Lecture 22

October 26, 2005

In the last lecture we saw an introduction to first order logic and a proof of the relevance lemma. In this class, we will see how one can describe the functioning of a circuit using first order logic.

1 Searle's Chinese Room

Remember the story by John Searle about a person in a room with a big book listing questions and answers in Chinese. A person outside the door can pass the person in the room pieces of paper with questions on them in Chinese. Using the Chinese book, the person in the room can write down matching answers to the questions and pass them back. How can the person outside the room know if the person inside the room knows Chinese or not? The answer is because conversations have memory. Without an infinite book, there is no way to introduce memory into such a conversation. In reality, we don't have infinite books or Turning machines with infinite tapes so we need a way to represent a model with memory.

2 Circuit modeling

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

Definition 1 *A* **sequential circuit** *is a circuit that has memory. It is represented by a tuple* $C = \langle I, O, R, f, g, Start \rangle$*, where:*

- I *is the set of input characters*
- O *is the set of output characters*
- R *is a set of registers*
- $f \in 2^I \times 2^R \rightarrow 2^R$ *is the transition function.*
- $g \in 2^I \times 2^R \rightarrow 2^O$ *is the output function*

- *Start* $\in 2^R$ *is the initial, or starting, state*

Note that we can describe f (the transition function) as a sequence of functions $f_i \in 2^I \times 2^R \rightarrow \{0,1\}$. Each such function is the transition function for one register. Similarly, we can describe g as a sequence of functions $g_i \in 2^I \times 2^R \rightarrow \{0,1\}$. For a given set of registers and inputs, f drives the values of the registers and g drives the outputs based on the current state of the registers.

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

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

The first step to modeling sequential circuits as relational structures is to pick the domain and vocabulary. The objective is not so much to provide a model of the circuit itself but a model of its behavior which is well-suited to the problem at hand. 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: $E^{(x)}$

$$\begin{gathered}
A_c = (2^I \times 2^R, i_1^A, \ldots, {r_1}^A, \ldots, {o_1}^A, \ldots, Init, E^A) \\
{i_j}^A(\overline{i}, \overline{r}) \leftrightarrow (i_j(\overline{i}) = 1) \\
{r_j}^A(\overline{i}, \overline{r}) \leftrightarrow (r_j(\overline{r}) = 1) \\
{o_j}^A(\overline{i}, \overline{r}) \leftrightarrow (g_j(\overline{i}, \overline{r}) = 1) \\
Init(\overline{i}, \overline{r}) \leftrightarrow (\overline{r} = \overline{r_0}) \\
E^A((\overline{i}, \overline{r}), (\overline{i}', \overline{r}')) \leftrightarrow Trans(\overline{i}, \overline{r}) = \overline{r}'
\end{gathered}$$

Now that we have a mathematical model, does this satisfy our requirements? Not yet. We want to be able to assert certain qualities about our system, like the quality that Bad Things^{TM} don't happen. The most common requirement we have is *mutual exclusion*, which doesn'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 don't want to assert that this property holds in all states of the system, just in all states which are *reachable* from our start state. We then say that $(\overline{i_k}, \overline{r_k})$ is *reachable*. So then we can write:

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

However, the function $reachable(x)$ is not described. Is it possible to express such a function in first order logic? The solution for this is not quite clear yet – we will leave this as a problem for now.

Other material from the 2003 lecture 19 that we have already covered in previous lectures in 2004

3 Semantics

Recall: The meaning of $(\exists x)$: $(\exists x)(\exists y)(\exists z)(\exists n)\ x^n + y^n = z^n$
is that there are some particular values: $x = a, y = b, n = m$ such that $a^m + b^m = c^m$

- $First - class\ objects$: Elements of D
- $Second - class\ objects$: Relations and functions over D
- $Third - class\ objects$: Takes a function as an argument and returns a function (set of functions)

4 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 try to find the set of occurred variables in a formula:

Definition 3 *Variables in a term t, which we denote as $var(t)$, are given by:*

1. $var : Terms \to 2^{var}$
2. $var(x) = \{x\}$
3. $var(f(t_1, \ldots, t_k)) = \bigcup_i var(t_i)$

Definition 4 *Variables in a formula φ, denoted by $var(\varphi)$, are given as:*

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

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

Definition 5 *Free variables in a formula φ, denoted by $fvar(\varphi)$, are given as:*

1. $fvar : Form \to 2^{var}$
2. $fvar(P(t_1, \ldots, t_k)) = \bigcup_i var(t_i)$
3. $fvar(\neg\varphi) = fvar(\varphi)$
4. $fvar(\theta \circ \psi) = fvar(\theta) \cup fvar(\psi)$
5. $fvar((\exists x)\varphi) = fvar((\forall x)\varphi) = fvar(\varphi) - \{x\}$

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

Definition 6 *Closed formula or sentence is a formula φ such that it has no free variables ($fvars(\varphi) = \emptyset$)*

Theorem 1 *Let A be a structure, φ be a sentence, and α_1, α_2 be assignments. Then $A, \alpha_1 \models \varphi$ iff $A, \alpha_2 \models \varphi$.*