Logic in Computer Science

The design is [...] to investigate the fundamental laws of the operations of the mind by which reasoning is performed; to give expression to them in the symbolic language of a calculus, and upon this foundation to establish the science of logic ...

George Boole, *An Investigation of the Laws of Thought*, 1854

It is reasonable to expect that the relationship between computer science and mathematical logic will be as fruitful in the next century as that between physics and analysis in the last.

John McCarthy, 1963

Logic

Logic deals with the formalization of natural language and reasoning methods.

A variety of logical systems have been developed, including

propositional logic,
predicate logic,
temporal logics, and
modal logics.

In this course we will discuss several logical systems relevant for applications to computing, with an emphasis on their computational aspects.

Typical applications of logic in computing include

logic programming,
automated verification, and
reasoning about knowledge

We will begin with a brief review of propositional logic.

From the Files of Inspector Craig

The following facts are known about a robbery:

1. If A is guilty and B is innocent, then C is guilty.

2. C never works alone.

3. A never works with C.

4. No one other than A, B, or C was involved, and at least one of them is guilty.

Can one infer from these facts who is guilty and who is innocent?

The Case of McGregor's Shop

Mr. McGregor phoned Scotland Yard that his shop had been robbed. Three suspects A, B, C were rounded up for questioning and the following facts were established:

1. Each of A, B, C had been in the shop on the day of the robbery, and no one else had been in the shop that day.

2. If A was guilty, then he had exactly one accomplice.

3. If B is innocent, so is C.

4. If exactly two are guilty, then A is one of them.

5. If C is innocent, so is B.

Whom did Inspector Craig indict?

For other cases see *What Is the Name of This Book?* by Raymond Smullyan.

Fundamental Notions in Logic

The above examples can be formalized in *propositional logic*, a system based on well-known logical connectives, such as negation, conjunction, disjunction, and implication. Most applications of logic to computing require richer logical languages with special logical operators.

Some of the key questions in the study of logical systems are:

> When is a given logical formula true? (*Validity*)
>
> Do given assumptions logically imply a given formula? (*Logical consequence*)
>
> How can we deduce a desired conclusion from given axioms? (*Provability*)

The relationship between the concepts of *truth* and *proof* within specified a logical system often plays a central role.

Propositional Logic

Propositional logic is a formal system in which the basic units are *propositions*. These represent statements and can be combined via *logical connectives* into more complex propositions.

The basic assumption is that

> *each proposition is either true or false* (but not both).

Simple propositions are denoted by *(propositional) variables* or by constants representing true and false.

The connectives used to form more complex propositions include negation ($\neg$, read "not"), conjunction ($\wedge$, read "and"), disjunction ($\vee$, read "or"), and implication ($\rightarrow$, read "implies").

Syntax of Propositional Logic

The syntax of propositional formulas is specified by the following rules:

$$
\begin{array}{lll}
\langle \text{proposition} \rangle & ::= & \bot \mid \top \mid \langle \text{variable} \rangle \\
& & \mid (\neg \langle \text{proposition} \rangle) \\
& & \mid (\langle \text{proposition} \rangle \wedge \langle \text{proposition} \rangle) \\
& & \mid (\langle \text{proposition} \rangle \vee \langle \text{proposition} \rangle) \\
& & \mid (\langle \text{proposition} \rangle \rightarrow \langle \text{proposition} \rangle) \\
\langle \text{variable} \rangle & ::= & P \mid Q \mid R \mid \ldots
\end{array}
$$

We use the letters α and β to denote propositional formulas.

Other common connectives are exclusive disjunction ($\oplus$, read "either-or") and biconditional ($\leftrightarrow$, read "if and only if").

Parentheses are often omitted to increase readability (provided the intended expression remains unambiguous).

The notion of a *subformula* can be defined in the expected way. For example, P and $(P \rightarrow Q)$ are both subformulas of $(\neg(P \rightarrow Q))$.

Semantics of Propositional Logic

The constants $\top$ and $\bot$ are also called *truth values* and represent *truth* and *falsity*, respectively.

The semantics of propositional logic formulas rests on so-called *truth functions* for the logical connectives.

Traditionally truth functions are given by way of truth tables, though they can also be defined by suitable identities:

$$\begin{array}{lcl} \neg\top & \approx & \bot \\ \neg\bot & \approx & \top \end{array} \qquad \begin{array}{lcl} \top \rightarrow \top & \approx & \top \\ \top \rightarrow \bot & \approx & \bot \\ \bot \rightarrow \top & \approx & \top \\ \bot \rightarrow \bot & \approx & \top \end{array}$$

$$\begin{array}{lcl} \top \wedge \top & \approx & \top \\ \top \wedge \bot & \approx & \bot \\ \bot \wedge \top & \approx & \bot \\ \bot \wedge \bot & \approx & \bot \end{array} \qquad \begin{array}{lcl} \top \vee \top & \approx & \top \\ \top \vee \bot & \approx & \top \\ \bot \vee \top & \approx & \top \\ \bot \vee \bot & \approx & \bot \end{array}$$

Formally, these identities define an equivalence (in fact, a *congruence*) relation on propositional formulas. We define:

$$\alpha \approx \beta$$

if, and only if, β can be obtained from α by repeatedly using the above identities to replace a subformula matching a right-hand side by the corresponding left-hand side.

For example,

$$(\top \vee \bot) \rightarrow \bot \approx \top \rightarrow \bot \approx \bot.$$

If a propositional formula contains no variables it is equivalent either to $\top$ or $\bot$ (but not both).

One of Inspector Craig's Cases

The known facts can be represented by the formulas

1. $P \wedge \neg Q \rightarrow R$
2. $R \rightarrow P \vee Q$
3. $P \rightarrow \neg R$
4. $P \vee Q \vee R$

where P represents the statement "A is guilty," Q the statement "B is guilty," and R the statement "C is guilty."

Let α be the conjunction of the above four formulas. We get the following truth table for α:

P	Q	R	α
$\bot$	$\bot$	$\bot$	$\bot$
$\bot$	$\bot$	$\top$	$\bot$
$\bot$	$\top$	$\bot$	$\top$
$\bot$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\bot$
$\top$	$\bot$	$\top$	$\bot$
$\top$	$\top$	$\bot$	$\top$
$\top$	$\top$	$\top$	$\bot$

Truth Valuations

A (truth) *valuation* is a mapping from propositiona) variables to truth values.

It is usually sufficient to consider truth valuations with a *finite* domain; various notations are used to denote such mappings, e.g.,

$$[P \mapsto \top, Q \mapsto \top, R \mapsto \bot]$$

or

$$[\top/P, \top/Q, \bot/R].$$

If $\alpha = \alpha(P_1, \ldots, P_n)$ is a formula containing variables $P_1, \ldots, P_n$, and σ is a truth valuation, then by $\alpha\sigma$ we denote the result of replacing in α each occurrence of a variable P_i by its truth value, as specified by σ.

Propositional Equivalence

Two propositional formulas α and β are said to be *(propositionally) equivalent*, written $\alpha \sim \beta$, if and only if $\alpha\sigma \approx \beta\sigma$, for all truth valuations σ whose domain includes all variables occurring in α or β.

Basically, one may check equivalence of propositional formulas by inspecting truth tables.

Examples.

$$\begin{array}{rcl} \neg P \to Q & \sim & P \vee Q \\ P \wedge \neg P & \sim & \neg(P \to P) \\ P \to P & \sim & \top \end{array}$$

Note that $\sim$ extends $\approx$ in the sense that for all variable-free propositional formulas α and β, we have $\alpha \sim \beta$ if, and only if, $\alpha \approx \beta$.

Some Basic Equivalences

$$
\begin{array}{rcl}
P \vee P & \sim & P \\
P \wedge P & \sim & P \\
P \vee Q & \sim & Q \vee P \\
P \wedge Q & \sim & Q \wedge P \\
P \wedge (P \vee Q) & \sim & P \\
P \wedge (Q \vee Z) & \sim & (P \wedge Q) \vee (P \wedge Z) \\
P \vee \neg P & \sim & \top \\
P \wedge \neg P & \sim & \bot \\
\neg\neg P & \sim & P \\
P \vee \top & \sim & \top \\
P \wedge \top & \sim & P \\
P \vee \bot & \sim & P \\
P \wedge \bot & \sim & \bot \\
\neg(P \vee Q) & \sim & \neg P \wedge \neg Q \\
\neg(P \wedge Q) & \sim & \neg P \vee \neg Q \\
P \rightarrow Q & \sim & \neg P \vee Q
\end{array}
$$

Substitution

Valuations are a special kind of substitutions. In general, by a *(propositional) substitution* we mean a mapping from (propositional) variables to (propositional) formulas.

We will use the letters σ and τ to denote substitutions, and write $\alpha\sigma$ to denote the result of applying the substitution σ to the formula α.

Note that applying a substitution means to *simultaneously* replace all occurrences of variables by the indicated formulas.

For example, if

$$\sigma = [P \mapsto P \wedge Q, Q \mapsto \neg R]$$

then $((P \vee Q) \rightarrow P)\sigma$ is $((P \wedge Q) \vee \neg R) \rightarrow (P \wedge Q)$.

Substitution Theorem.

> For all propositional formulas α and β and propositional substitutions σ, if $\alpha \sim \beta$, then $\alpha\sigma \sim \beta\sigma$.

Replacement

We write $\alpha[\beta]$ to indicate that β occurs as a subformula of α, and (ambiguously) denote by $\alpha[\beta']$ the result of *replacing* a particular occurrence of β in α by β'.

If necessary, one indicates the occurrence by writing $\alpha[\beta]_p$, where p specifies the position of the subformula, e.g., in Dewey decimal notation. (The subformula of α at position p is often denoted by $\alpha|_p$.)

For example, if α is $(P \wedge Q) \vee R$, then $\alpha_{1.2}$ is Q and $\alpha[P]_{1.2}$ is $(P \wedge P) \vee R$.

Replacement Theorem.

> If α, β and β' are propositional formulas with $\beta \sim \beta'$, and p is a position in α, then $\alpha[\beta]_p \sim \alpha[\beta']_p$.

Tautologies and Contradictions

A propositional formula α is said to be *satisfiable* if $\alpha\sigma \approx \top$, for some truth valuation σ.

A propositional formula α is called a *tautology* if it always evaluates to true, i.e., if $\alpha\sigma \approx \top$ for every truth valuation σ whose domain contains all variables occurring in α.

Similarly, α is called a *contradiction* (or *unsatisfiable*) if it always evaluates to false.

For example, $P \vee \neg P$ is a tautology, whereas $P \wedge \neg P$ is a contradiction.

Theorem. [Tautology and contradiction]

> A propositional formula α is a tautology if and only if its negation $\neg\alpha$ is a contradiction.

Theorem. [Tautology and equivalence]

> Two propositional formulas α and β are logically equivalent if and only if the formula $(\alpha \rightarrow \beta| \wedge (\beta \rightarrow \alpha)$ is a tautology.

Logical Consequence

A (propositional) formula α is called a *logical consequence* of a set of formulas N, written

$$N \models \alpha$$

if α is true for every valuation σ under which each formula in N is true.

For instance, α is a logical consequence of a finite set $N = \{\alpha_1, \ldots, \alpha_n\}$ if

$$\alpha\sigma \approx \top \text{ whenever } \alpha_1\sigma \approx \cdots \approx \alpha_n\sigma \approx \top.$$

Theorem [Tautology and logical consequence]

> A formula α is a logical consequence of $\alpha_1, \ldots, \alpha_n$ if, and only if, the implication
>
> $$\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \alpha$$
>
> is a tautology.

We call a set of formulas N *satisfiable* if there is a valuation σ that makes each formula α in N true. A set of formulas is *unsatisfiable* if it is not satisfiable.

Theorem [Logical consequence and unsatisfiability]

> A propositional formula α is a logical consequence of a set of propositional formulas N if, and only if the set $N \cup \{\neg\alpha\}$ is unsatisfiable.

Adequacy

The logical system we have discussed so far is based on a few selected connectives. Suppose we extend the language by introducing additional logical connectives; for instance, an (ternary) *if-then-else* operator.

Does this extension increase the expressiveness of our system? That is, can we express statements that could not have been expressed (in equivalent form) before?

It turns out that the three connectives $\neg$, $\wedge$ and $\vee$ form what is called an *adequate* set, in that every other propositional operator can be expressed in terms of these three.

For instance,

$$\text{if } P \text{ then } Q \text{ else } R \approx (\neg P \vee Q) \wedge (P \vee R).$$

The sets $\{\neg, \wedge\}$ and $\{\neg, \vee\}$ are adequate as well, as is the set consisting of a single connective, the *Sheffer stroke*, which is defined by:

$$\begin{array}{lcl} \top|\top & \approx & \bot \\ \top|\bot & \approx & \top \\ \bot|\top & \approx & \top \\ \bot|\bot & \approx & \top \end{array}$$

Proving Inadequacy

To prove that a set of connectives is *not* adequate, one essentially has to prove that some standard connective cannot be expressed by its connectives.

For example, the set $\{\neg\}$ is not adequate. A formula that uses only negation and two variables, say P and Q, is equivalent to P or $\neg P$, or Q or $\neg Q$; hence none of the standard binary connectives can be expressed with just negation.

The set $\{\wedge\}$ is also not adequate. If $\alpha = \alpha(P)$ is a formula that uses only conjunction, then

$$\alpha[P \mapsto \bot] \approx \bot,$$

so that negation cannot be expressed via conjunction.

Is the ternary connective below adequate?

$$\begin{array}{lcl}
\mathcal{C}(\top,\top,\top) & \approx & \top \\
\mathcal{C}(\top,\top,\bot) & \approx & \top \\
\mathcal{C}(\top,\bot,\top) & \approx & \bot \\
\mathcal{C}(\top,\bot,\bot) & \approx & \bot \\
\mathcal{C}(\bot,\top,\top) & \approx & \top \\
\mathcal{C}(\bot,\top,\bot) & \approx & \bot \\
\mathcal{C}(\bot,\bot,\top) & \approx & \top \\
\mathcal{C}(\bot,\bot,\bot) & \approx & \bot
\end{array}$$

The Satisfiability Problem: Computational Aspects

The satisfiability problem and the tautology problem are computationally hard problems, and are relevant to the famous $P = NP$ question.

The satisfiability problem is in the class NP: one can check in polynomial time (in terms of the size of the given formulas) whether a given truth assignment satisfies the formulas, so that there is a nondeterministic polynomial-time procedure for solving the problem.

No polynomial-time algorithm for the satisfiability is known, though. In fact the problem is NP-complete, so that the existence of a polynomial-time algorithm would imply that $P = NP$ (i.e., if the satisfiability problem can be solved in polynomial time, then all problems in NP have a polynomial-time algorithm).

An Example

Is the implication $M \rightarrow \neg F$ a logical consequence of the following formulas?

$$
\begin{aligned}
A \wedge B \wedge C &\rightarrow D && (1)\\
\neg A \wedge M &\rightarrow L && (2)\\
F \wedge E &\rightarrow \neg D && (3)\\
G \wedge M &\rightarrow C && (4)\\
\neg B \wedge F &\rightarrow \neg H && (5)\\
\neg D \wedge B \wedge E &\rightarrow G && (6)\\
M \wedge \neg I &\rightarrow J && (7)\\
H \wedge M &\rightarrow K && (8)\\
K \wedge J \wedge \neg L &\rightarrow E && (9)\\
\neg H \wedge F &\rightarrow L && (10)\\
M \wedge L &\rightarrow \neg F && (11)\\
K \wedge I \wedge A &\rightarrow E && (12)
\end{aligned}
$$

Checking validity via the corresponding truth table is possible, but rather time consuming, considering that the table has $2^{13} = 8192$ rows.