8.2 First-Order Logic

The relational calculus is a specialization of first-order logic.

8.2.1 Syntax

- each **first-order language** contains the following distinguished symbols:
  - "(" and ")", logical symbols $\neg$, $\wedge$, $\vee$, $\rightarrow$, quantifiers $\forall$, $\exists$,
  - an infinite set of variables $X, Y, X_1, X_2, \ldots$.
- An individual first-order language is then given by its **signature** Σ. Σ contains **function symbols** and **predicate symbols**, each of them with a given arity.

Aside/Preview: First-Order Modeling Styles

- the choice between predicate and function symbols and different arities allows multiple ways of modeling (see Slide 435).

For databases:

- the relation names are the predicate symbols (with arity), e.g. $continent/2$, $encompasses/3$, etc.
- there are only 0-ary function symbols, i.e., **constants**; in a relational database these are only the literal values (numbers and strings).
- thus, the database schema $\mathbf{R}$ is the signature.

Syntax (Cont'd)

Terms

The set of **terms** over Σ, Term$_\Sigma$, is defined inductively as

- each variable is a term,
- for every function symbol $f \in \Sigma$ with arity n and terms $t_1, \ldots, t_n$, also $f(t_1, \ldots, t_n)$ is a term.

  0-ary function symbols: c, 1,2,3,4, "Berlin",...

  Example: for $plus/2$, the following are terms: $plus(3, 4)$, $plus(plus(1, 2), 4)$, $plus(X, 2)$.
- **ground terms** are terms without variables.

For databases:

- since there are no function symbols,
- the only terms are the **constants** and **variables**
  e.g., 1, 2, "D", "Germany", X, Y, etc.

Syntax (Cont'd): Formulas

Formulas are built inductively (using the above-mentioned special symbols) as follows:

Atomic Formulas

(1) For a predicate symbol (i.e., a relation name) R of arity k, and terms $t_1, \ldots, t_k$, $R(t_1, \ldots, t_k)$ is a formula.

(2) (**for databases only, as special predicates**)
A **selection condition** is an expression of the form $t_1 \, \theta \, t_2$ where t_1, t_2 are terms, and θ is a comparison operator in $\{=, \neq, \leq, <, \geq, >\}$.
Every selection condition is a formula.

(both are also called **positive literals**)

For databases:

- the atomic formulas are the **predicates** built over relation names and these constants, e.g.,
  continent("Asia",4.5E7), encompasses("R","Asia",X), country(N,CC,Cap,Prov,Pop,A).
- comparison predicates (i.e., the "selection conditions") are atomic formulas, e.g.,
  $X =$ "Asia", $Y > 10.000.000$ etc.

Syntax (Cont'd)

Compound Formulas

(3) For a formula F, also $\neg F$ is a formula. If F is an atom, $\neg F$ is called a **negative literal**.

(4) For a variable X and a formula F, $\forall X : F$ and $\exists X : F$ are formulas. F is called the **scope** of $\exists$ or $\forall$, respectively.

(5) For formulas F and G , the **conjunction** $F \wedge G$ and the **disjunction** $F \vee G$ are formulas.

For formulas F and G, where G (regarded as a string) is contained in F, G is a **subformula** of F.

The usual priority rules apply (allowing to omit some parentheses).

- instead of $F \vee \neg G$, the **implication** syntax $F \leftarrow G$ or $G \rightarrow F$ can be used, and
- $(F \rightarrow G) \wedge (F \leftarrow G)$ is denoted by the **equivalence** $F \leftrightarrow G$.

Syntax (Cont'd)

Bound and Free Variables

An occurrence of a variable X in a formula is

- **bound** (by a quantifier) if the occurrence is in a formula A inside $\exists X : A$ or $\forall X : A$ (i.e., in the scope of an appropriate quantifier).
- **free** otherwise, i.e.,if it is not bound by any quantifier.

Formulas without free variables are called **closed**.

Example:

- $continent($“Asia”$, X)$: X is free.
- $continent($“Asia”$, X) \wedge X > 10.000.000$: X is free.
- $\exists X : (continent($“Asia”$, X) \wedge X > 10.000.000)$: X is bound.
  The formula is closed.
- $\exists X : (continent(X, Y))$: X is bound, Y is free.
- $\forall Y : (\exists X : (continent(X, Y)))$: X and Y are bound.
  The formula is closed.

Outlook:

- closed formulas either hold in a database state, or they do not hold.
- free variables represent answers to queries:
  ?- $continent$("Asia", X) means "for which value x does $continent$("Asia", x) hold?"
  Answer: for $x = 4.5E7$.
- $\exists Y : (continent(X, Y))$: means
  "for which values x is there an y such that $continent(x, y)$ holds? – we are not interested in the value of y"
  The answer are all names of continents, i.e., that x can be "Asia", "Europe", or ...

... so we have to **evaluate** formulas ("semantics").

8.2.2 Semantics

The semantics of first-order logic is given by **first-order structures** over the signature:

First-Order Structure

A **first-order structure** $\mathcal{S} = (I, \mathcal{D})$ over a signature Σ consists of a nonempty set $\mathcal{D}$ (**domain**; often also denoted by $\mathcal{U}$ (**universe**)) and an interpretation I of the signature symbols over $\mathcal{D}$ which maps

- every constant c to an element $I(c) \in \mathcal{D}$,
- every n-ary function symbol f to an n-ary function $I(f) : \mathcal{D}^n \to \mathcal{D}$
  (note that for relational databases, there are no function symbols with arity > 0)
- every n-ary predicate symbol p to an n-ary relation $I(p) \subseteq \mathcal{D}^n$.

General:

- constants are interpreted by elements of the domain
- predicate symbols and function symbols are *not* mapped to domain objects, but to relations/functions over the domain.
  $\Rightarrow$ First-order logic cannot express relations/relationships between predicates/functions.

Aside/Preview: First-Order-based Semantic Styles

- There are different frameworks that are based on first-order logic that specialize/simplify FOL (see Slide 435).
- Higher-Order logics allow to make statements about predicates and/or functions by higher-order predicates.

First-Order Structures: An Example

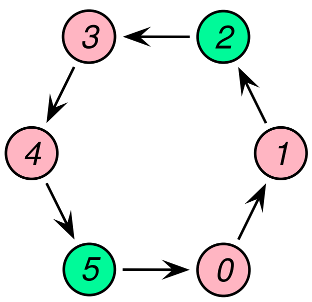
Example 8.1 (First-Order Structure)

Signature: *constant symbols:* $zero,\ one,\ two,\ three,\ four,\ five$

predicate symbols: $green/1,\ red/1,\ sees/2$

function symbols: $to\_right/1$, $plus/2$

Structure $\mathcal{S}$*:*



Domain $\mathcal{D} = \{0,\ 1,\ 2,\ 3,\ 4,\ 5\}$

Interpretation of the signature:

$I(zero) = 0,\ I(one) = 1, \ldots, I(five) = 5$

$I(green) = \{(2),\ (5)\},\ \ I(red) = \{(0),\ (1),\ (3),\ (4)\}$

$I(sees) = \{(0,3),\ (1,4),\ (2,5),\ (3,0),\ (4,1),\ (5,2)\}$

$I(to\_right) = \{\ (0) \mapsto (1),\ (1) \mapsto (2),\ (2) \mapsto (3),$
$(3) \mapsto (4),\ (4) \mapsto (5),\ (5) \mapsto (0)\}$

$I(plus) = \{(n, m) \mapsto (n + m)$ mod $6 \mid n, m \in \mathcal{D}\}$

Terms: $one,\ to\_right(four),\ to\_right(to\_right(X)),\ to\_right(to\_right(to\_right(four))),$
$plus(X, to\_right(zero)),\ to\_right(plus(to\_right(four), five))$

Atomic Formulas: $green(one),\ red(to\_right(to\_right(to\_right(four)))),\ sees(X, Y),$
$sees(X, to\_right(Z))$, $sees(to\_right(to\_right(four)), to\_right(one))$,
$plus(to\_right(to\_right(four)), to\_right(one)) = to\_right(three)$ □

Summary: Notions for Databases

- a set $\mathbf{R}$ of relational schemata; logically spoken, $\mathbf{R}$ is the **signature**,
- a database state is a structure $\mathcal{S}$ over $\mathbf{R}$
- $\mathcal{D}$ contains all domains of attributes of the relation schemata,
- for every single relation schema $R = (\bar{X})$ where $\bar{X} = \{A_1, \ldots, A_k\}$, we write $R[A_1, \ldots, A_k]$. k is the **arity** of the relation name R.
- relation names are the predicate symbols. They are interpreted by relations, e.g., $I(encompasses)$
  (which we also write as $\mathcal{S}(encompasses)$).

For Databases:

- no function symbols with arity > 0
- constants are interpreted "by themselves":
  $I(4) = 4$, $I($"Asia"$) =$ "Asia"
- care for domains of attributes.

Evaluation of Terms and Formulas

Terms and formulas must be **evaluated** under a given interpretation – i.e., wrt. a given database state $\mathcal{S}$.

- Terms can contain variables.
- variables are not interpreted by $\mathcal{S}$.

A **variable assignment** over a universe $\mathcal{D}$ is a mapping

$$\beta : Variables \rightarrow \mathcal{D} .$$

For a variable assignment β, a variable X, and $d \in \mathcal{D}$, the **modified** variable assignment β_X^d is identical with β except that it assigns d to the variable X:

$$\beta_X^d = \begin{cases} Y \mapsto \beta(Y) & \text{for } Y \neq X , \\ X \mapsto d & \text{otherwise.} \end{cases}$$

Example 8.2
For variables X, Y, Z, $\beta = \{X \mapsto 1,\ Y \mapsto$ "Asia"$, Z \mapsto 3.14\}$ is a variable assignment.

$\beta_X^3 = \{X \mapsto 3,\ Y \mapsto$ "Asia"$, Z \mapsto 3.14\}$. □

Evaluation of Terms

Terms and formulas are interpreted

- wrt. a given structure $\mathcal{S} = (I, \mathcal{D})$, and
- wrt. a given variable assignment β.

Every structure $\mathcal{S}$ together with a variable assignment β induces an evaluation $\mathcal{S}$ of terms and predicates:

- Terms are mapped to elements of the universe: $\mathcal{S}$: $\text{Term}_\Sigma \times \beta \rightarrow \mathcal{D}$
- (Closed) formulas are true or false in a structure: $\mathcal{S}$: $\text{Fml}_\Sigma \times \beta \rightarrow \{\text{true, false}\}$

For Databases:

- Σ is a purely relational signature,
- $\mathcal{S}$ is a database state for Σ,
- no function symbols with arity > 0, no nontrivial terms,
- constants are interpreted "by themselves".

Evaluation of Terms

$$\mathcal{S}(x, \beta) := \beta(x) \quad \text{for a variable } x\ ,$$
$$\mathcal{S}(c, \beta) := I(c) \quad \text{for any constant } c\ .$$
$$\mathcal{S}(f(t_1, \ldots, t_n), \beta) := (I(f))(\mathcal{S}(t_1, \beta), \ldots, \mathcal{S}(t_n, \beta))$$

for a function symbol $f \in \Sigma$ with arity n and terms $t_1, \ldots, t_n$.

Example 8.3 (Evaluation of Terms)
Consider again Example 8.1.

- *For variable-free terms:* $\beta = \emptyset$.
- $\mathcal{S}(one, \emptyset) = I(one) = 1$
- $\mathcal{S}(to\_right(four), \emptyset) = I(to\_right(\mathcal{S}(four, \emptyset)) = I(to\_right(4)) = 5$
- $\mathcal{S}(to\_right(to\_right(to\_right(four))), \emptyset) = I(to\_right(\mathcal{S}(to\_right(to\_right(four)), \emptyset))) =$
  $I(to\_right(I(to\_right(\mathcal{S}(to\_right(four), \emptyset))))) =$
  $I(to\_right(I(to\_right(I(to\_right(\mathcal{S}(four)), \emptyset))))) =$
  $I(to\_right(I(to\_right(I(to\_right(4), \emptyset))))) =$
  $I(to\_right(I(to\_right(5)))) = I(to\_right(0)) = 1$ □

Example 8.3 (Continued)

- *Let* $\beta = \{X \mapsto 3\}$.
  $\mathcal{S}(to\_right(to\_right(X)), \beta) = I(to\_right(\mathcal{S}(to\_right(X), \beta))) =$
  $I(to\_right(I(to\_right(\mathcal{S}(X, \beta))))) = I(to\_right(I(to\_right(\beta(X))))) =$
  $I(to\_right(I(to\_right(3)))) = I(to\_right(4)) = 5$

- *Let* $\beta = \{X \mapsto 3\}$.
  $\mathcal{S}(plus(X, to\_right(zero)), \emptyset) = I(plus(\mathcal{S}(X, \beta), \mathcal{S}(to\_right(zero), \beta))) =$
  $I(plus(\beta(X), I(to\_right(\mathcal{S}(zero, \beta))))) = I(plus(3, I(to\_right(I(zero))))) =$
  $I(plus(3, I(to\_right(0)))) = I(plus(3, 1)) = 4$ $\Box$

EVALUATION OF FORMULAS

Formulas can either hold, or not hold in a database state.

Truth Value

Let F a formula, $\mathcal{S}$ an interpretation, and β a variable assignment of the free variables in F (denoted by $free(F)$).

Then we write $\mathcal{S} \models_\beta F$ if "F is true in $\mathcal{S}$ wrt. β".

Formally, $\models$ is defined inductively.

Truth Values of Formulas: Inductive Definition

Motivation: variable-free atoms

For an atom $R(a_1, \ldots, a_k)$, where a_i, $1 \leq i \leq k$ are constants,

$R(a_1, \ldots, a_k)$ is **true** in $\mathcal{S}$ if and only if $(I(a_1), \ldots, I(a_k)) \in \mathcal{S}(R)$.

Otherwise, $R(a_1, \ldots, a_k)$ is **false** in $\mathcal{S}$.

Base Case: Atomic Formulas

The **truth value** of an atom $R(t_1, \ldots, t_k)$, where t_i, $1 \leq i \leq k$ are terms, is given as

$\mathcal{S} \models_\beta R(t_1, \ldots, t_k)$ if and only if $(\mathcal{S}(t_1, \beta), \ldots, \mathcal{S}(t_k, \beta)) \in \mathcal{S}(R)$.

For Databases:

- the t_i can only be constants or variables.

Truth Values of Formulas: Inductive Definition

- $t_1 \, \theta \, t_2$ with θ a comparison operator in $\{=,\neq,\leq,<,\geq,>\}$:
  $\mathcal{S} \models_\beta t_1 \, \theta \, t_2$ if and only if $\mathcal{S}(t_1, \beta) \; \theta \; \mathcal{S}(t_2, \beta)$ holds.
- $\mathcal{S} \models_\beta \neg G$ if and only if $\mathcal{S} \not\models_\beta G$.
- $\mathcal{S} \models_\beta G \wedge H$ if and only if $\mathcal{S} \models_\beta G$ and $\mathcal{S} \models_\beta H$.
- $\mathcal{S} \models_\beta G \vee H$ if and only if $\mathcal{S} \models_\beta G$ or $\mathcal{S} \models_\beta H$.
- (Derived; cf. next slide) $\mathcal{S} \models_\beta F \rightarrow G$ if and only if $\mathcal{S} \models_\beta \neg F$ or $\mathcal{S} \models_\beta G$.
- $\mathcal{S} \models_\beta \forall X G$ if and only if for all $d \in \mathcal{D}$, $\mathcal{S} \models_{\beta^d_X} G$.
- $\mathcal{S} \models_\beta \exists X G$ if and only if for some $d \in \mathcal{D}$, $\mathcal{S} \models_{\beta^d_X} G$.

Derived Boolean Operators

There are some minimal sets (e.g. $\{\neg, \wedge, \exists\}$) of boolean operators from which the others can be derived:

- The **implication** syntax $F \rightarrow G$ is a shortcut for $\neg F \vee G$ (cf. Slide 416):
  $\mathcal{S} \models_\beta F \rightarrow G$ if and only if $\mathcal{S} \models_\beta \neg F$ or $\mathcal{S} \models_\beta G$.
  "whenever F holds, also G holds" – this is called *material implication* instead of "causal implication".
  Note: *if* F implies G causally in a scenario, then all (possible) states satisfy $F \rightarrow G$.
- note that $\wedge$ and $\vee$ can also be expressed by each other, together with $\neg$:
  $F \wedge G$ is equivalent to $\neg(\neg F \vee \neg G)$, and $F \vee G$ is equivalent to $\neg(\neg F \wedge \neg G)$.
- The quantifiers $\exists$ and $\forall$ are in the same way "dual" to each other:
  $\exists x : F$ is equivalent to $\neg \forall x : (\neg F)$, and $\forall x : F$ is equivalent to $\neg \exists x : (\neg F)$.
- Proofs: exercise.
  Show e.g. by the definitions that whenever $\mathcal{S} \models_\beta \exists x : F$ then $\mathcal{S} \models_\beta \neg \forall x : (\neg F)$.

Example 8.4 (Evaluation of Atomic Formulas)
Consider again Example 8.1.

- *For variable-free formulas, let* $\beta = \emptyset$
- $\mathcal{S} \models_\emptyset green(one) \;\Leftrightarrow\; \mathcal{S}(one) \in I(green) \;\Leftrightarrow\; (1) \in I(green)$ *– which is not the case.*
  Thus, $\mathcal{S} \not\models_\emptyset green(one)$.
- $\mathcal{S} \models_\emptyset red(to\_right(to\_right(to\_right(three)))) \;\Leftrightarrow$
  $$(\mathcal{S}(to\_right(to\_right(to\_right(three))), \emptyset)) \in I(red) \;\Leftrightarrow\; (0) \in I(red)$$
  which is the case. Thus, $\mathcal{S} \models_\emptyset red(to\_right(to\_right(to\_right(three))))$.
- *Let* $\beta = \{X \mapsto 3,\ Y \mapsto 5\}$.
  $\mathcal{S} \models_\beta sees(X, Y) \;\Leftrightarrow\; (\mathcal{S}(X, \beta), \mathcal{S}(Y, \beta)) \in I(sees) \;\Leftrightarrow\; (3, 5) \in I(sees)$
  which is not the case.
- *Again,* $\beta = \{X \mapsto 3,\ Y \mapsto 5\}$.
  $\mathcal{S} \models_\beta sees(X, to\_right(Y)) \;\Leftrightarrow\; (\mathcal{S}(X, \beta), \mathcal{S}(to\_right(Y), \beta)) \in I(sees) \;\Leftrightarrow\; (3, 0) \in I(sees)$
  which is the case.
- $\mathcal{S} \models_\beta plus(to\_right(to\_right(four)), to\_right(one)) = to\_right(three) \;\Leftrightarrow$
  $$\mathcal{S}(plus(to\_right(to\_right(four)), to\_right(one)), \emptyset) = \mathcal{S}(to\_right(three), \emptyset) \;\Leftrightarrow\; 2 = 4$$
  which is not the case. □

Example 8.5 (Evaluation of Compound Formulas)
Consider again Example 8.1.

- $\mathcal{S} \models_{\emptyset} \exists X : red(X) \Leftrightarrow$

  there is a $d \in \mathcal{D}$ *such that* $\mathcal{S} \models_{\emptyset^d_X} red(X) \Leftrightarrow$ *there is a* $d \in \mathcal{D}$ *s.t.* $\mathcal{S} \models_{\{X \mapsto d\}} red(X)$

  Since we have shown above that $\mathcal{S} \models_{\emptyset} red(6)$*, this is the case.*

- $\mathcal{S} \models_{\emptyset} \forall X : green(X) \Leftrightarrow$

  for all $d \in \mathcal{D}$, $\mathcal{S} \models_{\emptyset^d_X} green(X) \Leftrightarrow$ *for all* $d \in \mathcal{D}$, $\mathcal{S} \models_{\{X \mapsto d\}} green(X)$

  Since we have shown above that $\mathcal{S} \not\models_{\emptyset} green(1)$ *this is not the case.*

- $\mathcal{S} \models_{\emptyset} \forall X : (green(X) \vee red(X)) \Leftrightarrow$ *for all* $d \in \mathcal{D}$, $\mathcal{S} \models_{\{X \mapsto d\}} (green(X) \vee red(X))$.
  One has now to check whether $\mathcal{S} \models_{\{X \mapsto d\}} (green(X) \vee red(X))$ *for all* $d \in domain$.
  We do it for $d = 3$*:*

  $\mathcal{S} \models_{\{X \mapsto 3\}} (green(X) \vee red(X)) \Leftrightarrow$

  $\mathcal{S} \models_{\{X \mapsto 3\}} green(X)$ *or* $\mathcal{S} \models_{\{X \mapsto 3\}} red(X) \Leftrightarrow$

  $(\mathcal{S}(X, \{X \mapsto 3\})) \in I(green)$ *or* $(\mathcal{S}(X, \{X \mapsto 3\})) \in I(red) \Leftrightarrow$

  $(3) \in I(green)$ *or* $(3) \in I(red)$

  which is the case since $(3) \in I(red)$.

- *Similarly,* $\mathcal{S} \not\models_{\emptyset} \forall X : (green(X) \wedge red(X))$ □

Some Notions

Consider a formula F with some free variables.

- *$\mathcal{S}$ is a model for F under β* if $\mathcal{S} \models_{\beta} F$.
- (for closed formulas: $\mathcal{S}$ *is a model for* F if $\mathcal{S} \models F$)
- F is *satisfiable* if F has some model (e.g., $F = \exists x, y : (p(x) \wedge q(x, y))$ is satisfiable).
- F is *unsatifisfiable* if F has no model (e.g., $F = \exists x : (p(x) \wedge \neg p(x)$ is unsatisfiable)
- F is *valid* (german: "allgemeingültig") if F holds in every structure:
  (e.g., $F = (\forall x : (p(x) \to q(x)) \wedge \forall y : (q(y) \to r(y))) \to \forall z : (p(z) \to r(z)))$ is valid)
  Application: verification of a system has the goal to show that $\varphi \to \psi$ is valid where φ is a formula that contains the specification (usually a large conjunction) and φ is a conjunction of guaranteed properties.
- two FOL formulas F and G are *equivalent*, $F \equiv G$ if every model of F is also a model of G and vice versa.
- a FOL formula *F entails a FOL formula G*, $F \models G$ if every model of F is also a model of G. (note the overloading of $\models$ for $\mathcal{S} \models F$ and $F \models G$).

Example 8.6

For the following pairs F and G of formulas, check whether one implies the other (if not, give a counterexample), and whether they are equivalent:

1. $F = (\forall x : p(x)) \vee (\forall x : q(x)),\quad G = \forall v : (p(v) \vee q(v)).$
2. $F = \forall x : ((\exists y : p(y)) \to q(x)),\quad G = \forall v, \forall w : p(v) \to q(w).$
3. $F = \forall x : \exists y : p(x, y),\quad G = \exists v : \forall w : p(v, w).$ □

8.3 FOL-based Modeling Styles and Frameworks

- Full FOL allows for several restrictions, shortcuts and extensions
- variants developed depending on the application and the intended reasoning mechanisms.

Recall

- note: the FOL signature is disjoint from the domain $\mathcal{D}$, e.g. $\mathsf{germany}$ is a constant symbol, mapped to the element *germany* $\in \mathcal{D}$.
- each FOL signature consists of
  - predicate symbols
    - 0-ary predicates: “boolean predicates”, just being interpreted as true/false (formally $I(p_0) \subseteq \mathcal{D}^0$, where $\mathcal{D}^0 = 1$ means true, while $\emptyset$ means false).
    - n-ary predicates, interpreted as $\mathcal{I}(p) \subseteq \mathcal{D}^n$.
  - function symbols
    - 0-ary functions: constants, interpreted by elements of the domain. (formally $I(c) : \mathcal{D}^0 \to \mathcal{D}$, e.g. for the constant $\mathsf{germany}$: $I(\mathsf{germany}) : () \mapsto$ *germany*; $\mathcal{S}(\mathsf{germany}) = \mathcal{I}(\mathsf{germany}()) =$ *germany*)
    - n-ary functions, interpreted as $\mathcal{I}(f) : \mathcal{D}^n \to \mathcal{D}$.

8.3.1 FOL with (atomic) Datatypes

Common extension: FOL($D_1, \ldots, D_n$) where $D_1, \ldots, D_n$ are datatypes like strings, numbers, dates.

- for these, the values are both 0-ary constant symbols and elements of the domain,
- appropriate predicates and functions are contained in the signature and as built-in predicates and functions (i.e., are not explicitly mentioned when giving an interpretation).

Example 8.1 revisited

Example 8.1 can be formulated in FOL(INT):

- integers $0, 1, 2, \ldots \in \Sigma$ as constant symbols (instead of *one, two, …*).
- $I(0) = 0$, $I(1) = 1$, … is implicit.
- no interpretation of the constant symbols *one, two, …* required.
- function $+/2$ (i.e., binary function "+") instead of *plus/2*, its interpretation comes implicitly from integers.
- interpretation of user-defined predicates *green, sees, to_right* as before (over the domain $\mathcal{D} \supseteq INT$) .

8.3.2 Purely Relational Object-Oriented Modeling

- Closely related with the ER Model:
- the domain $\mathcal{D}$ contains instances/individuals/"resources" *germany*, *berlin*, … and datatype literals.
- – Entity types = Classes: unary predicates
    germany $\in I(\text{Country})$, *berlin* $\in I(\text{City})$, *eu* $\in I(\text{Organization})$.
  – Attributes: binary predicates
    (*germany*, "Germany") $\in I(\text{name})$,
    (*berlin*, "3472009") $\in I(\text{population})$
  – Relationships: binary predicates
    (*germany*, *berlin*) $\in I(\text{capital})$,
    (*germany*, *eu*) $\in I(\text{isMember})$.
- closely related: RDF – Resource Description Framework as the data model underlying the Semantic Web (cf. Slide 440).
- closely related: Specific family of logics called "Description Logic" as a *decidable* subset of FOL (cf. Slide 441)

Examples

The following sets specify answers to sample queries:

- Names of all countries such that there is a city with more than 1,000,000 inhabitants in the country:

  $\{n \mid \ \exists x : \mathsf{Country}(x) \wedge \mathsf{name}(x,n) \wedge$

  $\quad \exists y,p : (\mathsf{City}(y) \wedge \mathsf{inCountry}(x,y) \wedge \mathsf{population}(y,p) \wedge p > 1,000,000)\ \}$

- Names of all countries such that all its cities have more than 1,000,000 inhabitants:

  $\{n \mid \ \exists x : \mathsf{Country}(x) \wedge \mathsf{name}(x,n) \wedge$

  $\quad \forall y : (\mathsf{City}(y) \wedge \mathsf{inCountry}(x,y) \rightarrow \exists p : (\mathsf{population}(y,p) \wedge p > 1,000,000))\ \}$

- Names of all countries such that the capital of the country has more than 1,000,000 inhabitants:

  $\{n \mid \ \exists x : \mathsf{Country}(x) \wedge \mathsf{name}(x,n) \wedge$

  $\quad \exists y,p : (\mathsf{City}(y) \wedge \mathsf{capital}(x,y) \wedge \mathsf{population}(y,p) \wedge p > 1,000,000)\ \}$

- Names of all countries such that the country is a member of the organization with abbreviation “EU”:

  $\{n \mid \ \exists x : \mathsf{Country}(x) \wedge \mathsf{name}(x,n) \wedge$

  $\quad \exists o : (\mathsf{Organization}(o) \wedge \mathsf{abbrev}(o,\text{“EU”}) \wedge \mathsf{isMember}(x,o))\ \}$

Problem

⇒ attributed relationships (like isMember with membertype) can only be modeled via reification.

Example

(***deInEU***) $\in I(\mathsf{Membership})$,
(***deInEU***, ***germany***) $\in I(\mathsf{ofCountry})$.
(***deInEU***, ***eu***) $\in I(\mathsf{inOrganization})$.
(***deInEU***, “full member”) $\in I(\mathsf{memberType})$.

Names of all countries such that the country is a member of the organization with abbreviation “EU”:

$\{n \mid \exists x : (\mathsf{Country}(x) \wedge \mathsf{name}(x,n) \wedge$

$\quad \exists o,m,t : (\ \mathsf{Organization}(o) \wedge \mathsf{abbrev}(o,\text{“EU”}) \wedge$

$\qquad \wedge\ \mathsf{Membership}(m) \wedge \mathsf{ofCountry}(m,x) \wedge \mathsf{inOrganization}(m,o) \wedge \mathsf{memberType}(m,t)))\ \}$

RDF – Resource Description Framework

- most prominent Semantic Web data model.
- graph-based: objects and literals are nodes, properties are the edges.
- instance data represented by (subject predicate object) triples that can be seen as unary (class membership) and binary (properties and relationships) predicates:

:germany a mon:Country.	– Country(germany)
:germany mon:name “Germany”	– name(germany, “Germany”)
:germany mon:population 83536115.	– population(germany, 83536115)
:germany mon:capital :berlin.	– capital(germany, berlin)

- optional: XML serialization
- domain: URIs and literals (using the XML namespace concept)
  - URIs serve as constant symbols and (web-wide) object/resource identifiers,
  - property and class names are also URIs.

Description Logics

- traditional framework, became popular as a base for the Semantic Web,
- subset of FOL where the *formulas* are restricted,

⇒ modular family of logics, most of which are decidable.

- special syntax that can be translated into the 2-variable fragment of FOL (decidable).
- focus of DL is on the definition of concepts:

$$\mathit{CoastCity} \equiv \mathit{City} \sqcap \exists \mathit{locatedAt}.\mathit{Sea}\ .$$

FOL: $\forall x : \mathit{CoastCity}(x) \leftrightarrow \mathit{City}(x) \wedge \exists y : (\mathit{locatedAt}(x, y) \wedge \mathit{Sea}(y))$.

8.3.3 FOL Object-Oriented Modeling with Functions

- $\mathcal{S} = (I, \mathcal{D})$ as follows:
- the domain $\mathcal{D}$ contains elements *germany*, *berlin*, ... and datatype literals
- Predicates Country/1, City/1, Organization/1, ismember/2 etc. as before,
- functions capital/1, headq/1, population/1 for *functional* attributes and relationships:
  (*germany*) $\mapsto$ *berlin* $\in I$(capital),
  (*eu*) $\mapsto$ *brussels* $\in I$(headq),
  (*berlin*) $\mapsto 3472009 \in I$(population).
- some example formula that evaluates to true:

$$\mathcal{S} \models \exists o, c : \text{Organization}(o) \wedge \text{name}(o) = \text{“Europ.Union”} \wedge \text{isMember}(c, o) \wedge \text{headq}(o) = \text{capital}(c)$$

(FOL with equality)

8.3.4 Relational Calculus (“Domain Relational Calculus”)

- The signature Σ is a relational database schema $\mathbf{R} = \{R_1, \ldots, R_n\}$.
  $\Rightarrow$ everything is modeled by predicates.
- the domain consists only of *datatype literals* (strings, numbers, dates, ...).
- constant symbols are the literals themselves, with e.g. $I(3) = 3$ and I(“Berlin”) = “Berlin”.

$\Rightarrow$ a relational database state $\mathcal{S} = (I, (\text{Strings} + \text{Numbers} + \text{Dates}))$ over $\mathbf{R}$ is an interpretation of $\mathbf{R}$. For every relation name $R_i \in \mathbf{R}$, $I(R_i)$ is a finite set of tuples:
(“Germany”, “D”, 356910, 83536115, “Berlin”, “Berlin”) $\in I$(country),
(“D”, “Europe”, 100) $\in I$(encompasses).

- I (and by this, also $\mathcal{S}$) can be described as a *finite* set of ground atoms over predicate symbols (= relation names): country(“Germany”, “D”, 356910, 83536115, “Berlin”, “Berlin”), encompasses(“D”, “Europe”, 100).
- the purely value-based “modeling” without individuals/object identifiers/0-ary constant symbols requires the use of primary/foreign keys.
- semantics and model theory as in traditional FOL;
  quantifiers range over the literals – “Domain Relational Calculus”
- usage: theoretical framework for queries; mapped to *nonrecursive Datalog with negation*.

Examples

The following sets specify answers to sample queries:

- Names of all countries such that there is a city with more than 1,000,000 inhabitants in the country:

  $\{n \mid \exists cc, ca, cp, cap, capprov : \text{Country}(n, cc, ca, cp, cap, capprov) \wedge$
  $\quad \exists ctyn, ctyprov, ctypop, lat, long :$
  $\quad\quad (\text{City}(ctyn, ctyprov, cc, ctypop, lat, long) \wedge ctypop > 1{,}000{,}000)\ \}$

- Names of all countries such that all its cities have more than 1,000,000 inhabitants:

  $\{n \mid \exists cc, ca, cp, cap, capprov : \text{Country}(n, cc, ca, cp, cap, capprov) \wedge$
  $\quad \forall ctyn, ctyprov, ctypop, lat, long :$
  $\quad\quad (\text{City}(ctyn, ctyprov, cc, ctypop, lat, long) \rightarrow ctypop > 1{,}000{,}000)\ \}$

- Names of all countries such that the country is a member of the organization with name “Europ.Union”:

  $\{n \mid \exists cc, ca, cp, cap, capprov : \text{Country}(n, cc, ca, cp, cap, capprov) \wedge$
  $\quad \exists abbr, hq, hqp, hqc, est, t :$
  $\quad\quad (\text{Organization}(abbr, \text{“Europ.Union”}, hq, hqc, hqp, est) \wedge \text{isMember}(cc, abbr, t))\ \}$

8.3.5 Relational Calculus (“Tuple Relational Calculus”)

- Logical connectives and quantifiers as in FOL,
- syntax and semantics different from FOL:
  quantifiers range over tuples “Tuple Relational Calculus”
- Each relation name of $\mathbf{R}$ acts as unary predicate, holding *tuples*,
- attributes of tuples are accessed by *path expressions* variable.attrname,

Example

Names of all countries that have a city with more than 1,000,000 inhabitants:

$$\{x.\text{name} \mid \ \text{Country}(x) \wedge \exists y : (\text{City}(y) \wedge y.\text{country} = x.\text{code} \wedge y.\text{population} > 1{,}000{,}000)\ \}$$

- The Tuple Relational Calculus is a “parent” of SQL:

```
SELECT x.name
FROM country x, city y
WHERE y.country = x.code
  AND y.population > 1000000
```

```
SELECT x.name
FROM country x
WHERE EXISTS (SELECT *
              FROM city y
              WHERE y.country = x.code
              AND y.population > 1000000)
```

Examples

The following sets specify answers to sample queries:

- Names of all countries such that all its cities have more than 1,000,000 inhabitants:

  $\{c.\text{name} \mid \text{Country}(c) \wedge \forall y : ((\text{City}(y) \wedge y.\text{country} = c.\text{code}) \rightarrow y.\text{population} > 1000000)\ \}$

- Names of all countries such that the capital of the country has more than 1,000,000 inhabitants:

  $\{c.\text{name} \mid \text{Country}(c) \wedge$
  $\exists y : (\text{City}(y) \wedge c.\text{capital} = y.\text{name} \wedge c.\text{code} = y.\text{country} \wedge c.\text{capprov} = y.\text{province} \wedge$
  $y.\text{population} > 1000000)\ \}$

- Names of all countries such that the country is a member of the organization with name "Europ.Union":

  $\{c.\text{name} \mid \text{Country}(c) \wedge \exists o, m : (\text{Organization}(o) \wedge o.\text{name} = \text{``Europ.Union''} \wedge$
  $m.\text{country} = c.\text{code} \wedge m.\text{organization} = o.\text{abbrev})\ \}$

8.4 Formulas as Queries

Formulas can be seen as **queries** against a given database state:

- For a formula F with free variables $X_1, \ldots, X_n$, $n \geq 1$, write $F(X_1, \ldots, X_n)$.
- each formula $F(X_1, \ldots, X_n)$ defines – dependent on a given interpretation $\mathcal{S}$ – an **answer relation** $\mathcal{S}(F(X_1, \ldots, X_n))$.

  The **answer set** to $F(X_1, \ldots, X_n)$ wrt. $\mathcal{S}$ is the set of tuples $(a_1, \ldots, a_n)$, $a_i \in \mathcal{D}$, $1 \leq i \leq n$, such that F is true in $\mathcal{S}$ when assigning each of the variables X_i to the constant a_i, $1 \leq i \leq n$.

  Formally:

  $\mathcal{S}(F) = \{\{\beta(X_1), \ldots, \beta(X_n)\} \mid \mathcal{S} \models_\beta F$ where β is a variable assignment of $free(F)\}$.

  Each β such that $\mathcal{S} \models_\beta F$ is called an **answer**.

- for $n = 0$, the answer to F is **true** if $\mathcal{S} \models_\emptyset F$ for the empty variable assignment $\emptyset$; the answer to F is **false** if $\mathcal{S} \not\models_\emptyset F$ for the empty variable assignment $\emptyset$.

Example

Consider the query $F(X) = r(X) \wedge \exists Y : s(X, Y)$
and the database state $\mathcal{S}$:

r
1
2

s	
1	a
1	b
3	a

The answer set is given by variable assignments β (for X), such that $\mathcal{S} \models_\beta F$:

$\mathcal{S} \models_\beta F \;\Leftrightarrow\; \mathcal{S} \models_\beta r(X)$ and $\mathcal{S} \models_\beta \exists Y : s(X,Y)$

$\Leftrightarrow$ $(\beta(X) \in r)$ and for a variable assignment $\beta' = \beta_Y^d$, that assigns Y with some $d \in \mathcal{D}$ and which is identical with β up to Y, $\mathcal{S} \models_{\beta'} s(X,Y)$

$\Leftrightarrow$ " $(\beta'(X), \beta'(Y)) \in s$

$\Leftrightarrow$ " $(\beta(X), \beta'(Y)) \in s$

$\Leftrightarrow$ $(\beta(X) = 1$ or $\beta(X) = 2)$ and $((\beta(X) = 1$ and $\beta'(Y) \in \{a,b\})$ or $(\beta(X) = 3$ and $\beta'(Y) = a))$

$\Leftrightarrow$ $\beta(X) = 1$ and $\beta'(Y) \in \{a,b\}$

So, the answer set is $\{\{X/1\}\}$.

Example 8.7
Consider the MONDIAL *schema.*

- *Which cities (CName, Country) have at least 1,000,000 inhabitants?*

  $$F(CN, C) = \exists\, Pr, Pop, L_1, L_2 : (\textit{city}(CN, C, Pr, Pop, L_1, L_2) \wedge\; Pop \geq 1000000)$$

  The answer set is
  $\{\{CN/$*"Berlin"*$, C/$*"D"*$\}, \{CN/$*"Munich"*$, C/$*"D"*$\}, \{CN/$*"Hamburg"*$, C/$*"D"*$\},$
  $\{CN/$*"Paris"*$, C/$*"F"*$\}, \{CN/$*"London"*$, C/$*"GB"*$\}, \{CN/$*"Birmingham"*$, C/$*"GB"*$\}, \ldots\}$.

- *Which countries (CName) belong to Europe?*

  $$\begin{aligned} F(CName) = \exists\, & CCode, Cap, Capprov, Pop, A, ContName, ContArea, Perc : \\ & (\textit{country}(CName, CCode, Cap, Capprov, Pop, A) \wedge \\ & \textit{continent}(ContName, ContArea) \wedge \\ & ContName = \text{"Europe"} \wedge \textit{encompasses}(CCode, ContName, Perc)\,) \end{aligned}$$

□

CONJUNCTIVE QUERIES

... the above ones are *conjunctive queries*:

- use only logical conjunction of positive literals
  (i.e., no disjunction, universal quantification, negation)
- conjunctive queries play an important role in database optimization and research.
- in SQL: only a single simple SFW clause without subqueries.

Example 8.7 (Continued)

- *Again, relational division ...*
  Which organizations have at least one member on each continent

$$F(Abbrev) = \exists O, HeadqN, HeadqC, HeadqP, Est : (organization(O, Abbrev, HeadqN, HeadqC, HeadqP, Est) \wedge \forall Cont : ((\exists ContArea : continent(Cont, ContArea)) \rightarrow \exists Country, Perc, Type : (encompasses(Country, Cont, Perc) \wedge isMember(Country, Abbrev, Type))))$$

- *Negation*
  All pairs (country,organization) such that the country is a member in the organization, and all its neighbors are not.

$$F(CCode, Org) = \exists CName, Cap, Capprov, Pop, Area, Type : (\textsf{country}(CName, CCode, Cap, Capprov, Pop, Area) \wedge \textsf{isMember}(CCode, Org, Type) \wedge \forall CCode' : (\exists Length : \textsf{sym\_borders}(CCode, CCode', Length) \rightarrow \neg\exists Type' : \textsf{isMember}(CCode', Org, Type')))$$

□

8.5 Comparison of the Algebra and the Calculus

Algebra:

- The semantics is given by evaluating an algebraic expression (i.e., an operator tree) "**algebraic** Semantics" (which is also some form of a *declarative* semantics).
- The algebraic semantics also induces a naive, but already polynomial bottom-up evaluation algorithm based on the algebra tree.

Calculus:

- The semantics (= answer) of a query in the relational calculus is defined via the truth value of a logical formula wrt. an interpretation
  "**logical** Semantics" (which is some form of a *declarative* semantics)
- The logical semantics can be evaluated by a (FOL) Reasoner
  FOL is undecidable.

$\Rightarrow$ translate "FOL" formulas over a simple database into the algebra ...

Example: Expressing Algebra Operations in the Calculus

Consider relation schemata $R[A, B]$, $S[B, C]$, and $T[A]$.

(Note: $[A, B]$ is the *format* of the relationships wrt. the relational model with named columns; X and Y are variables used in the *positional* relational calculus)

Projection $\pi[A](R)$: $F(X) = \exists Y\, R(X, Y)$

Selection $\sigma[A = B](R)$: $F(X, Y) = R(X, Y) \wedge X = Y$

Join $R \bowtie S$: $F(X, Y, Z) = R(X, Y) \wedge S(Y, Z)$

Union $R \cup (T \times \{b\})$: $F(X, Y) = R(X, Y) \vee (T(X) \wedge Y = b)$

Difference $R - (T \times \{B : b\})$: $F(X, Y) = R(X, Y) \wedge \neg(T(X) \wedge Y = b)$

Division $R \div T$: $F(Y) = (\exists X : R(X, Y)) \wedge \forall X : (T(X) \rightarrow R(X, Y))$ or
$F(Y) = (\exists X : R(X, Y)) \wedge \neg\exists X : (T(X) \wedge \neg R(X, Y))$

SAFETY AND DOMAIN-INDEPENDENCE

- For some formulas, the actual answer set does not depend on the actual database state, but on the domain of the interpretation.
- If the domain is infinite, the answer relations to some expressions of the calculus can be infinite!

Example 8.8
Recall $\mathcal{S} = (I, \mathcal{D})$*, usually* $\mathcal{D} = Strings + Numbers + Dates$ *(cf. Slide 443).*

- *Consider* $F(X) = \neg R(X)$ *("all* a *such that* $R(a)$ *does not hold")*
  where $I(R) = \{(1)\}$.
  For every domain $\mathcal{D}$*, the answers to* $\mathcal{S}(F)$ *are all elements of the domain. For an infinite domain, e.g.,* $\mathcal{D} = \mathbb{N}$*, the set of answers is infinite.*
- *Consider* $F(X, Z) = \exists Y(R(X, Y) \vee S(Y, Z))$,
  where $I(R) = \{(1, 2)\}$*, arbitrary* $\mathcal{S}(S)$ *(even empty).*
  How to determine Z*? – return* $\{X/1, Y/d\}$ *for every element* d *of the domain?*
- *Consider* $F(X) = \forall Y : R(X, Y)$
  where $I(R) = \{(1, 1), (1, 2)\}$*. For* $\mathcal{D} = \{1, 2\}$ *the answer set is* $\{\{X/1\}\}$*, for any larger domain, the answer set is empty.* □

Example 8.9
Consider a FOL interpretation $\mathcal{S} = (I, \mathcal{D})$ *of persons:*

Signature $\Sigma = \{\text{married}/2\}$*, married(*$X, Y$*):* X *is married with* Y.

$F(X) = \neg married(john, X) \wedge \neg(X = john)$.

What is the answer?

- *Consider* $\mathcal{D} = \{john, mary\}$, $I(married) = \{(john, mary), (mary, john)\}$.
  $\mathcal{S}(F) = \emptyset$.
  - *there is no person (except John) who is not married with John*
  - *all persons are married with John???* □
- *Consider* $\mathcal{D} = \{john, mary, sue\}$, $I(married) = \{(john, mary), (mary, john)\}$.
  $\mathcal{S}(F) = \{\{X/sue\}\}$.
  The answer depends not only on the database, but on the domain (that is a purely logical *notion)*
  Obviously, it is meant "All persons in the database who are not married with $john$*".*

Active Domain

Requirement: the answer to a query depends only on

- constants given in the query
- constants in the database

Definition 8.1
Given a formula F of the relational calculus and a database state $\mathcal{S} = (I, \mathcal{D})$, $ADOM(F)$ contains

- *all constants in F,*
- *and all constants in $I(R)$ where R is a relation name that occurs in F.*

*$ADOM(F \cup I)$ is called the **active domain** domain of F wrt. the interpretation I.* □

$ADOM(F \cup I)$ is finite.

Domain-Independence

Formulas in the relational calculus are required to be **domain-independent**:

Definition 8.2
*A formula $F(X_1, \ldots, X_n)$ is **domain-independent** if for all interpretations I of the predicates and constants, and for all $\mathcal{D} \supseteq ADOM := ADOM(F \cup I)$,*

$$\begin{aligned}(I, ADOM)(F) &= \\ &= \{(\beta(X_1), \ldots, \beta(X_n)) \mid (I, ADOM) \models_\beta F,\ \beta(X_i) \in ADOM \text{ for all } 1 \leq i \leq n\} \\ &= \{(\beta(X_1), \ldots, \beta(X_n)) \mid (I, \mathcal{D}) \models_\beta F,\ \beta(X_i) \in \mathcal{D} \text{ for all } 1 \leq i \leq n\} = (I, \mathcal{D})(F).\end{aligned}$$ □

It is undecidable whether a formula F is domain-independent!
(follows from Rice's Theorem).

Instead, **(syntactical) safety** is required for queries:

- stronger condition
- can be tested algorithmically

Idea: every formula guarantees that variables can only be bound to values from the database or that occur in the formula.

Safety: SRNF

Definition 8.3
*A formula F is in **SRNF (Safe Range Normal Form)** [Abiteboul, Hull, Vianu: Foundations of Databases] if and only if it satisfies the following conditions:*

- *variable renaming: no variable symbol is bound twice with different scopes by different quantifiers; no variable symbol occurs both free and bound.*
- *remove universal quantifiers by replacing $\forall X : G$ by $\neg\exists X : \neg G$,*
- *remove implication by replacing $F \rightarrow G$ by $\neg F \vee G$,*
- *push negations down through $\wedge$ and $\vee$.*
  Negated formulas are then either of the form $\neg\exists F$ or $\neg$atom (push negations down through $\wedge$ and $\vee$),
- *flatten $\wedge$, $\vee$ and $\exists$ (i.e., replace $F \wedge (G \wedge H)$ by $F \wedge G \wedge H$, and $\exists X : \exists Y : F$ by $\exists X, Y : F$).* □

... then, check, if it is safe range.

Safety Check for SRNF formulas

Definition 8.4

1. *For a formula F in SRNF, $rr(F)$ is defined (and computable) via structural induction:*

$$
\begin{array}{lll}
(1) & F = R(t_1,\ldots,t_n) & \Rightarrow \quad rr(F) \text{ is the set of variables occurring in } t_1,\ldots,t_n \\
(2) & F = x = a \text{ or } a = b & \Rightarrow \quad rr(F) = \{x\} \\
(3) & F = F_1 \wedge F_2 & \Rightarrow \quad rr(F) = rr(F_1) \cup rr(F_2) \\
(4) & F = F_1 \wedge X = Y & \Rightarrow \left\{ \begin{array}{ll} rr(F) = rr(F_1) \cup \{x,y\} & \text{if } rr(F_1) \cap \{x,y\} \neq \emptyset \\ rr(F) = rr(F_1) & \text{if } rr(F_1) \cap \{x,y\} = \emptyset \end{array} \right. \\
(5) & F = F_1 \vee F_2 & \Rightarrow \quad rr(F) = rr(F_1) \cap rr(F_2) \\
(6) & F = \neg F_1 & \Rightarrow \quad rr(F) = \emptyset \\
(7) & F = \exists \bar{X} : F_1 & \Rightarrow \left\{ \begin{array}{ll} rr(F) = rr(F_1) - \bar{X} & \text{if } \bar{X} \subseteq rr(F_1) \\ \text{return } \perp & \text{if } \bar{X} \not\subseteq rr(F_1) \end{array} \right.
\end{array}
$$

2. *if $free(F) = rr(F)$ and no subformula returned $\perp$, F is safe range.* □

Note:

∗ The $\forall$-quantifier is not allowed in any formula in SRNF (i.e. replace $\forall X F$ by $\neg\exists X \neg F$).

∗ The definition does not contain any explicit syntactical hints how to write such a formula.

Example 8.10
and Exercise

Consider the formulas

1. $F(X,Y,Z) = p(X,Y) \wedge (q(Y) \vee r(Z))$,
2. $F(X,Y) = p(X,Y) \wedge (q(Y) \vee r(X))$,
3. $F(X) = p(X) \wedge \exists Y : (q(Y) \wedge \neg r(X,Y))$,
4. $F(X) = p(X) \wedge \neg\exists Y : (q(Y) \wedge \neg r(X,Y))$ *– the relational division pattern,*
5. $F(X,Y) = p(X,Y) \wedge \neg\exists Z : r(Y,Z)$,

Are they safe-range?

Give $rr(G)$ for each of their subformulas.

Translate the formulas into SQL and into the relational algebra. □

Safe Range and Domain Independence

Theorem 8.1
If a formula F is in SRNF and is safe-range, then it is domain-independent. □

... one can prove this by induction, but this will also follow in a more useful way.

How to evaluate calculus queries?

- the underlying framework is FOL, undecidable, no complete reasoners exist. incomplete reasoners would do it, but they have high complexity and bad performance.

  (this issue will be the same when continuing with Datalog “knowledge” bases.)

- the goal is that the relational calculus is equivalent with the relational algebra; i.e. much weaker than full FOL, but polynomial.

  (Datalog variants are also weaker than FOL, but some of them harder than polynomial)

$\Rightarrow$ get a translation to the relational algebra.

(this problem will be solved by algebra+fixpoint and Logic-Programming-based implementations)

Comments on SRNF

- underlying idea: the formula can be evaluated from the database relations, never using the (purely logical concept of) "domain".
- subformulas of a conjunction $F(\ldots, X, \ldots) \wedge G(X, Y)$ whose evaluation would not be domain-independent alone (i.e., $rr(G) \subsetneq free(G)$) are "cured" by other parts of the conjunction (cf. solution to Example 8.10);
  - cf. *correlated subqueries* (SQL) or *correlated joins* in SQL/OQL/XQuery;
  - cf. index-based join in SQL: compute $E_1 \bowtie E_2$ by iterating over results of E_1 and accessing matching tuples in E_2 via index.
  - also called "sideways information passing strategy".
- ... but the relational algebra does not have correlated subqueries (no subqueries in selection conditions at all!) and no correlated joins.
  The algebra's theory is only bottom-up (cf. the relational algebra translations from Example 8.10 which provide some insights into the next definition ...).

Self-Containedness of Subformulas

Definition 8.5
*A formula F that is in SRNF and which is safe-range is in **RANF (Relational Algebra Normal Form)** if:*

1. *(from SRNF) F does not contain $\forall$ quantifiers (replace $\forall X G$ by $\neg \exists X \neg G$),*
2. *(from SRNF) negated formulas are either of the form $\neg \exists F$ or $\neg$atom (push negations down through $\wedge$ and $\vee$),*
3. *and if each subformula G of F is self-contained, where a subformula G is* self-contained *if*
   (0) *if G is an atom, or if $G = G_1 \wedge \ldots \wedge G_k$*
   (in this case, no additional explicit condition is stated, but requirements are made whenever such a G is used as a subformula in (i)-(iii)),
   (i) *if $G = H_1 \vee \ldots \vee H_k$ and for all i, $rr(H_i) = free(G)$*
   (which implies that $free(H_i) = free(G) = rr(H_i)$ for all i),
   (ii) *if $G = \exists \bar{X} : H$ and $rr(H) = free(H)$*
   (which due to SRNF(7) is equivalent to $rr(G) = free(G)$),
   (iii) *if $G = \neg H$ and $rr(H) = free(H)$.* □

(note: typo in [Abiteboul, Hull, Vianu: Foundations of Databases] in (ii) and (iii)!)

Self-Containedness of Subformulas

- Recall "correlated joins/subqueries" via $F(\ldots, X, \ldots) \wedge G(X, Y)$ that refer to an "outer" query that provides bindings for –in this case– X.
- self-containedness requires that the evaluation of G does actually *not* depend on propagation of bindings from "outside".
- For that,

  $$rr(G) = free(G) \qquad (*)$$

  would be a sufficient criterion
  (i.e., each subformula G is in SRNF itself).
  This criterion is enforceable, except for negated subformulas.

Self-Containedness

Consider again

$$rr(F) = free(F) \qquad (*)$$

- The definition of "self-contained" does not state any explicit condition on conjunctions $G = G_1 \wedge \ldots \wedge G_k$.
  For them, the property $(*)$ follows from the other requirements:
  if G is in a disjunction (from (3a)), in a negated subformula (from (3b)), and in an existence formula (from (3c) and SRNF (1.7)), and if $G = F$, then from SRNF (2).
- Self-containedness implies and requires that $(*)$ holds for all formulas that are not of the form $F = \neg G$.
- For negations $F = \neg G$, $rr(F) = \emptyset$, and $(*)$ is implied and required only for their body: $rr(G) = free(G)$.
  Negations as a whole and isolated cannot satisfy $(*)$ – they depend on propagation from outside.
- idea: hardcode the subformula that generates the relevant bindings into the subformula.

From SRNF to RANF

Application of the following *rewriting rules (recursively – top-down)* translates SRNF formulas to RANF.
[Abiteboul, Hull, Vianu: Foundations of Databases]

1. Assume that $(*)$ holds for the whole formula F: $free(F) = rr(F)$.
2. This is the case for each SRNF formula, so the starting point is well-defined.
3. input to each rewriting rule is a conjunction F of the form $F = F_1 \wedge \ldots \wedge F_n$ s.t. $free(F) = rr(F)$ where one or more of the F_i are not self-contained (let m the number of such F_i).

⇒ Make them self-contained!

4. each application of a rewriting rule will handle one such conjunct.
5. after m applications, F has been transformed into a conjunction $F' = F'_1 \wedge \ldots \wedge F'_k$, $k \leq n$, where all F'_i are self-contained.
6. then, the assumption in $(*)$ is valid for them (for negations: for their immediate subformula), and the formulas on lower levels can be rewritten.
7. as seen above, rewriting rules must only care for conjunctions (where the bindings propagation takes place).

From SRNF to RANF -2-

- W.l.o.g. assume that the conjunct to be treated is the rightmost one.
- Push-into-or: $F = F_1 \wedge \ldots \wedge F_n \wedge G$ where $G = G_1, \ldots, G_m$ is a disjunction, G is not self-contained, i.e., $rr(G) \subsetneq free(G)$ (which actually is the case if for some disjunct $rr(G_i) \subsetneq free(G)$).
  (w.l.o.g., G is the last conjunct)

  Known: $rr(F) = free(F)$; the missing variable(s) must be in $rr(F_1, \ldots, F_n)$.

  Choose any subset $F_{i_1}, \ldots, F_{i_k}$, $k \leq n$ such that
  $G' = (F_{i_1} \wedge \ldots \wedge F_{i_k} \wedge G_1) \vee \ldots \vee (F_{i_1} \wedge \ldots \wedge F_{i_k} \wedge G_m)$ satisfies $rr(G') = free(G')$.
  - choosing all F_i is correct, but usually "inefficient".
  - note: $rr(G') \supseteq rr(G)$ ("=" in the best case), and for each disjunct G'_i in G', $rr(G'_i) = free(G'_i) = free(G')$ (before, $free(G_i) \neq free(G_j)$ was possible)

  Let $j_1, \ldots, j_{n-k}$ the indexes from $\{1, \ldots, n\} \setminus \{i_1, \ldots, i_k\}$; i.e., the non-chosen ones.

  Replace F by $F' = SRNF(F_{j_1} \wedge \ldots \wedge F_{j_{n-k}} \wedge G')$ and go on recursively.
  ($SRNF(\_)$ for renaming vars, flattening, etc.)
- ... two more rewriting rules see next slide.

From SRNF to RANF -3-

Example 8.11

- *Recall Example 8.10 (2) and its algebra translation.*
- *Recall Example 8.10 (3) for guessing the next rule.*
- *... recall Example 8.10 (4) for guessing the third rule.* □

... other rewriting rules in the same style:

- Push-into-exists: $F = F_1 \wedge \ldots \wedge F_n \wedge \exists \bar{X} : G$ where $rr(F) = free(F)$; $rr(G) \subsetneq free(G)$.
  Choose again F_is such that $G' = F_{i_1} \wedge \ldots \wedge F_{i_k} \wedge G$ as above. Replace F by $F' = SRNF(F_{j_1} \wedge \ldots \wedge F_{j_{n-k}} \wedge \exists x : G')$ and go on recursively.
- Push-into-not-exists: $F = F_1 \wedge \ldots \wedge F_n \wedge \neg\exists \bar{X} : G$ where $rr(F) = free(F)$; $rr(G) \subsetneq free(G)$.
  Do the same as above for $G' = F_{i_1} \wedge \ldots \wedge F_{i_k} \wedge G$, replace F by $F' = SRNF(F_1 \wedge \ldots \wedge F_n \wedge \neg\exists x : G')$ (keeping all F_i also outside!) and go on recursively.
- what about "Push-into-negation"?
  Recall from Definition 8.5(2) that $\neg$ occurs only as $\neg\exists F$ (see above) or $\neg$*atom* (always self-contained).

Exercise

Consider the formula

$$F(X, Y) = \exists V : (r(V, X) \wedge \neg s(X, Y, V)) \wedge \exists W : (r(W, Y) \wedge \neg s(Y, X, W))$$

- Give $rr(F)$ for all its subformulas,
- is it in SRNF?
- if yes, transform it to RANF.

This is an example, where no conjunct of the original formula is self-contained.

Exercise

Give an algorithm that transforms RANF formulas to the Relational Algebra.

PREVIEW

RANF is not only necessary for the translation into the Relational Algebra, but also for translation into (Nonrecursive Stratified) Datalog; cf. next section.

An Alternative Formulation

[Ullman, J. D., Principles of Database and Knowledge-Base Systems, Vol. 1]

Definition 8.6
A formula F is safe *(SAFE) if:*

1. *F does not contain $\forall$ quantifiers (replace $\forall X G$ by $\neg\exists X \neg G$),*
2. *if $F_1 \vee F_2$ is a subformula of F, then F_1 and F_2 must have the same free variables,*
3. *for all maximal conjunctive subformulas $F_1 \wedge \ldots \wedge F_m, m \geq 1$ of F:*
   *All free variables must be **limited**, where limited is defined as follows:*
   - *if F_i is neither a comparison, nor a negated formula, any free variable in F_i is limited,*
   - *if F_i is of the form $X = a$ or $a = X$ with a a constant, then X is limited,*
   - *if F_i is of the form $X = Y$ or $Y = X$ and Y is limited, then X is also limited.*

*(a subformula G of a formula F is a **maximal conjunctive subformula**, if there is no conjunctive subformula H of F such that G is a subformula of H).* □

Theorem 8.2
Safe formulas are domain-independent. □

Safety (Cont'd)

Example 8.12

- *$p(X, Y) \vee X = Y$ is not safe: $X = Y$ is a maximal conjunctive subformula where none of the variables is limited (it is also not domain-independent).*
- *$p(X, Y) \wedge X = Z$ is safe: $p(X, Y)$ limits X and Y, then $X = Z$ also limits Z.*
- *$p(X, Y) \wedge (q(X) \vee r(Y))$ is not safe, but the equivalent formula $(p(X, Y) \wedge q(X)) \vee (p(X, Y) \wedge q(Y))$ is safe.*
- *$p(X, Y, Z) \wedge \neg(q(X, Y) \vee r(Y, Z))$ is not safe, but the logically equivalent formula $p(X, Y, Z) \wedge \neg q(X, Y) \wedge \neg r(Y, Z)$ is safe.*
- *$F(X) = p(X) \wedge \neg\exists Y : (q(Y) \wedge \neg s(X, Y))$ is not safe*
  because $F'(X) = \exists Y : (q(Y) \wedge \neg r(X, Y)$ is a maximal conjunctive subformula, but it does not limit X);
  the logically equivalent, but less intuitive formula
  $F(X) = p(X) \wedge \neg\exists Y : (p(X) \wedge q(Y) \wedge \neg r(X, Y))$ is safe.
  (again the relational division pattern) □

Notes

- condition RANF(3b) is not required by SAFE. Nevertheless, since in $\neg G$, G is a maximal conjunctive formula (maybe with $m = 1$), SAFE(3) applies to it and implies RANF(3b).
- condition RANF(3a) is stronger than SAFE(2), but implied by SAFE(3) since in $G_1 \vee G_2$ each disjunct is a maximal conjunctive subformula which implies that all its variables must be limited.
- SAFE(3) explicitly requires for each negated formula $\neg F(\bar{X})$ that it *must* occur in some conjunction $G = (\ldots \wedge F(\bar{X}) \wedge \ldots)$ with positive formulas that limit the Xs:

  Otherwise, if any non-conjunctive formula G contains $\neg F(\bar{X})$ as an immediate subformula, $\neg F(\bar{X})$ would be a maximal conjunctive formula in F where $\bar{X}$ are not limited.
- In contrast, RANF does not state an explicit condition on the occurrence of negated subformulas. Implicitly, the same condition follows from the fact that $rr(\neg F(\bar{X})) = \emptyset$ (SNRF(6)), and the remark on the bottom of Slide 463: $\bar{X} \subset free(G)$, so there must be a conjunct G_i "neighboring" the negated formula to such that $rr(G_i) \subseteq \bar{X}$.

Safety: universal quantification

Consider again from Example 8.8:

$$F(X) = \forall Y : R(X, Y)$$

- This formula is not allowed to be considered since $\forall$ must be rewritten:

  $$F_2(X) = \neg \exists Y : \neg R(X, Y)$$

  is not safe since $\neg R(X, Y)$ is a maximal conjunctive subformula.
- Start again with F: the problem in Example 8.8 was that it is not known which Y have to be considered (the whole domain?)
- restrict to Y that satisfy some condition (e.g., all country codes).

  An upper bound is to consider all elements of the active domain, let (assume relations $R_{/2}, S_{/1}, \ldots$)

  $$ADOM(Z) = (\exists Y : R(Z, Y) \vee \exists X : R(X, Z) \vee S(Z) \vee \ldots) \quad :$$

  $$F_3(X) = \forall Y : (ADOM(Y) \rightarrow R(X, Y))$$

  (continue next slide)

Safety: universal quantification (cont'd)

- ... and rewrite $\forall$:

$$F_4(X) = \neg\exists Y : \neg(ADOM(Y) \rightarrow R(X,Y))$$

push negation down and rewrite $F \rightarrow G$ as $\neg F \vee G$:

$$F_5(X) = \neg\exists Y : (ADOM(Y) \wedge \neg R(X,Y))$$

- $ADOM(Y) \wedge \neg R(X,Y)$ is still not safe. X must be bound; use again $ADOM$:

$$F_6(X) = \neg\exists Y : (ADOM(X) \wedge ADOM(Y) \wedge \neg R(X,Y))$$

- is safe, but unintuitive. Pulling out X yields ...

$$F_7(X) = ADOM(X) \wedge \neg\exists Y : (ADOM(Y) \wedge \neg R(X,Y))$$

... which is the relational division pattern!

Aside: Another Alternative Formulation

[Allen Van Gelder and Rodney W. Topor. Safety and translation of relational calculus queries. ACM Transactions on Database Systems (TODS), 16(2):235-278, 1991.]

- based on two syntactical, inductively defined properties $con(X)$ ("constrained") and $gen(X)$ ("generated"),
- a formula is "evaluable" if
  - for every free variable in $Q(X) = F(X)$, $gen(X,F)$ holds,
  - for every subformula $\exists X : F$, $con(X,F)$ holds,
  - for every subformula $\forall X : F$, $con(X,\neg F)$ holds,
- claimed that this definition is the largest class of domain-independent formulas that can be characterized by syntactical restrictions;
- proven that for queries without repetitions of predicate symbols the definition coincides with domain-independence.
  - The (simple) formula $Q(x) = p(x) \wedge \forall y : \neg q(x,y)$ is in SRNF, and evaluable, but the equivalent PLNF (prenex literal normal form) $Q'(x) = \forall y : (p(x) \wedge \neg q(x,y))$ is not in SRNF (equivalent to $\neg\exists y : \neg(p(x) \vee \neg q(x,y))$, where $y \notin rr(\neg(p(x) \vee \neg q(x,y)))$), but still "evaluable". Later, for Datalog always the (SRNF-compatible) variant where the scope of the universal quantifier is *only a single, negative literal* is relevant.

Summary: A Higher-Level View on Domain Independence/Safety vs RANF

Domain Independence

- Domain independence is absolutely necessary for a query to have a well-defined meaning (humans evaluate such queries when the context gives the domain, e.g. “who is not registered for the exam?” [domain: the participants of the lecture]).
- Domain independence is undecidable.

Safety

- safety is defined purely syntactically,
- safety can be tested effectively,
- safety implies domain-independence.

Metalevel: Reconsider FOL vs Herbrand Style

- FOL:
  Σ: predicate symbols $p, q, r, \ldots$, function symbols $f, g, \ldots$, constant symbols $a, b, c, \ldots$,
  $\mathcal{I} = (I, \mathcal{D})$; $I(p) \subseteq \mathcal{D}^n$ for n-ary p.
  $\mathcal{I} \models p(a, b, c) \quad \Leftrightarrow \quad (I(a), I(b), I(c)) \in I(p)$.
  - The abstraction level of I is needed in FOL model theory, especially if function symbols are used.
  - the notion of the domain $\mathcal{D}$ is needed for the semantics of the universal quantifier and proving validity of a formula.
- Herbrand/DB with *safe formulas*:
  Σ: predicate symbols p, q, r, ...,
  constants $a, b, c, \ldots$ + datatype values 1, 2, 3, ..., “D”,“CH”, ...
  Database state $\mathcal{S}$ over the relations p, q, r,...;
  with values from the constants and datatype values,
  $\mathcal{S} \models p(a, b, c) \quad \Leftrightarrow \quad (a, b, c) \in p$.
  $\Rightarrow$ neither need the notions of I nor $\mathcal{D}$ – everything is immediately contained in $\mathcal{S}$.

Domain Independence is inherent in the relational algebra and in SQL

Algebra

- Basic algebra expressions/leaves of the algebra tree are always relations (database relations or constants),
- (non-atomic) “negation” in the relation algebra only via “minus”,
- proof by structural induction: the left subtree of “minus” is always domain-independent ⇒ the whole expression is domain-independent.

SQL

- FROM clause always refers (positively) to relations or to SQL subqueries,
- (non-atomic) negation only in subqueries in the WHERE clause, sideways-information-passing.
- whole SQL expression is domain-independent.

A Higher-Level View on Domain Independence/Safety vs RANF

- Logics: domain-independent formulas can be evaluated;
- Relational algebra: requires RANF for strict bottom-up evaluation;
- SQL:
  - relaxed criterion (cf. Example 8.10) for (negated) existential quantification;
  - not relaxed for disjunction/union;
  - ⇒ internal compiler from SQL into an internal (relational) algebra that supports *sideways information passing*;
- SPARQL (query language for RDF): also relaxed for disjunction/union.
- Datalog will require RANF since every subexpression is represented by an own “local” rule;
  “global” semantics and internal compilation by Logic Programming-based (Prolog) top-down proof tree strategy supports *sideways information passing*.

8.6 Equivalence of Algebra and (safe) Calculus

As for the algebra, the attributes of each relation are assumed to be ordered.

Theorem 8.3
For each expression Q of the relational algebra there is an equivalent safe formula F of the relational calculus, and vice versa; i.e., for every state $\mathcal{S}$, Q and F define the same answer relation. □

Proof Summary

- give mappings (A) "Algebra $\to$ Calculus" and (B) "Calculus $\to$ Algebra"
- (A) gives insights how to express a textual (or SQL) query by Datalog Rules,
- (B) gives insight how to write SQL statements for a given textual (or logical) query (and how one could implement a Calculus evaluation engine via SQL).

Proof: (A) Algebra to Calculus

Let Q an expression of the relational algebra. The proof is done by induction over the structure of Q (as an operator tree).

All generated formulas are safe.
As an invariant, the variable names $A, B, C, \ldots$ correspond always to the column names A,B,C,... of the format of the respective algebra expression.

Induction base: Q does not contain operators.

- if $Q = R$ where R is a relation symbol of arity $n \geq 1$ with format $\mathrm{A}_1, \ldots, \mathrm{A}_n$:

$$F(A_1, \ldots, A_n) = R(A_1, \ldots, A_n)$$

R	
A_1	A_2
a	1
b	2

answer to $R(A_1, A_2)$:

A_1	A_2
a	1
b	2

- otherwise, $Q = \{\text{A:c}\}$ where c is a constant.
  Then, $F(A) = (A = c)$.

A:c
A
c

Answer to $A = c$:

A
c

Induction step:

- Case $Q = Q_1 \cup Q_2$. Thus, $\Sigma_{Q_1} = \Sigma_{Q_2} = \mathsf{A}_1, \ldots, \mathsf{A}_n$.

$$F(A_1, \ldots, A_n) = F_1(A_1, \ldots, A_n) \vee F_2(A_1, \ldots, A_n)$$

Example:

Q_1	
A_1	A_2
a	b
c	d

$F_1($

A_1	A_2
a	b
c	d

$)$

Q_2	
A_1	A_2
1	2
c	d

$F_2($

A_1	A_2
1	2
c	d

$)$

$F($

A_1	A_2
a	b
c	d
1	2

$)$

- Case $Q = Q_1 - Q_2$. Analogously; replace $\ldots \vee \ldots$ by $(\ldots) \wedge \neg (\ldots)$.
- Case $Q = \pi[\bar{Y}](Q_1)$ with $\bar{Y} = \{A_{i_1}, \ldots, A_{i_k}\} \subseteq \Sigma_{Q_1}$, $k \geq 1$.
  Let $\{j_1, \ldots, j_{n-k}\} = \{1, \ldots, n\} \setminus \{i_1, \ldots, i_k\}$ (the indices not in $\bar{Y}$).

$$F(A_{j_1}, \ldots, A_{j_{n-k}}) = \exists A_{i_1}, \ldots, A_{i_k} : F_1(A_1, \ldots, A_n) \ .$$

Example:

Q_1	
A_1	A_2
a	b
c	d

$F_1($

A_1	A_2
a	b
c	d

$)$

Let $\bar{Y} = \{\mathsf{A}_2\}$: $F(A_2) = \exists A_1 : F_1(A_1, A_2)$

$F($

A_2
b
d

$)$

- Case $Q = \sigma[\alpha](Q_1)$ where α is a condition over $\Sigma_{Q_1} = \{\mathsf{A}_1, \ldots, \mathsf{A}_n\}$.

  $F(A_1, \ldots, A_n) = F_1(A_1, \ldots, A_n) \wedge \alpha'$, where α' is obtained by replacing each column name A_i by the variable A_i in σ.

  Example:

Q_1	
A_1	A_2
1	2
3	4

$F_1($

A_1	A_2
1	2
3	4

$)$

  Let σ = "$\mathsf{A}_1 = 3$": $F(A_1, A_2) = F_1(A_1, A_2) \wedge A_1 = 3$

$F($

A_1	A_2
3	4

$)$

- Case $Q = \rho[\mathsf{A}_1 \to \mathsf{B}_1, \ldots, \mathsf{A}_m \to \mathsf{B}_m](Q_1)$, $\Sigma_{Q_1} = \{\mathsf{A}_1, \ldots, \mathsf{A}_n\}$, $n \geq m$.

  $F(B_1, \ldots, B_m, A_{m+1}, \ldots, A_n) = \exists A_1, \ldots, A_m : (F_1(A_1, \ldots, A_n) \wedge B_1 = A_1 \ldots \wedge B_m = A_m)$

  Example:

Q_1	
A_1	A_2
1	2
3	4

$F_1($

A_1	A_2
1	2
3	4

$)$

  Consider $\rho[\mathsf{A}_1 \to \mathsf{B}_1](Q_1)$: $F(B_1, A_2) = \exists A_1 : (F_1(A_1, A_2) \wedge A_1 = B_1)$

$F($

B_1	A_2
1	2
3	4

$)$

- Case $Q = Q_1 \bowtie Q_2$ and $\Sigma_{Q_1} = \{\mathsf{A}_1, \ldots, \mathsf{A}_n\}$, $\Sigma_{Q_2} = \{\mathsf{A}_1, \ldots, \mathsf{A}_k, \mathsf{B}_{k+1}, \ldots, \mathsf{B}_m,\}$, $n, m \geq 1$ and $0 \leq k \leq n, m$.

$$F(A_1, \ldots, A_n, B_{k+1}, \ldots, B_m) = F_1(A_1, \ldots, A_n) \wedge F_2(A_1, \ldots, A_k, B_{k+1}, \ldots, B_k)\ .$$

Example:

Q_1	
A_1	A_2
1	2
3	4

Q_2	
A_1	B_2
5	6
1	7

$F_1($

A_1	A_2
1	2
3	4

$)$

$F_2($

A_1	B_2
5	6
1	7

$)$

$F(A_1, A_2, B_2) = F_1(A_1, A_2) \wedge F_2(A_1, B_2)$

$F($

A_1	A_2	B_2
1	2	7

$)$

- Note that in all cases, the resulting formulas F are domain-independent, in SRNF, RANF, and SAFE.
(which came up automatically, because it is built-in in the structure induced by the algebra expressions)

(B) Calculus to Algebra

Consider a relational schema $\Sigma = \{R_1, \ldots, R_n\}$ and a SAFE formula $F(X_1, \ldots, X_n)$, $n \geq 1$ of the relational calculus.

First, an algebra expression $ADOM$ that computes the active domain $ADOM(\mathcal{S})$ of the database state is derived:

For every R_i with arity k_i,

$$ADOM(R_i) = \pi[\$1](R_i) \cup \ldots \cup \pi[\$k_i](R_i).$$

(where $\pi[\$i]$ denotes the projection to the i-th column).
Let

$$ADOM = ADOM(R_1) \cup \ldots \cup ADOM(R_n) \cup \{a_1, \ldots, a_m\},$$

where $a_1, \ldots, a_m$ are the constants occurring in F.

- For a given database state $\mathcal{S}$ over Σ, $ADOM(\mathcal{S})$ is a unary relation that contains the whole active domain of the database, i.e., all values occurring in any tuple in any position.

An equivalent algebra expression Q is now constructed by induction over the number of maximal conjunctive subformulas of F.

Induction base: F is a conjunction of positive literals. Thus, $F = G_1 \wedge \ldots \wedge G_l$, $l \geq 1$.

(1) Case $l = 1$. F is a single positive safe literal.
Then, either is of the form $F = R_i(a_1, \ldots, a_{i_k})$, where each a_j is a variable or a constant, or F is a comparison of one of the forms $F = (X = c)$ or $F = (c = X)$, where X is a variable and c is a constant (note that all other comparisons would not be safe).

- Case $F = R(a_1, \ldots, a_{i_k})$: contains some (free, maybe duplicate) variables, and some constants that state a condition on the matching tuples.

  ⇒ encode the condition into a selection, and do a projection to the columns where variables occur – one column for each variable and name the columns with the variables:

  e.g. $F(X,Y) = R(a, X, b, Y, a, X)$. Then, let

  $$Q(F) = \rho[\$2 \to X, \$4 \to Y](\pi[\$2, \$4](\sigma[\Theta_1 \wedge \Theta_2](R))) \ ,$$

  where $\Theta_1 = (\$1 = a \wedge \$3 = b \wedge \$5 = a)$ and $\Theta_2 = (\$2 = \$6)$.

- Case $F = (X = c)$ or $F = (c = X)$. Let $Q(F) = \{X : c\}$

X
c

(2) Case $l > 1$ (cf. example below) Then, w.l.o.g.

$$F = G_1 \wedge \ldots \wedge G_m \wedge G_{m+1} \wedge \ldots \wedge G_l$$

s.t. $1 < m \leq l$, where all G_i, $1 \leq i \leq m$ as in (1) and all G_j, $m+1 \leq j \leq l$ are other comparisons (i.e., unsafe literals like $X = Y$, $X < 3$).

For every G_i, $1 \leq i \leq m$ take an algebra expression $Q(G_i)$ as done in (1). The format $\Sigma_{Q(G_i)}$ is the set of free variables in G_i. Let

$$Q' = \bowtie_{i=1}^{m} Q(G_i).$$

With Θ the conjunction of the additional conditions $G_{m+1}, \ldots, G_l$,

$$Q(F) = \sigma[\Theta](Q') \ .$$

Example 8.13
Consider $F = R(a, X, b, Y, a, X) \wedge S(X, Z, a) \wedge X = Y \wedge Z < 3$
as $F = G_1 \wedge G_2 \wedge G_3 \wedge G_4$*:*

$$Q(G_1) = \rho[\$2 \to X, \$4 \to Y](\pi[\$2, \$4](\sigma[\$1 = a \wedge \$3 = b \wedge \$5 = a \wedge \$2 = \$6](R)))$$
$$Q(G_2) = \rho[\$1 \to X, \$2 \to Z](\pi[\$1, \$2](\sigma[\$3 = a](S)))$$
$$Q(F) = \sigma[X = Y \wedge Z < 3](Q(G_1) \bowtie Q(G_2))$$

□

Structural Induction Step: For formulas $G, G_1, \ldots, G_l, H$ the equivalent algebra expressions are $Q(G), Q(G_1), \ldots, Q(G_l), Q(H), \ldots$.

(3) $F = G \vee H$:

$$Q(F) = Q(G) \cup Q(H)$$

(safety guarantees that G and H have the same free variables, thus, $Q(G)$ and $Q(H)$ have the same format).

(4) $F = \exists X : G$:

$$Q(F) = \pi[\textit{Vars}(Q(G)) \setminus \{X\}](Q(G)) \ ,$$

(5) $F = \neg G$, where $Q(G)$ has columns/variables $X_1, \ldots, X_k$:

$$Q(F) = \rho[\$1 \to X_1, \ldots, \$k \to X_k](ADOM^k) - Q(G)$$

(6) $F = G_1 \wedge \ldots \wedge G_l$, $l \geq 2$ is a maximal conjunctive subformula (difference to (2): now it's the induction step where the conjuncts are allowed to be complex subformulas): $Q(F)$ is then constructed analogously to (2) as a join.

Understanding the Proof: Negation as Minus

The $ADOM^k$ in "calculus to algebra" item (5) looks awkward. What is it good for? What does it *mean*?

- according to Def. 8.3 (4) (max. conjunctive subformulas), all the variables $X_1, \ldots, X_k$ in a negative conjunct $\neg G$ must occur positively in some other conjunct (and be bound by this).

$\Rightarrow$ instead of $ADOM^k$, the cartesian product (or any overestimate of it) of the possible values of $X_1, \ldots, X_k$ can be used.

- Formal example next slide,
- practical MONDIAL example second next slide.

Understanding the Proof: Negation as Minus

Formal Example

$$F(X,Y) = p(X,Y,Z) \wedge \neg\exists V : q(Y,Z,V) \ .$$

- $F_1(X,Y,Z) = p(X,Y,Z) \quad \Rightarrow \quad E_1 = \rho[\$1{\to}X, \$2{\to}Y, \$3{\to}Z](p)$,
- $F_2(Y,Z,V) = q(Y,Z,V) \quad \Rightarrow \quad E_2 = \rho[\$1{\to}Y, \$2{\to}Z, \$3{\to}V](q)$,
- $F_3(Y,Z) = \exists V : F_2(Y,Z,V) \quad \Rightarrow \quad E_3 = \pi[Y,Z](E_2) =$
  $\pi[Y,Z](\rho[\$1{\to}Y, \$2{\to}Z, \$3{\to}V](q))$,
- $F_4(Y,Z) = \neg F_3(Y,Z) \ \Rightarrow \ \rho[\$1{\to}Y, \$2{\to}Z](ADOM^2) - E_3 =$
  $\rho[\$1{\to}Y, \$2{\to}Z](ADOM^2) - \pi[Y,Z](\rho[\$1{\to}Y, \$2{\to}Z, \$3{\to}V](q))$
  (yields all possible $(y,z) \in ADOM^2$ that are not in ...)
- $F_5(X,Y,Z) = F_1 \wedge F_4 \quad \Rightarrow \quad E_1 \bowtie E_4 =$
  $E_1 \bowtie (\rho[\$1 \to Y, \$2 \to Z](ADOM^2) - \pi[Y,Z](\rho[\$1{\to}Y, \$2{\to}Z, \$3{\to}V](q)))$
  Only pairs (Y,Z) can survive the join that are in the result of the first component. Thus, instead taking the "overestimate" $ADOM^2$, $\pi[Y,Z](E_1)$ can be used:
  $E_1 \bowtie (\pi[Y,Z](E_1) - \pi[Y,Z](\rho[\$1{\to}Y, \$2{\to}Z, \$3{\to}V](q)))$.

Negation as Minus - A practical example

- Ever seen this $ADOM$ construct in exercises to the relational algebra? – No. Why not?

Consider relations country(name,country) and city(name,country,population):

$$F(CN,C) = \text{country}(CN,C) \wedge \neg\exists Cty, Pop : (\text{city}(Cty,C,Pop) \wedge Pop > 1000000)$$

Structural generation of an equivalent algebra expression:

- $F_1(CN,C) = \text{country}(CN,C) \quad \Rightarrow \quad E_1 = \rho[\$1 \to CN, \$2 \to C](\text{country})$,
- $F_2(Cty,C,Pop) = \text{city}(Cty,C,Pop) \wedge Pop > 1000000$
  $\Rightarrow \quad E_2 = \rho[\$1 \to Cty, \$2 \to C, \$3 \to Pop](\sigma[\$3 > 1000000](\text{city}))$,
- $F_3(C) = \exists Cty, Pop : F_2(Cty,C,Pop)$
  $\Rightarrow \quad E_3 = \pi[C](\rho[\$1 \to Cty, \$2 \to C, \$3 \to Pop](\sigma[\$3 > 1000000](\text{city})))$,
- $F_4(C) = \neg F_3(C) \ \Rightarrow \quad E_4 \ = \ \rho[\$1 \to C](ADOM) - E_3$ (abbreviating $\pi(\rho(\ldots))$ in E_3)
  $= \ \rho[\$1 \to C](ADOM) - \pi[\$2 \to C](\sigma[\$3 > 1000000](\text{city}))$
  (yields all possible C that are not in ...)
  At this point, one knows that not the complete $ADOM$ (all values anywhere in the database) has to be considered, but that it is sufficient to consider all countrycodes:
  $E_4' = \pi[\$2 \to C](country) - \pi[\$2 \to C](\sigma[\$3 > 1000000](\text{city}))$

Example (Cont'd)

And now, both parts of the outer conjunction are combined by a join:

$$F(CN, C) = F_1(CN, C) \wedge F_4(C)$$
$$\Rightarrow \; E_1 \bowtie E_4' =$$
$$\rho[\$1{\rightarrow}CN, \$2{\rightarrow}C](\mathsf{country}) \bowtie (\pi[\$2{\rightarrow}C](country) - \pi[\$2{\rightarrow}C](\sigma[\$3 > 1000000](\mathsf{city})))$$

8.7 Symbolic Reasoning

- Logics in general, and FOL are mathematical concepts.
  Research mathematically investigates different logics and their properties.
- *Symbolic Reasoning* applies logic-based *algorithms* on concrete problems, e.g.,
  - Software and hardware verification (e.g., correctness of automobile or airplane systems)
  - Answering queries against knowledge bases
- algorithms must operate on the syntax level:
  - formulas (i.e., parse-trees of formulas)
  - terms (i.e., parse-trees of terms)
  - sets of variable bindings
    - term unification,
    - answer bindings (to unification/matching and to queries)

DATALOG: HERBRAND SEMANTICS

Logic programming (LP) frameworks (e.g., Prolog and Datalog) use the *Herbrand Semantics* (after the French logician Jacques Herbrand):

- a *Herbrand Interpretation* $\mathcal{H} = (H, \mathcal{D}_\Sigma)$ for a given signature Σ uses always the *Herbrand Universe* $\mathcal{D}_\Sigma$ that consists of all terms that can be constructed from the function symbols (incl. constants) in Σ: john, $\mathsf{father(john)}$, $\mathsf{germany}$, $\mathsf{capital(germany)}$, berlin, ...

⇒ "every term is interpreted by itself"

- the relation names are the predicate symbols in Σ, and they are also "interpreted by themselves (as a relation)", i.e., $H(\mathsf{encompasses}) = \mathsf{encompasses}$.
- the *Herbrand Base* $\mathcal{HB}_\Sigma$ is the set of all *ground atoms* over elements of the Herbrand Universe and the predicate symbols of Σ.

⇒ A Herbrand Interpretation is a (finite or infinite) subset of the Herbrand Base.

- $\mathcal{H} \models \mathsf{hasAncestor(john,father(john))}$ if $(\mathsf{john}, \mathsf{father(john)}) \in \mathsf{hasAncestor}$.
- in contrast, in traditional FOL:
  $(I, \mathcal{D}) \models \mathsf{hasAncestor(john,father(john))}$ if $(I(\mathsf{john}), I(\mathsf{father}(I(\mathsf{john})))) \in I(\mathsf{hasAncestor})$.
- if function symbols are allowed, usually with equality predicate $\approx$, e.g., $\mathsf{father(john)} \approx \mathsf{jack}$.

Deductive Databases: Datalog

- the domain consists of constant symbols and datatype literals.
- an interpretation $\mathcal{H}$ is explicitly seen as a *finite* set of ground atoms over the predicate symbols and the Herbrand Universe:
  country(ger,"Germany","D", berlin, 356910,83536115), encompasses(ger, eur, 100).

  $\mathcal{H} \models \mathsf{encompasses(ger,eur,100)}$ if and only if $(\mathsf{ger}, \mathsf{eur}, 100) \in \mathcal{H}(\mathsf{encompasses})$
  if and only if $\mathsf{encompasses(ger, eur,100)} \in \mathcal{H}$.
- Unique Name Assumption (UNA): different symbols mean different things.
- Datalog restricts the allowed formulas (cf. Slides 557 ff.):
  - conjunctive queries,
  - Datalog knowledge bases consist of rules of the form $head \leftarrow body$
    (variants: positive nonrecursive, recursive, + negation in the body, + disjunction in the head)
- special semantics/model theories for each of the variants: minimal model, stratified model, well-founded model, stable models
  – each of them characterized as sets of ground atoms.

SEMANTIC WEB: RDF, RDFS, AND OWL

- RDF data model (see also Slide 440)
  - unary and binary predicates over literal values and URIs (Object (identifier)s; classes and properties are also represented by URIs)
- RDFS (RDF Schema): adds second order flavour:
  - RDF triples can have properties or classes as subject and object,
  - then use *predefined* RDFS predicates:
  - capital rdfs:domain Country; rdfs:range City.
    capital rdfs:subPropertyOf hasCity
  - semantics can be encoded in FOL rule patterns:
    $\forall x, y : \mathsf{capital}(x, y) \rightarrow \mathsf{Country}(x) \wedge \mathsf{City}(y)$
    $\forall x, y : \mathsf{capital}(x, y) \rightarrow \mathsf{hasCity}(x, y)$
  - mapped to FOL model theory.
  - RDFS and "OWL Lite" (see next slide) can be mapped to *positive recursive* Datalog
    $\Rightarrow$ polynomial
    * just positive rules: CWA and OWA semantics coincide

Semantic Web: RDF, RDFS, and OWL (cont'd)

- OWL: additional specialized vocabulary for describing Description Logic concepts
- Second order predicates – predicates about predicates:

borders a owl:SymmetricProperty.	SymmetricProperty(borders)
hasChild rdfs:subPropertyOf hasDescendant	hasChild $\sqsubseteq$ hasDescendant
hasDescendant a owl:TransitiveProperty.	TransitiveProperty(hasDescendant)

- many OWL (OWL Lite) constructs can be translated into FOL (and Datalog) rule patterns:
  $\forall x, y : \mathsf{borders}(x, y) \rightarrow \mathsf{borders}(y, x)$.
  $\forall x, y : \mathsf{hasChild}(x, y) \rightarrow \mathsf{hasDescendant}(x, y)$.
  $\forall x, y, z : \mathsf{hasDescendant}(x, y) \wedge \mathsf{hasDescendant}(y, z) \rightarrow \mathsf{hasDescendant}(x, z)$.
- Queries about data against RDF(+RDFS+OWL Lite) knowledge bases: *algebraic evaluation, polynomial.*
- Queries against RDF+OWL DL knowledge base: *reasoning, exponential.*