First-order Logic

Specifications section, Logic topic, Lecture 3

Pavel Panchekha
CS 6110, U of Utah
14 January 2020
Facts about Booleans

That's what you can say; what does it mean?

Depends on the values of the variables

\[
\begin{align*}
\llbracket v \rrbracket &= \text{"v is True"} \\
\llbracket \neg x \rrbracket &= \text{not } \llbracket x \rrbracket \\
\llbracket x \land y \rrbracket &= \text{both } \llbracket x \rrbracket \text{ and } \llbracket y \rrbracket \\
\llbracket x \lor y \rrbracket &= \text{either } \llbracket x \rrbracket \text{ or } \llbracket y \rrbracket \text{ or both}
\end{align*}
\]
What use is a Spec?

That’s what you can say; what does it mean?

Depends on the values of the variables 😞

We want to use the spec before running a program

Variables don’t have values yet!

“Could it be true?” “Must it be true?” “If this, then that?”

Satisfiability  Validity  Implication
“Must \((x \lor y) \land (\neg x \lor z) \land (\neg y \lor z) \land \neg z\) be false?”  Yes.

If \((x \lor y)\) and \((\neg x \lor z)\) then \((y \lor z)\)

\[
\begin{array}{c}
x \lor y \\
\uparrow & \\
\neg x \lor z \\
\downarrow & \\
y \lor z
\end{array}
\]

Logical Resolution
DPLL algorithm

Proof by resolution, presented as a program

1. Put into conjunctive form
2. Check for variables all alone
3. Check for variables with one polarity
4. Pick a variable and try setting to True
5. If it doesn’t work, it must be False
First-order Logic

From booleans to **integers**

Addition/multiplication, new kinds of questions

Semantics of **quantifiers**

Alternation, closed terms, and internalization

**Generalizing** integers to arbitrary data types

The recipe for an arbitrary logic
Integer Specifications

A language for arithmetic facts
Specifications

Generally describe **facts about program values**

Today, we go **from boolean to integer values**

Like before, we’ll discuss **syntax**, then **semantics**
Boolean Logic Syntax

\[
p, q := \textcolor{red}{v} \mid \neg p \mid p \land q \mid p \lor q\]

 Addition, multiplication?

\[
x, y := \textcolor{blue}{v} \mid \textcolor{blue}{n} \mid -x \mid x + y \mid x \times y\]

\text{Var, Constants}
Integer Logic Syntax

Boolean Expressions

\[ p, q := \neg p \mid p \land q \mid p \lor q \mid x = y \mid x < y \]

Integer Expressions

\[ x, y := v \mid n \mid -x \mid x + y \mid x \times y \]

Equality, comparison?
Examples

Addition is **commutative** (order-independent)

\[ x + y = y + x \]

Adding to a **comparison** on both sides is valid

\[ x < y \iff x + z < y + z \]

\[ p \iff q = (p \land q) \lor (\neg p \land \neg q) \]
Examples

**Multiplying a comparison** by a positive number is valid

\[ 0 < z \rightarrow (x < y \leftrightarrow x \times z < y \times z) \]

Condition on the equivalence

Integers are **either even or odd**

\[ (n = 2 \times k) \vee (n = 2 \times k + 1) \]
Integer Semantics

In integer logic, what does a statement mean?

- $\llbracket p \rrbracket \in \{ \text{True}, \text{False} \}$
- $\llbracket v \rrbracket = \text{value of } v$
- $\llbracket n \rrbracket = \text{the integer } n$

- $\llbracket \neg p \rrbracket = \text{not } \llbracket p \rrbracket$
- $\llbracket x = y \rrbracket = \llbracket x \rrbracket \text{ equals } \llbracket y \rrbracket$
- $\llbracket x < y \rrbracket = \llbracket x \rrbracket \text{ is less than } \llbracket y \rrbracket$

- $\llbracket [x] \rrbracket \in \{ \ldots, -1, 0, 1, \ldots \}$
- $\llbracket -x \rrbracket = \text{negative } \llbracket x \rrbracket$
- $\llbracket x \times y \rrbracket = \text{product of } \llbracket x \rrbracket \text{ and } \llbracket y \rrbracket$
- $\llbracket x + y \rrbracket = \text{sum of } \llbracket x \rrbracket \text{ and } \llbracket y \rrbracket$

- $\llbracket p \wedge q \rrbracket = \text{both } \llbracket p \rrbracket \text{ and } \llbracket q \rrbracket$
- $\llbracket p \lor q \rrbracket = \text{either } \llbracket p \rrbracket \text{ or } \llbracket q \rrbracket \text{ or both}$
Using a Specification

“Could it be true?”  “Must it be true?”  “If this, then that?”

Satisfiability  Validity  Implication

Next time: what evidence can we provide of these?

\[(n = 2 \times k) \lor (n = 2 \times k + 1)\]

Satisfiable?  Valid?  Something else?
Quantifiers

New questions, internalization, and models
Using a Specification

\[(n = 2 \times k) \lor (n = 2 \times k + 1)\]

Satisfiable?  
Valid?  
Something else?

None of the above. Validity on \( n \), satisfiability for \( k \)

Must it be the case, for all \( n \)

That it could be the case, for some \( k \)

\[(n = 2 \times k) \lor (n = 2 \times k + 1)\]
Complex Questions

Perfect squares are zero or one modulo 4

For all $n$

$n = k \times k \iff (n = 4 \times m + 1) \lor (n = 4 \times m)$

For some $k$  For some $m$

Solution: include all/some in the specification itself

$p, q := \ldots \mid \forall v, p \mid \exists v, p$

For all integers $v$, $p$ is true  For some integer $v$, $p$ is true
Complex Questions

Perfect squares are zero or one modulo 4

For all $n$

$$\forall n, \left( \exists k, n = k \times k \right) \implies \exists m, \left( n = 4 \times m + 1 \right) \lor \left( n = 4 \times m \right)$$

For some $k$ \hspace{2cm} For some $m$

Solution: include all/some in the specification itself

$$p, q := \ldots \mid \forall v, p \mid \exists v, p$$

For all integers $v$, $p$ is true \hspace{1cm} For some integer $v$, $p$ is true
Quantifier Semantics

\[ p, q \ := \ldots \mid \forall v, p \mid \exists v, p \]

For all integers \( v \), \( p \) is true

For some integer \( v \), \( p \) is true

\[ \forall x, \exists y, x < y \]

For some integer \( y \), \( x < y \) is true

What is \( x \)?

\[ \Gamma = \{ x : 3, y : 5, \ldots \} \]

\[ [v]_\Gamma = \text{value of } v \text{ in } \Gamma \]

\[ [x+y]_\Gamma = \text{sum of } [x]_\Gamma \text{ and } [y]_\Gamma \]
Quantifier Semantics

\[ p, q ::= \ldots \mid \forall v, p \mid \exists v, p \]

For all integers \( v \), \( p \) is true

\[ [v]_\Gamma = \text{value of } v \text{ in } \Gamma \]

For some integer \( v \), \( p \) is true

\[ [x + y]_\Gamma = \text{sum of } [x]_\Gamma \text{ and } [y]_\Gamma \]

\[ [\forall v, p]_\Gamma = [p]_{\Gamma'} \text{ for all } \Gamma', \text{ where } \Gamma'[x] = \Gamma[x] \text{ for all } x \text{ except } v \]

Semantics of \( \forall \)
Internalization

∀x, p → “For all x, p is true” → “p must be true”

Quantifiers internalize the notion of validity / satisfiability

“p is satisfiable”

∃x, ∃y, ..., ∃z, p

“p is valid”

∀x, ∀y, ..., ∀z, p

Closed terms (all variables quantified) sufficient

Only one question to ask: is \( p \) true?
<table>
<thead>
<tr>
<th>∀x, ∃y, x &lt; y</th>
<th>∀x, ∃y, x + y = y</th>
<th>∀x, ∃y, x = y × y</th>
</tr>
</thead>
<tbody>
<tr>
<td>∃x, ∀y, x &lt; y</td>
<td>∃x, ∀y, x + y = y</td>
<td>∃x, ∀y, x = y × y</td>
</tr>
<tr>
<td>∃x, ∃y, x &lt; y</td>
<td>∀y, ∃x, x + y = y</td>
<td>∀y, ∃x, x = y × y</td>
</tr>
<tr>
<td>∀x, ∀y, x &lt; y</td>
<td>∀y, ∀x, x + y = y</td>
<td>∀y, ∀x, x = y × y</td>
</tr>
<tr>
<td></td>
<td>∃x, ∃y, x + y = y</td>
<td>∃x, ∃y, x = y × y</td>
</tr>
<tr>
<td></td>
<td>∀x, ∀y, x + y = y</td>
<td>∀x, ∀y, x = y × y</td>
</tr>
</tbody>
</table>
Equivalences

\(-\forall x, p \leftrightarrow \exists x, \neg p\)
\(- (p \land q) \leftrightarrow \neg p \lor \neg q\)

Prefix form: all quantifiers at the beginning

\((\forall x, p) \lor q \rightarrow \forall x, p \lor q\)
\((\forall x, p) \land q \rightarrow \forall x, p \land q\)

Quantifier alternations are what matter

\(\forall x, \forall y, p \leftrightarrow \forall y, \forall x, p\)
\(\forall x, \exists y, p \leftrightarrow \exists y, \forall x, p\)
Course Updates

More about the class project
Sample Projects

Class project in **groups of two**. (Result of survey!)

- Symbolic execution for LLVM IR
- Fuzzing existing SMT solvers
- Building a parallel SAT/SMT solver
- Modeling distributed systems in TLA+
Alternative Assignment

Solo alternative to project. Released soon.

1. Implement a simple programming language
2. Implement weakest precondition generation
3. Implement static analysis for array bounds
4. Verify a quicksort implementation
Beyond Arithmetic
Domains, relations, and theories
Theory of Arrays

How would we write statements about arrays?

\[ p, q = \ldots | x \in a \]
\[ x, y = \ldots | a[x] | \text{len}(a) \]
\[ a = v | a[x := y] \]

**New logic** for every data type?  **No!**

- Syntax
- Semantics
- Equivalences
- Normal forms
Theory of Arrays

First-order Logic

\[ p, q := \neg p \mid p \lor q \mid p \land q \mid \forall v, p \mid \exists v, p \]

\[ \mid x = y \mid x < y \mid x \in a \quad \text{Relations} \]

Constants

\[ x, y := u \mid n \mid -x \mid x + y \mid x \times y \mid a[x] \mid \text{len}(a) \]

Functions

\[ a := v \mid a[x := y] \]

Sorts
Theory of Arrays

Theory: a set of sorts, constants, functions, and relations

Theories are like programs, the logic like an OS

### Sorts
- Int
- Array

### Constants
- $n : \text{Int}$

### Functions
- $\neg \text{Int} : \text{Int}$
- $\text{Int} + \text{Int} : \text{Int}$
- $\text{Int} \times \text{Int} : \text{Int}$
- $\text{Array}[\text{Int}] : \text{Int}$
- $\text{len}(\text{Array}) : \text{Int}$
- $\text{Array}[\text{Int} := \text{Int}] : \text{Int}$

### Relations
- $\text{Int} = \text{Int}$
- $\text{Int} < \text{Int}$
- $\text{Int} \in \text{Array}$

Separate the logic from the data and operations
Syntax

\[ p, q ::= \neg p \mid p \lor q \mid p \land q \mid \forall v, p \mid \exists v, p \]

For each **sort** \( T \)
\[ e_T ::= v_T \]

For each **constant** \( c : T \)
\[ e_T ::= c \]

For each **function** \( f(T_1, T_2, \ldots) : T \)
\[ e_T ::= f(e_{T_1}, e_{T_1}, \ldots) \]

For each **relation** \( R(T_1, T_2, \ldots) \)
\[ p ::= R(e_{T_1}, e_{T_1}, \ldots) \]
Semantics

For each sort $T$

A value set $T$ (or “Domain”)
$$[[e_T]] \in T$$

For each constant $c : T$

A value $c \in T$
$$[[c]] = c$$

For each function $f(T_1, T_2, \ldots) : T$

A function $f : T_1, T_2, \ldots \rightarrow T$
$$[[f(a, b, \ldots)]] = f([[a]], [[b]], \ldots)$$

For each relation $R(T_1, T_2, \ldots)$

A relation $R : T_1, T_2, \ldots \rightarrow \text{Bool}$
$$[[R(a, b, \ldots)]] = R([[a]], [[b]], \ldots)$$
The Recipe

First-order logic

Sorts, constants, functions, relations

+ Semantics

Theory
## Examples

<table>
<thead>
<tr>
<th></th>
<th>Graphs</th>
<th>Binary search trees</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sorts</strong></td>
<td>Node, Edge</td>
<td>Tree, Value</td>
</tr>
<tr>
<td><strong>Constants</strong></td>
<td>None</td>
<td>Empty : Tree</td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td>Edge.from : Node</td>
<td>Node(Value, Tree, Tree) : Tree</td>
</tr>
<tr>
<td></td>
<td>Edge.to : Node</td>
<td>Tree.left : Tree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tree.right : Tree</td>
</tr>
<tr>
<td><strong>Relations</strong></td>
<td>Node = Node</td>
<td>Value &lt; Value</td>
</tr>
<tr>
<td></td>
<td>Edge = Edge</td>
<td>Value ∈ Tree</td>
</tr>
</tbody>
</table>
Next class: First-order Proofs

To do:
☐ Course feedback
☐ Reading in textbook
☐ Assignment 1
First-order Logic

From booleans to **integers**

Addition/multiplication, new kinds of questions

Semantics of **quantifiers**

Alternation, closed terms, and internalization

**Generalizing** integers to arbitrary data types

The recipe for an arbitrary logic
EVIDENCE

NO TABLES

FINITE PROOFS
MATH AS GAME

PROOF RULES
Completeness

The Map

Vs

The Territory
Next class:

First-order Proofs

To do:

☐ Course feedback
☐ Reading in textbook
☐ Assignment 1