Chapter 3
Databases vs. Knowledge Bases

Overview

- basic notions of first-order logic: syntax, semantics, model theory, tableau – see slides for Database Theory Lecture
- Recall: the Relational Model is a specialization of First-Order Logic:
  - no function symbols. Only constant literals.

Query Answering in DB vs. KB

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

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

  We are interested whether 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 given by a *model theory*:

  Deductive Databases/Datalog: minimal model, stratified model, well-founded model, stable models (with specific algorithms)

  First-Order Logic: all models ("traditional" reasoning)
- 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?

LOGICAL MODELING OF ONTOLOGIES

- common "recipe" for mapping ER diagrams to the relational model (cf. MONDIAL):
  - n-ary entity tables with primary keys,
  - n-ary tables for (mostly binary) n:m-relationships with foreign keys,
  - integrate n:1 relationships with the n side,
  - modeling subclass relationships is not a primary concern of the "recipe".
- Alternative (storage) variants, traditionally rather seen as physical data models:
  - column stores,
  - "vertically partitioned": two-column tables with format (id,value) for each attribute or binary relationship; introducing "object" ids.
    - saves space if null values are frequent
    - lots of joins,
    - smaller tuples,
    - less projections
    - better suited for Web context where properties of the same object can be stored at different locations
    - coincides with the modeling in Description Logics

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

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

Example: Domain and Range Constraints

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

Example: Cardinality Constraints

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

Example: Further Constraints

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

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

- 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"

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

  - this causes a semantical problem with the boss: an infinite chain is needed - leading either to only infinite models, or a cycle.
  - add axioms that guarantee transitivity and irreflexivity for "subordinate":
    $\forall x : \neg\mathsf{sub}(x, x)$ and $\forall x, y, z : (\mathsf{sub}(x, y) \wedge \mathsf{sub}(y, z) \rightarrow \mathsf{sub}(x, z))$.
    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 $\mathsf{Axioms}_{Company}$.

- For any database/knowledge base $\mathcal{S}$ using this scenario, $\mathcal{S} \models \mathsf{Axioms}_{Company}$ is required.
- a database then describes 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:

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

  (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 $\mathcal{D}$:
  $I(\text{Emp}) = \{$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:

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

Example: Instances (Alternative)

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

$\text{Axioms}_{Company} \wedge$
$\exists a, b, \ldots, s, \ldots : (\text{name}(a, \text{"Alice"}) \wedge \ldots \wedge \text{Emp}(a) \wedge \text{Emp}(b) \wedge \ldots \wedge \text{Dept}(s) \wedge \ldots \wedge \text{Mgr}(a) \wedge \ldots \wedge \text{wf}(a, s) \wedge \ldots \wedge \text{mg}(a, s) \wedge \ldots \wedge \text{sub}(m, a) \wedge \ldots)$

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

Consider the following formula F:

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

- 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 $\mathbf{D}$, $\mathbf{D} \models \neg sibling(\text{“Alice”}, \text{“Carol”})$.
  - 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 ($\forall$) 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 $\models \phi$?

VALIDITY AND DECIDABILITY

- preferably use a decidable logic/formalism
- with a *complete* calculus/reasoning mechanism
- Propositional logic: decidable
- First-order logic: undecidable
- Horn subset of FOL
  (logical rules of the form $head \leftarrow body$ over literals); decidable
- 2-variable-subset of FOL: decidable
- Description Logic subsets of FOL: range from decidable to undecidable

3.2 Reasoning about Ontologies

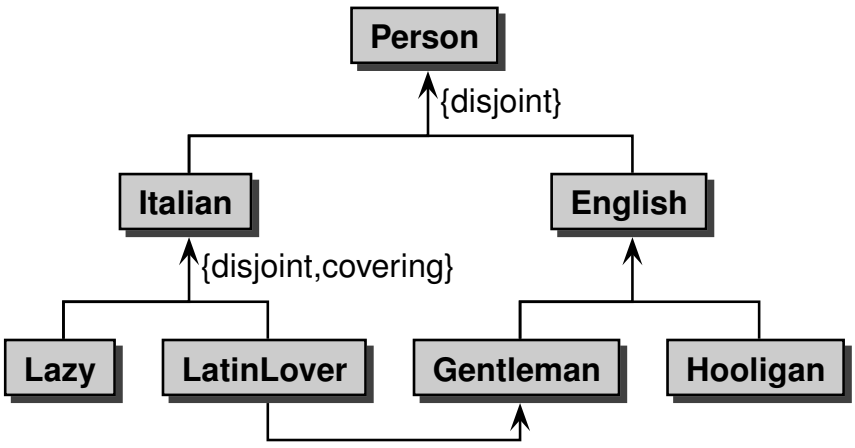
Traditional Description Logics Research

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.

$\Rightarrow$ focus on metadata ("T-Box"); query answering wrt. instances ("A-Box") did not play a big role.

EXAMPLE: ITALIANS AND ENGLISHMEN



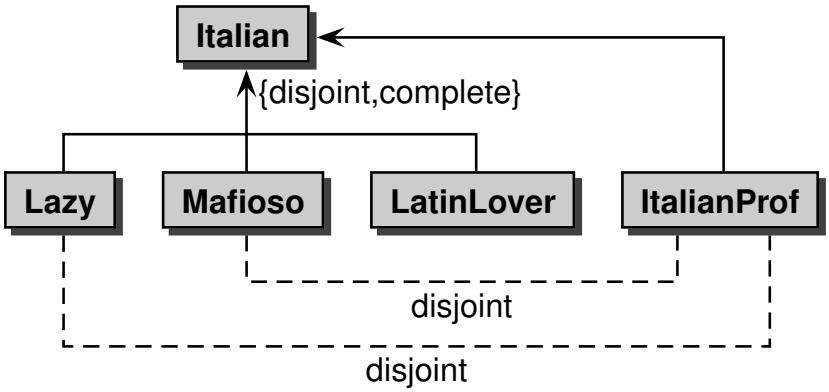
Exercise: write down as concise as possible *everything* that is implied by this ontology in text, set theory and first-order logic.

[by Enrico Franconi, REWERSE Summer School 2005]

[see Slide 71 for an excerpt and a relevant proof]

.

EXAMPLE: ITALIAN PROFESSORS

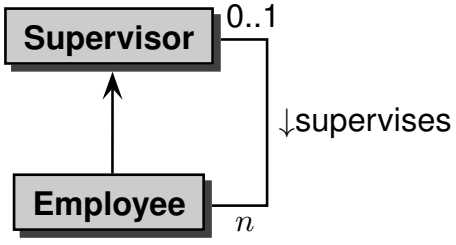


Exercise: write down as concise as possible *everything* that is implied by this ontology in text, set theory and first-order logic.

[by Enrico Franconi, REWERSE Summer School 2005]

EXAMPLE: THE DEMOCRATIC COMPANY

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

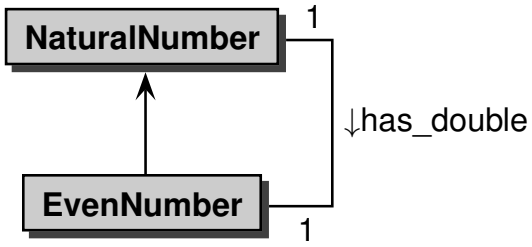


Exercise: Consider $n = 1$ and $n = 2$. Write down as concise as possible *everything* that is implied by this ontology in text, set theory and first-order logic.

[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 elements (standard model: infinitely many),
- 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]

3.3 Reasoning

- 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.

RECALL: 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 a set $\mathcal{F}$ of formulas:

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

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-rule (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:

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

apply σ to the whole tableau.

Result

Definition 3.1
A branch T in a tableau $\mathcal{T}$ is *closed*, if it contains the formula $\bot$.
A tableau $\mathcal{T}$ is *closed* if every branch is closed. □

Correctness

Definition 3.2
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. □

Theorem 3.1
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. □

TABLEAU CALCULUS: EXAMPLE/EXERCISE

Consider $\text{Axioms}_{Company} \wedge \text{mg}(\text{Alice}, \text{Sales})$. (Slides 38 ff.).

Does

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

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

see next slide ...

Tableau Calculus: Example

$$\forall y, d : (\mathsf{mg}(y, d) \rightarrow \mathsf{wf}(y, d))$$
$$\forall x : \forall d : (\mathsf{wf}(x, d) \rightarrow (\mathsf{Emp}(x) \wedge \mathsf{Dept}(d)))$$
$$\mathsf{mg}(\mathsf{Alice}, \mathsf{Sales})$$
$$\neg\mathsf{Emp}(\mathsf{Alice}) \quad (**)$$
$$|$$
$$\mathsf{mg}(Y_1, D_1) \rightarrow \mathsf{wf}(Y_1, D_1)$$

/ \

$\neg\mathsf{mg}(Y_1, D_1)$ | $\mathsf{wf}(Y_1, D_1)$

$\square\{Y_1 \rightarrow \mathsf{Alice}, D_1 \rightarrow \mathsf{Sales}\}$ | $\mathsf{wf}(\mathsf{Alice}, \mathsf{Sales})$

|

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

/ \

$\neg\mathsf{wf}(X_2, D_2)$ | $\mathsf{Emp}(X_2) \wedge \mathsf{Dept}(D_2)$

$\square\{X_2 \rightarrow \mathsf{Alice}, D_2 \rightarrow \mathsf{Sales}\}$ | $\mathsf{Emp}(\mathsf{Alice}) \wedge \mathsf{Dept}(\mathsf{Sales})$

$\mathsf{Emp}(\mathsf{Alice}) \quad (**)$

$\mathsf{Dept}(\mathsf{Sales})$

$\square(**)$

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 $\mathsf{sub}(x, y)$ 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.

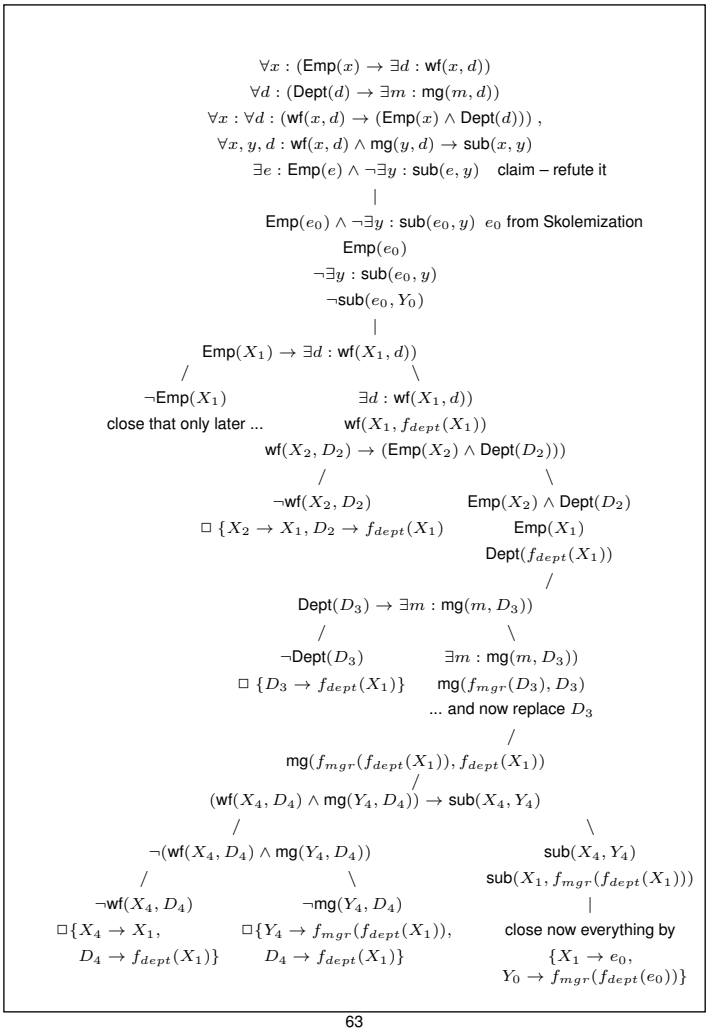


TABLEAU CALCULI: APPLICATION FOR (DATABASE) QUERY ANSWERING

Consider two binary predicates p and q (in a deductive DB context called "EDB" (extensional database)) and a derived unary predicate r as $\forall x : (r(x) \leftrightarrow \exists y : (p(x,y) \vee q(x,y)))$, and the query $?- r(Z)$.

$\forall x : (r(x) \leftrightarrow \exists y : (p(x,y) \vee q(x,y)))$

$\neg r(Z)$ add the negated query with a free variable

$\forall x : (r(x) \leftarrow \exists y : (p(x,y) \vee q(x,y)))$ need only "$\leftarrow$" direction to close

$\forall x : (r(x) \to \exists y : (p(x,y) \vee q(x,y)))$ not needed

$r(X) \leftarrow \exists y : (p(X,y) \vee q(X,y))$

$r(X)$ $\square\{Z \to X\}$

$\neg\exists y : (p(X,y) \vee q(X,y))$

$\neg p(X,Y)$ can be closed with any tuple $p(x,y)$

$\neg q(X,Y)$ can be closed with any tuple $q(x,y)$

- collect all substitutions that can be used to close the tableau:
  $\rho[X \to Z](\pi[\$1](p) \cup \pi[\$1](q))$ (where $\$n$ denotes the nth column)

Tableau Calculi for (Database) Query Answering (same example, other way)

Same as before, only that (the same r) is now expressed in Datalog style by two rules:

- different ways how to close the tableau $\Rightarrow$ union of answer substitutions

$$\forall x, y : (p(x, y) \rightarrow r(x))$$
$$\forall x, y : (q(x, y) \rightarrow r(x))$$
$$\neg r(Z) \quad \text{add the negated query with a free variable}$$
$$p(X_1, Y_1) \rightarrow r(X_1)$$
$$q(X_2, Y_2) \rightarrow r(X_2)$$

two possibilities which rule to expand:

the first one with p ...

$\neg p(X_1, Y_1)$ can be closed with any tuple $p(x, y)$ | $r(X_1)$ $\square\{Z \rightarrow X_1\}$

... or the second one with q

$\neg q(X_2, Y_2)$ can be closed with any tuple $q(x, y)$ | $r(X_2)$ $\square\{Z \rightarrow X_2\}$

- again, collect all substitutions that can be used to close the tableau:
  $\rho[X \rightarrow Z](\pi[\$1](p)) \cup \rho[X \rightarrow Z](\pi[\$1](q))$ (again, $\$n$ denotes the nth column)

Example: Existential Knowledge

- the answer substitution can comprise the application of a Skolem function. Then, the "answer" can only be described as a thing that satisfies a certain existential formula.

Example: Tableau with Existential Knowledge

$\forall x : (\text{person}(x) \rightarrow (\exists y : \text{person}(y) \wedge \text{father}(x, y)))$

$\forall x, y, z : ((\text{father}(x, y) \wedge \text{father}(y, z)) \rightarrow \text{grandfather}(x, z))$

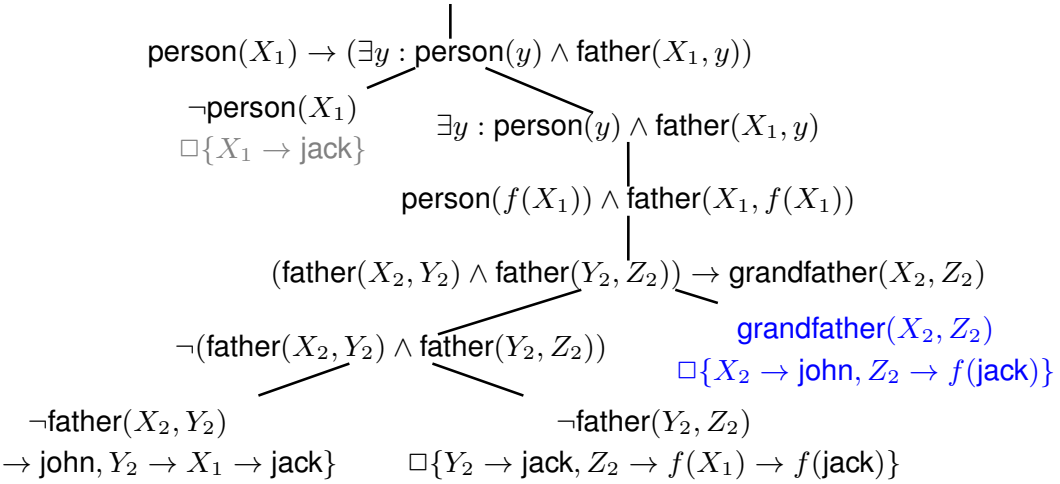
person(john)

person(jack)

father(john, jack)

Negated query: $\neg$ grandfather(john, X)

Human-reasoning-inspired proof: Jack's father is John's grandfather



close all with

$\{X_1 \rightarrow \text{jack},\ X_2 \rightarrow \text{john},\ Y_2 \rightarrow X_1 \rightarrow \text{jack},\ Z_2 \rightarrow f(X_1) \rightarrow f(\text{jack}),\ X \rightarrow Z_2 \rightarrow f(X_1) \rightarrow f(\text{jack})\}$.

Example: Existential Knowledge

- the answer contains an instance of the introduced skolem function $f(\_)$ (which stands for "father of _") to describe a person for which no constant identifier is known.
- Are there further answers? - yes:
  - instantiate the rule
    $\forall x : (\text{person}(x) \rightarrow (\exists y : \text{person}(y) \wedge \text{father}(x, y)))$
    for John, get a new instance of person $f(\text{john})$, instantiate the rule again for $f(\text{john})$ and get $f(f(\text{john}))$ and grandfather(john, $f(f(\text{john}))$).
  - here, functionality of father is required:
    $\forall x, y_1, y_2 : \text{father}(x, y_1) \wedge \text{father}(x, y_2) \rightarrow y_1 = y_2$.
  - equality must then be used to find a most simple expression for the answers.
- more general case: query ?- grandfather(X, Y) for all answers:
  - instantiations of the "father-generating" rule lead to infinitely many father-chains for every known instance of person (using the skolem function)
  - infinitely many answers.

$\Rightarrow$ tableau expansion for only existentially known instances must be blocked when nothing "relevant" is known about them.

$\Rightarrow$ Preview: SPARQL will *not* return any answers that contain skolem expressions.

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 52. Consider the statement “all Italians are lazy”. Prove it or give a counterexample.
- Consider again the italian-professors ontology from Slide 53. 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 52 and the statement “all Italians are lazy”.

$\forall x$: italian$(x) \rightarrow \neg$english(x) [1]
$\forall x$: english$(x) \rightarrow \neg$italian(x) [2]
$\forall x$: italian$(x) \rightarrow$ (lazy$(x) \vee$ latinlover$(x))$ [3]
$\forall x$: lazy$(x) \rightarrow \neg$latinlover$(x))$ [4]
$\forall x$: latinlover$(x) \rightarrow \neg$lazy(x) [5]
$\forall x$: latinlover$(x) \rightarrow$ gentleman(x) [6]
$\forall x$: gentleman$(x) \rightarrow$ english(x) [7]
$\exists x$: italian$(x) \wedge \neg$lazy(x) [8] (negation of the claim)
| (skolemization of [8])
italian$(c) \wedge \neg$lazy(c)
italian(c)
$\neg$lazy(c)
| (use [3])
$\forall x$: italian$(x) \rightarrow$ (lazy$(x) \vee$ latinlover$(x))$
italian$(X_1) \rightarrow$ (lazy$(X_1) \vee$ latinlover$(X_1))$
/ \
$\neg$italian(X_1) lazy$(X_1) \vee$ latinlover(X_1)
$\square$ $\{X_1 \rightarrow c\}$ lazy$(c) \vee$ latinlover(c)
/ \
lazy(c) latinlover(c)
$\square$

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, generic 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.
  $\Rightarrow$ Unicode, preferably an XML representation (then, no special parser is needed).
  (cf. Unicode 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

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

For natural numbers:

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

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).