Tuple Relational Calculus (TRC)

Introduction

Procedural Query language
- query specification involves giving a step by step process of obtaining the query result
   e.g., relational algebra
- usage calls for detailed knowledge of the operators involved
- difficult for the use of non-experts

Declarative Query language
- query specification involves giving the logical conditions the results are required to satisfy
- easy for the use of non-experts

Predicates

We have studied "Predicate Logic" in Discrete Maths course

What is a "predicate"?

A predicate is an expression with place-holders (aka variables)
   – it takes a Boolean value when values are substituted for variables

   lessThan(x,y) – lessThan(2,5) is true; lessThan(5,1) is false

   livesIn(p,c) – livesIn(SindhuPV, Hyderabad) – true

                   – livesIn(PriyankaVadra, Hyderabad) – false

One can have *n*-place predicates also…

Relations and Predicates

Can we interpret

       "relation names" of a relational model as predicates??

Recall: DB Relation – finite set of tuples

Tuple – sequence of values;
       each value corresponds to an attribute
           and comes from a domain of atomic values

Each DB relation name with n attributes
           – can be thought of as an *n-place* predicate

We make Closed-World Assumption (CWA) where...

    – tuples in the relation instance make the predicate true

    – those not present in the instance make it false

TRC – a declarative query language

Tuple variable – associated with a relation

(called the *range relation*)

- takes tuples as its values
- t : tuple variable over relation r with scheme R(A,B,C)
    t.A stands for value of column/attribute A in t etc

TRC Query – basic form:

$$\{ \, t_1.A_{i_1}, \, t_2.A_{i_2}, \dots t_m.A_{i_m} \mid \theta \, \}$$

predicate calculus/logic expression
involving tuple variables
$t_1, \, t_2, \dots, \, t_m, \, t_{m+1}, \dots, t_s$
     - specifies the condition to be satisfied

An example TRC query

student (*rollNo, name, degree, year, sex, deptNo, advisor*)
department (*deptId, name, hod, phone*)

Obtain the rollNo, name of all girl students
in the Maths Dept(deptId = 2)

{s.rollNo,s.name| student(s)∧ s.sex='F'∧ s.deptNo=2}

attributes
required in
the result

This predicate is true whenever
value of *s* is a tuple from the
Student instance, false otherwise

In general, if *t* is a tuple variable with range
relation *r*, *r*(*t*) is taken as a predicate which
is true if and only if the value of *t* is a tuple in *r*

General form of the condition in TRC queries

Atomic expressions are the following:

   1. $r(t)$ --   true if t is a tuple in the relation instance r
   2. $t_1.A_i$ *<compOp>* $t_2.A_j$ *compOp* is one of $\{<, \leq, >, \geq, =, \neq \}$
   3. $t.A_i$ *<compOp> c*   *c* is a constant of appropriate type

Composite expressions:

   1. Any atomic expression
   2. $F_1 \wedge F_2$, $F_1 \vee F_2$, $\neg F_1$ where F_1 and F_2 are expressions
   3. $(\forall t)(F), (\exists t)(F)$ where F is an expression
                                    and t is a tuple variable

   Free Tuple Variable – a variable that is not quantified
   Bound Tuple Variable – a quantified variable

Interpretation of the query in TRC

All possible tuple assignments to the free variables in the query are considered.

For any specific assignment,
    if the expression to the right of the vertical bar evaluates to true,
        that combination of tuple values
        would be used to produce a tuple in the result relation.

While producing the result tuple, the values of the attributes for the corresponding tuple variables as specified on the left side of the vertical bar would be used.

Note: The only free variables are the ones that appear to the left of the vertical bar

Example TRC queries

```
Obtain the rollNo, name of all girl students in the
        Maths Dept
```

```
{s.rollNo, s.name | student(s) ∧ s.sex='F' ∧
                    (∃d)(department(d) ∧ d.name='Maths'
                                ∧ d.deptId = s.deptNo)}
```

s: free tuple variable   *d*: existentially bound tuple variable

Existentially or universally quantified tuple variables can be used on the RHS of the vertical bar to specify query conditions

Attributes of free (or unbound) tuple variables can be used on LHS of vertical bar to specify attributes required in the results

Example Relational Scheme

student (<u>rollNo</u>, name, degree, year, sex, deptNo, advisor)

department (<u>deptId</u>, name, hod, phone)

professor (<u>empId</u>, name, sex, startYear, deptNo, phone)

course (<u>courseId</u>, cname, credits, deptNo)

enrollment (<u>rollNo, courseId, sem, year</u>, grade)

teaching (<u>empId, courseId, sem, year</u>, classRoom)

preRequisite (<u>preReqCourse, courseID</u>)

Example queries in TRC (1/5)

1) Determine the departments that do not have any girl students

student (<u>rollNo</u>, name, degree, year, sex, deptNo, advisor)
department (<u>deptId</u>, name, hod, phone)

```
{d.name|department(d) ∧
          ¬(∃s)(student(s) ∧
                   s.sex ='F' ∧ s.deptNo = d.deptId)
```

Examples queries in TRC (2/5)

2) Obtain the names of courses enrolled by student
   named Mahesh

```
{c.name | course(c) ∧
         (∃s)(∃e)(student(s) ∧ enrollment(e)
                  ∧ s.name = "Mahesh"
                  ∧ s.rollNo = e.rollNo
                  ∧ c.courseId = e.courseId }
```

Examples queries in TRC (3/5)

3) Get the names of students who have scored 'S' in all subjects they have enrolled. Assume that every student is enrolled in at least one course.

```
{s.name | student(s) ∧
        (∀e)(( enrollment(e) ∧
                e.rollNo = s.rollNo) → e.grade ='S')}
```

Person P with all S grades:
   for enrollment tuples not having her roll number, LHS (of →) is false
   for enrollment tuples having her roll number, LHS is true, RHS is also true
         So the implication is *true* for all e tuples

Person Q with some non-S grades:
   for enrollment tuples not having her roll number, LHS is false
   for enrollment tuples having her roll number, LHS is true, but RHS is false for
                                                            at least one tuple.

         So the implication is *not true* for at least one tuple.

Examples queries in TRC (4/5)

4) Get the names of students who have taken at least
   one course taught by their advisor

```
{s.name | student(s) ∧
          (∃e)(∃t)(enrollment(e) ∧ teaching(t) ∧
                    e.courseId = t.courseId ∧
                    e.rollNo = s.rollNo ∧
                    t.empId = s.advisor}
```

5) Display the departments whose HODs are teaching
   at least one course in the current semester

```
{d.name | department(d) ∧(∃t)(teaching(t) ∧
                    t.empid = d.hod
          ∧ t.sem = 'odd' ∧ t.year = '2019')}
```

Examples queries in TRC (5/5)

6) Determine the students who are enrolled for **every** course taught by Prof Ramanujam. Assume that Prof Ramanujam teaches at least one course.

```
1.    {s.rollNo | student (s) ∧
2.       (∀c)(course (c) ∧
3.            ((∃t),(∃p)( teaching(t) ∧ professor(p) ∧
4.                 t.courseId = c.courseId ∧
5.                 p.name = "Ramanujam" ∧
6.                 p.empId = t.empId )) →
7.                   (∃e) (enrollment(e) ∧
8.                       e.courseId = c.courseId ∧
9.                       e.rollNo = s.rollNo)
10.   )
11.  }
```

Problem with unrestricted use of Negation

What is the result of the query:

```
{s.rollNo|¬ student(s)} ?
```

Infinite answers !!

Unsafe TRC expression :
Any expression whose result uses "constants / values" that do not appear in the instances of any of the database relations.

Unsafe expressions are to be avoided while specifying TRC queries.

Expressive power of TRC and Relational Algebra

It can be shown that
both Tuple Relational Calculus and Relational Algebra
have the same expressive power

A query can be formulated in (safe) TRC
if and only if it can be formulated in RA

Both *can not* be used to formulate queries involving
transitive closure
-- find all direct or indirect pre-requisites of a course
-- find all subordinates of a specific employee etc.