Propositional Logic - Review

Propositional Logic: formalisation of reasoning involving propositions

- *Proposition*: a statement that can be either true or false.
- *Propositional variable*: variable intended to represent the most primitive propositions relevant to our purposes
- Given a set *S* of propositional variables, the set *F* of propositional formulas is defined recursively as:
  Basis: any propositional variable in *S* is in *F*
  Induction step: if *p* and *q* are in *F*, then so are $\neg p$, $(p \wedge q)$, $(p \vee q)$, $(p \rightarrow q)$ and $(p \leftrightarrow q)$

1

Predicate Logic - Review

Predicate (First-order) Logic: formalisation of reasoning involving predicates.

- *Predicate* (sometimes called parameterized proposition): a Boolean-valued function.
- *Domain:* the set of possible values for a predicate's arguments.

2

Predicate Logic – Review cont'

- A *first-order language* consists of:
  - an infinite set of *variables*
  - a set of *predicate symbols*
  - a set of *constant symbols*
- A *term* is a variable or a constant symbol
- An *atomic formula* is an expression of the form *p(t1,…,tn)*, where *p* is a n-ary predicate symbol and each *ti* is a term.

3

Predicate Logic - Review cont'

Given a first-order language *L*, the set *F* of *predicate (first-order) formulas* is constructed inductively as follows:
Basis: any atomic formula in *L* is in *F*
Inductive step: if *e* and *f* are in *F* and *x* is a variable in *L*, then so are the following: $\neg e$, $(e \wedge f)$, $(e \vee f)$, $(e \rightarrow f)$, $(e \leftrightarrow f)$, $\forall x\ e$, $\exists s\ e$.

- An occurrence of a variable *x* is *free* in a formula *f* if and only if it does not occur within a subformula *e* of *f* of the form $\forall x\ e$ or $\exists x\ e$.
- A formula that has no free variables is called a *sentence*.

4

Relational Calculus

- *Relational Calculus:* applied predicate calculus tailored to relational databases
- Comes in two flavors: *Tuple relational calculus* (TRC) and *Domain relational calculus* (DRC).
- Calculus has *variables, constants, comparison operators*, *logical connectives* and *quantifiers*.
  - *TRC*: Variables range over *tuples*.
  - *DRC*: Variables range over *domain elements*
  - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called *formulas*.

5

Tuple Relational Calculus

For a given relational schema,

- *a* term is a constant or *t.a*, where *t* is a tuple variable and *a* is an attribute (*t.a* indicates the value of tuple *t* on attribute *a*).
- an *atomic formula* is one of the following:
  - $t \in R$ or *R(t),* where *t* is a tuple variable and *R* is a relation
  - *t.a operator s.b,* where *t* and *s* are tuple variables and *a* and *b* are attributes
  - *t.a operator constant*
- A *formula* is obtained following the rules of predicate calculus.

6

Tuple Relational Calculus

- *Query:* of the form: {t | f(t)}, where *f(t)* denotes a TRC formula in which the tuple variable *t* is the only free variable.
- *Answer:* the set of all tuples *t* for which the formula *f*(t) evaluates to *true*.

7

Tuple Relational Calculus

Movies(title, year, length)
Actors(name, address, age)
Contracts(movie_title, actor_name, salary)

Selection:
Find all movies made in 1979.
{*t* | *Movies(t)* ∧ *t.year* = 1979 }

Projection:
List the names of the movies longer than 120 minutes.
{*t* | ∃*s, Movies(s)* ∧ *s.length* >120 ∧ *s.titles = t.title*}

t is a projection of *Movies* since only the title is mentioned and *t* is not specified to range over any relation in the schema.

8

Tuple Relational Calculus

Join
Find the movies who starred Harrison Ford in 1980.
{*t* | ∃*s, Contracts(s)* ∧ Movies(t) ∧ *t.year* =1980 ∧ *s.title = t.title* ∧ s.name = "Harrison Ford"}

Note the use of ∃ to find a tuple in Contracts that `joins with' the Movies tuple under consideration.

9

Tuple Relational Calculus

Division:
Find the actors who have starred in all movies.
{t | Actors(t) ∧ (∀s, Movies(s) →(∃u, Contracts(u) ∧ t.name = u.name ∧ s.title = u.title))}

Find all actors who have starred in all movies produced in 1990
{t | Actors(t) ∧ (∀s, Movies(s) → (s.year=1990 → (∃u, Contracts(u) ∧ t.name = u.name ∧ s.title = u.title))}

or

{t | Actors(t) ∧ ∀s, Movies(s) ∧ (s.year<>1990 \/ (∃u, Contracts(u) ∧ t.name = u.name ∧ s.title = u.title))}

10

Domain Relational Calculus

For a given relational schema,
- *a* term is a constant or *variable*
- an *atomic formula* is one of the following:
  - *(x1, ..., xn)* ∈ *R* or *R(x1, ..., xn),* where *t* is a tuple variable and *R* is a relation
  - *x operator y,* where *x* and *y* are domain variables
  - *x operator constant*
- A *formula* is obtained following the rules of predicate calculus.

11

Domain Relational Calculus

- *Query:* of the form: {(x1,..., xn) | f(x1, ..., xn)}, where *f(x1, ..., xn)* denotes a DRC formula in which the domain variables *x1, .., xn* are the only free variables.
- *Answer:* the set of all tuples *(x1, ...xn)* for which the formula *f*(x1, ..., xn) evaluates to *true*.

12

Domain Relational Calculus

Movies(<u>title</u>, year, length)
Actors(<u>name</u>, address, age)
Contracts(movie_title, actor_name, salary)

Selection:
Find all movies made in 1979.
{*(x, y, z)*| *Movies(x,y,z)* ∧ *x* = 1979 }

Projection:
List the names of the movies longer than 120 minutes.
{*x* | ∃*y,z, Movies(x,y,z)* ∧ *z* > 120}

13

Domain Relational Calculus

Join
Find the movies who starred Harrison Ford in 1980.
{*(x,y,z)* | ∃*t,u,v, Contracts(t,u,v)* ∧ Movies(x,y,z) ∧ *y* =1980 ∧ x = *t* ∧ u = "Harrison Ford"}

Note the use of ∃ to find a tuple (t,u,v) in Contracts that `joins with' the Movies tuple (x,y,z) and the join condition x = t.

Alternative:
{*(x,y,z)* | ∃*t,u,v, Contracts(x,u,v)* ∧ Movies(x,y,z) ∧ *y* =1980 ∧ u = "Harrison Ford"}

14

Domain Relational Calculus

Division:
Find the actors who have starred in all movies.
{(x,y) | Actors(x,y)∧(∀s,t,u, Movies(s,t,u) → (∃w,Contracts(s,x,w)))}

Find all actors who have starred in all movies produced in 1990
{(x,y) | Actors(x,y) ∧ (∀s,t,u, (Movies(s,t,u) ∧ t=1990) → (∃v, Contracts(s,x,v)))}
or
{(x,y) | Actors(x,y) ∧ (∀s,t,u, ⌐(Movies(s,t,u) ∧ t=1990) V (∃v, Contracts(s,x,v)))}

15

Domain Relational Calculus

Unsafe queries: syntactically correct calculus queries that have an infinite number of answers.

List the people who are not actors
{*t* | ⌐*Actors(t)*}
{*(x,y)* | ⌐*Actors(x,y)*}
Answer????

Safe queries: the answer set is finite.

16

From Relational Algebra to Safe Relational Calculus

Each expression *e* in RA producing a n-ary relation can be translated into a formula f(x1,...,xn) in RC.
Consider a relational schema R(A1,...,An), S(B1,...,Bm)

Relational Algebra	Relational Calculus
R	R(x1, ..., xn)
$\sigma_{condition}(R)$	R((x1,..,xn) /\ condition
$\pi_{A1,...,Am}(R)$	∃ y1,...,yp, R(x1,...,xq,y1,...,yp)
R X S	R(x1,...,xn) /\ S(y1,..,ym)
R U S	R(x1,...,xn) /\ S(x1,..,xn)
R - S	R(x1,...,xn) /\ ⌐S(x1,..,xn)

17

Relational Calculus

- **Expressive Power** (Theorem due to Codd):
  every query in Relational Algebra can be expressed as a safe query in DRC / TRC; the converse is also true
- ***Relational Completeness***: property of a query language = ability to express every query that is expressible in relational algebra/safe calculus.

18