Advanced Software Testing and Debugging (CS598)

Formal Methods Basics

Fall 2020

Lingming Zhang

Some slides borrowed from UW CSE 507: Computer-Aided Reasoning for Software
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>Formula</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\neg F_1)</td>
<td>“not”</td>
<td>(negation)</td>
</tr>
<tr>
<td>(F_1 \land F_2)</td>
<td>“and”</td>
<td>(conjunction)</td>
</tr>
<tr>
<td>(F_1 \lor F_2)</td>
<td>“or”</td>
<td>(disjunction)</td>
</tr>
<tr>
<td>(F_1 \rightarrow F_2)</td>
<td>“implies”</td>
<td>(implication)</td>
</tr>
<tr>
<td>(F_1 \iff F_2)</td>
<td>“if and only if”</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 \( I \vDash F \), if \( F \) evaluates to true under \( I \)
  • A satisfying interpretation is also called a **model**

• \( I \) is a **falsifying interpretation** of \( F \), written as \( I \nvDash F \), if \( F \) evaluates to false under \( I \)
Semantics of propositional logic: definition

• **Base cases:**

  - $I \models T$
  - $I \not\models \bot$
  - $I \models p$ iff $I[p] \models \text{true}$
  - $I \not\models p$ iff $I[p] \not\models \text{false}$

• **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 $I \models F_1$ 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

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

\[ I \models F \]
Topics

• Propositional logic review
• Boolean satisfiability problem (SAT)
• Satisfiability Modulo Theories (SMT)
Satisfiability & validity of propositional formulas

- \( F \) is satisfiable iff \( I \not
models F \) 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 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.
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)

\[
\begin{align*}
I ⊨ \neg F & \quad \text{Premise} \\
I \not\models F & \quad \text{Conclusion} \\
I ⊨ F_1 \land F_2 & \\
I \models F_1, I \models F_2 \\
I ⊨ F_1 \lor F_2 & \\
I \models F_1 \mid I \models F_2 \\
I ⊨ F_1 \rightarrow F_2 & \\
I \not\models F_1 \mid I \models F_2 \\
I ⊨ F_1 \leftrightarrow F_2 & \\
I \not\models F_1 \lor F_2 \mid I \models F_1 \land F_2 \\
I \not\models F_1 \land F_2 & \\
I \models F_1 \land \neg F_2 \\
I \not\models \neg F_1 \lor F_2 & \\
I \models F_1 \land \neg F_2
\end{align*}
\]
Proof by deduction: semantic arguments

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

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

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

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

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

\[
I \models \neg F
\]
\[
I \models F
\]

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

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

\[
I \not\models F_1 \rightarrow F_2
\]
\[
I \models F_1, I \not\models F_2
\]

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

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

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

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

\[
I \not\models p
\]

\[
I \models q
\]

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 | \( \top \) | \( \bot \)
Literal := Atom | \( \neg \)Atom
Formula := Literal | Formula op Formula
op := \( \land \) | \( \lor \)

• The only allowed connectives are \( \land \), \( \lor \), and \( \neg \).
• \( \neg \) can appear only in literals

Conversion to NNF performed using De Morgan’s Laws:
\[ \neg(F \land G) \equiv \neg F \lor \neg G \]
\[ \neg(F \lor G) \equiv \neg F \land \neg G \]
Disjunctive Normal Form (DNF)

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

• 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 ∧ (G ∨ H)) ↔ (F ∧ G) ∨ (F ∧ H)
((G ∨ H) ∧ F) ↔ (G ∧ F) ∨ (H ∧ F)
Conjunctive Normal Form (CNF)

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

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

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

\[(F ∨ (G ∧ H)) ⇔ (F ∨ G) ∧ (F ∨ H)\]
\[((G ∧ H) ∨ F) ⇔ (G ∨ F) ∧ (H ∨ F)\]
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

\[
x → (y \land z)
\]

\[
a_1
\]

\[
a_1 ↔ (x → a_2)
\]

\[
a_2 ↔ (y \land z)
\]

\[
a_1
\]

\[
¬a_1 ∨ (¬x ∨ a_2)
\]

\[
(x → a_2) → a_1
\]

\[
a_2 ↔ (y \land z)
\]

\[
¬a_2 ∨ a_1
\]

\[
x ∨ a_1
\]

\[
¬a_2 ∨ a_1
\]

\[
¬a_2 ∨ y
\]

\[
¬a_2 ∨ z
\]

\[
¬y ∨ ¬z ∨ a_2
\]

\[
¬a_1 ∨ (¬x ∨ a_2)
\]

\[
x ∨ a_1
\]

\[
¬a_2 ∨ a_1
\]

\[
¬a_2 ∨ y
\]

\[
¬a_2 ∨ z
\]

\[
¬y ∨ ¬z ∨ a_2
\]
Solving CNF: Proof by resolution

Resolution rule

\[
\begin{align*}
& a_1 \lor \ldots \lor an \lor \beta \\
& b_1 \lor \ldots \lor bm \lor \neg \beta
\end{align*}
\]

\[\frac{a_1 \lor \ldots \lor an \lor b_1 \lor \ldots \lor bm}{a_1 \lor \ldots \lor an \lor b_1 \lor \ldots \lor bm}\]

Unit resolution rule

\[
\begin{align*}
& \beta \\
& b_1 \lor \ldots \lor bm \lor \neg \beta
\end{align*}
\]

\[\frac{\beta \lor b_1 \lor \ldots \lor bm \lor \neg \beta}{b_1 \lor \ldots \lor bm}\]

• 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

• Unit resolution specializes the resolution rule to the case where one of the clauses is **unit** (a single literal)
A basic solver: Davis-Putnam-Logemann-Loveland (DPLL, 1962)

// 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 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 it has a satisfying assignment (expressed in assumptions). Otherwise, it’s unsatisfiable.
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
- \( \langle v, w \rangle \in E \) iff \( v \neq w \), \( \neg v \in \text{antecedent}(w) \), and \( \langle v, w \rangle \) is labeled with \( \text{antecedent}(w) \)
- \( \text{antecedent}(v) \): the clause from which \( v \) got decided
- A unit clause \( c \) is the antecedent of its sole unassigned literal
DPLL: example

Can we learn from conflicts and avoid repeating that?
Conflict-Driven Clause Learning (CDCL)

ANALYZECONFLICT():

\[ d \leftarrow \text{level}(\text{conflict}) \]

if \( d=0 \) then return -1

\[ c \leftarrow \text{antecedent}(\text{conflict}) \]

while !oneLitAtLevel(c, d)

\[ t \leftarrow \text{lastAssignedLitAtLevel}(c, d) \]

\[ v \leftarrow \text{varOfLit}(t) \]

\[ a \leftarrow \text{antecedent}(t) \]

\[ c \leftarrow \text{resolve}(a, c, v) \]

\[ b \leftarrow \text{assertingLevel}(c) \]

return \( \langle b, c \rangle \)

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

\[ c_1: \neg x_1 \lor x_5 \lor x_6 \]
\[ c_2: \neg x_5 \lor x_7 \]
\[ c_3: \neg x_1 \lor x_6 \lor \neg x_7 \]
\[ c_4: \neg x_1 \lor x_2 \lor x_5 \]
\[ c_5: \neg x_1 \lor \neg x_3 \lor x_5 \]
\[ c_6: \neg x_1 \lor \neg x_4 \lor x_5 \]
\[ c_7: \neg x_1 \lor \neg x_5 \]

Implication graph

\[ t \leftarrow \text{lastAssignedLitAtLevel}(c, d) \]
\[ v \leftarrow \text{varOfLit}(t) \]
\[ a \leftarrow \text{antecedent}(t) \]
\[ c \leftarrow \text{resolve}(a, c, v) \]

Decision tree

START

\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ x_4 \]
\[ x_5 \]
\[ x_6 \]
\[ x_7 \]

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
...
SMT: basic idea

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

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

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

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

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

\[ \neg p_1 \lor \neg p_2 \lor \neg p_4 \]

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

Conflict clause

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: \( x, y, z, \ldots \)

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

\[
\text{for } (i=1; i\leq 10; i++) \{ \\
   a[j+i] = a[j]; \\
\} \\
\]

A LIA formula that is unsatisfiable iff this transformation is valid:
\[
(i \geq 1) \land (i \leq 10) \land (j + i = j)
\]

\[
\text{int } v = a[j]; \\
\text{for } (i=1; i\leq 10; i++) \{ \\
   a[j+i] = v; \\
\}
\]
Theory of arrays

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

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

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

- **Z3**: [https://github.com/Z3Prover/z3](https://github.com/Z3Prover/z3)
  - **Supported theories**: empty theory, linear arithmetic, nonlinear arithmetic, bitvectors, arrays, datatypes, quantifiers, strings

- **CVC4**: [https://cvc4.github.io/](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](https://github.com/stp/stp)
  - **Supported theories**: bitvectors, arrays

- **Boolector**: [https://github.com/Boolector/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/SATHandbook-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!