Foundations of Databases

Datalog with Negation

(Slides by Thomas Eiter)

The Issue

- In While$^{(+)}$ and CALC$^{(+)}$-μ, we have negation $(\neg)$ as operator
- Thus, queries like complement of a relation, complement of transitive closure can be easily expressed in these languages
- These queries can not be expressed in datalog (monotonicity)
- Desired: Extension of datalog with negation

  Example: *ready(D)* $\leftarrow$ *device(D),* $\neg$ *busy(D)*
- Giving a semantics is not straightforward because of possible cyclic definitions

  Example:

  single(X) $\leftarrow$ *man(X),* $\neg$ *husband(X)*
  husband(X) $\leftarrow$ *man(X),* $\neg$ *single(X)*

Datalog$^\neg$ Syntax

Defn. A *datalog$^\neg$ program* P is a finite set of datalog$^\neg$ rules r of the form

$$A \leftarrow B_1, \ldots, B_n \quad (1)$$

where $n \geq 0$ and

- A is an atom $R_0(\vec{x}_0)$
- Each B_i is an atom $R_i(\vec{x}_i)$ or *a negated atom* $\neg R_i(\vec{x}_i)$
- $\vec{x}_0, \ldots, \vec{x}_n$ are vectors of variables and constants (from $\mathbf{dom}$)
- Every variable in $\vec{x}_0, \ldots, \vec{x}_n$ must occur in some atom $B_i = R_i(\vec{x}_i)$ ("safety")
- the head of r is A, denoted $H(r)$.
- the body of r is $\{B_1, \ldots, B_n\}$, denoted $B(r)$, and
  $B^+(r) = \{R(\vec{x}) \mid \exists i\, B_i = R(\vec{x})\}$, $B^-(r) = \{R(\vec{x}) \mid \exists i\, B_i = \neg R(\vec{x})\}$,

P has extensional and intensional relations, $edb(P)$ resp. $idb(P)$, like a datalog program.

Remarks: – "$\neg$" is as in LP often denoted by "not" (e.g., in DLV)
– Equality (=) and inequality ($\neq$, as $\neg =$) are usually available as built-ins, but usage must be "safe"

Datalog$^\neg$ Semantics – The Problem

- **Idea:** Naturally extend the minimal-model semantics of datalog (equivalently, the least fixpoint-semantics) to negation
- Generalize to this aim the immediate consequence operator

$$\mathbf{T}_P(\mathbf{K}) : inst(sch(P)) \rightarrow inst(sch(P))$$

Defn. Given a datalog$^\neg$ program P and $\mathbf{K} \in inst(sch(P))$, a fact $R(\vec{t})$ is an *immediate* consequence for $\mathbf{K}$ and P, if either

- $R \in edb(P)$ and $R(\vec{t}) \in \mathbf{K}$, or
- there exists some ground instance r of a rule in P such that
  - $H(r) = R(\vec{t})$,
  - $B^+(r) \subseteq \mathbf{K}$, and
  - $B^-(r) \cap \mathbf{K} = \emptyset$.

(That is, evaluate "$\neg$" w.r.t. $\mathbf{K}$)

Problems with Least Fixpoints

- Natural trial: Define the semantics of datalog$^\neg$ in terms of least fixpoint of $\mathbf{T}_P$.
- However, this suffers from several problems:

1. $\mathbf{T}_P$ may not have a fixpoint:

$$P_1 = \{\, known(a) \leftarrow \neg known(a) \,\}$$

2. $\mathbf{T}_P$ may not have a least (i.e., single minimal) fixpoint:

$$P_2 = \{ \quad \begin{array}{r} single(X) \leftarrow man(X), \neg husband(X) \\ husband(X) \leftarrow man(X), \neg single(X) \end{array} \quad \}$$

$\mathbf{I} = \{man(dilbert)\}$

3. The least fixpoint of $\mathbf{T}_P$ including $\mathbf{I}$ may not be constructible by fixpoint iteration (i.e., not as limit $\mathbf{T}_P^\omega(\mathbf{I})$ of $\{\mathbf{T}_P^i(\mathbf{I})\}_{i \geq 0}$):

$$P_3 = P_2 \cup \{husband(X) \leftarrow \neg husband(X), single(X)\}$$

$\mathbf{I} = \{man(dilbert)\}$) as above

Note: Operator $\mathbf{T}_P$ is not monotonic!

Problems with Minimal Models

There are similar problems for model-theoretic semantics

- We can associate with P naturally a first-order theory Σ_P as in the negation-free case (write rules as implications):

$$R(\vec{x}) \leftarrow (\neg)R_1(\vec{x}_1), \ldots, (\neg)R_n(\vec{x}_n)$$

$$\rightsquigarrow$$

$$\forall\vec{x}\forall\vec{x}_1 \cdots \forall\vec{x}_n(((\neg)R_1(\vec{x}_1) \wedge \cdots \wedge (\neg)R_n(\vec{x}_n)) \supset R(\vec{x}))$$

- Still, $\mathbf{K} \in inst(sch(P))$ is a model of Σ_P iff $\mathbf{T}_P(\mathbf{K}) \subseteq \mathbf{K}$ (and models are not necessarily fixpoints)
- However, multiple minimal models of Σ_P containing $\mathbf{I}$ might exist (dilbert example).

Solution Approaches

Different kinds of proposals have been made to handle the problems above

- **Give up single fixpoint / model semantics:** Consider alternative fixpoints (models), and define results by *intersection*, called *certain semantics*.

  Most well-known: Stable model semantics (Gelfond & Lifschitz, 1988;1991).
  Still suffers from 1.

- **Constrain the syntax of programs:** Consider only fragment where negation can be "naturally" evaluated to a single minimal model.

  Most well-known: semantics for stratified programs (Apt, Blair & Walker, 1988), perfect model semantics (Przymusinski, 1987).

- **Give up 2-valued semantics:** Facts might be true, false or *unknown*

  Adapt and refine the notion of immediate consequence.

  Most well-known: Well-founded semantics (Ross, van Gelder & Schlipf, 1991).
  Resolves all problems 1-3

- **Give up fixpoint / minimality condition:** Operational definition of result.

  Most well-known: Inflationary semantics (Abiteboul & Vianu, 1988)

Semi-Positive Datalog

"Easy" case: Datalog¬ programs where negation is applied only to edb relations.

- Such programs are called *semi-positive*
- For a semi-positive program, the operator $\mathbf{T}_P$ is monotonic if the edb-part is fixed, i.e., $\mathbf{I}|edb(P) = \mathbf{J}|edb(P)$ implies $\mathbf{T}_P(\mathbf{I}) \subseteq \mathbf{T}_P(\mathbf{J})$

Theorem. Let P be a semi-positive datalog program and $\mathbf{I} \in inst(sch(P))$. Then,

1. $\mathbf{T}_P$ has a unique minimal fixpoint $\mathbf{J}$ such that $\mathbf{I}|edb(P) = \mathbf{J}|edb(P)$.
   $\mathbf{T}_P(\mathbf{I}) \subseteq \mathbf{T}_P(\mathbf{J})$
2. Σ_P has a unique minimal model $\mathbf{J}$ such that $\mathbf{I}|edb(P) = \mathbf{J}|edb(P)$.

Example

Semi-positive datalog can express the transitive closure of the complement of a graph G:

$$neg\_tc(x,y) \leftarrow \neg G(x,y)$$
$$neg\_tc(x,y) \leftarrow \neg G(x,z), neg\_tc(z,y)$$

Stratified Semantics

- **Intuition**: For evaluating the body of a rule instance r containing $\neg R(\vec{t})$, the value of the "negated" relation $R(\vec{t})$ should be known.
  1. Evaluate first R
  2. if $R(\vec{t})$ is false, then $\neg R(\vec{t})$ is true,
  3. if $R(\vec{t})$ is true, then $\neg R(\vec{t})$ is false and the rule is not applicable.
- **Example**:

$$boring(chess) \leftarrow \neg interesting(chess)$$
$$interesting(X) \leftarrow difficult(X)$$

  For $\mathbf{I} = \{\}$, compute result $\{boring(chess)\}$.
- **Note**: this introduces *procedurality* (violates declarativity)!

Dependency graph for Datalog$^{\neg}$ programs

Associate with each datalog$^{\neg}$ program P a directed graph $DEP(P) = (N, E)$, called *Dependency Graph*, as follows:

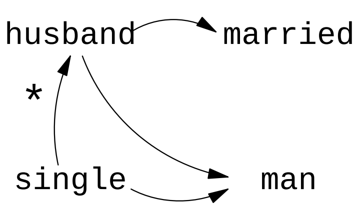
- $N = sch(P)$, i.e., the nodes are the relations.
- $E = \{\langle R, R' \rangle \mid \exists r \in P : H(r) = R \wedge R' \in B(r)\}$, i.e., arcs $R \rightarrow R'$ from the relations in rule heads to the relations in the body.
- Mark each arc $R \rightarrow R'$ with "*", if $R(\vec{x})$ is in the head of a rule in P whose body contains $\neg R'(\vec{y})$.

Remark: edb relations are often omitted in the dependency graph

Example

$$P: \quad husband(X) \leftarrow man(X),\ married(X).$$
$$single(X) \leftarrow man(X),\ \neg husband(X).$$

$dep(P)$:



Stratification Principle

If $R = R_0 \rightarrow R_1 \rightarrow R_2 \rightarrow \cdots R_{n-1} \rightarrow R_n = R'$ such that some $R_i \rightarrow R_{i+1}$ is marked with "*", then R' must be evaluated prior to R.

Stratification

Defn. A *stratification* of a datalog program P is a partitioning

$$\Sigma = \bigcup_{i \geq 1}^{n} P_i$$

of $sch(P)$ into nonempty, pairwise disjoint sets P_i such that

(a) if $R \in P_i$, $R' \in P_j$, and $R \rightarrow R'$ is in $DEP(P)$, then $i \geq j$;

(b) if $R \in P_i$, $R' \in P_j$, and $R \rightarrow R'$ is in $DEP(P)$ marked with "*," then $i > j$.

$P_1, \ldots, P_n$ are called the *strata* of P w.r.t. Σ.

Defn. A datalog program P is called *stratified*, if it has some stratification Σ.

Evaluation Order

A stratification Σ gives an *evaluation order* for the relations in P, given $\mathbf{I} \in inst(edb(P))$:

1. First evaluate the relations in P_1 (which is $\neg$-free).

   $\Rightarrow$ All relations R in heads of P_1 are defined. This yields $\mathbf{J}_1 \in inst(sch(P_1))$.

2. Evaluate P_2 considering relations in $edb(P)$ and P_1 as $edb(P_1)$, where $\neg R(\vec{t})$ is true if $R(\vec{t})$ is false in $\mathbf{I} \cup \mathbf{J}_1$;

   $\Rightarrow$ All relations R in heads of P_2 are defined. This yields $\mathbf{J}_2 \in inst(sch(P_2))$.

   . . .

3. Evaluate P_i considering relations in $edb(P)$ and $P_1, \ldots, P_{i-1}$ as $edb(P_i)$, where $\neg R(\vec{t})$ is true if $R(\vec{t})$ is false in $\mathbf{I} \cup \mathbf{J}_1 \cup \cdots \cup \mathbf{J}_{i-1}$;

4. The result of evaluating P on $\mathbf{I}$ w.r.t. Σ, denoted $P_\Sigma(\mathbf{I})$, is given by $\mathbf{I} \cup \mathbf{J}_1 \cup \cdots \cup \mathbf{J}_n$;

Example

$$P = \{ \quad husband(X) \leftarrow man(X),\ married(X)$$
$$single(X) \leftarrow man(X),\ \neg husband(X)\ \}$$

Stratification Σ:

$P_1 = \{man, married\}$, $P_2 = \{husband\}$, $P_3 = \{single\}$

$\mathbf{I} = \{man(dilbert)\}$:

1. Evaluate P_1: $\mathbf{J}_1 = \{\}$
2. Evaluate P_2: $\mathbf{J}_2 = \{\}$
3. Evaluate P_3: $\mathbf{J}_3 = \{single(dilbert)\}$
4. Hence, $P_\Sigma(\mathbf{I}) = \{man(dilbert)\}, single(dilbert)\}$

Formal Definition of Stratified Semantics

Let P be a stratified Datalog$^\neg$ program with stratification $\Sigma = \bigcup_{i=1}^{n} P_i$.

- Let P_i^* be the set of rules from P whose relations in the head are in P_i, and set $edb(P_1^*) = edb(P)$, $edb(P_i^*) = rels(\bigcup_{j=1}^{i-1} P_j^*) \cup edb(P)$, $i > 1$.
- For every $\mathbf{I} \in inst(edb(P))$, let $\mathbf{I}_0^\Sigma = \mathbf{I}$ and define

$$\begin{array}{lllllll}
\mathbf{I}_1^\Sigma & = & \mathbf{T}_{P_1^*}^\omega(\mathbf{I}_0^\Sigma) & = & lfp(\mathbf{T}_{P_1^*}(\mathbf{I}_0^\Sigma)) & \supseteq & \mathbf{I}_0^\Sigma \\
\mathbf{I}_2^\Sigma & = & \mathbf{T}_{P_2^*}^\omega(\mathbf{I}_1^\Sigma) & = & lfp(\mathbf{T}_{P_2^*}(\mathbf{I}_1^\Sigma)) & \supseteq & \mathbf{I}_1^\Sigma \\
 & \ldots & & & & & \\
\mathbf{I}_i^\Sigma & = & \mathbf{T}_{P_i^*}^\omega(\mathbf{I}_{i-1}^\Sigma) & = & lfp(\mathbf{T}_{P_i^*}(\mathbf{I}_{i-1}^\Sigma)) & \supseteq & \mathbf{I}_{i-1}^\Sigma \\
 & \ldots & & & & & \\
\mathbf{I}_n^\Sigma & = & \mathbf{T}_{P_n^*}^\omega(\mathbf{I}_{n-1}^\Sigma) & = & lfp(\mathbf{T}_{P_n^*}(\mathbf{I}_{n-1}^\Sigma)) & \supseteq & \mathbf{I}_{n-1}^\Sigma
\end{array}$$

  where $\mathbf{T}_Q^\omega(\mathbf{J}) = \lim\{\mathbf{T}_Q^i(\mathbf{J})\}_{i\geq 0}$ with $\mathbf{T}_Q^0(\mathbf{J}) = \mathbf{J}$ and $\mathbf{T}_Q^{i+1} = \mathbf{T}_Q(\mathbf{T}_Q^i(\mathbf{J}))$, and $lfp(\mathbf{T}_Q(\mathbf{J}))$ is the least fixpoint $\mathbf{K}$ of $\mathbf{T}_Q$ such that $\mathbf{K}|edb(Q) = \mathbf{J}|edb(Q)$.

- Denote $P_\Sigma(\mathbf{I}) = \mathbf{I}_n^\Sigma$

Proposition. For every $i \in \{1, \ldots, n\}$,

- $lfp(\mathbf{T}_{P_i^*}(\mathbf{I}_{i-1}^{\Sigma}))$ exists,
- $lfp(\mathbf{T}_{P_i^*}(\mathbf{I}_{i-1}^{\Sigma})) = \mathbf{T}_{P_i^*}^{\omega}(\mathbf{I}_{i-1}^{\Sigma})$ holds,
- $\mathbf{I}_{i-1}^{\Sigma} \subseteq \mathbf{I}_{i}^{\Sigma}$.

Therefore, $P_\Sigma(\mathbf{I})$ is always well-defined.

Stratified semantics singles out a model, and in fact a minimal model.

Theorem. $P_\Sigma(\mathbf{I})$ is a minimal model $\mathbf{K}$ of P such that $\mathbf{K}|edb(P) = \mathbf{I}$.

Dilbert Example cont'd

P = { $husband(X) \leftarrow man(X),\ married(X)$
$single(X) \leftarrow man(X),\ \neg husband(X)$ }

$edb(P) = \{man\}$

Stratification Σ: $P_1 = \{man, married\}$, $P_2 = \{husband\}$, $P_3 = \{single\}$

1. $P_1 = \{\}$
2. $P_2 = \{husband(X) \leftarrow man(X),\ married(X)\}$
3. $P_3 = \{single(X) \leftarrow man(X),\ \neg husband(X)\}$

$\mathbf{I} = \{man(dilbert)\}$:

1. $\mathbf{I}_1^{\Sigma} = \{man(dilbert)\}$
2. $\mathbf{I}_2^{\Sigma} = \{man(dilbert)\}$
3. $\mathbf{I}_3^{\Sigma} = \{man(dilbert), single(dilbert)\}$

Hence, $P_\Sigma(\mathbf{I}) = \{man(dilbert), single(dilbert)\}$

Stratification Theorem

- The stratification Σ above is not unique.
- Alternative stratification Σ':
  $P_1 = \{man, married, husband\}$, $P_2 = \{single\}$
- Evaluation with respect to Σ' yields same result!

The choice of a particular stratification is irrelevant:

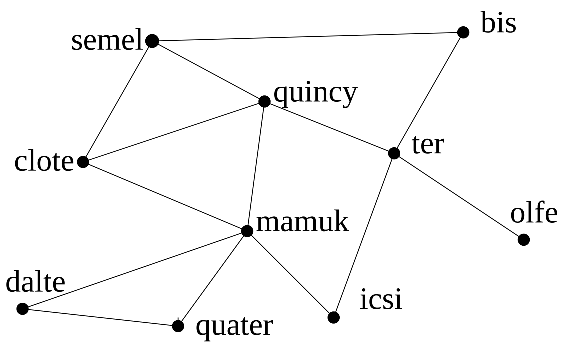
Stratification Theorem. Let P be a stratifiable datalog$^\neg$ program. Then, for any stratifications Σ and Σ' and $\mathbf{I} \in inst(sch(P))$, $P_\Sigma(\mathbf{I}) = P_{\Sigma'}(\mathbf{I})$.

- Thus, syntactic stratification yields semantically a canonical way of evaluation.
- The result $P_{str}(\mathbf{I})$ is called the *perfect model* or *stratified model* of P for $\mathbf{I}$.

Remark: Prolog features SLDNF – SLD resolution with (finite) negation as failure

Example: Railroad Network

Determine whether safe connections between locations in a railroad network



- **Cutpoint c for a and b:** if c fails, there is no connection between a and b
- **Safe connection between a and b:** no cutpoints between a and b exist
- E.g., ter is a cutpoint for olfe and semel, while quincy is not.

Relations:

$link(X, Y)$:	direct connection from station X to Y (edb facts)
$linked(A, B)$:	symmetric closure of $link$.
$connected(A, B)$:	there is path between A and B (one or more links)
$cutpoint(X, A, B)$:	each path from A to B goes through station X
$circumvent(X, A, B)$:	there is a path between A and B not passing X
$has\_icut\_point(A, B)$:	there is at least one cutpoint between A and B.
$safely\_connected(A, B)$:	A and B are connected with no cutpoint.
$station(X)$:	X is a railway station.

Railroad program P:

R_1 : $linked(A, B) :- link(A, B).$

R_2: $linked(A, B) :- link(B, A).$

R_3: $connected(A, B) :- linked(A, B).$

R_4: $connected(A, B) :- connected(A, C), linked(C, B).$

R_5: $cutpoint(X, A, B) :- connected(A, B), station(X),$
$\neg circumvent(X, A, B).$

R_6: $circumvent(X, A, B) :- linked(A, B), X \neq A, station(X), X \neq B.$

R_7: $circumvent(X, A, B) :- circumvent(X, A, C), circumvent(X, C, B).$

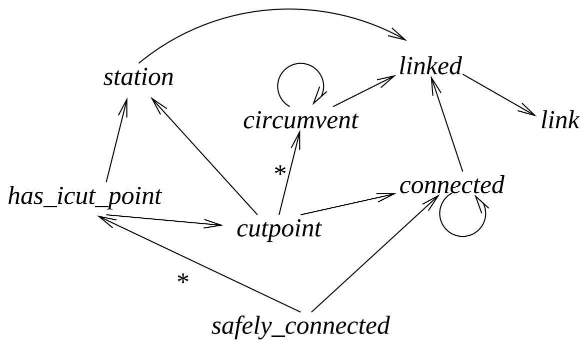
R_8: $has\_icut\_point(A, B) :- cutpoint(X, A, B), X \neq A, X \neq B.$

R_9: $safely\_connected(A, B) :- connected(A, B),$
$\neg has\_icut\_point(A, B).$

R_{10}: $station(X) :- linked(X, Y).$

Remark: Inequality ($\neq$) is used here as built-in. It can be easily defined in stratified manner.

$DEP(P)$:



Stratification Σ:

$P_1 = \{link, linked, station, circumvent, connected\}$

$P_2 = \{cutpoint, has\_icut\_point\}$

$P_3 = \{safely\_connected\}$

$\mathbf{I}(link)$ = { ⟨*semel, bis*⟩, ⟨*bis, ter*⟩, ⟨*ter,olfe*⟩, ⟨*ter, icsi*⟩, ⟨*ter, quincy*⟩,
⟨*quincy,semel*⟩, ⟨*quincy,clote*⟩, ⟨*quincy,mamuk*⟩,..., ⟨*dalte,quater*⟩ }

Evaluation $P_\Sigma(\mathbf{I})$:

1. $P_1 = \{link, linked, station, circumvent, connected\}$:

   $\mathbf{J}_1 = linked(semel, bis), linked(bis, ter), linked(ter, olfe), \ldots,$
   $connected(semel, olfe), \ldots, circumvent(quincy, semel, bis), \ldots$

2. $P_2 = \{cutpoint, has\_icut\_point\}$:

   $\mathbf{J}_2 = cutpoint(ter, semel, olfe), has\_icut\_point(semel, olfe) \ldots$

3. $P_3 = \{safely\_connected\}$:

   $\mathbf{J}_3 = safely\_connected(semel, bis), safely\_connected(semel, ter)$

   But, $safely\_connected(semel, olfe) \notin \mathbf{J}_3$

Algorithm STRATIFY

Input: A datalog$^\neg$ program P.
Output: A stratification Σ for P, or "no" if none exists.

1. Construct the directed graph $G := DEP(P)$ (=$\langle N, E\rangle$) with markers "*";
2. **For each** pair $R, R' \in N$ **do**
   if R reaches R' via some path containing a marked arc
   then begin $E := E \cup \{R \to R'\}$; mark $R \to R'$ with "*" **end**;
3. $i := 1$;
4. Identify the set K of all vertices p in G s.t. no marked $R \to R'$ is in E.
5. **If** $K = \emptyset$ and G has vertices left, **then** output "no"
   else begin output K as stratum P_i;
   Remove all vertices in K and corresponding arcs from G.
   end;
6. **If** G has vertices left **then begin** $i := i + 1$; **goto** step 4 **end**
   else stop.

Runs in polynomial time!

Inflationary Semantics for Datalog

Idea: A adopt a production-oriented view of datalog$^\neg$, similar as in rule-base expert systems

- A rule should be applied (fired) if the premises (=body literals) are satisfied with respect to the current state
- Rather than applying one rule at a time (as in expert systems), fire *all* applicable rules in parallel
- New facts may fire other rules
- Repeat application of rules, until no more new facts are generated.
- This amounts to the least fixpoint of the inflationary version of $\mathbf{T}_P(\mathbf{K})$.

For any datalog$^\neg$ program P, let $\mathbf{T}_P^+ : inst(sch(P)) \rightarrow inst(sch(P))$ denote the inflationary variant of $\mathbf{T}_P$:

$$\mathbf{T}_P^+(\mathbf{K}) = \mathbf{K} \cup \mathbf{T}_P(\mathbf{K})$$

Defn. Given a datalog$^\neg$ program P and $\mathbf{I} \in inst(edb(P))$, the inflationary semantics of P w.r.t. $\mathbf{I}$, denoted $P_{inf}(\mathbf{I})$, is the limit of the sequence $\{\mathbf{T}_P^{+i}(\mathbf{I})\}_{i \geq 0}$, where $\mathbf{T}_P^{+0}(\mathbf{I}) = \mathbf{I}$ and $\mathbf{T}_P^{+i+1}(\mathbf{I}) = \mathbf{T}_P^+(\mathbf{T}_P^{+i}(\mathbf{I}))$.

Notice:

- $P_{inf}(\mathbf{I})$ is well-defined for each program P and input database $\mathbf{I}$.
- $P_{inf}(\mathbf{I})$ is a model of P containing $\mathbf{I}$, but not necessarily a minimal model.
- $P_{inf}(\mathbf{I})$ is the not necessarily a minimal fixpoint of $\mathbf{T}_P^+$ containing $\mathbf{I}$.

Example

$$P = \{q(b) \leftarrow \neg p(a), \quad r(c) \leftarrow \neg q(b) \quad p(a) \leftarrow r(c), \neg p(b)\}$$

Consider $\mathbf{T}_P^{+i}(\mathbf{I})$, $i \geq 0$, for $\mathbf{I} = \emptyset$:

- $\mathbf{T}_P^{+0}(\mathbf{I}) = \mathbf{I} = \{\}$.
- The first two rules are applicable, as $\neg p(a)$, $\neg q(b)$ are satisfied wrt. $\mathbf{I}_0$.
- $\mathbf{T}_P^{+1}(\mathbf{I}) = \{q(b), r(c)\}$.
- The third rule is now applicable, as $r(c)$, $\neg p(b)$ are satisfied wrt. $\mathbf{I}_1$.
- $\mathbf{T}_P^{+2}(\mathbf{I}) = \{q(b), r(c), p(a)\}$.
- No new facts can be obtained, as all rules have been applied.
- Hence, $P_{inf}(\mathbf{I}) = \mathbf{T}_P^{+2}(\mathbf{I})$.

Note that $P_{inf}(\mathbf{I})$ is not a minimal model of P containing I.

Example: One-Step-Behind Technique

Undirected graph $G = \langle V, E \rangle$, distance $d : V^2 \longrightarrow \{0, 1, 2, \ldots\} \cup \infty$
($d(x, y)$ = length of shortest path between x, y; ∞ if no path exists)

Define $shorter(x, y, x', y') \leftrightarrow_{df} dist(x, y) < dist(x', y') < \infty$

Program P ($edb(P) = \{v, e\}$, where e is symmetric):

$$\begin{aligned} t(x,x) &\leftarrow v(x) \\ t(x,y) &\leftarrow t(x,z), e(z,y) \\ t1(x,y) &\leftarrow t(x,y) \\ shorter(x_1,y_1,x_2,y_2) &\leftarrow t1(x_1,y_1), t(x_2,y_2), \neg t1(x_2,y_2) \end{aligned}$$

$t1(x, y)$ is “one step behind” $t(x, y)$

$$\begin{aligned} i \geq 0: \quad t(x,y) \in \mathbf{T}_P^{+i}(\mathbf{I}) &\Leftrightarrow dist(x,y) \leq i-1, \\ t1(x,y) \in \mathbf{T}_P^{+i}(\mathbf{I}) &\Leftrightarrow dist(x,y) \leq i-2 \end{aligned}$$

Inflationary vs Stratified Semantics

- Inflationary Semantics is well-defined for *all* datalog$^\neg$ programs, not only for stratified programs. It was used e.g. in the FLORID system.
- For semi-positive programs, inflationary and stratified semantics coincide.
- Datalog$^\neg$ queries under stratified semantics are subsumed by inflationary semantics:

  Theorem. For every stratified datalog$^\neg$ program P with “output” relation R, there exists a datalog$^\neg$ program P' such that $edb(P') = edb(P)$ and for all $\mathbf{I} \in inst(edb(P))$, $P'_{inf}(\mathbf{I})(R) = P_{strat}(\mathbf{I})(R)$.

- The converse fails, i.e., there are datalog$^\neg$ queries P under inflationary semantics non-equivalent to any datalog$^\neg$ query under stratified semantics (Kolaitis, 1991).

  Intuitive reason: Stratified semantics has a static, fixed number of negation layers, while inflationary semantics allows dynamically many.

Stable Models Semantics

- **Idea**: Try to construct a (minimal) fixpoint by iteration from input

  If the construction succeeds, the result is the semantics.

- **Problem**: Application of rules might be compromised.

  Example:

$$P = \{p(a) \leftarrow \neg p(a), \quad q(b) \leftarrow p(a), \quad p(a) \leftarrow q(b)\}$$

  ($edb(P)$ is void, thus $\mathbf{I}$ is immaterial and omitted)

  – $\mathbf{T}_P$ has the least fixpoint $\{p(a), q(b)\}$
  – It is iteratively constructed $\mathbf{T}_P^\omega = \{p(a), q(b)\}$
  – $p(a)$ is included into $\mathbf{T}_P^1$ by the first rule, since $p(a) \notin \mathbf{T}_P^0 = \emptyset$.
  – This compromises the rule application, and $p(a)$ is not “foundedly” derived!
  – Note: $\mathbf{T}_P^+ = \{p(a), q(b)\}$

Fixed Evaluation of Negation

- **Reason:** $\mathbf{T}_P$ is not monotonic.

- **Solution:** Keep negation throughout fixpoint-iteration fixed.

  Evaluation negation w.r.t. a fixed candidate fixpoint model $\mathbf{J}$.

- Introduce for datalog$^\neg$ program and $\mathbf{J} \in inst(sch(P))$ a new immediate consequence operator $\mathbf{T}_{P,\mathbf{J}}$:

Immediate Consequences under Fixed Negation

Defn. Given a datalog$^{\neg}$ program P and $\mathbf{J}, \mathbf{K} \in inst(sch(P))$, a fact $R(\vec{t})$ is an *immediate* consequence for $\mathbf{K}$ and P under negation $\mathbf{J}$, if either

- $R \in edb(P)$ and $R(\vec{t}) \in \mathbf{K}$, or
- there exists some ground instance r of a rule in P such that
  - $H(r) = R(\vec{t})$,
  - $B^{+}(r) \subseteq \mathbf{K}$, and
  - $B^{-}(r) \cap \mathbf{J} = \emptyset$.

  (That is, evaluate "$\neg$" under $\mathbf{J}$ instead of $\mathbf{K}$)

Defn. For any datalog$^{\neg}$ program P and $\mathbf{J}, \mathbf{K} \in inst(sch(P))$, let

$\mathbf{T}_{P,\mathbf{J}}(\mathbf{K})$ = $\{A \mid A$ is an immediate consequence for $\mathbf{K}$ and P under negation $\mathbf{J}\}$

Notice:

- $\mathbf{T}_P(\mathbf{K})$ coincides with $\mathbf{T}_{P,\mathbf{K}}(\mathbf{K})$
- $\mathbf{T}_{P,\mathbf{J}}$ is a monotonic operator, hence has for each $\mathbf{K} \in inst(sch(P))$ a least fixpoint containing $\mathbf{K}$, denoted $lfp(\mathbf{T}_{P,\mathbf{J}}(\mathbf{K}))$
- $lfp(\mathbf{T}_{P,\mathbf{J}}(\mathbf{I}))$ coincides with $\mathbf{I}$ on $edb(P)$ and is the limit $\mathbf{T}^{\omega}_{P,\mathbf{J}}$ of the sequence

$$\{\mathbf{T}^{i}_{P,\mathbf{J}}(\mathbf{I})\}_{i \geq 0}$$

  where $\mathbf{T}^{0}_{P,\mathbf{J}}(\mathbf{I}) = \mathbf{I}$ and $\mathbf{T}^{i+1}_{P,\mathbf{J}}(\mathbf{I}) = \mathbf{T}_{P,\mathbf{J}}(\mathbf{T}^{i}_{P,\mathbf{J}}(\mathbf{I}))$.

Stable Models

Using $\mathbf{T}_{P,\mathbf{J}}$, stable models are defined by requiring that $\mathbf{J}$ is reproduced by the program:

Defn. Let P be a datalog$^\neg$ program P and $\mathbf{I} \in inst(edb(P))$. Then, a stable model for P and $\mathbf{I}$ is any $\mathbf{J} \in inst(sch(P))$ such that

1. $\mathbf{J}|edb(P) = \mathbf{I}$, and
2. $\mathbf{J} = lfp(\mathbf{T}_{P,\mathbf{J}}(\mathbf{I}))$.

Notice: Monotonicity of $\mathbf{T}_{P,\mathbf{J}}$ ensures that at no point in the construction of $lfp(\mathbf{T}_{P,\mathbf{J}})(\mathbf{I})$ using fixpoint iteration from $\mathbf{I}$, the application of a rule can be compromised later.

Example

$$P = \{\ p(a) \leftarrow \neg p(a), \qquad q(b) \leftarrow p(a), \qquad p(a) \leftarrow q(b)\ \}$$

($edb(P)$ is void, thus $\mathbf{I}$ is immaterial and omitted)

- Take $\mathbf{J} = \{p(a), q(b)\}$. Then
  - $\mathbf{T}^0_{P,\mathbf{J}} = \emptyset$
  - $\mathbf{T}^1_{P,\mathbf{J}} = \emptyset$
- Thus $lfp(\mathbf{T}_{P,\mathbf{J}}) = \emptyset \neq \mathbf{J}$.
- Hence, the fixpoint $\mathbf{J}$ of $\mathbf{T}_P$ is refuted.
- For P, no stable model exists; thus, it may be regarded as “inconsistent”.

Nondeterminism

- **Problem**: A datalog program may have multiple stable models:

$$P = \{ \quad single(X) \leftarrow man(X), \neg husband(X)$$
$$husband(X) \leftarrow man(X), \neg single(X) \quad \}$$

$\mathbf{I} = \{man(dilbert)\}$

- $\mathbf{J}_1 = \{man(dilbert),\ single(dilbert)\}$ is a stable model:
  - $\mathbf{T}^0_{P,\mathbf{J}_1}(\mathbf{I}) = \{man(dilbert)\}$
  - $\mathbf{T}^1_{P,\mathbf{J}_1}(\mathbf{I}) = \{man(dilbert), single(dilbert)\}$ (apply 2nd rule)
  - $\mathbf{T}^2_{P,\mathbf{J}_1}(\mathbf{I}) = \{man(dilbert), single(dilbert)\} = \mathbf{T}^\omega_{P,\mathbf{J}_1}(\mathbf{I})$

- Similarly, $\mathbf{J}_1 = \{man(dilbert),\ husband(dilbert)\}$ is a stable model (symmetry)

Stable Model Semantics – Definition

- **Solution**: Define stable semantics of P as the intersection of all stable models (*certain semantics*):

Denote for a datalog$^\neg$ program P and $\mathbf{I} \in inst(edb(P))$ by $SM(P, \mathbf{I})$ the set of all stable models for $\mathbf{I}$ and P.

Defn. The stable models semantics of a datalog$^\neg$ program P for $\mathbf{I} \in inst(edb(P))$, denoted $P_{sm}(\mathbf{I})$, is given by

$$P_{sm}(\mathbf{I}) = \begin{cases} \bigcap SM(P, \mathbf{I}), & \text{if } SM(P, \mathbf{I}) \neq \emptyset, \\ \mathbf{B}(P, \mathbf{I}), & \text{otherwise.} \end{cases}$$

Examples

- $$P = \{ \quad single(X) \leftarrow man(X), \neg husband(X)$$
  $$husband(X) \leftarrow man(X), \neg single(X) \quad \}$$

  $P_{sm}(\{man(dilbert)\}) = \{man(dilbert)\}$

- $$P = \{p(a) \leftarrow \neg p(a), \quad q(b) \leftarrow p(a), \quad p(a) \leftarrow q(b)\}$$

  $P_{sm}(\emptyset) = \{p(a), p(b), q(a), q(b)\} = \mathbf{B}(P, \mathbf{I})$.

Some Properties

- **Proposition.** Each $J \in SM(P, \mathbf{I})$ is a minimal model $\mathbf{K}$ of P such that $\mathbf{K}|edb(P) = \mathbf{I}$.
- **Proposition.** Each $J \in SM(P, \mathbf{I})$ is a minimal fixpoint $\mathbf{K}$ of $\mathbf{T}_P$ such that $\mathbf{K}|edb(P) = \mathbf{I}$.
- **Theorem.** If P is a stratified program, than for every $\mathbf{I} \in edb(P)$, $P_{sm}(\mathbf{I}) = P_{strat}(\mathbf{I})$.

  Thus, stable model semantics extends stratified semantics to a larger class of programs
- Evaluation of stable semantics is intractable: Deciding whether $R(\vec{c}) \in P_{sm}(\mathbf{I})$ for given $R(\vec{c})$ and $\mathbf{I}$ (while P is fixed) is coNP-complete.

Well-Founded Semantics

- **Principle:** Use three truth values: Some facts are true, some false, all others are *unknown*.
- **Intuition:**
  - Positive literals must be derived by applying rules whose body is true
  - Conclude that a negated atom $\neg A$ is true, if A can not be derived by assuming that all facts which are not true are false.

  Example:

  Program P: $q(a)$:– $\neg p(a), r(a)$ $\quad r(a) \leftarrow \neg u(a)$

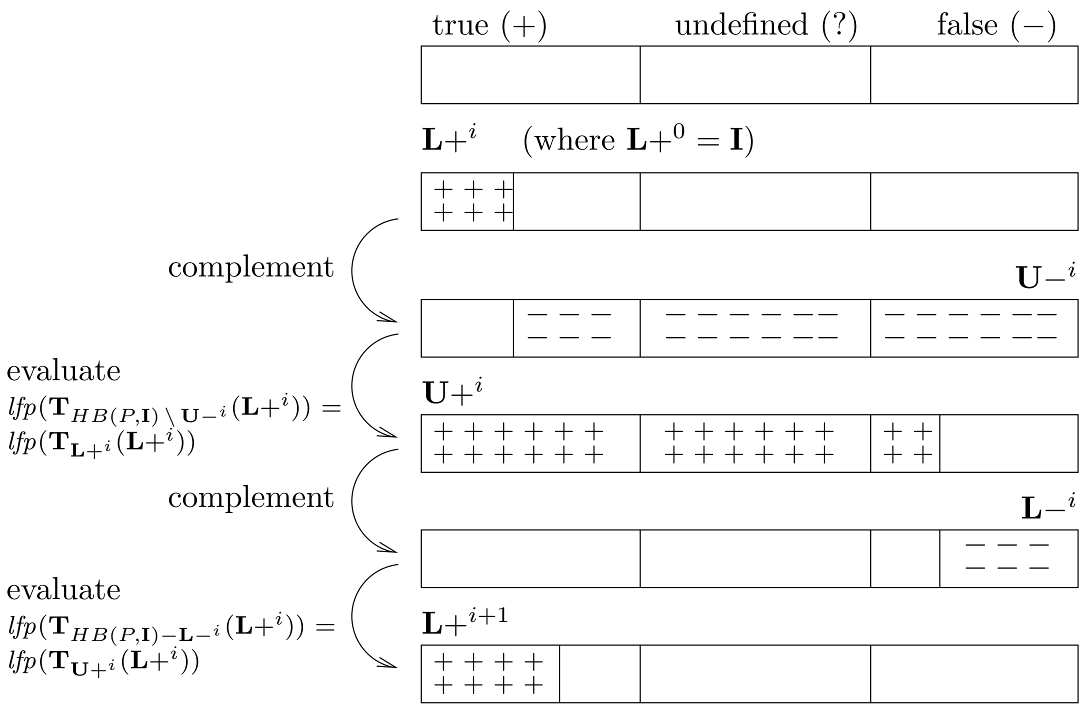
  $s(a)$:– $\neg t(a)$ $\quad p(a) \leftarrow u(a)$

  $t(a)$:– $\neg s(a)$

  $\mathbf{I} = \{\}$

Let $HB(P, \mathbf{I})$ be the set of all possible facts with constants $adom(P, \mathbf{I})$ for input $\mathbf{I}$.

1. $\mathbf{I}$ is a *lower bound* of the derivable positive facts $\mathbf{J}_+$.
2. All other facts $HB(P, \mathbf{I}) \setminus \mathbf{I}$ are an *upper bound* of the facts $\mathbf{J}_-$ which can't be derived (and thus are safely false), denoted $\mathbf{U}-$.
3. Thus, the consequences for $\mathbf{I}$ and P under negation at boundary ($\mathbf{I} = HB(P, I) \setminus \mathbf{U}-$) give an *upper bound* $\mathbf{U}+$ for the derivable positive facts.
4. All other facts $HB(P, \mathbf{I}) \setminus \mathbf{U}+$ give then a *lower bound* $\mathbf{L}-$ of the facts which can be safely false.
5. Thus, the consequences for $\mathbf{L}+$ and P under negation at boundary ($\mathbf{U}+ = HB(P, \mathbf{I}) \setminus \mathbf{L}-$) are a new *lower bound* for the derivable positive facts, denoted $\mathbf{L}+$
6. $\mathbf{I} \subseteq \mathbf{L}+ \Rightarrow$ iterate the process



Formal Definition

Define for P and $\mathbf{J} \in inst(sch(P))$ the operator $\widehat{\mathbf{T}_{P,\mathbf{J}}}$ on $inst(sch(P))$ by

$$\widehat{\mathbf{T}_{P,\mathbf{J}}}(\mathbf{K}) = \mathit{lfp}(\mathbf{T}_{P,\mathbf{K}}(\mathbf{J}))$$

i.e., the least fixpoint under negation as by $\mathbf{K}$, which includes $\mathbf{J}$.

Notice:

- $\widehat{\mathbf{T}_{P,\mathbf{J}}}(\mathbf{K})$ is computable by fixpoint iteration of $\mathbf{T}_{P,\mathbf{K}}$ starting from $\mathbf{J}$.
- $\widehat{\mathbf{T}_{P,\mathbf{J}}}$ is anti-monotonic, i.e., $\mathbf{K} \subseteq \mathbf{K}'$ implies that $\widehat{\mathbf{T}_{P,\mathbf{J}}}(\mathbf{K}') \subseteq \widehat{\mathbf{T}_{P,\mathbf{J}}}(\mathbf{K})$.
- Therefore, the "square operator" $\widehat{\mathbf{T}_{P,\mathbf{J}}}^2(\mathbf{K}) := \widehat{\mathbf{T}_{P,\mathbf{J}}}(\widehat{\mathbf{T}_{P,\mathbf{J}}}(\mathbf{K}))$ is monotonic (in fact continuous).
- Thus, $\widehat{\mathbf{T}_{P,\mathbf{J}}}^2$ has a least fixpoint, $\mathit{lfp}(\widehat{\mathbf{T}_{P,\mathbf{J}}}^2)$, which can be obtained by fixpoint iteration from $\emptyset$.

Example

Program P: $q(a) \leftarrow \neg p(a), r(a)$ $\quad p(a) \leftarrow u(a)$ $\quad s(a) \leftarrow \neg t(a)$

$r(a) \leftarrow \neg u(a)$ $\quad t(a) \leftarrow \neg s(a)$

Fixpoint iteration of $\widehat{\mathbf{T}_{P,\mathbf{I}}}^2$ for $\mathbf{I} = \{\}$:

$$\widehat{\mathbf{T}_{P,\mathbf{I}}}^0 = \emptyset$$
$$\widehat{\mathbf{T}_{P,\mathbf{I}}}^1 = \mathit{lfp}(\mathbf{T}_{P,\emptyset}(\mathbf{I})) = \{r(a), s(a), t(a)\}$$
$$\widehat{\mathbf{T}_{P,\mathbf{I}}}^2 = \mathit{lfp}(\mathbf{T}_{P,\{r(a),s(a),t(a)\}}(\mathbf{I})) = \{r(a), q(a)\}$$
$$\widehat{\mathbf{T}_{P,\mathbf{I}}}^3 = \mathit{lfp}(\mathbf{T}_{P,\{r(a),q(a)\}}(\mathbf{I})) = \{r(a), q(a), s(a), t(a)\}$$
$$\widehat{\mathbf{T}_{P,\mathbf{I}}}^4 = \mathit{lfp}(\mathbf{T}_{P,\{r(a),q(a),s(a),t(a)\}}(\mathbf{I})) = \{r(a), q(a)\} = \widehat{\mathbf{T}_{P,\mathbf{I}}}^2 = \mathit{lfp}(\widehat{\mathbf{T}_{P,\mathbf{I}}}^2)$$
$$\widehat{\mathbf{T}_{P,\mathbf{I}}}^5 = \widehat{\mathbf{T}_{P,\mathbf{I}}}^3$$

- Intuitively, the facts $r(a)$ and $q(a)$ are derivable, and thus should be true.
- The facts in $\mathbf{HB}(P, \mathbf{I}) \setminus \widehat{\mathbf{T}_{P,\mathbf{I}}}^3 = \{u(a), p(a)\}$ are then not derivable and should be false.
- The remaining facts $s(a)$ and $t(a)$ are unknown

Well-founded Semantics

Defn. For any datalog$^\neg$ program P and input $I \in inst(edb(P))$, a fact $A \in HB(P, \mathbf{I})$ is under well-founded semantics

- true, if $A \in \mathit{lfp}(\widehat{\mathbf{T}_{P,\mathbf{I}}}^2)$,
- false if $A \notin \widehat{\mathbf{T}_{P,\mathbf{I}}}(\mathit{lfp}(\widehat{\mathbf{T}_{P,\mathbf{I}}}^2))$, and
- unknown otherwise.

The positive outcome of program P for $\mathbf{I}$ under well-founded semantics, denoted $P_{wf}(\mathbf{I})$, is $\mathit{lfp}(\widehat{\mathbf{T}_{P,\mathbf{I}}}^2)$.

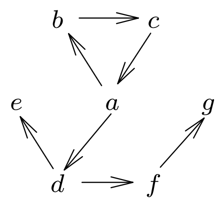
Example: For P and $\mathbf{I}$ above,

$$P_{wf}(\mathbf{I}) = \{r(a), q(a)\}$$

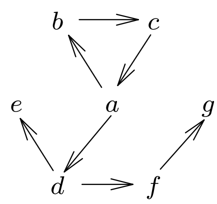
Example: Winning Positions

A two player game on a directed graph $G = \langle V, E \rangle$.

- Players I and II draw alternating.
- The drawing player moves from the current position following some arc to the next position.
- A player loses, if he can't move.



Example: Winning Positions/2



- **Wanted:** *winning positions*, i.e., nodes x from which the drawing player has a winning strategy (can play so that he will certainly win)
- In the example, the winning positions are d and f
- Elegant solution in datalog$^{\neg}$ under well-founded semantics:

$$P = \{\ win(X)\ \leftarrow\ e(X,Y), \neg win(Y)\ \}$$

Some Important Properties

- **Proposition.** The well-founded semantics is well-defined for every datalog$^\neg$ program P and input database $\mathbf{I}$.
- **Theorem.** If P is a stratified datalog$^\neg$ program, then for every $\mathbf{I} \in inst(edb(P))$ it holds that $A \in HB(P, \mathbf{I})$ is true (resp., false) under well-founded semantics iff $A \in P_{strat}(\mathbf{I})$ (resp., $A \notin P_{strat}(\mathbf{I})$).

  Well-founded semantics properly extends stratified semantics and approximates the stable semantics
- **Theorem.** For every datalog$^\neg$ program P and $\mathbf{I} \in inst(edb(P))$, if $A \in \mathbf{HB}(P, \mathbf{I})$ is true (resp., false) under well-founded semantics, then A is true (resp., false) in every stable model of P for $\mathbf{I}$.
- Evaluation of well-founded semantics is tractable: Deciding whether $R(\vec{c}) \in P_{wf}(\mathbf{I})$ for given $R(\vec{c})$ and $\mathbf{I}$ (while P is fixed) is feasible in polynomial time.

Readings

- S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

  Chapter 14 and 15.