SE 2011
Review of Logic

Ron van der Meyden

University of New South Wales
Sydney, Australia

March 16, 2016

A specification example

Requirements R: A burglar alarm system for a house is to operate as follows. The alarm should not sound unless the system has been armed or there is a fire. If the system has been armed and a door is disturbed, the alarm should ring. Irrespective of whether the system has been armed, the alarm should go off when there is a fire.

Conclusion C: If the alarm is ringing and there is no fire, then the system must have been armed.

Questions:

- Will every system correctly implementing requirements R satisfy C?
- Is the final sentence of the requirements redundant?

Gottfried Wilhelm von Leibniz (1646-1716)

Leibniz' (1646-1716) Dream

a generalized mathematics, by which thinking could be replaced by calculation:

"If we had it, we should be able to reason in metaphysics and morals in much the same way as in geometry and analysis... If controversies were to arise, there would be no more need of disputation between two philosophers than between two accountants. For it would suffice to take their pencils in their hands, to sit down to their slates, and to say to each other (with a friend as witness, if they liked): let us calculate."

Leibniz' Dream modernised

a generalized mathematics, by which thinking could be replaced by *computation*

If we had it, we should be able to reason in *computer science and its applications areas* in much the same way as in geometry and analysis... If controversies were to arise, there would be no more need of disputation between two *computer scientists* than between two accountants. For it would suffice to *sit down at their laptops, enter the question at hand, and say to the machine: calculate this!*

The main relationship between symbols and the world of concern in logic is that of a *sentence of a language* being *true* in the world.
A sentence of a natural language (like English, Chinese, Warlpiri) is *declarative*, or a *proposition*, if it can be meaningfully be said to be either true or false.
Examples:

- Richard Nixon was president of Equador.
- The square root of 16 is 4.
- This program gets stuck in an infinite loop if you input 3.
- Whatever list of numbers you give as input to this program, it outputs the same list but in increasing order.
- $x^n + y^n = z^n$ has no nontrivial integer solutions for $n > 2$.

The following are *not* declarative sentences of English:

- Gubble gimble goo
- For Pete's sake, take out the garbage!
- Did you watch South Park last night?
- Please waive the prerequisites for this subject for me.

Declarative sentences in natural languages can be *compound* sentences, built out of other sentences.
Propositional Logic is a formal representation of some constructions for which the truth value of the compound sentence can be determined from the truth value of its components.

- Cook is a bit of a Romeo *and* Kenny is always getting killed.
- Either Bill is a liar *or* Hillary is innocent of Whitewater.
- *It is not the case that* this program always halts.

Not all constructions of natural language are truth-functional:

- *Hillary suspects that* Starr has the memo.
- *Cook said* they killed Kenny.
- This program always halts *because* it contains no loops.
- The disk crashed *after* I saved my file.

Connectives of Propositional Logic and their Truth Tables

$\wedge$ "and", "but", ";", ":"

$\vee$ "or", "either . . . or . . . "

$\neg$ "not", "it is not the case that"

A	B	A $\wedge$ B
F	F	F
F	T	F
T	F	F
T	T	T

A	B	A $\vee$ B
F	F	F
F	T	T
T	F	T
T	T	T

A	$\neg$ A
F	T
T	F

Somewhat more controversially, consider the following constructions:

- if A then B
- A only if B
- B if A
- A implies B
- it follows from A that B
- whenever A, B
- A is a sufficient condition for B
- B is a necessary condition for A

Each has the property that if true, and A is true, then B is true.

We can *approximate* the English meaning of these by "not (A and not B)", written $A \Rightarrow B$, which has the following truth table:

A	B	A $\Rightarrow$ B
F	F	T
F	T	T
T	F	F
T	T	T

While only an approximation to the English, 100+ years of experience have shown this to be adequate for capturing *mathematical reasoning*. (Moral: mathematical reasoning does not need all the features of English.)

Unless

A *unless* B can be approximated as $\neg B \Rightarrow A$

E.g. I go swimming unless it rains = If it is not raining I go swimming.

Correctness the translation is perhaps easier to see in:

I don't go swimming unless the sun shines = If the sun does not shine then I don't go swimming.

Note that "I go swimming unless it rains, but sometimes I swim even though it is raining" makes sense, so the translation of "A unless B" should not imply $B \Rightarrow \neg A$.

Propositional Logic as a Formal Language

A *string* is just a sequence of symbols.

Let $Prop = \{p, q, r, \ldots\}$ be a set of basic propositional letters.

The set of formulae of propositional logic is the smallest set such that

- If $x \in Prop$ then x is a formula,
- If ϕ is a formula then so is $\neg\phi$
- If ϕ and ψ are formulae, then so are $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \Rightarrow \psi)$, and $(\phi \Leftrightarrow \psi)$.

Convention: we often drop parentheses when there is no ambiguity. $\neg$ binds more tightly than $\wedge$ and $\vee$, which in turn bind more tightly than $\Rightarrow$ and $\Leftrightarrow$.

Applications of Propositional Logic I: Digital Circuits

A formula, can via its parse tree, be viewed as defining a digital circuit, which computes a boolean function of the input propositions. The function is given by the truth table of the formula.

Logical Equivalence

Two formulas ϕ, ψ are *logically equivalent*, denoted $\phi \equiv \psi$ if they have the same truth value for all values of their basic propositions.

Application: If ϕ and ψ are two formulae such that $\phi \equiv \psi$, then the circuits corresponding to ϕ and ψ compute the same function. Thus, proving equivalence of formulas can be used to optimize circuits.

Some well known equivalences:

Excluded Middle	$p \vee \neg p \equiv \top$
Contradiction	$p \wedge \neg p \equiv \bot$
Identity	$p \vee \bot \equiv p$
	$p \wedge \top \equiv p$
Domination	$p \vee \top \equiv \top$
	$p \wedge \bot \equiv \bot$
Idempotence	$p \vee p \equiv p$
	$p \wedge p \equiv p$
Double Negation	$\neg\neg p \equiv p$
Commutativity	$p \vee q \equiv q \vee p$
	$p \wedge q \equiv q \wedge p$

Associativity	$(p \vee q) \vee r \equiv p \vee (q \vee r)$
	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Distribution	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
De Morgan's laws	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
	$\neg(p \vee q) \equiv \neg p \wedge \neg q$
Implication	$p \Rightarrow q \equiv \neg p \vee q$
	$p \Leftrightarrow q \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$

Example:

$$
\begin{aligned}
&((r \wedge \neg p) \vee (r \wedge q)) \vee ((\neg r \wedge \neg p) \vee (\neg r \wedge q)) \\
&\quad \equiv \quad ((r \wedge (\neg p \vee q)) \vee (\neg r \wedge (\neg p \vee q) && \text{Assoc.} \\
&\quad \equiv \quad (r \vee \neg r) \wedge (\neg p \vee q) && \text{Assoc., Commut.} \\
&\quad \equiv \quad \top \wedge (\neg p \vee q) && \text{Excl. Mid.} \\
&\quad \equiv \quad (\neg p \vee q) && \text{Ident.}
\end{aligned}
$$

Examples

A formula is *satisfiable*, if it evaluates to T for some assignment of truth value to its basic propositions.

Example:

A	B	$\neg(A \Rightarrow B)$
F	F	F
F	T	F
T	F	T
T	T	F

Applications of Propositional Logic II: Constraint Satisfaction Problems

These are problems such as timetabling, activity planning, etc. Many can be understood as showing that a formula is satisfiable.

Example: You are planning a party, but your friends are a bit touchy about who will be there.

- If John comes, he will get very hostile if Sarah is there.
- Sarah will only come if Kim will be there also.
- Kim says she will not come unless John does.

Who can you invite without making someone unhappy?

Translation to logic: let $J, (S, K)$ represent "John (Sarah, Kim) comes to the party". Then the constraints are:

- $J \Rightarrow \neg S$
- $S \Rightarrow K$
- $K \Rightarrow J$

Thus, for a succesful party to be possible, we want the formula $\phi = (J \Rightarrow \neg S) \wedge (S \Rightarrow K) \wedge (K \Rightarrow J)$ to be satisfiable.
Truth values for J, S, K making this true are called *satisfying assignments.*

We figure out where the conjuncts are false, below.
(so blank = T)

J	K	S	$J \Rightarrow \neg S$	$S \Rightarrow K$	$K \Rightarrow J$	ϕ
F	F	F				
F	F	T		F		F
F	T	F			F	F
F	T	T			F	F
T	F	F				
T	F	T	F	F		F
T	T	F				
T	T	T	F			F

Conclusion: a party satisfying the constraints can be held.
Invite nobody, or invite John only, or invite Kim and John.

Arguments

An *argument* consists of a set of declarative sentences called *premises* and a declarative sentence called the *conclusion*.

Example:

Premises:	Frank took the Ford or the Toyota.
	If Frank took the Ford he will be late.
	Frank is not late.
Conclusion:	Frank took the Toyota

An argument is *valid* if the conclusions are true *whenever* all the premises are true. Thus: if we believe the premises, we should also believe the conclusion.
(Note: we don't care what happens when one of the premises is false.)
Other ways of saying the same thing:

- The conclusion *logically follows* from the premises.
- The conclusion is a *logical consequence* of the premises.
- The premises *entail* the conclusion.

The argument above is valid. The following is invalid:

Premises:	Frank took the Ford or the Toyota.
	If Frank took the Ford he will be late.
	Frank is late.
Conclusion:	Frank took the Ford.

For arguments in propositional logic, we can capture validity as follows:

Let $\phi_1, \ldots, \phi_n$ and ϕ be formulae of propositional logic. Draw a truth table with columns for each of $\phi_1, \ldots, \phi_n$ and ϕ. The argument with premises $\phi_1, \ldots, \phi_n$ and conclusion ϕ is valid, denoted $\phi_1, \ldots, \phi_n \models \phi$, if in every row of the truth table where $\phi_1, \ldots, \phi_n$ are all true, ϕ is true also.

we mark only true locations (blank = F)

Frd	$Tyta$	$Late$	$Frd \vee Tyta$	$Frd \Rightarrow Late$	$\neg Late$	$Tyta$
F	F	F		T	T	
F	F	T		T		
F	T	F	T	T	T	T
F	T	T	T	T		T
T	F	F	T		T	
T	F	T	T	T		
T	T	F	T		T	T
T	T	T	T	T		T

(This shows $Frd \vee Tyta$, $Frd \Rightarrow Late$, $\neg Late \models Tyta$)

The following row shows $Frd \vee Tyta, Frd \Rightarrow Late, Late \not\models Frd$

Frd	$Tyta$	$Late$	$Frd \vee Tyta$	$Frd \Rightarrow Late$	$Late$	Frd
F	T	T	T	T	T	F

Validity of formulae

A formula ϕ is *valid*, or a *tautology*, denoted $\models \phi$, if it evaluates to T for *all* assignments of truth value to its basic propositions.

Example:

A	B	$(A \Rightarrow B) \Rightarrow (\neg B \Rightarrow \neg A)$
F	F	T
F	T	T
T	F	T
T	T	T

Validity, Equivalence and Entailment

Theorem: The following are equivalent:

- $\phi_1, \ldots \phi_n \models \psi$
- $\models (\phi_1 \wedge \ldots \wedge \phi_n) \Rightarrow \psi$
- $\models \phi_1 \Rightarrow (\phi_2 \Rightarrow \ldots (\phi_n \Rightarrow \psi) \ldots)$

Theorem: $\phi \equiv \psi$ if and only if $\models \phi \Leftrightarrow \psi$

Applications of Propositional Logic III: Reasoning about Requirements/Specifications

Suppose a set of English language requirements R for a software/hardware system can be formalised by a set of formulae $\{\phi_1, \ldots \phi_n\}$.
Suppose C is a statement formalised by a formula ψ. Then

- The requirements cannot be implemented if $\phi_1 \wedge \ldots \wedge \phi_n$ is not satisfiable.
- If $\phi_1, \ldots \phi_n \models \psi$ then every correct implementation of the requirements R will be such that C is always true in the resulting system.
- If $\phi_1, \ldots \phi_{n-1} \models \phi_n$, then the condition ϕ_n of the specification is redundant and need not be stated in the specification.

Example III

Requirements R: A burglar alarm system for a house is to operate as follows. The alarm should not sound unless the system has been armed or there is a fire. If the system has been armed and a door is disturbed, the alarm should ring. Irrespective of whether the system has been armed, the alarm should go off when there is a fire.
Conclusion C: If the alarm is ringing and there is no fire, then the system must have been armed.

Questions

- Will every system correctly implementing requirements R satisfy C?
- Is the final sentence of the requirements redundant?

Expressing the requirements as formulas of propositional logic, with

- S = the alarm sounds = the alarm rings
- A = the system is armed
- D = a door is disturbed
- F = there is a fire

we get

Requirements:

- $S \Rightarrow (A \vee F)$
- $(A \wedge D) \Rightarrow S$
- $F \Rightarrow S$

Conclusion: $(S \wedge \neg F) \Rightarrow A$

The questions correspond to

- Does
  $S \Rightarrow (A \vee F),\ (A \wedge D) \Rightarrow S,\ F \Rightarrow S \models (S \wedge \neg F) \Rightarrow A$?
- Does $(S \Rightarrow (A \vee F)),\ (A \wedge D) \Rightarrow S \models F \Rightarrow S$?

Predicate Calculus

Declarative sentences in natural language have a lot more structure than we have captured so far. Moreover, this structure plays an essential role in intuitively valid arguments:

> *Margaret is John's supervisor.*
> *Thus, John has a supervisor.*

The argument is valid, but if we express this in logic as two assertions P, Q, we find that $P \not\models Q$. More is needed!

More examples ....

Dilbert is an employee. Every employee except the CEO has a supervisor. Dilbert is not the CEO. Thus, Dilbert has a supervisor.

Every employee's supervisor is out of his depth. Dilbert is an employee. Thus, Dilbert's supervisor is out of his depth.

Every pixel that is not contained in some window has the background colour. There is a pixel not contained in any window. Thus, there is a pixel that has the background colour.

All transactions in the database have been checked. Every transaction made yesterday is in the database. Thus, every transaction made yesterday has been checked.

What sorts of structure is there in these sentences that might account for the validity of these arguments?

- Names for objects: "John", "Margaret", "Dilbert"
- Complex expressions for objects: "Dilbert's supervisor"
- Properties of objects: "has a supervisor", "is in the database"
- Relationships between objects: "... is the manager of ...", "... is contained in ..."
- Quantifiers: "Every", "Somebody", "All", "there is"

Predicate Logic or *First Order Logic*, attempts to formalize these sorts of constructs.

Syntax of Predicate Logic, and Intuitive Semantics

Predicate Logic deals with some domain of discourse. *Terms* refer to objects in this domain.
Terms are constructed from:

- Constant terms: $a, b, c, \ldots$
  — these represent names, such as "Dilbert"
- Variables: $u, v, w, x, y, z \ldots$
  — these act as place holders for expressions referring to objects
- Function Symbols: $f, g, h, \ldots$
  — these represent functional operations such as "'s supervisor"
  — Each function symbol has an *arity*, i.e. the number of arguments it takes.

Terms are defined recursively: A terms is

- a constant c, or
- a variable x, or
- an expression $f(t_1, \ldots, t_n)$, where f is an n-ary function symbol and $t_1, \ldots, t_n$ are terms.

Examples:

1. `dilbert`
2. `supervisor(dilbert)`
3. `salary(dilbert)`
4. `sum(salary(dilbert),salary(supervisor(dilbert)))`
5. `sum(salary(x),salary(supervisor(x)))`

Intuitively,

1. refers to Dilbert
2. refers to Dilbert's supervisor
3. refers to Dilbert's salary
4. refers to the sum of Dilbert's salary and the salary of his supervisor
5. refers to the sum of the salary of the person referred to as "x" and the salary of the supervisor of that person. Once we decide what "x" refers to we can calculate this.

Predicates

The language of predicate logic also contains a set of predicate symbols: $P, Q, R, \ldots$

Each has an arity, i.e. a number of arguments that it takes.

Intuitively, a predicate symbol represents a relationship that may hold between its arguments.

Examples:

- `Stingy` of arity one
- `Supervises` of arity 2
- `Walked` of arity 3 (.. took .. for a walk at place ...)

An atomic formula of predicate logic has the form $P(t_1, \ldots, t_n)$, where P is a predicate symbol and $t_1, \ldots, t_n$ are terms.

Intuitively, this states that the relationship referred to by P applies to (is true of) the objects referred to by $t_1, \ldots, t_n$ (in that order).

Examples:

- `Stingy(supervisor(dilbert))`
- `Tiny(salary(dilbert))`
- `Supervises(snoopy, dilbert)`
- `Walked(charlie,snoopy,park)`
- `Supervises(snoopy, supervisor(x))`

Note that in the last case, `x` is a variable, so we can't determine a truth value until we say what this refers to.

Intuitively, these formulas state that

- Dilberts supervisor is stingy.
- Dilberts salary is tiny.
- Snoopy supervises Dilbert.
- Charlie took Snoopy for a walk in the park.
- Snoopy supervises the supervisor of x.

From the basis of atomic formulas, we can form more complex formulas using the operators of propositional logic.
Examples

- $\neg$`Supervises(snoopy, dilbert)`
  Snoopy does not supervise Dilbert.
- `Supervises(snoopy, dilbert)`$\Rightarrow$`Tiny(salary(dilbert))`
  (If Snoopy supervises Dilbert then Dilbert's salary is tiny.)
- `Walked(charlie,snoopy,park)`$\vee$`Walked(snoopy,charlie,park)`
  (Either Charlie took Snoopy for a walk in the park or Snoopy took Charlie for a walk in the park.)
- `Supervises(snoopy, dilbert)`$\Rightarrow$`Tiny(salary(x))`
  (If Snoopy supervises Dilbert then x's salary is tiny.)

Universal Quantifier

The universal quantifier "$\forall$" is used to capture expressions such as "All ... ." and "Every ... .".

Formally, if ϕ is a formula and x is a variable, then $\forall x(\phi)$ is a formula.

Intuitively, this states that ϕ holds for *every* possible value of the variable x.

Examples:

- Every project of Dilbert fails.
  $\forall x(\texttt{ProjectOf}(\texttt{Dilbert}, x) \Rightarrow \texttt{Fails}(x))$
- All employees have tiny salaries.
  $\forall x(\texttt{Employee}(x) \Rightarrow \texttt{Tiny}(\texttt{salary}(x)))$
- Every owner walks all their dogs in the park
  $\forall x(\texttt{Owner}(x) \Rightarrow \forall y(\texttt{DogOf}(x, y) \Rightarrow \texttt{Walks}(x, y, \texttt{park})))$
  Or ...
  $\forall x \forall y(\texttt{Owns}(x, y) \Rightarrow \texttt{Walks}(x, y, \texttt{park}))$

Existential Quantifier

The quantifier "$\exists$" is used to capture expressions such as "Some... ...." "There is ... that ...." and "There exists... that ....".

Formally, if ϕ is a formula and x is a variable, then $\exists x(\phi)$ is a formula.

Intuitively, this states that ϕ holds for *some* possible value of the variable x.

Examples:

- Some project of Dilbert fails.
  $\exists x(\texttt{ProjectOf}(\texttt{dilbert}, x) \wedge \texttt{Fails}(x))$
- There is an employee who has a tiny salary.
  $\exists x(\texttt{Employee}(x) \wedge \texttt{Tiny}(\texttt{salary}(x)))$
- Someone owns a dog and walks it in the park
  $\exists x(\texttt{Owner}(x) \wedge \exists y(\texttt{DogOf}(x, y) \wedge \texttt{Walks}(x, y, \texttt{park})))$
  Or ...
  $\exists x \exists y(\texttt{Owns}(x, y) \wedge \texttt{Dog}(y) \wedge \texttt{Walks}(x, y, \texttt{park}))$
- Someone walks all their dogs in the park.
  $\exists x(\texttt{Person}(x) \wedge \forall y(\texttt{DogOf}(x, y) \Rightarrow \texttt{Walks}(x, y, \texttt{park})))$

Note the general pattern for these translations:

- "All A's B" is translated as $\forall x(A(x) \Rightarrow B(x))$
- "Some A B's" is translated as $\exists x(A(x) \wedge B(x))$