Systems and Solving Techniques for Knowledge Representation

– Datalog –

Marco Maratea
University of Genoa, Italy

066 011 Double degree programme Computational Logic (Erasmus-Mundus)
066 931 Computational Intelligence
066 937 Software Engineering & Internet Computing
Institute of Information Systems

What is Datalog?

Datalog:

- **A logic language for querying databases**
- Overcomes some limits of Relational Algebra and SQL

  $\rightarrow$Recursive definitions

Why Datalog?

- The basic fragment of ASP

What is Datalog?

Datalog:

- **A logic language for querying databases**
- Overcomes some limits of Relational Algebra and SQL

  $\rightarrow$Recursive definitions

Why Datalog?

- The basic fragment of ASP

What is Datalog?

Datalog:

- **A logic language for querying databases**
- Overcomes some limits of Relational Algebra and SQL

  $\rightarrow$Recursive definitions

Why Datalog?

- The basic fragment of ASP

Datalog:

- **A logic language for querying databases**
- Overcomes some limits of Relational Algebra and SQL

  $\rightarrow$Recursive definitions

Why Datalog?

- The basic fragment of ASP

What is Datalog?

Datalog:

- **A logic language for querying databases**
- Overcomes some limits of Relational Algebra and SQL

  $\rightarrow$Recursive definitions

Why Datalog?

- The basic fragment of ASP

What is Datalog?

Datalog:

- **A logic language for querying databases**
- Overcomes some limits of Relational Algebra and SQL

  $\rightarrow$Recursive definitions

Why Datalog?

- The basic fragment of ASP

Rule:

$head(\overline{H}) \;\text{:-}\; body_1(\overline{X_1}), \ldots, body_n(\overline{X_n}).$

Intuitively:

infer $head(\overline{h})$ if $body_1(\overline{x_1}), \ldots, body_n(\overline{x_n})$ is true.

Fact:

A rule with empty body (:- symbol is omitted)

$\rightarrow$ *Facts are true and model the input database* $\leftarrow$

Variables:

are allowed in atom's arguments, Prolog-like syntax

Safety:

all variables must occur in the body

Datalog Syntax

Example

Program and query:

$$father(X) \text{ :- } parent(X, Y), male(X).$$

Database:

$male(rob).$
$parent(rob, ann).$
$parent(mary, ann).$

Query Result:

$father(rob).$

Practice

Download a (Datalog) implementation (clasp)

`http://potassco.sourceforge.net/`

We need also a grounder (gringo)

`http://potassco.sourceforge.net/`

Practice

Download a (Datalog) implementation (clasp)

`http://potassco.sourceforge.net/`

We need also a grounder (gringo)

`http://potassco.sourceforge.net/`

Example (Reachable airports)

Input: A set of direct connections between some cities represented by *connected*(_, _). [or,*connected*/2.]

Query: Retrieve all the cities reachable by flight from Vienna airport, through a direct or undirect connection.

...can you write an SQL query?

Recursive Example Datalog

Example (Reachable airports)

Input: A set of direct connections between some cities represented by *connected*(_, _). [or,*connected*/2.]

Query: Retrieve all the cities reachable by flight from Vienna airport, through a direct or undirect connection.

Datalog:

$reaches(vienna, B) :– connected(vienna, B).$

$reaches(vienna, C) :– reaches(vienna, B), connected(B, C).$

Datalog Programs (1)

Datalog Program:

- A set of rules
- **EDB:** predicates appearing only in bodies or in facts
- **IDB :** predicates defined (also) by rules

Datalog Programs (1)

Datalog Program:

- A set of rules
- **EDB:** predicates appearing only in bodies or in facts
- **IDB :** predicates defined (also) by rules

Example (Reachability)

Input: a graph encoded by relation *edge*(_, _).
Problem: Find all pairs of reachable nodes.

% if there is an edge from X to Y
% then X is reachable from Y
$reachable(X, Y) \text{ :- } edge(X, Y).$

% Reachability is transitive
$reachable(X, Y) \text{ :- } reachable(X, Z), edge(Z, Y).$

Datalog Programs (1)

Datalog Program:

- A set of rules
- **EDB:** predicates appearing only in bodies or in facts
- **IDB :** predicates defined (also) by rules

Example (Reachability)

Input: a graph encoded by relation *edge*(_, _).
Problem: Find all pairs of reachable nodes.

% if there is an edge from X to Y
% then X is reachable from Y
$reachable(X, Y) \mathrel{:-} edge(X, Y).$ $\leftarrow$ EDB

% Reachability is transitive
$reachable(X, Y) \mathrel{:-} reachable(X, Z), edge(Z, Y).$

Datalog Program:

- A set of rules
- **EDB:** predicates appearing only in bodies or in facts
- **IDB :** predicates defined (also) by rules

Example (Reachability)

Input: a graph encoded by relation *edge*(_, _).
Problem: Find all pairs of reachable nodes.

% if there is an edge from X to Y
% then X is reachable from Y
$reachable(X, Y)$:- $edge(X, Y)$. $\leftarrow$ IDB

% Reachability is transitive
$reachable(X, Y)$:- $reachable(X, Z), edge(Z, Y)$.

Example (Reachability)

Input: a graph encoded by relation *edge*(_, _).
Problem: Find all pairs of reachable nodes.

```
% if there is an edge from X to Y
% then X is reachable from Y
reachable(X, Y) :- edge(X, Y).

% Reachability is transitive
reachable(X, Y) :- reachable(X, Z), edge(Z, Y).
```

Intuitive reasoning: *(bottom-up evaluation)*

"Start with the facts in the EDB and iteratively derive facts for IDBs until no new fact is derived."

Fully Declarative Language

Example (Ancestor)

Input: parent relation modeled by $parent(\_, \_)$.
Problem: Define the relation of arbitrary ancestors.

Solution 1:

$ancestor(A, B) :- parent(A, B).$
$ancestor(A, C) :- ancestor(A, B), ancestor(B, C).$

Fully Declarative Language

Example (Ancestor)

Input: parent relation modeled by $parent(\_, \_)$.
Problem: Define the relation of arbitrary ancestors.

Solution 1:

$ancestor(A, B) :- parent(A, B).$
$ancestor(A, C) :- ancestor(A, B), ancestor(B, C).$

Solution 2:

$ancestor(A, B) :- parent(A, B).$
$ancestor(A, C) :- ancestor(A, B), parent(B, C).$

Fully Declarative Language

Example (Ancestor)

Input: parent relation modeled by $parent(\_, \_)$.
Problem: Define the relation of arbitrary ancestors.

Solution 1:

$$ancestor(A, B) :\!- \; parent(A, B).$$
$$ancestor(A, C) :\!- \; ancestor(A, B), ancestor(B, C).$$

Solution 3: Declarative: Atoms' and Rules' order is immaterial!

$$ancestor(A, C) :\!- \; ancestor(A, B), parent(B, C).$$
$$ancestor(A, B) :\!- \; parent(A, B).$$

Arithmetic and comparison operators

- $+, -, *, /$
- $<, >, <=, >=, =$

Example (Fibonacci numbers)

$fib(1, 0).$
$fib(2, 1).$

$fib(N + 2, Y1 + Y2) \text{ :- } fib(N, Y1), fib(N + 1, Y2).$

For recursive definitions an upper bound for integers has to be specified, either as a system setting, or as a domain definition.

Arithmetic Expressions and Builtins

Arithmetic and comparison operators

- $+, -, *, /$
- $<, >, <=, >=, =$

Example (Fibonacci numbers)

$fib(1, 0).$
$fib(2, 1).$

$fib(N + 2, Y1 + Y2) \text{ :- } fib(N, Y1), fib(N + 1, Y2).$

For recursive definitions an upper bound for integers has to be specified, either as a system setting, or as a domain definition.

Example pure Datalog limits

Example (No Peroni here!)

Input: Information about bars and drinks represented by facts of the form
$type(drink, name)$. $sells(bar, drink)$

Query: Retrieve all bars that do not sell Peroni

...can you write an Datalog query?

Example pure Datalog limits

Example (No Peroni here!)

Input: Information about bars and drinks represented by facts of the form
$type(drink, name)$. $sells(bar, drink)$

Query: Retrieve all bars that do not sell Peroni

Datalog:

$$noPeroni(Bar) \text{ :- } sells(Bar, Drink),$$
$$\text{not } sellsPeroni(Bar).$$
$$sellsPeroni(Bar) \text{ :- } sells(Bar, Drink), type(Drink, peroni).$$

Datalog with Negation

Rule:

$$head(\overline{H}) \; \text{:-} \; body_1(\overline{X_1}), \ldots, body_n(\overline{X_n}),$$
$$\text{not } body_{n+1}(\overline{X_{n+1}}), \ldots, \text{not } body_m(\overline{X_m}).$$

Positive and Negative Body:

$body_1(\overline{x_1}), \ldots, body_n(\overline{x_n}) \leftarrow$ positive body
$body_{n+1}(\overline{x_{n+1}}), \ldots, body_m(\overline{x_m}). \leftarrow$ negative body

Intuitively:

infer $head(\overline{h})$ if all atoms in the positive body are true and all atoms in the negative body are false

Safety:

all variables must occur in a positive body literal

Stratification (intuitive):

negation must not be involved in recursive definitions!

Stratification (i.e., no recursion trough negation)

Example (Stratified Program)

$$p(X) \mathrel{:\!-} p(X), \mathrm{not}\ q(X).$$
$$q(X) \mathrel{:\!-} l(X), \mathrm{not}\ m(b).$$

Example (Unstratified Program)

$$p(X) \mathrel{:\!-} l(X), \mathrm{not}\ q(X).$$
$$q(X) \mathrel{:\!-} l(X), \mathrm{not}\ p(X)$$

Needed Restrictions for Safety ...

Safety:

$$s(X) \text{ :- } not\ r(X).$$

$$s(X, Y) \text{ :- } r(Y).$$

$$s(X, Y) \text{ :- } r(X), Y = Y.$$

Intuitively:

In each of these cases the result is infinite!?!

More on this later...

Needed Restrictions for Safety ...

Safety:

$$s(X) :- \textit{not } r(X).$$

$$s(X, Y) :- r(Y).$$

$$s(X, Y) :- r(X), Y = Y.$$

Intuitively:

In each of these cases the result is infinite!?!

More on this later...