

Description Logics for Conceptual Design, Information Access, and Ontology Integration

PhD course, Copenhagen 2002

Enrico Franconi

franconi@inf.unibz.it

<http://www.inf.unibz.it/~franconi>

Faculty of Computer Science, Free University of Bozen-Bolzano

Summary

- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- Queries with an Ontology

Summary

- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- Queries with an Ontology

What is an Ontology

- An ontology is a formal conceptualisation of the world: a [conceptual schema](#).

What is an Ontology

- An ontology is a formal conceptualisation of the world: a **conceptual schema**.
- An ontology specifies a set of **constraints**, which declare what should necessarily hold in any possible world.

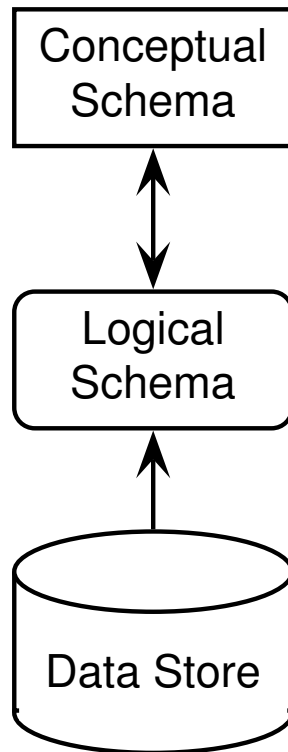
What is an Ontology

- An ontology is a formal conceptualisation of the world: a **conceptual schema**.
- An ontology specifies a set of **constraints**, which declare what should necessarily hold in any possible world.
- Any possible world should conform to the constraints expressed by the ontology.

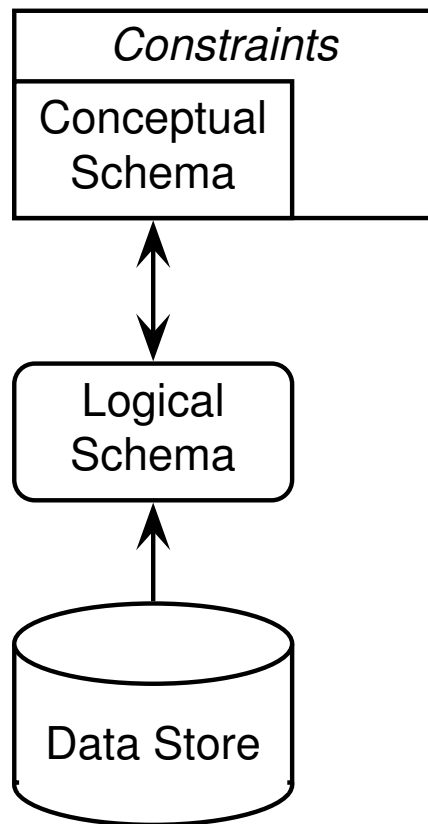
What is an Ontology

- An ontology is a formal conceptualisation of the world: a **conceptual schema**.
- An ontology specifies a set of **constraints**, which declare what should necessarily hold in any possible world.
- Any possible world should conform to the constraints expressed by the ontology.
- Given an ontology, a ***legal world description*** is a finite possible world satisfying the constraints.

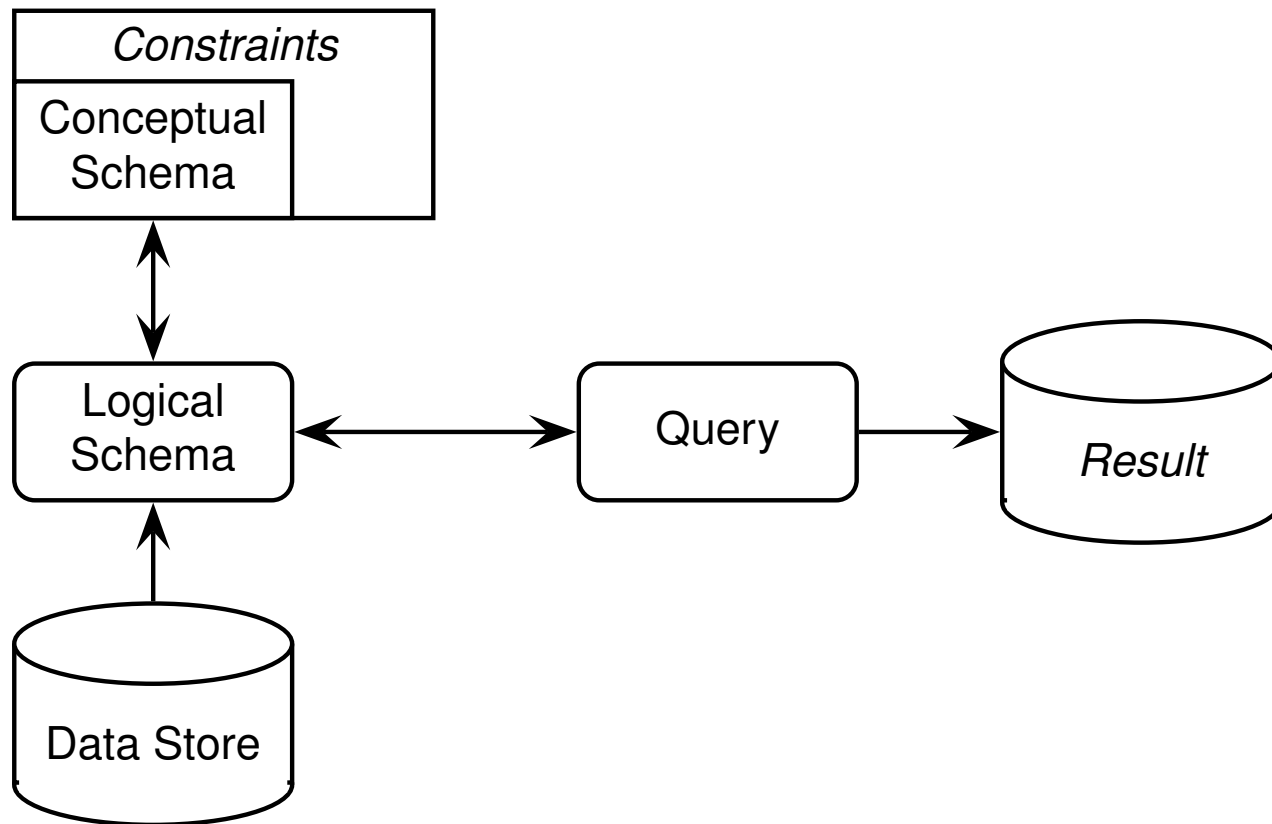
The role of a Conceptual Schema



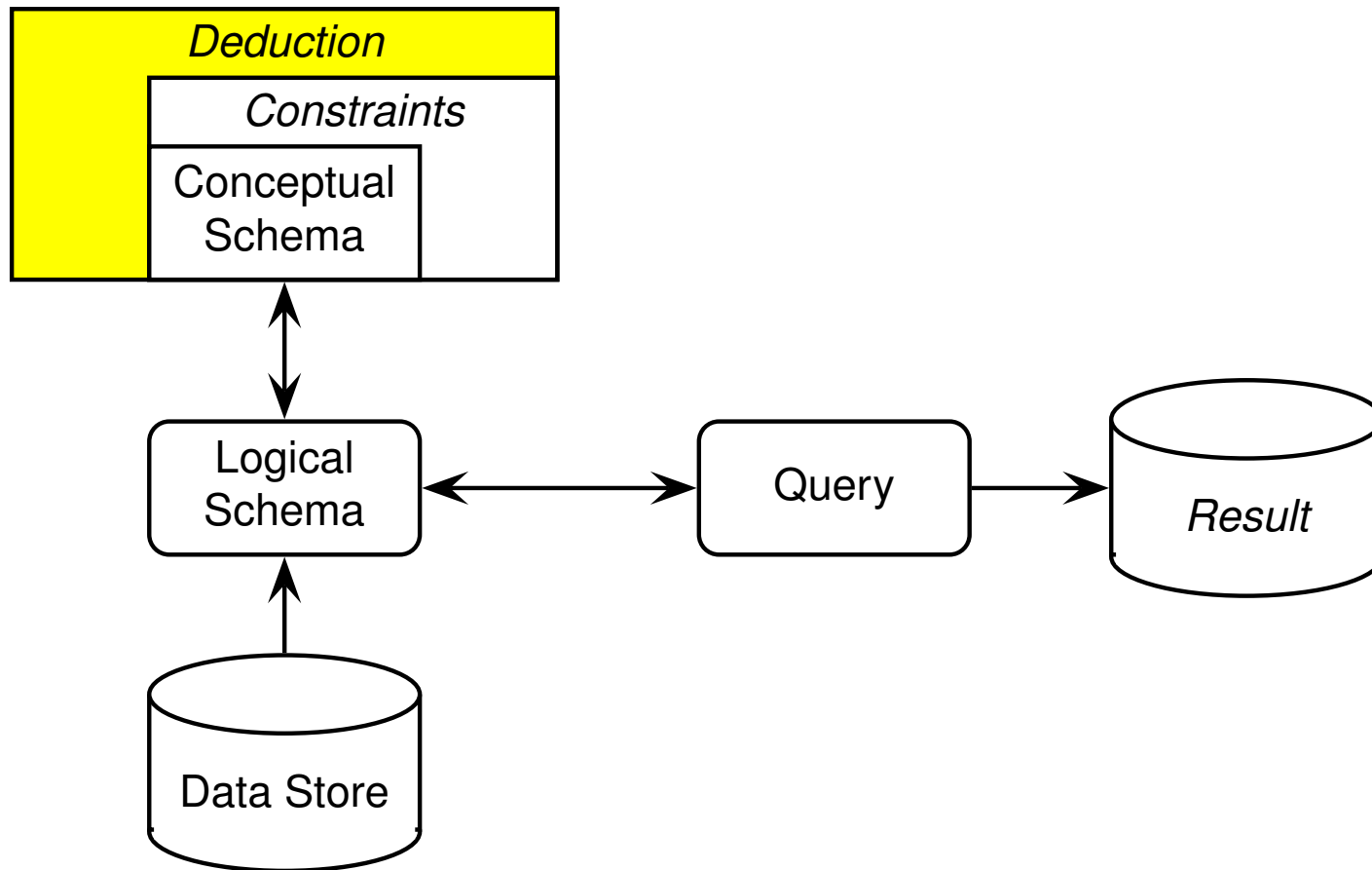
The role of a Conceptual Schema



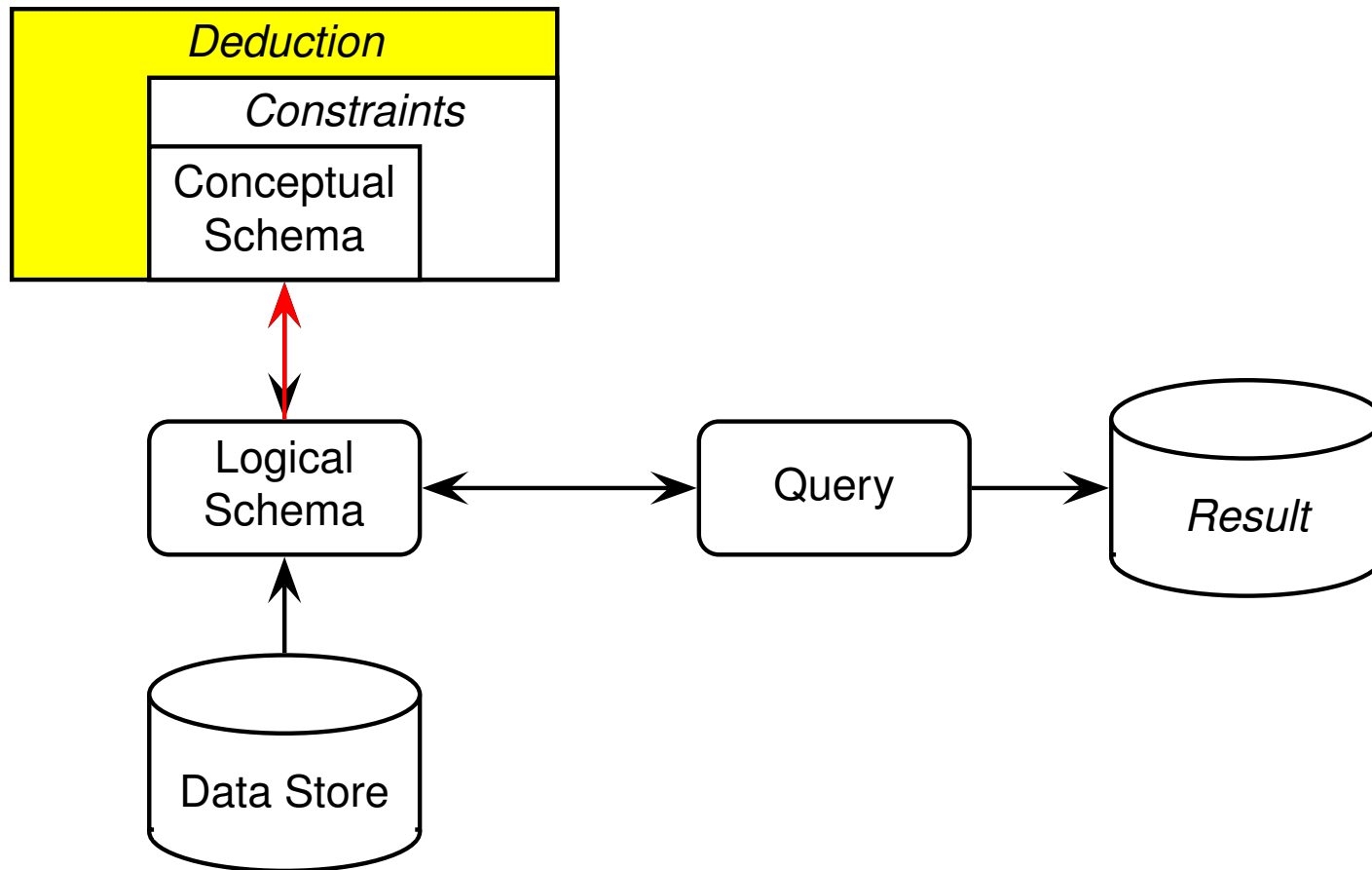
The role of a Conceptual Schema



The role of a Conceptual Schema



The role of a Conceptual Schema



Ontology languages and Conceptual Data Models

- An ontology language usually introduces **concepts** (aka classes, entities), **properties** of concepts (aka slots, attributes, roles), **relationships** between concepts (aka associations), and additional **constraints**.

Ontology languages and Conceptual Data Models

- An ontology language usually introduces **concepts** (aka classes, entities), **properties** of concepts (aka slots, attributes, roles), **relationships** between concepts (aka associations), and additional **constraints**.
- Ontology languages may be simple (e.g., having only concepts and taxonomies), frame-based (having only concepts and properties), or logic-based (e.g. Ontolingua and DAML+OIL).

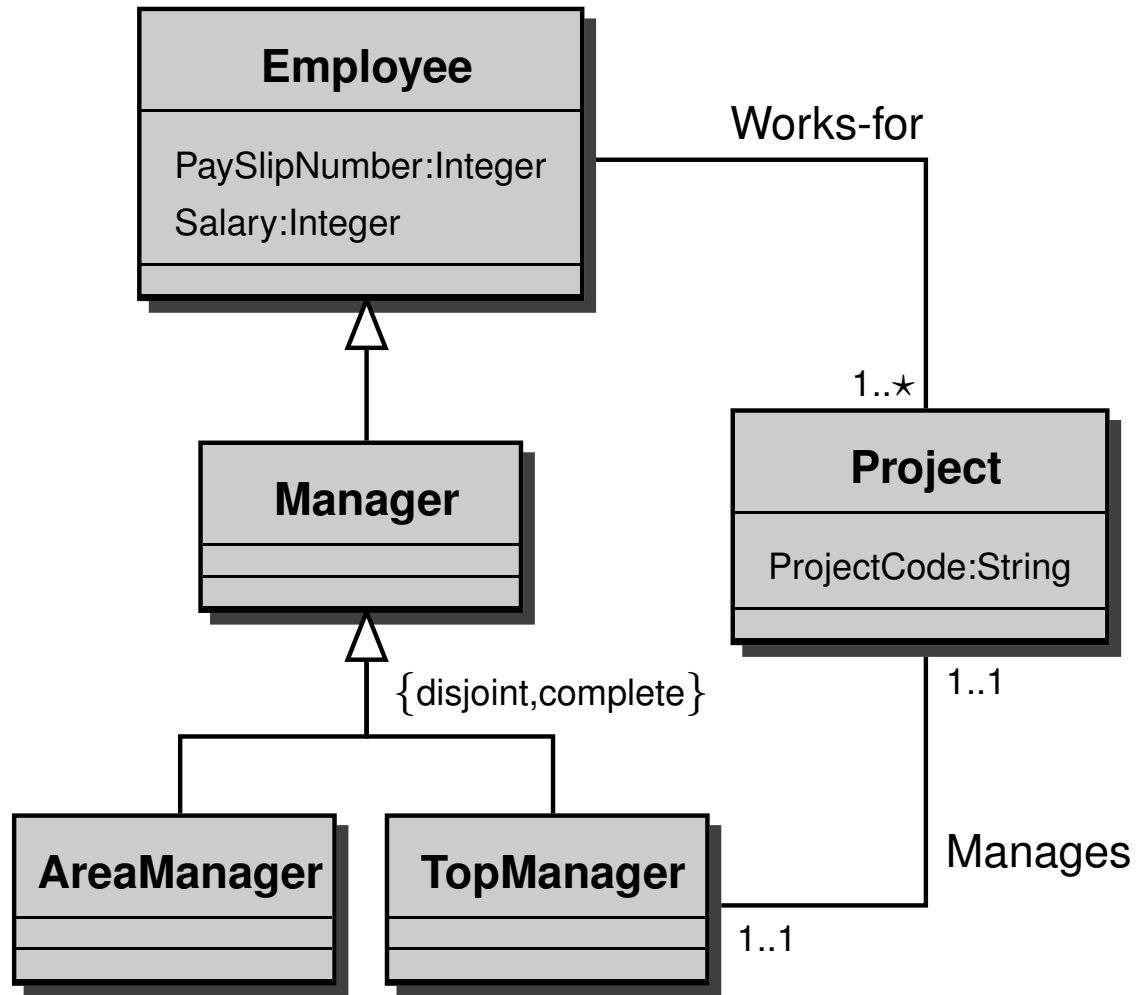
Ontology languages and Conceptual Data Models

- An ontology language usually introduces **concepts** (aka classes, entities), **properties** of concepts (aka slots, attributes, roles), **relationships** between concepts (aka associations), and additional **constraints**.
- Ontology languages may be simple (e.g., having only concepts and taxonomies), frame-based (having only concepts and properties), or logic-based (e.g. Ontolingua and DAML+OIL).
- Ontology languages are typically expressed by means of diagrams.

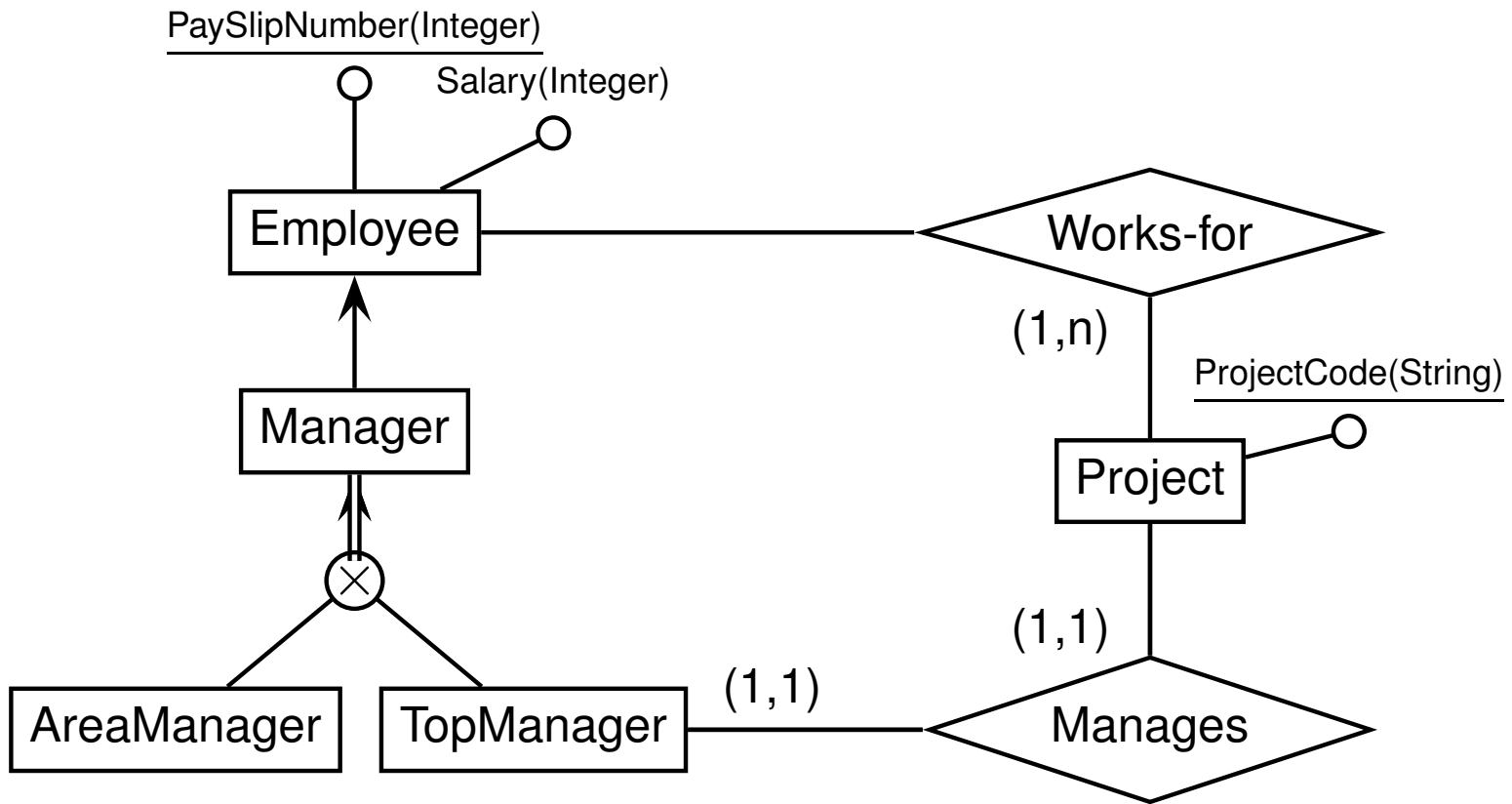
Ontology languages and Conceptual Data Models

- An ontology language usually introduces **concepts** (aka classes, entities), **properties** of concepts (aka slots, attributes, roles), **relationships** between concepts (aka associations), and additional **constraints**.
- Ontology languages may be simple (e.g., having only concepts and taxonomies), frame-based (having only concepts and properties), or logic-based (e.g. Ontolingua and DAML+OIL).
- Ontology languages are typically expressed by means of diagrams.
- **Entity-Relationship** schemas and **UML** class diagrams can be considered as ontologies.

UML Class Diagram



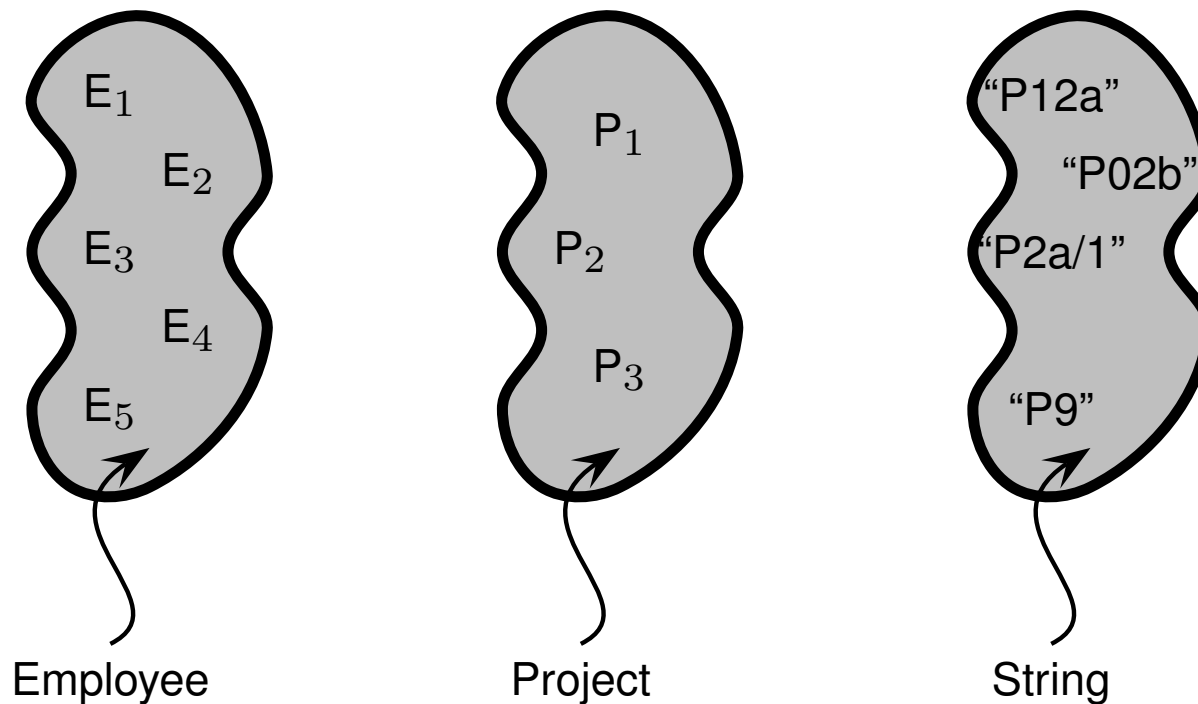
Entity-Relationship Schema



Semantics

In a specific world:

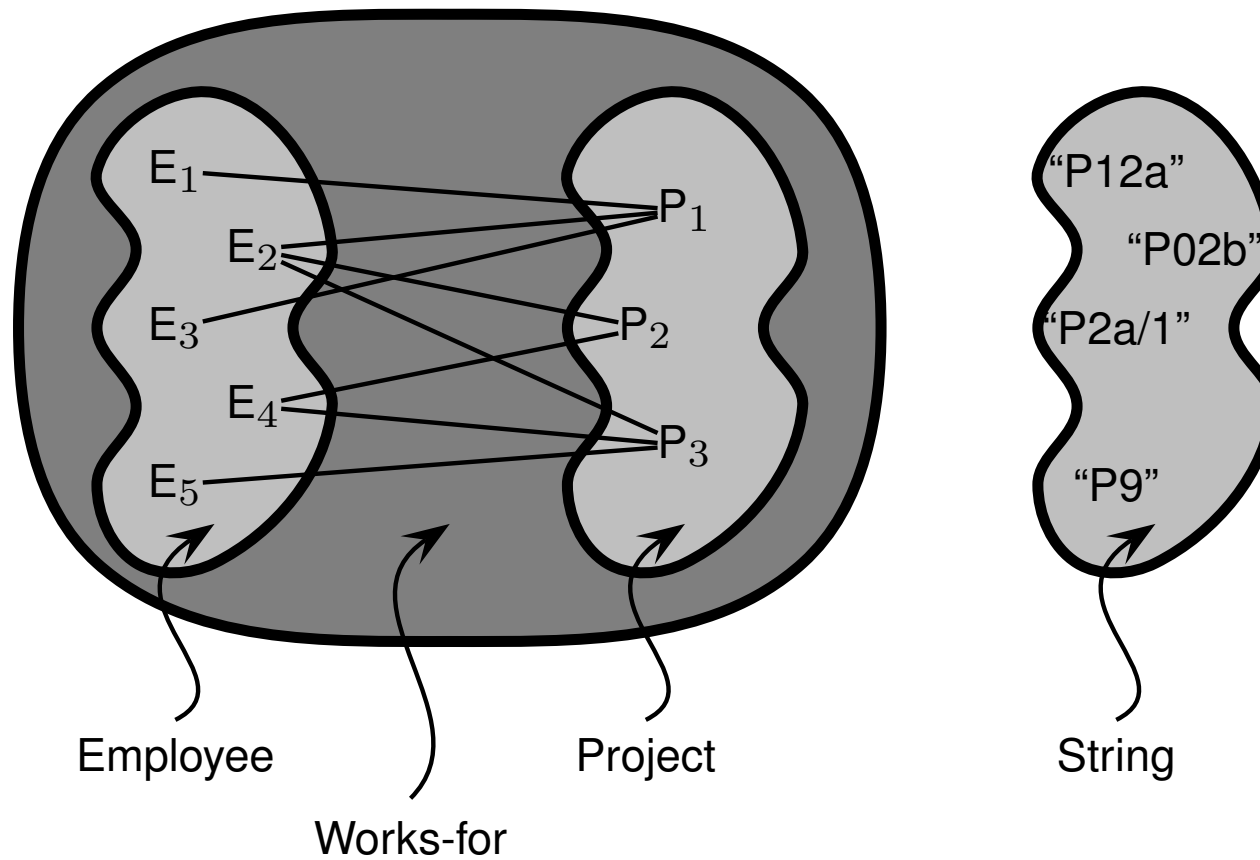
- A class is a ***set of instances***;
- a n-ary relationship is a ***set of n-tuples of instances***;
- an attribute is a ***set of pairs of an instance and a domain element***.



Semantics

In a specific world:

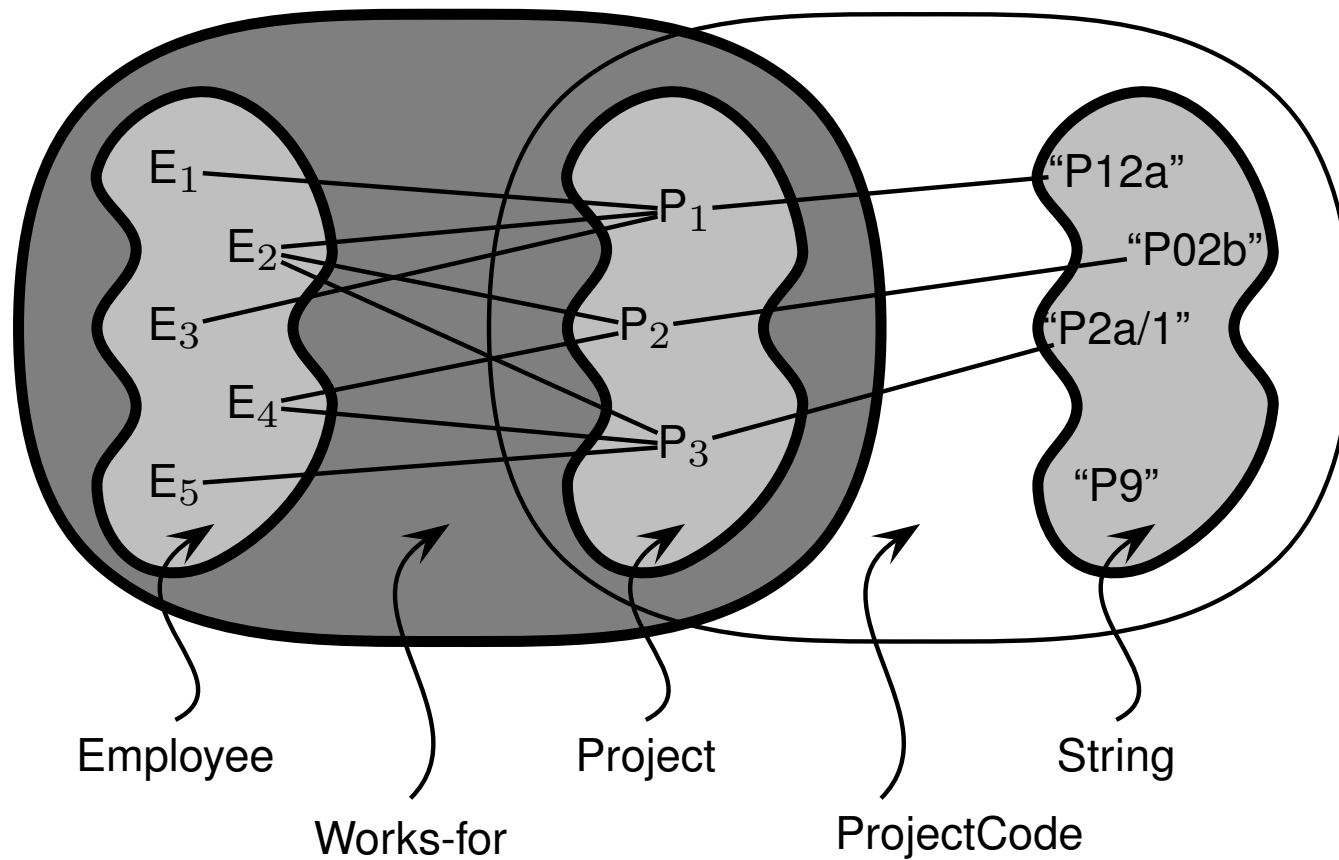
- A class is a **set of instances**;
- a n-ary relationship is a **set of n-tuples of instances**;
- an attribute is a **set of pairs of an instance and a domain element**.



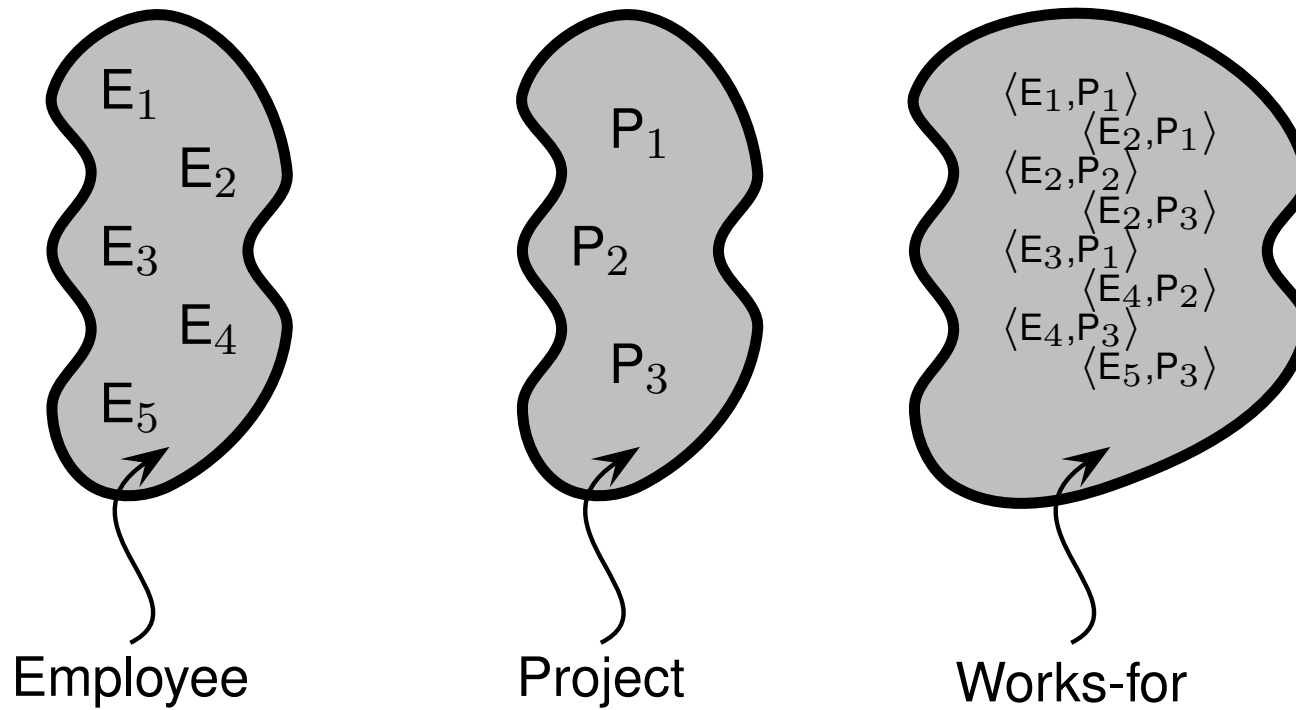
Semantics

In a specific world:

- A class is a **set of instances**;
- a n-ary relationship is a **set of n-tuples of instances**;
- an attribute is a **set of pairs of an instance and a domain element**.



A world is described by sets of instances



The relational representation of a world

Employee

<i>employeeId</i>
E ₁
E ₂
E ₃
E ₄
E ₅

Project

<i>projectId</i>
P ₁
P ₂
P ₃

String

<i>anystring</i>
"P12a"
"P02b"
"P2a/1"
"P9"
...

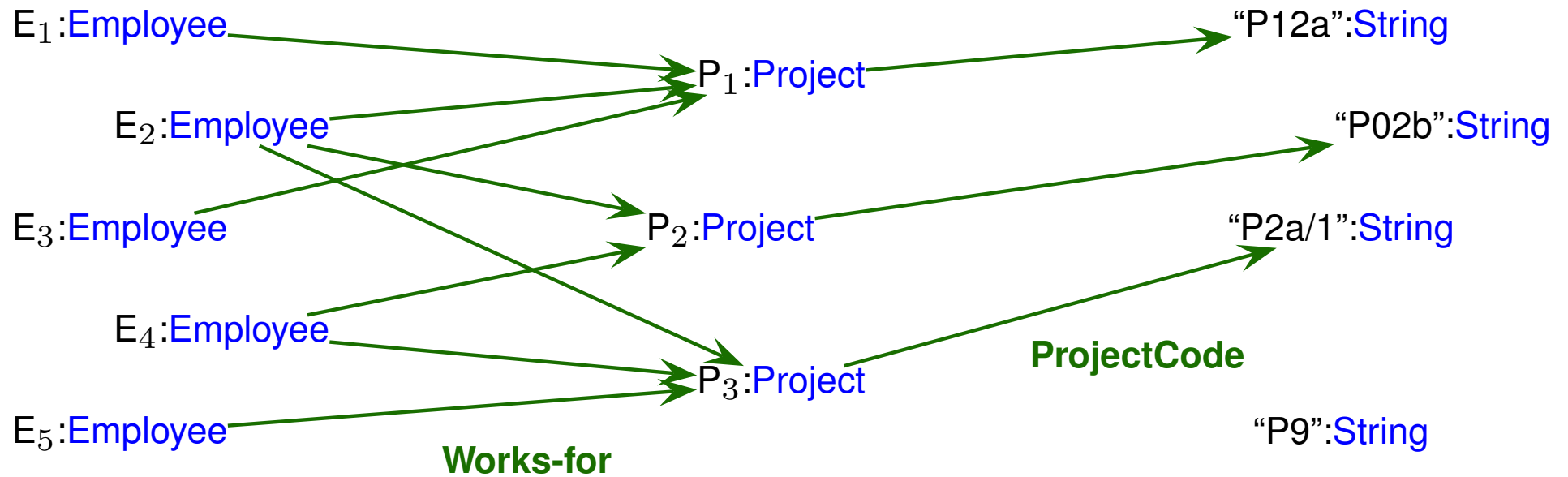
Works-for

<i>employeeId</i>	<i>projectId</i>
E ₁	P ₁
E ₂	P ₁
E ₂	P ₂
E ₂	P ₃
E ₃	P ₁
E ₄	P ₂
E ₄	P ₃
E ₅	P ₃

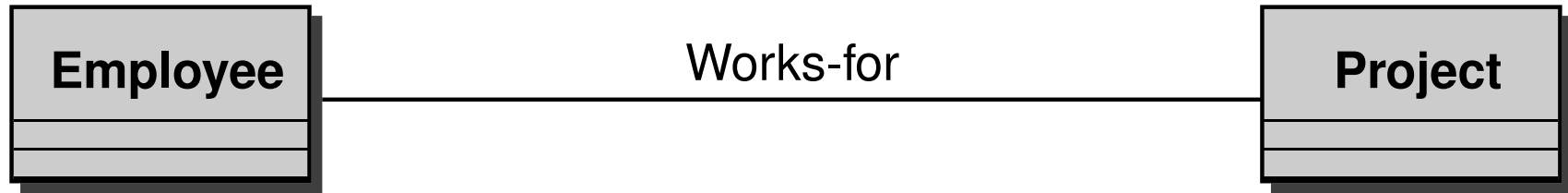
ProjectCode

<i>projectId</i>	<i>pcode</i>
P ₁	"P12a"
P ₂	"P02b"
P ₃	"P2a/1"

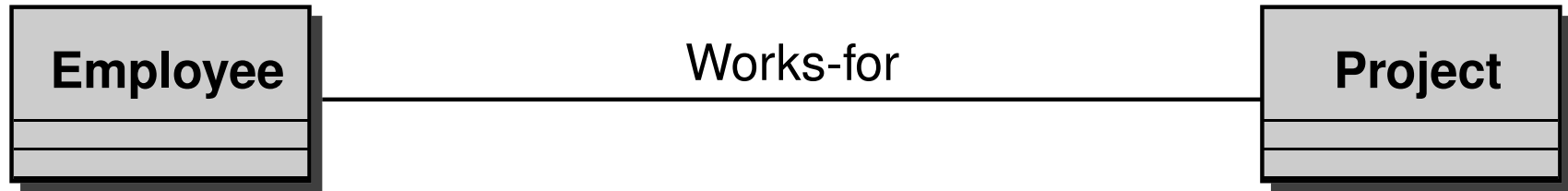
The graph representation of a world – e.g. RDF triples



Constraints introduced by Relationships

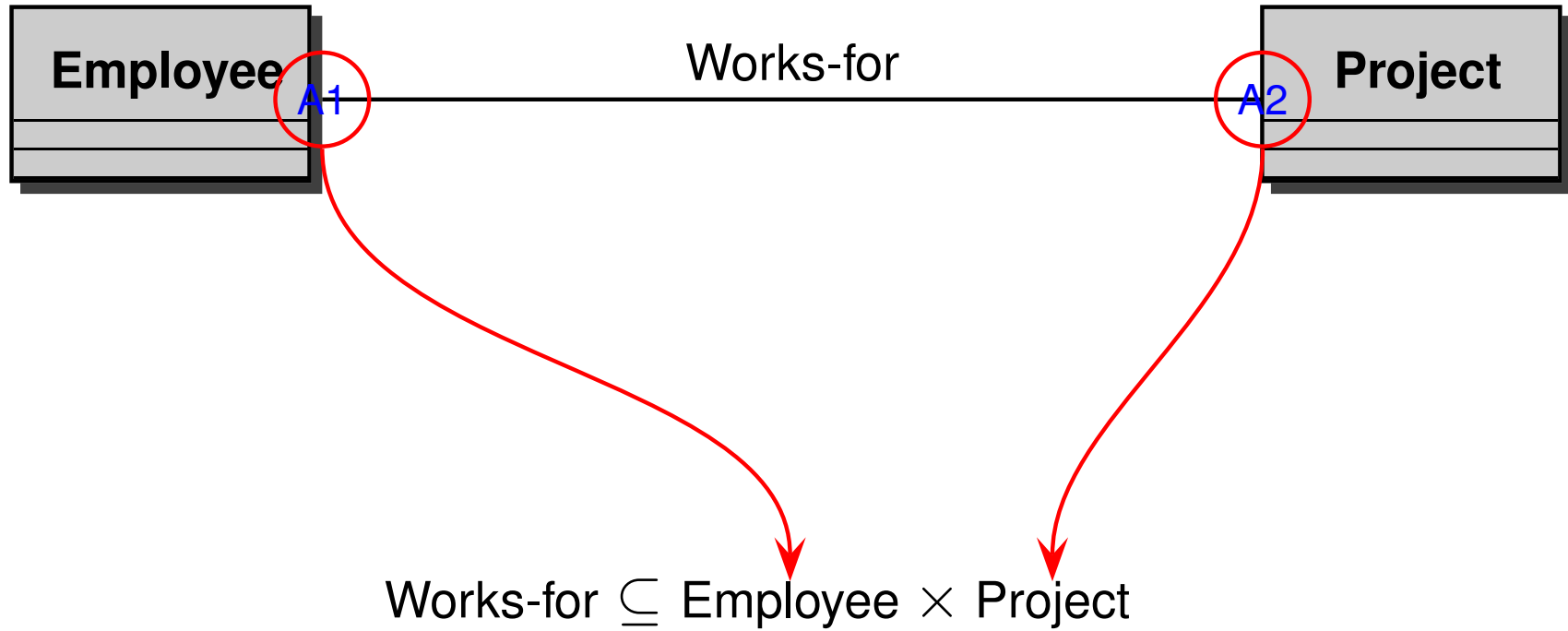


Constraints introduced by Relationships

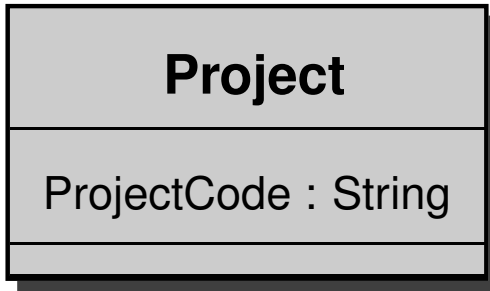


$$\text{Works-for} \subseteq \text{Employee} \times \text{Project}$$

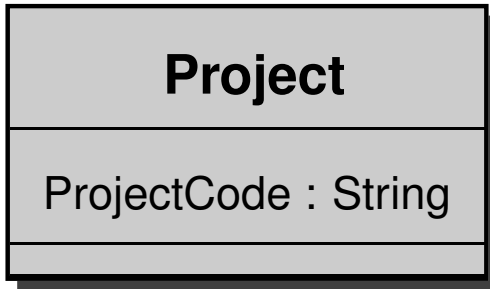
Constraints introduced by Relationships



Constraints introduced by Attributes

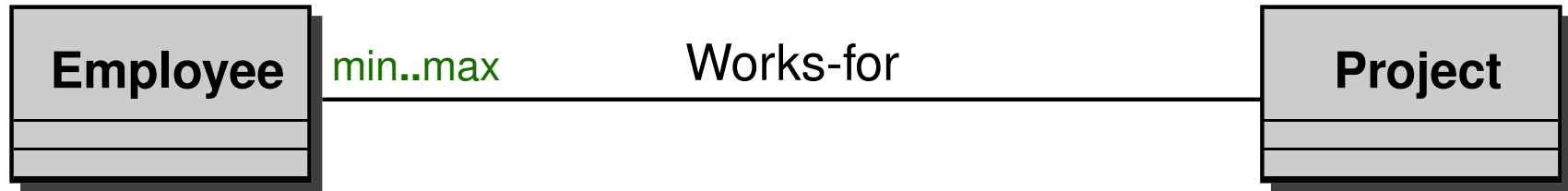


Constraints introduced by Attributes

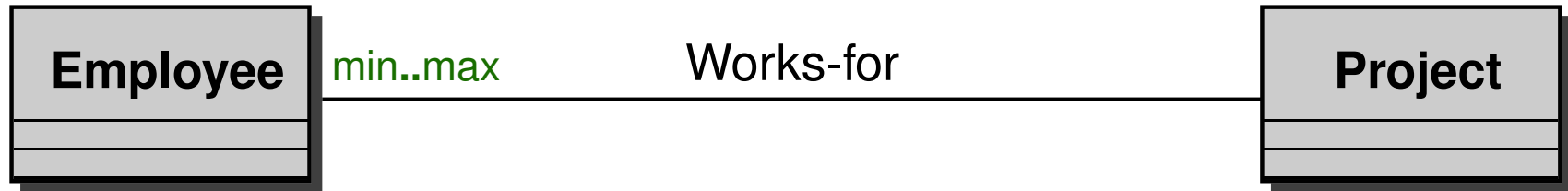


$$\text{Project} \subseteq \{p \mid \#(\text{ProjectCode} \cap (\{p\} \times \text{String})) \geq 1\}$$

Constraints introduced by Cardinality Constraints



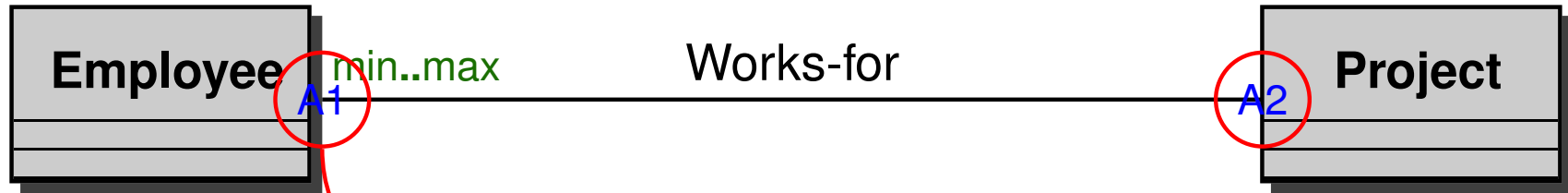
Constraints introduced by Cardinality Constraints



$$\text{TopManager} \subseteq \{m \mid \text{max} \geq \#(\text{Manages} \cap (\{m\} \times \Omega)) \geq \text{min}\}$$

(where Ω is the set of all instances)

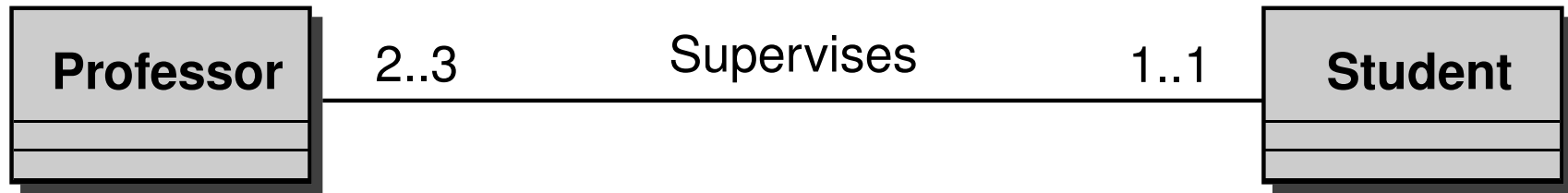
Constraints introduced by Cardinality Constraints



$$\text{TopManager} \subseteq \{m \mid \text{max} \geq \#(\text{Manages} \cap (\{m\} \times \Omega)) \geq \text{min}\}$$

(where Ω is the set of all instances)

The Cardinality Construct: An Example



A valid Database is:

Professor

<i>professorId</i>
Alex
Bob

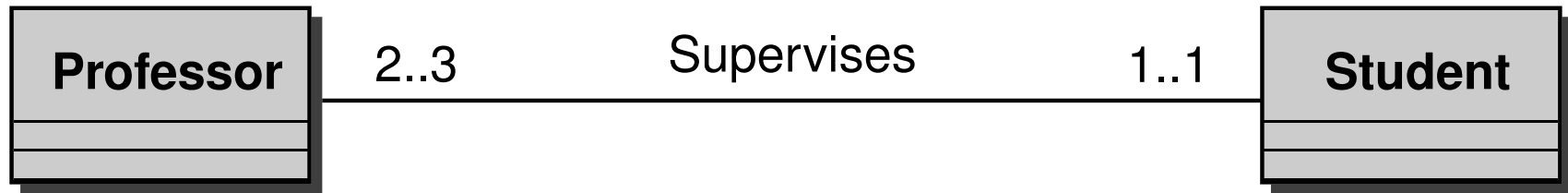
Student

<i>studentId</i>
John
Mary
Nick
Paul
Laura

Supervises

<i>professorId</i>	<i>studentId</i>
Alex	John
Bob	Laura
Alex	Mary
Bob	Nick
Alex	Paul

The Cardinality Construct: An Example



An invalid Database is:

Professor

<i>professorId</i>
Alex
Bob

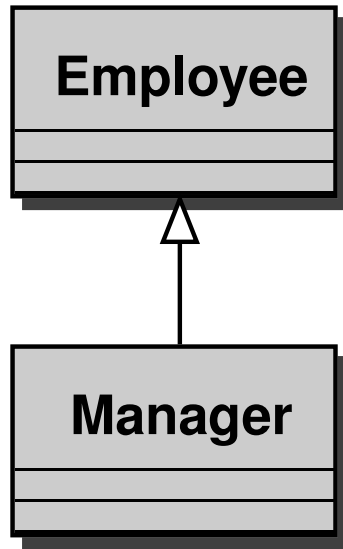
Student

<i>studentId</i>
John
Mary
Nick
Paul
Laura

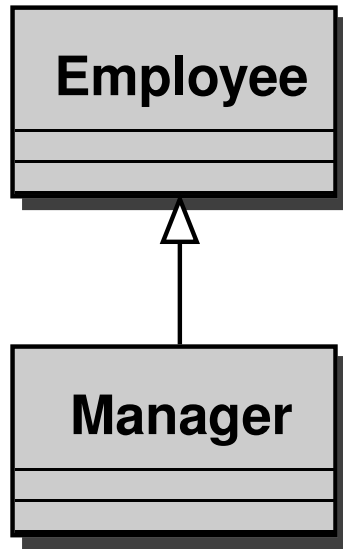
Supervises

<i>professorId</i>	<i>studentId</i>
Alex	John
Bob	Laura
Alex	Mary
Bob	Nick
Alex	Paul
Alex	Laura

Constraints introduced by ISA

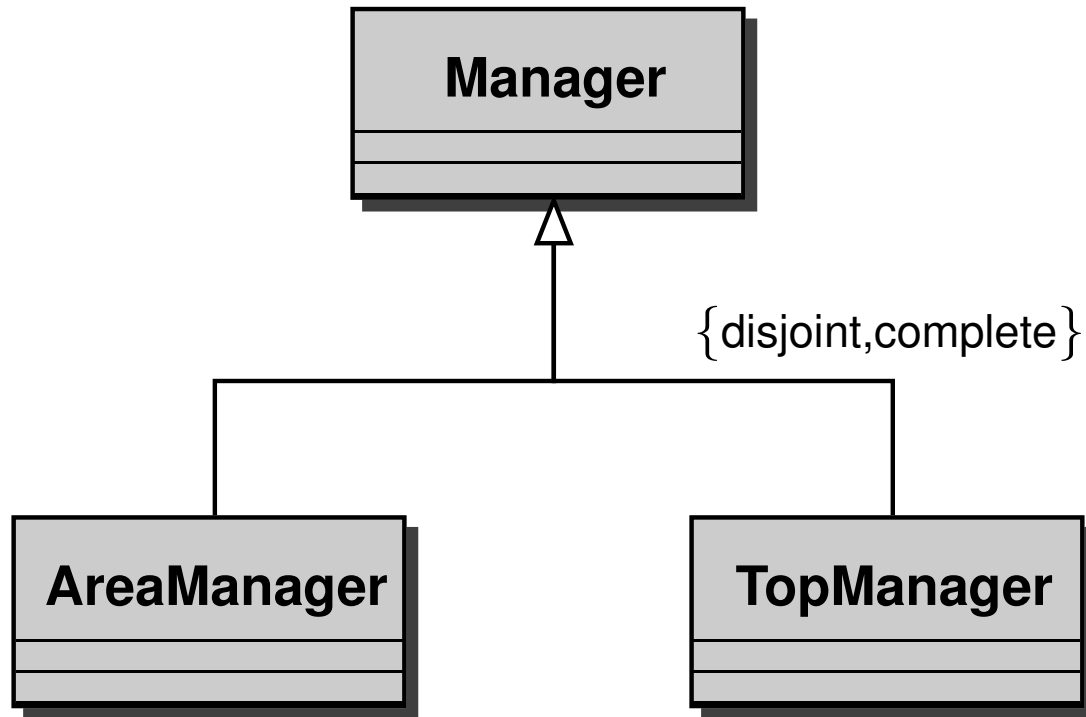


Constraints introduced by ISA

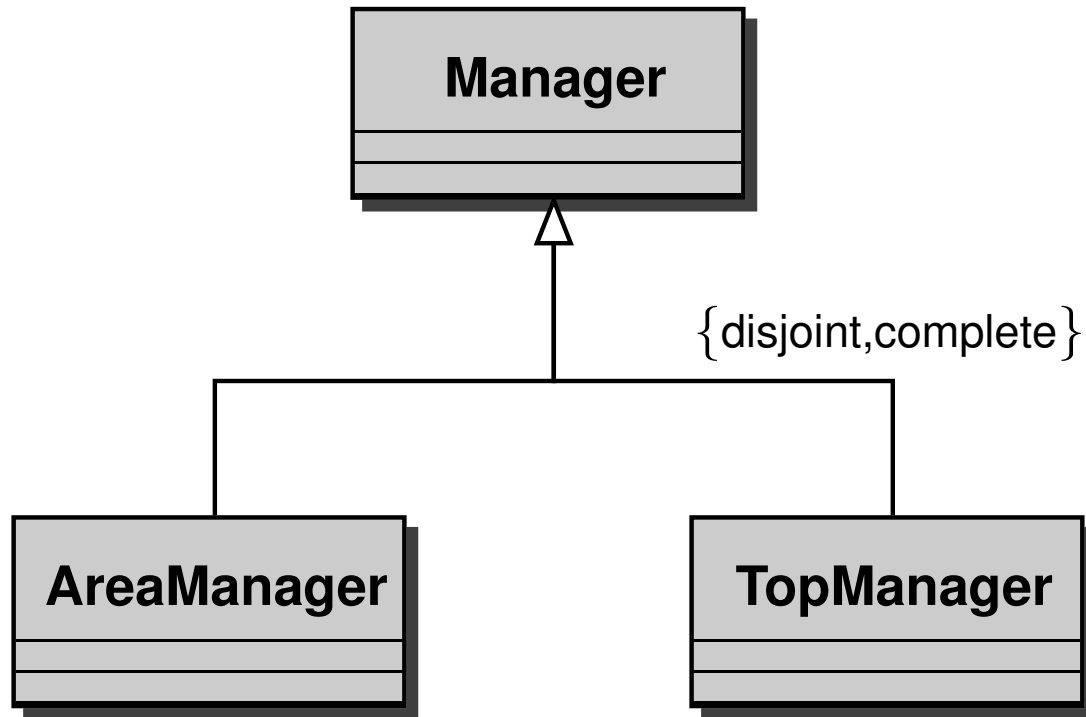


Manager \subseteq Employee

Disjoint and Total constraints



Disjoint and Total constraints



- *ISA*: $\text{AreaManager} \subseteq \text{Manager}$
- *ISA*: $\text{TopManager} \subseteq \text{Manager}$
- *disjoint*: $\text{AreaManager} \cap \text{TopManager} = \emptyset$
- *total*: $\text{Manager} \subseteq \text{AreaManager} \cup \text{TopManager}$

Constraints introduced by the initial diagram

Works-for \subseteq Employee \times Project

Manages \subseteq TopManager \times Project

Employee $\subseteq \{e \mid \#(\text{PaySlipNumber} \cap (\{e\} \times \text{Integer})) \geq 1\}$

Employee $\subseteq \{e \mid \#(\text{Salary} \cap (\{e\} \times \text{Integer})) \geq 1\}$

Project $\subseteq \{p \mid \#(\text{ProjectCode} \cap (\{p\} \times \text{String})) \geq 1\}$

TopManager $\subseteq \{m \mid 1 \geq \#(\text{Manages} \cap (\{m\} \times \Omega)) \geq 1\}$

Project $\subseteq \{p \mid 1 \geq \#(\text{Manages} \cap (\Omega \times \{p\})) \geq 1\}$

Project $\subseteq \{p \mid \#(\text{Works-for} \cap (\Omega \times \{p\})) \geq 1\}$

Manager \subseteq Employee

AreaManager \subseteq Manager

TopManager \subseteq Manager

AreaManager \cap TopManager = \emptyset

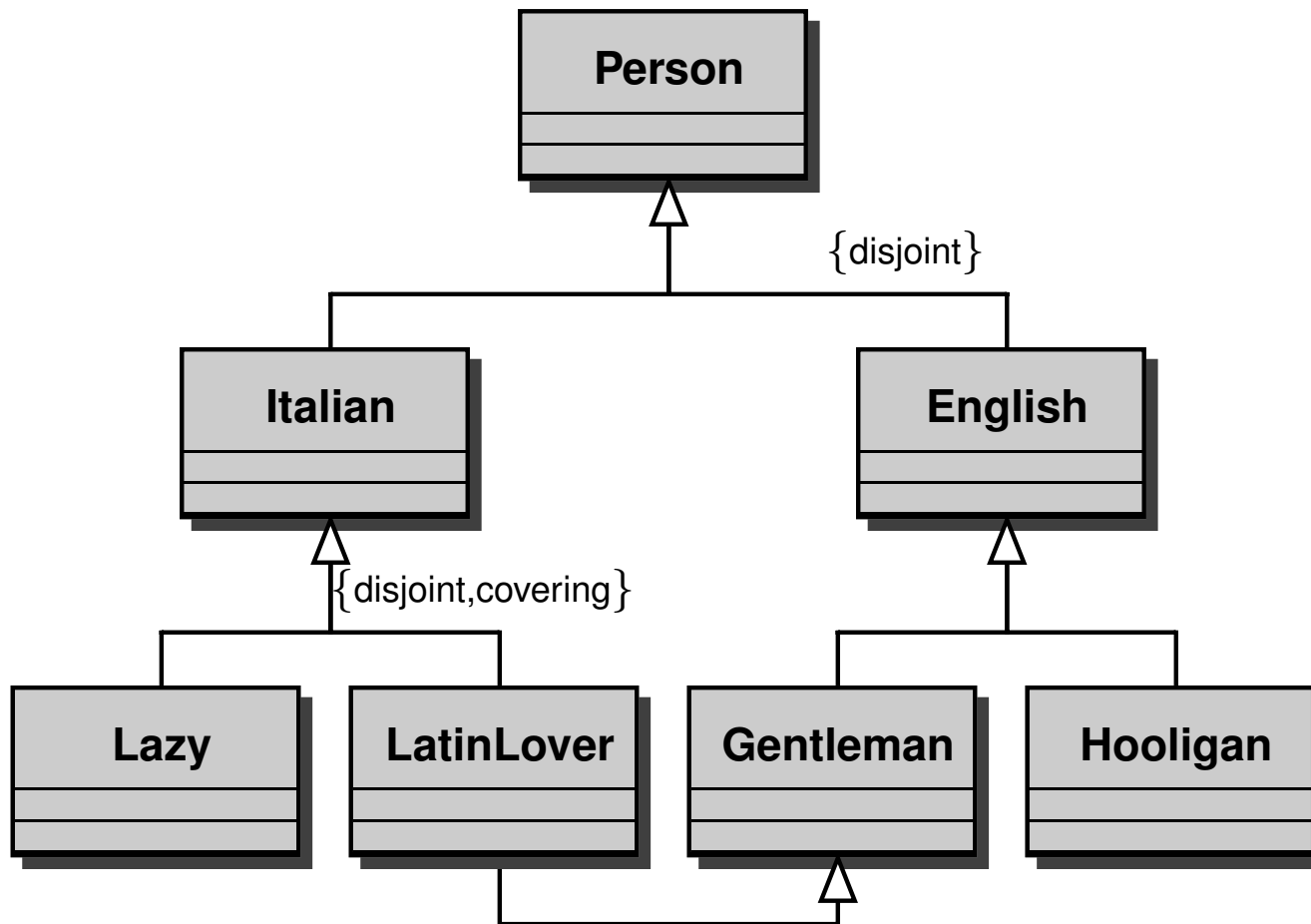
Manager \subseteq AreaManager \cup TopManager

Reasoning

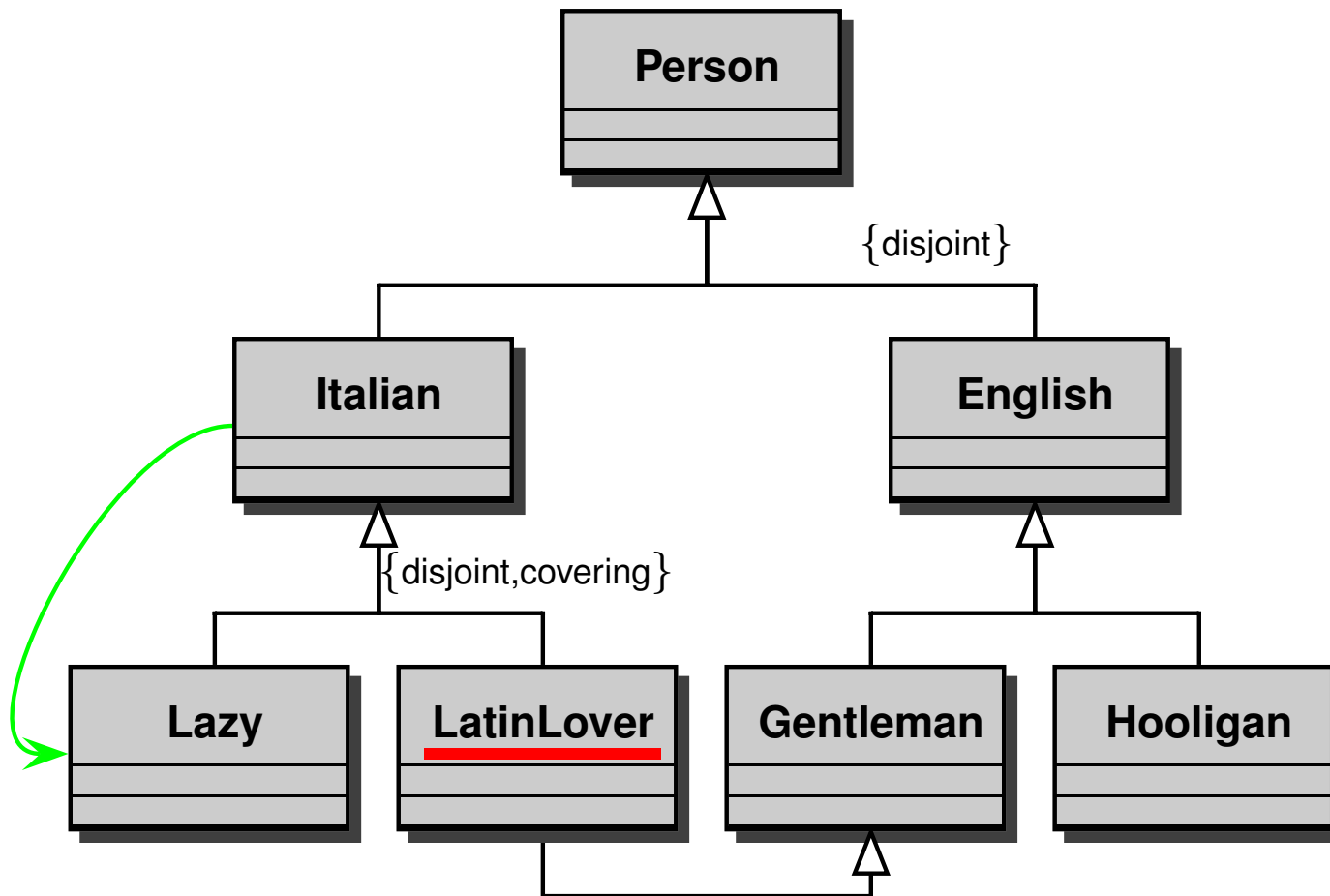
Given an ontology – seen as a collection of constraints – it is possible that additional constraints can be inferred.

- A class is **inconsistent** if it denotes the empty set in any legal world description.
- A class is a **subclass** of another class if the former denotes a subset of the set denoted by the latter in any legal world description.
- Two classes are **equivalent** if they denote the same set in any legal world description.
- A **stricter** constraint is inferred – e.g., a **cardinality** constraint – if it holds in in any legal world description.
- . . .

Simple reasoning example



Simple reasoning example



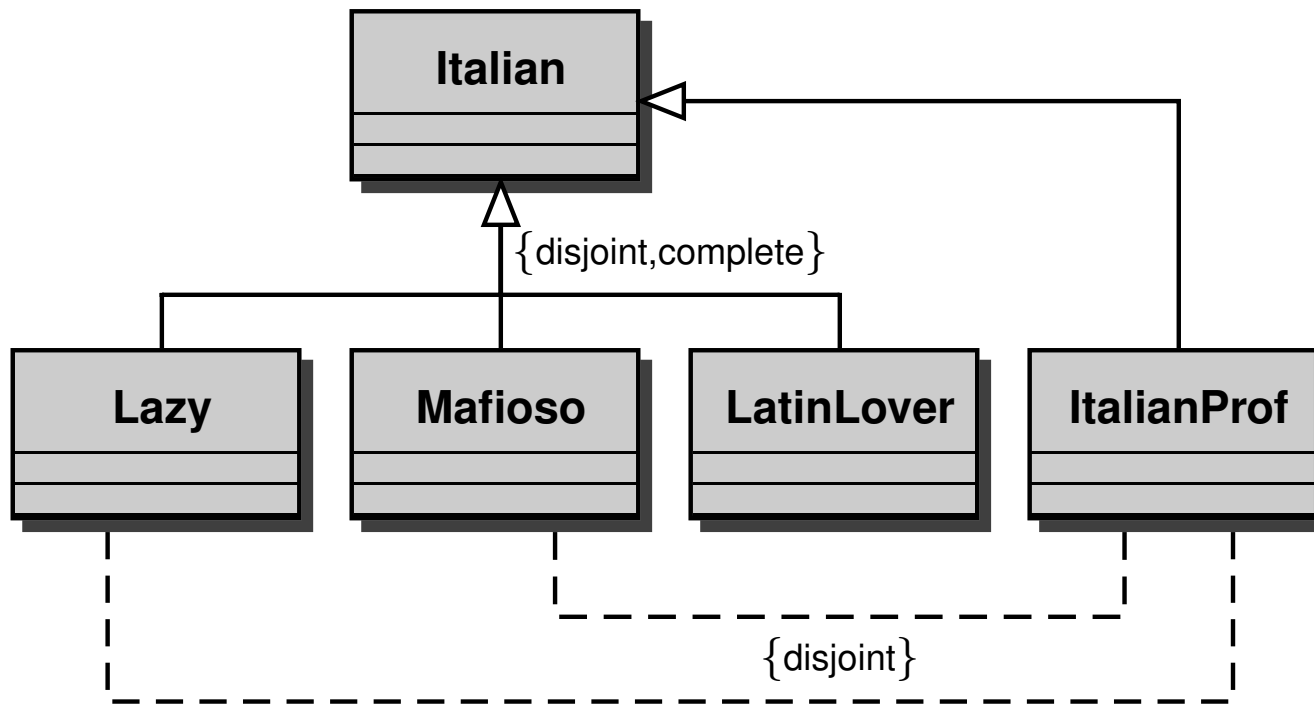
implies

LatinLover = \emptyset

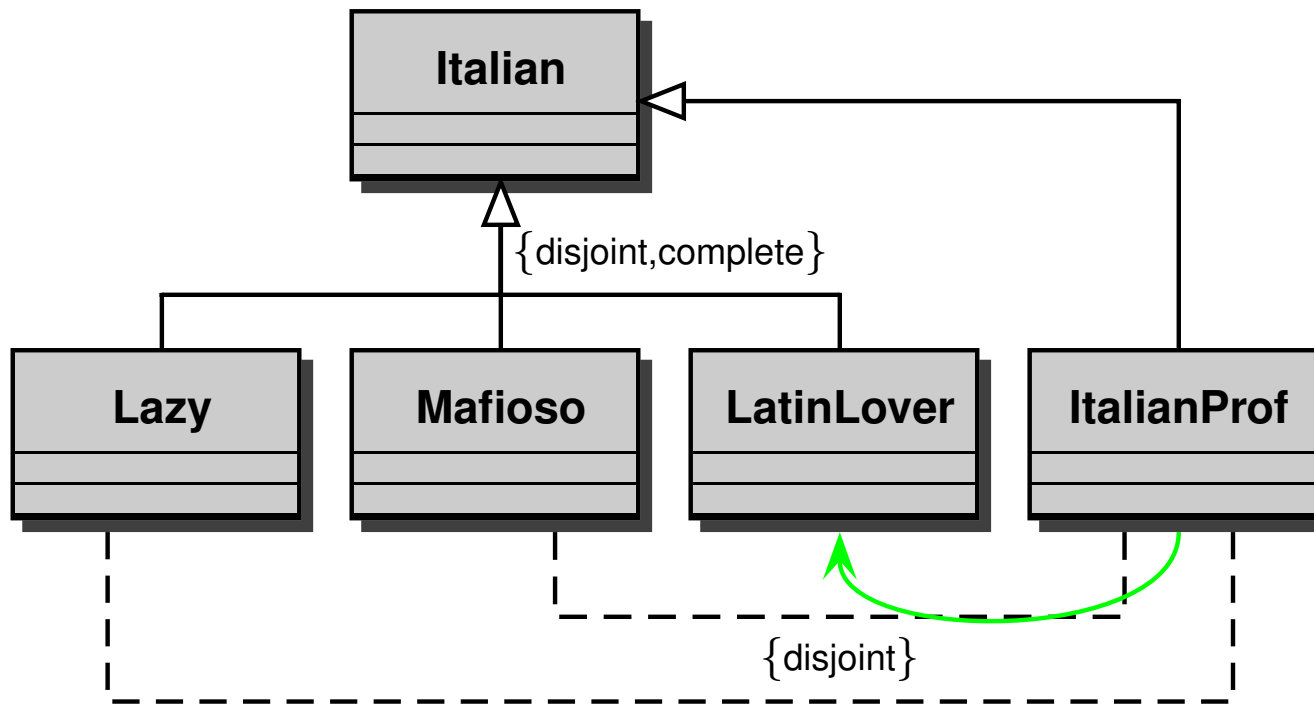
Italian \subseteq Lazy

Italian \equiv Lazy

Reasoning by cases



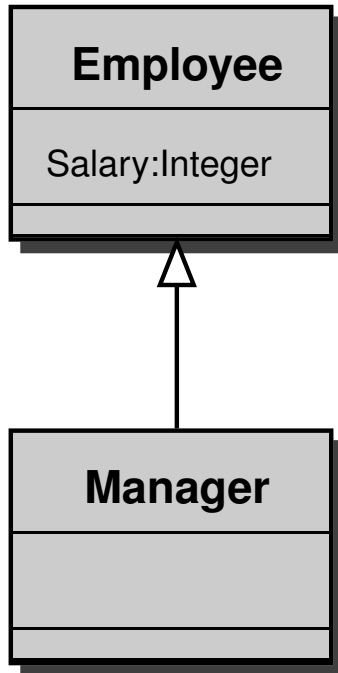
Reasoning by cases



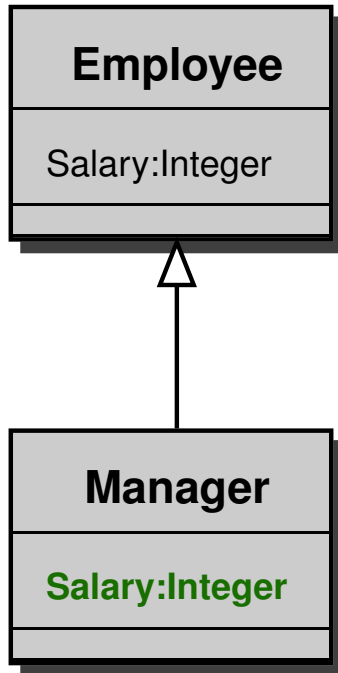
implies

ItalianProf \subseteq LatinLover

ISA and Inheritance



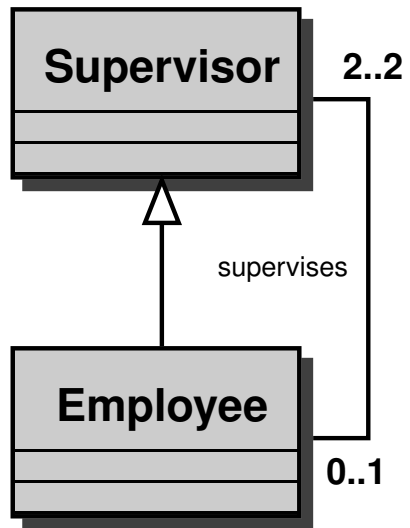
ISA and Inheritance



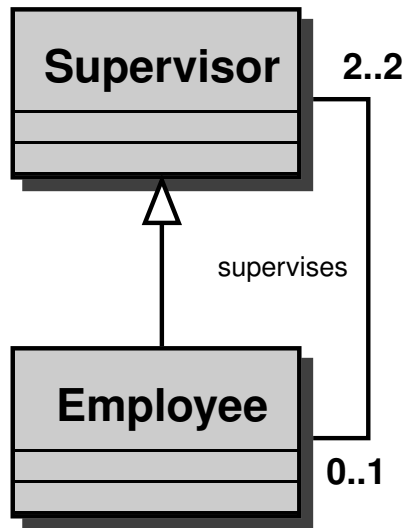
implies

$$\text{Manager} \subseteq \{m \mid \#(\text{Salary} \cap (\{m\} \times \text{Integer})) \geq 1\}$$

Infinite worlds



Infinite worlds

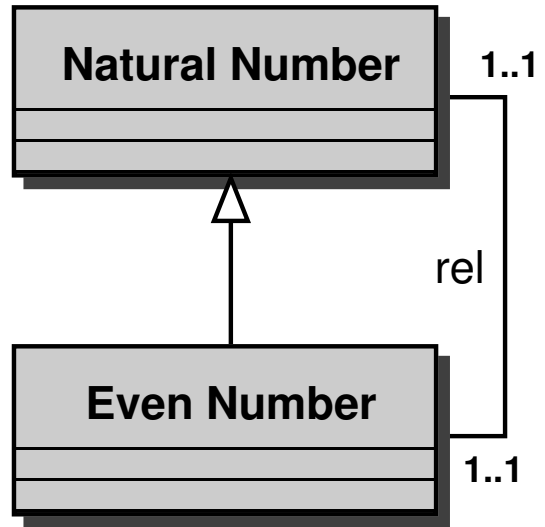


implies

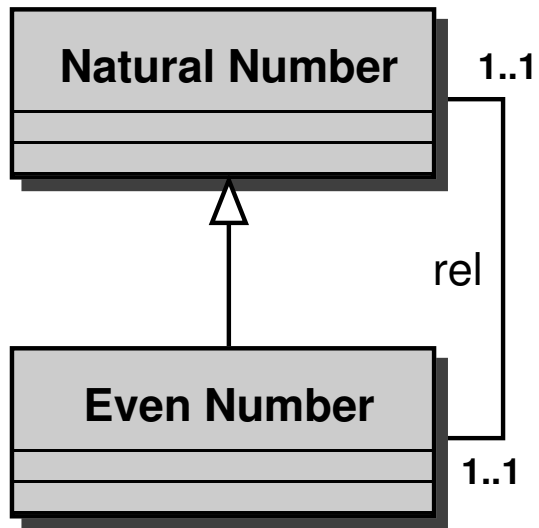
“the classes **Employee** and **Supervisor** contain an infinite number of instances”.

Therefore, the schema is inconsistent.

Bijection



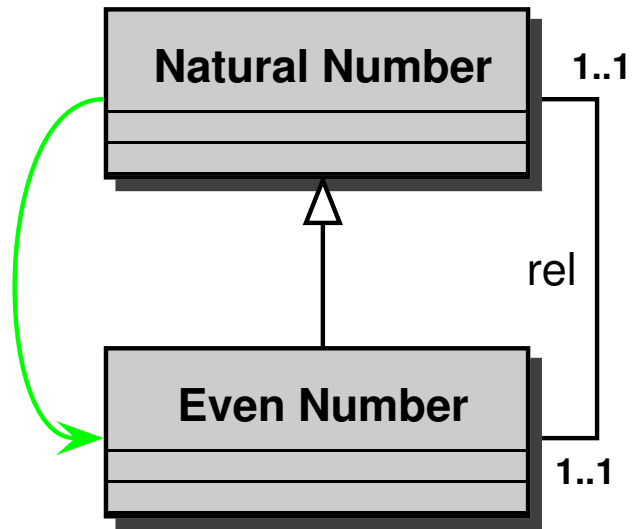
Bijection



implies

“the classes *'Natural Number'* and *'Even Number'* contain the same number of instances”.

Bijection



implies

“the classes *'Natural Number'* and *'Even Number'* contain the same number of instances”.

If the domain is finite: $\text{Natural Number} \equiv \text{Even Number}$

i●com: Intelligent Conceptual Modelling tool

- i●com allows for the specification of multiple UML (or EER) diagrams and inter- and intra-schema constraints;
- Complete logical reasoning is employed by the tool using a hidden underlying (description logic) inference engine;
- i●com verifies the specification, infers implicit facts and stricter constraints, and manifests any inconsistencies during the conceptual modelling phase.
- `www.cs.man.ac.uk/~franconi/icom/`

Ontologies in First Order Logic

- We have introduced ontology languages that specify a set of constraints that should be satisfied by the world of interest.
- The *interpretation* of an ontology is therefore defined as the collection of all the *legal world descriptions* – i.e., all the (finite) relational structures which conform to the constraints imposed by the ontology.
- An alternative way to define the interpretation: an ontology is mapped into a set of **First Order Logic** (FOL) formulas.
- The legal world descriptions (i.e., the interpretation) of an ontology are all the **models** of the FOL theory associated to it.

FOL: The Alphabet

The Alphabet of the FOL language will have the following set of *Predicate* symbols:

- 1-ary predicate symbols: E_1, E_2, \dots, E_n for each Class (Entity);
 D_1, D_2, \dots, D_m for each Basic Domain.
- binary predicate symbols: A_1, A_2, \dots, A_k for each Attribute.
- n-ary predicate symbols: R_1, R_2, \dots, R_p for each Association (Relation).

FOL Notation

- *Vector variables* indicated as \bar{x} stand for an n-tuple of variables:

$$\bar{x} = x_1, \dots, x_n$$

- *Counting existential quantifier* indicated as $\exists^{\leq n}$ or $\exists^{\geq n}$.

$$\exists^{\leq n} x. \varphi(x) \equiv$$

$$\begin{aligned} \forall x_1, \dots, x_n, x_{n+1}. \varphi(x_1) \wedge \dots \wedge \varphi(x_n) \wedge \varphi(x_{n+1}) \rightarrow \\ (x_1 = x_2) \vee \dots \vee (x_1 = x_n) \vee (x_1 = x_{n+1}) \vee \\ (x_2 = x_3) \vee \dots \vee (x_2 = x_n) \vee (x_2 = x_{n+1}) \vee \\ \dots \vee (x_n = x_{n+1}) \end{aligned}$$

$$\exists^{\geq n} x. \varphi(x) \equiv$$

$$\begin{aligned} \exists x_1, \dots, x_n. \varphi(x_1) \wedge \dots \wedge \varphi(x_n) \wedge \\ \neg(x_1 = x_2) \wedge \dots \wedge \neg(x_1 = x_n) \wedge \\ \neg(x_2 = x_3) \wedge \dots \wedge \neg(x_2 = x_n) \wedge \\ \dots \wedge (x_{n-1} = x_n) \end{aligned}$$

The Interpretation function

Interpretation: $\mathcal{I} = \langle \mathbf{D}, \cdot^{\mathcal{I}} \rangle$, where \mathbf{D} is an arbitrary non-empty set such that:

- $\mathbf{D} = \Omega \cup \mathcal{B}$, where:
 - $\mathcal{B} = \cup_{i=1}^m \mathcal{B}_{Di}$. \mathcal{B}_{Di} is the set of values associated with each basic domain (i.e., integer, string, etc.); and $\mathcal{B}_{Di} \cap \mathcal{B}_{Dj} = \emptyset, \forall i, j. i \neq j$
 - Ω is the abstract entity domain such that $\mathcal{B} \cap \Omega = \emptyset$.

The Formal Semantics for the Atoms

\mathcal{I} is the interpretation function that maps:

- *Basic Domain Predicates* to elements of the relative basic domain:

$$D_i^{\mathcal{I}} = \mathcal{B}_{Di} \quad (\text{e.g., } \text{String}^{\mathcal{I}} = \mathcal{B}_{\text{String}}).$$

- *Entity-set Predicates* to elements of the entity domain:

$$E_i^{\mathcal{I}} \subseteq \Omega.$$

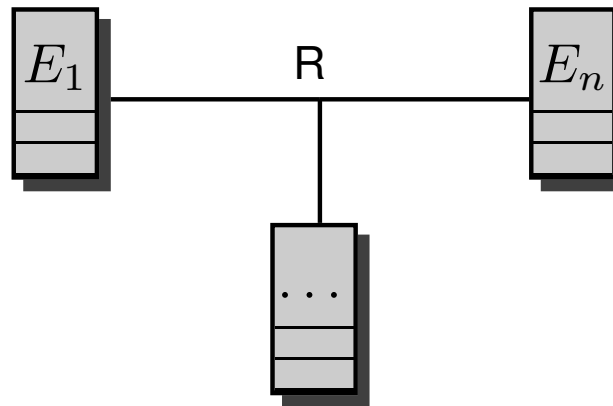
- *Attribute Predicates* to binary relations such that:

$$A_i^{\mathcal{I}} \subseteq \Omega \times \mathcal{B}.$$

- *Relationship-set Predicates* to n-ary relations over the entity domain:

$$R_i^{\mathcal{I}} \subseteq \Omega \times \Omega \dots \times \Omega = \Omega^n.$$

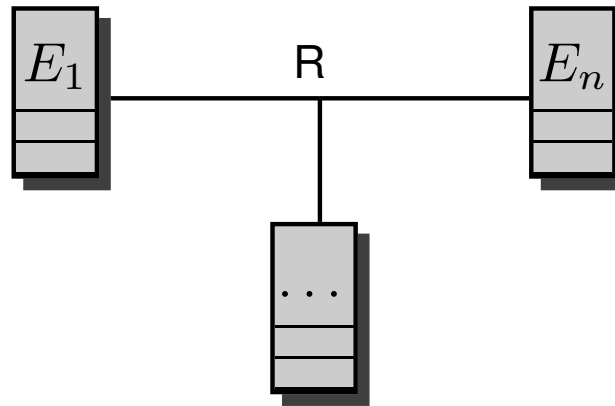
The Relationship Construct



- The meaning of this constraint is:

$$R^{\mathcal{I}} \subseteq E_1^{\mathcal{I}} \times \dots \times E_n^{\mathcal{I}}$$

The Relationship Construct



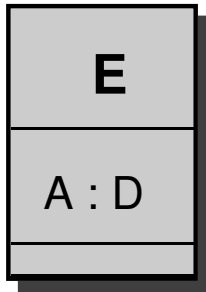
- The meaning of this constraint is:

$$R^{\mathcal{I}} \subseteq E_1^{\mathcal{I}} \times \dots \times E_n^{\mathcal{I}}$$

- The FOL translation is the formula:

$$\forall x_1, \dots, x_n. R(x_1, \dots, x_n) \rightarrow E_1(x_1) \wedge \dots \wedge E_n(x_n)$$

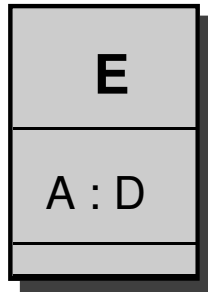
The Attribute Construct



- The meaning of this constraint is:

$$E^{\mathcal{I}} \subseteq \{e \in \Omega \mid \#(A^{\mathcal{I}} \cap (\{e\} \times \mathcal{B}_D)) \geq 1\}$$

The Attribute Construct



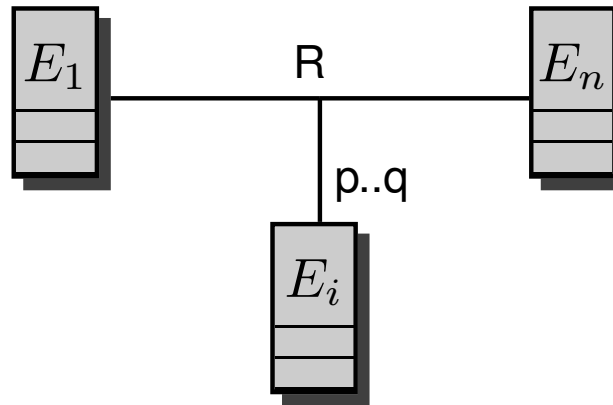
- The meaning of this constraint is:

$$E^{\mathcal{I}} \subseteq \{e \in \Omega \mid \#(A^{\mathcal{I}} \cap (\{e\} \times \mathcal{B}_D)) \geq 1\}$$

- The FOL translation is the formula:

$$\forall x. E(x) \rightarrow \exists y. A(x, y) \wedge D(y)$$

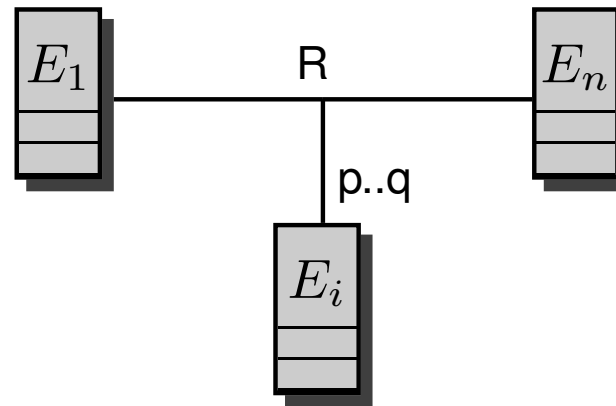
The Cardinality Constraint



- The meaning of this constraint is:

$$E_i^{\mathcal{I}} \subseteq \{e_i \in \Omega \mid p \leq \#(R^{\mathcal{I}} \cap (\Omega \times \{e_i\} \times \Omega)) \leq q\}$$

The Cardinality Construct



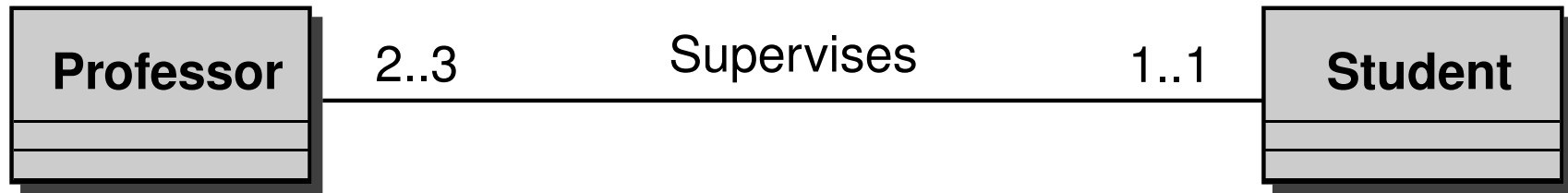
- The meaning of this constraint is:

$$E_i^{\mathcal{I}} \subseteq \{e_i \in \Omega \mid p \leq \#(R^{\mathcal{I}} \cap (\Omega \times \{e_i\} \times \Omega)) \leq q\}$$

- The FOL translation is the formula:

$$\forall x_i. E(x_i) \rightarrow \exists^{\geq p} x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n. R(x_1, \dots, x_n) \wedge \\ \exists^{\leq q} x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n. R(x_1, \dots, x_n)$$

The Cardinality Construct: An Example



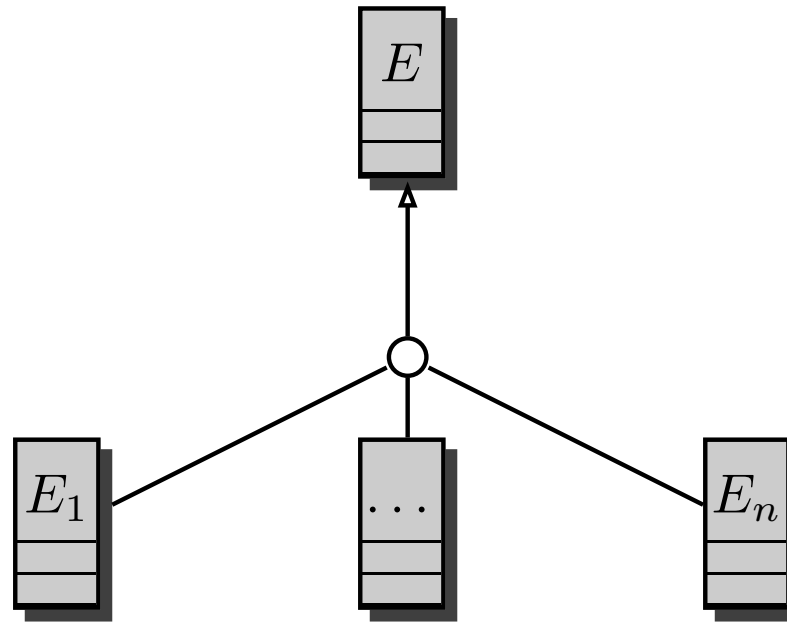
- The FOL translation is:

$$\forall x, y. \text{Supervises}(x, y) \rightarrow \text{Professor}(x) \wedge \text{Student}(y)$$

$$\forall x. \text{Professor}(x) \rightarrow \exists^{\geq 2} y. \text{Supervises}(x, y) \wedge \\ \exists^{\leq 3} y. \text{Supervises}(x, y)$$

$$\forall y. \text{Student}(y) \rightarrow \exists^{=1} x. \text{Supervises}(x, y)$$

ISA Relations



- The meaning of this constraint is:

$$E_i^{\mathcal{I}} \subseteq E^{\mathcal{I}}, \text{ for all } i = 1, \dots, n.$$

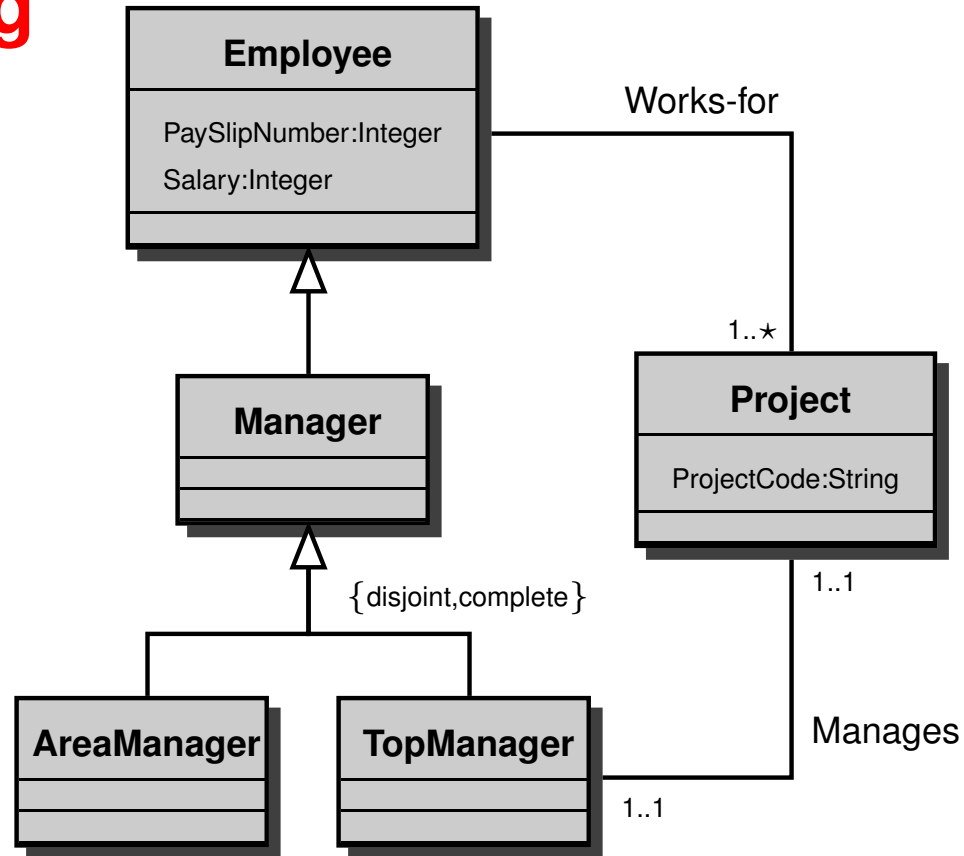
- The FOL translation is the formula:

$$\forall x. E_i(x) \rightarrow E(x), \text{ for all } i = 1, \dots, n.$$

Disjoint and covering constraints

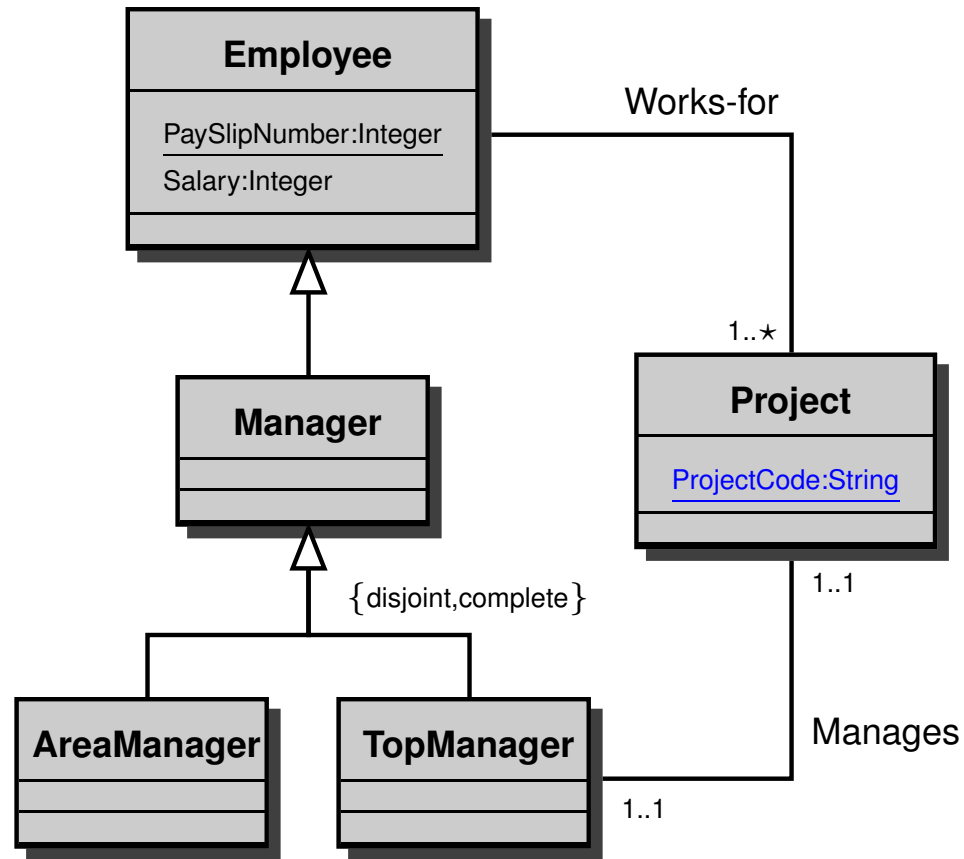
The encoding in FOL of disjoint and covering constraints is left as an exercise.

FOL encoding



- $\forall x, y. \text{Works-for}(x, y)$ \rightarrow $\text{Employee}(x) \wedge \text{Project}(y)$
- $\forall x, y. \text{Manages}(x, y)$ \rightarrow $\text{Top-Manager}(x) \wedge \text{Project}(y)$
- $\forall y. \text{Project}(y)$ \rightarrow $\exists x. \text{Works-for}(x, y)$
- $\forall y. \text{Project}(y)$ \rightarrow $\exists^{=1} x. \text{Manages}(x, y)$
- $\forall x. \text{Top-Manager}(x)$ \rightarrow $\exists^{=1} y. \text{Manages}(x, y)$
- $\forall x. \text{Manager}(x)$ \rightarrow $\text{Employee}(x)$
- $\forall x. \text{Manager}(x)$ \rightarrow $\text{Area-Manager}(x) \vee \text{Top-Manager}(x)$
- $\forall x. \text{Area-Manager}(x)$ \rightarrow $\text{Manager}(x) \wedge \neg \text{Top-Manager}(x)$
- $\forall x. \text{Top-Manager}(x)$ \rightarrow $\text{Manager}(x)$

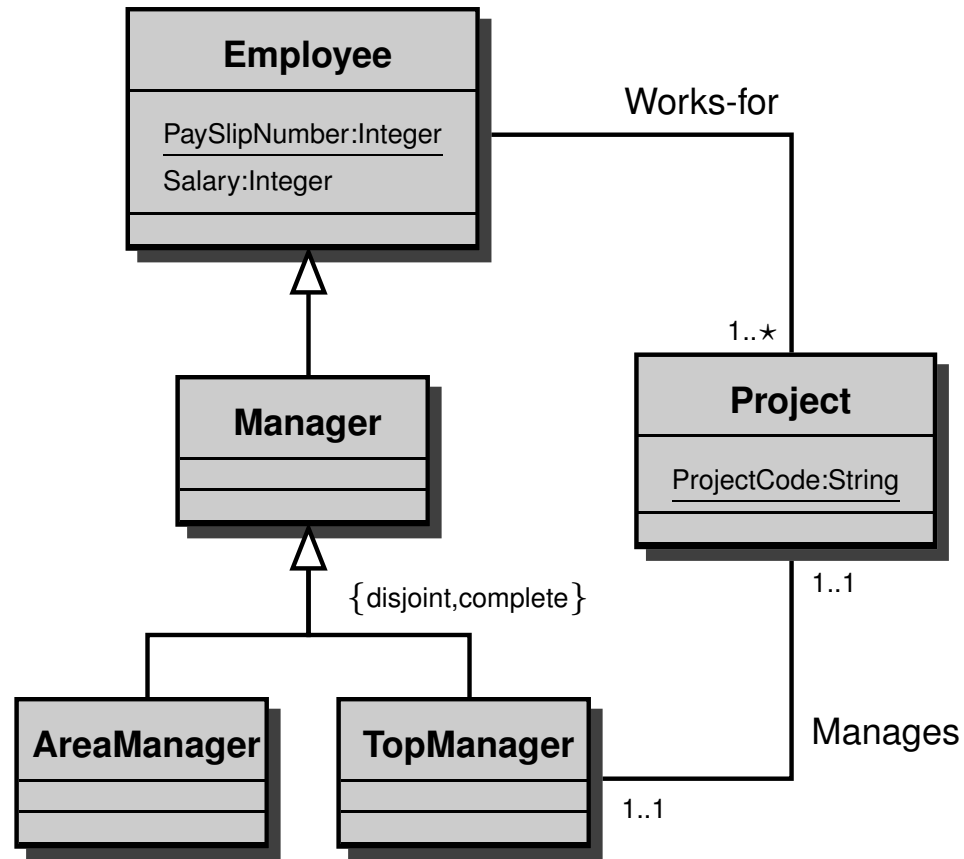
Key constraints



$$\forall x. \text{Project}(x) \rightarrow \exists^{=1} y. \text{ProjectCode}(x, y) \wedge \text{String}(y)$$

$$\forall y. \exists x. \text{ProjectCode}(x, y) \rightarrow \exists^{=1} x. \text{ProjectCode}(x, y) \wedge \text{Project}(x)$$

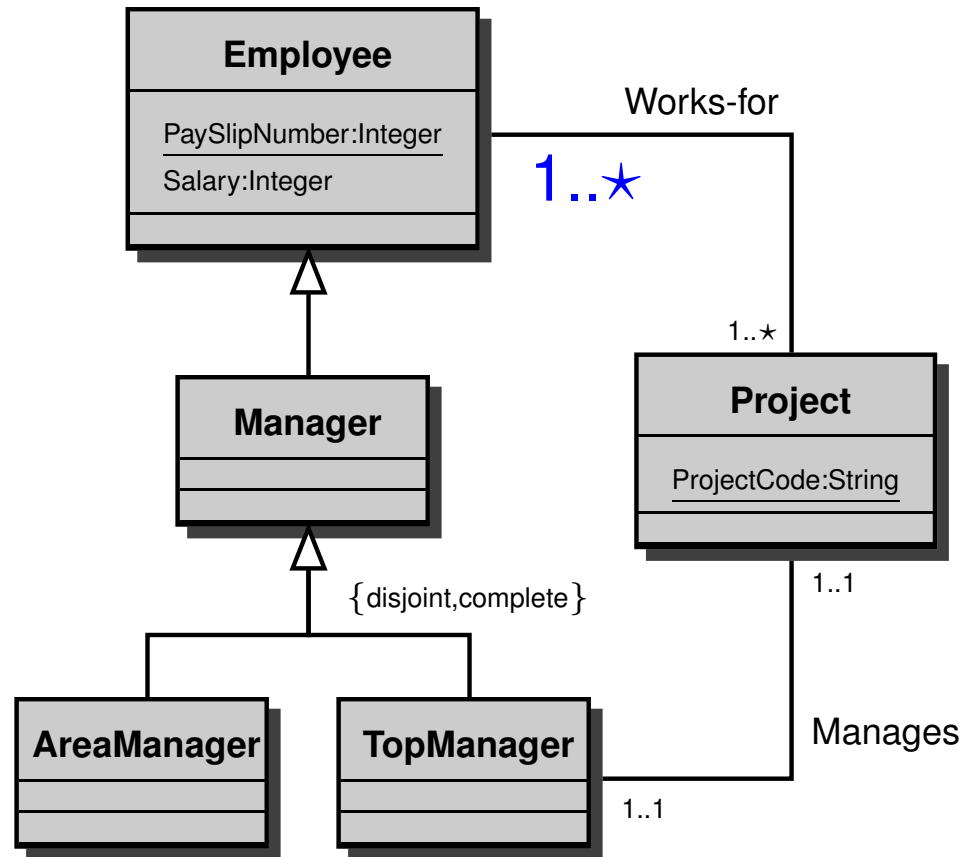
Additional constraints



- Managers do not work for a project (she/he just manages it):

$$\forall x. \text{Manager}(x) \rightarrow \forall y. \neg \text{WORKS-FOR}(x, y)$$

Additional constraints



- Managers do not work for a project (she/he just manages it):

$$\forall x. \text{Manager}(x) \rightarrow \forall y. \neg \text{WORKS-FOR}(x, y)$$

- If the **minimum cardinality** for the participation of employees to the *works-for* relationship is increased, then . . .

Summary

- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- Queries with an Ontology

The \mathcal{DLR} Description Logic – a fragment of FOL

- **relationships:** interpreted as **sets of tuples** of a given arity

$$R \rightarrow \top_n \mid RN \mid \neg R \mid R_1 \sqcap R_2 \mid R_1 \sqcup R_2 \mid i/n : C$$

- **classes:** interpreted as **sets of objects**

$$C \rightarrow \top \mid CN \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists^{\leq k}[i]R$$

- **conceptual schema:** $R \sqsubseteq R' \mid C \sqsubseteq C' \mid R \not\sqsubseteq R' \mid C \not\sqsubseteq C'$

Works-for \sqsubseteq subj/2 : Employee \sqcap obj/2 : Project

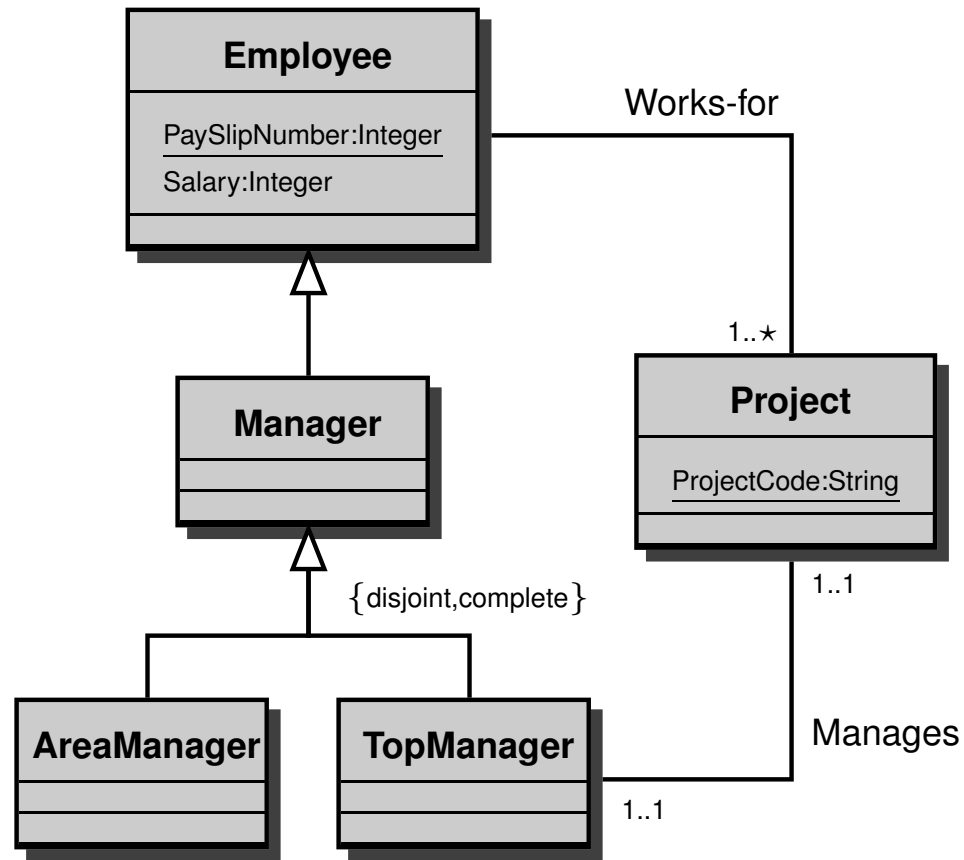
TopManager \sqsubseteq Manager \sqcap $\exists^{=1}[\text{man}]$ Manages

Encoding ontologies in Description Logics

- Object-oriented data models (e.g., UML and ODMG)
- Semantic data models (e.g., EER and ORM)
- Frame-based ontology languages (e.g., DAML+OIL)

Encoding ontologies in Description Logics

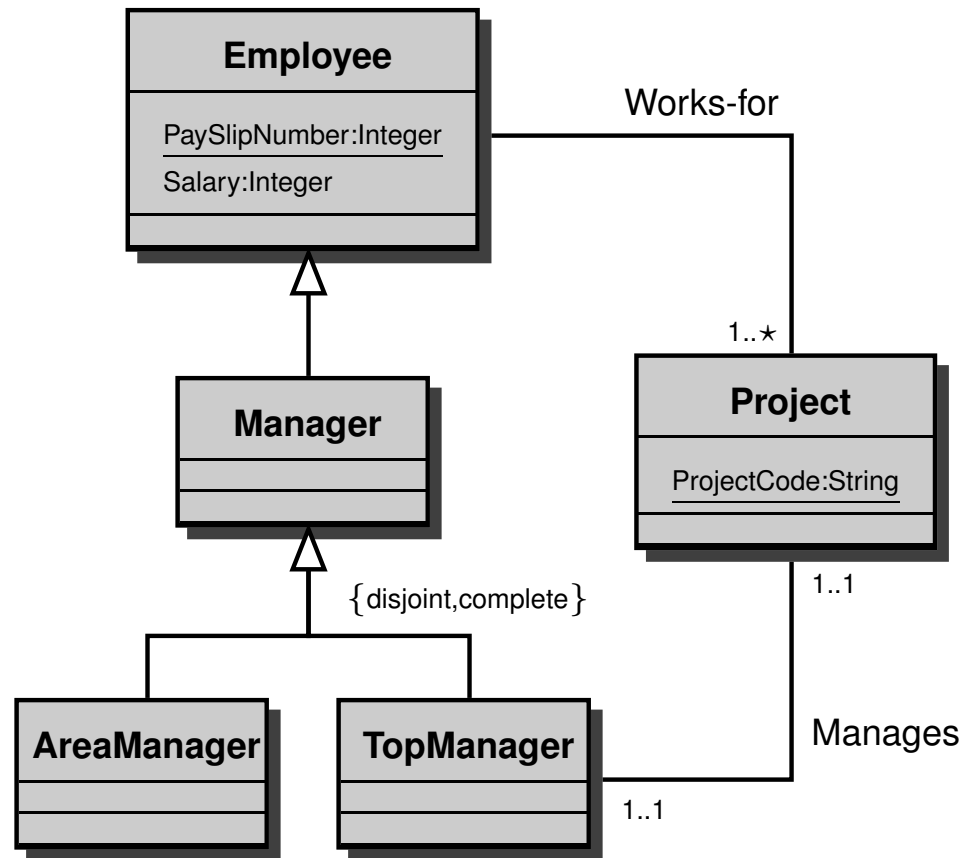
- Object-oriented data models (e.g., UML and ODMG)
- Semantic data models (e.g., EER and ORM)
- Frame-based ontology languages (e.g., DAML+OIL)
- Theorems **prove** that an ontology and its encoding as DL knowledge bases constrain every world description in the same way – i.e., the models of the DL theory correspond to the legal world descriptions of the ontology, and vice-versa.



Works-for	\sqsubseteq	$\text{emp}/2 : \text{Employee} \sqcap \text{act}/2 : \text{Project}$
Manages	\sqsubseteq	$\text{man}/2 : \text{TopManager} \sqcap \text{prj}/2 : \text{Project}$
Employee	\sqsubseteq	$\exists^{=1}[\text{worker}](\text{PaySlipNumber} \sqcap \text{num}/2 : \text{Integer}) \sqcap$ $\exists^{=1}[\text{payee}](\text{Salary} \sqcap \text{amount}/2 : \text{Integer})$
T	\sqsubseteq	$\exists^{\leq 1}[\text{num}](\text{PaySlipNumber} \sqcap \text{worker}/2 : \text{Employee})$
Manager	\sqsubseteq	$\text{Employee} \sqcap (\text{AreaManager} \sqcup \text{TopManager})$
AreaManager	\sqsubseteq	$\text{Manager} \sqcap \neg \text{TopManager}$
TopManager	\sqsubseteq	$\text{Manager} \sqcap \exists^{=1}[\text{man}]\text{Manages}$
Project	\sqsubseteq	$\exists^{\geq 1}[\text{act}]\text{Works-for} \sqcap \exists^{=1}[\text{prj}]\text{Manages}$

...

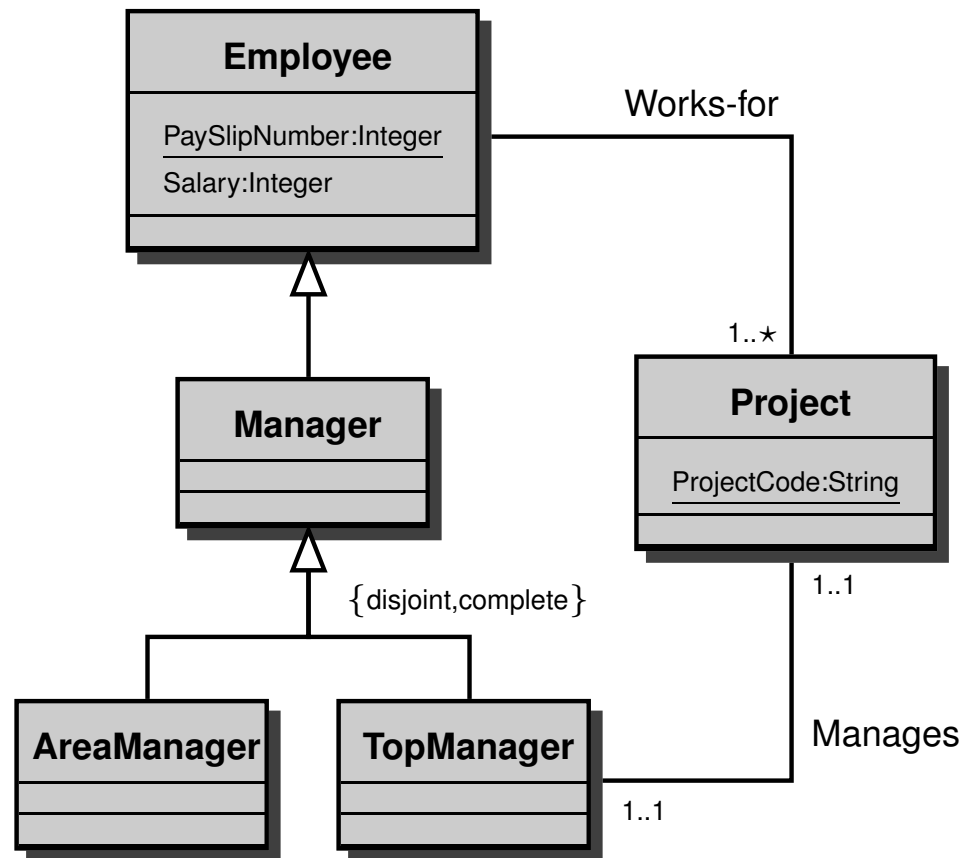
Deducing constraints



Managers are employees who do not work for a project (she/he just manages it):

$$\text{Employee} \sqcap \neg(\exists^{\geq 1}[\text{emp}]\text{Works-for}) \sqsubseteq \text{Manager}, \quad \text{Manager} \sqsubseteq \neg(\exists^{\geq 1}[\text{emp}]\text{Works-for})$$

Deducing constraints



Managers are employees who do not work for a project (she/he just manages it):

$$\text{Employee} \sqcap \neg(\exists^{\geq 1}[\text{emp}]\text{Works-for}) \sqsubseteq \text{Manager}, \quad \text{Manager} \sqsubseteq \neg(\exists^{\geq 1}[\text{emp}]\text{Works-for})$$

⇒ For every project, there is at least one employee who is not a manager:

$$\text{Project} \sqsubseteq \exists^{\geq 1}[\text{act}](\text{Works-for} \sqcap \text{emp} : \neg\text{Manager})$$

Extensions of DLR

- DLR_{reg} : regular expressions and recursive views (beyond FOL)
- DLR_{US} : temporal constructs to model temporal databases (temporal logic)
- DLR_{key} : general key constraints

Reasoning with Ontologies

- Exploit the \mathcal{DLR} reasoning procedures for solving reasoning problems in the ontology enriched with constraints.
- Logical implication and consistency for \mathcal{DLR} knowledge bases is decidable and EXPTIME-complete, and practical, proved correct and complete algorithms exist in implemented systems.

Reasoning with Ontologies

- Exploit the \mathcal{DLR} reasoning procedures for solving reasoning problems in the ontology enriched with constraints.
- Logical implication and consistency for \mathcal{DLR} knowledge bases is decidable and EXPTIME-complete, and practical, proved correct and complete algorithms exist in implemented systems.
- \rightsquigarrow Ontology consistency checking with constraints and [logical implication of constraints in ontologies](#) are all decidable EXPTIME-complete problems.

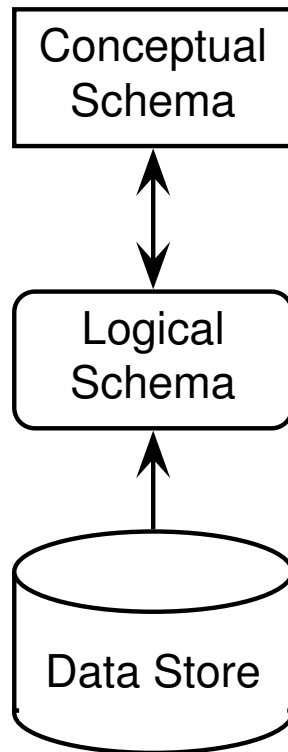
Reasoning with Ontologies

- Exploit the \mathcal{DLR} reasoning procedures for solving reasoning problems in the ontology enriched with constraints.
- Logical implication and consistency for \mathcal{DLR} knowledge bases is decidable and EXPTIME-complete, and practical, proved correct and complete algorithms exist in implemented systems.
- \rightsquigarrow Ontology consistency checking with constraints and [logical implication of constraints in ontologies](#) are all decidable EXPTIME-complete problems.
- i●com is an implemented conceptual modelling tool using in the background a \mathcal{DLR} ontology server supporting the ontology design.

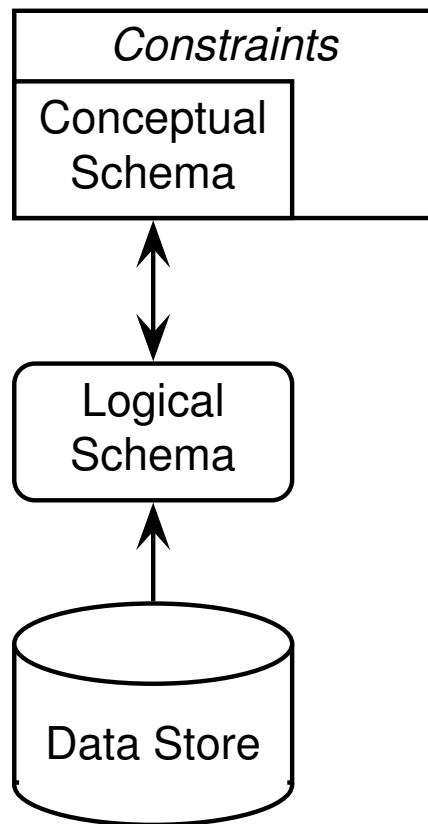
Summary

- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- Queries with an Ontology

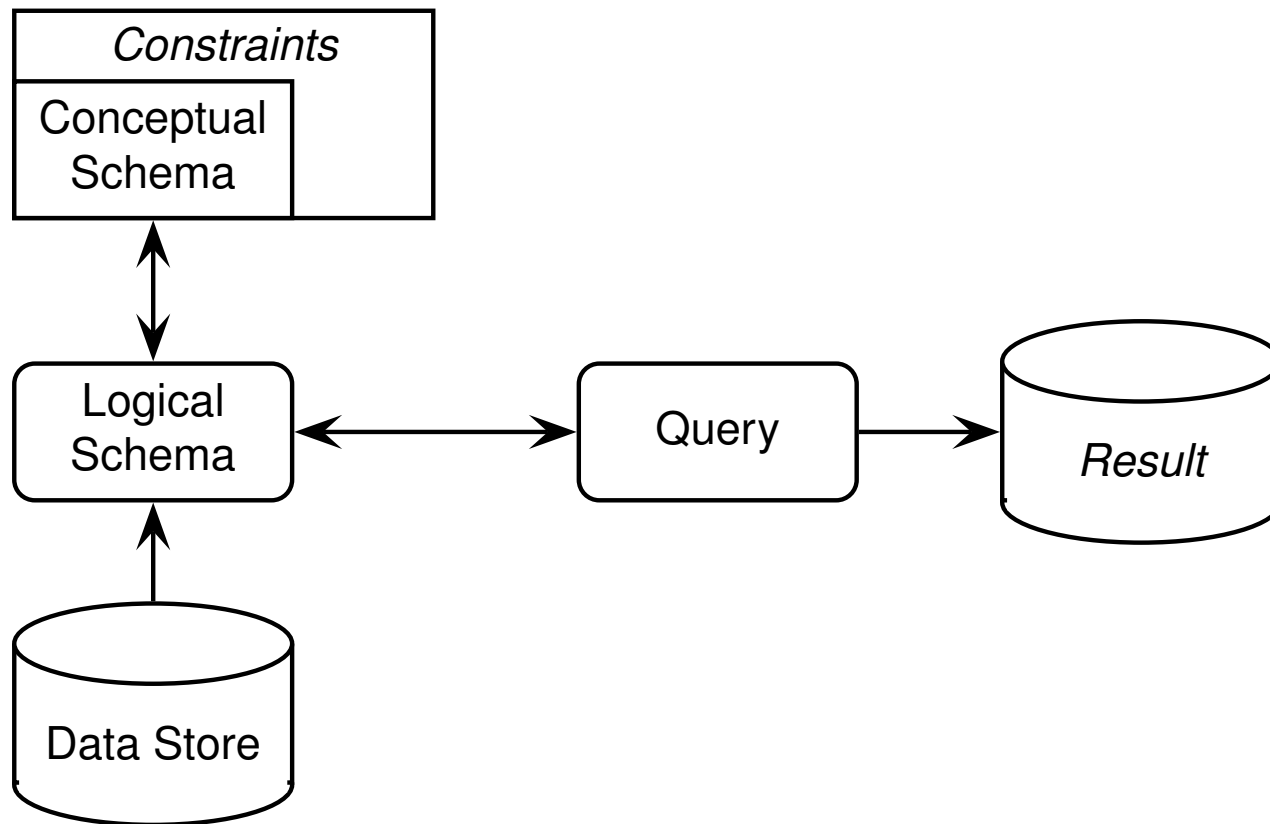
The role of a Conceptual Schema – revisited



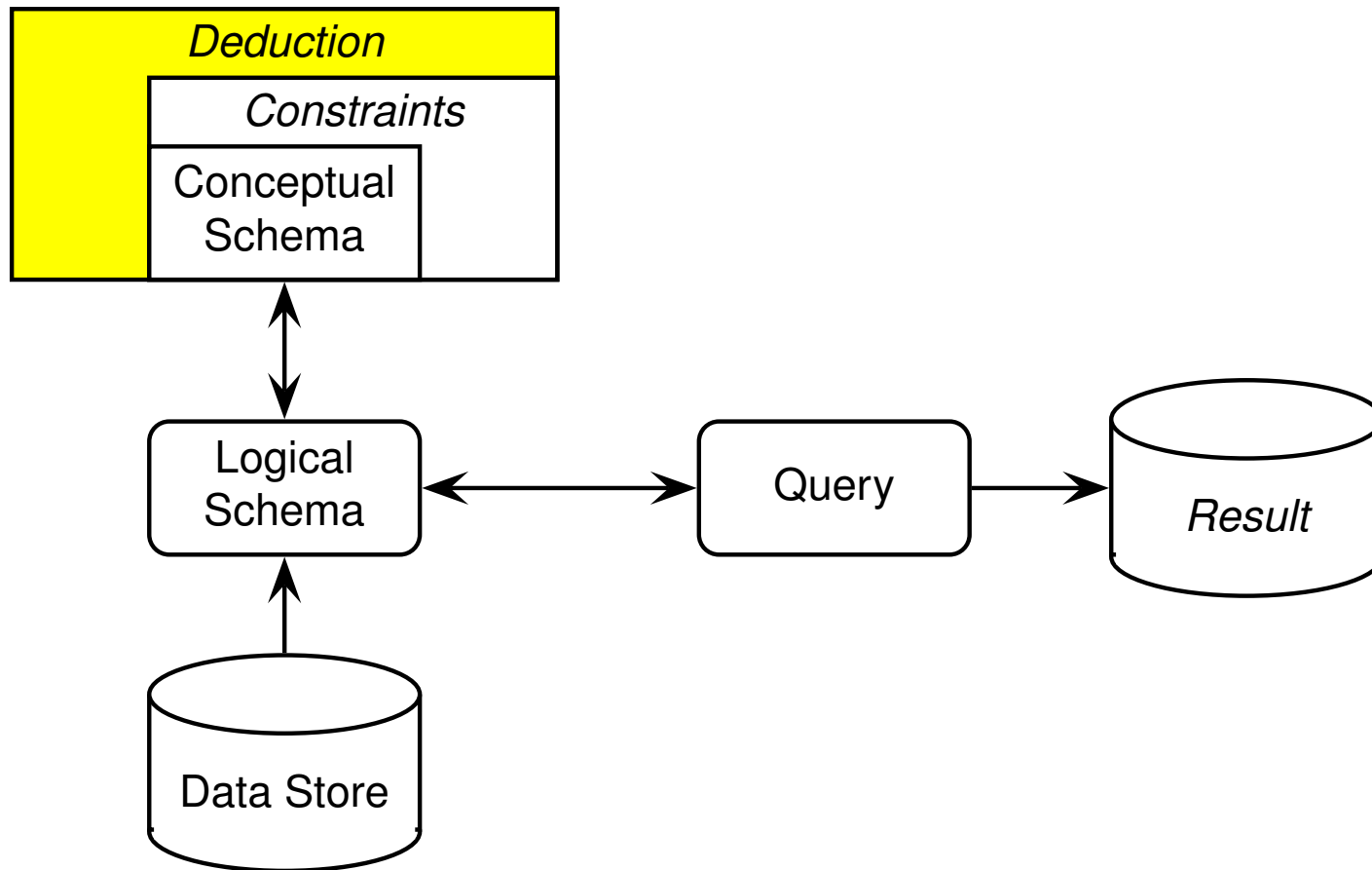
The role of a Conceptual Schema – revisited



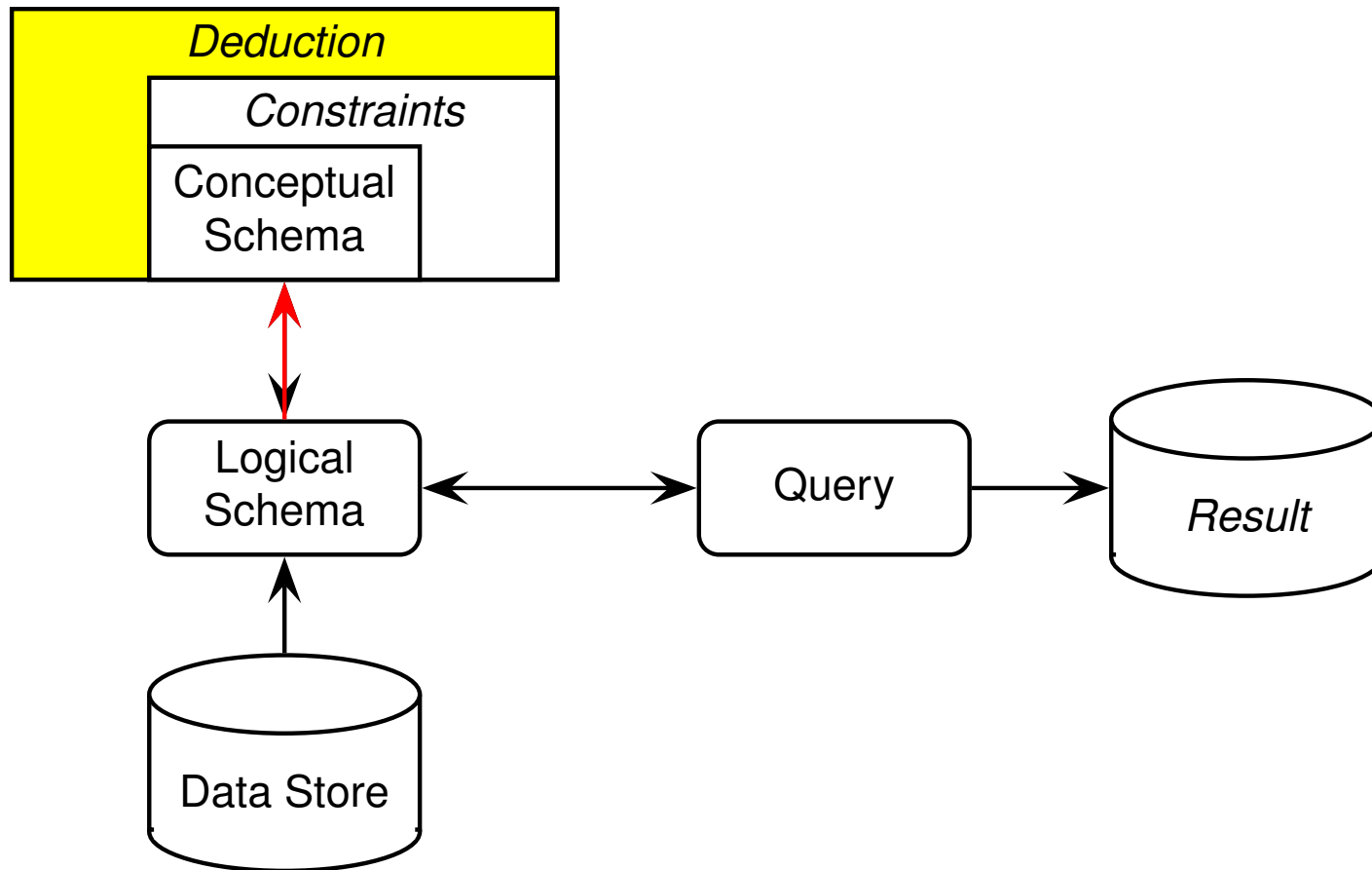
The role of a Conceptual Schema – revisited



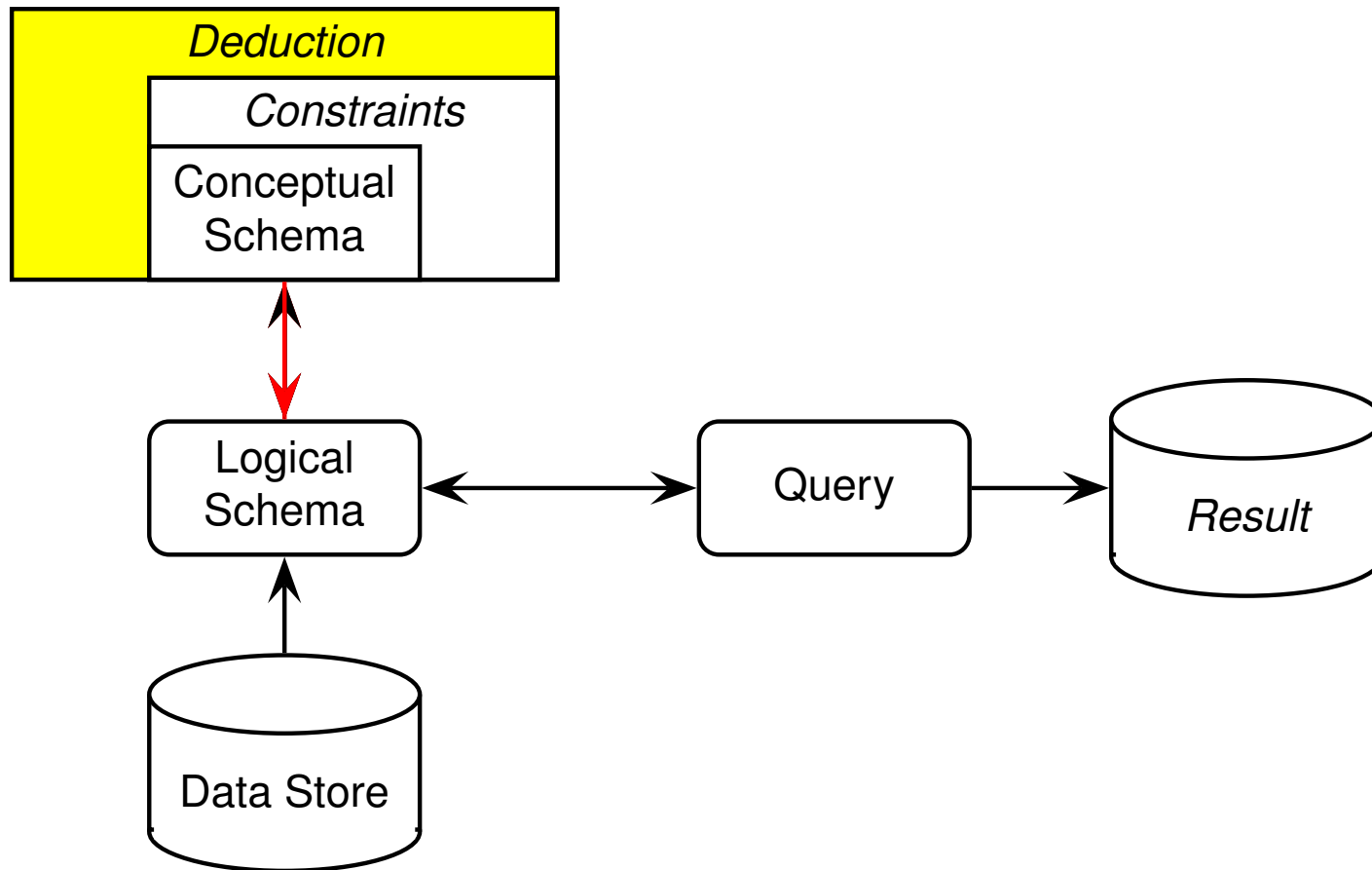
The role of a Conceptual Schema – revisited



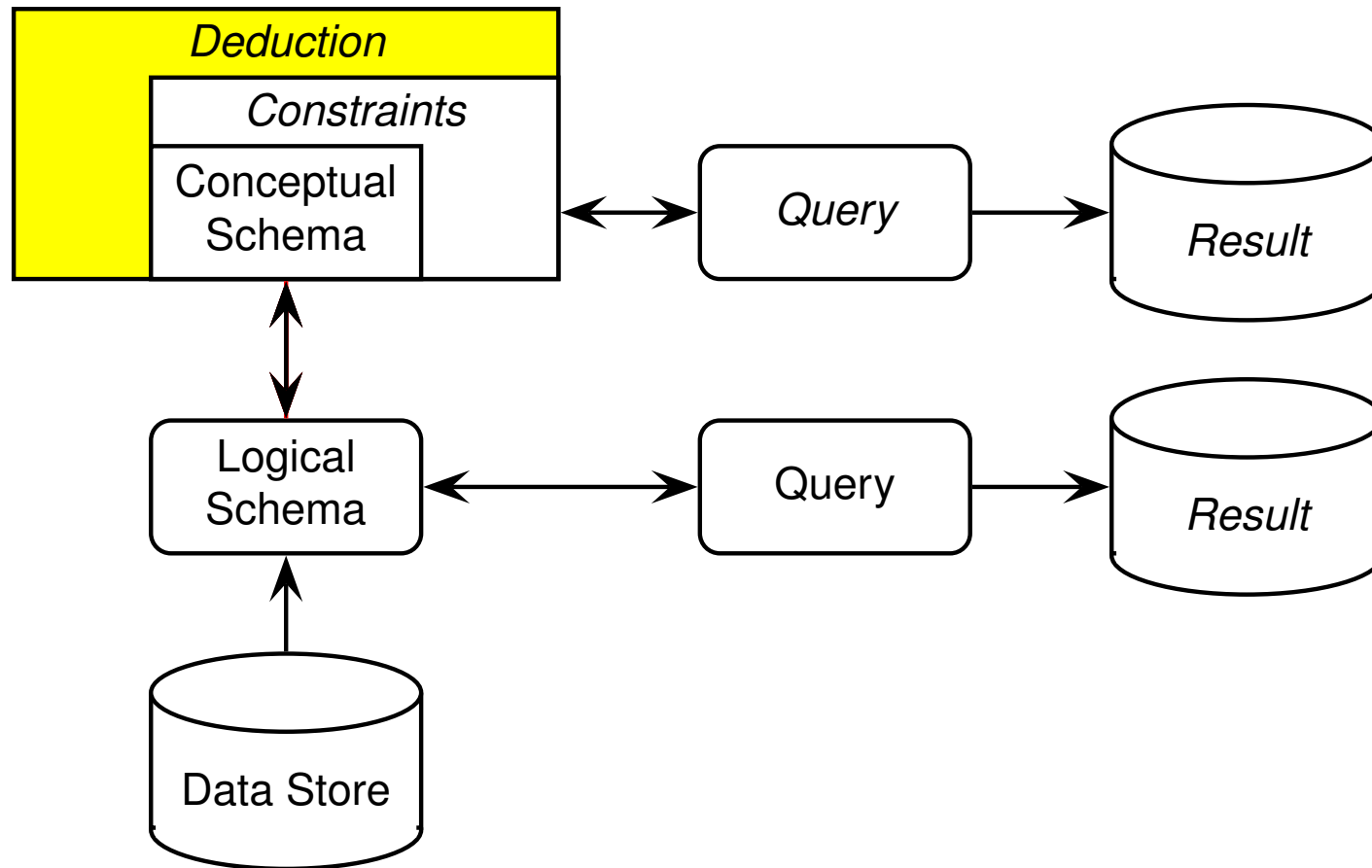
The role of a Conceptual Schema – revisited



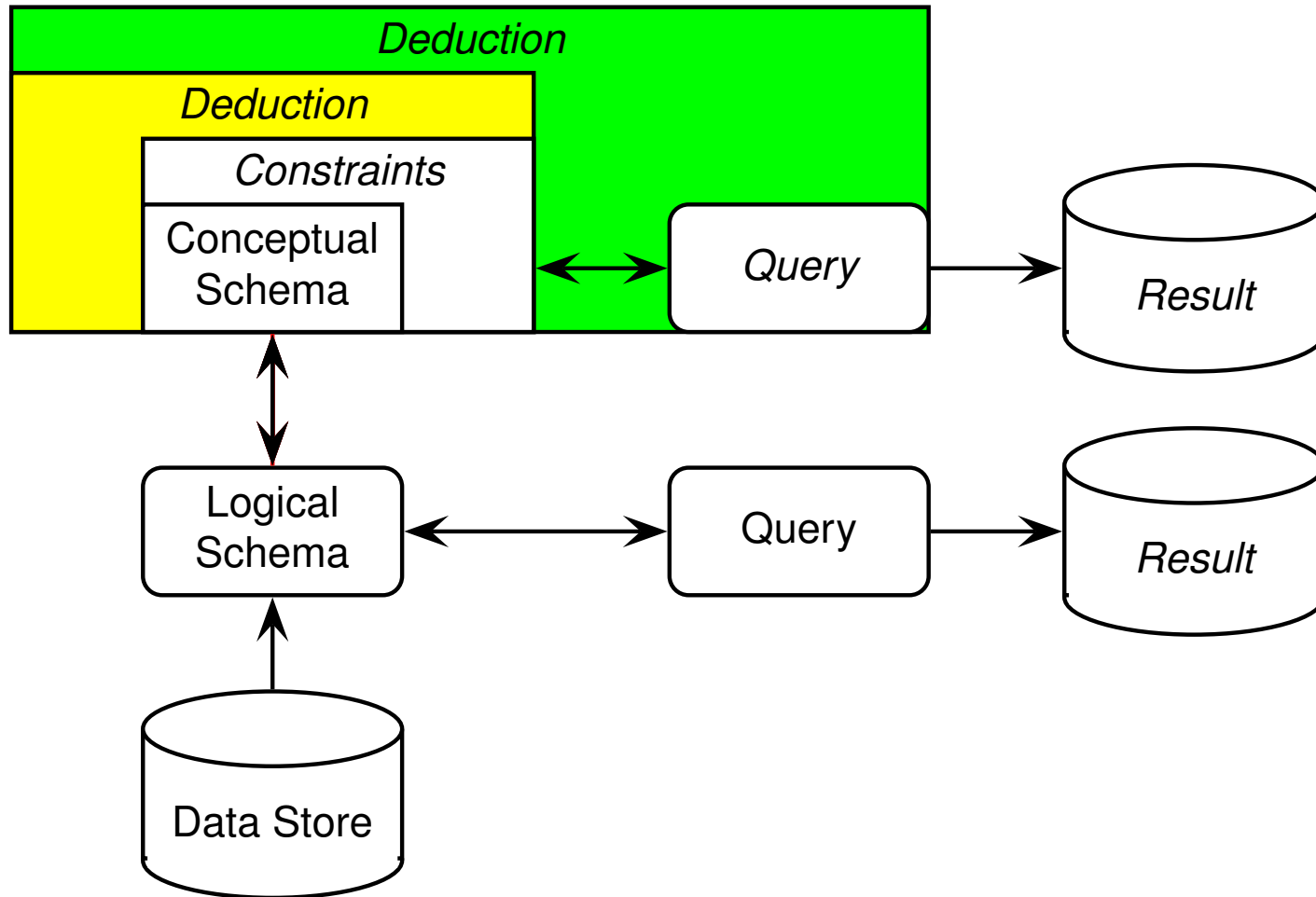
The role of a Conceptual Schema – revisited



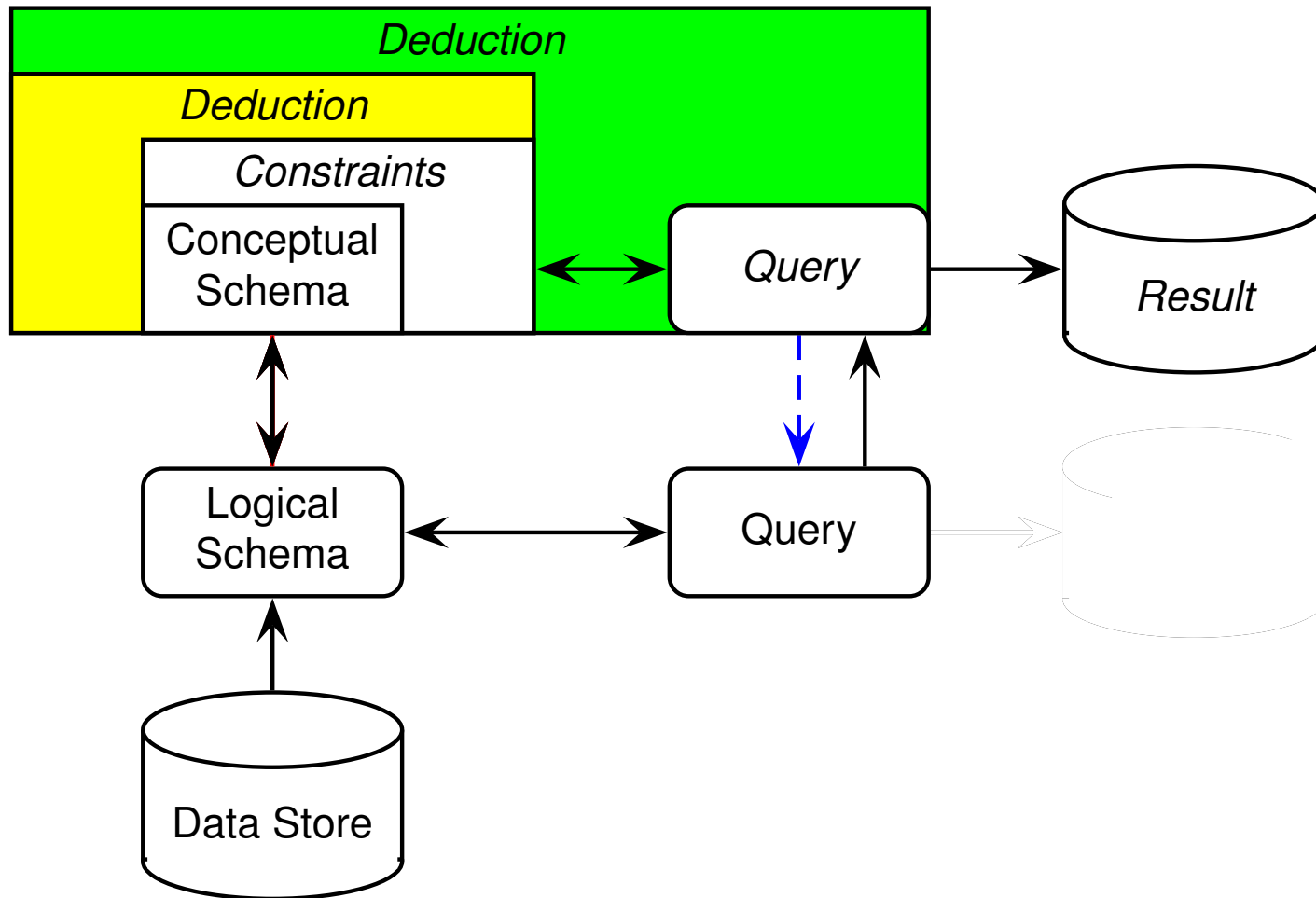
The role of a Conceptual Schema – revisited



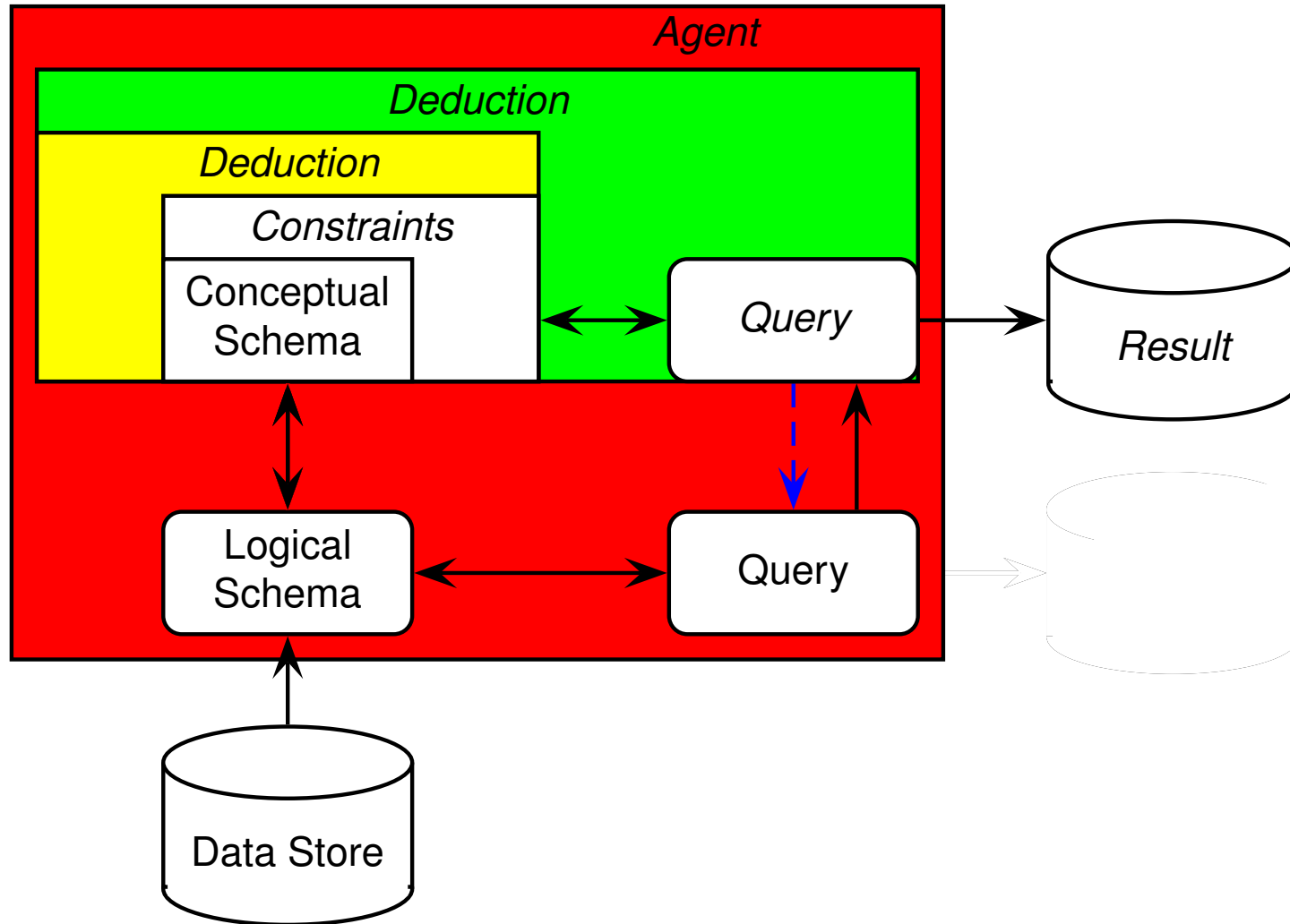
The role of a Conceptual Schema – revisited



The role of a Conceptual Schema – revisited



The role of a Conceptual Schema – revisited



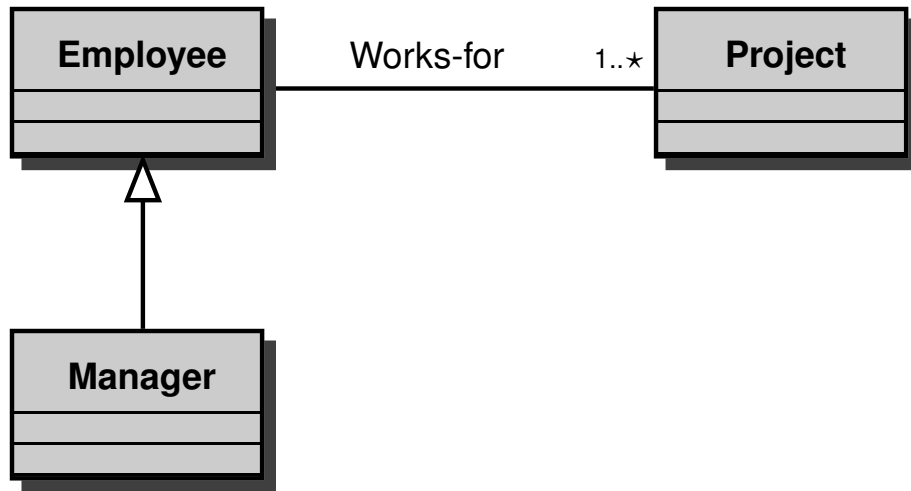
Queries with Ontologies: the DB assumption

- Basic assumption: **consistent** information with respect to the constraints introduced by the ontology
- DB assumption: **complete information about each term** appearing in the ontology
- *Problem*: answer a query over the ontology vocabulary

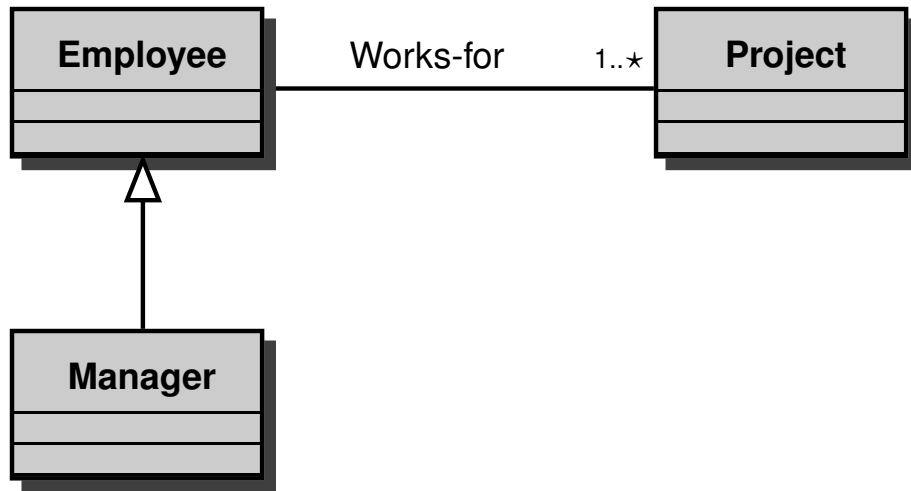
Queries with Ontologies: the DB assumption

- Basic assumption: **consistent** information with respect to the constraints introduced by the ontology
- DB assumption: **complete information about each term** appearing in the ontology
- *Problem*: answer a query over the ontology vocabulary
- *Solution*: use a standard DB technology (e.g., SQL, datalog, etc)

Example



Example



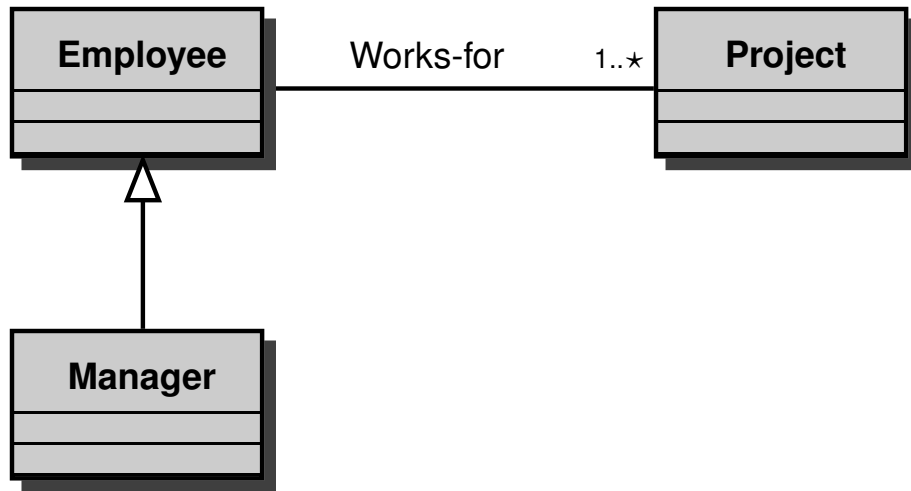
Employee = { **John, Mary, Paul** }

Manager = { **John, Paul** }

Works-for = { (**John, Prj-A**), (**Mary, Prj-B**) }

Project = { **Prj-A, Prj-B** }

Example



Employee = { **John, Mary, Paul** }

Manager = { **John, Paul** }

Works-for = { (**John, Prj-A**), (**Mary, Prj-B**) }

Project = { **Prj-A, Prj-B** }

$Q(X) \text{ :- Manager}(X), \text{ Works-for}(X, Y), \text{ Project}(Y)$

$\implies \{ \text{John} \}$

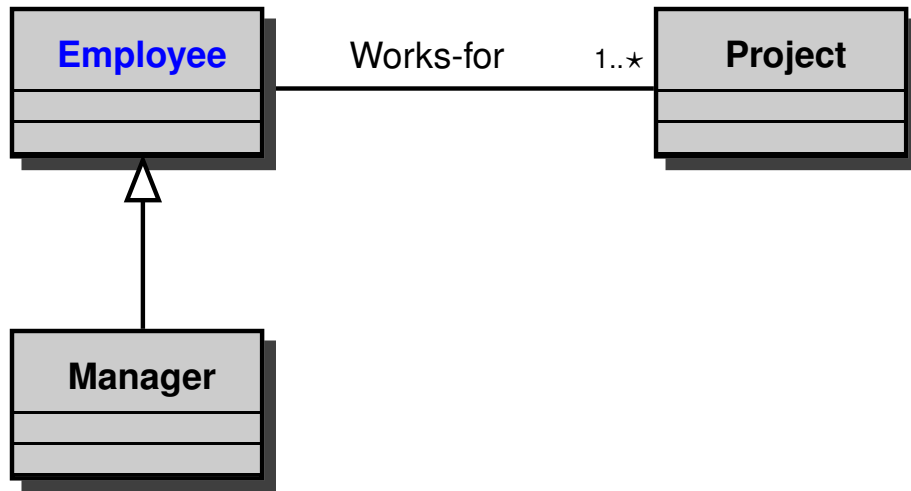
Weakening the DB assumption

- The DB assumption is against the principle that an ontology presents a richer vocabulary than the data stores.

Weakening the DB assumption

- The DB assumption is against the principle that an ontology presents a richer vocabulary than the data stores.
- Partial DB assumption: **complete information about some term** appearing in the ontology
- Standard DB technologies do not apply
- The query answering problem in this context is inherently complex

Simple Example

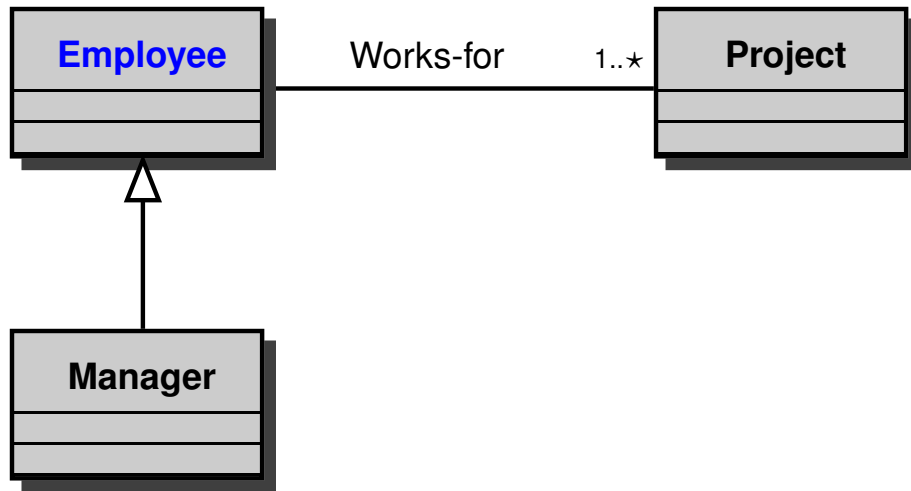


Manager = { **John, Paul** }

Works-for = { (**John, Prj-A**), (**Mary, Prj-B**) }

Project = { **Prj-A, Prj-B** }

Simple Example



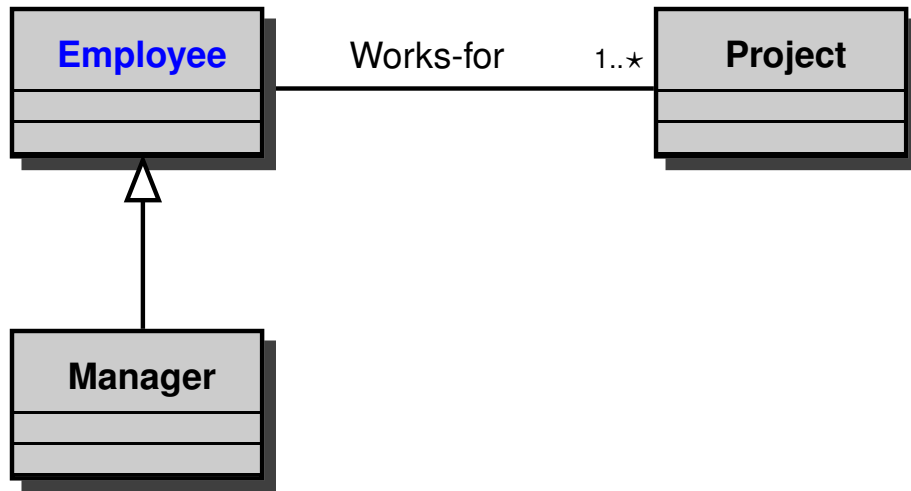
Manager = { **John, Paul** }

Works-for = { (**John, Prj-A**), (**Mary, Prj-B**) }

Project = { **Prj-A, Prj-B** }

$Q(X) :- \text{Employee}(X)$

Simple Example



Manager = { **John, Paul** }

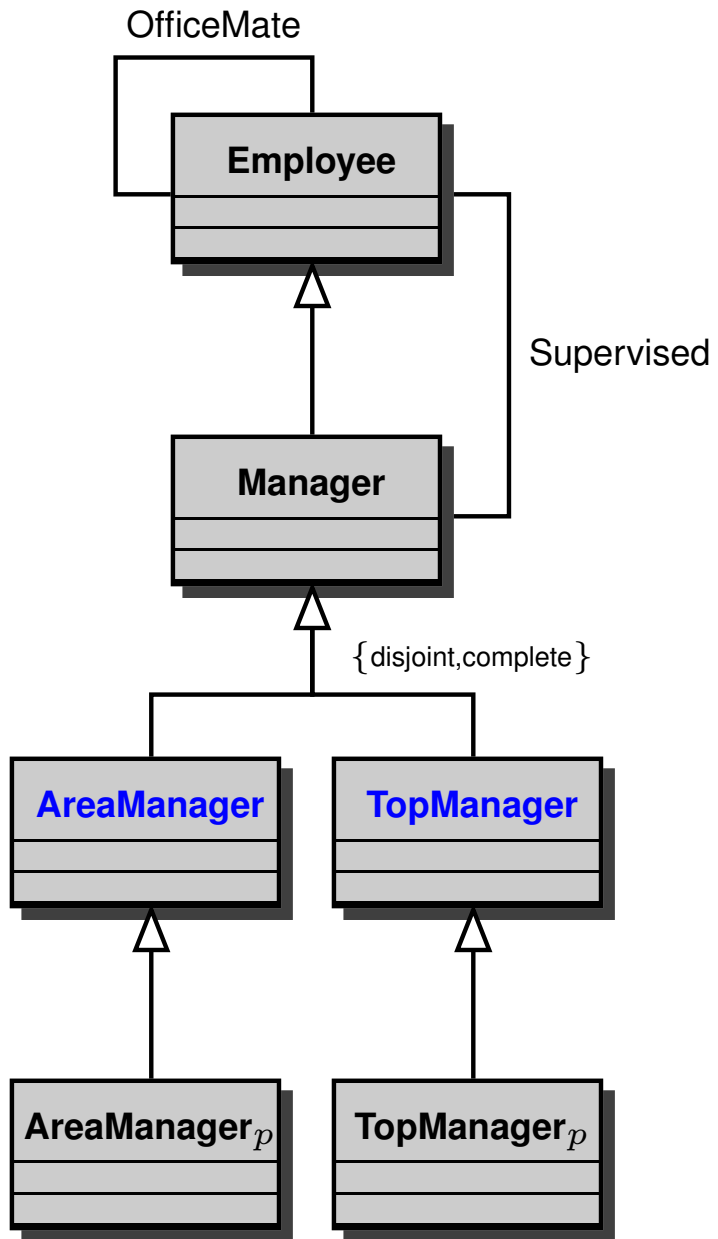
Works-for = { (**John, Prj-A**), (**Mary, Prj-B**) }

Project = { **Prj-A, Prj-B** }

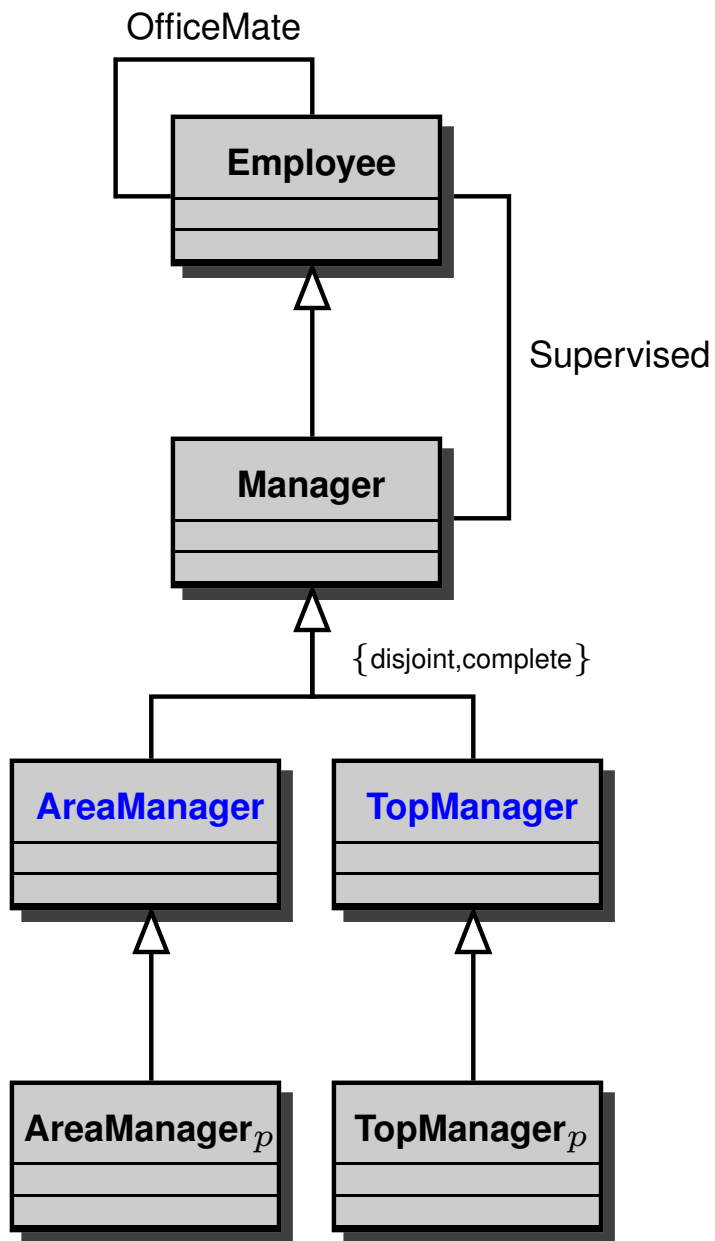
$Q(X) :- \text{Employee}(X)$

\Rightarrow { **John, Paul, Mary** }

Andrea's Example



Andrea's Example



Employee = { **Andrea, Paul, Mary, John** }

Manager = { **Andrea, Paul, Mary** }

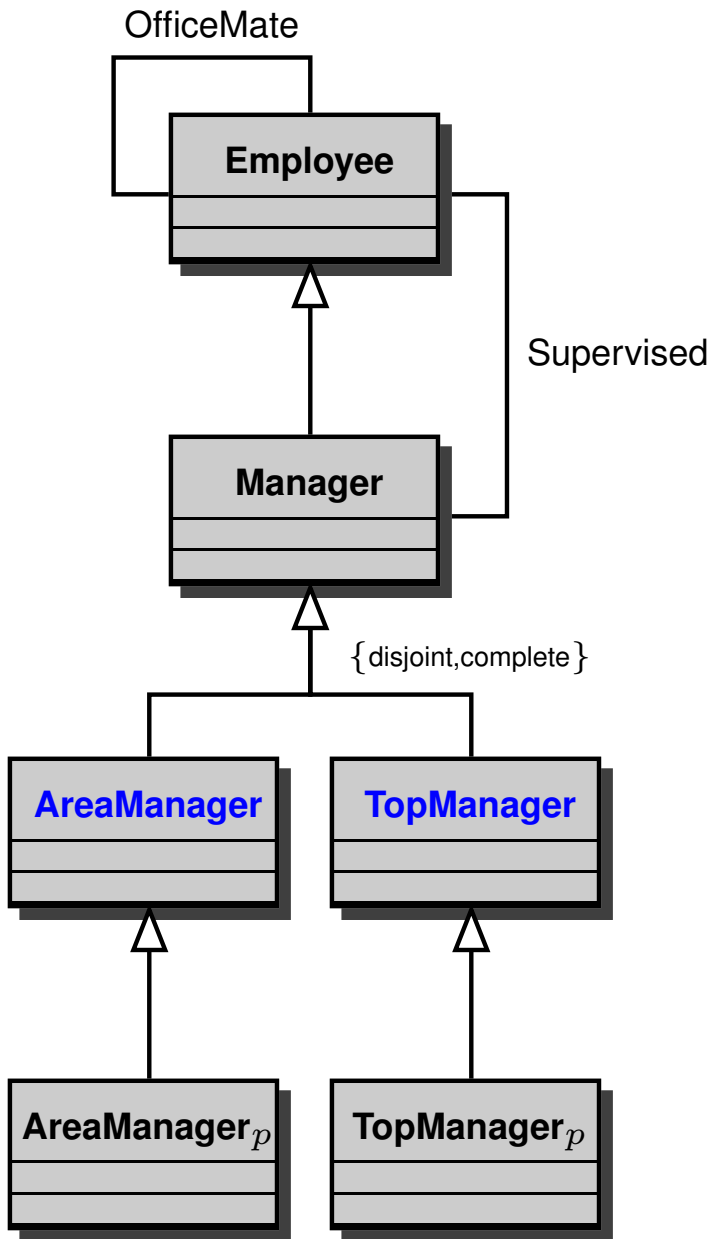
AreaManager_p = { **Paul** }

TopManager_p = { **Mary** }

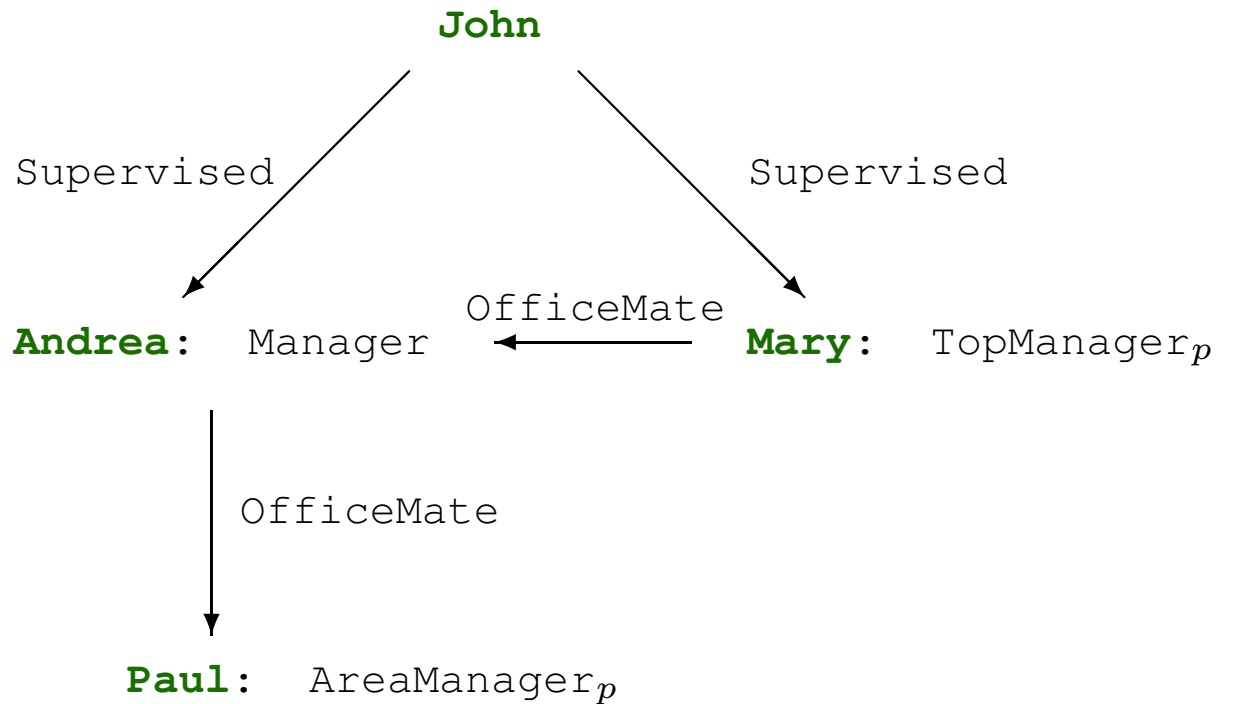
Supervised = { (**John, Andrea**), (**John, Mary**) }

OfficeMate = { (**Mary, Andrea**), (**Andrea, Paul**) }

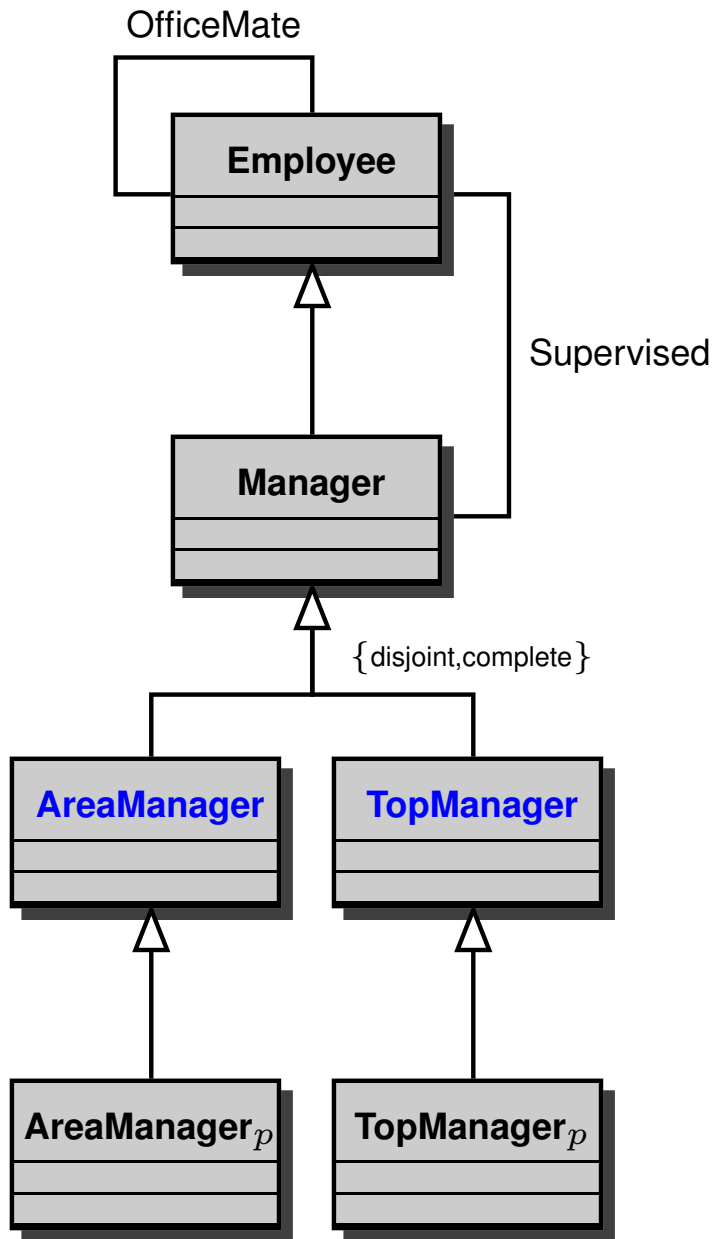
Andrea's Example



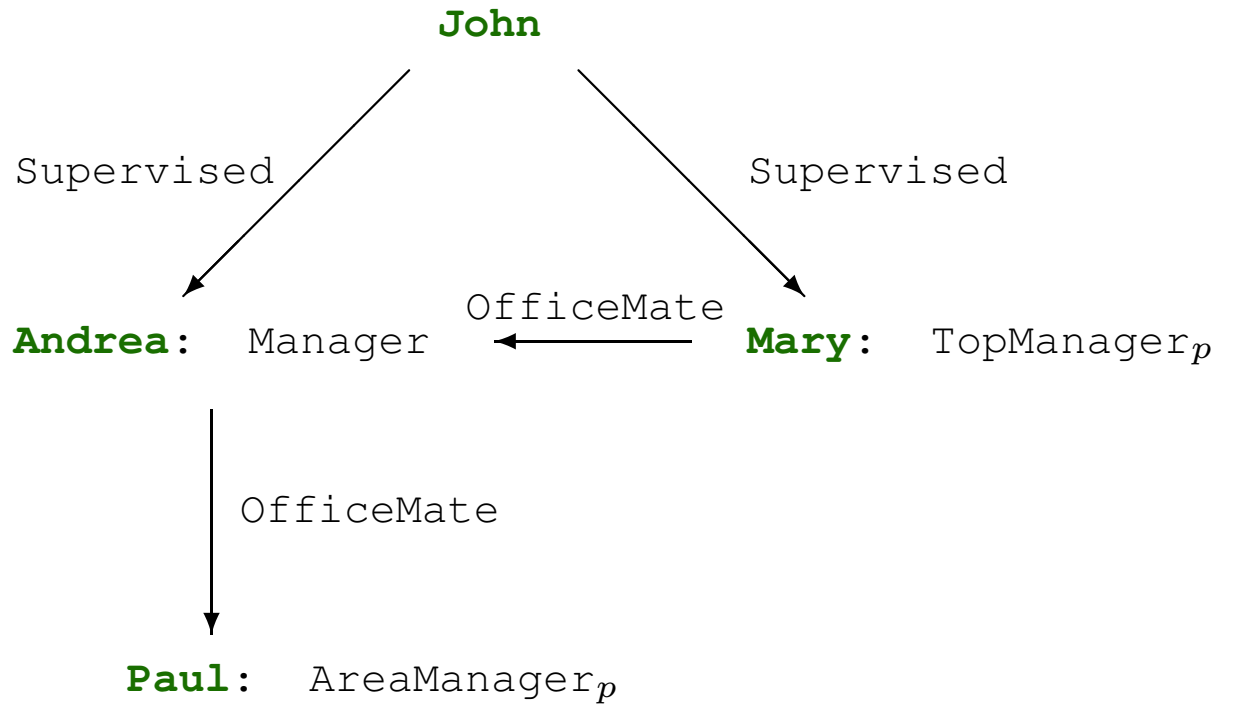
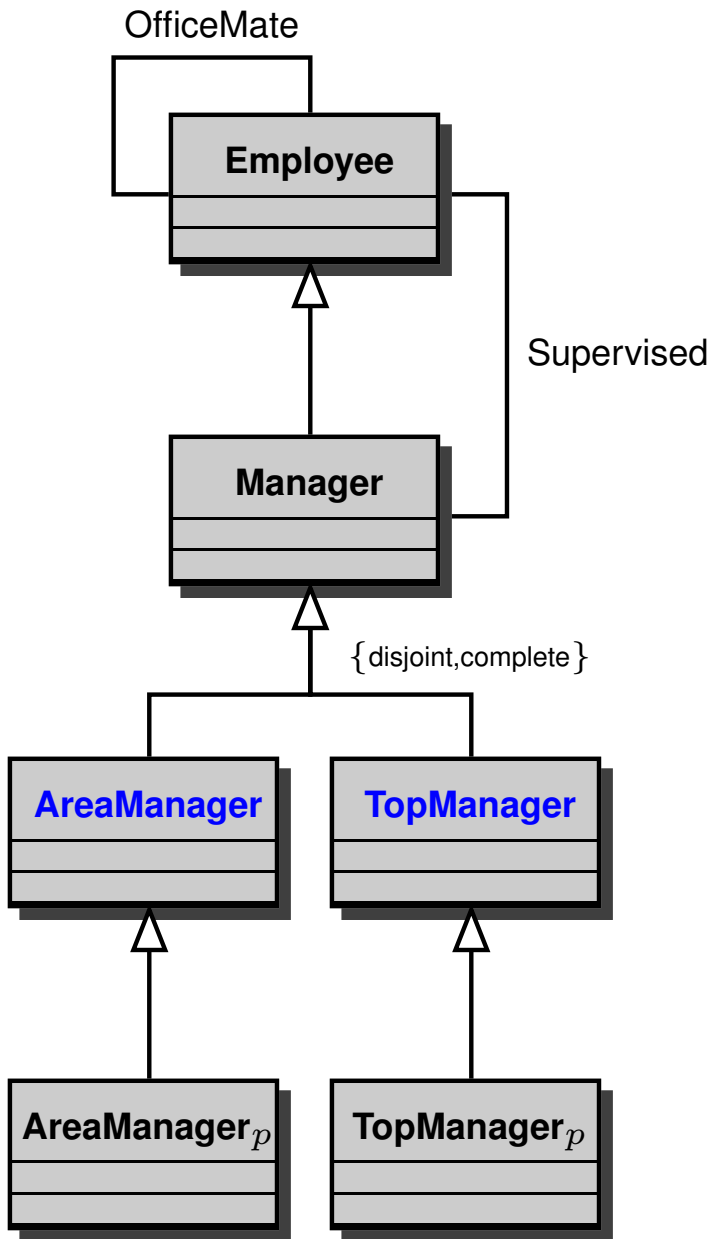
Employee = { **Andrea, Paul, Mary, John** }
 Manager = { **Andrea, Paul, Mary** }
 AreaManager_p = { **Paul** }
 TopManager_p = { **Mary** }
 Supervised = { (**John, Andrea**), (**John, Mary**) }
 OfficeMate = { (**Mary, Andrea**), (**Andrea, Paul**) }



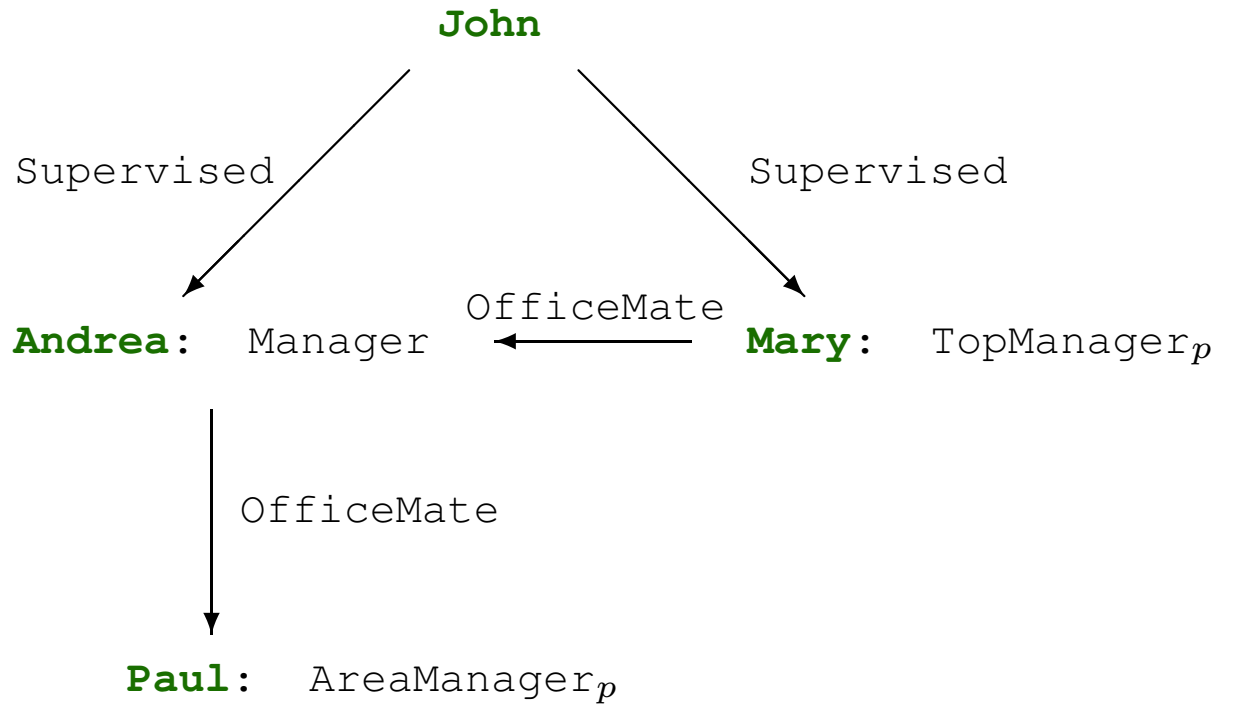
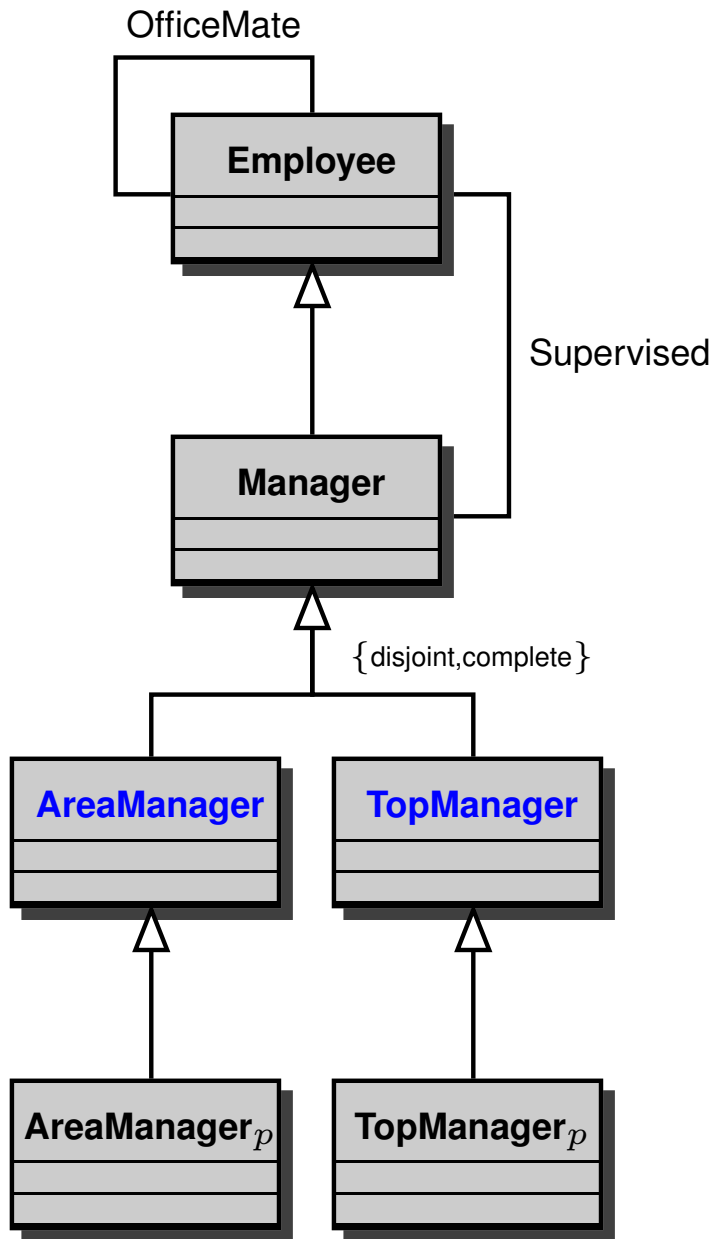
Andrea's Example (cont.)



Andrea's Example (cont.)

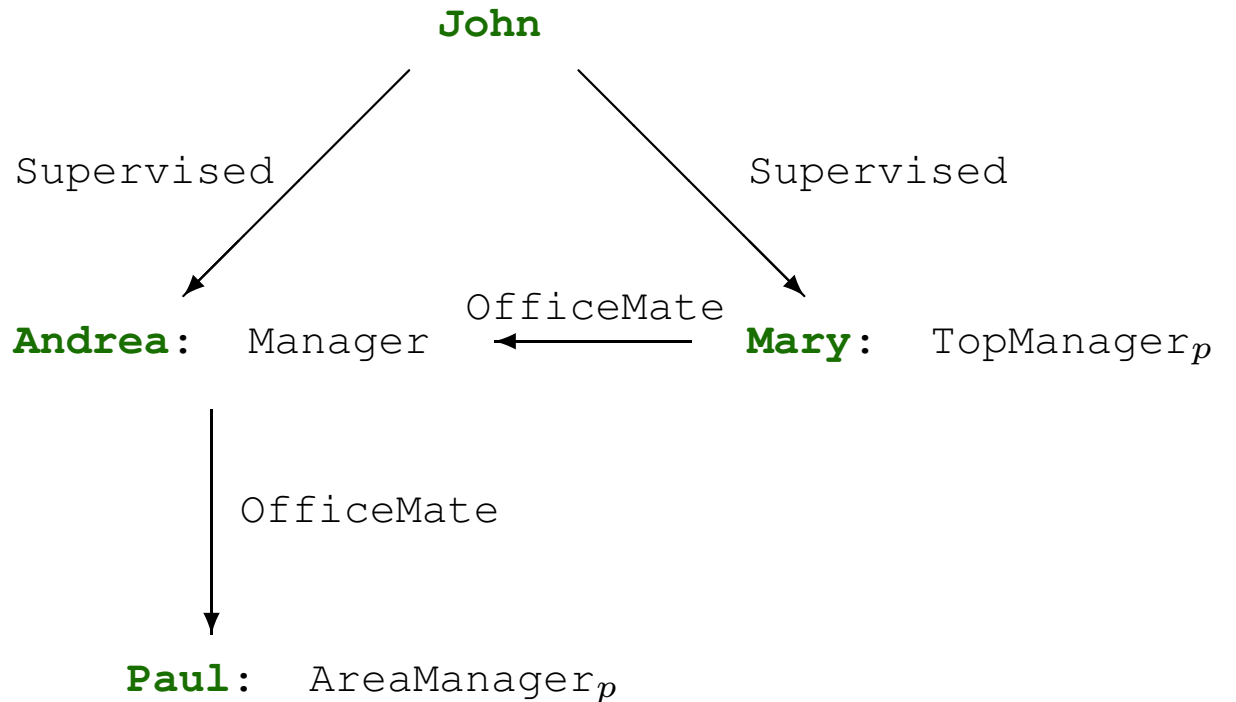
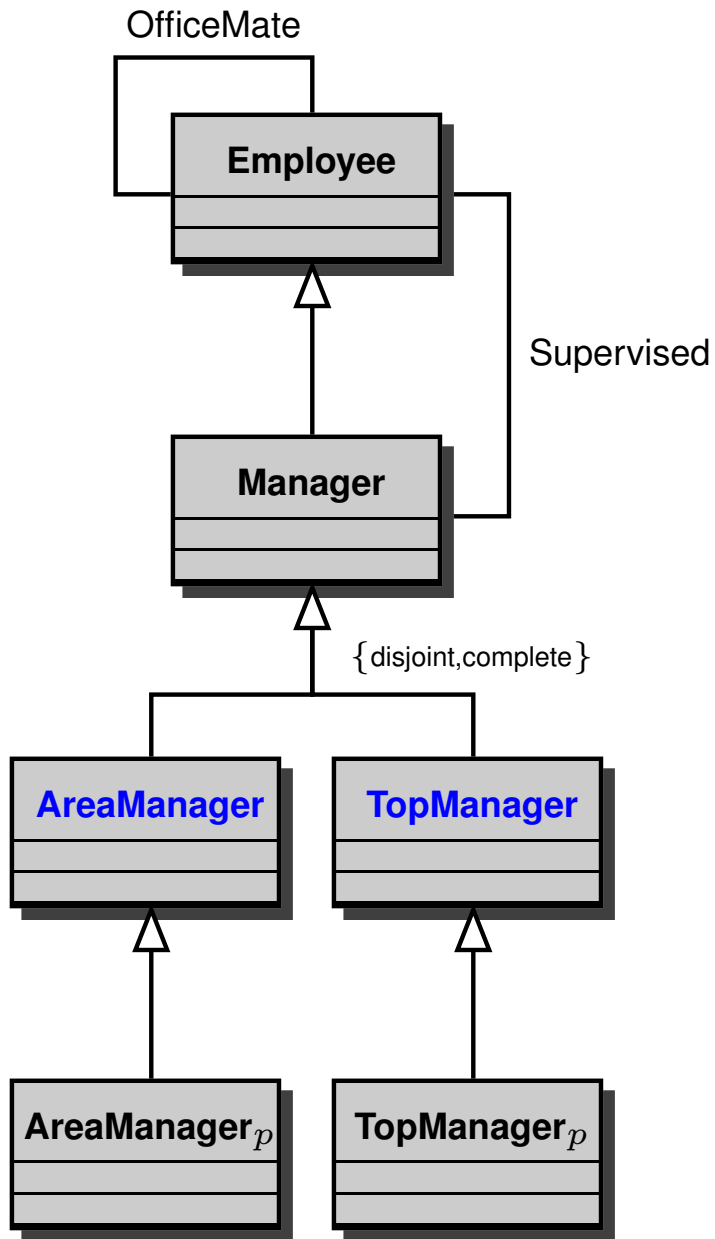


Andrea's Example (cont.)



$Q(X) :- \text{Supervised}(X, Y), \text{TopManager}(Y),$
 $\text{Officemate}(Y, Z), \text{AreaManager}(Z)$

Andrea's Example (cont.)



$Q(X) :- \text{Supervised}(X, Y), \text{TopManager}(Y),$
 $\text{Officemate}(Y, Z), \text{AreaManager}(Z)$

$\Rightarrow \{ \text{John} \}$

View based query processing

In general, the **mapping** between the ontology and the information source terms can be given in terms of a set of **views**:

View based query processing

In general, the **mapping** between the ontology and the information source terms can be given in terms of a set of **views**:

- **GAV** (global-as-view): a view over the information source is given for *some* term in the ontology;

View based query processing

In general, the **mapping** between the ontology and the information source terms can be given in terms of a set of **views**:

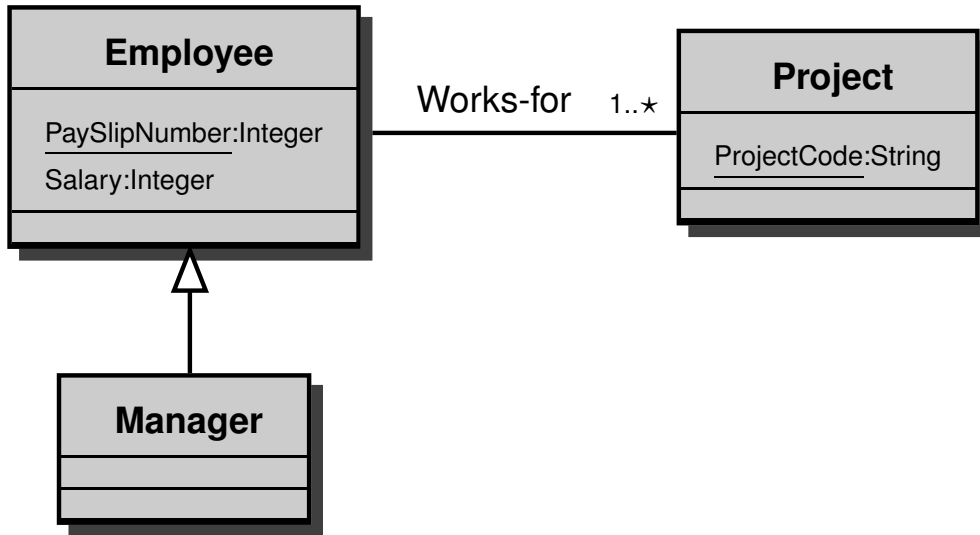
- **GAV** (global-as-view): a view over the information source is given for *some* term in the ontology;
 - an ER schema can be easily mapped to its corresponding relational schema in normal form via a total GAV mapping.

View based query processing

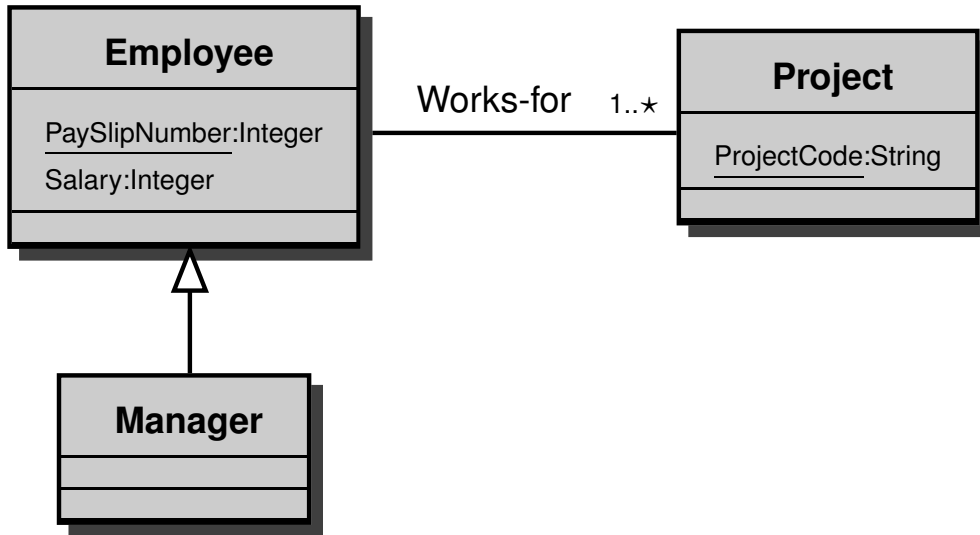
In general, the **mapping** between the ontology and the information source terms can be given in terms of a set of **views**:

- **GAV** (global-as-view): a view over the information source is given for *some* term in the ontology;
 - an ER schema can be easily mapped to its corresponding relational schema in normal form via a total GAV mapping.
- **LAV** (local-as-view): a view over the ontology terms is given for each term in the information source;
- **GLAV**: mixed from the above.

Total GAV mapping



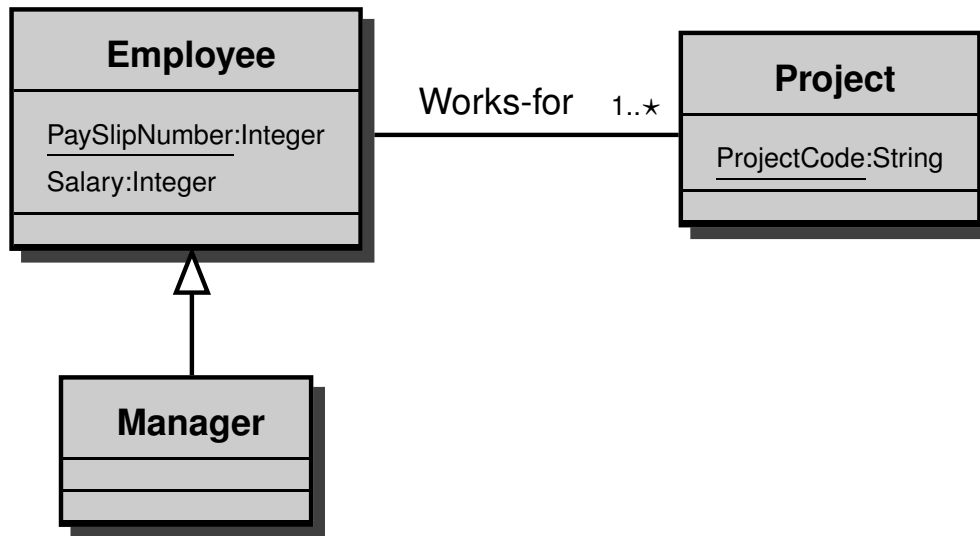
Total GAV mapping



NF-`Employee` (PaySlipNumber, Salary, ManagerP)

NF-`Works-for` (PaySlipNumber, ProjectCode)

Total GAV mapping



NF-Employee (PaySlipNumber, Salary, ManagerP)

NF-Works-for (PaySlipNumber, ProjectCode)

Employee (X) :- NF-Employee (X, Y, Z)

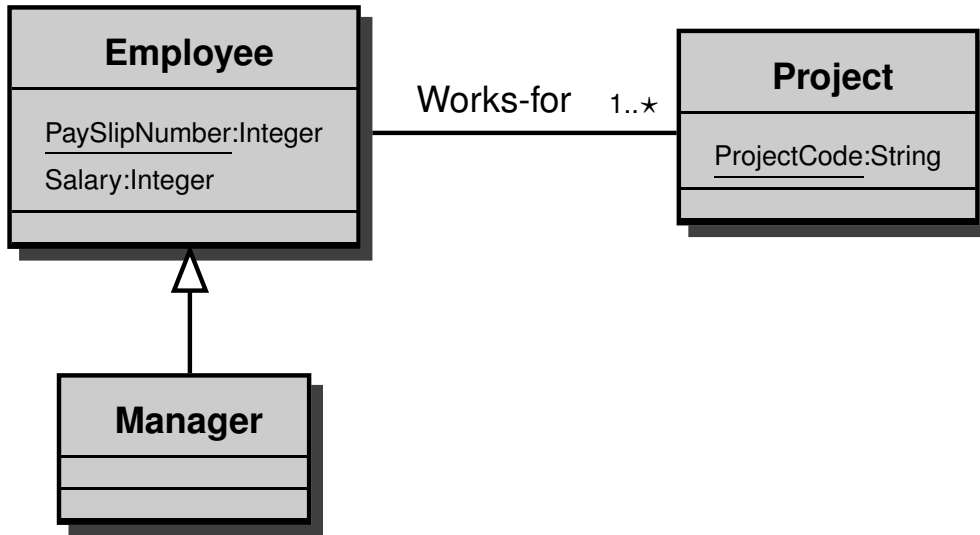
Manager (X) :- NF-Employee (X, Y, true)

Salary (X, Y) :- NF-Employee (X, Y, Z)

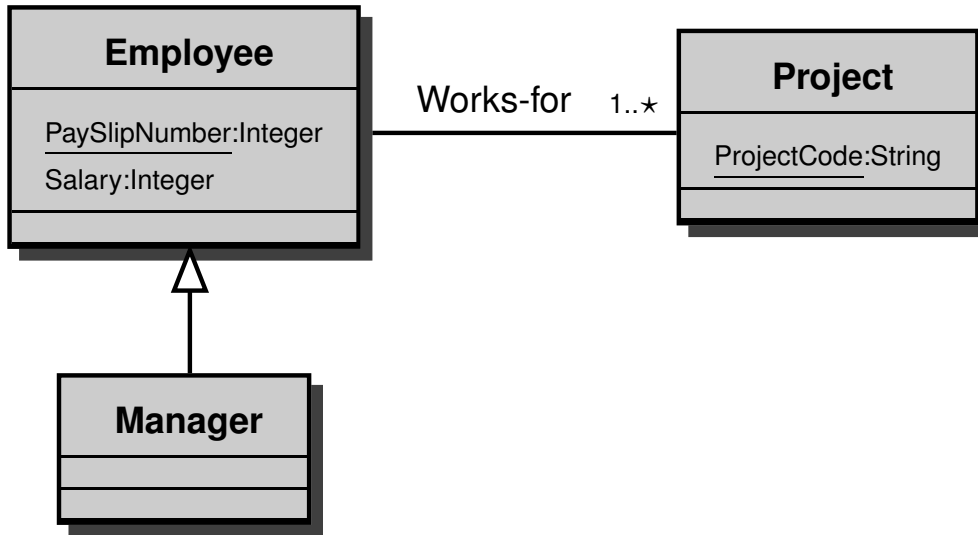
Works-for (X, Y) :- NF-Works-for (X, Y)

Project (X) :- NF-Works-for (X, Y)

LAV mapping



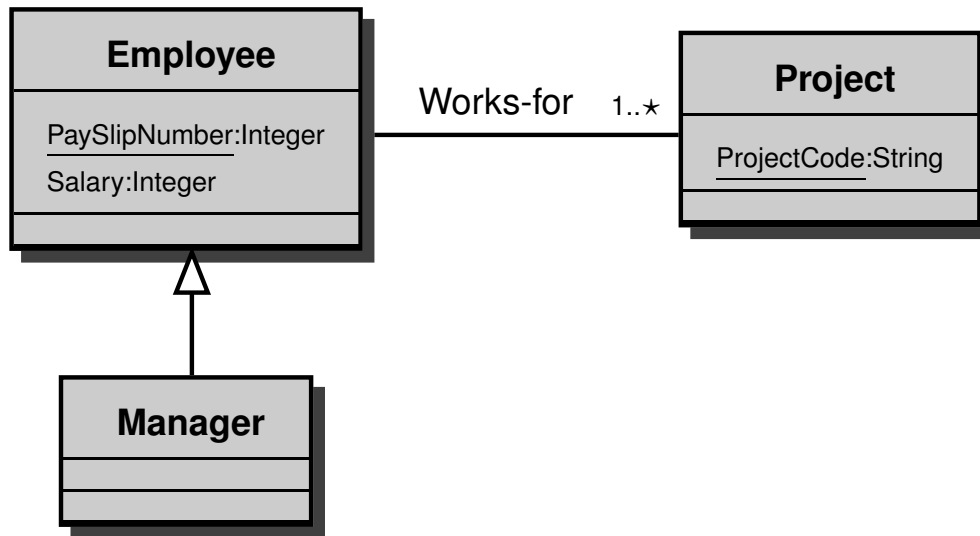
LAV mapping



NF-Employee (PaySlipNumber, Salary, ManagerP)

NF-Works-for (PaySlipNumber, ProjectCode)

LAV mapping



NF-Employee (PaySlipNumber, Salary, ManagerP)

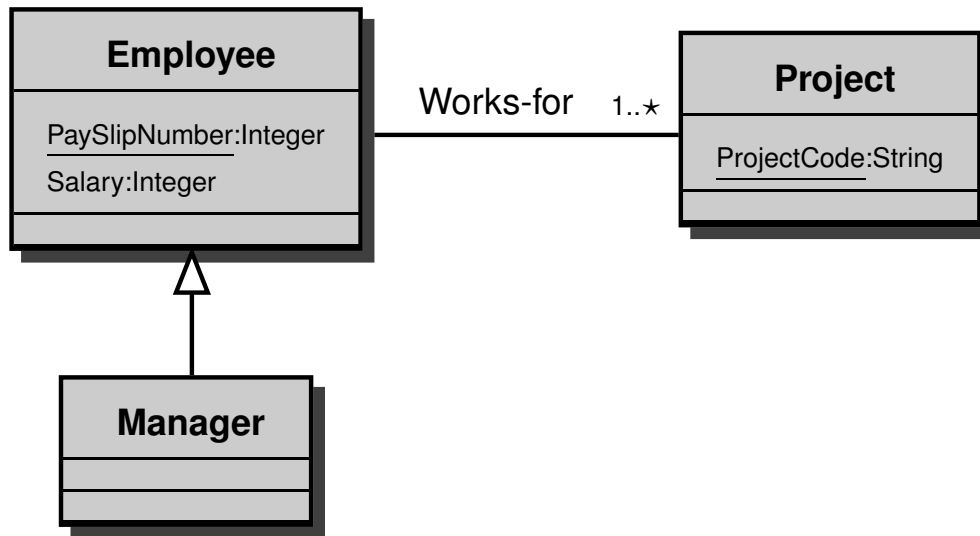
NF-Works-for (PaySlipNumber, ProjectCode)

NF-Employee(X, Y, Z) :- Manager(X), Salary(X, Y), Z=true

NF-Employee(X, Y, Z) :- Employee(X), ¬Manager(X), Salary(X, Y), Z=false

NF-Works-for(X, Y) :- Works-for(X, Y)

Queries with LAV mapping



NF-Employee (PaySlipNumber, Salary, ManagerP)

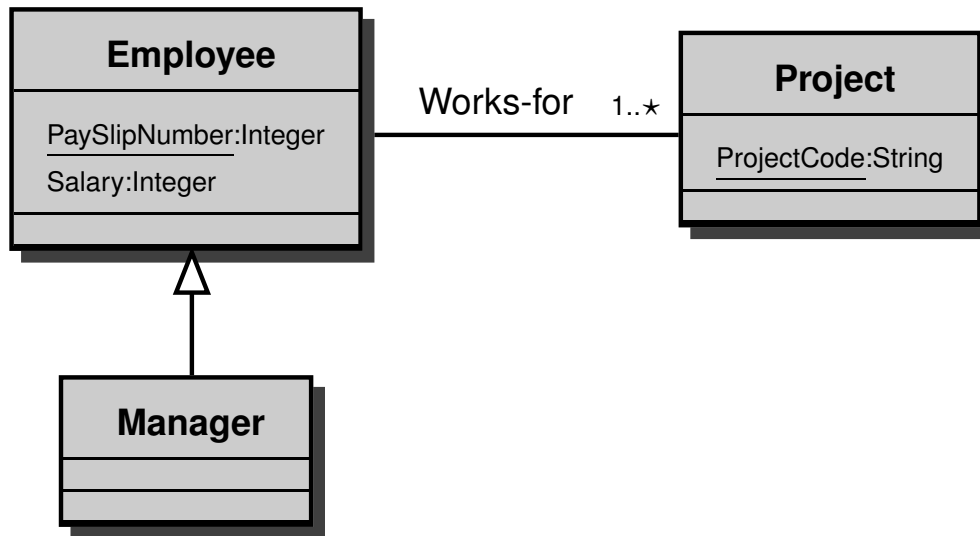
NF-Works-for (PaySlipNumber, ProjectCode)

NF-Employee (X, Y, Z) :- Manager (X), Salary (X, Y), Z=true

NF-Employee (X, Y, Z) :- Employee (X), -Manager (X), Salary (X, Y), Z=false

NF-Works-for (X, Y) :- Works-for (X, Y)

Queries with LAV mapping



NF-`Employee` (PaySlipNumber, Salary, ManagerP)

NF-`Works-for` (PaySlipNumber, ProjectCode)

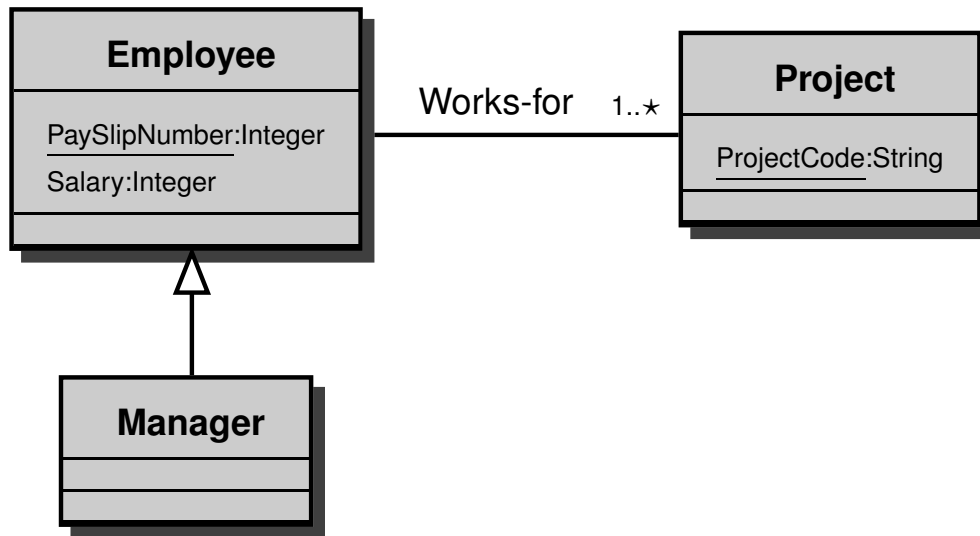
NF-`Employee` (X, Y, Z) :- Manager (X), Salary (X, Y), Z=true

NF-`Employee` (X, Y, Z) :- Employee (X), -Manager (X), Salary (X, Y), Z=false

NF-`Works-for` (X, Y) :- Works-for (X, Y)

Q (X) :- Manager (X), Works-for (X, Y), Project (Y)

Queries with LAV mapping



NF-Employee (PaySlipNumber, Salary, ManagerP)

NF-Works-for (PaySlipNumber, ProjectCode)

NF-Employee (X, Y, Z) :- Manager (X), Salary (X, Y), Z=true

NF-Employee (X, Y, Z) :- Employee (X), -Manager (X), Salary (X, Y), Z=false

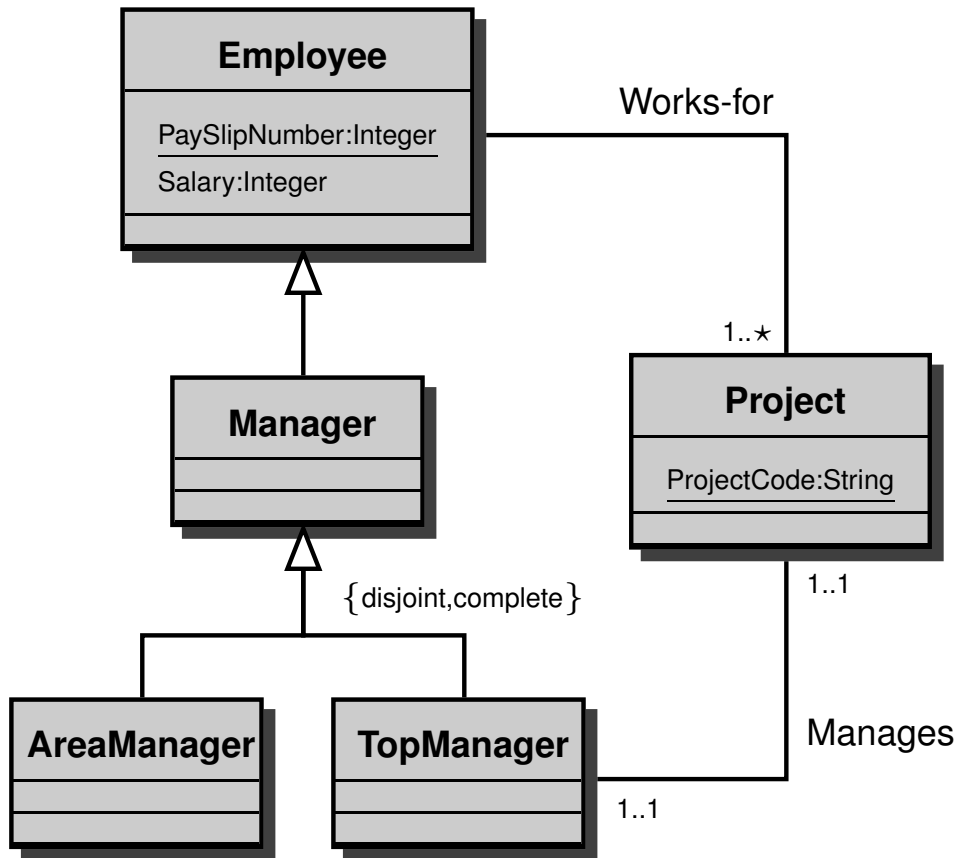
NF-Works-for (X, Y) :- Works-for (X, Y)

Q (X) :- Manager (X), Works-for (X, Y), Project (Y)

⇒ Q' (X) :- NF-Employee (X, Y, true), NF-Works-for (X, Z)

Reasoning over queries

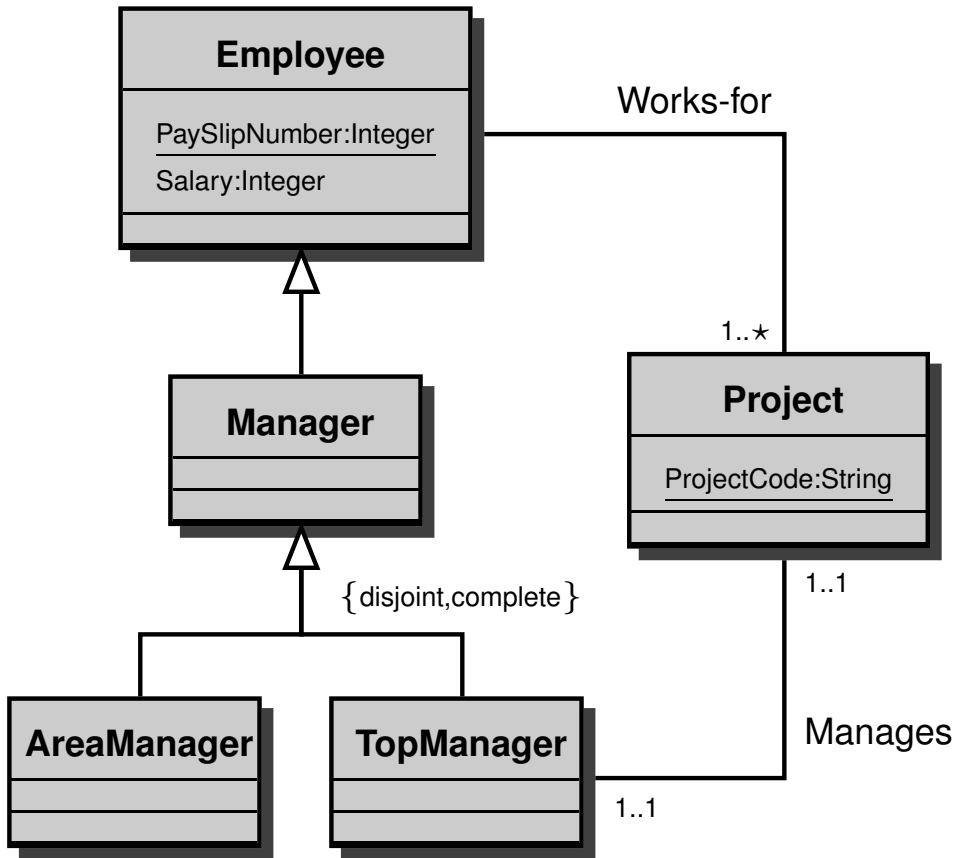
$Q(X, Y) :- \text{Employee}(X), \text{Works-for}(X, Y), \text{Manages}(X, Y)$



$\forall x. \text{Manager}(x) \rightarrow \forall y. \neg \text{WORKS-FOR}(x, y)$

Reasoning over queries

$Q(X, Y) :- \text{Employee}(X), \text{Works-for}(X, Y), \text{Manages}(X, Y)$



$\forall x. \text{Manager}(x) \rightarrow \forall y. \neg \text{WORKS-FOR}(x, y)$



INCONSISTENT QUERY!

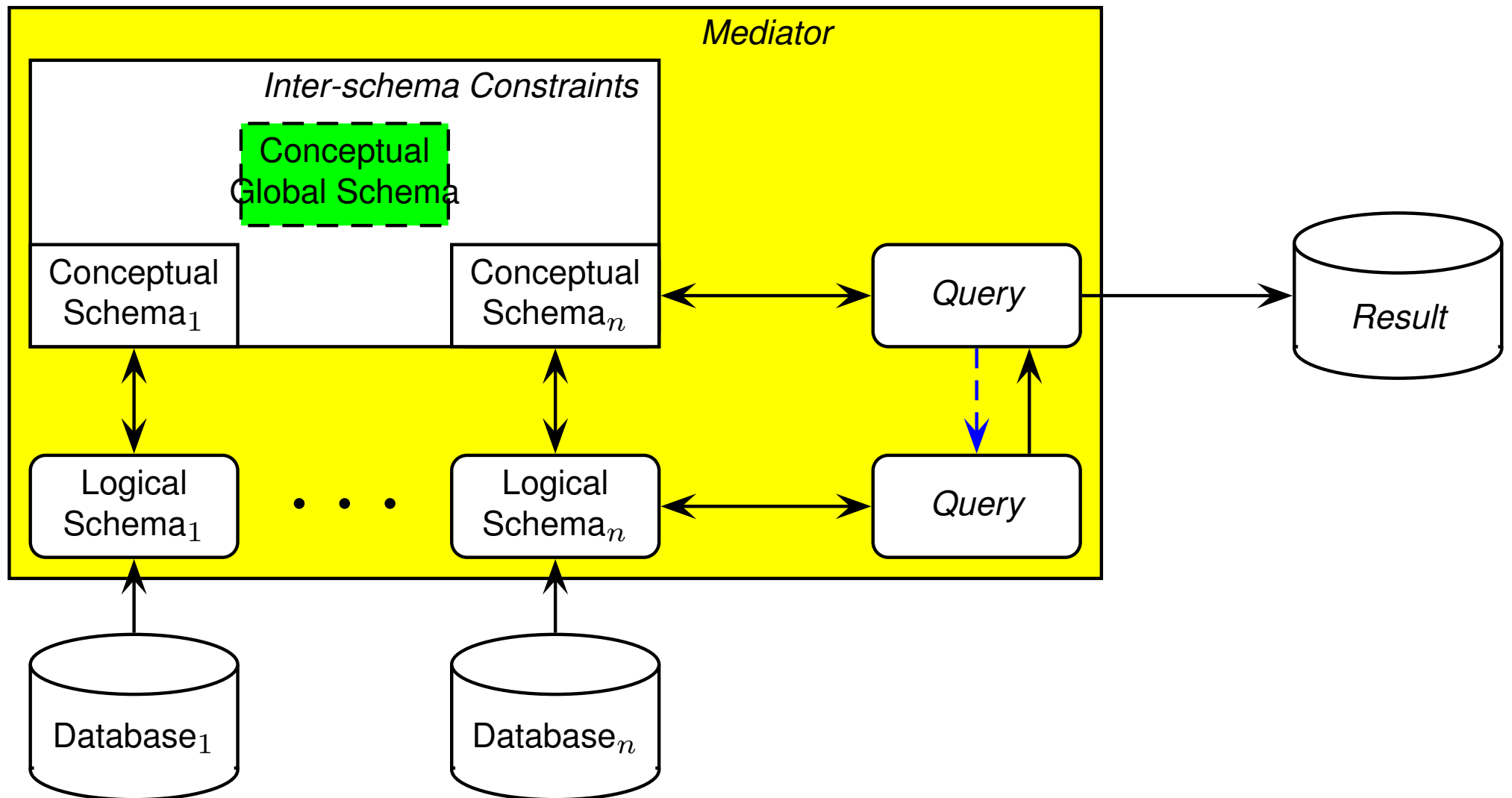
Summary

- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- Queries with an Ontology
- Ontology Integration

Usefulness of View-based Query Processing

- In data integration, the views represent the only information sources accessible to answer a query.
- A data warehouse can be seen as a set of materialised views, and, therefore, query processing reduces to view-based query answering.
- In query optimisation, view-based query processing is relevant because using the views may speed up query processing.
- Since the views provide partial knowledge on the database, view-based query processing can be seen as a special case query answering with incomplete information.

Mediator Architecture for Ontology Integration



Local-as-view vs. Global-as-view

Local-as-view

- High modularity and reusability (when a source changes, only its view definition is changed).
- Relationships between sources can be inferred.
- Computationally more difficult (query reformulation).

Global-as-view

- Whenever the source changes or a new one is added, the view needs to be reconsidered.
- Needs to understand the relationships between the sources.
- Query processing sometimes easy (unfolding), when the ontology is *very simple*. Otherwise it requires sophisticated query evaluation procedures.

Possible scenarios

- Empty ontology / very simple Ontology
 - Global-as-view

 - Local-as-view

- Full Ontology / Integrity Constraints
 - Global-as-view

 - Local-as-view

Possible scenarios

- Empty ontology / very simple Ontology
 - Global-as-view
 - The problem reduces to standard DB technology.
 - Can not express Ontology Integration needs.
 - Not modular.
 - Local-as-view
 - “Standard” view-based query processing.
 - Can express only few Ontology Integration needs.
 - Modular.
- Full Ontology / Integrity Constraints
 - Global-as-view
 - Local-as-view

Possible scenarios

- Empty ontology / very simple Ontology
 - Global-as-view
 - The problem reduces to standard DB technology.
 - Can not express Ontology Integration needs.
 - Not modular.
 - Local-as-view
 - “Standard” view-based query processing.
 - Can express only few Ontology Integration needs.
 - Modular.
- Full Ontology / Integrity Constraints
 - Global-as-view
 - Requires sophisticated query evaluation procedures (involving deduction).
 - Can express Ontology Integration needs.
 - Not modular.
 - Local-as-view

Possible scenarios

- Empty ontology / very simple Ontology
 - Global-as-view
 - The problem reduces to standard DB technology.
 - Can not express Ontology Integration needs.
 - Not modular.
 - Local-as-view
 - “Standard” view-based query processing.
 - Can express only few Ontology Integration needs.
 - Modular.
- Full Ontology / Integrity Constraints
 - Global-as-view
 - Requires sophisticated query evaluation procedures (involving deduction).
 - Can express Ontology Integration needs.
 - Not modular.
 - Local-as-view
 - View-based query processing under constraints.
 - Can express Ontology Integration needs.
 - Modular.

Current Practice

- Most implemented ontology based systems:

Current Practice

- Most implemented ontology based systems:
 - either assume no Ontology or a very simple Ontology with a global-as-view approach,

Current Practice

- Most implemented ontology based systems:
 - either assume no Ontology or a very simple Ontology with a global-as-view approach,
 - or include an Ontology or Integrity Constraints in their framework, but adopt a naive query evaluation procedure, based on query unfolding: no correctness of the query answering can be proved.

Conclusions

Conclusions

Do you have an ontology in your application?

Conclusions

Do you have an ontology in your application?

Pay attention!