Lecture 21

October 24, 2005

In the last lecture we saw an introduction to first order logic, discussing both its syntax and semantics. After defining semantics, the reader may have wondered if quantified variables (i.e. those that appear next to the $\forall$ and $\exists$ quantifiers) should be distinguished from the other, non-quantified variables in a formula. This can be helpful to prove some results and so we will start by differentiating them.

The central topic of this lecture, however, is to state and prove a "relevance lemma" for first order logic. After this, we will see how one can describe the function of a circuit using first order logic.

1 Distinction between free and bound variables

It is clear that variables used in quantifiers are different from the other variables in that the quantifier prevents the assignment from affecting it.
For example, consider the formula:

$$(\exists x)(\exists x)p(x)$$

This formula will be satisfiable if the following are satisfiable:

- $A, \alpha \models (\exists x)(\exists x)p(x)$ or
- $A, \alpha[x \mapsto a] \models (\exists x)p(x)$ or
- $A, \alpha[x \mapsto a][x \mapsto b] \models p(x)$

That means if there is an $a \in D$ and there is a $b \in D$, we can find a structure A and an assignment α so that $A, \alpha[x \mapsto a][x \mapsto b] \models p(b)$. From the definition of mapping on assignments, this becomes $A, \alpha[x \mapsto b] \models p(b)$. So the outermost quantifier does not affect the formula at all because the inner quantifier somehow makes x "disappear" from the formula.

Let's formalize this. First we define the set of variables that occur in a formula:

Definition 1 *The set of variables in a term t, denoted by $Vars(t)$, can be thought of as a function $Vars : Terms \rightarrow 2^{Variables}$, and is defined as:*

1. $Vars(x) = \{x\}$
2. $Vars(f(t_1, \ldots, t_k)) = \bigcup_i Vars(t_i)$

Definition 2 *The set of variables in a formula φ, denoted by $Vars(\varphi)$, can also be thought of as a function $Vars : Terms \rightarrow 2^{Variables}$ and equals:*

1. $Vars(P(t_1, \ldots, t_k)) = \bigcup_i Vars(t_i)$
2. $Vars(\neg\varphi) = Vars(\varphi)$
3. $Vars(\theta \wedge \psi) = Vars(\theta) \cup Vars(\psi)$
4. $Vars((\exists x)\varphi) = Vars((\forall x)\varphi) = Vars(\varphi) \cup \{x\}$

We can see from examples that variables occurring with a quantifier are treated differently in the assignment. To distinguish them, we call quantified variables **bound**, and unquantified variables **free**.

Definition 3 *The set of free variables in a formula φ, denoted by $FVars(\varphi)$, can be thought of as a function $FVars : Form \rightarrow 2^{Variables}$ and is defined as:*

1. $FVars(P(t_1, \ldots, t_k)) = \bigcup_i Vars(t_i)$
2. $FVars(\neg\varphi) = FVars(\varphi)$
3. $FVars(\theta \circ \psi) = FVars(\theta) \cup FVars(\psi)$
4. $FVars((\exists x)\varphi) = FVars((\forall x)\varphi) = FVars(\varphi) - \{x\}$

For assignments, we can show that we only need to worry about free variables. This is the "Relevance Lemma " for first-order logic.

Definition 4 *A closed formula or sentence is a formula φ that has no free variables ($FVars(\varphi) = \emptyset$)*

From this definition, it can be inferred that there are no variable assignments that can affect the truth value of a sentence, since in a sentence all variables are quantified and thus we are only interested in "there is a" or "for all". This can be stated formally in the following theorem.

Theorem 1 *Let φ be a sentence, A be a structure, and α_1, α_2 be any assignments. Then, $A, \alpha_1 \models \varphi$ iff $A, \alpha_2 \models \varphi$. In this case we can write $A \models \varphi$.*

2 The relevance lemma

When comparing different variable assignments and their effect on a given formula φ, those variables that do not occur in the formula or are not free variables in the formula should not affect the fact that a given structure A logically implies the formula or not. This intuitive result is formalized as the *Relevance Lemma*, and can be stated as:

Lemma 1 (Relevance lemma) *Let A be a structure, φ a formula, and α_1, α_2 be variable assignments such that: $\alpha_1\big|_{FVars(\varphi)} = \alpha_2\big|_{FVars(\varphi)}$. Then,*

$$A, \alpha_1 \models \varphi \textit{ iff } A, \alpha_2 \models \varphi$$

To prove the relevance lemma, it will be very conveninent to use the following intuitive result, which states that the recursive evaluation of a term under variable assignments that coincide in the term's variables is the same.

Lemma 2 *Let $t \in$ TERM, and let A be a structure and α_1 and α_2 be variable assignments. Then, if $\alpha_1\big|_{Vars(t)} = \alpha_2\big|_{Vars(t)}$ then $\overline{\alpha_1}(t) = \overline{\alpha_2}(t)$.*

Proof: We give a proof by structural induction.

- **Base case**: If t is a variable x then $\alpha_1(x) = \alpha_2(x)$. Notice also that by definition of $\overline{\alpha}$ in this case $\overline{\alpha_1}(t) = \alpha_1(t)$ and the same is true for $\overline{\alpha_2}(t)$. Thus,

$$\overline{\alpha_1}(t) = \alpha_1(t) = \alpha_2(t) = \overline{\alpha_2}(t).$$

  and therefore $\overline{\alpha_1}(t) = \overline{\alpha_2}(t)$. Here the first and the third equalities follow from the definition of $\overline{\alpha}$, and the second equality comes from the assumption of the lemma.

- **Inductive case**: The other option for t is to be a function $f^k(t_1, \ldots, t_k)$, where $t_1, \ldots, t_k$ are terms. By definition of $Vars$, $Vars(f^{(k)}(t_1, \ldots, t_k) = \bigcup_{i=1}^{k} Vars(t_i)$ and so $\textsc{Vars}(t_i) \subseteq \textsc{Vars}(t)$. Then by inductive hypothesis we have that $\overline{\alpha_1}(t_i) = \overline{\alpha_2}(t_i)$ for all i, and thus

$$\begin{array}{rcl}
\overline{\alpha_1}(f^{(k)}(t_1, \ldots, t_k)) & = & \text{(by definition)} \\
f^A(\overline{\alpha_1}(t_1), \ldots, \overline{\alpha_1}(t_k)) & = & \text{(by I.H.)} \\
f^A(\overline{\alpha_2}(t_1), \ldots, \overline{\alpha_2}(t_k)) & = & \text{(by definition)} \\
\overline{\alpha_2}(f^{(k)}(t_1, \ldots, t_k)) & &
\end{array}$$

◊◊◊

We are now ready to prove the Relevance Lemma.

Proof:

- **Base case**: Suppose that φ is a k-ary predicate (relation) $P^{(k)}(t_1, \ldots, t_k)$. By definition $FVars(\varphi) = \bigcup_{i=1}^{k} FVars(t_i)$. We now have the following proof sequence:

$$\begin{array}{ll} & A, \alpha_1 \models \varphi \\ \text{iff} & \langle \overline{\alpha_1}(t_1), \ldots, \overline{\alpha_1}(t_k) \rangle \in P^A \text{ (by definition)} \\ \text{iff} & \langle \overline{\alpha_2}(t_1), \ldots, \overline{\alpha_2}(t_k) \rangle \in P^A \text{ (by Lemma 2)} \\ \text{iff} & A, \alpha_2 \models \varphi \text{ (by definition)} \end{array}$$

- **Inductive case**

  1. $\varphi = \neg\psi$. By definition $A, \alpha_1 \models \neg\psi$ iff $A, \alpha_1 \not\models \psi$. Also by definition $FVars(\varphi) = FVars(\psi)$. Thus, by inductive hypothesis we have $A, \alpha_2 \not\models \psi$ which leads to $A, \alpha_2 \models \varphi$.
  2. $\varphi = \psi \wedge \theta$. By definition $A, \alpha_1 \models \varphi$ iff $A, \alpha_1 \models (\psi \wedge \theta)$ iff $A, \alpha_1 \models \psi$ and $A, \alpha_1 \models \theta$. Clearly $FVars(\psi) \subseteq FVars(\varphi)$ and $FVars(\theta) \subseteq FVars(\varphi)$, so by inductive hypothesis $A, \alpha_2 \models \psi$ and $A, \alpha_2 \models \theta$. This is another way of saying $A, \alpha_2 \models \varphi$.
  3. $\varphi = (\exists x)\psi$. $A, \alpha_1 \models (\exists x)\psi$ by definition means that there is some $d \in D$ such that $A, \alpha_1[x \mapsto d] \models \psi$. To use the inductive hypothesis we need agreement on the free variables. Observe that $FVars(\varphi) = FVars(\psi) - \{x\}$ and therefore $FVars(\psi) \subseteq FVars(\varphi) \cup \{x\}$.
     Since $\alpha_1\big|_{FVars(\varphi)} = \alpha_2\big|_{FVars(\varphi)}$ it must also be true that $\alpha_1[x \mapsto d]\big|_{FVars(\psi)} = \alpha_2[x \mapsto d]\big|_{FVars(\psi)}$ which is the agreement on free variables that we need. We quote once again the inductive hypothesis and conclude that $A, \alpha_2[x \mapsto d] \models \varphi$ which is equivalent to $A, \alpha_2 \models (\exists x)\varphi$.

  The proof for the remaining connectives and one quantifier can be derived from those above stated by using DeMorgan's laws. $\Diamond\Diamond\Diamond$

3 Circuit modelling

Logic has emerged as an important field because of its great modelling power. In this lecture we will see one example: modelling a circuit. First, we define what it is:

Definition 5 *A* **sequential circuit** *is a tuple* $C = \langle I, O, R, Out, Trans, Start \rangle$, *where:*

- I *is the input*
- O *is the output*

- *$Out \in 2^I \times 2^R \to 2^O$ is the output function*
- *$Start \in 2^R$ is the starting state*
- *$Trans \in 2^I \times 2^R \to 2^R$ is the transition function.*

Note that we can describe $Trans$ (transition function) as a sequence of functions $Trans_i \in 2^I \times 2^R \to \{0,1\}$. Each such function is the transition function for one register. Similarly, we can describe $Output$ as a sequence of functions $Output_i \in 2^I \times 2^R \to \{0,1\}$.

Definition 6 *A* **run** *of a sequential circuit $C = \langle I, O, R, Out, Trans, Start\rangle$ is a sequence $(\bar{i_0}, \bar{r_0}, \bar{o_0})(\bar{i_1}, \bar{r_1}, \bar{o_1})...(\bar{i_k}, \bar{r_k}, \bar{o_k})$ of elements in $2^I \times 2^R \times 2^O$ such that:*

1. *Initiation: $\bar{r}_0 = start$ (Circuit starts properly)*
2. *Conseqution: $\bar{r}_{j+1} = Trans(\bar{i}_j, \bar{r}_j)$*
3. *Output: $\bar{o}_j = Out(\bar{i}_j, \bar{r}_j$ and $\bar{o}_j \in 2^{Output}$*

The first step to modelling sequential circuits as relational structures is to pick the domain and vocabulary. Here we choose to pick our domain to be the set of states, that is $D = 2^I \times 2^R$.

We also define the vocabulary: I, R, O as unary relations (for $i \in I$, $i(x) = true$ if high or $false$ if low).

For the behavior of the state, define a binary translation which is a transition relation: $E^{(x)}$

$$\begin{gathered}
A_c = (2^I \times 2^R, i^A, ..., {r_1}^A, ..., {o_1}^A, ..., E^A) \\
{i_j}^A(\bar{i}, \bar{r}) \leftrightarrow i_j = 1 \\
{r_j}^A(\bar{i}, \bar{r}) \leftrightarrow r_j = 1 \\
{o_j}^A(\bar{i}, \bar{r}) \leftrightarrow out(\bar{i}, \bar{r}|_j = 1 \\
E^A((\bar{i}, \bar{r}), (\bar{i}', \bar{r}')) \leftrightarrow Trans(\bar{i}, \bar{r}) = \bar{r}'
\end{gathered}$$

Now that we have a mathematical model, does this satisfy our requirements? Not yet. The most common requirement we have is *mutual exclusion* (don't allow two processes to write to the same register at the same time). For this, we can introduce the

Safety Property:
r_j and r_k cannot be high at the same time, which leads us to write:

$$(\forall x)[\neg(r_j(x) \wedge r_k(x))]$$

However, this is not sufficient, since we don't have an inherent "ranking" of states. We then say that $(\overline{i_k}, \overline{r_k})$ is *reachable*. So then we can write:

$$(\forall x)[reachable(x) \to \neg(r_j(x) \wedge r_k(x))]$$

However, reachable is not described in our vocabulary either. The solution for this is not quite clear yet – we will leave this as a problem for now.