Lecture 1

August 26, 2005

1 Logic in Computer Science

Historically logic developed within the domains of mathematics and philosophy, but in this class we will concentrate on the application of logic in computer science. Throughout the course we will make references to areas that use logic heavily. These include **Circuit Design** (also known as Digital Logic), **Computability** (a mathematical investigation of what can and cannot be computed) and **Complexity Theory** (a refinement of computability–what is feasibly computable). In addition, logic is also used in Database Theory and Relational Databases, Design Verification, Security (e.g. proof carrying code), Programming languages and semantics, AI reasoning and common sense reasoning, software engineering (providing specifications to programmers), and more.

2 Basics of Propositional Logic

One of the key ideas of mathematics is that of an abstraction. Numbers allow us to abstract away the "what" of a problem and concentrate on the arithmetic. Similarly, in logic we abstract away some of the details and concentrate on the underlying form. Two millennia ago Aristotle realized that truth follows from form. For example, consider the following syllogism:

All *humans are mortal.*
All *Greeks are humans.*
Therefore all *Greeks are mortal.*

While all sentences in this example have a clear meaning, this need not be so. The following argument is also valid, although meaningless:

All *Bojums are slithy.*
All *Toves are Bojums.*
Therefore all *Toves are slithy.*

Both examples share the same form. If we replace "humans" by p, "mortal" by q, and "Greeks" by r we expose the form of the syllogism:

If p ***then*** q.
If q ***then*** r.
Therefore If p ***then*** r.

This syllogism has a name: transitivity of implication.

p and q above are called symbolic variables. A **symbolic variable** reflects some underlying quantity which is, at this point, unknown. The possible values of these variables are the truth values. Symbolic variables obey the **The Law of the Excluded Middle** which says that variables are either true or false, and there is nothing in between. This is a very sharp world. Consider the sand heap paradox. Start with a large heap of sand. Take the sand away one grain at a time. Eventually the large heap of sand will stop being a large heap of sand, but we will not be able to tell precisely when. This world true and false is idealized. We abstract away the gray cases.

This abstraction corresponds to the digital abstraction in electrical engineering. In electrical engineering voltages above some threshold indicate 1, and below another threshold indicate 0. One might say computer science begins when you have a circuit you can pretend is digital.

These symbolic variables are propositions. Assertions of truth about the world. In the first part of the course we will regard propositions as indivisible units without intrinsic structure. This paradigm is called **propositional logic**, also known as **sentential logic**.

In order to create more interesting formulas we use **propositional connectives**. The connectives can be classified according to the number of proposition that they apply to:

- *Unary:* **not** $\neg$
- *Binary:* **and** $\wedge$, **or** $\vee$, **implies** $\rightarrow$, **equivalent** $\leftrightarrow$
- *Ternary:* **if-then-else**

As a shorthand, sometimes we will use $\circ$ to indicate any binary connective.

The set of atomic propositions is called PROP. PROP, the set of connectives and the left **'('** and right parentheses **')'** form the alphabet of propositional logic. Using this alphabet we now can write strings of alphabetic symbols, or **expressions**. For example:

$(p \wedge q \rightarrow r)$
$)q \rightarrow (r\wedge$

Intuitively, there is something wrong with the second expression, but it still fits the definition of an expression. We will use the term **formulas** to indicate expressions with the "right" form. This intuitive notion can be formalized as following:

Definition 1 *The set of formulas* FORM *is the smallest set with the following properties:*

1. PROP $\subseteq$ FORM

2. *If* $\varphi \in$ FORM*, then* $(\neg\varphi) \in$ FORM
   If φ *and* $\psi \in$ FORM*, then* $(\varphi \circ \psi) \in$ FORM.

The second property in the definition is called the **closure** property. Also note that in some texts the formulas are called **well-formed formulas** or **wff**.

In order to be sure that FORM is well defined, we need to show that there exists a unique smallest set that has the above properties. This task is the subject of the next lemma.

Lemma 1 (Smallest set) *There is a smallest set of expressions that contains* PROP *and satisfies closure.*

Proof: Let FORM' $= \bigcap \{\ P : \text{PROP} \subseteq P$ and P satisfies closure $\}$.
Claim: FORM' is the smallest set with these two properties (closure and containing PROP).

1. PROP $\subseteq$ FORM$'$: PROP is a subset of FORM' because FORM' is the intersection of sets that contain PROP.

   FORM' satisfies closure: Suppose that φ and $\psi \in$ FORM$'$. From the way we constructed FORM' follows that φ and ψ are in all expression sets P that satisfy closure and contain PROP. In particular, closure implies that $(\neg\varphi)$ and $(\varphi \circ \psi)$ must also be in all these sets P, and consequently, also in their intersection. This means that $(\neg\varphi)$ and $(\varphi \circ \psi)$ are in FORM', therefore FORM' satisfies closure.

2. FORM' is the smallest set with these properties: To show this, we need to show that FORM' is a subset of every set that has the two properties. To that end, suppose that P is a set that contains PROP and satisfies closure. From the way we constructed FORM' follows that P is one of the sets in the intersection, and the intersection of two or more sets is always a subset of them all. The intersection is FORM', therefore FORM$' \subseteq P$.

$\Diamond\Diamond\Diamond$

Thus we have shown that there is a smallest set (namely FORM') that satisfies closure and contains PROP. Therefore FORM = FORM$'$.

Next lecture we will create an inductive definition, show it is equivalent to the above definition, and demonstrate the benefits of induction.

Typeset with $\LaTeX\,2_\varepsilon$