Computational Logic

Relational Query Languages with Negation

Free University of Bozen-Bolzano, 2010

Werner Nutt

(Slides adapted from Thomas Eiter and Leonid Libkin)

Queries with "All"

"Who are the directors whose movies are playing in all theaters?"

- What does it actually mean?

$$\Big\{ \text{dir} \;\Big|\; \exists\, \text{tl'}, \text{act'}\ \text{Movie}(\text{tl'}, \text{dir}, \text{act'}) \wedge \forall\, \text{th}\ \big(\exists\, \text{tl''}\ \text{Schedule}(\text{th}, \text{tl''}) \rightarrow \exists\, \text{tl}, \text{act}\ \text{Schedule}(\text{th}, \text{tl}) \wedge \text{Movie}(\text{tl}, \text{dir}, \text{act})\big) \Big\}$$

- To understand this, we revisit rule-based queries, and write them in logical notation.

Expressing Rules in Logic

- By now, we have become familiar with queries like the one below:

  answer(th) :– movie(tl, ’Polanski’, act), schedule(th,tl)

- How can we phrase this query in English?
- It specifies those theaters th such that the following holds:

  There exist a movie (tl) and an actor (act) such that
  (th,tl) is in Schedule and (tl, ’Polanski’, act) is in Movie

- Using notation from mathematical logic, we can introduce a query predicate Q(·) and define it by the property above:

  $\text{Q(th)} \iff \exists\ \text{tl}\ \exists\ \text{act Movie(tl, 'Polanski', act)} \wedge \text{Schedule(th,tl)}$

Other Queries in Logical Notation

- Rule-based query:

  answer(th) :– movie(tl, dir, ’Nicholson’), schedule(th,tl)

- Query as formula:

  $\text{Q(th)} \iff \exists\ \text{tl}\ \exists\ \text{dir Movie(tl, dir, 'Nicholson')} \wedge \text{Schedule(th,tl)}$

- In general, every single-rule query can be written in this logical notation using only:

  existential quantification $\exists$

  and

  logical conjunction $\wedge$

SPJRU Queries in Logical Notation

"Who are the actors who played in movies directed by Kubrick *OR* Polanski?"

- Rule-based notation, using two rules:

  answer(act) :– movie(tl,dir,act), dir='Kubrick'

  answer(act) :– movie(tl,dir,act), dir='Polanski'

- Logical notation:

$$\text{Q(act)} \iff \exists\,\text{tl}\ \exists\,\text{dir}\ (\text{Movie(tl,dir,act)} \ \wedge \ (\text{dir} = \text{'Kubrick'} \vee \text{dir} = \text{'Polanski'}))$$

  The new element here is logical disjunction $\vee$ (OR)

Proposition. SPJRU queries can be expressed in logical notation using

– existential quantifiers $\exists$

– conjunction "$\wedge$" and disjunction "$\vee$"

Queries with "All" (cntd)

$$\Big\{ \text{dir} \ \Big| \ \exists\,\text{tl', act' Movie(tl', dir, act')} \wedge \forall\,\text{th}\ \big(\exists\,\text{tl'' Schedule(th, tl'')} \rightarrow \exists\,\text{tl, act Schedule(th, tl)} \wedge \text{Movie(tl, dir, act)}\big) \Big\}$$

- The new element here is universal quantification $\forall$ ("for all")

- We know:

$$\forall x F(x) \ \equiv \ \neg\exists x\, \neg F(x)$$

  So, we can capture this if we introduce *negation*

Relational Calculus

- Relational calculus consists of queries written in the logical notation using:

  relation names (e.g., Movie)

  constants (e.g., 'Nicholson')

  conjunction $\wedge$, disjunction $\vee$, implication $\rightarrow$

  negation $\neg$

  existential quantifiers $\exists$ and universal quantifiers $\forall$

- The logical symbols $\wedge, \exists, \neg$ suffice:

$$\forall x F(x) \equiv \neg\exists x \neg F(x)$$
$$F \vee G \equiv \neg(\neg F \wedge \neg G)$$
$$F \rightarrow G \equiv \neg F \vee G$$

- Relational calculus has exactly the syntax of first-order predicate logic.

Bound and Free Variables

When considering a formula φ as a query, the free variables of φ play an outstanding role.

- An occurrence of a variable x in formula φ is *bound* if it is within the scope of a quantifier $\exists x$ or $\forall x$
- An occurrence of a variable in φ is *free* iff it is not bound
- A variable of formula φ is *free* if it has a free occurrence
- Free variables go into the output of a query

Queries in Relational Calculus

Essentially, a query is nothing but a formula.

We use two special notations to highlight the free variables $\vec{x}$ of φ:

- $Q(\vec{x}) \iff \varphi$
- $\{\vec{x} \mid \varphi\}$

Examples for the second notation:

- $\{x, y \mid \exists z \left(R(x, z) \land S(z, y)\right)\}$
- $\{x \mid \forall y R(x, y)\}$

Queries without free variables are called *Boolean queries*. Their output is *true* or *false*. Examples:

- $\forall x R(x, x)$
- $\forall x \exists y R(x, y)$

Reminder: Semantics of First-Order Predicate Logic

In predicate logic, the semantics of formulas is defined in terms of two ingredients

- *interpretations* I, where I

  has a set Δ^I as *domain of interpretation*

  maps constants c to elements $c^I \in \Delta^I$

  maps n-ary relation symbols r to relations $r^I \subseteq (\Delta^I)^n$

- *assignments* $\alpha\colon \mathbf{var} \to \Delta^I$, where $\mathbf{var}$ is the set of all variables.

One defines recursively over the structure of formulas when a pair I, α *satisfies* a formula φ, written

$$I, \alpha \models \varphi$$

Database Instances as First-Order Interpretations

In a straightforward way, every database instance $\mathbf{I}$ gives rise to a first-order interpretation $I_{\mathbf{I}}$ that

– has domain $\Delta^{I_{\mathbf{I}}} = \mathbf{dom}$

– maps every constant to itself, i.e., $c^{I_{\mathbf{I}}} = c$ for all $c \in \mathbf{dom}$

– maps every n-ary relation symbol R to $R^{I_{\mathbf{I}}} = \mathbf{I}(R) \subseteq \mathbf{dom}^n$.

To simplify our notation, we will often identify $\mathbf{I}$ and $I_{\mathbf{I}}$.

Semantics of Queries

- If $\vec{x}$ is a tuple of variables and $\alpha\colon \mathbf{var} \to \mathbf{dom}$ is an assignment, then $\alpha(\vec{x})$ is a tuple of constants.
- Let $Q = \{\vec{x} \mid \varphi\}$ be a query. We define the answer of Q over $\mathbf{I}$ as

$$Q(\mathbf{I}) = \{\, \alpha(\vec{x}) \mid \mathbf{I}, \alpha \models \varphi \,\}$$

How does this relate to our previous definition of answers to conjunctive queries?

Negation in the Calculus Requires Care

- What is the meaning of the query

$$Q = \{x \mid \neg R(x)\} \quad ?$$

  It says something like, "Give me everything that is *not* in the database"

- According to our formal definition, $Q(\mathbf{I}) = \mathbf{dom} \setminus \mathbf{I}(R)$.

  But this is an *infinite* set!

Safe Queries

Definition (Safety). A calculus query is *safe* if it returns finite results over all (finite) databases.

- Clearly, practical languages can only allow safe queries.
- Bad news: Safety is undecidable. (That is: No algorithm exists to check whether a query is safe.)
- Good news: All SPJRU queries are safe.

  Reason: Everything constant that occurs in the output must have occurred in the input.
- We conclude: Queries can become unsafe if we allow negation.

Negation in Relational Algebra: Difference

Definition (Difference in the Named Perspective). If R and S are two relations with the same set of attributes, then $R \setminus S$ is their set difference, i.e., the set of all tuples that occur in R but not in S.

Example:

A	B
a1	b1
a2	b2
a3	b3

$\setminus$

A	B
a2	b2
a3	b3
a4	b4

$=$

A	B
a1	b1

For which relations can one define difference in the unnamed perspective?

Definition. The (full) relational algebra comprises the operators projection, selection, cartesian product, renaming and difference.

How Does Relational Calculus Compare to Relational Algebra?

We have seen that close connections exist between fragments of relational algebra and fragments of relational calculus, e.g.,

- SPC queries $\leftrightarrow$ conjunctive queries
- SPCU queries $\leftrightarrow$ unions of conjunctive queries

Observation. All relational algebra queries are safe, but not all calculus queries

$\Longrightarrow$ not all calculus queries can be expressed in algebra

Questions:

- Can we characterize the calculus queries that can be expressed in algebra?
- Can all safe queries be expressed in algebra?

Query Semantics (cntd)

- When fixing the semantics of calculus queries, we defined the domain of $I_{\mathbf{I}}$ as

$$\Delta^{I_{\mathbf{I}}} = \mathbf{dom}.$$

  However, there are more options.

- For an instance $\mathbf{I}$ and a query Q let
  - *adom*$(\mathbf{I})$ = the set of constants occurring in $\mathbf{I}$; the *active domain* of $\mathbf{I}$
  - *adom*(Q) = the set of constants occurring in Q; the *active domain* of Q
  - *adom*$(Q, \mathbf{I}) =$ *adom*$(Q) \cup$ *adom*$(\mathbf{I})$; the *active domain* of Q and $\mathbf{I}$
- A set $\mathbf{d} \subseteq \mathbf{dom}$ is *admissible* for Q and $\mathbf{I}$ if *adom*$(Q, \mathbf{I}) \subseteq \mathbf{d}$.
- Given an admissible $\mathbf{d}$ we define $I^{\mathbf{d}}_{\mathbf{I}}$ similarly as $I_{\mathbf{I}}$, with the exception that

$$\Delta^{I^{\mathbf{d}}_{\mathbf{I}}} = \mathbf{d}.$$

Query Semantics (cntd)

- Let **d** be admissible for $Q = \{\vec{x} \mid \varphi\}$ and $\mathbf{I}$
- Then we define the *answer* of Q over $\mathbf{I}$ *relative to* $\mathbf{d}$ as

$$Q_{\mathbf{d}}(\mathbf{I}) = \{\, \alpha(\vec{x}) \mid I^{\mathbf{d}}_{\mathbf{I}}, \alpha \models \varphi \,\}$$

  Intuitively, different semantics have different quantifier ranges.

- The extreme cases are:
  - *Natural semantics* $Q_{nat}(\mathbf{I})$: unrestricted interpretation, that is $\mathbf{d} = \mathbf{dom}$
  - *Active domain semantics* $Q_{adom}(\mathbf{I})$: the range of quantifiers is the set of all constants in Q and in $\mathbf{I}$, that is $\mathbf{d} =$ *adom*$(Q, \mathbf{I})$.

Domain Dependent Queries

Sometimes, the answer $Q_{\mathbf{d}}(\mathbf{I})$ can be different for the same Q and $\mathbf{I}$ if $\mathbf{d}$ varies.

Examples:

- $\{x, y, z \mid \neg$Movie$(x, y, z)\}$
- $\{x, y \mid$ Movie(x,Polanski,Nicholson) $\vee$ Movie(Chinatown,Polanski,y)$\}$

The results of these queries are *domain dependent*.

Observation. Relational Algebra queries do not depend on the domain.

Domain Dependent Queries (cntd)

- The previous examples of domain dependent queries were not safe.
  One may think that the problem of domain dependence is the one of possibly infinite query outputs.
- But something more subtle plays a role: the range of quantifiers
- Example:

$Q(x) = \{x \mid \forall y\, R(x, y)\}$ $\mathbf{I} =$

R	A	B
	a	a
	a	b

For this query Q over this interpretation $\mathbf{I}$ we have

$Q_{nat}(\mathbf{I}) = \emptyset$

$Q_{adom}(\mathbf{I}) = \{\langle a \rangle\}$.

Domain Independence

Definition. A calculus query Q is *domain independent* if for all $\mathbf{I}$ and all admissible $\mathbf{d}$, $\mathbf{d}'$ we have that

$$Q_{\mathbf{d}}(\mathbf{I}) = Q_{\mathbf{d}'}(\mathbf{I}).$$

Examples.

- Positive examples:

  $\exists$ tl $\exists$ act Movie(tl, 'Polanski', act) $\wedge$ Schedule(th,tl)

  Every SPJU query, rewritten to logical notation

- Negative examples:

  $\{x, y \mid$ Movie(x,Polanski,Nicholson) $\vee$ Movie(Chinatown,Polanski,y)$\}$

  $\{x \mid \forall y$ Schedule(y, x)$\}$

Domain Independence (cntd)

Proposition. If Q is domain independent, then for all instances $\mathbf{I}$ and all admissible $\mathbf{d} \subseteq \mathbf{dom}$ we have that

$$Q_{adom}(\mathbf{I}) = Q_{\mathbf{d}}(\mathbf{I}) = Q_{nat}(\mathbf{I})$$

Definition. The *Domain-independent Relational Calculus* (DI-RelCalc) consists of the domain-independent queries in RC.

Domain Independence (cntd)

Theorem. Domain independence is undecidable.

- Consequence: It is undecidable whether a given formula $Q(\vec{x})$ belongs to DI-RelCalc
- Still, there are (decidable) syntactic properties of queries that imply domain independence
- There are even domain-independent fragments of RelCalc that can be efficiently recognized and that are as expressive as the full DI-RelCalc (e.g., *safe range queries*)

Equivalence Theorem of Relational Query Languages

Theorem. The following query languages have the same expressivity:

- Relational Algebra with the operations π, σ, $\times$, $\cup$, $\setminus$, ρ
- Domain-independent Relational Calculus (DI-RelCalc)
- Relational Calculus under Active Domain Semantics

We won't give a formal proof of this statement (which can be found in the book in Section 5.3), but try to explain why it is true.
As a side effect, we will see some examples of relational algebra usage

Proof Sketch: From Relational Algebra to DI-RelCalc

- Show that unnamed relational algebra can be expressed by relational calculus
- Use only $\exists$ quantifiers in the transformation
- Ensure that each free variable x, resp. each variable quantified by an $\exists x$ is "grounded" in some atom $R(..., x, ...)$
- This yields for each RelAlg expression E a domain-independent transform φ_E such that the semantics of E and of φ_E coincide
- In particular, the semantics of E and the Active Domain Semantics of φ_E coincide

From Relational Algebra to DI-RelCalc /1

Principle: Each expression E producing an n-ary relation is translated into a formula $\varphi_E(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$

- $R \;\mapsto\; R(x_1, \ldots, x_n)$
- $\sigma_C(E) \;\mapsto\; \varphi_E(x_1, \ldots, x_n) \wedge C$

Example: Suppose R is binary. Then

$$\sigma_{1=2}(R) \;\mapsto (R(x_1, x_2) \wedge x_1 = x_2).$$

From Relational Algebra to Dl-RelCalc/2

- If E has arity $(n+m)$, then

$$\pi_{1,\ldots,n}(E) \mapsto \exists y_1,\ldots,y_m\ \varphi_E(x_1,\ldots,x_n,y_1,\ldots,y_m).$$

The attributes that are not *projected are quantified.*

Example: Suppose R is binary. Then

$$\pi_1(R) \ \mapsto\ \exists x_2 R(x_1,x_2).$$

- For any E, F with arity n, m, resp.

$$E \times F \ \mapsto\ \varphi_E(x_1,\ldots,x_n) \wedge \varphi_F(y_1,\ldots,y_m)$$

(note that the formula has $n+m$ distinct free variables)

From Relational Algebra to Dl-RelCalc/3

- If E and F both have the same arity, say n, then

$$E \cup F \ \mapsto\ \varphi_E(x_1,\ldots,x_n) \vee \varphi_F(x_1,\ldots,x_n)$$

(note that the output has n distinct free variables)

- If E and F both have the same arity, say n, then

$$E \setminus F \ \mapsto\ E(x_1,\ldots,x_n) \wedge \neg F(x_1,\ldots,x_n)$$

(note that the output has again n distinct free variables)

From DI-RelCalc to Relational Algebra: Translation

The *active domain* of a relation is the set of all constants that occur in it.

- Example:

R_1	A	B
	a_1	b_1
	a_2	b_2

has active domain $\{a_1, a_2, b_1, b_2\}$.

- We can express the active domain of a relation R in relational algebra. Suppose R has attributes $A_1, \ldots, A_n$. Then:

$$\mathrm{ADOM}(R) \;=\; \rho_{B \leftarrow A_1}(\pi_{A_1}(R)) \cup \ldots \cup \rho_{B \leftarrow A_n}(\pi_{A_n}(R))$$

- The active domain is a relation with one attribute (here: B)
- We can also express the active domain of a database:

$$\mathrm{ADOM}(R_1, \ldots, R_k) = \mathrm{ADOM}(R_1) \cup \cdots \cup \mathrm{ADOM}(R_k)$$

From DI-RelCalc to Relational Algebra

Let $Q(\vec{x})$ be a query over the relations $R_1, \ldots, R_n$.

- If Q is domain-independent,
  then $Q(\vec{x})$ can wlog be evaluated over $\mathrm{ADOM}(R_1, \ldots, R_n)$.
- Thus, we need to show how to translate relational calculus queries over $\mathrm{ADOM}(R_1, \ldots, R_n)$ into relational algebra queries.
- We will translate a relational calculus formula $\varphi(x_1, \ldots, x_n)$ into a relation algebra expression E_φ with n attributes.

We will mix named an unnamed perspective
and use whatever is more convenient

From DI-RelCalc to Relational Algebra /2

Easy cases. Let R be a relation with attributes $A_1, \ldots, A_n$:

- $R(x_1, \ldots, x_n) \;\mapsto\; R$
- $\exists x_1 R(x_1, \ldots, x_n) \;\mapsto\; \pi_{A_2,\ldots,A_n}(R)$

Not so easy cases. Conditions and negation:

- $C(x_1, \ldots, x_n) \;\mapsto\; \sigma_C(\text{ADOM} \times \cdots \times \text{ADOM})$

  E.g., $x_1 = x_2$ is translated into $\sigma_{1=2}(\text{ADOM} \times \text{ADOM})$
- $\neg R(\vec{x}) \;\mapsto\; (\text{ADOM} \times \cdots \times \text{ADOM}) \setminus R$

  We only compute the tuples of database elements that do not belong to R

From DI-RelCalc to Relational Algebra /3

The hardest case. Disjunction:

- Let both R and S be binary. Consider the relational calculus query:

$$Q(x, y, z) \iff R(x, y) \lor S(x, z)$$

- The result is ternary and consists of tuples (x, y, z) such that

  either $(x, y) \in R$, $z \in \text{ADOM}$, or $(x, z) \in S$, $y \in \text{ADOM}$
- The first disjunct translates simply to $R \times \text{ADOM}$
- The second translation is more complex: $\pi_{1,3,5}(\sigma_{1=4 \land 2=5}(S \times \text{ADOM} \times S))$
- Taking the two together yields

$$Q(x, y, z) \;\mapsto\; R \times \text{ADOM} \,\cup\, \pi_{1,3,5}(\sigma_{1=4 \land 2=5}(S \times \text{ADOM} \times S))$$

From DI-RelCalc to Relational Algebra /4

Conjunction is mapped to (natural) join

- Suppose we have mapped

$$\varphi(x_1,\ldots,x_m,y_1,\ldots,y_n) \quad \mapsto \quad E(A_1,\ldots,A_m,B_1,\ldots,B_n)$$
$$\psi(x_1,\ldots,x_m,z_1,\ldots,z_k) \quad \mapsto \quad F(A_1,\ldots,A_m,C_1,\ldots,C_k)$$

- Then

$$\varphi(x_1,\ldots,x_m,y_1,\ldots,y_n) \wedge \psi(x_1,\ldots,x_m,z_1,\ldots,z_k) \;\mapsto\; E \bowtie F$$

Recall that the natural join can be defined in terms of ρ, $\times$, σ, and π

Queries with “All” in Relational Algebra

- “Find directors whose movies are playing in all theaters”

$$\Big\{ \text{dir} \;\Big|\; \exists\, \text{tl'}, \text{act'}\ \text{Movie}(\text{tl'}, \text{dir}, \text{act'}) \wedge \forall\, \text{th}\ \big(\exists\, \text{tl''}\ \text{Schedule}(\text{th}, \text{tl''}) \rightarrow \exists\, \text{tl}, \text{act}\ \text{Schedule}(\text{th}, \text{tl}) \wedge \text{Movie}(\text{tl}, \text{dir}, \text{act})\big) \Big\}$$

- Define, using M for Movie and S for Schedule,

$$D = \pi_{\text{director}}(M), \quad T = \pi_{\text{theater}}(S), \quad DT = \pi_{\text{director,theater}}(M \bowtie S)$$

- D has all directors, T has all theaters,
  DT has all directors and theaters where their movies are playing

- Our query is (mixing slightly logic and algebra):

$$\{\, d \mid d \in D \wedge \forall t\,(t \in T \rightarrow (d,t) \in DT)\,\}$$

Queries with "All" (cntd)

- We can rewrite the query $\{ d \mid d \in D \wedge \forall t\,(t \in T \rightarrow (d,t) \in DT) \}$ as

$$\{ d \mid d \in D \wedge \neg\exists t\,(t \in T \wedge (d,t) \notin DT) \}$$

- This is the relative complement in D of the query

$$\{ d \mid d \in D \wedge \exists t\,(t \in T \wedge (d,t) \notin DT) \},$$

- This can be equivalently transformed into

$$\{ d \mid \exists t\,(d \in D \wedge t \in T \wedge (d,t) \notin DT) \},$$

- Finally, this can be expressed as

$$\pi_{\mathsf{director}}(D \times T \setminus DT)$$

Queries with "All" (cont'd)

- Hence, the answer to the entire query is

$$D \setminus \pi_{\mathsf{director}}(D \times T \setminus DT).$$

- Putting everything together, the answer is:

$$\pi_{\mathsf{director}}(M) \setminus \pi_{\mathsf{director}}\Big(\pi_{\mathsf{director}}(M) \times \pi_{\mathsf{theater}}(S) \setminus \pi_{\mathsf{director,theater}}(M \bowtie S)\Big)$$

- This is much less intuitive than the logical description of the query.

Safe-Range Queries

Safe range queries are a syntactically defined fragment of Relational Calculus that contains *only* domain-independent queries

(and thus are also a fragment of DI-RelCalc)

- One can show: Safe-Range RelCalc $\equiv$ DI-RelCalc
- Steps in defining safe-range queries:
  - a syntactic *normal form* of the queries
  - a mechanism for determining whether a variable is *range restricted*

  Then a query is safe-range iff all its free variables are range-restricted.

Safe-Range Normal Form (SRNF)

Equivalently rewrite query formula φ

- **Rename variables apart:** Rename variables such that each variable x is quantified at most once and has only free or only bound occurrences.
- **Eliminate $\forall$:** Rewrite $\forall x\, \varphi \;\mapsto\; \neg\exists x\, \neg\varphi$
- **Eliminate implications:** Rewrite $\varphi \rightarrow \psi \;\;\mapsto\;\; \neg\varphi \vee \psi$ (and similarly for $\leftrightarrow$)
- **Push negation down as far as possible:** Use the rules

  $\neg\neg\varphi \;\mapsto\; \varphi$

  $\neg(\varphi_1 \wedge \varphi_2) \;\mapsto\; \neg\varphi_1 \vee \neg\varphi_2)$

  $\neg(\varphi_1 \vee \varphi_2) \;\mapsto\; \neg\varphi_1 \wedge \neg\varphi_2)$
- *Flatten 'and's:* No child of an 'and' in the formula parse tree is an 'and'.
  Similarly for 'or's, and '$\exists$'s

Safe-Range Normal Form /2

- The result of rewriting a query Q is called *SRNF*(Q)
- A query Q is in *safe-range normal form* if $Q =$ *SRNF*(Q)
- Examples:

Q_1(th) = ∃ tl ∃ dir (Movie(tl, dir,'Nicholson') ∧ Schedule(th,tl))

SRNF(Q_1) = ∃ tl, dir (Movie(tl, dir,'Nicholson') ∧ Schedule(th,tl))

Q_2(dir) = ∀ th ∀ tl' (Schedule(th,tl') → ∃ tl ∃ act (Schedule(th,tl) ∧ Movie(tl, dir, act)))

SRNF(Q_2) = ¬∃ th, tl' (Schedule(th,tl') ∧ ¬∃ tl, act (Schedule(th,tl) ∧ Movie(tl, dir, act)))

Range Restriction

Three elements:

- Syntactic condition on formulas in SRNF.
- Intuition: all possible values of a variable lie in the active domain.
- If a variable does not fulfill this, then the query is rejected

Algorithm Range Restriction (rr)

Input: formula φ in SRNF

Output: subset of the free variables or $\perp$ (indicating that a quantified variable is not range restricted)

case φ **of**

$R(t_1, \ldots, t_n)$: $rr(\varphi) :=$ the set of variables from $t_1, \ldots, t_n$.

$x = a, a = x$: $rr(\varphi) := \{x\}$

$\varphi_1 \wedge \varphi_2$: $rr(\varphi) := rr(\varphi_1) \cup rr(\varphi_2)$

$\varphi_1 \wedge x = y$: **if** $\{x, y\} \cap rr(\varphi_1) = \emptyset$ **then** $rr(\varphi) := rr(\varphi_1)$
else $rr(\varphi) := rr(\varphi_1) \cup \{x, y\}$

$\varphi_1 \vee \varphi_2$: $rr(\varphi) := rr(\varphi_1) \cap rr(\varphi_2)$

$\neg\varphi_1$: $rr(\varphi) := \emptyset$

$\exists x_1, \ldots, x_n \varphi_1$: **if** $\{x_1, \ldots, x_n\} \subseteq rr(\varphi_1)$ **then** $rr(\varphi) := rr(\varphi_1) \setminus \{x_1, \ldots, x_n\}$
else return $\perp$

end case

Here, $S \cup \perp = \perp \cup S = \perp$ and similarly for $\cap, \setminus$

Range Restriction (cntd)

Examples (contd):

SRNF(Q_1) = $\exists$ tl, dir (Movie(tl, dir,'Nicholson') $\wedge$ Schedule(th,tl))

$rr($*SRNF*$(Q_1))$ = {th}

SRNF(Q_2) = $\neg\exists$ th, tl' (Schedule(th,tl') $\wedge$ $\neg\exists$ tl, act (Schedule(th,tl) $\wedge$ Movie(tl, dir, act)))

$rr($*SRNF*$(Q_2))$ = {}

Safe-Range Calculus

Definition. A query $Q(\vec{x})$ in Relational Calculus is *safe-range* iff

$$\textit{rr}(\textit{SRNF}(Q)) = \textit{free}(Q).$$

The set of all safe-range queries is denoted by SR-RelCalc.

Intuition: A query is safe-range iff *all* its variables are bound by a database atom or by an equality atom.

Examples: Q_1 is a safe-range query, while Q_2 is not.

Theorem. SR-RelCalc $\equiv$ DI-RelCalc

(The proof of this theorem is technically involved.)

"For All" and Negation in SQL

- Two main mechanisms: set theoretic operators and subqueries
- Subqueries are often more natural
- SQL syntax for $R \cap S$:

```
R INTERSECT S
```

- SQL syntax for $R \setminus S$:

```
R EXCEPT S
```

- Find all actors who are not directors resp. also directors:

```
SELECT Actor AS Person
FROM Movie
  EXCEPT
SELECT Director AS Person
FROM Movie
```

```
SELECT Actor AS Person
FROM Movie
  INTERSECT
SELECT Director AS Person
FROM Movie
```

"For All" and Negation in SQL /2

Subqueries with `NOT EXISTS, NOT IN`

- Example: Who are the directors whose movies are playing in all theaters?
- SQL's way of saying this: Find directors such that there does not exist a theater where their movies do not play.

```
SELECT M1.Director
FROM Movie M1
WHERE NOT EXISTS (SELECT S.Theater
                  FROM Schedule S
                  WHERE NOT EXISTS (SELECT M2.Director
                                    FROM Movie M2
                                    WHERE M2.Title=S.Title AND
                                          M1.Director=M2.Director))
```


"For All" and Negation in SQL /2

Same query using `EXCEPT`.

```
SELECT M.Director
FROM   Movie M
WHERE  NOT EXISTS (SELECT S.Theater
                   FROM Schedule S
                   EXCEPT
                   SELECT S1.Theater
                   FROM Schedule S1, Movie M1
                   WHERE S1.Title=M1.Title AND
                         M1.Director=M.Director)
```

- Other conditions: `IN`, `NOT IN`, `EXISTS`

More examples of nested queries: using EXISTS and IN

Find directors whose movies are playing at Le Champo:

- ```
  SELECT M.Director
  FROM Movie M
  WHERE EXISTS (SELECT *
                FROM Schedule S
                WHERE S.Title=M.Title AND
                      S.Theater='Le Champo')
  ```
- ```
  SELECT M.Director
  FROM Movie M
  WHERE M.Title IN (SELECT S.Title
                    FROM Schedule S
                    WHERE S.Theater='Le Champo')
  ```


More examples of nested queries: using NOT IN

Find actors who did not play in a movie by Kubrick.

- ```
  SELECT M.Actor
  FROM Movie M
  WHERE M.Actor NOT IN
           (SELECT M1.Actor
            FROM Movie M1
            WHERE M1.Director='Kubrick')
  ```

The subquery finds actors playing in some movie by Kubrick; the top two lines take the complement of that.
  ```

Bibliography

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems – The Complete Book*. Prentice Hall, 2002.

[3] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Rockville, Md., 1983.

[4] J. D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.