Chapter 3
An Introduction to Formal Logic

Overview

- basic notions of logics: syntax, semantics
- the relational calculus is a specialization of first-order logic, tailored to relational databases, equivalent to the relational algebra and SQL.
  Straightforward: the only structuring means of relational databases are relations – each relation can be seen as an interpretation of a predicate.
- there exists a **declarative** semantics,
- given by *model theory*,
- supported by reasoning methods.

3.1 Syntax

- **first-order language** contains a set of 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.

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 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:
  ?- $continent(\text{“Asia”}, X)$ means “for which value x does $continent(\text{“Asia”}, x)$ 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”).

3.2 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.
  $\Rightarrow$ we cannot express relations/relationships between predicates/functions.

First-Order Structures: An Example

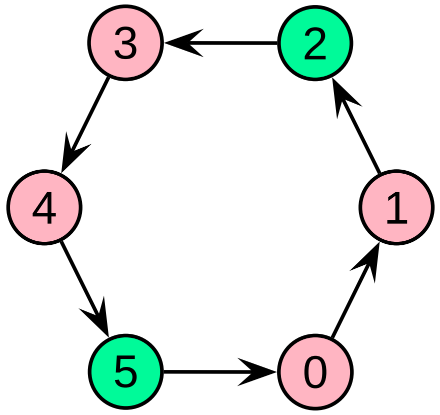
Example 3.1 (First-Order Structure)

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: $green(1),\ red(to\_right(to\_right(to\_right(four)))),\ sees(X, Y),$
$sees(X, to\_right(Z)),\ sees(to\_right(to\_right(four)), to\_right(one)),$
$plus(to\_right(to\_right(four)), to\_right(one)) = to\_right(three)$ □

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.,
  $I(encompassed)$
  (which we also write as $\mathcal{S}(encompassed)$).

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

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

that *binds* every variable to an element d of the domain.

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

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

Example 3.2
For variables X, Y, Z, $\beta = \{X \mapsto 1,\ Y \mapsto \text{“Asia”}, Z \mapsto 3.14\}$ is a variable assignment.

$\beta^3_X = \{X \mapsto 3,\ Y \mapsto \text{“Asia”}, Z \mapsto 3.14\}$. □

Evaluation of Terms and Formulas

Terms and formulas are interpreted

- under a given interpretation $\mathcal{S}$, and
- wrt. a given variable assignment β.

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

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

For Databases:

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

Evaluation of Terms

$\mathcal{S}(x, \beta) := \beta(x)$ for a variable x ,

$\mathcal{S}(c, \beta) := c$ for a 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 3.3 (Evaluation of Terms)
Consider again Example 3.1.

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

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 $(I(a_1), \ldots, I(a_k)) \in I(R)$.

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 $(\mathcal{S}(t_1, \beta), \ldots, \mathcal{S}(t_k, \beta)) \in I(R)$.

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 $d \in \mathcal{D}$, $\mathcal{S} \models_{\beta_X^d} G$.
- $\mathcal{S} \models_\beta \exists X G$ if and only if for some $d \in \mathcal{D}$, $\mathcal{S} \models_{\beta_X^d} G$.

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 instead of $\neg F \vee G$ (cf. Slide 39):
  $\mathcal{S} \models_\beta F \rightarrow G$ 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 domain, then all models 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 3.4 (Evaluation of Atomic Formulas)
Consider again Example 3.1.

- 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, $\mathcal{S} \models_\emptyset \; red(to\_right(to\_right(to\_right(four))))$.
- 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. $\square$

Example 3.5 (Evaluation of Compound Formulas)
Consider again Example 3.1.

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

  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 $\mathcal{S} \models_{\emptyset} red(six)$, 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 $\mathcal{S} \not\models_{\emptyset} green(one)$ 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, $\mathcal{S} \not\models_{\emptyset} \forall X : (green(X) \wedge red(X))$ □

3.3 Model Theory and Logical Entailment

Model

- a structure $\mathcal{S}$ where a formula F is true, is called a *model* of the formula.

Databases

- the signature of a (relational) database is a first-order signature
- the possible database states are interpretations of that signature (i.e., assigning a relation to each relation symbol)
- *integrity constraints* constrain the “allowed” states
- each integrity constraint can be expressed as a logical formula over the signature
  - check constraints like “population $\geq$ 0”:

$$\forall N, C, Pop, Area, Cap, CapProv : (country(N, C, Pop, Area, Cap, CapProv) \rightarrow Pop \geq 0)$$

  - referential integrity constraints:

$$\forall N, C, Pop, Area, Cap, CapProv : (country(N, C, Pop, Area, Cap, CapProv) \rightarrow \exists Pop', Long, Lat : city(Cap, CapProv, C, Pop', Long, Lat))$$

DATABASES VS. KNOWLEDGE BASES

- A *database (state)* is a *relational structure*.

  We can check if a formula holds there, or for which values of X it holds (which is then a query).

  The *semantics* of a database is the *current database state*.

- A (first-order) *knowledge base* is a set of closed (first-order) formulas. It contains *facts*, but also other *formulas*.

  We are interested if a knowledge base $\mathcal{K}$ *implies* a fact or a formula F. This means, if for *all* models $\mathcal{M}$ of $\mathcal{K}$, F must be true in $\mathcal{M}$.

  The semantics of a knowledge base (or in general a set of formulas) is the *set of all its models*.

- an intermediate form occurs when a database is extended by axiomatic formulas (subclasses etc.) or rules that can be used to derive additional facts.
  Then, the semantics is given by the model(s) of the database state and the rules.
  Is the (Semantic Web) more like a database or more like a knowledge base?

Example: Axiomatization of the “Company” Ontology

Consider again the ER diagram from Slide 22.

- give the *first-order signature* Σ of the ontology,
- formalize the constraints given in the
  - subclass constraints
  - range and domain constraints
  - cardinality constraints

  and

  - additional constraints/definitions that cannot be expressed by the ER model.

  (this set of formulas is called a first-order “theory” or “axiomatization” of the ontology)
- express the instance level as an interpretation of the signature Σ.

Example: Signature

- Classes are represented by unary predicates: Emp/1, Mgr/1, AMgr/1, TMgr/1, Dept/1.
- Attributes are represented by binary predicates: name/2, salary/2 (optionally, this could be modeled by unary functions)
- (binary) relationships are represented by binary relationships: wf/2, mg/2, sub/2.

Thus,
$\Sigma_{company} = \{$Emp/1, Mgr/1, AMgr/1, TMgr/1, Dept/1, name/2, salary/2, wf/2, mg/2, sub/2$\}$.

Example: Subclass Constraints

$$\begin{aligned} \forall x : \text{Mgr}(x) &\rightarrow \text{Emp}(x)\ , \\ \forall x : \text{AMgr}(x) &\rightarrow \text{Mgr}(x)\ , \\ \forall x : \text{TMgr}(x) &\rightarrow \text{Mgr}(x)\ , \\ \forall x : \text{Mgr}(x) &\rightarrow (\text{AMgr}(x) \vee \text{TMgr}(x)) \ \ \text{since declared as covering} \end{aligned}$$

Example: Domain and Range Constraints

$$\forall x : (\exists n : \textsf{name}(x,n) \rightarrow (\textsf{Emp}(x) \vee \textsf{Dept}(x)))\ ,$$
$$\forall x : (\exists s : \textsf{salary}(x,s) \rightarrow (\textsf{Emp}(x)))\ ,$$
$$\forall x : \forall y : (\textsf{sub}(x,y) \rightarrow (\textsf{Emp}(x) \wedge \textsf{Mgr}(y)))\ ,$$
$$\forall x : \forall d : (\textsf{wf}(x,d) \rightarrow (\textsf{Emp}(x) \wedge \textsf{Dept}(d)))\ ,$$
$$\forall x : \forall d : (\textsf{mg}(y,d) \rightarrow (\textsf{Mgr}(y) \wedge \textsf{Dept}(d)))\ .$$

Example: Cardinality Constraints

$$\forall m : (\textsf{TMgr}(m) \rightarrow \exists d : \textsf{mg}(m,d))\ ,$$
$$\forall m, d_1, d_2 : ((\textsf{mg}(m,d_1) \wedge \textsf{mg}(m,d_2)) \rightarrow d_1 = d_2)\ ,$$
$$\forall d : (\textsf{Dept}(d) \rightarrow \exists m : \textsf{mg}(m,d))\ ,$$
$$\forall d, m_1, m_2 : ((\textsf{mg}(m_1,d) \wedge \textsf{mg}(m_2,d)) \rightarrow m_1 = m_2)\ ,$$
$$\forall d : (\textsf{Dept}(d) \rightarrow \exists x : \textsf{wf}(x,d))\ ,$$
$$\forall x : (\textsf{Emp}(x) \rightarrow \exists d : \textsf{wf}(x,d))\ ,$$
$$\forall x : ((\exists d_1, d_2, d_3, d_4 : \textsf{wf}(x,d_1) \wedge \textsf{wf}(x,d_2) \wedge \textsf{wf}(x,d_3) \wedge \textsf{wf}(x,d_4)) \rightarrow$$
$$(d_1 = d_2 \vee d_1 = d_3 \vee d_1 = d_4 \vee d_2 = d_3 \vee d_2 = d_4 \vee d_3 = d_4))$$

Example: Further Constraints

- a person is subordinate to the manager of each department he/she works for:

$$\forall x, y, d : \mathsf{wf}(x, d) \land \mathsf{mg}(y, d) \land x \neq y \rightarrow \mathsf{sub}(x, y)$$

- should we have mg $\subseteq$ wf, or mg $\cap$ wf $= \emptyset$?
  The first is OK: $\forall y, d : \mathsf{mg}(y, d) \rightarrow \mathsf{wf}(y, d)$

- cardinality of “subordinate”? “Every employee has a boss” [Topic for Discussion]

$$\forall x : \mathsf{Emp}(x) \rightarrow \exists y : \mathsf{sub}(x, y)$$

  – this causes a semantical problem with the boss: an infinite chain is needed - leading either to only infinite models, or a cycle.
  – add an axiom that guarantees irreflexivity for “subordinate”: $\forall x : \neg\mathsf{sub}(x, x)$.
    Then the set of axioms has only one model: Emp is empty, everything is empty.
  – add an axiom that guarantees that the company has at least one employee

$$\exists x : \mathsf{Emp}(x)$$

    then the set of axioms is unsatisfiable.
  – such investigations help to validate an ontology.
    Ontology design tools allow to check for inconsistency, empty classes etc.

Axiomatization of the “company” scenario

Denote the conjunction of the above formulas by $\text{Axioms}_{Company}$.

- For any database/knowledge base $\mathcal{S}$ using this scenario, $\mathcal{S} \models \text{Axioms}_{Company}$ is required.
- a database then described the individuals and their individual properties in this world.

Example: Instances

- The signature is extended by constant symbols for all *named* elements of the domain:

$$\Sigma_{my\_company} := \Sigma_{company} \cup \{\text{Alice/f0, Bob/f0, John/f0, Mary/f0, Tom/f0, Larry/f0}\}$$

  (Note: the signature symbols are capitalized, wereas alice, bob etc denote the elements of the domain).

- first-order structure $\mathcal{S} = (I, \mathcal{D})$ as ...

- Domain $\mathcal{D} = \{$alice, bob, john, mary, tom, larry, sales, prod, mgm$\}$

- map constant symbols (nullary function symbols) to $\mathcal{D}$:
  $I(\text{Alice}) = \text{alice}$, $I(\text{Bob}) = \text{bob}$, $\ldots$, $I(\text{Sales}) = \text{sales}, \ldots$.

- map unary predicates to subsets of the domain $\mathcal{D}$:
  $I(\text{Emp}) = \{\text{alice, bob, john, mary, tom, larry}\}$, $I(\text{Mgr}) = \ldots$, $I(\text{Dept}) = \ldots$, $\ldots$,

- map binary predicates to subsets of $\mathcal{D} \times \mathcal{D}$:
  $I(\text{wf}) = \{$(alice, sales), (mary, sales), (larry, sales), (bob, prod), (bob, sales), (tom, prod), (john, mgm)$\}$
  $I(\text{mg})$, $I(\text{sub})$, $I(\text{name})$, $I(\text{salary})$ see Slide 24.

Example: Instances (Cont'd)

The axiomatization of "my company" with the given individuals is then given as the conjunction of

- all above constraints (general axiomatization of a company) and
- literal formulas describing the individuals:

$\text{Axioms}_{Company} \ldots \wedge \text{Emp}(\text{Alice}) \wedge \text{Emp}(\text{Bob}) \wedge \ldots \wedge \text{Dept}(\text{Sales}) \wedge \ldots \wedge \text{Mgr}(\text{Alice}) \wedge \ldots \wedge \text{wf}(\text{Alice}, \text{Sales}) \wedge \ldots \wedge \text{mg}(\text{Alice}, \text{Sales}) \wedge \ldots \wedge \text{sub}(\text{Mary}, \text{Alice}) \wedge \ldots.$

Example: Instances (Alternative)

- alternatively, instead of a signature extensions, all individuals can be described by *existentially* quantified variables:

$\text{Axioms}_{Company} \wedge$
$\exists alice, bob, \ldots, sales, \ldots : (\text{name}(alice, \text{“Alice”}) \wedge \ldots \wedge \text{Emp}(alice) \wedge \text{Emp}(bob) \wedge \ldots \wedge \text{Dept}(sales) \wedge \ldots \wedge \text{Mgr}(alice) \wedge \ldots \wedge \text{wf}(alice, sales) \wedge \ldots \wedge \text{mg}(alice, sales) \wedge \ldots \wedge \text{sub}(mary, alice) \wedge \ldots)$

Logical Entailment

Definition 3.1
Let F and G two (closed) formulas over a signature Σ. We write

$$F \models G \ \ (F \text{ logically entails } G)$$

when for *each* interpretation $\mathcal{S}$ over Σ, if $\mathcal{S} \models F$ then also $\mathcal{S} \models G$. $\square$

Example

$$\forall x : ((p(x) \rightarrow q(x)) \wedge (q(x) \rightarrow r(x))) \ \models \ \forall x : (p(x) \rightarrow r(x))$$

Logical Entailment as Proof

- usually F is a “large” conjunctive formula, containing the specification, and G is a “claim” to be shown to be a logical consequence of F.

Logical Entailment in a Knowledge Base

- for a FOL knowledge base, it is not always necessary to give all facts explicitly,
- axioms and *some* “basic” facts are often sufficient,
- further facts can be proven/added to the KB by *logical entailment*,
- further universally quantified formulas can be derived,
- entailment is also relevant when verifying consistency (satisfiability) of an ontology specification.

(most of this: see later)

How to Prove Entailment?

- it is not necessary (and not possible) to compute all models to check if something holds,
- it is sufficient to *prove* by symbolic reasoning if a formula is *implied* by a knowledge base.

Logical Entailment: Example

Consider $\text{Axioms}_{Company} \wedge \text{mg}(\text{Alice}, \text{Sales})$.

Does Emp(Alice) hold in each model $\mathcal{S} = (\mathcal{D}, I)$ (= is it logically entailed)?

- $\mathcal{S} \models \text{mg}(\text{Alice}, \text{Sales})$ implies $(I(\text{Alice}), I(\text{Sales})) \in I(\text{mg})$, i.e., $(\text{alice}, \text{sales}) \in I(\text{mg})$.
- $\mathcal{S} \models \forall y, d : \text{mg}(y, d) \rightarrow \text{wf}(y, d)$ (axiom)
  implies that for all $d_1, d_2 \in \mathcal{D}$, $\mathcal{S} \models_{\{y/d_1, d/d_2\}} \text{mg}(y, d) \rightarrow \text{wf}(y, d)$ which means that if $\mathcal{S} \models_{\{y/d_1, d/d_2\}} \text{mg}(y, d)$, then also $\mathcal{S} \models_{\{y/d_1, d/d_2\}} \text{wf}(y, d)$. The former is equivalent to $(d_1, d_2) \in I(\text{mg})$ that we have shown above for $(\text{alice}, \text{sales})$. Thus, we know that $(\text{alice}, \text{sales}) \in I(\text{wf})$.
- With the same argument as above, use the axiom
  $\mathcal{S} \models \forall x : \forall d : (\text{wf}(x, d) \rightarrow (\text{Emp}(x) \wedge \text{Dept}(d)))$ for concluding that alice $\in I(\text{Emp})$ which means that $\mathcal{S} \models \text{Emp}(\text{Alice})$.

How to use Entailment?

- “manual” mathematical proof, using the semantic level.
- use algorithms for deriving entailed facts or formulas, or for checking entailment by automated, symbolic reasoning.

VALIDITY AND DECIDABILITY

- preferably use a decidable logic/formalism
- with a *complete* calculus/reasoning mechanism
- Propositional logic: decidable
- First-order logic: undecidable
- Horn subset (= positive rules, with a special model theory) of FOL: decidable
  with negation in the body: still decidable
- 2-variable-subset of FOL: decidable
- Description Logic subsets of FOL: range from decidable to undecidable

3.4 DB vs. KB: Closed World vs. Open World

Consider the following formula F:

$$\begin{aligned} F \quad \equiv \quad & person(\text{“John”}, 35) \land person(\text{“Alice”}, 10) \land person(\text{“Bob”}, 8) \land \\ & \land person(\text{“Carol”}, 12) \land person(\text{“Jack”}, 65) \land \\ & \land child(\text{“John”}, \text{“Alice”}) \land child(\text{“John”}, \text{“Bob”}) \land \\ & \forall X, Y : (\exists Z : (child(Z, X) \land child(Z, Y) \land X \neq Y) \rightarrow sibling(X, Y)) \end{aligned}$$

- Does $child(\text{“John”}, \text{“Bob”})$ hold? – obviously yes.
- Does $G{:}{\equiv}sibling(\text{“Alice”}, \text{“Bob”})$ hold?
  - (Relational) Database: $sibling$ is a view. The answer is “yes”.
  - FOL KB: for all models $\mathcal{M}$ of F, G holds. Thus, $F \models sibling(\text{“Alice”}, \text{“Bob”})$.
- What about $G{:}{\equiv}sibling(\text{“Alice”}, \text{“Carol”})$?
  - (Relational) Database: no. For the database state $\mathbf{D}$, $\mathbf{D} \not\models sibling(\text{“Alice”}, \text{“Carol”})$.
  - FOL KB: there is a model $\mathcal{M}_1$ of F, where $\mathcal{M}_1 \not\models G$, but there is also a model $\mathcal{M}_2$ of F, where $\mathcal{M}_2 \models G$ (e.g., add the tuple $(\text{“John”}, \text{“Carol”})$ to the interpretation of $child$).

  For the Web, $child(\text{“John”}, \text{“Carol”})$ can e.g. be contributed by another Web Source.

DB vs. KB: Closed World vs. Open World

- What about $G :\equiv child(\text{“John”}, \text{“Jack”})$?
  - (Relational) Database: no. For the database state $\mathbf{D}$, $\mathbf{D} \not\models child(\text{“John”}, \text{“Jack”})$.
  - FOL KB: there is a model $\mathcal{M}_1$ of F, where $\mathcal{M}_1 \not\models G$, but there is also a model $\mathcal{M}_2$ of F, where $\mathcal{M}_2 \models G$.
- Obviously, the KB does not know that a child cannot be older than its parents.

  Add a constraint to F, obtaining F':

  $$F' :\equiv F \wedge \forall P, C, A_1, A_2, : (person(P, A_1) \wedge person(C, A_2) \wedge child(P, C)) \rightarrow A_1 > A_2$$

  - database: this assertion would prevent to add $child(\text{“John”}, \text{“Jack”})$ to the database.
  - for the KB, $F' \models \neg child(\text{“John”}, \text{“Jack”})$ allows to *infer* that Jack is not the child of John.

  Such information can be given with the *ontology* of a domain.

DB vs. KB: Closed World vs. Open World

- the Database Model Theory is called *"Closed World"*: things that are not known to hold are *assumed* not to hold.
- the FOL semantics is called *"Open World"*: things that are not known to be true or false are considered to be *possible*.

Consequences on Negation

- in databases there is no explicit negation. It is not necessary to specify that Jack is *not* a child of John.
- in a KB, it would be necessary to state ... $\wedge \neg child($"John"$, X)$ for all persons who are known not to be children of John.
  Additional constraints: extend the ontology, e.g., by stating that a person has exactly two parents – then all others cannot be parents – works only for persons whose parents are known. Similarly for the "age" constraint from the previous slide.
- note that the semantics of universal quantification (∀) is also effected: $\forall X : \phi$ is equivalent to $\neg\exists X : \neg\phi$.

Reasoning in Presence of Negation

Obtaining new information (e.g., by finding another Web Source) has different effects on Open vs. Closed world:

- Closed world: conclusions drawn before – “Carol is not a child of John”, or “John has exactly two children” from less information can become invalid.

  This kind of reasoning is *nonmonotonic*

- Open world: everything which is not known explicitly is taken into account to be possible (by considering all possible models).

  This kind of reasoning is *monotonic*:

  $$\text{Knowledge}_1 \subseteq \text{Knowledge}_2 \Rightarrow \text{Conclusions}_1 \subseteq \text{Conclusions}_2$$

- Open World can be combined with other forms of nonmonotonic reasoning, e.g., Defaults: “usually, birds can fly”. Knowing that Tweety is a bird allows to conclude that it flies.

  Obtaining the information that Tweety is a penguin (which can usually not fly) leads to invalidation of this conclusion.

The current Semantic Web research mainstream prefers Open World without default reasoning.

Comparison, Motivation etc.

Database vs. FOL

Relational Databases	relational schema	tuples	SQL queries	closed world
FOL	signature (predicates +functions)	facts (atoms)	$\mathcal{S} \models \phi$? (yes/no or answer $\psi \models \phi$? variable bindings)	mostly: open world sometimes closed world

Situations and tasks

Given	what to do	how?
facts/database	does $p(\ldots)$ hold in the DB? SQL query	by combining data
facts+constraints (SQL assertions or FOL formulas)	additionally: test if constraints satisfied	equivalent to first situation (query for violating tuples)
facts (DB) rules (KB)	does $p(\ldots)$ hold in DB+rules?	DB+views application of rules
facts (DB) knowledge base KB as FOL formulas	is a formula ϕ entailed by DB+KB?	reasoning, entailment, KB $\models \phi$?

3.5 Databases: Formulas as Queries in a Closed World

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

- For a formula F with free variables $X_1, \ldots, X_n$, $n \geq 1$, we 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 $free(F)\}$.
- 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 3.6

Consider the MONDIAL schema.

- *Which cities (CName, Country) have at least 1.000.000 inhabitants?*

$$F(CN, C) = \quad \exists\, Pr, Pop, L1, L2\ (\textsf{city}(CN, C, Pr, Pop, L1, L2) \wedge\ Pop \geq 1000000)$$

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

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

□

CONJUNCTIVE QUERIES

... the above ones were *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.

Playing with Relational Facts

The F-Logic system “Florid” (F-Logic Reasoning in Databases) can be used for evaluating conjunctive queries over a relational database:

- installed in the CIP pool (see lecture Web page for details)
- use `mondial-rel-facts.flp`, a relational representation of Mondial.

```
> florid mondial-rel-facts.flp
// reads and parses the file that contains the facts.
This is Florid, Version 4.0 <21/09/06>    // it’s its 4th life now.
Type ’sys.help.’ for further information.

?- sys.eval.    // evaluates the facts and adds it to the knowledge base
?- country(A,B,C,D,E,F).
// answers ...

// use don’t care variables for projection: exists _Cap, _CapProv,... :
?- country(N,CC,_Cap,_CapProv,_A,_P).
?- sys.eval.

?- sys.end.     // to leave it
```

Example 3.7

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

$$
\begin{aligned}
F(Abbrev) = \exists O, HeadqN, HeadqC, HeadqP, Est :\\
&(organization(O, Abbrev, HeadqN, HeadqC, HeadqP, Est) \wedge\\
&\forall Cont : \; ((\exists ContArea : \; continent(Cont, ContArea)) \rightarrow\\
&\qquad \exists Country, Perc, Type : (encompasses(Country, Cont, Perc) \wedge\\
&\qquad\qquad is\_member(Country, Abbrev, Type))))
\end{aligned}
$$

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

$$
\begin{aligned}
F(CCode, Org) = \exists CName, Cap, Capprov, Pop, Area, Type :\\
&(\textsf{country}(CName, CCode, Cap, Capprov, Pop, Area) \wedge\\
&\textsf{is\_member}(CCode, Org, Type) \wedge\\
&\forall CCode' : (\exists Length : \textsf{sym\_borders}(CCode, CCode', Length) \rightarrow\\
&\qquad \neg\exists Type' : \textsf{is\_member}(CCode', Org, Type')))
\end{aligned}
$$

□

Relational Query Formalisms: Comparison of the Algebra and the Calculus

Calculus: The semantics (= answer) of a query in the relational calculus is defined via the truth value of a formula wrt. an interpretation

“**model-theoretic**, **declarative** Semantics”.

Algebra: The semantics is given by evaluating an algebraic expression (i.e., an operator tree)

“**algebraic** Semantics” (it is also declarative).

Example: Expressing Algebra Operations in the Calculus

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

- **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** $R \cup U$:
$$F(X,Y) = R(X,Y) \vee U(X,Y)$$
- **Difference** $R - U$:
$$F(X,Y) = R(X,Y) \wedge \neg U(X,Y)$$
- **Division** $R \div T$:

$$F(Y) = \forall X : (T(X) \Rightarrow R(X,Y)) \qquad \text{or} \qquad F(Y) = \neg\exists X : (T(X) \wedge \neg R(X,Y))$$

Safety and Domain-Independence

- If the domain $\mathcal{D}$ is infinite, the answer relations to some expressions of the calculus can be infinite!

Example 3.8
Let

$$F(X) = \neg R(X),$$

("give me all a such that $R(a)$ does not hold")

where $\mathcal{S}(R) = \{1\}$.

$\mathcal{S}(F) = \mathcal{D} \setminus \{1\}$, which can be infinite (depending on what $\mathcal{D}$ is). $\Box$

Example 3.9
Let

$$F(X,Z) = \exists Y (R(X,Y) \vee S(Y,Z)),$$

Consider $\mathcal{S}(R) = \{(1,1)\}$, arbitrary $\mathcal{S}(S)$ (even empty).

Which Z? $\Box$

Example 3.10
Consider a database of persons:

married(X,Y): X is married with Y.

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

What is the answer?

- Consider $\mathcal{D} = \{john, mary\}$, $\mathcal{S}(married) = \{(john, mary), (mary, john)\}$.
  $\mathcal{S}(F) = \emptyset$.
  - there is no person (except John) who is not married with John
  - all persons are married with John???

  $\square$

- Consider $\mathcal{D} = \{john, mary, sue\}$, $\mathcal{S}(married) = \{(john, mary), (mary, john)\}$.
  $\mathcal{S}(F) = \{sue\}$.

  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 3.2
Given a formula F of the relational calculus and a database state $\mathcal{S}$, $DOM(F)$ contains

- all constants in F,
- and all constants in $\mathcal{S}(R)$ where R is a relation name that occurs in F.

$DOM(F)$ is called the **active domain** of F. $\square$

$DOM(F)$ is finite.

Domain-Independence

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

Definition 3.3
A formula $F(X_1, \ldots, X_n)$ is **domain-independent** if for all $D \supseteq DOM(F)$,

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

□

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

Definition 3.4
A formula F is **(syntactically) safe** if and only if it satisfies the following conditions:

1. F does not contain $\forall$ quantifiers. (for formal simplicity. 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 **bounded** [German: “bounded” = “beschränkt” im Gegensatz zu “bound” = “gebunden” bei Quantoren und Variablenbindungen].

   - there must be at least *one* conjunct for every variable that bounds it:
   - if a conjunct F_j is neither a comparison, nor a negated formula, any free variable in F_j is bounded,
   - if a conjunct F_j is of the form $X = a$ or $a = X$ with a a constant, then X is bounded,
   - if a conjunct F_j is of the form $X = Y$ or $Y = X$ and Y is bounded, then X is also bounded.

(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). $\square$

Example 3.11

- $R(X,Z) \vee X = Y$ is not safe
- $R(X,Y) \vee X = Y$ is not safe: $X = Y$ is a maximal conjunctive subformula where none of the variables is bounded
- $R(X,Y) \wedge X = Y$ is safe: $R(X,Y)$ bounds X and Y for the whole (conjunctive) formula.
- $R(X,Y) \wedge X = Z$ is safe: $R(X,Y)$ bounds X and Y, then $X = Z$ also bounds Z.
- $R(X,Y,Z) \wedge \neg(S(X,Y) \vee T(Y,Z))$ is not safe, but the logically equivalent formula

$$R(X,Y,Z) \wedge \neg S(X,Y) \wedge \neg T(Y,Z)$$

is safe. □

Summary

- safety is defined purely syntactically
- safety can be tested effectively
- safety implies domain-independence
(proof by induction on the number of maximal conjunctive subformulas).

3.6 Datalog Knowledge Bases

A Datalog “knowledge base” $\mathcal{K}$ consists of

- facts: $p(c_1, \ldots, c_n)$
- Rules of the form $p(X_1, \ldots, X_k) \leftarrow \exists X_{k+1}, \ldots, X_n : Q(X_1, \ldots, X_n)$
  where p is a k-ary predicate and Q is a conjunctive (positive!) query.
  - means: “whenever $Q(X_1, \ldots, X_n)$ holds for some $X_{k+1}, \ldots, X_n$ in my database, also $p(X_1, \ldots, X_k)$ is assumed to hold”.
  - SQL equivalent: p is a view.

SEMANTICS

Usually, the signature Σ is partitioned in two sets:

- Σ_{EDB}: predicates that occur only in the body of rules
  (“extensional database” – the interpretation of these predicates is given as facts in the knowledge base)
- Σ_{IDB}: predicates that occur in the head (and possibly also in the body) of rules
  (“intensional database” – the interpretation of these predicates is derived from the rules)

Interpretations and Models

- the domain/universe of a Datalog knowledge base $\mathcal{K}$ is the active domain of the database, i.e., the set of constants that occur in the facts and rules (called “Herbrand universe” after a the French logician Jacques Herbrand)
- for any model $\mathcal{S}$ of $\mathcal{K}$:
  - $\mathcal{S} \models p(c_1, \ldots, c_n)$ for each fact in $\mathcal{K}$,
  - $\mathcal{S} \models \forall X_1, \ldots, X_k : ((\exists X_{k+1}, \ldots, X_n : Q(X_1, \ldots, X_n)) \rightarrow p(X_1, \ldots, X_k))$.

Recursive Datalog

- *Recursive Datalog*: head predicate allowed to occur in the body

Transitive Closure

- tc(x,y) $\leftarrow$ p(x,y).
  tc(x,y) $\leftarrow \exists$ z: tc(x,z) $\wedge$ tc(z,y).

```
?- sys.load@("mondial-rel-facts.flp").
borders(Y,X,Z) :- borders(X,Y,Z).  // make it symmetric.
?- sys.strat.doIt.   // actually not necessary
reachable(X,Y) :- borders(X,Y,_).
reachable(X,Y) :- reachable(X,Z), borders(Z,Y,_).
?- sys.eval.
?- reachable("D",X).
?- reachable("D","THA").
?- reachable("D","USA").
```

[Filename: FL/ex-transitiveclosure.flp]

... so far: intuitive idea “derive $head$ if $body$ holds”.

SEMANTICS OF QUERIES WRT. A DATALOG KNOWLEDGE BASE

The formal semantics is given by *Herbrand Interpretations*:

Herbrand Interpretation

- Recall First-Order Logic interpretations $\mathcal{S} = (\mathcal{D}, I)$:
  0-ary constant symbols are interpreted by elements from the domain, e.g.,
  $I(\text{john}) = john \in \mathcal{D}$.

- Herbrand Interpretation: function symbols are “interpreted by themselves”: I(a) = a, where a is an element of the Herbrand Domain.

  (aside: same for terms: function symbols are also “interpreted by themselves – this is the base for the semantics of Prolog. Datalog has no (non-0-ary) function symbols)

Predicate symbols are actually interpreted by the Herbrand Interpretation:

A Herbrand Interpretation $\mathcal{H}$ to a program P is a set of *ground atoms* over the Herbrand Universe of P. (it is very similar to a “(relational) database” over identifiers.)

Example

$\{$country(a, vienna, 83850,8023244), country(d, berlin, 356910, 83536115), border(a,d,784), border(a,h,366), ...$\}$

Application of Rules on a Herbrand Interpretation

Consider a *positive* program (i.e., rules without negation).

- facts of the form $p(a_1, \ldots, a_n)$ can also be seen as rules:

  $$p(a_1, \ldots, a_n) \text{ :- true}$$

  "if true holds (which is always the case) then also $p(a_1, \ldots, a_n)$ must hold".

- Application of Rules:

  Example: reachable(X,Y) :- reachable(X,Z), borders(Z,Y,_).

  - General: a substitution σ is a mapping that maps terms to terms.
  - Here: we need only substitutions that map *variables* to *0-ary function symbols (constants)*:

  The set of ground facts that is derivable by a rule $H \leftarrow C_1 \wedge \ldots \wedge C_k$ wrt. a given Herbrand Interpretation $\mathcal{H}$ is formally specified as follows:

  $\{\sigma(B)$: σ is a ground substitution and there is a rule
  $B \leftarrow C_1 \wedge \ldots \wedge C_k$ in P such that $\sigma(C_1), \ldots, \sigma(C_k) \in \mathcal{H}\}$

  Example: reachable(X,Y) :- reachable(X,Z), borders(Z,Y,_)
  with $\sigma = \{$X $\mapsto$ d, Z $\mapsto$ bg, Y $\mapsto$ gr$\}$

Semantics of Queries wrt. a Datalog Knowledge Base

Consider a *positive* program (i.e., rules without negation).

- (ground (i.e. without variables)) facts of the form $p(a_1, \ldots, a_n)$,
- (non-ground) rules of the form $head$:- $body$.

For a set I of ground atoms,

$$\begin{aligned}
T_P(I) &:= \{\sigma(B)\text{: } \sigma \text{ is a ground substitution and there is a rule} \\
&\qquad B \leftarrow C_1 \wedge \ldots \wedge C_k \text{ in } P \text{ such that } \sigma(C_1), \ldots, \sigma(C_k) \in I\} \\
T_P^0(I) &:= I \\
T_P^1(I) &:= T_P(I) \\
T_P^{n+1}(I) &:= T_P(T_P^n(I)) \\
T_P^\omega(I) &:= \bigcup_{n \in \mathbf{N}} T_P^n(I)
\end{aligned}$$

- $T_P^\omega := T_P^\omega(\emptyset)$,
- The set T_P^ω contains all ground facts that can be derived from the program.

Example/Exercise

Consider the following program (including facts and rules):

$$
\begin{aligned}
P = \{ \quad & \text{border}(a,d).\ \text{border}(a,h).\ \text{border}(a,i).\ \text{border}(d,f).\ \text{border}(i,f).\\
& \text{border}(ch,f).\ \text{border}(ch,a).\ \text{border}(ch,d).\ \text{border}(ch,i).\ \text{border}(e,f).\ \text{border}(p,e).\\
& \text{border}(h,ua).\ \text{border}(ua,r).\ \text{border}(ra,br).\ \text{border}(bol,ra).\ \text{border}(bol,br).\\
& \text{border}(Y,X) \text{ :- } \text{border}(X,Y).\\
& \text{reachable}(X,Y) \text{ :- } \text{border}(X,Y).\\
& \text{reachable}(X,Y) \text{ :- } \text{reachable}(X,Z), \text{border}(Z,Y). \quad \}
\end{aligned}
$$

- Give $T_P^0(\emptyset)$, $T_P^1(\emptyset)$, $T_P^2(\emptyset)$, . . . , $T_P^\omega(\emptyset)$.
- for any derived fact reachable$(c_1, c_2) \in T_P^\omega(\emptyset)$, characterize the least i such that reachable$(c_1, c_2) \in T_P^i(\emptyset)$.

MINIMAL MODEL

For a positive Datalog program P, the *minimal model* is defined as the minimal model that satisfies the conditions given on Slide 86:

Theorem 3.1
For a positive Datalog program P and its *minimal model* $\mathcal{M}$,

- $\mathcal{M} \models p(c_1, \ldots, c_n) \Leftrightarrow p(c_1, \ldots, c_n) \in T_P^\omega$.
- $\mathcal{M} \models p(c_1, \ldots, c_n)$ if and only if for *all* models $\mathcal{S}$ of P, $\mathcal{S} \models p(c_1, \ldots, c_n)$. $\square$

Note:

- T_P is monotonous, i.e., if $I_1 \subseteq I_2$ then $T_P(I_1) \subseteq T_P(I_2)$.

Negation

- The T_P evaluation is not applicable for rules with negation in the body.
- Consider the previous example extended by the rule
  $\{$ unreachable(X, Y) :- country$(X) \wedge$ country$(Y) \wedge \neg$reachable(X, Y). $\}$.
  How would the T_P evaluation proceed for it?

Decision Procedures

Given: a positive Datalog program P
Question: does $p(c_1, \ldots, c_n)$ hold?

- bottom-up computation of T_P provides a *correct* and *complete* (wrt. the minimal model) procedure for checking if some *fact* holds in the minimal model.

Extended question (query):

for which elements $x_1, \ldots, x_n$ does $p(x_1, \ldots, x_n)$ hold can also be answered.

Every atom that is true in the minimal model has a “proof history” (tree) via the rules and facts that have been used for deriving it.

- an atom can be proven if it matches the head of a rule and all of the body atoms can be proven. Apply recursively.
  Note that multiple rule heads can match.

Top-down Decision Procedure

A “top-down” decision procedure does not compute the whole model, but starts with the claim and the knowledge base:

- resolution calculus

Resolution Calculus

The “Prolog” (**Pro**gramming in **Log**ic) language uses a restricted reasoning method, called resolution: (here presented in s simplified version restricted to facts)

- rules of the form $h(\bar{x}) \leftarrow b_1(\bar{x}), b_2(\bar{x}), \ldots, b_n(\bar{x})$
- equivalent to *Horn Clauses* $h(\bar{x}) \vee \neg b_1(x) \vee \neg b_2(\bar{x}) \vee \ldots \vee \neg b_n(\bar{x})$
  (Disjunction with only one positive literal; *generalized* Horn Clauses may have several positive literals)
- positive facts $p(\bar{c})$.
- given: a “program” P of rules and facts, and a claimed fact $p(\bar{c})$. Show: $P \models p(\bar{c})$?
- idea: for each clause, at least one literal must be satisfied.
- strategy: use unary clauses that match a literal in another one to “delete” the latter,
- try to derive the empty clause: then it is shown that $P \cup \{\neg p(\bar{c})\}$ is unsatisfiable, i.e., $P \models p(\bar{c})$,
- equivalent to minimal model semantics.

Resolution Calculus: Example

Consider the rule

$$\text{subordinate}(x,y) \leftarrow \text{works-for}(x,d) \wedge \text{manages}(y,d)$$

(forget about $x \neq y$ for now)

The clause is

$$\{\text{subordinate}(X,Y), \neg\text{works-for}(X,D), \neg\text{manages}(Y,D)\}$$

Consider the (unary) fact clauses $\{$works-for(mary,sales)$\}$ and $\{$manages(alice,sales)$\}$.

Does subordinate(mary,alice) hold?

$$\begin{tikzcd}
\{\neg\text{sub(m,a)}\} \arrow[dr, dash] & \{\text{sub}(X,Y), \neg\text{wf}(X,D), \neg\text{mg}(Y,D)\} \arrow[d, dash, "\{X \to \text{m}, Y \to \text{a}\}"'] & \{\text{wf(m,s)}\} \arrow[ddl, dash] & \{\text{mg(a,s)}\} \arrow[dddll, dash] \\
& \{\neg\text{wf}(\text{m},D), \neg\text{mg}(\text{a},D)\} \arrow[d, dash, "\{D \to \text{s}\}"'] & & \\
& \{\neg\text{mg(a,s)}\} \arrow[d, dash] & & \\
& \square & &
\end{tikzcd}$$

(any other order of applications is also correct, but taking a unary negative clause is the most goal-driven strategy)

Aside: Full Prolog

- first-order logic includes function symbols
- not just matching, but *unification* of terms (that contain variables somewhere)
- derives the empty clause *and* an answer substitution
  e.g. when asking ?-subordinate(X,alice).

  X/mary

  X/bob
- uses backtracking:
  - if search for an answer is not successful, try another way,
  - if an answer is found, report it and try another way,
  - generates a proof search tree.
- Prolog *Programming* goes even further: “cut” and “fail” to control the exploration of the search space.

⇒ theory and artificial intelligence lecture.

Negation in the Body

- rule body is allowed to contain also negative literals.
- Safety requirement: every variable that occurs in a negative literal must also occur in a positive one.

Intuitive Semantics

- $p(x) \leftarrow in\_domain(x) \wedge \neg q(x)$.
  "for all x for that $q(x)$ cannot be derived, one can conclude $p(x)$".
- unreachable(X,Y) :- country$(X) \wedge$ country$(Y) \wedge \neg$reachable(X,Y).

Formal Semantics

- The plain T_P operator is not suitable: In the first "round" things are false that will become true later

$\Rightarrow$ "wait" before using negative literals until this predicate is completely computed.

Negation in the Body: Stratification

- p depends on q if there is a rule with p in the head and q positively in the body,
- p depends *negatively* on q if there is a rule with p in the head and q negatively in the body,
- if the *dependency graph* does not contain a negative cycle, then there exists a simple, intuitive semantics:

Stratification

- define *strata* $S_0, \ldots, S_n \subset \Sigma_P$ such that
  - S_0: predicate symbols that do not depend negatively on any other predicate symbol
  - S_{i+1}: predicate symbols that depend negatively only on predicate symbols in $S_0, \ldots, S_i$.
  - if such a stratification is possible, a Datalog program P is called *stratifiable*.
  - allows to compute extensions of all $p \in S_i$ incrementally (bottom-up).
- Stratification applies *Closed World* reasoning to the EDB of a Datalog knowledge base (Negation as Failure in Prolog).

Stratified Nonrecursive Datalog with Negation

- Nonrecursive Datalog with (stratified) negation (see later for details about negation) is equivalent to the relational algebra
- we now can express division:
  (recall: division is defined in the relational algebra by two negations)

Organizations that have at least one member on each continent:

```
?- sys.load@("mondial-rel-facts.flp").
has_member_on(Org,Cont) :- is_member(C,Org,_Type), encompasses(C,Cont,_Perc).
?- sys.strat.doIt.
not_result(Org) :- organization(Org,_,_,_,_,_), continent(Cont,_Area),
       not has_member_on(Org,Cont).
?- sys.strat.doIt.
result(Org) :- organization(Org,_,_,_,_,_), not not_result(Org).
?- sys.strat.doIt.
?- result(Org).
```

[Filename: FL/ex-division.flp]

- compare with expressing this query in SQL.

Example

The knowledge base consists of facts + rules (= definitions of derived notions), e.g., what “landlocked” means:

- (ground) facts: literals over constants
  country(germany, “Germany”, “D”, 356910, 83536115, berlin)
  (“extensional knowledge”)
- (non-ground) *rules* (i.e., can contain constants and variables) (“intensional knowledge”)
  located-at-sea(C) :- country(C), sea(S,...), located(C,S).
  landlocked(C) :- country(C), $\neg$ located-at-sea(C).
  note: landlocked belongs to stratum S_1
- queries can be stated also against derived predicates:
  ?- landlocked(C), country(C, _, _, A, _, _), A $<$ 100000.
  e.g. yields C/switzerland

Exercise

Define the ontology of relationships (dt.: Verwandtschaftsbeziehungen) between persons as a Datalog KB.

Aside: Playing with Florid – Aggregation

- Syntax:
  result-var = agg{ var [group-by-vars]; query(that binds var and group-by-vars)}
- operators: count, min, max, avg, sum.

```
?- sys.load@("mondial-rel-facts.flp").
citypop(C,P) :- country(_,C,_,_,_,_),
     P= sum{ Pop [C]; city(_CN, C, _, Pop, _, _)}.
?- sys.eval.
?- citypop(C,P).
```

[Filename: FL/ex-aggregation.flp]

- Note: aggregation operations also require stratifications – the predicates used in the subquery must be computed before.

Negation in the Body: Cyclic Negative Dependencies

A program whose dependency graph contains a *negative cycle* cannot be stratified. There is no minimal model.

- Consider the program $P = \{p(b) \leftarrow \neg p(a)\}$ (without any assured facts). It has two models $\mathcal{MS} = \{p(b)\}$ and $\mathcal{MI} = \{p(a)\}$. Both are minimal.

  Which of the models is "preferable", given P as a knowledge base?

- stratification is not applicable
- well-founded semantics (still polynomial)
- stable semantics (answer set programming) (exponential)
- the rule is logically equivalent to $p(a) \vee p(b)$ – but as a rule, it can be read to have a more "directed" meaning:
  "if $p(a)$ cannot be shown, then assume $p(b)$".

$\Rightarrow$ ((not only) database) theory lecture

Example: Win-Move-Game

- 2 players,
- places on a board that are connected by (directed) moves (relation “move(x,y)”),
- first player puts a pebble on a position,
- players alternately move the pebble from x to a connected y,
- if a player cannot move, he loses.
- Question: which positions are “winning” positions, “losing” position, or “drawn” positions?

The following program completely “describes” the game:

```
win(X) :-  move(X,Y), not win(Y).
lose(X) :- not win(X).
```

Exercise:

- analyze the game given by move(1,2) and move(2,3).
- give sample situations with won, lost, and drawn positions.
- characterize the effect of cycles.

Restrictions of the Datalog/Minimal Model Semantics

Given a positive Datalog program P, the minimal model and the above procedure cannot decide the following:

- for a given FOL formula ϕ, does ϕ hold in *all* models of P?
- if $p(c_1, \ldots, c_n)$ can not be confirmed by the minimal or stratified model, this does *not* mean, that there is no model of P where $p(c_1, \ldots, c_n)$ holds.
  Even more, any positive fact can be added to a positive program without being inconsistent.

Closed-World-Assumption (CWA)

- For all facts that are not given in the database and that are not derivable, it is assumed that they do not hold (more explicitly: that their negation holds).
- CWA not appropriate in the Web: for things that I do not find in the Web, simply nothing is said.
  [Example: travel planning]

3.7 First-Order Logic and Ontologies

Consider the “company” ontology (cf. Slide 22).

- give the *first-order signature* Σ of the ontology,
- formalize the constraints given in the
  - subclass constraints
  - range and domain constraints
  - cardinality constraints

  and

  - additional constraints/definitions that cannot be expressed by the ER model.

  (this set of formulas is called a first-order “theory” or “axiomatization” of the ontology)
- express the instance level as an interpretation of the signature Σ.

First-Order Logic and Ontologies: Example

Theory of the “Company” Example:

- Axioms: see Slides 57

Instance level:

- give only base facts (here: name, salary, works-for and manages)
- derive the other relations (and classes as unary relations) logically (cf. Slide 66).

Example/Exercise

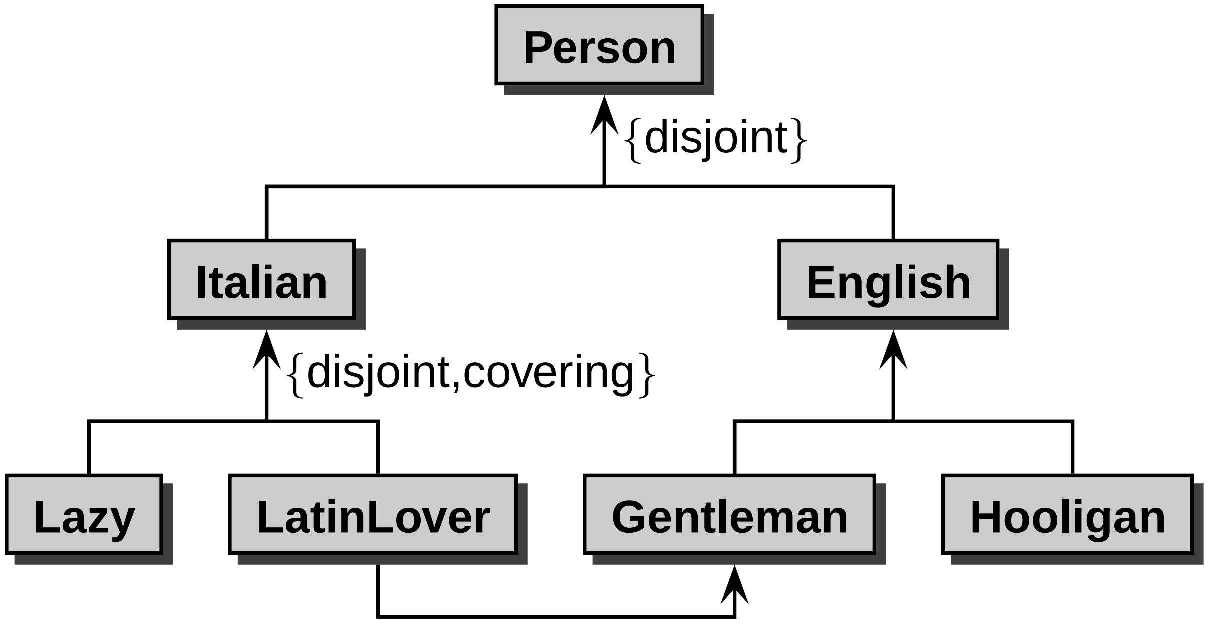
Prove the derivation from Slide 66 by using the tableau calculus (Slide 114).
[Proof see Slide 118]

Reasoning about Ontologies

Given an ontology,

- a class is *inconsistent* if it denotes the empty set in every model,
- a class C is a *subclass* of D if the extension of C is a subset of the extension of D in every model,
- two classes are *equivalent* if they denote the same set in every model,
- a constraint is *entailed* by an ontology if it holds in every model.

EXAMPLE: ITALIANS AND ENGLISHMEN



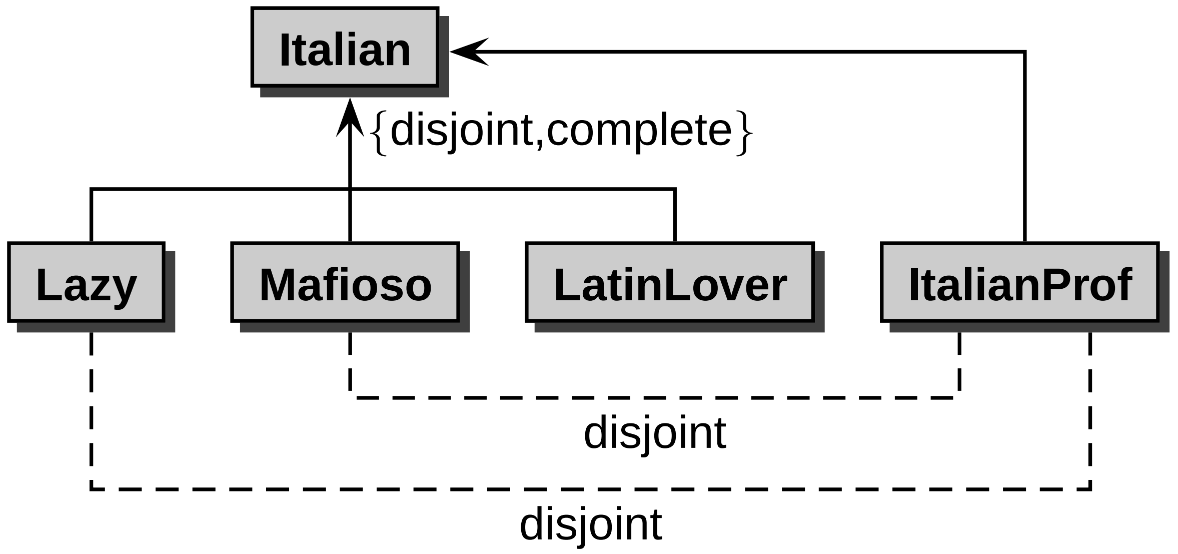
Exercise: write down as concise as possible *everything* that is implied by this ontology in text, set theory and first-order logic.

[by Enrico Franconi, REWERSE Summer School 2005]

[see Slide 126 for an excerpt and a relevant proof]

.

EXAMPLE: ITALIAN PROFESSORS

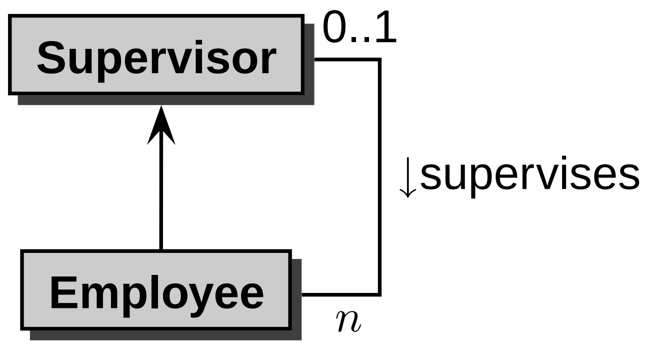


Exercise: write down as concise as possible *everything* that is implied by this ontology in text, set theory and first-order logic.

[by Enrico Franconi, REWERSE Summer School 2005]

Example: The Democratic Company

Every employee is also a supervisor. Every supervisor supervises 2 employees.

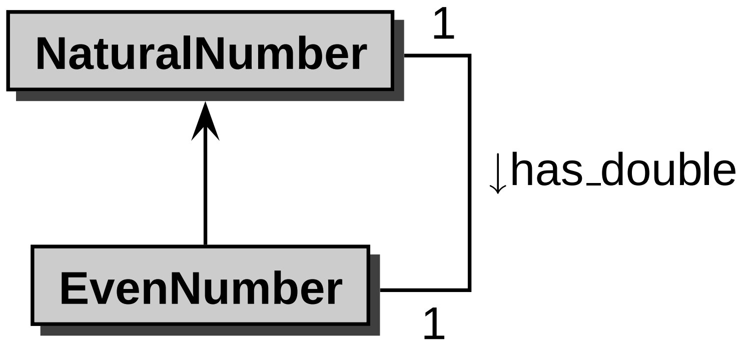


Exercise: Consider $n = 1$ and $n = 2$. Write down as concise as possible *everything* that is implied by this ontology in text, set theory and first-order logic.

[by Enrico Franconi, REWERSE Summer School 2005]

EXAMPLE: BIJECTION - HOW MANY NUMBERS

Every natural number is related to its double which is an even number.



- the classes "NaturalNumber" and "EvenNumber" contain the same number of,
- the same applied to a *finite* domain implies that NaturalNumber $\equiv$ EvenNumber (which e.g. holds in the *cyclic* modulo rings Z_3, Z_5 etc.).
  (note: in these rings, every element is even!)

[by Enrico Franconi, REWERSE Summer School 2005]

3.8 Reasoning

- Resolution calculus works only for clauses (extended Horn fragment).
- show $F \models G$: prove that $F \wedge \neg G$ is unsatisfiable:
  try to construct an example ("witness") for $F \wedge \neg G$ by a systematic algorithm.
  (below: e.g. using a tableau calculus)
- show satisfiability of a knowledge base/ontology:
  try to construct an example.
- answer a query (= a conjunctive formula Q with free variables) against a knowledge base given as a formula F:
  Construct counterexamples for $F \wedge \neg Q$; each of them is an answer.

$\Rightarrow$ one algorithm is sufficient.

First Order Tableau Calculus

- Systematic construction of an interpretation of a formula.
- Goal: show that this is not possible. Otherwise a counterexample is generated.
- counterexamples can be interpreted as answers to a query.

Start the tableau with an set $\mathcal{F}$ of formulas:

$$\boxed{\dfrac{\text{input set } \mathcal{F}}{F \quad \text{for all } F \in \mathcal{F}}}$$

The tableau is then extended by expansion rules.

TABLEAU RULES

α-rule (conjunctive): $\dfrac{F \wedge G}{\begin{array}{c}F\\G\end{array}}$ $\qquad$ $\dfrac{\neg(F \vee G)}{\begin{array}{c}\neg F\\\neg G\end{array}}$

β-rule (disjunctive): $\dfrac{F \vee G}{F \mid G}$ $\qquad$ $\dfrac{\neg(F \wedge G)}{\neg F \mid \neg G}$

γ-rule (universal): $\dfrac{\forall x : F}{F[X/x]}$ $\qquad$ $\dfrac{\neg\exists x : F}{\neg F[X/x]}$

where X is a new variable.

δ (existential): $\dfrac{\exists x : F}{F[f(\text{free}(T))/x]}$ $\qquad$ $\dfrac{\neg\forall x : F}{\neg F[f(\text{free}(T))/x]}$

where f is a new *Skolem function symbol* (after the Norwegian logician Thoralf Skolem) and T is the current branch of the tableau.

Closure Rule:

$$\dfrac{\begin{array}{c}\sigma(A)\\\neg\sigma(A)\end{array}}{\bot}$$

apply σ to the whole tableau.

RESULT

Definition 3.5

A branch T in a tableau $\mathcal{T}$ is *closed*, if it contains the formula $\perp$.
A tableau $\mathcal{T}$ is *closed* if every branch is closed. $\square$

CORRECTNESS

Definition 3.6

A Tableau $\mathcal{T}$ is *satisfiable* it there exists an interpretation $\mathcal{S} = (U, I)$ such that for every assignment of the free variables there is a branch T in $\mathcal{T}$ such that $(\mathcal{S}, \beta) \models T$ holds. $\square$

Theorem 3.2

If a tableau $\mathcal{T}$ is satisfiable, and $\mathcal{T}'$ is obtained from $\mathcal{T}$ by application of one of the above rules, then $\mathcal{T}'$ is also satisfiable. $\square$

Examples, Proof: to do in the lecture, sketch of two cases on Slide 117.

Issues: completeness of the method (only possible for decidable logics) and termination of the algorithm: how to detect when a tableau cannot be closed, and to restrict the expansion to promising rule applications.

Correctness of the FOL Tableau Calculus: Proof Sketch

Assume $\mathcal{T}$ satisfiable; $\mathcal{T}'$ obtained from applying a tableau rule. We show only two cases:

- Disjunction: Application of the rule to a formula of the form $A \vee B$. There is an interpretation $\mathcal{M}$ such that for each assignments β of free variables, there is some branch T (= the set of formulas on this branch) such that $\mathcal{M} \models_\beta T$. If T is not the branch of $\mathcal{T}$ that is extended in this step, T does not change. Otherwise, $M \models_\beta A \vee B$. By definition, $\mathcal{M} \models_\beta A$ or $\mathcal{M} \models_\beta B$. Thus, for (at least one) one of the two branches, T_1^* or T_2^* obtained from the application, $\mathcal{M} \models_\beta T^*$.

- Existential: Application of the rule to a formula of the form $\exists y : F(X_1, \ldots, X_n, y)$ to a branch T. Again, consider any β (which assigns $\beta(X_1), \ldots, \beta(X_n)$ to the free variables in F) such that $\mathcal{M} \models_\beta T$.

  This means, for every $\beta(X_1), \ldots, \beta(X_n)$, there is some element of the universe that "fits" for the existential formula. Extend the signature with a new n-ary "Skolem" function f_F that takes the values of $X_1, \ldots, X_n$ as input and is interpreted to return the appropriate element (and that returns an arbitrary value for those β' where $\mathcal{M} \not\models_{\beta'} T$).

  The extended branch T^* appends $F(X_1, \ldots, X_n, f_F(X_1, \ldots, X_n))$ to T.

  For the extended interpretation $\mathcal{M}'$ (which is the same as $\mathcal{M}$ except for the new function), $\mathcal{M}' \models_\beta T^*$ whenever $\mathcal{M} \models_\beta T$.

Tableau Calculus: Example

Proof of the derivation from Slide 66: Does

$(\forall y, d : (\mathsf{mg}(y,d) \rightarrow \mathsf{wf}(y,d))) \wedge (\forall x : \forall d : (\mathsf{wf}(x,d) \rightarrow (\mathsf{Emp}(x) \wedge \mathsf{Dept}(d)))) \wedge \mathsf{mg}(\mathsf{Alice}, \mathsf{Sales})$

imply $\mathsf{Emp}(\mathsf{Alice})$?

see next slide ...

Tableau Calculus: Example

$$\begin{tikzcd}[column sep=small]
 & \forall y, d : (\text{mg}(y,d) \rightarrow \text{wf}(y,d)) & & \\
 & \forall x : \forall d : (\text{wf}(x,d) \rightarrow (\text{Emp}(x) \wedge \text{Dept}(d))) & & \\
 & \text{mg}(\text{Alice}, \text{Sales}) & & \\
 & \neg\text{Emp}(\text{Alice}) \;\; (**) \arrow[d, dash] & & \\
 & \text{mg}(Y_1, D_1) \rightarrow \text{wf}(Y_1, D_1) \arrow[dl, dash] \arrow[dr, dash] & & \\
\neg\text{mg}(Y_1, D_1) & & \text{wf}(Y_1, D_1) & \\
\Box\{Y_1 \rightarrow \text{Alice}, D_1 \rightarrow \text{Sales}\} & & \text{wf}(\text{Alice}, \text{Sales}) \arrow[d, dash] & \\
 & & \text{wf}(X_2, D_2) \rightarrow (\text{Emp}(X_2) \wedge \text{Dept}(D_2))) \arrow[dl, dash] \arrow[dr, dash] & \\
 & \neg\text{wf}(X_2, D_2) & & \text{Emp}(X_2) \wedge \text{Dept}(D_2) \\
 & \Box\{X_2 \rightarrow \text{Alice}, D_2 \rightarrow \text{Sales}\} & & \text{Emp}(\text{Alice}) \wedge \text{Dept}(\text{Sales}) \\
 & & & \text{Emp}(\text{Alice}) \;\; (**) \\
 & & & \text{Dept}(\text{Sales}) \\
 & & & \Box(**)
\end{tikzcd}$$

Example: Tableau Expansion for an Existential Variable

Consider again the Company scenario. Show: for every employee x, there is an employee y ($x = y$ allowed) such that sub(x, y) holds. (sketch: for every employee x there is a at least a “primary” department $f_{dept}(x)$ where this person works, and every department d has a manager $f_{mg}(d)$ that manages the department and that thus is a subordinate of x.

Note that in case that x works in several departments, any of them can be chosen for $f_{dept}(x)$. e is subordinate to $f_{mg}(f_{dept}(x))$.

Tableau: next slide.

$\forall x$: (Emp$(x) \rightarrow \exists d$: wf(x, d))

$\forall d$: (Dept$(d) \rightarrow \exists m$: mg(m, d))

$\forall x$: $\forall d$: (wf$(x, d) \rightarrow$ (Emp$(x) \wedge$ Dept(d))) ,

$\forall x, y, d$: wf$(x, d) \wedge$ mg$(y, d) \rightarrow$ sub(x, y)

$\exists e$: Emp$(e) \wedge \neg\exists y$: sub(e, y) claim – refute it

|

Emp$(e_0) \wedge \neg\exists y$: sub(e_0, y) e_0 from Skolemization

Emp(e_0)

$\neg\exists y$: sub(e_0, y)

$\neg$sub(e_0, Y_0)

|

Emp$(X_1) \rightarrow \exists d$: wf(X_1, d))

/ \

$\neg$Emp(X_1) $\exists d$: wf(X_1, d))

close that only later ... wf$(X_1, f_{dept}(X_1))$

wf$(X_2, D_2) \rightarrow$ (Emp$(X_2) \wedge$ Dept(D_2)))

/ \

$\neg$wf(X_2, D_2) Emp$(X_2) \wedge$ Dept(D_2)

$\Box$ $\{X_2 \rightarrow X_1, D_2 \rightarrow f_{dept}(X_1)$ Emp(X_1)

Dept$(f_{dept}(X_1))$

/

Dept$(D_3) \rightarrow \exists m$: mg(m, D_3))

/ \

$\neg$Dept(D_3) $\exists m$: mg(m, D_3))

$\Box$ $\{D_3 \rightarrow f_{dept}(X_1)\}$ mg$(f_{mgr}(D_3), D_3)$

... and now replace D_3

/

mg$(f_{mgr}(f_{dept}(X_1)), f_{dept}(X_1))$

/

(wf$(X_4, D_4) \wedge$ mg(Y_4, D_4)) $\rightarrow$ sub(X_4, Y_4)

/ \

$\neg$(wf$(X_4, D_4) \wedge$ mg(Y_4, D_4)) sub(X_4, Y_4)

/ \ sub$(X_1, f_{mgr}(f_{dept}(X_1)))$

$\neg$wf(X_4, D_4) $\neg$mg(Y_4, D_4) |

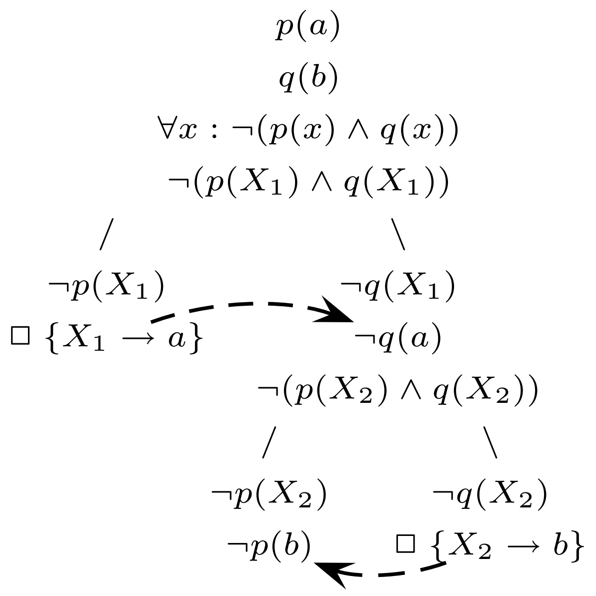
$\Box\{X_4 \rightarrow X_1,$ $\Box\{Y_4 \rightarrow f_{mgr}(f_{dept}(X_1)),$ close now everything by

$D_4 \rightarrow f_{dept}(X_1)\}$ $D_4 \rightarrow f_{dept}(X_1)\}$ $\{X_1 \rightarrow e_0,$

$Y_0 \rightarrow f_{mgr}(f_{dept}(e_0))\}$

A Non-Closed Tableau and Universal Quantification

Is the axiom $\forall x : \neg(p(x) \wedge q(x))$ together with the “database” $\{p(a),\ q(b)\}$ consistent?



- there is no way to close the tableau
- its non-closed path describes a model of the input formula (where $\neg q(a)$ and $\neg p(b)$ hold which are not specified in the database – open world reasoning)

Tableau Calculi: Application for Query Answering

Consider the database $\{\forall x : (p(x) \rightarrow q(x)),\ p(a), q(b)\}$ and the query $? - q(X)$.

$$\forall x : (p(x) \rightarrow q(x))$$

$$p(a)$$

$$q(b)$$

$\neg q(X)$ add the negated query with a free variable

- collect all substitutions of X that can be used to close the tableau.
- note: the substitution can comprise a the application of a Skolem function. Then, the "answer" can only be described as a thing that satisfied a certain existential formula.

  Consider $\forall x : (\text{person}(x) \rightarrow (\exists y : \text{person}(y) \wedge \text{father}(x,y)))$,
  $\forall x,y,z : ((\text{father}(x,y) \wedge \text{father}(y,z)) \rightarrow \text{grandfather}(x,z))$,
  person(john), person(jack), father(john,jack) and the query ?- grandfather(john,X).

TABLEAU CALCULI

- intuitive idea
- can be designed in this way for any logic (modal logics, description logics etc.)
- implementations use more efficient heuristics

EXAMPLES + EXERCISES

- Consider again the italian-vs-english ontology from Slide 109. Consider the statement “all Italians are lazy”. Prove it or give a counterexample.
- Consider again the italian-professors ontology from Slide 110. Is there anything interesting to prove?

[have also a look at the i•com tool at `http://www.inf.unibz.it/~franconi/icom` which uses a (hidden) Description Logic prover]

Tableau Proof (Example)

Tableau for the italian-vs-english ontology from Slide 109 and the statement “all Italians are lazy”.

$$\forall x : \text{italian}(x) \rightarrow \neg\text{english}(x)\ [1]$$

$$\forall x : \text{english}(x) \rightarrow \neg\text{italian}(x)\ [2]$$

$$\forall x : \text{italian}(x) \rightarrow (\text{lazy}(x) \vee \text{latinlover}(x))\ [3]$$

$$\forall x : \text{lazy}(x) \rightarrow \neg\text{latinlover}(x))\ [4]$$

$$\forall x : \text{latinlover}(x) \rightarrow \neg\text{lazy}(x)\ [5]$$

$$\forall x : \text{latinlover}(x) \rightarrow \text{gentleman}(x)\ [6]$$

$$\forall x : \text{gentleman}(x) \rightarrow \text{english}(x)\ [7]$$

$$\exists x : \text{italian}(x) \wedge \neg\text{lazy}(x)\ \ [8]\ \ \text{(negation of the claim)}$$

$$|\ \ \text{(skolemization of [8])}$$

$$\text{italian}(c) \wedge \neg\text{lazy}(c)$$

$$\text{italian}(c)$$

$$\neg\text{lazy}(c)$$

$$|\ \ \text{(use [3])}$$

$$\forall x : \text{italian}(x) \rightarrow (\text{lazy}(x) \vee \text{latinlover}(x))$$

$$\text{italian}(X_1) \rightarrow (\text{lazy}(X_1) \vee \text{latinlover}(X_1))$$

$$\begin{array}{cc}
/ & \backslash \\
\neg\text{italian}(X_1) & \text{lazy}(X_1) \vee \text{latinlover}(X_1) \\
\Box\ \{X_1 \rightarrow c\} & \text{lazy}(c) \vee \text{latinlover}(c) \\
 & \begin{array}{cc} / & \backslash \\ \text{lazy}(c) & \text{latinlover}(c) \\ \Box & \end{array}
\end{array}$$

Continue right branch using [6], [7] and finally [1] or [2].

First-Order Logic Decision Procedures

- calculi (=algorithms) for checking if $F \models G$
  (often by proving that $F \wedge \neg G$ is unsatisfiable)
- write $F \vdash_C G$ if calculus C proves that $F \models G$.
- Correctness of a calculus: $F \vdash_C G \Rightarrow F \models G$
- Completeness of a calculus: $F \models G \Rightarrow F \vdash_C G$
- there are complete calculi and proof procedures for propositional logic (e.g., Tableau Calculus or Model Checking)
- if a logic is undecidable (like first-order logic) then there cannot be any complete calculus!

What to do?

$\Rightarrow$ use a decidable logic (i.e., weaker than FOL).

$\Rightarrow$ use an undecidable logic and a correct, but incomplete calculus.

Inference Systems

- use *inference rules* (dt.: Schlussregeln) as patterns
- "Modus Ponens": $$\frac{\textit{head} \leftarrow \textit{body} \quad , \quad \textit{fml} \quad , \quad \sigma(\textit{fml}) \rightarrow \sigma(\textit{body})}{\sigma(\textit{head})}$$
- simple case: Datalog-style rules (many other systems use similar derivation rules)
$$\frac{\textit{head\_atom} \leftarrow \textit{atom}_1, \ldots, \textit{atom}_n \ \ , \text{ ground } \textit{atom'}_1, \ldots, \textit{atom'}_n \ \ , \text{ for all } i\text{: } \sigma(\textit{atom}_i) = \textit{atom'}_i}{\sigma(\textit{head\_atom})}$$
- Resolution Calculus:
a *clause* is a set of literals. Clause resolution takes two clauses that contain contradictory literals:
$$\frac{\ell_1 \vee \ldots \vee \boxed{\ell_i} \vee \ldots \vee \ell_k \quad , \quad \ell_{k+1} \vee \ldots \vee \boxed{\neg\ell_{k+j}} \vee \ldots \vee \ell_{k+m} \quad , \quad \boxed{\sigma(\ell_i) = \sigma(\ell_{k+j})}}{\sigma(\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee \ell_{k+1} \vee \ldots \vee \ell_{k+j-1} \vee \ell_{k+j+1} \vee \ldots \vee \ell_{k+m})}$$
- simple case: unit resolution: $j = m = 1$
- since a derivation rule *head*$\leftarrow$*body* is equivalent to $\neg$*body* $\vee$ *head*, the bottom-up evaluation of derivation rules is a special case of resolution.

Specialized Inference Rules

- Modus ponens and resolution are purely syntactical, general-purpose rules that do not depend on the semantics of the literals.
- Logics can extend them with specialized semantical rules that apply to special “predicates”.
  - class hierarchy: classes and subclasses, transitivity
  - signatures
  - cardinalities

  Examples: Default Logic (Reiter, 1980; only built-in reasoning), Description Logics (late 1980s; only built-in reasoning), F-Logic (Kifer, Lausen, 1989; Datalog style + built-in), Hybrid Logics (combining Description Logics with derivation)
- the latter *built-in axioms* for certain fixed notions are referred to as “the model theory of a certain logic”.

Summary

Above: short introduction to some Knowledge Representation formalisms:

- Derivation Rules
- Automated Reasoning

Semantic Web Reasoning

- use mechanisms of logics and “Artificial Intelligence” invented in the 60s-90s
- disappointment about AI in the 90s:
  - promised too much (expert systems etc.)
  - often worked only for toy examples
- further investigations in the 90s
  - better understanding of decidable and tractable fragments (“tractable” = polynomial complexity) and efficiency issues
- design Semantic Web technology according to these investigations.

Levels of Inference

In the following, three levels of inference and knowledge modeling are combined:

1. The underlying inference system:
   choice of certain underlying *expressive logics* with their built-in model theory
   - First-order logic: quite expressive, no built-in model theory
   - Description logics:
     - less expressive (only unary and binary predicates, quite restricted construction of formulas)
     - some built-in notions + model theory
     - user can exploit these notions
2. a domain ontology (= formulas in the underlying logic that express global properties and constraints of the domain)
   [mainly describing the classes and properties; optionally some “important” individuals]
3. facts that describe a certain state
   [the “Web contents”, talking about individuals; incomplete knowledge]

Data Format and Reasoning for the Semantic Web

- data model: intuitive modeling capabilities on the conceptual level
- describe data and metadata
  $\Rightarrow$ metadata notions are also objects of the domain of discourse.
- tailored to the Web: multiple sources describe the same things, semantic interoperability
- data format/representation: syntactic interoperability/data exchange in the Web required.
  $\Rightarrow$ ASCII, preferably an XML representation (then, no special parser is needed).
  note: for XML, the ASCII was a serialization/representation of the XML tree model
  for the new data model an XML-tree representation will be one possible representation of the data model.
- reasoning: decidable fragment of FOL vs. undecidable FOL vs. even more expressiveness (reasoning about metadata)

Aside: Why "First-Order"-Logic?

Recall:

- there is a domain $\mathcal{D}$. Functions and precidates talk *about* elements of $\mathcal{D}$.
- there is no way to talk *about* functions or predicates.

Higher-Order-Logics

- the elements of the domain $\mathcal{D}$ are "first-order things"
- sets, functions and predicates are "second-order things"
- predicates about predicates are higher-order things
- higher-order logics can be used for reasoning *about* metadata

Example

- Transitivity as a property of predicates is second order:
  $\forall p : \text{transitive}(p) \rightarrow (\forall x, z : (\exists y : (p(x, y) \wedge p(y, z)) \rightarrow p(x, z)))$
  Note that transitivity of *a certain* predicate is first-order:
  $\forall x, z : ((\exists y : (\text{ancestor}(x, y) \wedge \text{ancestor}(y, z))) \rightarrow \text{ancestor}(x, z))$

Aside: Induction Axiom as Example for Second Order Logic

- a well-founded domain d (i.e., a finite set of minimal elements (for which min(d,x) holds) from which the domain can be enumerated by a successor predicate (Natural numbers: 1, succ(i,i+1))
- well-founded: unary 2nd-order predicate over sets

$$\forall p, d: \quad (\mathsf{well\text{-}founded}(d) \wedge (\forall x : \mathsf{min}(d, x) \rightarrow p(x)) \wedge (\forall x, y : p(x) \wedge \mathsf{succ}(x, y) \rightarrow p(y))) \rightarrow \\ (\forall x : d(x) \rightarrow p(x))$$

For natural numbers:

$$\forall p : (p(1) \wedge (\forall x : p(x) \rightarrow p(x + 1))) \rightarrow (\forall x \in \mathbb{N} : p(x))$$

Aside: Paradoxes can be formulated in 2nd Order Logic

"X is the set of all sets that do not contain themselves"

$$X = \{z : z \notin z\}$$

A set "is" a unary predicate: $X(z)$ holds if z is an element of X
(for example, classes, i.e., Person(x), City(x))

Logical characterization of X: $\quad X(z) \leftrightarrow \neg X(z)$,

applied to X: $\quad X(X) \leftrightarrow \neg X(X)$.

... can neither be true nor false.

How to avoid paradoxes

Paradoxes can be avoided if each variable *either* ranges over first-order things (elements of the domain) or over second-order things (predicates).