Lecture 5

September 7, 2005

1 Lecture Overview

The lecture started with stating that there are two aspects or levels of language (or logic) - *syntax* and *semantics*. While syntax deals with the form or structure of the language, it is semantics that adds meaning to the form.

In the last lecture we introduced the *philosopher's perspective* and *electrical engineer's perspective* of semantics associated with formulas (in propositional logic). In this lecture we finish our discussion of semantics by introducing the *software engineer's perspective*.

Recall the philosopher's view: $\models \subseteq 2^{Prop} \times Form$ is a binary relation, where the left side is a truth assignment and the right side is a formula. So, for example, we can say:

1. $\tau \models (\varphi_1 \wedge \varphi_2)$ iff $\tau \models \varphi_1$ and $\tau \models \varphi_2$
2. $\tau \models (\varphi_1 \vee \varphi_2)$ iff $\tau \models \varphi_1$ or $\tau \models \varphi_2$

An important feature of this definition is the *compositionality* of $\models$. An example is $\tau \models (\varphi \wedge \psi)$. We only need to know about $\tau \models \varphi$ and $\tau \models \psi$ in order to reason about the original conjunction, and both of these subformulas are independent of each other. This sort of non-ambiguity does not exist in the English language. For example, if someone says "I know that I am cold", parsing the statement unambigiuously depends on knowing who is speaking.

Also recall the second point of view, the electrical engineer's perspective: $\tau : Prop \rightarrow \{0, 1\}$. In other words, the electrical engineer maps each wire to voltage. The logical connectives are interpreted as gates (AND-gate, OR-gate, Inverter, etc), and the formulas are interpreted as circuits. .

Following this brief overview, the lecture concentrated on the software engineer's view of logic and a brief overview of complexity theory.

2 Semantics

For completeness, we review the definitions of the *semantical domain* and *semantics*. The *semantical domain* (also know as *domain of discourse*) is the set 2^{Prop} where *Prop* is the set of all atomic propositions. And, *semantics* is the relation between $Form$, the set of all formulas and 2^{Prop}. We call this relation *satisfaction*, represented by $\models$, where $\models \subseteq 2^{Prop} \times Form$.

3 Software Engineer's View

Consider a snippet of code: sqrt(x). The underlying assumption is that x must be greater than 0. In other words, the code makes sense given the condition that $x \geq 0$, or in the worlds where $x \geq 0$. We can say that each program *defines* a set of inputs: precisely those inputs that are acceptable by the program.

Similarly, a logical formula defines a set of assignments: precisely those assignments that make the formula true. This mapping from formula to sets of truth assignments is called $models$, where $models : Form \rightarrow 2^{2^{Prop}}$. This idea is formalized in the following definition:

Definition 1 (Models) *Let φ be a formula. We define $models(\varphi)$ as follows:*

1. $\varphi = p : models(p) = \{\tau : \tau(p) = 1\}$, *where* $p \in Prop$
2. $\varphi = (\neg\theta) : models(\neg\theta) = 2^{Prop} - models(\theta)$
3. $\varphi = (\theta \wedge \psi) : models(\theta \wedge \psi) = models(\theta) \cap models(\psi)$
4. $\varphi = (\theta \vee \psi) : models(\theta \vee \psi) = models(\theta) \cup models(\psi)$
5. $\varphi = (\theta \rightarrow \psi) : models(\theta \rightarrow \psi) = (2^{Prop} - models(\theta)) \cup models(\psi)$
6. $\varphi = (\theta \leftrightarrow \psi) : models(\theta \leftrightarrow \psi) = ((2^{Prop} - models(\theta)) \cap (2^{Prop} - models(\psi))) \cup (models(\theta) \cap models(\psi))$

It is not immediately clear whether the above definition captures the property that motivated it. In the next lemma we prove that $models(\varphi)$ is precisely the set of assignments for which φ is true.

Lemma 1 *Let $\varphi \in Form$ and $\tau \in 2^{Prop}$. The following two statements hold:*

1. $\varphi(\tau) = 1$ *iff* $\tau \in models(\varphi)$
2. $models(\varphi) = \{\tau : \varphi(\tau) = 1\}$.

Proof: By structural induction:

1. Base case: By definition, $\tau \in models(p)$ iff $\tau(p) = 1$ iff $p(\tau)$
2. Inductive Step:
   - If φ is $(\neg\psi)$: $\tau \in models(\neg\psi)$ iff $\tau \notin models(\psi)$ iff $\psi(\tau) = 0$ (inductive hypothesis used) iff $\neg(\psi(\tau)) = 1$ iff $(\neg\psi)(\tau) = 1$.
   - If φ is $(\theta \vee \psi)$: $\tau \in models(\theta \vee \psi)$ iff $\tau \in models(\theta \cup \psi)$ iff $\tau \in models(\theta)$ or $\tau \in models(\psi)$ iff $\theta(\tau) = 1$ or $\psi(\tau) = 1$ (by inductive hypothesis) iff $(\theta \vee \psi)(\tau) = 1$

- The cases with the other binary connectives are equivalent.

∎

Note that φ is the characteristic function of the set $models(\varphi)$. (Recall that the characteristic function of a set is the function that "determines" which elements belong to the set).

4 Relevance Lemma

Consider the formula $\varphi = ((p \vee q) \rightarrow r)$, and the set of atomic propositions $\{p, q, r, s\}$. The value of the proposition s in any truth assignment has no effect on the truth value of the formula φ. In other words, *propositions that do not occur in a formula have no impact on its truth value.*

We would like to state this formally, and to that end we formalize the notion of *occurrence* of a proposition in a formula.

Definition 2 (AP(φ)) *For $\varphi \in Form$, the set $AP(\varphi)$ of atomic propositions that occur in φ is defined as follows:*

1. $AP(p) = p, where\ p \in Prop.$
2. $AP((\neg\theta)) = AP(\theta), where\ \theta \in Form.$
3. $AP((\theta \circ \psi)) = AP(\theta) \cup AP(\psi), where\ \theta, \psi \in Form.$

Definition 3 (Agreement of Assignments) *Let $X \subseteq Prop$, and τ_1 and let τ_2 be truth assignments. Then τ_1 and τ_2* agree *on X if $\tau_1(p) = \tau_2(p)$ for all $p \in X$. We denote this symbolically as $\tau_1 |_X = \tau_2 |_X$.*

We are now ready to state and prove the informal result from above.

Lemma 2 (Relevance lemma) *Let $\varphi \in Form$ and let $\tau_1, \tau_2 \in 2^{Prop}$. If τ_1 and τ_2 agree on $AP(\varphi)$, then $\varphi(\tau_1) = \varphi(\tau_2)$.*

Proof: (by induction)

- If $\varphi = p$, where $p \in Prop$:
  $\varphi(\tau_1) = \tau_1(p) = \tau_2(p) = \varphi(\tau_2)$.

- $\varphi = (\neg\theta)$:
  $\varphi(\tau_1) = \neg(\theta(\tau_1)) = \neg(\theta(\tau_2))(IH) = \varphi(\tau_2)$.

- $\theta = (\theta \circ \psi)$:
  $\varphi(\tau_1) = \circ(\theta(\tau_1), \psi(\tau_1))(\text{IH}) = \circ(\theta(\tau_2), \psi(\tau_2)) = \varphi(\tau_2)$.

$\blacksquare$

The importance of the Relevance Lemma will become apparent in the next few lectures when we discus satisfiability. Instead of talking about the infinite set of truth assignments, it allows us to focus on the finite set of propositions that occur in a logical formula.

5 Truth Evaluation

Parsing checks the syntactical correctness of an expression. *Truth evaluation* is the task of computing the semantical meaning of a formula.

Now, given a formula $\varphi \in Form$ and a truth assignment $\tau \in 2^{Prop}$, we want to design an algorithm $Eval$ such that $Eval(\varphi, \tau) = 1$ iff $\varphi(\tau) = 1$, that is, we want an algorithm that checks whether a given truth assignment satisfies a given formula.

Now for any algorithm we design we have to check two things:

1. Correctness - whether the algorithm meets our design intent.

2. Cost - how much space and time does the algorithm consume. Furthermore, the cost can be given in many ways. One way is worst case which gives us a bound on how the algorithm will perform on any input. Another way is average case, which gives us the average time that an algorithm will take given multiple inputs.

With this in mind, in the next lecture we will discus *computational complexity theory.*