COMP409 - Lecture Notes 22

April 28, 2003

1 Introduction

In the 1970s, Ted Codd (of IBM) realized that databases (formly thought of as a series of records or perhaps hierarchical structures) could be modeled as mathematical relations. In this framework, we think of a relation as a table; the rows of the table correspond to elements of the relation. We may query the database for particular tuples of elements; queries in the database framework correspond to formulas in the relational worldview. That is, $A, \alpha \models \varphi$ is the same as $\alpha \in \varphi(A)$, where $\varphi(A) = \{\alpha | A, \alpha \models \varphi\}$ is a query of the database. We can further write it as $\varphi(A) = \{\langle a_1, ..., a_k\rangle | A, \langle a_1, \ldots, a_k\rangle \models \varphi\}$. Here, φ is a formula with $x_1, x_2, ..., x_k$ as its free variables.

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.

2 Query Classification

Before, we had a classification of propositional formulas as valid, unsatisfiable, and 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, all α, we have $A, \alpha \models \varphi$.
2. φ is unsat if for no A, α, we have $A, \alpha \models \varphi$.
3. φ is sat if for some A, α, we have $A, \alpha \models \varphi$.
4. φ logically implies ψ if whenever $A, \alpha \models \varphi$ also $A, \alpha \models \psi$
5. φ, ψ are logically equivalent if $A, \alpha \models \varphi iff A, \alpha \models \psi$

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 $\varphi \models\!\!=\!\!\mid \psi$, 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, $\exists x(\alpha \wedge \beta) \models\!\!=\!\!\mid (\exists x\alpha) \wedge \beta$.

Generally speaking, *Algorithm = logic + structure*. For example, if we want an algorithm to solve the reachability problem in graph, we can inductively define it by logic formulae and the structure of graph: $E(x, y) \rightarrow Reach(x, y)$, $(\exists z)(Reach(x, z) \wedge Reach(z, y)) \rightarrow Reach(x, y)$. Here, $E(x, y)$ can be explained as there is a edge from x to y in this structure. And $Reach(x, y)$ means that there is a path from x to y, that is, y is reachable from x. Based upon this concept, Japanese engineers wanted to write everything in parallel Prolog, which is a logic definition language. This was the famous plan of the fifth generation computer in the eighties.

3 Query Evaluation

$A = (R, <, = 0, +, *)$ is infinite so it cannot be a database. In fact, a database is a *finite relational structure*. Now that we know how we can write queries, we should focus on finding efficient ways to 'execute' the queries. This is referred to as the query evaluation problem.

Query evaluation problem: Given A and φ, compute $\varphi(A)$. (Recall $\varphi(A) = \{\langle a_1, ..., a_k\rangle | A, < a_1, \ldots, a_k > \models \varphi\}$). Instead, we can ask: given A, α, and φ, does $A, \alpha \models \varphi$? After we have a way to answer this, then we can enumerate over all α to evaluate our query.

So, here is an algorithm:

```
Truth(A,α,φ)
Case
   φ is t₁ ≈ t₂ :  return (TermEval(t₁) = TermEval(t₂))
   φ is p(t₁,...,tₖ) :  return p^A(TermEval(t₁), ..., TermEval(tₖ))
   φ is ¬ψ :  return ¬Truth(A,α,ψ)
   φ is θ∘ψ :  return ∘(Truth(A,α,θ),Truth(A,α,ψ))
   φ is (∃x)ψ :
     B := false
```

```
    do for $a \in D$
      B := B∨Truth($A$,$\alpha[x \mapsto a]$,$\psi$)
    od
    return B
  $\varphi$ is $(\forall x)\psi$ :
    B := true
    do for $a \in D$
      B := B∧Truth($A$,$\alpha[x \mapsto a]$,$\psi$)
    od
    return B
```

In the above, `TermEval` is an equivalent of $\overline{\alpha}$ that we defined earlier. In an algorithmic form, it is:

```
TermEval($A$,$\alpha$,$t$)
Case
  $t$ is $x$ :  return $\alpha(x)$
  $t$ is $f(t_1, ..., t_k)$ :  return $f^A$(TermEval($A$,$\alpha$,$t_1$), ..., TermEval($A$,$\alpha$,$t_k$))
```

Notice that the last case for $Truth()$ is not really necessary, since we can actually rewrite the formula by $(\exists x)\varphi \models\!\!\!=\!\!\!\mid \neg(\forall x)(\neg\varphi)$ and $(\forall x)\varphi \models\!\!\!=\!\!\!\mid \neg(\exists x)(\neg\varphi)$

Theorem 1 *$A, \alpha \models \varphi$ iff $Truth(A, \alpha, \varphi) = 1$.*

This can be easily proved by the semantic definition of the formula.

Now, what is the complexity of $Truth()$ in terms of its inputs, i.e. in terms of $|A| + |\alpha| + |\varphi|$?

Firstly, We can assume that $f^A(...)$ evaluates in linear time in terms of A, since $f^{(1)}$ can be represented as a one-dimension array, $f^{(2)}$ can be represented as a two-dimension array and $f^{(k)}$ can be represented as a k-dimension array.

Secondly, complexity of $TermEval()$ is polynomial time of ($|A| + |\alpha| + |\varphi|$). because the relations can be represented in Boolean Array, that is, $P^{(1)}$ can be represented as a one-dimension boolean array, $P^{(2)}$ can be represented as a two-dimension boolean array and $P^{(k)}$ can be represented as a k-dimension array. Space is therefore also polynomial in terms of input, since in this case $Space \subseteq Time$.

Now, let's consider the complexity of $Truth()$: All of the cases do not generate many more recursions, except for the $\varphi = (\forall x)\psi$ and $\varphi = (\exists x)\psi$ cases, which actually include nested loops. So, the worst-case possible formula is $\forall x_1 \exists x_2 \forall x_3 \exists x_4 ... \forall x_{n-1} \exists x_n p(x_1, x_2, x_3, ..., x_n)$. This is $O(|A|^{\#vars})$. Therefore, the time complexity is $(|A| + |\alpha| + |\varphi|)^{|\varphi|}$, which is exponential in $|\varphi|$. The space complexity is $|\varphi| * log(|A|)$, which is polynomial in φ, and logarithmic in A. So, this is still in $PSPACE$.

Recall that $P \subseteq NP \subseteq PSPACE$. This problem is $PSPACE - complete$. However, if we consider this problem in terms of different parts of the input, we get different complexities:

- Complexity of $\{\langle A, \alpha\rangle | A, \alpha \models \varphi\}$. So, this is the complexity if we keep φ fixed. This is called *data complexity*, and it is PolyTime, and LogSpace.

- Complexity of $\{\langle \alpha, \varphi\rangle | A, \alpha \models \varphi\}$. This is the complexity of we keep the structure fixed. This is the case with databases, since the database uses a specific structure, so this is called *query complexity*. This is ExpTime, and PolySpace. Actually, this is also $PSPACE - complete$.

- Complexity of $\{\langle A, \alpha, \varphi\rangle | A, \alpha \models \varphi\}$. The *combined complexity* makes this $PSPACE - complete$.

The final moral is that there is an exponential gap between data and query complexity.