IA008: Computational Logic
3. Prolog

Achim Blumensath   blumens@fi.muni.cz

Faculty of Informatics, Masaryk University, Brno

Prolog

Prolog

Syntax

A **Prolog** program consists of a sequence of statements of the form

$$p(\bar{s}). \quad \text{or} \quad p(\bar{s}) :- q_0(\bar{t}_0), \ldots, q_{n-1}(\bar{t}_{n-1}).$$

p, q_i relation symbols, $\bar{s}, \bar{t}_i$ tuples of terms.

Semantics

$$p(\bar{s}) :- q_0(\bar{t}_0), \ldots, q_{n-1}(\bar{t}_{n-1}).$$

corresponds to the implication

$$p(\bar{s}) \leftarrow \exists \bar{y}[q_0(\bar{t}_0) \wedge \cdots \wedge q_{n-1}(\bar{t}_{n-1})]$$

where $\bar{y}$ are all the variables that do not appear in $\bar{s}$.

Example

father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).

parent_of(X, Y) :− father_of(X, Y).
parent_of(X, Y) :− mother_of(X, Y).

sibling_of(X, Y) :− parent_of(Z, X), parent_of(Z, Y).

ancestor_of(X, Y) :− father_of(X, Z), ancestor_of(Z, Y).

Interpreter

On input

$$p_0(\bar{s}_0), \ldots, p_{n-1}(\bar{s}_{n-1}).$$

the program finds all values for the variables satisfying the conjunction

$$p_0(\bar{s}_0) \wedge \cdots \wedge p_{n-1}(\bar{s}_{n-1}).$$

Example

```
?- sibling_of(sam, tina).
Yes

?- sibling_of(X, Y).
X = sam, Y = tina
```

Execution

Input

- program Π (set of Horn clauses)
- goal $\varphi(\bar{X}) = \psi_0(\bar{X}) \wedge \cdots \wedge \psi_{n-1}(\bar{X})$

Evaluation strategy

- transform $\neg\varphi = \neg\psi_0 \vee \cdots \vee \neg\psi_{n-1}$ into a clause;
- use resolution to check for which values of $\bar{X}$ the union $\Pi \cup \{\neg\varphi(\bar{X})\}$ is unsatisfiable.

Remark

As we are dealing with a set of Horn clauses, we can use **linear resolution.** The variant used by Prolog-interpreters is called **SLD-resolution.**

SLD-resolution

- ▶ Current goal: $\neg\psi_0 \vee \cdots \vee \neg\psi_{n-1}$
- ▶ If $n = 0$, stop.
- ▶ Otherwise, find a clause $\psi' \leftarrow \vartheta_0 \wedge \cdots \wedge \vartheta_{m-1}$ from Π such that ψ_0 and ψ' can be unified.
- ▶ If no such clause exists, backtrack.
- ▶ Otherwise, resolve them to produce the new goal

$$\tau(\neg\vartheta_0) \vee \cdots \vee \tau(\neg\vartheta_{m-1}) \vee \sigma(\neg\psi_1) \vee \cdots \vee \sigma(\neg\psi_{n-1}).$$

(σ, τ is the most general unifier of ψ_0 and ψ'.)

Implementation

Use a stack machine that keeps the current goal on the stack.
(→ **Warren Abstract Machine**)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal   ¬sibling_of(tina, sam)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal $\neg$sibling_of(tina, sam)

unify with   sibling_of(X, Y) $\leftarrow$ parent_of(Z, X) $\wedge$ parent_of(Z, Y)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal $\neg$sibling_of(tina, sam)

unify with   sibling_of$(X, Y) \leftarrow$ parent_of$(Z, X) \wedge$ parent_of(Z, Y)

unifier X = tina, Y = sam

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal $\neg$sibling_of(tina, sam)

unify with   sibling_of(X, Y) $\leftarrow$ parent_of(Z, X) $\wedge$ parent_of(Z, Y)

unifier X = tina, Y = sam

new goal $\neg$parent_of(Z, tina), $\neg$parent_of(Z, sam)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal $\neg parent\_of(Z, tina),\ \neg parent\_of(Z, sam)$

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal $\neg$parent_of$(Z, \text{tina}), \neg$parent_of(Z, sam)

unify with   parent_of$(X, Y) \leftarrow$ mother_of(X, Y)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input sibling_of(tina, sam)

goal $\quad\quad$ ¬parent_of(Z, tina), ¬parent_of(Z, sam)

unify with $\quad$ parent_of$(X, Y) \leftarrow$ mother_of(X, Y)

unifier $\quad\quad X = Z, \ Y = \text{tina}$

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal $\neg$parent_of(Z, tina), $\neg$parent_of(Z, sam)

unify with   parent_of$(X, Y) \leftarrow$ mother_of(X, Y)

unifier $X = Z$, $Y = \text{tina}$

new goal $\neg$mother_of(Z, tina), $\neg$parent_of(Z, sam)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input sibling_of(tina, sam)

goal $\neg$mother_of(Z, tina), $\neg$parent_of(Z, sam)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal   ¬mother_of(Z, tina), ¬parent_of(Z, sam)

unify with   mother_of(sara, john)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal $\neg$mother_of(Z, tina), $\neg$parent_of(Z, sam)

unify with   mother_of(sara, john)

fails

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal	$\neg$mother_of(Z, tina), $\neg$parent_of(Z, sam)
unify with	mother_of(sara, john)
fails	
backtrack to	$\neg$parent_of(Z, tina), $\neg$parent_of(Z, sam)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input sibling_of(tina, sam)

goal $\neg$parent_of(Z, tina), $\neg$parent_of(Z, sam)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal $\neg$parent_of(Z, tina), $\neg$parent_of(Z, sam)

unify with   parent_of$(X, Y) \leftarrow$ father_of(X, Y)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal $\neg\text{parent\_of}(Z, \text{tina}), \neg\text{parent\_of}(Z, \text{sam})$

unify with $\text{parent\_of}(X, Y) \leftarrow \text{father\_of}(X, Y)$

unifier $X = Z, \; Y = \text{tina}$

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal	$\neg$parent_of$(Z, \text{tina}), \neg$parent_of(Z, sam)
unify with	parent_of$(X, Y) \leftarrow$ father_of(X, Y)
unifier	$X = Z, Y = \text{tina}$
new goal	$\neg$father_of$(Z, \text{tina}), \neg$parent_of(Z, sam)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input sibling_of(tina, sam)

goal $\neg father\_of(Z, tina), \neg parent\_of(Z, sam)$

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal $\neg$father_of$(Z, \text{tina}), \ \neg$parent_of(Z, sam)

unify with   father_of(peter, sam)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal $\neg$father_of(Z, tina), $\neg$parent_of(Z, sam)

unify with   father_of(peter, sam)

fails

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input sibling_of(tina, sam)

goal $\neg$father_of(Z, tina), $\neg$parent_of(Z, sam)

unify with father_of(peter, sam)

fails

unify with father_of(peter, tina)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input sibling_of(tina, sam)

goal $\neg$father_of(Z, tina), $\neg$parent_of(Z, sam)

unify with father_of(peter, sam)

fails

unify with father_of(peter, tina)

unifier Z = peter

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal $\neg$father_of(Z, tina), $\neg$parent_of(Z, sam)

unify with   father_of(peter, sam)

fails

unify with   father_of(peter, tina)

unifier Z = peter

new goal $\neg$parent_of(peter, sam)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

**Input**   sibling_of(tina, sam)

goal   ¬parent_of(peter, sam)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal   ¬parent_of(peter, sam)

. . .   . . .

goal   ¬father_of(peter, sam)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal   ¬parent_of(peter, sam)

. . .   . . .

goal   ¬father_of(peter, sam)

unify with   father_of(peter, sam)

Example

```
father_of(peter, sam).
father_of(peter, tina).
mother_of(sara, john).
parent_of(X, Y) :- mother_of(X, Y).
parent_of(X, Y) :- father_of(X, Y).
sibling_of(X, Y) :- parent_of(Z, X), parent_of(Z, Y).
```

Input   sibling_of(tina, sam)

goal   ¬parent_of(peter, sam)

...   ...

goal   ¬father_of(peter, sam)

unify with   father_of(peter, sam)

new goal   empty

Search tree
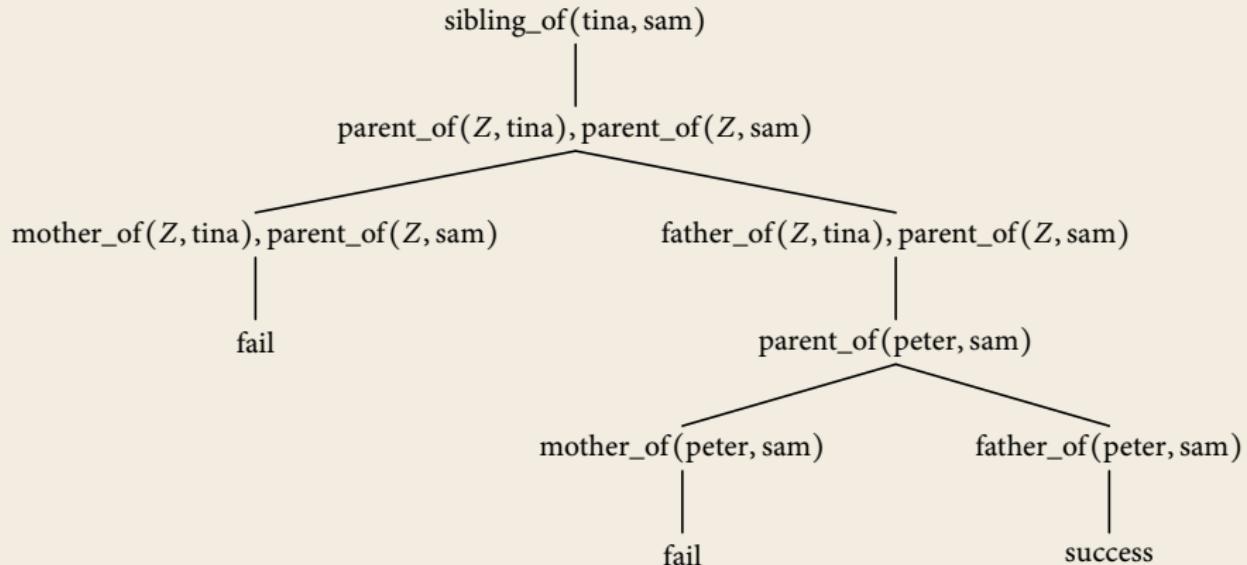
Caveats

Prolog-interpreters use a simpler (and **unsound**) form of unification that ignores multiple occurrences of variables. E.g. they happily unify $p(x, f(x))$ with $p(f(y), f(y))$ (equating $x = f(y)$ for the first x and $x = y$ for the second one).

Caveats

Prolog-interpreters use a simpler (and **unsound**) form of unification that ignores multiple occurrences of variables. E.g. they happily unify $p(x,f(x))$ with $p(f(y),f(y))$ (equating $x = f(y)$ for the first x and $x = y$ for the second one).

It is also easy to get infinite loops if you are not careful with the ordering of the clauses:

```
edge(c,d).
path(X,Y) :- path(X,Z),edge(Z,Y).
path(X,Y) :- edge(X,Y).
```

produces

```
?- path(X,Y).
path(X,Z), edge(Z,Y).
path(X,U), edge(U,Z), edge(Z,Y).
path(X,V), edge(V,U), edge(U,Z), edge(Z,Y).
...
```

Example: List processing

```
append([], L, L).
append([H|T], L, [H|R]) :- append(T, L, R).

?- append([a,b], [c,d], X).
X = [a,b,c,d]

?- append(X, Y, [a,b,c,d])
X = [], Y = [a,b,c,d]
X = [a], Y = [b,c,d]
X = [a,b], Y = [c,d]
X = [a,b,c], Y = [d]
X = [a,b,c,d], Y = []
```

Example: List processing

```
reverse(Xs, Ys) :- reverse_(Xs, [], Ys).

reverse_([], Ys, Ys).
reverse_([X|Xs], Rs, Ys) :- reverse_(Xs, [X|Rs], Ys).

reverse([a,b,c], X)
reverse_([a,b,c], [], X)
reverse_([b,c], [a], X)
reverse_([c], [b,a], X)
reverse_([], [c,b,a], X)
X = [c,b,a]
```

Example: Natural language recognition

```prolog
sentence(X,R) :- noun(X, Y), verb(Y, R).
sentence(X,R) :- noun(X, Y), verb(Y, Z), noun(Z, R).

noun_phrase(X,           R) :- noun(X, R).
noun_phrase(['a'   | X], R) :- noun(X, R).
noun_phrase(['the' | X], R) :- noun(X, R).

noun(['cat'   | R], R).
noun(['mouse' | R], R).
noun(['dog'   | R], R).

verb(['eats'  | R], R).
verb(['hunts' | R], R).
verb(['plays' | R], R).
```

Cuts

Control backtracking using **cuts:**

$$p :- q_0, q_1, !, q_2, q_3.$$

When backtracking across a cut !, directly jump to the head of the rule and assume it fails. Do not try other rules.

Example

$s \leftarrow p$

$s \leftarrow t$

$p \leftarrow q_1, q_2, !, q_3, q_4$

$p \leftarrow r$

r

q_1

q_2

q_3
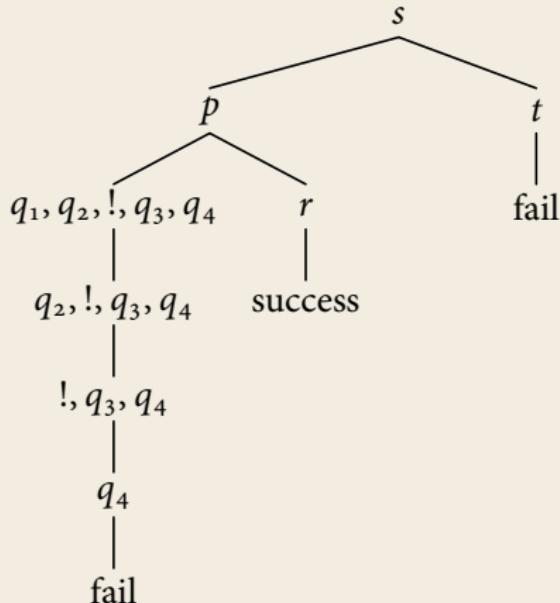
Negation

Problem

If we allow **negation,** the formulae are no longer **Horn** and SLD-resolution does no longer work.

Possible Solutions

- **Closed World Assumption**
  If we cannot derive p, it is false (**Negation as Failure**).

- **Completed Database**
  $p \leftarrow q_0, \ldots, p \leftarrow q_n$ is interpreted as the stronger statement $p \leftrightarrow q_0 \vee \cdots \vee q_n$.

Examples

Being connected by a path of non-edges:

```
q(X,X).
q(X,Y) :- q(X,Z), not(p(Z,Y)).
```

Implementing negation using cuts:

```
not(A) :- A, !, fail.
not(A).
```

Some cuts can be implemented using negation:

```
p :- a, !, b.          p :- a, b.
p :- c.                p :- not(a), c.
```

Databases

Databases

Definition

A **database** is a set of relations called **tables.**

Example

flight	from	to	price
LH8302	Prague	Frankfurt	240
OA1472	Vienna	Warsaw	300
UA0870	London	Washington	800
…			

Formal Definitions

We treat a database as a structure $\mathfrak{A} = \langle A, R_0, \ldots, R_n \rangle$ with

- universe A containing all entries and
- one relation $R_i \subseteq A \times \cdots \times A$ per table.

The **active domain** of a database is the set of elements appearing in some relation.

Example

In the previous table, the active domain contains:

LH8302, OA1472, UA0870, 240, 300, 800,
Prague, Frankfurt, Vienna, Warsaw, London, Washington

Queries

A **query** is a function mapping each database to a relation.

Example

Input: database of direct flights
Output: table of all flight connections possibly including stops

In Prolog: database **flight**, query **connection**.

```
flight('LH8302', 'Prague', 'Frankfurt',  240).
flight('OA1472', 'Vienna', 'Warsaw',     300).
flight('UA0870', 'London', 'Washington', 800).

connection(From, To) :- flight(X, From, To, Y).
connection(From, To) :-
  flight(X, From, T, Y), connection(T, To).
```

Relational Algebra

Syntax

- basic relations
- boolean operations ∩, ∪, ∖, All
- cartesian product ×
- selection σ_{ij}
- projection $\pi_{u_0 \ldots u_{n-1}}$

Examples

- $\pi_{1,0}(R) = \left\{ (b,a) \mid (a,b) \in R \right\}$
- $\pi_{0,3}(\sigma_{1,2}(E \times E)) = \left\{ (a,c) \mid (a,b),(b,c) \in E \right\}$

Relational Algebra

Syntax

- basic relations
- boolean operations $\cap$, $\cup$, $\smallsetminus$, All
- cartesian product $\times$
- selection σ_{ij}
- projection $\pi_{u_0 \ldots u_{n-1}}$

Examples

- $\pi_{1,0}(R) = \left\{ (b, a) \mid (a, b) \in R \right\}$
- $\pi_{0,3}(\sigma_{1,2}(E \times E)) = \left\{ (a, c) \mid (a, b), (b, c) \in E \right\}$

Join

$$R \bowtie_{ij} S := \sigma_{ij}(R \times S)$$

Expressive Power

Theorem
Relational Algebra = First-Order Logic

Expressive Power

Theorem
Relational Algebra = First-Order Logic

Proof
$(\leq)$

$$R^* := R(x_0, \ldots, x_{n-1})$$
$$(s \cap t)^* := s^* \wedge t^*$$
$$(s \cup t)^* := s^* \vee t^*$$
$$(s \smallsetminus t)^* := s^* \wedge \neg t^*$$
$$\text{All}^* := \text{true}$$
$$(s \times t)^* := s^*(x_0, \ldots, x_{m-1}) \wedge t^*(x_m, \ldots, x_{m+n-1})$$
$$\sigma_{ij}(s)^* := s^* \wedge x_i = x_j$$
$$\pi_{u_0, \ldots, u_{n-1}}(s)^* := \exists \bar{y} \left[s^*(\bar{y}) \wedge \bigwedge_{i<n} x_i = y_{u_i} \right]$$

Expressive Power

Theorem
Relational Algebra = First-Order Logic

Proof
($\geq$)

$$R(x_{u_0}, \ldots, x_{u_{n-1}})^* := \pi_{\bar{v}}(R \times \text{All} \times \cdots \times \text{All})$$

$$v_i := \begin{cases} k & \text{if } i = u_k \\ n + i & \text{otherwise} \end{cases}$$

$$(x_i = x_j)^* := \sigma_{ij}(\text{All} \times \cdots \times \text{All})$$

$$(\varphi \wedge \psi)^* := \varphi^* \cap \psi^*$$

$$(\varphi \vee \psi)^* := \varphi^* \cup \psi^*$$

$$(\neg \varphi)^* := \text{All} \times \cdots \times \text{All} \smallsetminus \varphi^*$$

$$(\exists x_i \varphi)^* := \pi_{0,\ldots,i-1,n,i+1,\ldots,n-1}(\varphi^* \times \text{All})$$

Datalog

Simplified version of Prolog developed in database theory:

- no function symbols,
- no cut, no negation, etc.

A **datalog program** for a database $\mathcal{A} = \langle A, R_0, \ldots, R_n \rangle$ is a set of Horn formulae

$$p_0(\bar{X}) \leftarrow q_{0,0}(\bar{X}, \bar{Y}) \wedge \cdots \wedge q_{0,m_0}(\bar{X}, \bar{Y})$$
$$\vdots$$
$$p_n(\bar{X}) \leftarrow q_{n,0}(\bar{X}, \bar{Y}) \wedge \cdots \wedge q_{n,m_n}(\bar{X}, \bar{Y})$$

where $p_0, \ldots, p_n$ are **new** relation symbols and the q_{ij} are either relation symbols from $\mathcal{A}$, possibly negated, or one of the new symbols p_k (not negated).

Datalog queries

The **query** defined by a datalog program

$$p_0(\bar{X}) \leftarrow q_{0,0}(\bar{X}, \bar{Y}) \wedge \cdots \wedge q_{0,m_0}(\bar{X}, \bar{Y})$$
$$\vdots$$
$$p_n(\bar{X}) \leftarrow q_{n,0}(\bar{X}, \bar{Y}) \wedge \cdots \wedge q_{n,m_n}(\bar{X}, \bar{Y})$$

maps a database $\mathcal{A}$ to the relations $p_0, \ldots, p_n$ defined by these formulae.

Evaluation strategy

▶ Start with empty relations $p_0 = \varnothing, \ldots, p_n = \varnothing$.

▶ Apply each rule to add new tuples to the relations.

▶ Repeat until no new tuples are generated.

Note

The relations computed in this way satisfy the **Completed Database** assumption.

Example

$$path(X, Y) \leftarrow edge(X, Y)$$
$$path(X, Y) \leftarrow path(X, Z) \wedge path(Z, Y)$$
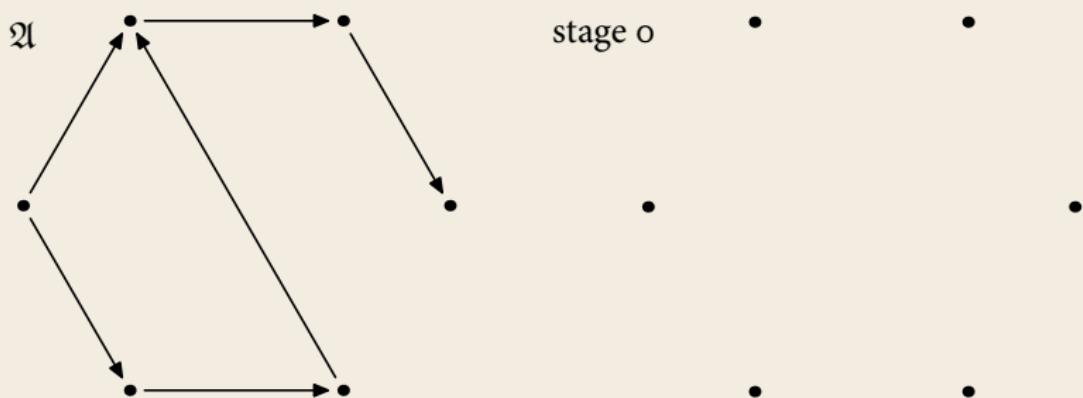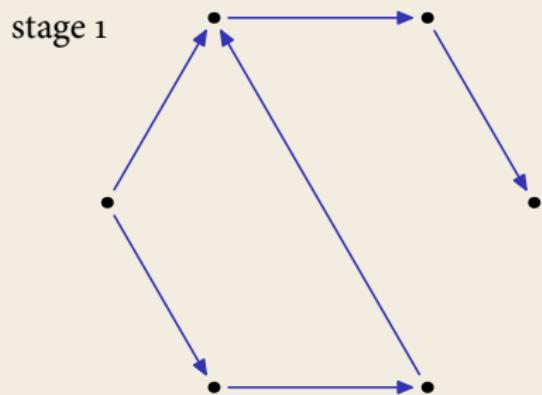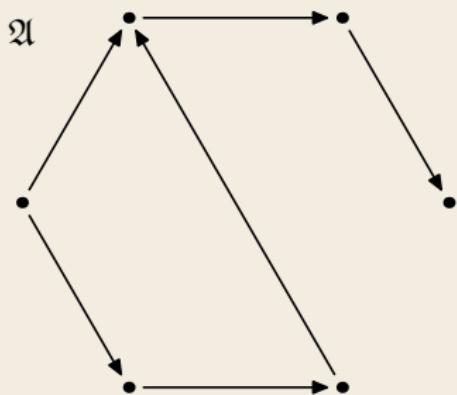
$\mathfrak{A}$

stage 0

Example

$$\text{path}(X, Y) \leftarrow \text{edge}(X, Y)$$
$$\text{path}(X, Y) \leftarrow \text{path}(X, Z) \land \text{path}(Z, Y)$$

Example

$$\text{path}(X, Y) \leftarrow \text{edge}(X, Y)$$
$$\text{path}(X, Y) \leftarrow \text{path}(X, Z) \land \text{path}(Z, Y)$$
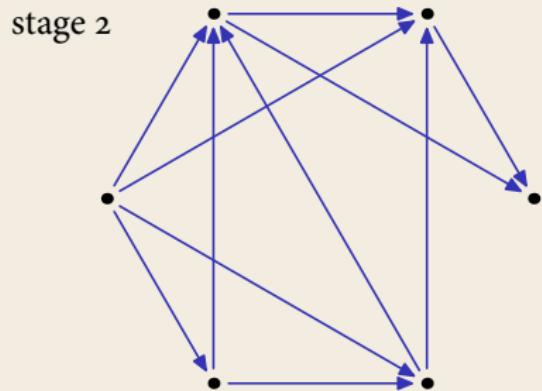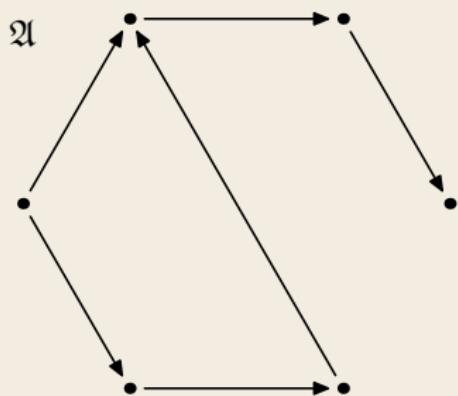
Example

$$\text{path}(X, Y) \leftarrow \text{edge}(X, Y)$$
$$\text{path}(X, Y) \leftarrow \text{path}(X, Z) \land \text{path}(Z, Y)$$
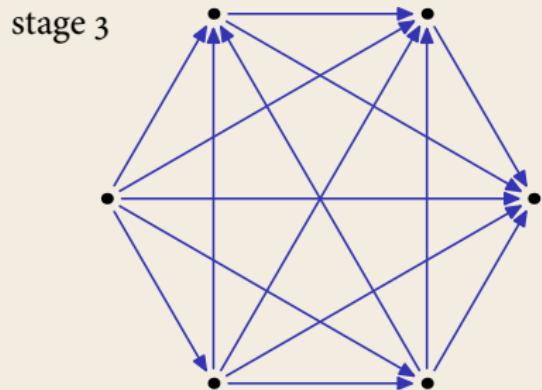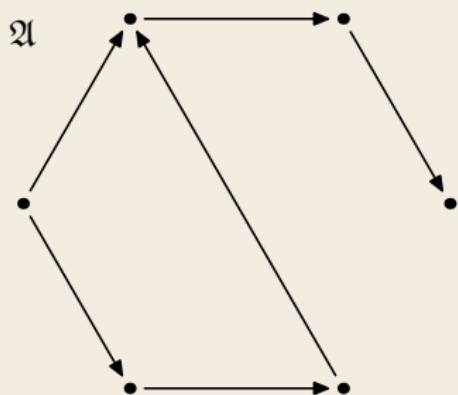
Example: Arithmetic

$$\text{Add}(x, y, z) \leftarrow y = 0 \wedge x = z$$
$$\text{Add}(x, y, z) \leftarrow E(y', y) \wedge E(z', z) \wedge \text{Add}(x, y', z')$$
$$\text{Mul}(x, y, z) \leftarrow y = 0 \wedge z = 0$$
$$\text{Mul}(x, y, z) \leftarrow E(y', y) \wedge \text{Add}(x, z', z) \wedge \text{Mul}(x, y', z')$$

stage 0   ∅

Example: Arithmetic

$$\text{Add}(x, y, z) \leftarrow y = 0 \wedge x = z$$
$$\text{Add}(x, y, z) \leftarrow E(y', y) \wedge E(z', z) \wedge \text{Add}(x, y', z')$$
$$\text{Mul}(x, y, z) \leftarrow y = 0 \wedge z = 0$$
$$\text{Mul}(x, y, z) \leftarrow E(y', y) \wedge \text{Add}(x, z', z) \wedge \text{Mul}(x, y', z')$$

stage 0 $\varnothing$
stage 1 $(k, 0, k)$

Example: Arithmetic

$$\text{Add}(x, y, z) \leftarrow y = 0 \wedge x = z$$
$$\text{Add}(x, y, z) \leftarrow E(y', y) \wedge E(z', z) \wedge \text{Add}(x, y', z')$$
$$\text{Mul}(x, y, z) \leftarrow y = 0 \wedge z = 0$$
$$\text{Mul}(x, y, z) \leftarrow E(y', y) \wedge \text{Add}(x, z', z) \wedge \text{Mul}(x, y', z')$$

stage 0	$\varnothing$
stage 1	$(k, 0, k)$
stage 2	$(k, 0, k), (k, 1, k+1)$

Example: Arithmetic

$$\text{Add}(x, y, z) \leftarrow y = 0 \wedge x = z$$
$$\text{Add}(x, y, z) \leftarrow E(y', y) \wedge E(z', z) \wedge \text{Add}(x, y', z')$$
$$\text{Mul}(x, y, z) \leftarrow y = 0 \wedge z = 0$$
$$\text{Mul}(x, y, z) \leftarrow E(y', y) \wedge \text{Add}(x, z', z) \wedge \text{Mul}(x, y', z')$$

stage 0	$\varnothing$
stage 1	$(k, 0, k)$
stage 2	$(k, 0, k), (k, 1, k + 1)$
stage 3	$(k, 0, k), (k, 1, k + 1), (k, 2, k + 2)$

Example: Arithmetic

$$\text{Add}(x, y, z) \leftarrow y = 0 \wedge x = z$$
$$\text{Add}(x, y, z) \leftarrow E(y', y) \wedge E(z', z) \wedge \text{Add}(x, y', z')$$
$$\text{Mul}(x, y, z) \leftarrow y = 0 \wedge z = 0$$
$$\text{Mul}(x, y, z) \leftarrow E(y', y) \wedge \text{Add}(x, z', z) \wedge \text{Mul}(x, y', z')$$

stage 0	$\varnothing$
stage 1	$(k, 0, k)$
stage 2	$(k, 0, k), (k, 1, k + 1)$
stage 3	$(k, 0, k), (k, 1, k + 1), (k, 2, k + 2)$
	...
stage n	$(k, 0, k), (k, 1, k + 1),\ldots, (k, n - 1, k + n - 1)$
	...

Complexity

Theorem

For databases $\mathfrak{A} = \langle A, \bar{R}, \leq \rangle$ equipped with a linear order $\leq$, a query Q can be expressed as a Datalog program if, and only if, it can be evaluated in **polynomial type.**