... now, back into the database area:

Chapter 10
Datalog Knowledge Bases I

In this section:

- Nonrecursive Datalog with Negation:
  equivalent to the relational algebra, to the relational calculus and to SQL.
- Stratified Recursive Datalog with Negation
  equivalent to the relational algebra or SQL with recursion (e.g., transitive closure)

In later sections:

- the really new things: well-founded and stable model semantics.

CONJUNCTIVE QUERIES

- $F(X_1, \ldots, X_n) = \exists Y_1, \ldots, Y_m : p_1(\ldots) \wedge \ldots \wedge p_k(\ldots)$ (note constants and variables may occur in the p_i arguments)
- Note: most systems allow also atomic comparisons over built-in datatypes:
  $F(X_1, \ldots, X_n) = \exists Y_1, \ldots, Y_m : p_1(\ldots) \wedge \ldots \wedge p_k(\ldots) \wedge$ *atomic comparisons*
- equivalent: SPJR-Algebra (selection, projection, join, renaming)
- SQL: broad `SELECT` $X_1, \ldots, X_n$ `FROM` $p_1, \ldots, p_n$ `WHERE` $cond$
  where $cond$ contains the join conditions and selection conditions
- efficient evaluation using indexes etc.
- Restricted expressiveness:
  - only very restricted negation (if at all) of the form $x_i \theta x_j$, $x_i \theta c$ where $\theta \in \{\neq, <, \leq, >, \geq\}$
  - no negation/set difference,
  - no universal quantification,
  - no disjunction/set union,
  - no recursion/no transitive closure.

XSB: LET'S START WITH CONJUNCTIVE QUERIES

- a PROLOG dialect developed at State Univ. of NY at Stony Brook (SUNYSB).
  (so one can actually do everything that is allowed in PROLOG, but we use only Datalog)
- XSB extends the original SB-PROLOG with tabled resolution and HiLog (higher-order logic programming).
- open source: `http://xsb.sourceforge.net/`

Starting XSB at IFI

Installed in the CIP Pool:

- `alias xsb='rlwrap ~dbis/LP-Tools/XSB/bin/xsb'` ← put this into .bashrc
- `user@bla:~$ xsb`

```
[xsb_configuration loaded]
[sysinitrc loaded]
XSB Version 3.3.4 (Pignoletto) of July 2, 2011
[i686-pc-linux-gnu 32 bits; mode: optimal; engine: slg-wam; scheduling: local]
[Patch date: 2011/07/08 04:32:08]
| ?-
```

- `?- [mondial].` loads the content of a file (from the current directory).
- `?- country(CN,C,Pop,Area,Cap,CapProv).` state a query
- <return>to return to XSB shell
- any key + <return>to get next answer
- CTRL-D: leave XSB

Datalog Syntax

Consider a CQ with only atoms in the body (i.e., positive!)

$$F(X_1, \ldots, X_n) = \exists Y_1, \ldots, Y_m : cq(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$$

Write

$$\texttt{?- } cq(X_1, \ldots, X_n, \_Y_1, \ldots, \_Y_m).$$

where

- the X_i are the free variables,
- replace Y_i by $\_Y_i$ if it occurs in more than one atom,
- replace Y_i by _ if it occurs only once ("don't-care-variables").

Example: countries whose population is > 1000000 and the capital population is not known:

```
?- country(CN,C,_Cap,_CapProv,_,_Pop), city(_Cap,C,_CapProv,null,_,_,_),
    _Pop > 1000000.
```

Note: null is not a built-in XSB term, but just a constant like 1, bla, 'Bla'.

10.1 Datalog Positive Conjunctive Queries – Formal Semantics

Definition 10.1

Given a relational schema $\mathbf{R}$ *and a (safe) "pure" CQ with only relational atoms in the body (i.e., positive!, no comparisons)*

$$F(X_1, \ldots, X_n) = \exists Y_1, \ldots, Y_m : r_1(\bar{u}_1) \wedge \ldots \wedge r_k(\bar{u}_k) \qquad r_i \in \mathbf{R}$$

whose Datalog syntax is $q(X_1, \ldots, X_n)$:- $r_1(\bar{v}_1), \ \ldots, \ r_k(\bar{v}_k)$.
(note that the $\bar{v}_j$ *contain* X_i *and "*$\_Y_i$*"-variables, the "_" don't-care, and constants),*
its answer relation wrt. a database state $\mathcal{S}$ *is*

$$\mathcal{S}(q) := \{(\beta(X_1), \ldots, \beta(X_n)) \mid \beta(\bar{u}_i) \in \mathcal{S}(r_i) \text{ for all } 1 \leq i \leq k\} .$$

□

Proposition 10.1

- $\mathcal{S}(q)$ *contains only values from* $ADOM(\mathcal{S} \cup q)$,
- *For positive conjunctive queries, the Datalog semantics coincides with the classical FOL semantics:*

$$\mathcal{S}(q) := \{(\beta(X_1), \ldots, \beta(X_n)) \mid \mathcal{S} \models_\beta F(X_1, \ldots, X_n)\}$$

□

10.2 Positive Datalog: Views as Rules

Conjunctive queries as View Definitions

A Datalog "knowledge base" $\mathcal{K}$ (also called a Datalog program) consists of

- facts of the form: $r(c_1, \ldots, c_n)$ (SQL equivalent: the tuples in the database),
- rules of the form $p(X_1, \ldots, X_k) \leftarrow \exists X_{k+1}, \ldots, X_n : Q(X_1, \ldots, X_n)$
  where p is a k-ary predicate and Q is a conjunctive (positive!) query.
  - means: "whenever $Q(X_1, \ldots, X_n)$ holds for some $X_{k+1}, \ldots, X_n$, also $p(X_1, \ldots, X_k)$ is assumed to hold".
  - SQL equivalent: p is a view.

The signature Σ is partitioned into two sets:

- Σ_{EDB}: predicates that occur only in the body of rules
  ("extensional database" – the interpretation of these predicates is given as facts in the knowledge base)
- Σ_{IDB}: predicates that occur in the head (and possibly also in the body) of rules
  ("intensional database" – the interpretation of these predicates is derived from the rules)

XSB Example

- compiler directive: `:- include(`*`filename`*`).`
  (note: no "-" in the filename allowed)
- comments: %

```
:- include(mondial).
% if special characters in filename: include('bla-blubb.P')
europeanCountry(C) :- encompasses(C,'Europe',_).
asianCountry(C) :- encompasses(C,'Asia',_).
result(O) :- europeanCountry(C), country(_,C,Cap,CapP,_,_), isMember(C,O,_),
             organization(O,_,Cap,C,CapP,_).
result(O) :- asianCountry(C), country(_,C,Cap,CapP,_,_), isMember(C,O,_),
             organization(O,_,Cap,C,CapP,_).
```

[Filename: Datalog/headquartercaps.P]

```
> xsb
?- [headquartercaps].
?- result(X).
X = EU
```

SEMANTICS OF A DATALOG KNOWLEDGE BASE

The formal semantics is given by *Herbrand Interpretations* (cf. Slide 497):

Herbrand Interpretation

- the domain consists of constant symbols and datatype literals.
- an interpretation $\mathcal{H}$ is explicitly seen as a *finite* set of ground atoms over the predicate symbols and the Herbrand Domain:
  country(ger,"Germany","D", berlin, 356910,83536115), encompasses(ger, eur, 100).

$$\mathcal{H} \models \text{encompasses(ger,eur,100)} \quad \text{if and only if} \quad \text{(ger, eur,100)} \in \text{encompasses}$$
$$\text{if and only if} \quad \text{encompasses(ger, eur,100)} \in \mathcal{H} \ .$$

Examples

- {country(ger,"Germany","D", berlin, 356910,83536115), country(aut, "Austria", "A", vienna, 83850,8023244), . . . , border(aut,ger,784), border(aut,hun,366), ...}
- the file `mondial.P` has the same schema as Mondial for SQL and uses only atomic values with keys/foreign keys.

Three Approaches to Semantics

- a model-theoretic approach (that differs from traditional FOL model theory),
- a fixpoint approach (effectively computable, "bottom-up"),
- a proof-theoretic approach (efficiently computable, "top-down", same as for PROLOG)

$\Rightarrow$ all of them turn out to be equivalent.

10.2.1 The Fixpoint Approach to Positive Datalog

Consider a *positive* program (i.e., rules without negation).

- facts of the form $p(a_1, \ldots, a_n)$ can also be seen as rules:
  $p(a_1, \ldots, a_n)$:- true
  "if true holds (which is always the case) then also $p(a_1, \ldots, a_n)$ must hold".
- application of rules:
  The set of ground atoms that is derivable by a rule $H \leftarrow B_1 \wedge \ldots \wedge B_k$ wrt. a given Herbrand Interpretation $\mathcal{H}$ is formally specified as follows:

  $\{\ \sigma(H)$: σ is a ground substitution and there is a rule
  $H \leftarrow B_1 \wedge \ldots \wedge B_k$ in P such that $\sigma(B_1), \ldots, \sigma(B_k) \in \mathcal{H}\ \}$

Example

Let $\mathcal{H}$ contain the facts from mondial.P. The rule

orgOnCont(O,Cont) :- isMember(C,O,_), encompasses(C,Cont,_).

with $\sigma = \{$C $\mapsto$ "D", O $\mapsto$ "EU", Cont $\mapsto$ "Europe"$\}$ where isMember("D","EU","member") $\in \mathcal{H}$ and where encompasses("D","Europe",100) $\in \mathcal{H}$ derives the atom orgOnCont("EU","Europe").

Bottom-Up-Semantics of Positive Datalog Programs

Consider a *positive* program P (i.e., facts, and rules without negation).

- (ground (i.e. without variables)) facts of the form $p(a_1, \ldots, a_n)$,
- (non-ground) rules of the form $head$:- $body$.

Definition 10.2 (T_P-Operator)
For a (positive) Datalog program P and a set I of ground atoms,

$T_P(I) := \{\ \sigma(H)$: *σ is a ground substitution and there is a rule*
$H \leftarrow B_1 \wedge \ldots \wedge B_k$ *in P such that* $\sigma(B_1), \ldots, \sigma(B_k) \in I\ \}$

$T_P(I)$ is called the "immediate consequence operator" since it takes I and applies the rules once.

$$\begin{aligned} T_P^0(I) &:= I \\ T_P^1(I) &:= T_P(I) \\ T_P^{n+1}(I) &:= T_P(T_P^n(I)) \\ T_P^\omega(I) &:= \bigcup_{n \in \mathbb{N}} T_P^n(I) \qquad \textit{infinite union!} \end{aligned}$$

- $T_P^\omega := T_P^\omega(\emptyset)$ – *usually, start with $\emptyset$*
- *Intuition: The set T_P^ω contains all ground facts that can be derived from the program.*
- *note: $T_P^1(\emptyset)$ contains the ground facts listed in the program.* □

T_P: Some Straightforward Examples

- Consider the program $P = \{\mathsf{p},\ \mathsf{q} \leftarrow \mathsf{p},\ \mathsf{r} \leftarrow \mathsf{q},\ \mathsf{s} \leftarrow \mathsf{r} \wedge \mathsf{q}\}$:

  $T_P^1(\emptyset) = T_P(\emptyset) = \{\mathsf{p}\},$

  $T_P^2(\emptyset) = T_P(\{\mathsf{p}\}) = \{\mathsf{p}, \mathsf{q}\},$ – note: p is derived *again*

  $T_P^3(\emptyset) = T_P(\{\mathsf{p}, \mathsf{q}\}) = \{\mathsf{p}, \mathsf{q}, \mathsf{r}\}$

  $T_P^4(\emptyset) = T_P(\{\mathsf{p}, \mathsf{q}, \mathsf{r}\}) = \{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}\}$

  $T_P^5(\emptyset) = T_P(\{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}\}) = \{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}\}$

- Consider the program $Q =$
  $\{\mathsf{p(1,2)},\ \mathsf{p(2,3)},\ \mathsf{p(3,4)},\ \mathsf{p(3,5)},\ \mathsf{p(1,6)},\ \mathsf{tc(X,Y)} \leftarrow \mathsf{p(X,Y)},\ \mathsf{tc(X,Y)} \leftarrow \mathsf{tc(X,Z)} \wedge \mathsf{p(Z,Y)}\}$:
  Let $EDB := T_P^1(\emptyset) = \{\mathsf{p(1,2)},\ \mathsf{p(2,3)},\ \mathsf{p(3,4)},\ \mathsf{p(3,5)},\ \mathsf{p(1,6)}\}$ for the ground facts.

  $T_Q^2(\emptyset) = EDB \cup \{\mathsf{tc(1,2)},\ \mathsf{tc(2,3)},\ \mathsf{tc(3,4)},\ \mathsf{tc(3,5)},\ \mathsf{tc(1,6)}\},$

  $T_Q^3(\emptyset) = EDB \cup \{\mathsf{tc(1,2)},\ \mathsf{tc(2,3)},\ \mathsf{tc(3,4)},\ \mathsf{tc(3,5)},\ \mathsf{tc(1,6)},\ \mathsf{tc(1,3)},\ \mathsf{tc(2,4)},\ \mathsf{tc(2,5)}\},$

  $T_Q^4(\emptyset) = EDB \cup \{\mathsf{tc(1,2)},\ \mathsf{tc(2,3)},\ \mathsf{tc(3,4)},\ \mathsf{tc(3,5)},\ \mathsf{tc(1,6)},\ \mathsf{tc(1,3)},\ \mathsf{tc(2,4)},\ \mathsf{tc(2,5)},$
  $\mathsf{tc(1,4)},\ \mathsf{tc(1,5)}\} = T_Q^5(\emptyset)$

T_P: Non-Straightforward Examples

Obvious: (Positive) programs with no facts will not derive anything when started with $\emptyset$.

- Consider the program $P = \{\mathsf{p} \leftarrow \mathsf{p}\}$:
  - $T_P^1(\emptyset) = T_P(\emptyset) = \emptyset = T_P^2(\emptyset) = T_P^\omega(\emptyset)$
  - when not starting with $\emptyset$, but with {p}:
    $T_P^1(\{\mathsf{p}\}) = \{\mathsf{p}\} = T_P^2(\{\mathsf{p}\}) = T_P^\omega(\{\mathsf{p}\})$
- Consider the program $P = \{\mathsf{p} \leftarrow \mathsf{q},\ \mathsf{q} \leftarrow \mathsf{p},\ \mathsf{r} \leftarrow \mathsf{p} \wedge \mathsf{q}\}$:
  - $T_P^1(\emptyset) = \{\emptyset\} = T_P^2(\emptyset) = T_P^\omega(\emptyset)$.
  - $T_P^1(\{\mathsf{p}, \mathsf{q}\}) = \{\mathsf{p}, \mathsf{q}, \mathsf{r}\} = T_P^2(\{\mathsf{p}, \mathsf{q}\}) = T_P^\omega(\{\mathsf{p}, \mathsf{q}\})$.
  - $T_P^1(\{\mathsf{p}\}) = \{\mathsf{q}\}$,
    $T_P^2(\{\mathsf{p}\}) = T_P(T_P^1(\{\mathsf{q}\})) = \{\mathsf{p}\}$,
    $T_P^3(\{\mathsf{p}\}) = T_P(T_P^2(\{\mathsf{p}\})) = T_P(\{\mathsf{p}\}) = \{\mathsf{q}\}$,
    ... the sequence then alternates ...
    $T_P^\omega(\{\mathsf{p}\}) = \bigcup_{n \in \mathbb{N}} T_P^n(\{\mathsf{p}\}) = \{\mathsf{p}, \mathsf{q}\}$, which is *not a model of* P*!* (r is missing!)
  - $T_P^1(\{\mathsf{r}\}) = \{\emptyset\} = T_P^2(\{\mathsf{r}\}) = T_P^\omega(\{\mathsf{r}\})$.

$\Rightarrow$ Starting with $I \neq \emptyset$ might show strange behaviour.
Don't do that. The argument is used only for the iteration $T_P(T_P^n(\emptyset))$.

Some Theoretical Properties of T_P

Proposition 10.2
$T_P^\omega \models P$. □

Proof:

- for all facts $R(c_1, \ldots, c_n)$ contained in P, $T_P^\omega \models R(c_1, \ldots, c_n)$, i.e., $R(c_1, \ldots, c_n) \in T_P^\omega$ ($R(c_1, \ldots, c_n) \in T_P^1(\emptyset)$).
- for all rules $p(X_1, \ldots, X_k) \leftarrow \exists X_{k+1}, \ldots, X_n : Q(X_1, \ldots, X_n)$ contained in P, $T_P^\omega \models \forall X_1, \ldots, X_n : p(X_1, \ldots, X_k) \leftarrow \exists X_{k+1}, \ldots, X_n : Q(X_1, \ldots, X_n)$:
  If $T_P^n(\emptyset) \models_\beta \exists X_{k+1}, \ldots, X_n : Q(X_1, \ldots, X_n)$, then $T_P^{n+1}(\emptyset) \models_\beta p(X_1, \ldots, X_k)$ by definition of T_P.

... so T_P^ω looks good. Is it special? What about the infinite union?

Some Theoretical Properties of T_P

Proposition 10.3
T_P is monotonous (recall, for positive *P), i.e., if $I_1 \subseteq I_2$ then $T_P(I_1) \subseteq T_P(I_2)$.* □

- As a consequence of this, $T_P^{n+1}(\emptyset) \supseteq T_P^n(\emptyset)$.
- $\bigcup_{n \in \mathbb{N}} T_P^n(\emptyset) = \lim_{n \to \infty} T_P^n(\emptyset)$ ... (infinite?) iteration – does it stop somewhere?
- Let $\mathcal{HB}_P$ denote the Herbrand Base of P, i.e., the set of all ground instances of predicates in P over the Herbrand universe (which consists of all constants occurring in the atoms in P).
  Then, $\mathcal{HB}_P \models P$ for every positive P.
- $T_P^n(\emptyset) \subseteq \mathcal{HB}_P$ for all $n \in \mathbb{N}$.

..... a monotonously growing sequence is bounded from above:

Theorem 10.1
For some (finite) $n \in \mathbb{N}$, a fixpoint*, i.e., $T_P(T_P^n(\emptyset)) = T_P^n(\emptyset)$ is reached after finitely many steps. For this n, $T_P^\omega = T_P^n(\emptyset)$.* □

- T_P^ω can effectively be computed ("bottom-up"),
- queries are then stated against T_P^ω.

Some Theoretical Properties of T_P

This is the general definition of the term “fixpoint”:

Definition 10.3
For an operator Ψ mapping from any mathematical domain $\mathcal{X}$ to $\mathcal{X}$, a fixpoint is any x such that $\Psi(x) = x$. □

Example:
$\sqrt{\cdot}$ is an operator $\mathbb{R} \to \mathbb{R}$. $\sqrt{1} = 1$ is a fixpoint of it.
T_P is an operator from sets of ground atoms (i.e., Herbrand interpretations) to sets of ground atoms.

Some Theoretical Properties of T_P

Proposition 10.4
For a Datalog program P,

a) every fixpoint $\mathcal{F}$ of T_P, i.e., $T_P(\mathcal{F}) = \mathcal{F}$, is a model of P (but not every model is a fixpoint!), and

b) for every model $\mathcal{H}$ of a Datalog program P, $T_P(\mathcal{H}) \subseteq \mathcal{H}$. □

Proof:

a) Since $\mathcal{F}$ is a fixpoint, $T_P(\mathcal{F}) \supseteq \mathcal{F}$, i.e. it contains all facts in P, and all instances of heads of applicable rule instances. Thus, it is a model of P.

b) by definition of T_P: $\mathcal{H}$ is a model of P, so it already contains all ground instances of heads of applicable rule instances.

Note: a model can also contain additional ground atoms (=facts) that are not required (“supported”) by the program, as long as it contains also their consequences. It is still a model.
[Example see next slide]

Outlook: the “Minimal Model” will be a distinguished model (later, the “Well-Founded Model” and “Stable Models” continue this idea of minimality).

Models of a Program

Further models of a program can be obtained by adding additional facts (they must be complete wrt. consequences from these).

Example 10.1

Consider $P = \{$*q(X) :- p(X); r(X) :-q(X); p(a)*$\}$.

- *Let* $\mathcal{M} := T_P^\omega(\emptyset) = \{p(a), q(a), r(a)\}$.
- *Other, bigger models are* $\mathcal{M}_1 = \{p(a), q(a), r(a), r(b)\}$ *and* $\mathcal{M}_2 = \{p(a), q(a), r(a), q(b), r(b)\}$.
- *Note that* $\mathcal{N} = \{p(a), q(a), r(a), q(b)\} \supsetneq \mathcal{M}$*, but it is not a model. Since* $\mathcal{N} \subseteq \mathcal{M}_2$*, it can obviously be extended to a model (cf. Slide 579).*
- *The* $\mathcal{M}_i$ *are not fixpoints of* T_P*:* $T_P(\mathcal{M}_1) = \{p(a), q(a), r(a)\} = \mathcal{M} \subsetneq \mathcal{M}_1$*, and* $T_P(\mathcal{M}_2) = \{p(a), q(a), r(a), r(b)\} = \mathcal{M}_1 \subsetneq \mathcal{M}_2$.
  - *in both cases, according to Proposition 10.4,* $T_P(\mathcal{M}_i) \subseteq \mathcal{M}_i$*, shows that they are models, i.e., all rules are satisfied.*
  - *The "*$\subsetneq$*" shows that some fact has been "invented" which is not forced ("supported") by the rules.*
- *Usually, fixpoints which are non-minimal models occur if the program contains some "self-supporting" rule* $p \leftarrow p$.

□

Some Theoretical Properties of T_P

Definition 10.4

For two Herbrand interpretations, $\mathcal{H}_1$ *and* $\mathcal{H}_2$*,* $\mathcal{H}_1 \leq \mathcal{H}_2$ *if* $\mathcal{H}_1 \subseteq \mathcal{H}_2$.

□

Proposition 10.5

T_P^ω *is the least fixpoint of* T_P.

□

Proof:

By Proposition 10.4, every fixpoint $\mathcal{F}$ is a model of P. To be a model of P, $\mathcal{F}$ contains all facts in P, i.e., $\mathcal{F} \supseteq T_P(\emptyset)$. By induction, $\mathcal{F} \supseteq T_P^n(\emptyset)$ for each $n \in \mathbb{N}$. Thus, $\mathcal{F} \supseteq T_P^\omega$.

(the full PROLOG case, where the $\mathcal{HB}_P$ argument does not hold and T_P^ω is not necessarily finite, follows from monotonicity by the Knaster-Tarski Theorem (fixpoint theory over complete lattices).)

Aside: T_P^ω in PROLOG

- PROLOG allows function symbols.
- Consider the program $P := \{\ p(a),\ \ p(f(X)) \leftarrow p(X)\ \}$: $T_P^\omega = \{p(f^n(a)) | n \in \mathbb{N}\}$ is infinite.

Example/Exercise

Consider the following (recursive) program (including atomic facts and rules):

$P = \{$ country(a). country(b). country(ch). country(d). country(e). country(f). ...
border(a, d). border(a, h). border(a, i). border(d, f). border(i, f).
border(ch, f). border(ch, a). border(ch, d). border(ch, i). border(e, f). border(p, e).
border(h, ua). border(ua, r). border(ra, br). border(bol, ra). border(bol, br).
border$(Y, X) \leftarrow$ border(X, Y).
reachable$(X, Y) \leftarrow$ border(X, Y).
reachable$(X, Y) \leftarrow$ reachable(X, Z), border(Z, Y). $\}$

- Give $T_P^0(\emptyset)$, $T_P^1(\emptyset)$, $T_P^2(\emptyset)$, ..., $T_P^\omega(\emptyset)$.
- for any derived fact reachable$(c_1, c_2) \in T_P^\omega(\emptyset)$, characterize the least i such that reachable$(c_1, c_2) \in T_P^i(\emptyset)$.

10.2.2 Model-Theoretic Characterization: Minimal Model

- Note: simple "Datalog" usually means "positive Datalog"

Definition 10.5
For a (positive) Datalog program P, the minimal model *is defined as the smallest Herbrand interpretation (wrt. $\leq$ as in Def. 10.4) that is a model of P.* □

Theorem 10.2
For a positive Datalog program P and its minimal model $\mathcal{M}$*, for all ground atoms $p(c_1, \ldots, c_n)$:*

- $\mathcal{M} \models p(c_1, \ldots, c_n) \Leftrightarrow p(c_1, \ldots, c_n) \in T_P^\omega$.
- $\mathcal{M} \models p(c_1, \ldots, c_n)$ *if and only if for* all *models $\mathcal{S}$ of P, $\mathcal{S} \models p(c_1, \ldots, c_n)$.*

(recall: $\models$ denotes the models-relation from First Order Logic) □

Proposition 10.6
The minimal model $\mathcal{M}$ of a (positive) Datalog program P is the intersection of all models (i.e., models wrt. First Order Logic model theory) of P. □

Proof: same as for Proposition 10.5.

Non-minimal Models

Let P a positive Datalog program with minimal model $\mathcal{M} = T_P^\omega = T_P^\omega(\emptyset)$, and $q \notin \mathcal{M}$ some ground atom.

- there exists a model $\mathcal{M}'$ of P that makes q true.
  (i.e., a positive program cannot force anything to be false; there is only "negation by default").
- Recall Slide 570: starting with q, i.e., $T_P^\omega(\{q\})$ is not appropriate (it might forget q, or even run into an alternating sequence).
- Compute $\mathcal{M}'' = T_{P\cup\{q\}}^\omega = T_{P\cup\{q\}}^\omega(\emptyset)$ to obtain the solution, which is the minimal model of $P \cup \{q\}$.
- For Example 10.1, $T_{P\cup\{q(b)\}}^\omega = \mathcal{M} \cup \{q(b), r(b)\}$.

Some comments on Negation

- Negative Literals:
  - The minimal model implements the *Closed-World-Assumption (CWA)*: any atom that is not contained or implied by P is assumed not to hold.
  - For the minimal model $\mathcal{M}$,
    if a ground atom is not in $\mathcal{M}$, i.e., $\mathcal{M} \models \neg p(a_1, \ldots, a_n)$, classical FOL semantics (open-world) does *not* entail that $P \models_{FOL} \neg p(a_1, \ldots, a_n)$.
    Note that $P \models_{FOL} \neg p(a_1, \ldots, a_n)$ does not hold for any ground atom – from a positive program P no negative statements are entailed at all under FOL semantics.
  - this coincides with the SQL semantics "`WHERE NOT EXISTS ...`".
- Negative literals in rule bodies:
  - The T_P evaluation is not applicable for rules with negation in the body.
  - Consider the previous example extended by the rule
    $\{$ unreachable$(X, Y) \leftarrow$ country$(X) \wedge$ country$(Y) \wedge \neg$reachable(X, Y). $\}$.
    How would the T_P evaluation proceed for it?
- derivation of negative facts/negative facts in rule heads:
  not applicable since CWA assumes all negative facts that are consistent with P ("negation by default")

10.2.3 Proof-Theoretic Approach: Resolution Calculus

Given: a positive Datalog program P
Question: does $p(c_1, \ldots, c_n)$ hold?

- bottom-up computation of T_P provides a *correct* and *complete* (wrt. the minimal model) procedure for checking if some *fact* holds in the minimal model.

Every atom that is true in the minimal model has a "proof history" (tree) via the rules and facts that have been used for deriving it.

General Resolution Calculus

- an *Inference System*.
- a *clause* is a set of literals (semantics: disjunctive).
  Clause resolution takes two clauses that contain contradictory literals:

$$\frac{\ell_1 \vee \ldots \vee \boxed{\ell_i} \vee \ldots \vee \ell_k \quad , \quad \ell_{k+1} \vee \ldots \vee \boxed{\neg \ell_{k+j}} \vee \ldots \vee \ell_{k+m} \quad , \quad \boxed{\sigma(\ell_i) = \sigma(\ell_{k+j})}}{\sigma(\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee \ell_{k+1} \vee \ldots \vee \ell_{k+j-1} \vee \ell_{k+j+1} \vee \ldots \vee \ell_{k+m})}$$

- rules of the form

$$h(\bar{x}) \leftarrow b_1(\bar{x}) \wedge b_2(\bar{x}) \wedge \ldots \wedge b_n(\bar{x})$$

  are equivalent to *Horn Clauses* (named after the logician Alfred Horn)

$$h(\bar{x}) \vee \neg b_1(\bar{x}) \vee \neg b_2(\bar{x}) \vee \ldots \vee \neg b_n(\bar{x})$$

  (Disjunction with only one positive literal).

ASIDE: GENERAL RESOLUTION CALCULUS: COMMENTS AND EXAMPLES

- Tableau calculus:
  - one rule for each FOL construct ($\wedge, \vee, \forall, \exists$, and the closure rule as the rule for $\neg$).
  - applicable to all kinds of FOL formulas.

  $\Rightarrow$ intuitive, very general, but a high number of possible expansions in each step.
- Resolution calculus:
  - only a single inference rule,
  - applicable to *a set of (arbitrary) disjunctions*.
- Any FOL formula ϕ can be translated as follows:
  - Prenex Normal Form: pull quantifiers in front ("prefix"): $\forall a, b\, \exists c, d\, \forall e \ldots : \phi'$ where ϕ' is quantifier-free ("matrix"),
  - transform the matrix into conjunctive normal form (i.e., a conjunction of disjunctions of literals).

  $\Rightarrow$ resolution calculus has the same expressiveness as tableau calculus.
  - it is intuitive, if a problem has a natural representation as a set of disjunctions.

Disjunctive Reasoning: Sudoku

Typical Sudoku situation: "cell (x, y_1) is either 2 or 7, cell (x, y_2) is either 2 or 6, so the "2" can only be in one of them, there is 7 in (x, y_1) or 6 in (x, y_2). As 6 is already in (x_2, y_2), the 2 must be in (x, y_2), and the 7 must be in (x, y_1)."

Consider the following example (sudoku taken from (german) wikipedia):

	A	B	C	D	E	F	G	H	I
9		3							
8				1	9	5			
7			8					6	
6	8				6				
5	4			8					1
4					2				
3		6					2	8	
2	?	?	?	4	1	9		3	5
1								7	

- 3-ary predicate p ("position"), e.g. for B9: $p(b, 9, 3)$:
- exclusion clause patterns like:
  $\{\neg p(x_1, y, n), \neg p(x_2, y, n), x_1 = x_2\}$
  for rows; analogously for columns and for subsquares.
- H2: must be 3 (all other numbers are already present in column H, in row 2 or in the lower right subsquare).
- A2: 2 or 3 or 7.; B2: 2 or 7 or 8.
- C2: 2 or 3 or 7.

$\Rightarrow$ A2 and C2: 2 or 7 $\Rightarrow$ B2: 8

Sudoku (cont'd)

- Query: $answer(X, Y, N) \leftarrow p(X, Y, N).$
- Whenever the empty clause can be derived, an answer is given by the applied substitutions. E.g. for an already known cell, $\{\neg p(X, Y, N)\}$ with $\{p(b, 9, 3\}$ and $\sigma = \{X \leftarrow b, Y \leftarrow 9, N \leftarrow 3\}$ yields the first solution.
- Simple cases like H2 (must be 3):

$\{p(h,2,1), p(h,2,2), p(h,2,3), p(h,2,4), p(h,2,5),$
$p(h,2,6), p(h,2,7), p(h,2,8), p(h,2,9)\}$, $\{p(e,2,1)\}$, $\{\neg p(e,2,1), \neg p(h,2,1)\}$

$\{\neg p(h,2,1)\}$

$\{p(h,2,2), p(h,2,3), p(h,2,4), p(h,2,5), p(h,2,6), p(h,2,7), p(h,2,8), p(h,2,9)\}$,

$\vdots$

$\{p(h,2,3)\}$ which closes with the negated query $\{\neg p(X, Y, N)\}$

- analogous reduction for cells A2, B2, C2:
  - $\{p(a,2,2), p(a,2,3), p(a,2,7)\}$, with $\{\neg p(a,2,3), \neg p(h,2,3)\}$ and $\{p(h,2,3)\}$ to $\{p(a,2,2), p(a,2,7)\}$;
  - $\{p(b,2,2), p(b,2,7), p(b,2,8)\}$,
  - $\{p(c,2,2), p(c,2,3), p(c,2,7)\}$ analogously to $\{p(c,2,2), p(c,2,7)\}$

Sudoku (cont'd)

- Situation
  - $\{p(a,2,2), p(a,2,7)\}$;
  - $\{p(b,2,2), p(b,2,7), p(b,2,8)\}$,
  - $\{p(c,2,2), p(c,2,7)\}$
- Both A2 and C2 are 2 or 7
- B2 (2, 7, or 8) must be 8
- no direct conclusion possible ...
- note: resolving to clauses with 2 literals usually yields two literals:
  $\{a, b\}$ with $\{\neg b, c\}$ yields $\{a, c\}$.
  Unary clauses can be derived by matches like
  $\{a, b\}$ with $\{a, \neg b\}$ yields $\{a\}$.

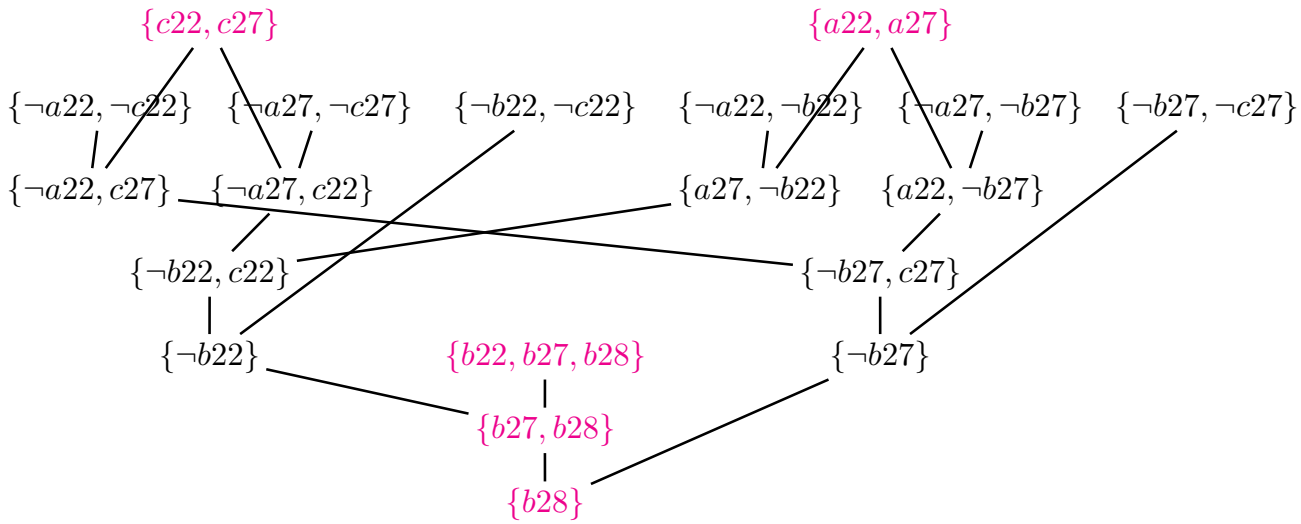
$\Rightarrow$ not only clauses that are connected by a pair of contradictory literals are interesting, but also clauses that contain the same literals can be useful.

$\Rightarrow$ a resolution reasoner maintains a connection graph for choosing its strategy.

- human reasoners must have a plan how to proceed ...

Sudoku (cont'd: Example proof for the contents of cell B2)

- write xyn for $p(x, y, n)$:
  - this is not only a notational shortcut, but also a mapping to Boolean Logic:
  - the second line is assumed to contain all ground instances of the exclusion clause (cf. Slide 584) stating which cells must not have the same value.
    Note: the smodels tool for stable models is based on the same idea of creating all ground instances and running boolean Model Checking.



General (FOL) Resolution Calculus

- Recall: open-world, with explicit negative literals.
- there are always multiple possibilities to choose pairs of clauses to be resolved

$\Rightarrow$ proof search strategy?

- ... not the right thing for deductive databases (closed-world-assumption, equivalence to the relational algebra and SQL),
- ... but a good basis ...
- ... go back first to consider positive rules as a special case of disjunction *head* $\vee \neg$*body*.

RESOLUTION CALCULUS FOR (POSITIVE) RULES

- a derivation rule $\mathit{head}(\bar{x}) \leftarrow b_1(\bar{x}) \wedge b_2(\bar{x}) \wedge \ldots \wedge b_n(\bar{x})$ is equivalent to $\neg b_1(\bar{x}) \vee \neg b_2(\bar{x}) \vee \ldots \vee \neg b_n(\bar{x}) \vee \mathit{head}(\bar{x})$, or, written as a Horn clause,
$$\{\neg b_1(\bar{x}), \neg b_2(\bar{x}), \ldots, \neg b_n(\bar{x}), \mathit{head}(\bar{x})\}$$
- such a Horn clause can be seen as a *directed* disjunction with a single distinguished positive (head) literal.
- a fact $p(\bar{c})$ corresponds to a unary clause consisting of a single positive literal $\{p(\bar{c})\}$.

Bottom-Up: Resolution as Forward Reasoning

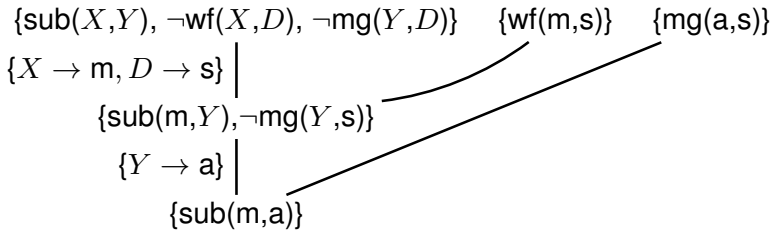
Example:

Consider the rule subordinate$(x, y) \leftarrow$ works-for$(x, d) \wedge$ manages(y, d)
(forget about $x \neq y$ for now)

The corresponding clause is

$$\{\text{subordinate}(X, Y), \neg\text{works-for}(X, D), \neg\text{manages}(Y, D)\} \ .$$

Consider the (unary) fact clauses {works-for(mary,sales)} and {manages(alice,sales)}.



... derives subordinate(mary,alice).

- obviously, for every ground atom $p(a_1, \ldots, a_n)$, $P \vdash_{\text{Res}} p(a_1, \ldots, a_n)$ if and only if $p(a_1, \ldots, a_n) \in T_P^\omega$.

TOP-DOWN: RESOLUTION CALCULUS AS BACKWARD REASONING

- used in PROLOG systems:
  *SLD Resolution (**S**election-Rule-Driven **L**inear Resolution for **D**efinite Clauses)*
- given: a "program" P of rules and facts, and a claimed fact $answer(\bar{c})$. Show: $P \models answer(\bar{c})$?
- Resolution as a refutation strategy: prove that $\neg answer(\bar{c})$ is inconsistent with P.
- a negated atom can be refuted if it matches the head of a rule and all of the body atoms of the rule can be proven. Apply recursively:
  - get a new "goal clause" (i.e., a clause containing only negative literals) [$\Rightarrow$] linear proof;
  - note that multiple rule heads can match (SLD: first rule first);
  - note that multiple literals can match: resolve literals from left to right (i.e., depth-first).
- try to derive the empty (goal) clause: then it is shown that $P \cup \{\neg answer(\bar{c})\}$ is unsatisfiable, i.e., $P \models answer(\bar{c})$.

SLD RESOLUTION: EXAMPLE

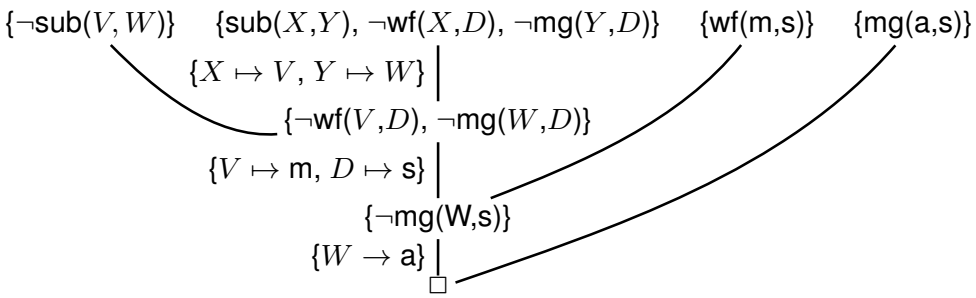
Consider again the rule

$$\text{subordinate}(x, y) \leftarrow \text{works-for}(x, d) \wedge \text{manages}(y, d)$$

and the corresponding clause

$$\{\text{subordinate}(X, Y), \neg\text{works-for}(X, D), \neg\text{manages}(Y, D)\} .$$

and e.g. ground fact clauses {works-for(mary,sales)} and {manages(alice,sales)}. For which V, W does subordinate(V,W) hold?



derives the answer substitution $\{V \mapsto \text{m}, W \mapsto \text{s}\}$
(in Prolog style written as $\{V/\text{m}, W/\text{s}\}$)

With a bigger database, further answers can be derived by other matches.

SLD Resolution for Answers

- the initial goal (=query) contains free variables,
- collect the union/concatenation of all substitutions applied
- if the empty clause is derived, the restriction of the resulting substitution to the variables in the query is the *answer substitution*.
- do backtracking (alternative closing substitutions with other facts, alternative rules with the same head),
- compute further answers.

SLD Resolution with Answers: Example

"All organizations that have their headquarters in the capital of a European member country with more than 10000000 inhabitants"

```
:- include(mondial).
europeanBigCountry(C) :- encompasses(C,'Europe',_),
                         country(_,C,_,_,_,Pop), Pop > 10000000.
hqInCapOf(O,C) :- country(_,C,Cap,CapP,_,_), organization(O,_,Cap,C,CapP,_).
result(O) :- europeanBigCountry(C), isMember(C,O,_), hqInCapOf(O,C).
?- result(X).
```

[Filename: Datalog/headquartercapsbig.P]

C_1 : $\{\neg\text{res}(X)\}$

C_2 : $\{\text{eBC}(C), \neg\text{enc}(C, \text{“Europe”}, \_), \neg\text{c}(\_, C, \_, \_, \_, P), \neg P > 10000000\}$

C_3 : $\{\text{hC}(O, C), \neg\text{c}(\_, C, Cap, CapP, \_, \_), \neg\text{org}(O, \_, Cap, C, CapP, \_)\}$

C_4 : $\{\text{res}(O), \neg\text{eBC}(C), \neg\text{isM}(C, O, \_), \neg\text{hC}(O, C)\}$

Resolve C_1 with C_4 (the only rule that matches) by $\sigma_1 : \{O \to X\}$:

C_5 : $\{\neg\text{eBC}(C), \neg\text{isM}(C, X, \_), \neg\text{hC}(X, C)\}$.

Resolve C_5 with C_2 (first literal):

C_6 : $\{\neg\text{enc}(C, \text{“Europe”}, \_), \neg\text{c}(\_, C, \_, \_, \_, P), \neg P > 10000000, \neg\text{isM}(C, X, \_), \neg\text{hC}(X, C)\}$.

Resolve C_6 with fact enc(“B”, “Europe”, 100) (one out of many candidates) by $\sigma_2 : \{C \rightarrow \text{“B”}\}$:

C_7 : $\{\neg\text{c}(\_, \text{“B”}, \_, \_, \_, P), \neg P > 10000000, \neg\text{isM}(\text{“B”}, X, \_), \neg\text{hC}(X, \text{“B”})\}$.

Resolve with fact c(“Belgium”,“B”,“Brussels”,“Brabant”, _, 10170241) by $\sigma_6 : \{P \rightarrow 10170241\}$ and remove the (false) instantiated literal $\neg 10170241 > 10000000$:

C_8 : $\{\neg\text{isM}(\text{“B”}, X, \_), \neg\text{hC}(X, \text{“B”})\}$.

Resolve with fact isM(“B”,“EU”,“member”) by $\sigma_4 : \{X \rightarrow \text{“EU”}\}$:

C_9 : $\{\neg\text{hC}(\text{“EU”}, \text{“B”})\}$.

Resolve with C_3 by $\sigma_5 : \{O \rightarrow \text{“EU”}, C \rightarrow \text{“B”}\}$:

C_{10} : $\{\neg\text{c}(\_, \text{“B”}, Cap, CapP, \_, \_), \neg\text{org}(\text{“EU”}, \_, Cap, \text{“B”}, CapP, \_)\}$.

Resolve with fact c(“Belgium”,“B”,“Brussels”,“Brabant”, _, _):

C_{11} : $\{\neg\text{org}(\text{“EU”},\text{“Brussels”},\text{“B”},\text{“Brabant”}, \_)\}$.

Resolve with fact org(“EU”, _, “Brussels”, “B”, “Brabant”, _) and obtain the empty clause.

This generates the first answer $X/$“EU”.

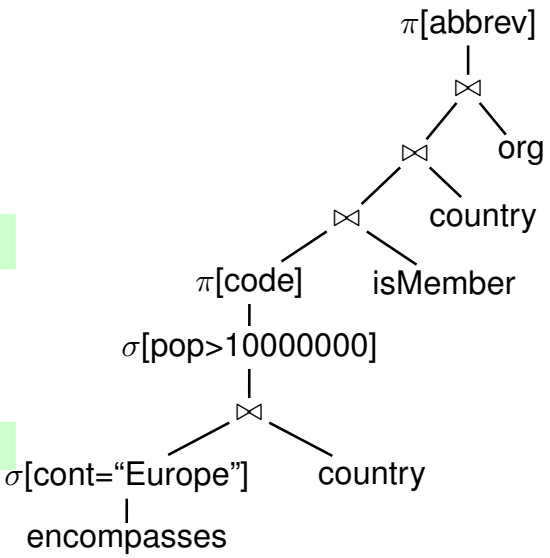
Backtracking ... resolve C_8 with fact isM(“B”, “UN”) to obtain

C_{11} : $\{\neg\text{hC}(\text{“UN”}, \text{“B”})\}$.

Resolve again with C_3 by $\{O \rightarrow \text{“UN”}, C \rightarrow \text{“B”}\}$ and continue as above. The empty clause cannot be derived (the headquarters of the UN are in New York). Backtrack again, resolve C_8 with NATO, return $X/$“NATO”, analogously check all organizations where Belgium is a member, and return all organizations located in Brussels.

Backtracking then to C_5, try the next european country etc.

- Note that all intermediate clauses only contain negative literals (“goal clauses”).
- at each timepoint there is exactly one (open) goal clause.



Comparison

The evaluation is actually an iterator-based evaluation of the algebra tree shown on the right.

Exercise

Do the same for European and Asian Big Countries.

10.3 Aside: Full Prolog

- allows function symbols
- not just matching, but *unification* of terms (that contain variables someewhere):
  $p(f(X, g(Y)))$ unifies with $\neg p(f(h(Z), Z))$ via $\sigma = \{Z \to g(Y), X \to h(g(Y))\}$.
- derives the empty clause *and* an answer substitution
  e.g. when asking ?-subordinate(X,alice).

  X/mary

  X/bob
- uses backtracking:
  - if search for an answer is not successful, try another way,
  - if an answer is found, report it and try another way (next substitution, next rule),
  - generates a proof search tree.
- Prolog *Programming* goes even further: "cut" and "fail" to control the exploration of the search space.
  Then, the order of rules and literals becomes extremely important.

Aside: Prolog Programming: Cut

The "cut" predicate (written as "!") fixes the bindings up to that literal and does not search for other proofs (e.g., for alternative bindings for existential variables):

$F(C) \equiv \exists CN, Cap, CapP, A, Pop$: country$(CN, C, Cap, CapP, A, Pop) \wedge$
$\exists Org, Abbr, Est, T$: organization$(Org, Abbr, Cap, C, CapP, Est)$, isMember$(C, Org, T)$.

- `?-res(C)` returns every result country several times – for each organization that has its city in the capital.
- `?-res2(C)` returns every result country only once, since there is no backtracking in rule `hqInCap2` that would cause to search other proofs for e.g. `hqInCap2('B')`

```
:- include(mondial).
res(C) :- country(_,C,Cap,CapP,_,_), hqInCap(C,Cap,CapP).
hqInCap(C,Cap,CapP) :- organization(Org,_,Cap,C,CapP,_), isMember(C,Org,_).
res2(C) :- country(_,C,Cap,CapP,_,_), hqInCap2(C,Cap,CapP).
hqInCap2(C,Cap,CapP) :- organization(Org,_,Cap,C,CapP,_), isMember(C,Org,_),!.
```

[Filename: Datalog/prologcut.P]

- cut san serve –declaratively– as a SQL DISTINCT
- in combination with EXISTS (each existing thing would otherwise be checked),
- and for optimization of traversing proof trees.

Aside: Prolog Programming: Output

- the "cut" predicate fixes the bindings up to that literal and does not search for other proofs:
- write any term to stdout with `write(`*term*`)`,
- the `nl` predicate outputs a newline to stdout.
- tell me, when Paris is investigated ...

```
:- include(mondial).
res(C) :- country(_,C,Cap,CapP,_,_), hqInCap(C,Cap,CapP).
hqInCap(C,Cap,CapP) :- organization(Org,_,Cap,C,CapProv,_), check(Cap,Org),
              country(_,C,Cap,CapProv,_,_), isMember(C,Org,_).
check(X,Y) :-  X = 'Paris',  write('test '), write(X), write(' '),
                   write(Y), nl.
check(X,Y) :-  X \= 'Paris'.
```

[Filename: Datalog/prologparis.P]

- in `check(X,Y)`, Y must be bound upon calling it (XSB warns) – the rules are not safe,
- but "safe" is a bottom-up Datalog issue, in Prolog Programming, such unsafe procedural rules are common. (when called, the variables are already bound from atoms evaluated before)

Aside: Prolog Programming: Input and fail

- `read(X)` reads a term. The input must be finished by a ".".
- predicate `fail` is used when Prolog should "execute" the rule as "proof search" to do something, and then ... fail:
- below, `shouldI` fails if "N" is input. Then, also `shouldICheck(X)` fails, and the body for `res(C)` is not satisfied for this C. Try next C.

```
:- include(mondial).
res(C) :- country(_,C,_,_,_,_), shouldICheck(C), hqInCap(C).
hqInCap(C) :- country(_,C,Cap,CapProv,_,_), write('CAP found ... '),
              isMember(C,Org,_), write('check Org: '), write(Org), nl,
              organization(Org,_,Cap,C,CapProv,_).
shouldICheck(X) :- write('Should I check '), write(X),
                   write(' ("y."/"n.")?'), read(Z), shouldI(Z,X).
shouldI(Z,X) :- write('test if yes ... '), nl,  Z = 'y'.
shouldI(Z,X) :- write('here ... country is still '), write(X), nl, fail.
shouldI(Z,X) :- Z \= 'y', write('OK, I will skip '), write(X), nl, fail.
```

[Filename: Datalog/prologask.P]

Aside: Prolog Exercise

Consider again the program prologask.P from the previous slide.
When running it, the output “here ... country is still ...” when it is actually already finished with the respective country, demonstrates that useless work is done.
Where to place a cut to avoid this?

Aside: Prolog Documentation

- see XSB Manual Part I, Section 6 “Standard Predicates and Predicates of General Use”.

10.4 Positive Recursive Datalog

- a Datalog Program is called *recursive* if ...

Dependency Graph

Definition 10.6
For a positive Datalog program P over a (relational) signature $\mathbf{R} = \{R_1, \ldots, R_n\}$, its Dependency Graph *$G = (V, E)$ is defined as follows:*

- *$V = \{R_1, \ldots, R_n\}$ is the set of vertexes,*
- *$R_i \to R_j \in E$ if P contains a rule with head predicate R_j and R_i occurs in its body (“R_j depends on R_i”).* □

Definition 10.7
A Datalog program is called recursive *if its dependency graph contains a cycle.* □

Consequences

... the definitions up to now hold for nonrecursive programs and for recursive ones:

- the minimal model is defined as usual,
- T_P and T_P^ω are defined as usual,
- the resolution proofs exist.
  - Systems based on PROLOG's SLD resolution potentially run into infinite proof search trees
    (can be blocked by (expensive) bookkeeping)
  - XSB supports "tabling" which makes it more efficient and prevents it from infinite loops (tabling stores derived facts for reuse),
  - must be activated (see below).

Example: Transitive Closure

- tc(x,y) $\leftarrow$ p(x,y).
  tc(x,y) $\leftarrow \exists$ z: tc(x,z) $\wedge$ tc(z,y).
- XSB: % as comment sign,
- `:- auto_table.` for activating automatic tabling,
- manual tabling can be switched on with
  `:- table` R_1/k_1`, ...,` R_n/k_n`.`
  for k_i-ary table R_i.

```
% :- auto_table.
:- table borders/3, reachable/2.
:- include(mondial).
borders(Y,X,Z) :- borders(X,Y,Z).    % make it symmetric.
reachable(X,Y) :- borders(X,Y,_).
reachable(X,Y) :- reachable(X,Z), borders(Z,Y,_).
```

[Filename: Datalog/transitiveclosure.P]

Exercise

Complete the program from Slide 407 such that it also includes rivers flowing through lakes into others.

Additional Syntax, Built-Ins

- arithmetic operations: + - * /
- assignment by *var* `is` *term* in the body
- comparisons: as usual, \= for $\neq$, =< and >= for $\leq$ and $\geq$.
- see also XSB Manual Part I Sections 3.10.5 (Inline Predicates) and 4.3 (Operators).

```
:- include(mondial).
cview(N,C,Pop,A,Density) :- country(N,C,_,_,A,Pop), Density is Pop/A.
```

[Filename: Datalog/arithmetics.P]

10.5 Datalog with Negation

Consider conjunctive queries that include *negative* Literals.

- e.g. $F(C) = \exists CN, Cap, CapP, A, Pop :$.
  $(\text{country}(CN, C, Cap, CapP, A, Pop) \wedge \neg \text{ismember}(C, \text{“EU”}, \text{“member”}))$
- the database contains only positive facts, so no negative information can be logically implied!
- SQL:

  ```
  SELECT code FROM country
  WHERE NOT (code,'EU','member') IN (SELECT * FROM ismember);
  ```

  yields 214 results.
- Databases: *“Closed World Semantics”* – tuples that are not stored are assumed not to hold.

$\Rightarrow$ database query semantics deviates from standard FOL model theory.

$\Rightarrow$ a different model theory applies!

Closed World/Default Negation

- actually known + used in SQL without problems,
- the idea of the Minimal Model is analogous:
  everything that cannot be proven is false in the Minimal Model.
  - But the *Minimal Model is not well-defined in presence of negation:*
    Consider $P = \{p \leftarrow \neg q\}$:
    Both $\mathcal{M}_1 = \{p\}$ and $\mathcal{M}_2 = \{q\}$ are minimal models of P.
- Prolog: SLD-resolution extended to SLD-NF-resolution:
  - NF: Negation (of $p(c_1, \ldots, c_n)$) as "(finite) failure" to prove $p(c_1, \ldots, c_n)$ (only for ground atoms; cf. safety):
  - Open a resolution proof for $\neg p(c_1, \ldots, c_n)$ as usual and show that after finitely many steps there is no more progress towards the empty clause.
  - Example: For $P = \{p \leftarrow \neg q\}$, SLD-NF for `?- p` starts a proof for the body, i.e., for $\neg q$ which fails (the rule is equivalent to the clause $\{p, q\}$) immediately.
    Thus $\neg q$ is "proven" and p is confirmed – the answer to `?- p` is "yes".
- Preview: both $\{p \leftarrow \neg q\}$ and $\{q \leftarrow \neg p\}$ are logically equivalent to $p \vee q$, but, as programs, have different semantics!

NEGATION IN THE BODY: DATALOG$^\neg$

The language Datalog$^\neg$ extends positive Datalog as follows:

- the rule body is allowed to contain also negative literals:
  Rules are now of the form
  $$H \leftarrow L_1 \wedge \ldots \wedge L_k$$
  where each L_i is a positive ($p(a_1, \ldots, a_n)$) or negative ($\neg p(a_1, \ldots, a_m)$) literal.
- Safety requirement: every variable that occurs in a negative literal must also occur in a positive one, e.g.
  $\text{unreachable}(X, Y) \leftarrow \text{country}(X) \wedge \text{country}(Y) \wedge \neg\text{reachable}(X, Y)$.

Formal Semantics

The T_P operator (cf. Slide 568) is extended as follows:

For a set I of ground atoms,

$$T_P(I) := \{\ \sigma(H) \text{: } \sigma \text{ is a ground substitution and there is a rule } H \leftarrow L_1 \wedge \ldots \wedge L_k \text{ in } P \text{ such that for each } i = 1..k \quad \sigma(p_i(\bar{a})) \in I \text{ if } L_i = p_i(\bar{a}) \text{ is positive}, \quad \sigma(p_i(\bar{a})) \notin I \text{ if } L_i = \neg p_i(\bar{a}) \text{ is negative } \}$$

- The plain T_P^ω computation is not suitable: In the first "round" things are false that will become true later

$\Rightarrow$ "wait" before evaluating a negative literal $\neg p(c_1, \ldots, c_n)$ until the predicate p is completely computed.

(note: SLD resolution does automatically "stratify" when it opens the subproof for $\neg p(c_1, \ldots, c_n)$ and tries to complete it with the rules for p.)

STRATIFICATION

Dependency Graph with Negation

Extend Definition 10.6:

Definition 10.8
For a Datalog$^\neg$ program P over a (relational) signature $\mathbf{R} = \{R_1, \ldots, R_n\}$, its Dependency Graph *$G = (V, E)$ is defined as follows:*

- *$V = \{R_1, \ldots, R_n\}$ is the set of vertexes,*
- *$R_i \to R_j \in E$ if P contains a rule with head predicate R_j and R_i occurs positively in its body ("R_j depends positively on R_i").*
- *$R_i \stackrel{\neg}{\to} R_j \in E$ if P contains a rule with head predicate R_j and R_i occurs negatively in its body ("R_j depends negatively on R_i").* □

If the dependency graph does not contain a negative cycle (i.e., a cycle where *at least* one edge is negative) then there exists a simple, intuitive semantics
(note that positive cycles are allowed).

Stratification

Definition 10.9

Given a Datalog$^\neg$ program P without negative cycles over a signature Σ, a stratification *is a partitioning of Σ into* strata *$S_1, \ldots, S_n$ by a* stratification mapping *$\sigma : \Sigma \to \{1, \ldots, n\}$ such that*

- *if p depends positively on q, then $\sigma(p) \geq \sigma(q)$,*
- *if p depends negatively on q, then $\sigma(p) > \sigma(q)$,*
- *if such a stratification is possible, P is called* stratifiable*.*

Define P_i to be the set of rules in P whose head predicate is in S_i. □

Properties

- S_1: predicate symbols (incl. facts) that do not depend negatively on any other predicate,
- S_i: predicate symbols that depend positively only on predicate symbols in $S_0, \ldots, S_i$,
- S_i: predicate symbols that depend negatively only on predicate symbols in $S_0, \ldots, S_{i-1}$.
- predicates that are positively cyclic dependent on each other belong to the same stratum.
- $\{P_1, \ldots, P_n\}$ is a partitioning of P.

Note: there may be several stratifications of a program (any partitioning that is compatible with the priority order given by the negative dependencies).

Stratification

Proposition 10.7

- *every nonrecursive Datalog$^\neg$ program is stratifiable,*
- *many recursive Datalog$^\neg$ programs are also stratifiable.*
  (cf. reachable, non-reachable) □

STRATIFIED MODEL

Stratification allows to compute a model incrementally (bottom-up): Compute each stratum by "freezing" the IDB predicates defined in the previous stratum like EDB relations/facts:

Definition 10.10

Let $P = \{P_1, P_2, \ldots, P_n\}$ be a stratified program. Then, $\mathcal{S}(P)$ defined as follows is the stratified model of P:

$$\begin{aligned} \mathcal{I}_0 &= \emptyset \\ \mathcal{I}_k &= T^{\omega}_{P_k \cup \mathcal{I}_{k-1}}(\emptyset) \quad \text{for } 1 \leq k \leq n \\ \mathcal{S}(P) &= \mathcal{I}_n \end{aligned}$$

(with every P_i a set of rules and every $\mathcal{I}_i$ a set of ground atoms, $P_k \cup \mathcal{I}_{k-1}$ is a Datalog program that fits into stratum $\mathcal{S}_k$). □

Proposition 10.8

- *$\mathcal{S}(P)$ does not depend on the chosen stratification,*
- *$\mathcal{S}(P)$ is a model of P,*
- *$\mathcal{S}(P)$ is minimal (i.e., no $\mathcal{M}' \subsetneq \mathcal{S}(P)$ is a model of P),*
- *for programs containing negation, there are in general several models that are minimal.* □

Comments

- bottom-up stratified evaluation is the counterpart to top-down SLD-NF evaluation,
- tabling fits well with stratification,
- XSB does stratification automatically if a program contains negation.

Exercise

Prove that Definition 10.10 is equivalent to the following characterization:

$$\begin{aligned} \mathcal{J}_0 &= \emptyset \\ \mathcal{J}_k &= T^{\omega}_{P_1 \cup \ldots \cup P_k}(\mathcal{J}_{k-1}) \quad \text{for } 1 \leq k \leq n \\ \mathcal{S}'(P) &= \mathcal{J}_n \end{aligned}$$

Monotonic vs. Nonmonotonic Reasoning

Definition 10.11
For a given set of input formulas ϕ, and a reasoning mechanism M, let $Th_M(\phi)$ denote the "theory of ϕ wrt. M", i.e., the set of conclusions ψ such that $\phi \models_M \psi$.

A reasoning mechanism M is monotonic *if*

$$\phi_1 \subseteq \phi_2 \Rightarrow Th_M(\phi_1) \subseteq Th_M(\phi_2)$$

□

- FOL is monotonic,
- The Minimal Model semantics is monotonic,
- Default Logic and Human Reasoning is nonmonotonic (allowing conclusions in presence of incomplete knowledge that can be revised upon additional information),
- Stratified semantics is nonmonotonic.

Exercise

- Give an example for the nonmonotonicity of the stratified semantics,
- show that for a stratifiable program P there can be multiple minimal models.

10.5.1 (Stratified) Nonrecursive Datalog with Negation vs. Relational Algebra and SQL

Theorem 10.3
Nonrecursive Datalog with (stratified) negation with a single-predicate result is equivalent to the relational calculus and to the relational algebra. □

- Means: every nonrecursive Datalog$^\neg$ program that defines a single n-ary result predicate res/n can be expressed by a calculus query with n free variables, and equivalently by a relational algebra expression with an N-ary result relation, and
- every n-ary relational algebra expression can be expressed by a nonrecursive Datalog$^\neg$ program that defines a single n-ary result predicate res/n.

Exercise:

- prove the "Algebra $\rightarrow$ Datalog" direction (by structural induction).
- Given a (safe) rule $H \leftarrow C_1 \wedge \ldots \wedge C_n \wedge D_{n+1} \wedge \ldots \wedge D_{n+k}$
  where the C_i are positive literals and the D_i are negative literals, give a relational algebra expression that returns the relation defined by it.

Example: Relational Division

- recall: the relational division is defined in the relational algebra by two negations

Organizations that have at least one member on each continent:

```
% :- auto_table.          % here not necessary
:- include(mondial).
orgOnCont(O,Cont) :- isMember(_C,O,_), encompasses(_C, Cont,_).
notResult(O) :- organization(O,_,_,_,_,_), continent(_Cont,_),
                not orgOnCont(O,_Cont).
result(O) :- organization(O,_,_,_,_,_), not notResult(O).
% ?-result(O).
% ?- findall(_O, result(_O), L).
```

[Filename: Datalog/orgOnContsDiv.P]

- note: call of the PROLOG standard predicate

  ```
  ?- findall(_O, result(_O), L).
  ```

  returns all answers as a PROLOG list.

- compare with expressing this query in SQL.

PROLOG FINDALL

Syntax:

findall(*variable*, *predicate*(*many variables*), *listvariable*)

- the *variable* must be bound in the predicate query; all other variables in the predicate query are local to it,
- *listvariable* does not occur in the predicate query.

```
?- findall(A,continent(N,A),L).

A = _h44
N = _h66
L = [9562488,45095292,8503474,30254708,39872000]

?- findall(N,city(N,'D',P,Pop,La,Lo,El),L).
% all names of german cities.
```

Large lists sometimes lead to a crash:

```
?- findall(Pop,city(N,C,P,Pop,La,Lo,El),L).
```

AGGREGATION

- example see next slide.
- PROLOG dialects supports aggregation
- XSB: via PROLOG collections:
  - collect values in a bag:
    bagof($var_1, var_2 \hat{} \ldots \hat{} var_n \hat{} pred(var_1, \ldots, var_n), collvar$)
    for $collvar :=$ bagof$\{var_1 \mid \exists var_2, \ldots, var_n : pred(var_1, \ldots, var_n)\}$
  - explicitly program the aggregation operator recursively over the collection.
  - collection is a PROLOG list organized as head, tail:
    Syntax: `[H|T]` or `.(H,T)`, empty list is `[]`.
- Note: aggregation operations also require stratification – the predicates used in the subquery must be computed before.


```
:- include(mondial).
citypops(C,B) :- bagof(Pop,N^P^Lo^La^El^city(N,C,P,Pop,La,Lo,El),B).

% citypops('A',L).
% L = [1583000,10102,87321,null,144000,203000,118000,238000,51102]
% citypops('A',.(H,T)).
% H = 1583000
% T = [10102,87321,null,144000,203000,118000,238000,51102]

sum(X,[H|T]) :- sum(Y,T), H \= null, Y \= null, X is H + Y.
sum(H,[H|T]) :- sum(null,T), H \= null.
sum(X,[null|T]) :- sum(X,T).
sum(null,[]).

% Test:  ?- sum(N,[1,2,3,4,5]).      yields 15

citypopsum(C,X) :- citypops(C,B), sum(X,B).

% citypopsum('A',X).
% X = 2434525
```

[Filename: Datalog/aggregation.P]

Demonstrate both collection syntaxes:

```
:- include(mondial).
citypops(C,B) :- bagof(Pop,N^P^Lo^La^El^city(N,C,P,Pop,La,Lo,El),B).

sum1(X,[H|T]) :- sum1(Y,T), H \= null, Y \= null, X is H + Y.
sum1(H,[H|T]) :- sum1(null,T), H \= null.
sum1(X,[null|T]) :- sum1(X,T).
sum1(null,[]).

sum2(X,.(H,T)) :- sum2(Y,T), H \= null, Y \= null, X is H + Y.
sum2(H,.(H,T)) :- sum2(null,T), H \= null.
sum2(X,.(null,T)) :- sum2(X,T).
sum2(null,[]).

citypopsum(C,X,Y) :- citypops(C,B), sum1(X,B), sum2(Y,B).
```

[Filename: Datalog/aggregation2.P]

Aside: Tabling with Answer Subsumption

- XSB Documentation, Section 5.4
- tabling with subsumption: “subsumed” (wrt. some ordering) answers are not stored

$\Rightarrow$ only “maximal” ones remain.

```
:- include(mondial).
:- table citypopmax(_,po(> /2)).  %% blank before "/" is important!
citypopmax(C,N) :- city(_,C,_,N,_,_,_), N \= null.
?- citypopmax('D',P).
```

[Filename: Datalog/aggrsubsumpt.P]

- works only for min/max, not count/sum (these are not idempotent)
- see documentation: shortest paths

10.5.2 (Stratified) Recursive Datalog with Negation

- The stratified semantics seamlessly covers stratifiable *recursive* Datalog$^\neg$ programs.
- expressiveness covers Algebra/Calculus + Recursion.

```
% :- auto_table.
:- table borders/3, reachable/2.
:- include(mondial).
borders(Y,X,Z) :- borders(X,Y,Z).    % make it symmetric.
reachable(X,Y) :- borders(X,Y,_).
reachable(X,Y) :- reachable(X,Z), borders(Z,Y,_).
notReachable(X,Y) :- country(_,X,_,_,_,_), country(_,Y,_,_,_,_),
                     not reachable(X,Y).
```

[Filename: Datalog/transitiveclosure2.P]

Exercise

- Give the intermediate steps of the T_P^ω-based stratified evaluation for the above program.

Summary

- bottom-up inefficient when regarding a single query.
- IDB predicates can be seen as views:
  - materialization of views not unusual in DB
    (when frequently used, seldomly changing)
  - view maintenance strategies (upon updates of underlying tables) in LP exist:
  - seminaive evaluation of T_P^n: consider only rule instantiations where at least one atom has been derived in the previous round for computing the next one.
- tabling is already a mixture between bottom-up and top-down.
- data transformation/integration applications:
  - transform the whole input database(s),
  - export certain IDB relations as “resulting database”,
  - (e.g. generation of the MONDIAL database from Web sources with F-Logic in 1998).

Summary: Expressiveness of Datalog$^\neg$

- negation in the body restricted to stratifiable knowledge bases
- no existentials
  note: it is e.g. not possible to express that every country has a capital if not all of them are explicitly known.
  Datalog is a database language, not an ontology language.
  $\Rightarrow$ Semantic Web uses different languages.
- no disjunction in the head $P \rightarrow (Q \vee R)$
- unique name assumption, no equality