:**

\[-(F \land G) \iff \neg F \lor \neg G\]
\[-(F \lor G) \iff \neg F \land \neg G\]
Disjunctive Normal Form (DNF)

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

• Disjunction of conjunction of literals
• Deciding satisfiability of a DNF formula is trivial
• However, may incur exponential increase in formula size

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)

- Conjunction of disjunction of literals
- Deciding the satisfiability of a CNF formula is hard
- \textbf{SAT solvers use CNF as their input language}
  - Linear increase in formula size

Atom := Variable | \( T \) | \( \bot \)
Literal := Atom | \( \neg \)Atom
Clause := Literal | Literal \( \lor \) Clause
Formula := Clause \( \land \) Formula

To convert to CNF, convert to NNF and distribute \( \lor \) over \( \land \):
\[
(FV(G \land H)) \iff (FV G) \land (FV H)
\]
\[
((G \land H) \lor F) \iff (G \lor F) \land (H \lor F)
\]
Propositional formula to CNF: Tseitin’s transformation

- Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses
Solving CNF: Proof by resolution

Resolution rule

\[
\begin{array}{c}
a_1 \lor \ldots \lor a_n \lor \beta \\
\hline
b_1 \lor \ldots \lor b_m \lor \neg \beta \\
\hline
a_1 \lor \ldots \lor a_n \lor b_1 \lor \ldots \lor b_m
\end{array}
\]

Unit resolution rule

\[
\begin{array}{c}
\beta \\
\hline
b_1 \lor \ldots \lor b_m \lor \neg \beta \\
\hline
b_1 \lor \ldots \lor b_m
\end{array}
\]

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

• Apply the rule until a contradiction, or no more applications are possible

• Unit resolution specializes the resolution rule to the case where one of the clauses is unit (a single literal)
// 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 (BCP)** applies unit resolution until fixed point
• If BCP cannot reduce $F$ to constant, we choose an unassigned variable and recurse assuming the variable is true or false
• If the formula is satisfiable under either assumption, then it has a satisfying assignment. Otherwise, it’s unsatisfiable.
DPLL: example

- An implication graph $G = (V, E)$ is a DAG recording the history of decisions and the resulting BCP deductions.
  - $v \in V$ is a literal and the decision level it got decided.
  - $\langle v, w \rangle \in E$ is labeled with antecedent($w$), i.e., the clause from which $w$ got decided.
DPLL: example

Can we learn from conflicts and avoid repeating that?

Implication graph

Decision literal
Implied literal
Conflict

Can we learn from conflicts and avoid repeating that?

Decision tree
Conflict-Driven Clause Learning (CDCL)

What gave rise to this implication graph?

**UIP:** any node (other than the conflict) on all paths from the current decision level to conflict.

**First UIP** is the one closest to conflict.

- A **conflict clause** blocks partial assignments leading to the conflict.
- Every cut that separates sources from the sink defines a valid conflict clause.
- Cut after the first **unique implication point (UIP)** gets the shortest conflict clause.

Should be “clause” (not “cause”), the video version is incorrect.
CDCL: algorithm

\text{ANALYZECONFLICT()}: \\
d \leftarrow \text{level}(\text{conflict}) \\
\text{if} \ d=0 \ \text{then return} \ -1 \\
c \leftarrow \text{antecedent}(\text{conflict}) \\
\text{while} \ !\text{oneLitAtLevel}(c, d) \\
\quad t \leftarrow \text{lastAssignedLitAtLevel}(c, d) \\
\quad v \leftarrow \text{varOfLit}(t) \\
\quad a \leftarrow \text{antecedent}(t) \\
\quad c \leftarrow \text{resolve}(a, c, v) \\
b \leftarrow \text{assertingLevel}(c) \\
\text{return} \ (b, c)

Start from the direct antecedent for conflict, traverse back until there is only one literal decided/implied at the current (highest) decision level in \( c \)

Apply resolution rule to \( a \) and \( c \) with respect to variable \( v \)

Backtrack to the second highest decision level in the newly derived constraint \( c \)

- Backtrack to level \( b \)
- Add \( c \) into the original formula
CDCL: example

c_1: ¬x_1 ∨ x_5 ∨ x_6

c_2: ¬x_5 ∨ x_7

c_3: ¬x_1 ∨ ¬x_6 ∨ ¬x_7

c_4: ¬x_1 ∨ x_2 ∨ x_5

c_5: ¬x_1 ∨ ¬x_3 ∨ x_5

c_6: ¬x_1 ∨ ¬x_4 ∨ x_5

c_7: x_1 ∨ ¬x_5

Implication graph

Decision tree

START

¬x_1

¬x_5

¬x_2

¬x_3

¬x_4

¬x_6

x_7

x_6

x_5

x_1

Implication graph

t ← lastAssignedLitAtLevel(c, d)
v ← varOfLit(t)
a ← antecedent(t)
c ← resolve(a, c, v)

Only x_5 at level 2, done!
Topics

• Propositional logic review
• Boolean satisfiability problem (SAT)
• Satisfiability Modulo Theories (SMT)
Satisfiability Modulo Theories (SMT)

• Some problems are more naturally expressed in other logics than propositional logic, e.g.:
  • Software verification needs reasoning about equality, arithmetic, data structures, ...

• SMT consists in deciding the satisfiability of a (quantifier-free) first-order formula with respect to a background theory

• Example:
  • Equality with Uninterpreted Functions (EUF)

\[ g(a)=c \land ( f(g(a)) \neq f(c) \lor g(a)=d ) \land c \neq d \]
Syntax of first-order logic (FOL)

• Logical symbols
  • Connectives: ¬, ∧, ∨, →, ↔
  • Parentheses: (, )
  • Quantifiers: ∃, ∀

• Non-logical symbols
  • Constants: x, y, z
  • N-ary functions: f(x), x+y
  • N-ary predicates: p(x), x>y
  • Variables: u, v, w

Usually only consider quantifier-free ground formulas
SMT: basic architecture

- Equality + UF
- Arithmetic
- Bit-vectors
- ...

SAT + Theory Solvers = SMT
SMT: basic idea

\[
x \geq 0, y = x + 1, (y > 2 \lor y < 1)
\]

\[
p_1, p_2, (p_3 \lor p_4)
\]

\[
p_1 \leftrightarrow (x \geq 0), p_2 \leftrightarrow (y = x + 1),
p_3 \leftrightarrow (y > 2), p_4 \leftrightarrow (y < 1)
\]

\[
p_1, p_2, \neg p_3, p_4
\]

\[
x \geq 0, y = x + 1, \neg (y > 2), y < 1
\]

Conflict clause

SAT

p1, p2, (p3 \lor p4)

p1 \leftrightarrow (x \geq 0), p2 \leftrightarrow (y = x + 1),
p3 \leftrightarrow (y > 2), p4 \leftrightarrow (y < 1)

p1, p2, \neg p3, p4

x \geq 0, y = x + 1, y < 1

\neg p1 \lor \neg p2 \lor \neg p3 \lor \neg p4

x \geq 0, y = x + 1, y < 1

Theory Solvers
Common theories

• Equality (and uninterpreted functions)
  • \( x = g(y) \)

• Fixed-width bitvectors
  • \( (b \gg 1) = c \)

• Linear arithmetic (over R and Z)
  • \( 2x + y \leq 5 \)

• Arrays
  • \( a[i] = x \)
Theories of linear integer and real arithmetic

• Signature
  • Integers (or reals)
  • Arithmetic operations: multiplication by an integer (or real) number, +, -.
  • Predicates: =, ≤.
  • Expanded with all constant symbols: x, y, z, ...

• Deciding TLIA and TLRA
  • Polynomial time for linear real arithmetic (LRA)
  • NP-complete for linear integer arithmetic (LIA)
LIA example: compiler optimization

for (i=1; i<=10; i++) {
    a[j+i] = a[j];
}

int v = a[j];
for (i=1; i<=10; i++) {
    a[j+i] = v;
}

A LIA formula that is unsatisfiable iff this optimization is valid:

\((i \geq 1) \land (i \leq 10) \land (j + i = j)\)
Theory of arrays

• Signature
  • Array operations: \texttt{read}, \texttt{write}
  • Equality: =
  • Expanded with all constant symbols: \texttt{x}, \texttt{y}, \texttt{z}, ...

• Axioms
  • \(\forall a, i, v. \text{read(write}(a, i, v), i) = v\)
  • \(\forall a, i, j, v. \neg(i = j) \rightarrow (\text{read(write}(a, i, v), j) = \text{read}(a, j))\)
  • \(\forall a, b. (\forall i. \text{read}(a, i) = \text{read}(b, i)) \rightarrow a = b\)

• Deciding \(T_A\)
  • Satisfiability problem: NP-complete
  • Used in many software verification tools to model memory
SMT tools

• Z3: https://github.com/Z3Prover/z3
  • Supported theories: empty theory, linear arithmetic, nonlinear arithmetic, bitvectors, arrays, datatypes, quantifiers, strings

• CVC4: https://cvc4.github.io/
  • Supported theories: rational and integer linear arithmetic, arrays, tuples, records, inductive data types, bitvectors, strings, and equality over uninterpreted function symbols

• STP: https://github.com/stp/stp
  • Supported theories: bitvectors, arrays

• Boolector: https://github.com/Boolector/boolector
  • Supported theories: bitvectors, arrays, and uninterpreted functions

• ...
Further readings

• https://rise4fun.com/z3/tutorial
• https://www.cs.princeton.edu/~zkincaid/courses/fall18/readings/SAT Handbook-CDCL.pdf
• https://cse442-17f.github.io/Conflict-Driven-Clause-Learning/
• https://homes.cs.washington.edu/~emina/blog/2017-06-23-a-primer-on-sat.html
Thanks and stay safe!