CS 2742 (Logic in Computer Science) – Fall 2008 Lecture 13

Antonina Kolokolova

October 29, 2009

4.1 Equivalences and normal forms

Recall the formula $\exists y Parent(x,y) \wedge (\exists y Parent(z,y))$. This was our example of a formula illustrating the scope of a quantifier: here, the scope of the second y is the parentheses where it is located, and the scope of the first quantifier is the rest of the formula. When we were discussing this, we said that a variable could be renamed, to avoid the confusion. But note that if all variables in a formula have different names, then it is possible to make the scope of each of them the whole formula. So the formula above is equivalent to $\exists y \exists u Parent(x,y) \wedge Parent(z,u)$. Note that although we moved $\exists u$ to the front of the formula, we did not change its order relative to $\exists y$. Although for quantifiers of the same type the order can be changed, since it cannot be changed between quantifiers of different types it is good to keep the order of quantifiers the same as it was in the formula. It is often convenient to convert formulas into such form with all the quantifiers in front (called "prenex normal form"). It makes it easier to see the order of quantifiers and perform operations such as negation.

So in a first-order formula we can rename a variable (to a variable name which does not occur in the subformula where we are making the change, obviously), and move the quantifiers to the front of the formula. What else are we allowed to do? Just as with propositional formulas, we are allowed to replace a subformula with another logically equivalent formula (preserving variable names). For example, if we have a formula $\exists x \forall y (P(y) \vee Q(y)) \wedge \neg P(x)$. then we cannot rename y to x. However, we can rename y to, say, z. Also, we know from DeMorgan's law that $P(x) \vee Q(x) \iff \neg(\neg P(x) \wedge \neg Q(x))$. By changing the variable to y in this equivalence, we can substitute the new subformula $\neg(\neg P(y) \wedge \neg Q(y))$ into the original quantified formula to obtain $\exists x \forall y \neg(\neg P(y) \wedge \neg Q(y)) \wedge \neg P(x)$

4.2 Predicates with finite domains and resolution

Consider the case when domain of a predicate is small (i.e., finite). In this case, it is possible to represent quantifiers using $\vee$ and $\wedge$, thus reducing a first-order case to a propositional case.

Example 1. Suppose that we consider the relation $Parent(x, y)$ on the domain consisting of 5 people: $\{John, Bob, Mary, George, Alex\}$. Consider a formula $\forall x \exists y Parent(x, y)$, saying now that each one of these 5 people has another of these 5 as a child (which is not possible, for a domain like that). Or, alternatively, consider a relation $\forall x \forall y \exists z Parent(z, x) \wedge Parent(z, y)$, saying that all people in this list are siblings.

Let's look at the first one of these relation. To save space, I will just write $Parent(x, y)$ as $P(x, y)$ here.

What does it mean that $\forall x A(x)$ some formula $A(x)$ is true? In the case of $A(x)$ being $\exists y P(x, y)$, it means that $A(x)$ is true for John and Bob and Mary and George and Alex. So we can write this as

$$(\exists y P(John, y)) \wedge (\exists y P(Bob, y)) \wedge (\exists y P(Mary, y)) \wedge (\exists y P(George, y)) \wedge (\exists y P(Alex, y))$$

Similarly, what does it mean for an existential quantifier to be true? In this case, the formula is true either for John, or for Bob, or for Mary and so on. So the formula becomes

$$\begin{aligned}
&(P(John, John) \vee P(John, Bob) \vee P(John, Mary) \vee P(John, George) \vee P(John, Alex))\\
\wedge&(P(Bob, John) \vee P(Bob, Bob) \vee P(Bob, Mary) \vee P(Bob, George) \vee P(Bob, Alex))\\
\wedge&(P(Mary, John) \vee P(Mary, Bob) \vee P(Mary, Mary) \vee P(Mary, George) \vee P(Mary, Alex))\\
\wedge&(P(George, John) \vee P(George, Bob) \vee P(George, Mary) \vee P(George, George) \vee P(George, Alex))\\
\wedge&(P(Alex, John) \vee P(Alex, Bob) \vee P(Alex, Mary) \vee P(Alex, George) \vee P(Alex, Alex))
\end{aligned}$$

Now, notice that there are no more free variables in predicates. So in effect they are not predicates anymore, but propositional variables! We can use, say, a variable $p_{m,b}$ to mean $P(Mary, Bob)$ is true, and same for the rest of the occurrences of $P()$. Now, we can write the formula above as a truly propositional formula $((p_{jj} \vee p_{jb} \vee \cdots \vee p_{ja}) \wedge \ldots$ Note that once we got this kind of formula, we can apply resolution to check whether it is a tautology/contradiction.

Example 2. For a more natural example consider the following formula $\exists y,\ 0 \leq y \leq 1\ \forall x,\ 2 \leq x \leq 4\ (y + 1 < x)$ Here, we include the description of the domain into the quantifier. For this example, suppose also that $x, y \in \mathcal{N}$. This formula has the same meaning as $\exists y\ (0 \leq y \leq 1) \wedge (\forall (2 \leq x \leq 4 \rightarrow\ y + 1 < x))$, but here we want to treat these restrictions as restrictions of the domain of the quantifiers.

In this case we have two possible values for y, $y = 0$ and $y = 1$, and three possible values for x, $2, 3$ and 4. After the same transformation as in the previous example, computing $y + 1$ for each y, we obtain the following formula:

$$(1 < 2 \land\ 1 < 3 \land\ 1 < 4) \lor (2 < 2\ \land 2 < 3\ \land 2 < 4)$$

Now, this is a propositional formula where we know the meanings of propositions (here, our propositions are $1 < 2$, $2 < 4$ and so on) so we can figure out its truth value. The only unequality here that is false is $2 < 2$, since we took the "strictly greater" relation $<$ here. This makes the subformula $(2 < 2\ \land 2 < 3\ \land 2 < 4)$ false. However, the subformula in the first set of parentheses is true, therefore the whole formula is true.

When we have many formulas $\lor$ or $\land$ together, it is often convenient to think about it as just one big $\lor$ or big $\land$ operator over many inputs (we can do this because of the associative logic identity). For example, the formula above can be written as $\bigvee_{0 \leq y \leq 1} \bigwedge_{2 \leq x \leq 4} y + 1 < x$. Even more useful this concept becomes when we talk about circuits. In circuits, it is possible to feed more than two wires into an AND gate, or an OR gate, essentially emulating these big $\lor$ and $\land$. This is how quantifiers (on finite domains) are represented using circuits.

4.3 Limitations of FO logic

Recall how we were saying that every Boolean function is expressible by a propositional formula for a given number of variables. Surely, you will say, with predicate logic we can express everything. But the fact is that some very natural properties cannot be expressed in predicate logic, because here the notion of "expressing" is quite different. A propositional formula only talks about finitely many things, and if worst comes to worst it can just list all the cases, describing a truth table. So when we talk about "expressing" boolean functions we do not talk about one formula for one function, we talk about a different formula for every number of inputs to the boolean function, a different formula for every truth table. Whereas in the predicate logic case we have the ability to talk about infinitely many things at once. And it is not possible to list all infinitely many cases of relationships among, say, natural numbers, at least not with a finite-length formula.

Here is an example of a very natural property which is not expressible in first-order logic.

Example 3 (Transitivity in databases)**.** Consider an airline database such as the one that travel agents use to find and book flights. This database could contain a relation *ScheduledFlight(number, departure_city,arrival_city)* which is true when flight *number* goes from *departure_city* to *arrival_city*. For simplicity, let us ignore the number for now, and just look at the relation *Flight(dep_city,arr_city)*, true on pairs of cities such that there is a flight between them (exercise: see how you can write the relation Flight(...,...) using the relation ScheduledFlight(...,...,...)). In real life, the answer of such query would be a list of flight numbers, rather than a simple true/false.

Suppose you want to go from St. John's (code YYT) to Bangalore, India (code BLR). Asking the query $Flight(YYT, BLR)$ is not going to help, because you know that there are only about 5 cities outside of Newfoundland that you can reach flying from St. John's airport, and Bangalore is definitely not one of them. So you need to change planes. The next attempt is to ask the query $\exists y\, Flight(YYT, y) \wedge Flight(y, BLR)$. This will be true if there is a city that has direct flights from St. John's and direct flights to Bangalore. So in fact you want to ask the database "give me a way to fly from St. John's to Bangalore, and I don't care how many times I need to change planes". Well, this kind of query, called "transitivity", is not expressible in first-order logic. For any fixed number of plane changes you can, indeed, ask if there is a way to get from YYT to BLR, however you always need to set a limit on the number of intermediate locations. For example, to ask if there is a way to get to Bangalore via 5 intermediate cities you would write

$$\begin{aligned}&\exists y_1 \exists y_2 \exists y_3 \exists y_4 \exists y_5 Flight(YYT, y_1) \wedge Flight(y_1, y_2) \wedge \\ &\qquad Flight(y_2, y_3) \wedge Flight(y_3, y_4) \wedge Flight(y_4, y_5) \wedge Flight(y_5, BLR)\end{aligned}$$

Some databases do have special ways of computing this kind of relation, but it is an addition to the language of first-order logic and can be quite computationally intensive, especially in large databases. More often, it seems, the database system just tries the first several queries, until they reach a number of intermediate steps that seems too large for them (you might have noticed that you pretty much never see a route with more than 3 intermediate cities, and sometimes get an answer "there is no way").

You can also ask a combination of such queries: for example, $Flight(YYT, BLR) \vee \exists y\, Flight(YYT, y) \wedge Flight(y, BLR)$ is true on the database when there is either a direct flight from YYT to BLR, or a flight with one intermediate city.

5 Set theory

5.1 Empty set and quantifiers

Recall that we talked about "sets" (that is, collections of elements) that were domains of our quantifiers. A set is defined by the elements it contains: remember that $x \in S$ meant that an element x is in S, $x \notin S$ meant that x is not a member of set S.

There is a special set, though, called "empty set" (denoted $\emptyset$) which is the set that contains no elements. That is, $\forall x\; x \notin \emptyset$ is the definition (here, the domain of the quantifier is everything).

What happens when the empty set is our domain? Then if our quantifier is the universal quantifier, then the formula is always true! For example, $\forall x Parent(x, x)$ is true, as well as $\forall x \forall y Parent(x, y) \wedge Parent(y, x)$. Why would such a strange thing happen?

Remember that one way to talk about domains is to put an implication that if the x is in the domain, then the formula under quantifier is true. That is, $\forall x A(x)$ can be stated as $\forall x(x \in domain \rightarrow A(x))$. But note that if $domain = \emptyset$ then the left side of the implication is always false. Therefore, the whole formula is true: for every x, since $x \in domain$ is false, $x \in domain \rightarrow A(x)$ is true. Note that from here you can also see that when an existential quantifier has empty domain, the formula is always false. One way of explaining it is to say that an existential quantifier is a negated universal quantifier, so if the universal is always true, then the existential is always false.

So what happens with big $\vee$ and big $\wedge$? Remember the logic identities $T \wedge p \iff p$, $F \vee p \iff p$? You can see this as saying that an "empty $\wedge$" is true, and an "empty $\vee$" is false. This agrees exactly with the fact that a formula with $\forall$ converted to $\wedge$ and $\exists$ converted to $\vee$ will be true when there are no predicates on $\wedge$ and false when there are no predicates on $\vee$.

Puzzle 1. A man walks into a bar and says to the barman: "pour everybody a drink! when I drink, everyone drinks!". After he finishes the round, he says again: "pour everybody a drink! when I drink, everyone drinks!". The crowd is quite pleased, until he says: "Give me the bill, I'll pay. When I pay, everybody pays!".

What does it have to do with logic, you may ask? Tell me, is there a man such that when he drinks, everybody drinks?