CS357 Software Verification

Logic for Program Specification

Lecturer:Rosemary Monahan

25 September 2017

Logic for Program Specification: Outline

- Logic
- Syntax of First Order Logic
- Propositional Logic
- Predicate Logic

Logic for Program Specification: Outline

- Logic
- Syntax of First Order Logic
- Propositional Logic
- Predicate Logic

What is logic?

Logic is the study of systems which are governed by a set of precise, well-defined rules

- A logic is *like* a programming language in that it is formal and exact.
- It is *unlike* a programming language in that it is not specifically directed at computers
- Programming languages describe how to “do” things. They give step-by-step instructions to a computer.
- Logic, on the other hand, describes things by stating their *properties*. It also provides rules for working out new facts about things, given some starting information.

Why logic?

Natural languages can be *ambiguous*.

That is, given a particular sentence we may interpret it in a number of different ways (or none at all) depending on how much we know about:

- the person who said it
- the situation to which it refers
- the general topic area
- local variations in the language
- .....

Logic is unambiguous

Logic is an unambiguous language for describing situations.

Thus we can, in certain situations, use it as an alternative to a natural language description. This is useful when:

- we want to tell something to a computer
- we wish to formally agree something with someone else (e.g. a contract)
- we are working in maths or some scientific area, where we need to be precise with proofs etc.

History of Logic

400BC - 1800AD: Logic is used for *philosophy*, as a way of telling correct arguments from incorrect ones.

Early 19th Century: Augustus de Morgan and George Boole revolutionise logic by using *algebra*, effectively bringing logic into the domain of mathematics.

Late 19th, early 20th Century: Gottlob Frege, Bertrand Russell, David Hilbert (and others) use logic as a basis for writing the axioms for the *foundations of mathematics*.

1936: Alan Turing, Alonzo Church and Emil Post independently discover *computability* while trying to solve the decision problem for logic.

Aristotle - the father of (European) logic

A **syllogism** is discourse in which, certain things being stated, something other than what is stated follows of necessity from their being so.

I mean by the last phrase that they produce the consequence, and by this, that no further term is required from without in order to make the consequence necessary.

- Aristotle (384-322 BC)

Example of Aristotle's Logic

First then take a universal negative with the terms A and B. If no B is A, neither can any A be B. For if some A (say C) were B, it would not be true that no B is A; for C is a B. But if every B is A then some A is B. For if no A were B, then no B could be A. But we assumed that every B is A.

Similarly too, if the premise is particular. For if some B is A, then some of the As must be B. For if none were, then no B would be A. But if some B is not A, there is no necessity that some of the As should not be B; e.g. let B stand for animal and A for man. Not every animal is a man; but every man is an animal.

Aristotle: Prior Analytics, Book I

George Boole

George Boole (1815-1864)

Boole's major contribution was to show how the basic operations of logic could be mapped onto algebra, thus incorporating logic into mathematics.

Boole was appointed Professor of Mathematics at Queen's College, Cork in 1849 and published *An investigation into the Laws of Thought, on Which are founded the Mathematical Theories of Logic and Probabilities* in 1854.

First Order Logic

Today we want to revise how to use first-order logic.

- You've met some logical operators and reasoning in your programming (Boolean algebra, de Morgan's Laws...).
- You've probably done some digital logic (gates, latches, counters, ...).
- We're (ultimately) interested in proving theorems in logic so that the proofs are *fully formal*, and can be checked by a computer.

Predicates

- The basic until of logic is the predicate, written like:

$$p(x_1, x_2, \ldots, x_n)$$

- This is like a boolean-valued function, returning “true” iff the relationship denoted by p hold for its arguments $x_1, x_2, \ldots, x_n$.
- The number of arguments (n in the above example) is fixed for any given predicate, and is called the arity of that predicate.
- Specific examples of arities include: nullary, unary, binary, ternary, ..., “n-ary”.

Binary Predicates

- *Binary* predicates are very common in maths - they can express a relationship between two numbers
- ... for example: is-equal-to, is-less-than, is-greater-than, ...

Binary Predicates

- *Binary* predicates are very common in maths - they can express a relationship between two numbers
- ... for example: is-equal-to, is-less-than, is-greater-than, ...

- These are so common that we often use an *infix* relational symbol rather than the *prefix* predicate symbol; thus:

instead of	we write
is-equal-to(x,y)	$x = y$
is-less-than(x,y)	$x < y$
is-greater-than(x,y)	$x > y$

Unary Predicates

- *Unary* predicates are also quite common, in maths and elsewhere
- ... for example: is-even, is-positive, is-prime, is-human, ...

Unary Predicates

- *Unary* predicates are also quite common, in maths and elsewhere
- ... for example: is-even, is-positive, is-prime, is-human, ...

- It's worth noting that every unary predicate corresponds directly to a set
- The set of all even numbers, of all positive numbers, of all prime numbers etc.
- Such a predicate is called the *characteristic predicate* of the set

Nullary Predicates

- *Nullary* predicates are the “trivial” case...
- Since there are no arguments, in logic we often omit writing the parentheses for them; thus we write $p, q, r, \ldots$ instead of $p(), q(), r(), \ldots$.

Nullary Predicates

- *Nullary* predicates are the “trivial” case...
- Since there are no arguments, in logic we often omit writing the parentheses for them; thus we write $p, q, r, \ldots$ instead of $p(), q(), r(), \ldots$.

- These evaluate to either true or false without reference to anything else - this they correspond to *boolean variables*
- In logic these are usually called propositions

Nullary Predicates

- *Nullary* predicates are the “trivial” case...
- Since there are no arguments, in logic we often omit writing the parentheses for them; thus we write $p, q, r, \ldots$ instead of $p(), q(), r(), \ldots$.
- These evaluate to either true or false without reference to anything else - this they correspond to *boolean variables*
- In logic these are usually called propositions

A proposition is just a special case of a predicate
- one that takes no arguments.

Logic for Program Specification: Outline

- Logic
- Syntax of First Order Logic
- Propositional Logic
- Predicate Logic

The operators

The main operators of first order logic are:

Symbol	Pronounced	Called
$\wedge$	and	conjunction
$\vee$	or	disjunction
$\neg$	not	negation
$\Rightarrow$	implies	implication
$\Leftrightarrow$	if-and-only-if	equivalence
$\forall$	for-all	universal quantifier
$\exists$	there-exists	existential quantifier

The operators

The main operators of first order logic are:

Symbol	Pronounced	Called
$\wedge$	and	conjunction
$\vee$	or	disjunction
$\neg$	not	negation
$\Rightarrow$	implies	implication
$\Leftrightarrow$	if-and-only-if	equivalence
$\forall$	for-all	universal quantifier
$\exists$	there-exists	existential quantifier

- $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$ are called connectives
- $\forall$, $\exists$ are called quantifiers

Semantics for the connectives

It is customary to describe the semantics of the five connectives by means of truth tables.

A	B	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
F	F	F	F	T	T
F	T	F	T	T	F
T	F	F	T	F	F
T	T	T	T	T	T

A	$\neg A$
F	T
T	F

Syntax for quantifiers: Types

- Our syntax for typed first-order logic is:
  - $\forall x : T \cdot P(x)$ to mean "for all x of type T, $p(x)$ holds"
  - $\exists x : T \cdot P(x)$ to mean "there exists an x of type T for which $p(x)$ holds"
- For simplicity, we will just regard types as being the names of sets

Sets

Typically we assume some given sets:

- $\mathbb{N}$, the set of natural numbers, $\{0, 1, 2, 3, \ldots\}$
- $\mathbb{Z}$, the set of integer numbers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
- $\mathbb{Q}$, the set of rational numbers (of the form a/b, for $a, b \in Z$)
- $\mathbb{R}$, the set of real numbers (includes all 'decimals')

Sets

Typically we assume some given sets:

- $\mathbb{N}$, the set of natural numbers, $\{0, 1, 2, 3, \ldots\}$
- $\mathbb{Z}$, the set of integer numbers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
- $\mathbb{Q}$, the set of rational numbers (of the form a/b, for $a, b \in Z$)
- $\mathbb{R}$, the set of real numbers (includes all 'decimals')

We form *new* sets either by:

- Enumeration: $Stooges \triangleq \{Larry, Curly, Moe\}$
- Comprehension: $Evens \triangleq \{x : \mathbb{N} \mid x \mod 2 = 0\}$

Sets

Typically we assume some given sets:

- $\mathbb{N}$, the set of natural numbers, $\{0, 1, 2, 3, \ldots\}$
- $\mathbb{Z}$, the set of integer numbers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
- $\mathbb{Q}$, the set of rational numbers (of the form a/b, for $a, b \in Z$)
- $\mathbb{R}$, the set of real numbers (includes all 'decimals')

We form *new* sets either by:

- Enumeration: $Stooges \triangleq \{Larry, Curly, Moe\}$
- Comprehension: $Evens \triangleq \{x : \mathbb{N} \mid x \mod 2 = 0\}$

We'll also assume the usual set-theoretic operations of union, intersection, difference, power-set, ...

Logic for Program Specification: Outline

- Logic
- Syntax of First Order Logic
- Propositional Logic
- Predicate Logic

Consider these arguments:

- If the train arrives late and there are no taxis at the station, then John is late for his meeting. John is not late for his meeting. The train did arrive late. *Therefore* there were taxis at the station.
- If it is raining and Jane does not have her umbrella with her, then she will get wet. Jane is not wet. It is raining. *Therefore* Jane has her umbrella with her.

Formalising these arguments

Both of these translate into the same argument; let

P	the train arrives late	it is raining
Q	there are taxis at the station	Jane has her umbrella with her
R	John is late for his meeting	Jane is wet

Then the argument becomes:

```
(P ∧ ~ Q) -> R
~ R
P
===================
Q
```

Proof: Notation and Terminology

- Notation:
  - $H_1, H_2, \ldots H_n \vdash G$
- Meaning: Whenever all of $H_1, H_2, \ldots H_n$ are true then so is G
- Each H_i is called a hypothesis (or a *premise*)
- G is called the goal (or the *conclusion*)

Proof: Notation and Terminology

Proof by natural deduction:

- We use elimination rules to break up the hypothesis
- We use introduction rules to form the goal
- We have an introduction and elimination rules for each of the connectives.

Logic for Program Specification: Outline

- Logic
- Syntax of First Order Logic
- Propositional Logic
- Predicate Logic

Motivation: why more logic?

Consider these arguments:

- If it is raining and Jane does not have her umbrella with her, then she will get wet. Jane is not wet. It is raining. *Therefore* Jane has her umbrella with her.

Motivation: why more logic?

Consider these arguments:

- If it is raining and Jane does not have her umbrella with her, then she will get wet. Jane is not wet. It is raining. *Therefore* Jane has her umbrella with her.

- If it is raining and a person does not have their umbrella with them, then they will get wet. Jane is not wet. It is raining. *Therefore* Jane has her umbrella with her.

Motivation: why more logic?

Consider these arguments:

- If it is raining and Jane does not have her umbrella with her, then she will get wet. Jane is not wet. It is raining. *Therefore* Jane has her umbrella with her.

- If it is raining and a person does not have their umbrella with them, then they will get wet. John is wet. Jane is not wet. It is raining. *Therefore* Jane has her umbrella with her.

Motivation: why more logic?

Consider these arguments:

- If it is raining and Jane does not have her umbrella with her, then she will get wet. Jane is not wet. It is raining. *Therefore* Jane has her umbrella with her.

- If it is raining and a person does not have their umbrella with them, then they will get wet. John has his cat with him. Jane is not wet. It is raining. *Therefore* Jane has her umbrella with her.

Predicate logic

Predicate logic is the logic for discussing things and relationships between things.

	Examples	Representation
things	the umbrella, a cat, Tom, Jane, ...	variables, constants
relationships	has, is-a, father-of, in-the-same-room-as, older-than, member-of, ...	predicates

Some examples

Assume some set P of people; then:

Everyone has a parent:	$(\forall x{:}P,\ (\exists y{:}P,\ \mathit{parent}\ x\ y))$
Everybody loves somebody:	$(\forall x{:}P,\ (\exists y{:}P,\ \mathit{loves}\ x\ y))$
Somebody loves everybody:	$(\exists x{:}P,\ (\forall y{:}P,\ \mathit{loves}\ x\ y))$
If one is bad, they all are:	$(\exists x{:}P,\ (\mathit{bad}\ x) \Rightarrow (\forall y{:}P,\ \mathit{bad}\ y))$
Everyone has *two* parents:	$(\forall x{:}P,\ (\exists y{:}P,\ (\exists z{:}P,\ (\mathit{parent}\ x\ y) \wedge (\mathit{parent}\ x\ z) \wedge (y \neq z))))$

Scope

- The proof rules for the quantifiers $\forall$ and $\exists$ involve manipulating *variables*.
- The important thing here is to make sure that you don't cause a variable to change its scope inappropriately.

Scope

- The proof rules for the quantifiers $\forall$ and $\exists$ involve manipulating *variables*.
- The important thing here is to make sure that you don't cause a variable to change its scope inappropriately.

- We write $i : S$ to denote the introduction of a variable i of type S into a section of a proof.
- In written proofs we also put a box around the lines where we use variable i so we can keep track of its scope.

Free and Bound variables

- Let ϕ be a formula in predicate logic.
- An occurrence of x in ϕ is free in ϕ if it is a leaf node in the parse tree of ϕ such that there is no path upwards from that node x to a node $\forall x$ or $\exists x$.
- Otherwise, that occurrence of x is called bound.

Free and Bound variables

- For $\forall x \phi$, we say that ϕ is the scope of $\forall x$.
- For $\exists x \phi$, we say that ϕ is the scope of $\exists x$.
- This definition of scope is always minus any of ϕ's subformulae $\forall x \psi$, or $\exists x \psi$.

Substitution

Substitution: Given a variable x, a term t and a formula ϕ we define $\phi[t/x]$ to be the formula obtained by replacing each free occurrence of variable x in ϕ with t.

Substitution Example

Let f be a function symbol with two arguments and ϕ the formula $\forall x((P(x) \rightarrow Q(x)) \wedge S(x, y))$

- $f(x, y)$ is a term and $\phi[f(x, y)/x]$ is ϕ i.e. there is no change.

This is because all occurrences of x are bound in ϕ, so none of them get substituted.

Natural Deduction Proof Rules for Predicate Logic

- Proofs are similar to those for propositional logic
- Proof rules are added for dealing with the quantifiers (introduction and elimination rules) and with the equality symbol.
- We overload the previously established proof rules for the propositional connectives $\wedge, \vee \ldots$ so any proof rule of propositional logic is still valid for logical formulas of predicate logic

Natural Deduction Proof Rules:Equality

- Equality is represented as a predicate with two arguments, which we write using the *infix* symbol "=".
- Its proof rules ensure that it represents the intended concept:

$$\frac{}{t = t} =_i \qquad\qquad \frac{t_1 = t_2 \qquad \phi[t_1/x]}{\phi[t_2/x]} =_e$$

Introduction Rule
(Identity)

Elimination Rule
(Leibniz Equality)

- These rules can only be used if t is a term (we cannot use them on formulae). Terms t_1 and t_2 have to be free for x in ϕ whevever we want to use the "=e" rule.