The Logics of Program Verification

Testing, Quality Assurance, and Maintenance
Winter 2020

Prof. Arie Gurfinkel
method factorial (n: int) returns (v: int)
  requires n >= 0;
  ensures v = fact(n);
{
  v := 1;
  if (n <= 1) { return v; }
  var i := 2;
  while (i <= n)
    invariant i <= n + 1
    invariant v = fact(i - 1)
    {
      v := i * v;
      i := i + 1;
    }
  return v;
}
Program Verification

How can we *argue* that a given program is correct
• i.e., satisfies its formal specifications?

Such an argument must combine
• Operational Semantics – to understand different programming constructs
• Propositional Reasoning – to break the problem into sub-goals that can be reasoned individually and combined later
• Mathematical Reasoning – properties of numbers, arithmetic, factorial, etc…
• Formal argument style – to mechanically check the flow of reasoning

All of this requires a LOGIC
• A formal language with well-defined semantics and strict reasoning rules
Three Logics of Program Verification

Propositional Logic
(logic of Boolean circuits)

First Order Logic
(logic of mathematical theories)

Hoare Logic
(logic of programs)

Program Verifier (Dafny)
SMT Solver (Z3)
SAT Solver (Z3)
# Plan for the next few weeks

<table>
<thead>
<tr>
<th>Week</th>
<th>Monday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 7 (Feb 24)</td>
<td>Propositional Logic</td>
<td>First Order Logic</td>
</tr>
<tr>
<td>Week 8 (March 2)</td>
<td>SAT/SMT Solving</td>
<td>Hoare Logic (part 1)</td>
</tr>
<tr>
<td>Week 9 (March 9)</td>
<td>NO CLASS</td>
<td>Hoare Logic (part 2)</td>
</tr>
</tbody>
</table>

Understanding formal logic can be **boring** hard. Don’t ignore suggested reading material!!!
Propositional Logic

Testing, Quality Assurance, and Maintenance
Winter 2020

Prof. Arie Gurfinkel
References

• Chapter 1 of Logic for Computer Scientists

• Chapter 1 of Calculus of Computation
What is Logic

According to Merriam-Webster dictionary logic is:

a (1) : a science that deals with the principles and criteria of validity of inference and demonstration

d : the arrangement of circuit elements (as in a computer) needed for computation; also: the circuits themselves
What is Formal Logic

Formal Logic consists of

• syntax – what is a legal sentence in the logic
• semantics – what is the meaning of a sentence in the logic
• proof theory – formal (syntactic) procedure to construct valid/true sentences

Formal logic provides

• a language to precisely express knowledge, requirements, facts
• a formal way to reason about consequences of given facts rigorously
Propositional Logic (or Boolean Logic)

Explores simple grammatical connections such as *and*, *or*, and *not* between simplest “atomic sentences”

\[
A = \text{“Paris is the capital of France”}
\]
\[
B = \text{“mice chase elephants”}
\]

The subject of propositional logic is to declare formally the truth of complex structures from the truth of individual atomic components

- \(A \text{ and } B\)
- \(A \text{ or } B\)
- if \(A\) then \(B\)
Syntax and Semantics

Syntax

• MW: the way in which linguistic elements (such as words) are put together to form constituents (such as phrases or clauses)

• Determines and restricts how things are written

Semantics

• MW: the study of meanings

• Determines how syntax is interpreted to give meaning
Syntax of Propositional Logic

An atomic formula has a form $A_i$, where $i = 1, 2, 3 \ldots$

Formulas are defined inductively as follows:

• All atomic formulas are formulas
• For every formula $F$, $\neg F$ (called not $F$) is a formula
• For all formulas $F$ and $G$, $F \land G$ (called and) and $F \lor G$ (called or) are formulas

Abbreviations

• use $A, B, C, \ldots$ instead of $A_1, A_2, \ldots$
• use $F_1 \rightarrow F_2$ instead of $\neg F_1 \lor F_2$ (implication)
• use $F_1 \leftrightarrow F_2$ instead of $(F_1 \rightarrow F_2) \land (F_2 \rightarrow F_1)$ (iff)
Syntax of Propositional Logic (PL)

truth_symbol ::= \top (true) | \bot (false)

variable ::= p, q, r, \ldots

atom ::= truth_symbol | variable

literal ::= atom | \neg atom

formula ::= literal |

\neg formula |

formula \land formula |

formula \lor formula |

formula \rightarrow formula |

formula \leftrightarrow formula
Example

\[ F = \neg ((A_5 \land A_6) \lor \neg A_3) \]

Sub-formulas are

\[ F, ((A_5 \land A_6) \lor \neg A_3), \]

\[ A_5 \land A_6, \neg A_3, \]

\[ A_5, A_6, A_3 \]
Semantics of propositional logic

For an atomic formula $A_i$ in $D$: \[ A'(A_i) = A(A_i) \]

\[
\begin{align*}
A'(\langle F \land G \rangle) & = 1 \quad \text{if } A'(F) = 1 \text{ and } A'(G) = 1 \\
& = 0 \quad \text{otherwise}
\end{align*}
\]

\[
\begin{align*}
A'(\langle F \lor G \rangle) & = 1 \quad \text{if } A'(F) = 1 \text{ or } A'(G) = 1 \\
& = 0 \quad \text{otherwise}
\end{align*}
\]

\[
\begin{align*}
A'(\lnot F) & = 1 \quad \text{if } A'(F) = 0 \\
& = 0 \quad \text{otherwise}
\end{align*}
\]
Example

\[ F = \neg (A \land B) \lor C \]

\[ \mathcal{A}(A) = 1 \]
\[ \mathcal{A}(B) = 1 \]
\[ \mathcal{A}(C) = 0 \]
### Truth Tables for Basic Operators

<table>
<thead>
<tr>
<th>$A(F)$</th>
<th>$A(G)$</th>
<th>$A((F \land G))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A(F')$</th>
<th>$A(\neg F)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A(F)$</th>
<th>$A(G)$</th>
<th>$A((F \lor G))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
\[ F = \neg(A \land B) \lor C \]

\[ \mathcal{A}(A) = 1 \]
\[ \mathcal{A}(B) = 1 \]
\[ \mathcal{A}(C) = 0 \]
Propositional Logic: Semantics

An assignment A is *suitable* for a formula F if A assigns a truth value to every atomic proposition of F.

An assignment A is a *model* for F, written $A \vDash F$, iff

- A is suitable for F
- $A(F) = 1$, i.e., F holds under A

A formula F is *satisfiable* iff F has a model, otherwise F is *unsatisfiable* (or contradictory).

A formula F is *valid* (or a tautology), written $\models F$, iff every suitable assignment for F is a model for F.
Determining Satisfiability via a Truth Table

A formula $F$ with $n$ atomic sub-formulas has $2^n$ suitable assignments.

Build a truth table enumerating all assignments.

$F$ is satisfiable iff there is at least one entry with 1 in the output.

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$\cdots$</th>
<th>$A_{n-1}$</th>
<th>$A_n$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$A_1(F)$</td>
</tr>
<tr>
<td>$A_2$:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$A_2(F)$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\ddots$</td>
<td>$\vdots$</td>
<td></td>
<td></td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$A_{2^n}$:</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$A_{2^n}(F)$</td>
</tr>
</tbody>
</table>
An example

\[ F = (\neg A \rightarrow (A \rightarrow B)) \]

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$\neg A$</th>
<th>$(A \rightarrow B)$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Validity and Unsatisfiability

Theorem:
A formula $F$ is valid if and only if $\neg F$ is unsatisfiable

Proof:
$F$ is valid $\iff$ every suitable assignment for $F$ is a model for $F$
$\iff$ every suitable assignment for $F$ is not a model for $\neg F$
$\iff \neg F$ does not have a model
$\iff \neg F$ is unsatisfiable
Normal Forms: CNF and DNF

A literal is either an atomic proposition $v$ or its negation $\neg v$

A clause is a disjunction of literals

- e.g., $(v_1 \lor \neg v_2 \lor v_3)$

A formula is in Conjunctive Normal Form (CNF) if it is a conjunction of disjunctions of literals (i.e., a conjunction of clauses):

- e.g., $(v_1 \lor \neg v_2) \land (v_3 \lor v_2)$

A formula is in Disjunctive Normal Form (DNF) if it is a disjunction of conjunctions of literals

\[
\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} L_{i,j}
\]

\[
\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} L_{i,j}
\]
From Truth Table to CNF and DNF

\[
(\neg A \land \neg B \land \neg C) \lor \\
(A \land \neg B \land \neg C) \lor \\
(A \land \neg B \land C)
\]

\[
(A \lor B \lor \neg C) \land \\
(A \lor \neg B \lor C) \land \\
(A \lor \neg B \lor \neg C) \land \\
(\neg A \lor \neg B \lor C) \land \\
(\neg A \lor \neg B \lor \neg C)
\]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Normal Form Theorem

**Theorem**: For every formula $F$, there is an equivalent formula $F_1$ in CNF and $F_2$ in DNF

**Proof**: (by induction on the structure of the formula $F$)
ENCODING PROBLEMS INTO CNF-SAT
Graph k-Coloring

Given a graph $G = (V, E)$, and a natural number $k > 0$ is it possible to assign colors to vertices of $G$ such that no two adjacent vertices have the same color.

Formally:
- does there exists a function $f : V \rightarrow [0..k)$ such that
- for every edge $(u, v)$ in $E$, $f(u) \neq f(v)$

Graph coloring for $k > 2$ is NP-complete

**Problem**: Encode k-coloring of $G$ into CNF
- construct CNF $C$ such that $C$ is SAT iff $G$ is k-colorable

https://en.wikipedia.org/wiki/Graph_coloring
**k-coloring as CNF**

Let a Boolean variable $f_{v,i}$ denote that vertex $v$ has color $i$

- if $f_{v,i}$ is true if and only if $f(v) = i$

Every vertex has at least one color

$$\bigvee_{0 \leq i < k} f_{v,i} \quad (v \in V)$$

No vertex is assigned two colors

$$\bigwedge_{0 \leq i < j < k} (\neg f_{v,i} \lor \neg f_{v,j}) \quad (v \in V)$$

No two adjacent vertices have the same color

$$\bigwedge_{0 \leq i < k} (\neg f_{v,i} \lor \neg f_{u,i}) \quad ((v, u) \in E)$$
PROPOSITIONAL REASONING
Propositional Resolution

\[
\begin{align*}
C \lor p & \quad \quad \quad D \lor \neg p \\
\hline
C \lor D
\end{align*}
\]

\[
\text{Res}({C, p}, {D, \neg p}) = \{C, D\}
\]

Given two clauses \((C, p)\) and \((D, \neg p)\) that contain a literal \(p\) of different polarity, create a new clause by taking the union of literals in \(C\) and \(D\).
Resolution Lemma

Lemma:
Let $F$ be a CNF formula. Let $R$ be a resolvent of two clauses $X$ and $Y$ in $F$. Then, $F \cup \{R\}$ is equivalent to $F$.
Proof System

An inference rule is a tuple \((P_1, \ldots, P_n, C)\)
- where, \(P_1, \ldots, P_n, C\) are formulas
- \(P_i\) are called premises and \(C\) is called a conclusion
- intuitively, the rules says that the conclusion is true if the premises are

A proof system \(P\) is a collection of inference rules

A proof in a proof system \(P\) is a tree (or a DAG) such that
- nodes are labeled by formulas
- for each node \(n\), \((\text{parents}(n), n)\) is an inference rule in \(P\)
Propositional Resolution

\[ C \lor p \quad \quad D \lor \neg p \]

\[ \therefore C \lor D \]

Propositional resolution is a sound inference rule.

Proposition resolution system consists of a single propositional resolution rule.
Example of a resolution proof

A refutation of \( \neg p \lor \neg q \lor r, p \lor r, q \lor r, \neg r \):
Resolution Proof Example

Show by resolution that the following CNF is UNSAT

\[-b \land (\neg a \lor b \lor \neg c) \land a \land (\neg a \lor c)\]

\[-a \lor b \lor \neg c\]

\[\text{a}\]

\[\overline{b \lor \neg c}\]

\[\text{b}\]

\[\text{a}\]

\[-a \lor c\]

\[\text{c}\]

\[\bot\]
Entailment and Derivation

A set of formulas $F$ entails a set of formulas $G$ iff every model of $F$ and is a model of $G$

$$F \models G$$

A formula $G$ is derivable from a formula $F$ by a proof system $P$ if there exists a proof whose leaves are labeled by formulas in $F$ and the root is labeled by $G$

$$F \vdash_P G$$
Soundness and Completeness

A proof system $P$ is **sound** iff

\[
(F \vdash_P G) \implies (F \models G)
\]

A proof system $P$ is **complete** iff

\[
(F \models G) \implies (F \vdash_P G)
\]
Completeness of Propositional Resolution

**Theorem:** Propositional resolution is sound and complete for propositional logic
Proof by resolution

Notation: positive numbers mean variables, negative mean negation
Let $\varphi = (1\ 3) \land (-1\ 2\ 5) \land (-1\ 4) \land (-1\ -4) \land (1\ -2)$
We’ll try to prove $\varphi \rightarrow (3\ 5)$
Resolution

Resolution is a sound and complete inference system for CNF.
If the input formula is unsatisfiable, there exists a proof of the empty clause.
Example: UNSAT Derivation

Notation: positive numbers mean variables, negative mean negation
Let $\varphi = (1\ 3) \land (-1\ 2) \land (1\ -2) \land (-1\ 4) \land (-1\ -4) \land (-3)$
Logic for Computer Scientists: Ex. 33

Using resolution show that

\[ A \land B \land C \]

is a consequence of

\[ \neg A \lor B \]
\[ \neg B \lor C \]
\[ A \lor \neg C \]
\[ A \lor B \lor C \]
Show using resolution that $F$ is valid

$$F = (\neg B \land \neg C \land D) \lor (\neg B \land \neg D) \lor (C \land D) \lor B$$

$$\neg F = (B \lor C \lor \neg D) \land (B \lor D) \land (\neg C \lor \neg D) \land \neg B$$