Lecture 27

November 7, 2005

1 Application of First Order Logic to Databases

In out last lecture, we talked about the CS tower of Abstraction. Our next topic, databases, takes place at the middle-ware level.

We need a way for many people to access and manipulate the same data. From this came the concept of a Central Database. An application program would get services from the Database and OS and the Database would get services from the OS. The data in a database needs structure. In the 1960's people came up with the idea of organizing data in a hierarchy, known as the hierarchical model. A Hierarchical database stores its data in a series of records each of which has a set of field values attached to it. It collects all instances of record together as a record type. To create links between these record types, the hierarchical model uses Parent-Child relationships which creates tree-like structures. A more general model called the Network Model for Databases, allows cycles in the hierarchy.

Ted Codd at IBM devised a new type of database structure. He thought it was too complicated to have to follow all of the record links in the Network model to find the data you wanted. He came up with the idea of Tables. He found out that there is a connection between Tables and First Order Logic. A table is a set of rows and a relation is a set of tuples. So the idea of a row in a table is analogous to a tuple in a relation. From this he devised what he called the Relational Model of databases. This consisted of centralized data, with a set $\sigma = R_1 \dots R_k, f_1 \dots f_l$ of relations and functions. Databases can be viewed as structures $A = (D, R_1^A \dots R_k^A, f_1^A \dots f_l^A)$.

He then asserted that a query can be expressed as a formula $\varphi(x_1, \dots, x_k)$. We can view a variable assignment α, where $\alpha(x_i) = a_i$, as a tuple and look at it as $\langle a_1, \dots, a_k \rangle$.

We can now say that $A, \langle a_1, \dots, a_k \rangle \models \varphi(x_1, \dots, x_k)$. Thus, we define $\varphi(x_1 ... x_k)(A) = \{\langle a_1 ... a_k \rangle \mid A, < a_1 ... a_k > \models \varphi\}$. A response to a query will be the set of variable assignments satisfy $\varphi(x_1, \dots, x_k)$.

The two fundamental ideas of Codd are:

1. A table is a relation.

2. A query is a formula

Example 1

EmpID	EmpName	EmpSalary
123	Moshe	1,000,000
456	Ted	1,000,000

The above table is the relation. The domain of the relation is $\{EmpIDs\} \cup \{EmpNames\} \cup \{EmpSalaries\}$.

However, $< Moshe, Moshe, Moshe >$ doesn't quite make sense in the above table. It is possible to split the domain D into $D_1 \times D_2 \times D_3$, and have elaborate type systems. In this class we will stay more abstract and ignore types.

Example 2

Suppose we have a database having the following relations

$Employee{-}Dept^{(2)}, Manager{-}Department^{(2)}, Manager{-}Secretary^{(2)}, Employee{-}Salary^{(2)}$ as well as the predicate $>^{(2)}$

Suppose we want to: 'Find all employees whose manager's secretary makes more money than the manager'.

To answer this query we need one free variable x for Employee:

$(\exists y)(\exists z)(\exists u)(\exists s1)(\exists s2) Employee{-}Department(x, y) \wedge Manager{-}Department(z, y) \wedge Manager{-}Secretary(z, u) \wedge Employee{-}Salary(z, s1) \wedge Employee{-}Salary(u, s2) \wedge > (s2, s1)$.

This query will return all the possible variable assignments to our free variables (Employee) that match the constraints in the formula.

What if φ has no free variables, i.e. it is a sentence? Then, $\varphi(A) = \{<> | A, <> \models \varphi\}$. So, if φ is a sentence, then $\varphi(A) = \emptyset$ or $\varphi(A) = \{<>\}$. By convention, $\{<>\}$ is "true", and $\emptyset$ is "false". Therefore, $\varphi(A)$ is either true or false if φ is a sentence. Such a query is referred to as a Boolean query.

These two ideas were fundamental enough for Ted Codd to be awarded the Turing Award. After Codd published this, two projects were started to test its viability: Ingress at UC Berkeley, and SystemR at IBM. Ingress used QUEL as a query language and SystemR used SEQUEL. Both of these were later combined to create SQL.

Typically, the formulas for a database are expressed as statements in SQL. The intent of SQL was that it would be simple enough for anyone to use. Ideally, you shouldn't be required to know First Order Logic in order to access database information.

2 Query Classification

Before, we talked about classifying propositional formulas as valid, unsatisfiable, or satisfiable. Now, if we have a formula $\varphi(x_1, ..., x_n)$, i.e. $fvar(\varphi) \subseteq \{x_1, ..., x_k\}$, then

1. φ is valid if for all structures A, and all assignments α: $A, \alpha \models \varphi$.
2. φ is unsat if no A, α exist such that $A, \alpha \models \varphi$.
3. φ is sat if for some A, α: $A, \alpha \models \varphi$.
4. $\varphi \models \psi$ means for every A, α: if $A, \alpha \models \varphi$ then $A, \alpha \models \psi$
5. φ, ψ are logically equivalent if for every A, α: $A, \alpha \models \varphi$ iff $A, \alpha \models \psi$

Now let us look at this as a database.

1. If φ is valid, then $\varphi(A) = D^n$, the entire domain, i.e the query will return all the tuples in the database
2. If φ is unsat, then $\varphi(A) = \emptyset$.
3. The interesting formulas in terms of queries are the ones that are satisfiable but not valid. Here, $\varphi(A) \neq D^n$ and $\varphi(A) \neq \emptyset$.
4. If we think of φ and ψ as queries, and for all structures A we have $\varphi(A) \subseteq \psi(A)$,i.e. $\varphi \models \psi$ or φ logically implies ψ, then the query φ is contained in the query ψ. This concept is called Query containment and it is a very fundamental concept. Query Containment is undecidable in general.
5. If $\varphi \models\dashv \psi$, i.e φ is logically equivalent to ψ. Then $\varphi(A) = \psi(A)$, for all A. This brings us to the issue of query optimization. This is the practice of replacing one query by an equivalent and "better" query, where "better" can mean faster, less space, less iteration, or anything we want. The query optimization is not computable in general. But sometimes, we can find some simple way, for example, $\exists x(\alpha \wedge \beta) \models\dashv (\exists x\alpha) \wedge \beta$.

Next time, we'll talk about evaluating queries.