

Languages for Ontologies

- There are several symbolic languages for representing ontologies
 - Languages of **Description Logic**
 - Languages for the **Semantic Web**: **RDF-S**, **OWL**, etc.
 - Extensions and relatives of Datalog: **Datalog[±]** languages
- Most frequently, in ontologies one uses 1-ary and 2-ary predicates (Datalog[±] gives more flexibility)

- Example: Introduce basic predicates:



- Unary predicates for **concepts**: *Employee(·)*, *Manager(·)*
- Binary predicates for **roles**: *BossOf(·,·)*, *ReportsTo(·,·)*

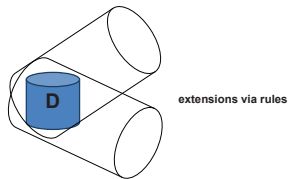
Symbolic statements go into the ontology

For example: $\forall x \forall y (BossOf(x, y) \rightarrow Employee(x))$, etc.

- Example of Datalog⁺: (sometimes denoted Datalog[∃])
A DB of employers and employees extended with rules and constraints
- Impose an **tuple-generating dependency** (TGD) (aka. inclusion dependency)
“every manager is an employee”
 $Employee(x) \leftarrow Manager(x)$
- Another TGD:
“every manager supervises someone”
 $\exists y Supervises(x, y) \leftarrow Manager(x)$
- Impose an IC: *“employees are not employers”*
As **negative constraint** (NC):
 $\perp \leftarrow Employee(x), Employer(x)$
(symbol \perp on the LHS is always false)
- A key constraint:
“every employee is supervised by at most one manager”
 $x = x' \leftarrow Supervises(x, y), Supervises(x', y)$

- The first is a usual Datalog rule, but not the last three
- New wrt. Datalog: **existential quantifiers** in the consequents, and use of **constraints** (the last two above) Hence, the “+”
- The “-” (coming ...) is due to syntactic restrictions on Datalog⁺ (ontological) programs for computational tractability of QA
- A Datalog⁺ program is combined with an extensional database (EDB)
- In contrast to classical Datalog, **when used as an ontology, a program plus an EDB is not subject to CWA: it may be incomplete**
- The rules allow to add data (while satisfying the constraints)
- **EDB is extended through the Datalog⁺ program**
Generating new tuples for EDB predicates, and full extensions for intensional predicates

- Most commonly, ontologies work under the “open world assumption” (OWA)
- Depending on the kind of rules, possibly several extensions
- Extensions are DBs (say Herbrand structures) that extend the EDB and are models, i.e. satisfy the rules as classical logical formulas
- Whatever is *true in all possible extensions* is considered to be *certain*
- We may want to materialize the extension(s) or keep them virtual
- Not a good idea ...
- Is there anything like a single model that “represents” all the others for QA?



And query them ...

- The **chase** (of the rules on the EDB) generates an instance that extends the EDB and “represents” the whole class of extensions

Propagate data via the TGDs, inserting **labelled nulls** for existentials

It turns out that **what is certain is what is true in the chase** (i.e. in the extension it produces)

- Example: (cont.) Initial DB (very simple):

Employee	name
	joe
	john

Manager	name
	joe
	sue

Supervises	name1	name2
	joe	john
	john	pete

Due to the IDs, we can propagate data (not always needed)

Employee	name
	joe
	john
	sue

Manager	name
	joe
	sue

Supervises	name1	name2
	joe	john
	john	pete
	sue	<i>null₁</i>

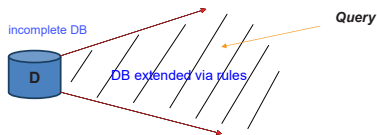
Query: $\exists x \text{Supervises}(sue, x)?$



Yes!

- With open queries (i.e. with variables), the certain answers obtained from the chase are those that do not exhibit labelled nulls
- In the example: $Q(x): \exists y \text{Supervises}(y, x)?$ $\{\text{john, pete}\}$
- When we query an incomplete DB which is extended with an ontology, we are querying the extension

Explicitly or implicitly



- No need to explicitly extend initial DB to answer the query
- Ideally, we want to avoid “completing” the DB first and then querying

- Example: Incomplete EDB $D = \{Person(john)\}$
- TGDs applied forward (as usual in Datalog), with value invention for existentials
- This is the main part of the “chase procedure”
- Set Σ of Datalog⁺ rules: $\exists x \text{ Father}(x, y) \leftarrow \text{Person}(y)$
 $\text{Person}(x) \leftarrow \text{Father}(x, y)$

The chase is a procedure that applies the TGDs in a forward manner, generating new tuples

$$\text{chase}(D, \Sigma) = \{ \text{Father}(z_1, \text{john}), \text{Person}(z_1), \\ \text{Father}(z_2, z_1), \text{Person}(z_2), \\ \text{Father}(z_3, z_2), \text{Person}(z_3), \dots \}$$

(each z_i is a labeled null value)

- Chase may create non-terminating loops
 So, the chase may not terminate

- The extensions of EDB may all be infinite, including the chase
- For arbitrary Datalog⁺ programs (no syntactic restrictions), QA is undecidable
- Related to (but not implied by) the fact that the chase procedure may be infinite
Finite or infinite, we can still query it, and is not always hopeless
- There are different families of “good” programs for which QA is computable, and even tractable
- QA complexity and techniques depend on syntactic restrictions on the ontology (the – in Datalog[±])
We concentrate on conjunctive queries
- So, Datalog[±] is a family of syntactic classes of programs that enjoy different good properties
- Different QA techniques for different good classes of programs

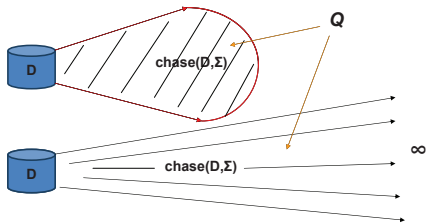
Syntactic Restrictions on Datalog⁺ Programs

- Restrictions are imposed on Datalog⁺ programs to ensure that queries are decidable (computable), or even tractable (in data)
- Datalog[±] stands for those “good” subclasses of Datalog⁺
Each of them is as a syntactic fragment of Datalog⁺

- Two cases for the chase
- In first case, QA is obviously decidable

If the chase can be built in PTIME (in data), QA is tractable

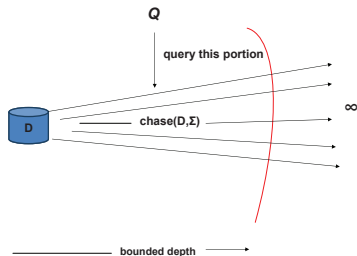
- In **second case**, QA may be (and sometimes is) undecidable
But also possibly decidable depending on the syntactic structure of the program



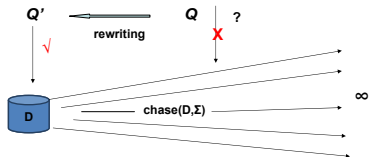
- Good cases when the chase is infinite?
Decidability of QA guaranteed by different syntactic conditions on the set of rules

(A) QA can be correctly done by **querying only a bounded, initial portion of the chase**

Hopefully a “short portion”



(B) FO query rewriting



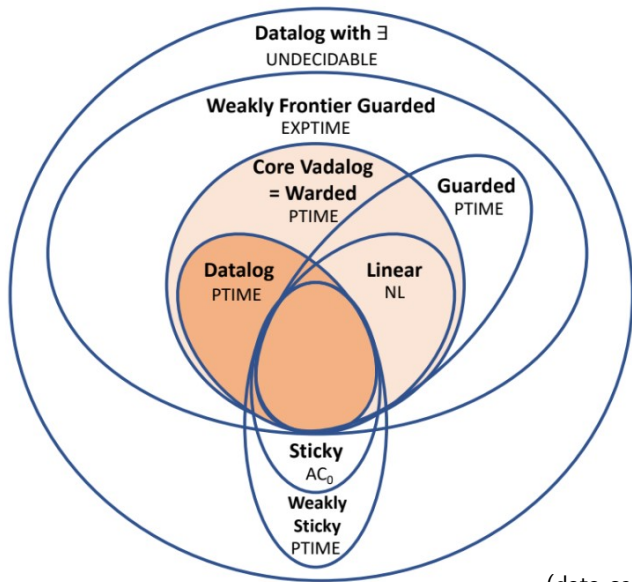
Instead of posing Q to the (infinite) chase, rewrite Q into new FO query Q' (independently from D)

This is done using the rules

Query D with Q' as usual

Definitely in PTIME in data

- Different syntactic restrictions on Datalog[±] programs (leading to different classes of Datalog[±] programs) ensure (A) or (B)
- Several classes of Datalog[±] programs ...



(data complexity)

- For the gist, what we mean by “syntactic restrictions”?
- The class *linear Datalog*⁺ contains programs whose rules are all of the form

$$\exists v S(x, v) \leftarrow P(x, y, z), \text{ or } R(x, z) \leftarrow S(x, y, z)$$



A single predicate in the body

- The class *guarded Datalog*⁺ contains programs whose rules are all of the form

$$\exists v S(x, v) \leftarrow \underbrace{P(x, y, z)}_{\text{guard}}, G(y, z), R(x), \text{ or}$$

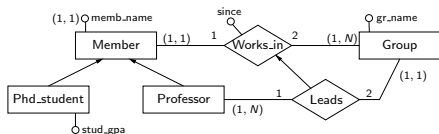


$$R(x, z) \leftarrow \underbrace{P(x, y, z)}_{\text{guard}}, U(y), S(x, z)$$

All the variables in a body appear also in a predicate in common (in the same body), the *guard*

- Etc.

- Example: ER model transformed (in part) into Datalog[±]



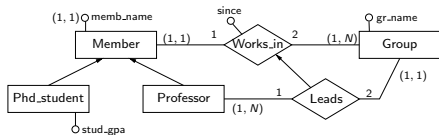
- Rules and Constraints: (mostly 1-ary and 2-ary predicates)

$leads(X, Y) \rightarrow works_in(X, Y)$	$professor(X) \rightarrow member(X)$
$professor(X) \rightarrow \exists Y leads(X, Y)$	$key(works_in) = \{1\}$
$member(X) \rightarrow \exists Y works_in(X, Y)$	$key(leads) = \{2\}$

- “Under” this ontology we could have a database containing professors, students, departments, etc.
- New wrt. Datalog: **existential quantifiers** in the consequents, and use of **constraints** (the last two above)
- Hence, the “+”

The “-” is due to syntactic restrictions on the ontologies to make QA computationally manageable

- A query:



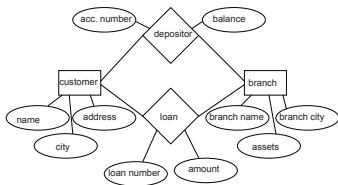
$leads(X, Y) \rightarrow works_in(X, Y)$	$professor(X) \rightarrow member(X)$
$professor(X) \rightarrow \exists Y leads(X, Y)$	$key(works_in) = \{1\}$
$member(X) \rightarrow \exists Y works_in(X, Y)$	$key(leads) = \{2\}$



$q(B) \leftarrow phd_student(A), memb_name(A, B), works_in(A, C),$
 $since(A, C, 2006), memb_name(C, db)$

- Asking about “Names of PhD students who work since 2006 in the DB group”

Exercise: Consider the ER model in the figure



From the ER model, obtain an ontology In Datalog[±], including some constraints

From Ontologies to the Semantic Web

- Ontologies gave an impulse to the [Semantic Web](#) (SW)
- The idea is to wrap web resources with a semantic layer that conveys the contents and its meaning



- It is a [WWW with meaning](#) ...
- In particular, to enable interoperability and data integration
- Today mostly to build DBs of “[linked data](#)” with limited reasoning capabilities

For computability and QA purposes

- Linking data has old tradition in AI: [conceptual graphs](#), [semantic networks](#), [frames](#), etc.

They are all behind ontologies and modern SW languages and representations

Even in high-school ...

taloma Bertossi '70'7

Octobre 10, 2007



- Much knowledge represented in a **semantic net** of **frames**

DOG
IS-A: ANIMAL
LEGS: NUMBER 0 TO 5 ASSUME 4
OWNER: HUMAN
COLOUR: BROWN OR RED OR WHITE OR BLACK

- ← frame name
- ← link to another frame
- other slots
- ← link to frame HUMAN

Description of **concept**: Dog

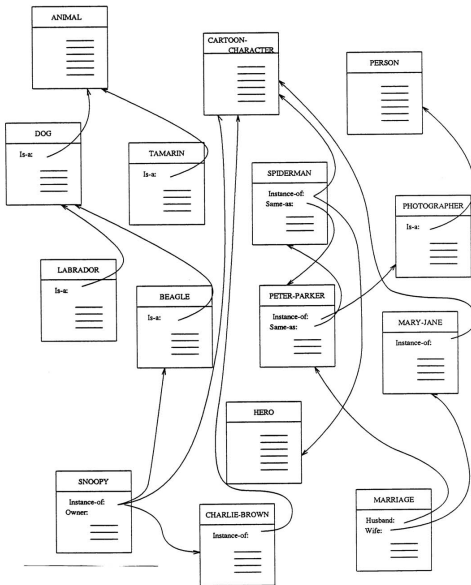
- Frames have slots

BEAGLE
IS-A: DOG
COLOUR: ASSUME BROWN AND WHITE

SNOOPY
INSTANCE-OF: BEAGLE
COLOUR: WHITE AND BLACK
OWNER: CHARLIE-BROWN

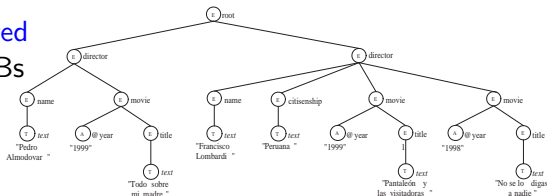
Descriptions of **subconcept**, and **instance** of a concept


- **Inheritance of properties**



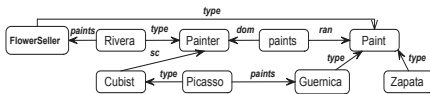
- Back to SW languages:
- Data on the Web, Ontologies, ...
- Many applications in Biology, Business, etc.
- Useful to represent **unstructured data**

As opposed to **structured data** as in relational DBs



- Common languages:
 - **XML**: Extensible Markup Language
 - **JSON**: JavaScript Object Notation 
 - **RDF, RDF-S**: Resource Description Framework (w/ Schema)
 - **OWL**: Ontology Web Language (different versions)
- Most have syntax resembling HTML

- Example: An RDF-S DB (its graphical representation)



- There are data and also conceptual, higher-level knowledge
A light-weight ontology And a graph-DB
- RDF DBs contain *triples*: $\langle \textit{Subject}, \textit{property}, \textit{Object} \rangle$

E.g. $\langle \textit{Picasso}, \textit{paints}, \textit{Guernica} \rangle$



- RDF is extended by RDF-S, for “schemas”

It has *properties* with fixed, built-in semantics (meaning):

type (for class membership), **sc** (for subclass, like an IS-A link), **dom** (for domain), **ran** (for range)

Properties of a class inherited by instances in subclass

- Notice the representation of data as a graph!

- In a computer an RDF-S DB could be a file containing triples in a markup language (think of HTML) (see below)
“Native” RDF ...
- Triples can also be stored/processed in a relational DB
Taking advantage of the two worlds ...
- Whole area of [Graph DBs](#) has emerged
- There are large RDF-S DBs for different applications
- RDF-S is also representation language of DBPEDIA (SW version of Wikipedia)

- We may consider OWL as the “official” language of the SW
- It extends RDF-S with more expressivity and built-in *properties*
- It uses basically the same computer representation language as RDF-S
- OWL inspired by (based on) the ontological (logical) languages of [Description Logic](#) (DL)
- There are three kinds of OWL based on different DLs
- So as there are different Datalog[±] languages, there are different DLs

For the same reason: tradeoff expressivity/computability

- OWLs go from light-weight to heavy-weight
With increasing level of expressivity
And of computational complexity ...