Processing of Declarative Knowledge
–Datalog–

Francesco Ricca

Computational Intelligence Curriculum
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

  $\rightarrow$ Deductive database applications, query answering

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

  $\rightarrow$ Deductive database applications, query answering

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

  $\rightarrow$ Deductive database applications, query answering

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

  $\rightarrow$ Deductive database applications, query answering

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

  $\rightarrow$ Deductive database applications, query answering

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

  $\rightarrow$ Deductive database applications, query answering

Datalog Syntax

Rule:

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

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:

$father(X) :\!- parent(X, Y), male(X).$

Database:

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

Query Result:

$father(rob).$

Recursive Example Datalog

Example (Reachable airports)

Input: A set of direct connections between some cities represented by *connected*(_, _).

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*(_, _).

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
$reachable(X, Y) :\!- edge(X, Y).$

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

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
$reachable(X, Y) :\!- edge(X, Y).$ ← EDB

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

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
$reachable(X, Y) :\!- edge(X, Y).$ ← IDB

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

Datalog Programs

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 meaning: *(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:

$$ancestor(A, B) \text{ :- } parent(A, B).$$
$$ancestor(A, C) \text{ :- } ancestor(A, B), ancestor(B, C).$$

Fully Declarative Language

Example (Ancestor)

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

Solution 1:

$$ancestor(A, B) \text{ :- } parent(A, B).$$
$$ancestor(A, C) \text{ :- } ancestor(A, B), ancestor(B, C).$$

Solution 2:

$$ancestor(A, B) \text{ :- } parent(A, B).$$
$$ancestor(A, C) \text{ :- } ancestor(A, B), parent(B, C).$$

Fully Declarative Language

Example (Ancestor)

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

Solution 1:

$$ancestor(A, B) \text{ :- } parent(A, B).$$
$$ancestor(A, C) \text{ :- } ancestor(A, B), ancestor(B, C).$$

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

$$ancestor(A, C) \text{ :- } ancestor(A, B), parent(B, C). \leftarrow \text{No LOOP!}$$
$$ancestor(A, B) \text{ :- } parent(A, B).$$

Arithmetic Expressions and Builtins

Arithmetic and comparison operators

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

Example (Fibonacci numbers)

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

$fib(N + 2, Y1 + Y2) :- fib(N, Y1), fib(N + 1, Y2).$

For recursive definitions an upper bound for integers (system setting) or a domain has to be specified.

Example pure Datalog limits

Example (No Peroni here!)

Input: Information about bars and beers represented by facts of the form
beers(*name*, *manufacturer*). *sells*(*bar*, *beer*)

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 beers represented by facts of the form
beers(*name*, *manufacturer*). *sells*(*bar*, *beer*)

Query: Retrieve all bars that do not sell Peroni

Datalog:

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

$$sellsPeroni(Bar) \text{ :- } sells(Bar, Beer), beer(Beer, peroni).$$

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 (Unstratified Program)

$$p(X) \text{ :- } l(X), \text{not } q(X).$$
$$q(X) \text{ :- } l(X), \text{not } p(X)$$

Example (Stratified Program)

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

Needed Restrictions

Safety:

$s(X) :\!- r(Y).$

$s(X) :\!- notr(X).$

$s(X) :\!- r(Y), X < Y.$

Intuitively:

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

Stratification:

Negation wrapped inside recursion is not that obvious

$a :\!-$ not $b.$ $\quad$ $b :\!-$ not $a.$

More on this later...

Needed Restrictions

Safety:

$s(X) :\!- r(Y).$

$s(X) :\!- notr(X).$

$s(X) :\!- r(Y), X < Y.$

Intuitively:

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

Stratification:

Negation wrapped inside recursion is not that obvious

$a :\!-$ not $b.$ $\qquad$ $b :\!-$ not $a.$

More on this later...

Practice

Practice

Download a Datalog implementation

`http://www.dlvsystem.com`

Download a GUI

`http://www.mat.unical.it/ricca/aspide`

Syntax & Notation

Terms: Constants and Variables

Atoms: of the form $predicate(t_1, \ldots, t_n)$

Literals: atoms a (pos.) and negated atoms $\text{not } a$ (neg.)

Rules: $h \text{ :- } p_1, \ldots, p_n, \text{not } n_1, \ldots \text{not } n_n.$

Head: $H(r) = h$

Body: $B(r) = \{p_1, \ldots, p_n, \text{not } n_1, \ldots \text{not } n_n.\}$

Positive Body: $B^+(r) = \{p_1, \ldots, p_n\}$

Negative Body: $B^-(r) = \{\text{not } n_1, \ldots \text{not } n_n\}$

Program: A set of rules

Safety: All variables occur in some positive body atom

Ground: no variable occurs in it

Positive Program: all rules are such that $B^-(r) = \emptyset$

Semantics Positive Programs

Interpretation: a set I of ground atoms

- atom a is true w.r.t. I if $a \in I$, it is false otherwise, and
- negative literal $\text{not } a$ is true w.r.t. I if $a \notin I$, it is false otherwise.

Satisfaction: a rule r is satisfied w.r.t. I if $H(r) \in I$ whenever all literals $\ell \in B(r)$ are true w.r.t. I

Model: an interpretation I is a model for program P if all rules in P are satisfied by I

Least Model: an interpretation I is the least or minimal model for program P if every $I' \subset I$ is not a model for P

Example Models

Given:

$a :- b, c.$

$c :- d.$

$d.$

Interpretations and Models:

$I_1 = \{b, c, d\}$, $I_2 = \{a, b, c, d\}$ $I_3 = \{c, d\}$

$\rightarrow$ only I_2 and I_3 are models!

Example Models

Given:

$a :- b, c.$

$c :- d.$

$d.$

Interpretations and Models:

$I_1 = \{b, c, d\}$, $I_2 = \{a, b, c, d\}$ $I_3 = \{c, d\}$

$\rightarrow$ only I_2 and I_3 are models!
$\rightarrow$ I_3 is minimal!

Semantics Positive Programs

Rule Instantiation: $I(r)$ is the set ground rules that can be obtained by replacing every variable in r by a constant occurring in P

Instantiation: $G(P) = \cup_{r \in P} I(r)$

Model: an interpretation M is a model for program P if M is a model of $G(P)$

Least Model: an interpretation M is the least model of program P if M is the least model of $G(P)$

Operational Semantics Positive Programs (Ground case)

Immediate Consequence Operator: Given ground program P, and Interpretation I

$T_p(I) = \{a | \exists r \in P \text{ s.t. } H(r) = a \vee \forall l \in B(r) \text{ are true in } I\}$

Example: $a :\!\!- b.\ c :\!\!- d.\ e :\!\!- a.\ I = \{b\}\ Tp(I) = \{a\}.$

Fixpoint procedure:

- Start with $I = \emptyset$
- Repeatedly apply T_p until a fixpoint $T_p(I) = I$ is reached.

Least Model: The least fixpoint T_p.
Theorem: A positive Datalog program P has a unique least model, which is the minimal model corresponding to the intersection of all models of P.

Operational Semantics Positive Programs (Ground case)

Immediate Consequence Operator: Given ground program P, and Interpretation I

$T_p(I) = \{a | \exists r \in P \text{ s.t. } H(r) = a \vee \forall l \in B(r) \text{ are true in } I\}$

Example: $a :\!\!- b.\ c :\!\!- d.\ e :\!\!- a.\ I = \{b\}\ Tp(I) = \{a\}.$

Fixpoint procedure:

- Start with $I = \emptyset$
- Repeatedly apply T_p until a fixpoint $T_p(I) = I$ is reached.

Least Model: The least fixpoint T_p.
Theorem: A positive Datalog program P has a unique least model, which is the minimal model corresponding to the intersection of all models of P.

Operational Semantics Positive Programs (Ground case)

Immediate Consequence Operator: Given ground program P, and Interpretation I

$T_p(I) = \{a | \exists r \in P \text{ s.t. } H(r) = a \vee \forall l \in B(r) \text{ are true in } I\}$

Example: $a :\!\!- b.\ c :\!\!- d.\ e :\!\!- a.\ I = \{b\}\ Tp(I) = \{a\}.$

Fixpoint procedure:

- Start with $I = \emptyset$
- Repeatedly apply T_p until a fixpoint $T_p(I) = I$ is reached.

Least Model: The least fixpoint T_p.
Theorem: A positive Datalog program P has a unique least model, which is the minimal model corresponding to the intersection of all models of P.

Operational Semantics

Ground + Fixpoint:

Given P, build $G(P)$, apply operator to compute fixpoint $T_{G(P)}(M) = M$

Consider:

$a(X) :- b(X), c(X).$

$b(a).\ b(b).\ c(a).\ c(c).$

Instantiation:

$a(a) :- b(a), c(a).$

$a(b) :- b(b), c(b).$

$a(c) :- b(c), c(c).$

$\ldots$

Operational Semantics

Ground + Fixpoint:

Given P, build $G(P)$, apply operator to compute fixpoint $T_{G(P)}(M) = M$

Consider:

$a(X) :- b(X), c(X).$

$b(a).\ b(b).\ c(a).\ c(c).$

Instantiation:

$a(a) :- b(a), c(a).$

$a(b) :- b(b), c(b).$

$a(c) :- b(c), c(c).$

$\ldots$

Operational Semantics

Ground + Fixpoint:

Given P, build $G(P)$, apply operator to compute fixpoint $T_{G(P)}(M) = M$

Consider:

$a(X) :- b(X), c(X).$

$b(a).\ b(b).\ c(a).\ c(c).$

Instantiation:

$a(a) :- b(a), c(a).$

$a(b) :- b(b), c(b).$

$a(c) :- b(c), c(c).$

$\ldots$ **Do we need all ground rules?**

Operational Semantics

Ground + Fixpoint:

Given P, build $G(P)$, apply operator to compute fixpoint $T_{G(P)}(M) = M$

Consider:

$a(X) :- b(X), c(X).$

$b(a).\ b(b).\ c(a).\ c(c).$

Instantiation:

$a(a) :- b(a), c(a).$

$a(b) :- b(b), c(b).$

$a(c) :- b(c), c(c).$

$\ldots$ **Do they have any chance to be satisfied?**

Operational Semantics

Ground + Fixpoint:

Given P, build $G(P)$, apply operator to compute fixpoint $T_{G(P)}(M) = M$

Consider:

$a(X) :- b(X), c(X).$

$b(a).\ b(b).\ c(a).\ c(c).$

Instantiation:

$a(a) :- b(a), c(a).$

$a(b) :- b(b), c(b).$

$a(c) :- b(c), c(c).$

$\ldots$ Start from facts, match bodies, apply ... fixpoint!

Example Semantics

Consider:

$grandParent(X, Y) :- parent(X, Z), parent(Z, Y).$

$parent(a, b).\ parent(b, c).$

Evaluation:

1. $I = \{parent(a, b), parent(b, c)\}$
2. the body can be instantiated
   $(parent(a, b), parent(b, c))$
   $I := I \cup \{grandParent(a, c)\}$
3. no body can be matched with atoms in I ... STOP!

Results: $\{parent(a, b), parent(b, c), grandParent(a, c)\}$ is the least model

Example Semantics

Consider:

$grandParent(X, Y) :\!- parent(X, Z), parent(Z, Y).$

$parent(a, b).\ parent(b, c).$

Evaluation:

1. $I = \{parent(a, b), parent(b, c)\}$
2. the body can be instantiated
   $(parent(a, b), parent(b, c))$
   $I := I \cup \{grandParent(a, c)\}$
3. no body can be matched with atoms in I ... STOP!

Results: $\{parent(a, b), parent(b, c), grandParent(a, c)\}$ is the least model

Example Semantics

Consider:

$grandParent(X, Y) :- parent(X, Z), parent(Z, Y).$

$parent(a, b).\ parent(b, c).$

Evaluation:

1. $I = \{parent(a, b), parent(b, c)\}$
2. the body can be instantiated
   $(parent(a, b), parent(b, c))$
   $I := I \cup \{grandParent(a, c)\}$
3. no body can be matched with atoms in I ... STOP!

Results: $\{parent(a, b), parent(b, c), grandParent(a, c)\}$ is the least model

Semantics c.t.d.

Immediate Consequence Operator:
Given ground program P, and Interpretation I

$$T_p(I) = \{H(r_g) | \exists r_g \text{ instantiating } r \in P \text{ s.t. the body of } r_g \text{ is true w.r.t. } I\}$$

Operational Semantics:

Compute $M = T_p(M)$ by repeatedly applying T_p starting from EDB.

Stratified Programs

Dependency Graph: Given program P, graph $DG(P)$ is as follows:

- a node p in V for each predicate in p occurring in P
- positive edge $p \leftarrow q$ if there is rule r s.t. p occurs in $H(r)$ and q occurs in $B^+(r)$
- negative edge $p \leftarrow_n q$ if there is rule r s.t. p occurs in $H(r)$ and q occurs in $B^-(r)$

Recursive Program: P is recursive if $DG(P)$ is cyclic.

Stratified Program: P is stratified if no cycle in $DG(P)$ contains a negative edge.

Negation and Recursion

Consider:

$p(X) \text{ :- } q(X), \text{not } p(X).$

$q(1).\ q(2).$

Evaluation:

1. $q = \{(1), (2)\}, p = \{\}$
2. $q = \{(1), (2)\}, p = \{(1), (2)\}$
3. $q = \{(1), (2)\}, p = \{\}$
4. ...

Stratified Program

Consider:

$r_1 : reach(X) :- source(X).$

$r_2 : reach(X) :- reach(Y), arc(Y, X).$

$r_3 : noReach(X) :- target(X), notreach(X).$

Dependency Graph:

- V = {reach,source,target,noReach,arc}
- E = {(reach,source), (reach,reach), (reach,arc), (noReach,target), (noReach,reach)$_n$}
- cyclic, but stratified!

Stratified Program

Consider:

$r_1 : reach(X) :- source(X).$

$r_2 : reach(X) :- reach(Y), arc(Y, X).$

$r_3 : noReach(X) :- target(X), notreach(X).$

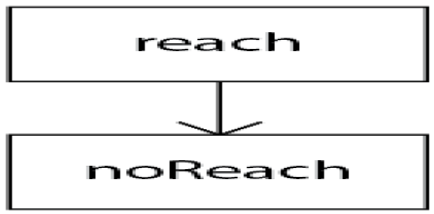
Dependency Graph:

- V = {reach,source,target,noReach,arc}
- E = {(reach,source), (reach,reach), (reach,arc), (noReach,target), $(noReach,reach)_n$}
- cyclic, but stratified!

Stratified Program - components and modules

Components and Subprograms:

- Let Comp(DG) be the set of the strongly connected components of DG
- Given $C \in Comp(DG)$ the subprogram associated to C is $Sub(P, C) = \{r \in P \text{ s.t. } H(r) \in C\}$
- Given C' depends on C'' if there is some (negative) arc in DG from a node in C'' to a node in C'



Example ctd:

- $Comp(DG) = \{\{reach\}, \{noReach\}\}$
- $Sub(P, \{reach\}) = \{r_1, r_2\}$
- $Sub(P, \{noReach\}) = \{r_3\}$

Stratified Program - Evaluation

Evaluation:

- Start from the components that do not depend on other components
- Evaluate subprograms associated to components as for positive programs
- Remove evaluated components
- repeat until all components are evaluated

Example ctd:

1. Evaluate $\{\{reach\}\}$
2. Evaluate $\{\{noReach\}\}$

Example Stratified Program

Consider:

$r_1 : reach(X) : -source(X).$

$r_2 : reach(X) : -reach(Y), arc(Y, X).$

$r_3 : noReach(X) : -target(X), notreach(X).$

EDB: $node(1).node(2).node(3).node(4).arc(1, 2).$
$arc(3, 4).arc(4, 3).source(1), target(2).target(3).$

Evaluate $Sub(P, \{reach\}) = \{r_1, r_2\}$**:**

1. $I = \{source(1), target(2), target(3), ...\}$
2. $I := I \cup \{reach(1)\}$
3. $I := I \cup \{reach(2)\}$...STOP!

Evaluate $Sub(P, \{noReach\}) = \{r_3\}$**:**

1. $I := I \cup \{noReach(3)\}$...STOP!

Example Stratified Program

Consider:

$r_1 : reach(X) : -source(X).$

$r_2 : reach(X) : -reach(Y), arc(Y, X).$

$r_3 : noReach(X) : -target(X), notreach(X).$

EDB: $node(1).node(2).node(3).node(4).arc(1, 2).$
$arc(3, 4).arc(4, 3).source(1), target(2).target(3).$

Evaluate $Sub(P, \{reach\}) = \{r_1, r_2\}$**:**

1. $I = \{source(1), target(2), target(3), ...\}$
2. $I := I \cup \{reach(1)\}$
3. $I := I \cup \{reach(2)\}$...STOP!

Evaluate $Sub(P, \{noReach\}) = \{r_3\}$**:**

1. $I := I \cup \{noReach(3)\}$...STOP!

Example Stratified Program

Consider:

$r_1 : reach(X) : -source(X).$

$r_2 : reach(X) : -reach(Y), arc(Y, X).$

$r_3 : noReach(X) : -target(X), notreach(X).$

EDB: $node(1).node(2).node(3).node(4).arc(1, 2).$
$arc(3, 4).arc(4, 3).source(1), target(2).target(3).$

Evaluate $Sub(P, \{reach\}) = \{r_1, r_2\}$**:**

1. $I = \{source(1), target(2), target(3), ...\}$
2. $I := I \cup \{reach(1)\}$
3. $I := I \cup \{reach(2)\}$...STOP!

Evaluate $Sub(P, \{noReach\}) = \{r_3\}$**:**

1. $I := I \cup \{noReach(3)\}$...STOP!

Exercise... limits

Given the following relational database schema (* indicates primary keys):

- $beers(name*, manufacturer)$
- $sells(bar*, beer*, price)$
- $associate(bar, bar)$

Write the following (if possible) in SQL, and Datalog

1. find the manufacturers of the beers "John's bar" sells
2. find the number of beers that "John's bar" sells at a price higher than "Anns's bar"
3. find the bars that sell exactly two beers
4. find the bars that sell more than three beers
5. find the bars that are associated, directly or indirectly, trough a chain of bar associations to "John's bar"
6. find the most expensive beer
7. find the bars that sell more beers