LPs Basics (for ease, Datalog)

- Predicates are n-ary
- Terms are variables or constants
- Facts ground atoms
  For instance,
  $$has\_parent(mary, jo)$$
- Rules are of the form
  $$P(\mathbf{x}) \leftarrow \varphi(\mathbf{x}, \mathbf{y})$$
  where $\varphi(\mathbf{x}, \mathbf{y})$ is a formula built from atoms of the form $B(\mathbf{z})$ and connectors $\wedge, \vee, 0, 1$
  For instance,
  $$has\_father(x, y) \quad \leftarrow \quad has\_parent(x, y) \wedge Male(y)$$

- Extensional database (EDB): set of facts
- Intentional database (IDB): set of rules
- Logic Program $\mathcal{P}$:
  - $\mathcal{P} = EDB \cup IDB$
  - No predicate symbol in *EDB* occurs in the head of a rule in *IDB*
    - The principle is that we do not allow that *IDB* may redefine the extension of predicates in *EDB*
- *EDB* is usually, stored into a relational database

LPs Semantics: FOL semantics

- $\mathcal{P}^*$ is constructed as follows:
  1. set $\mathcal{P}^*$ to the set of all ground instantiations of rules in $\mathcal{P}$;
  2. replace a fact $p(\mathbf{c})$ in $\mathcal{P}^*$ with the rule $p(\mathbf{c}) \leftarrow 1$
  3. if atom A is not head of any rule in $\mathcal{P}^*$, then add $A \leftarrow 0$ to $\mathcal{P}^*$;
  4. replace several rules in $\mathcal{P}^*$ having same head

$$\left.\begin{array}{l} A \leftarrow \varphi_1 \\ A \leftarrow \varphi_2 \\ \quad \vdots \\ A \leftarrow \varphi_n \end{array}\right\} \text{ with } A \leftarrow \varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_n \,.$$

- Note: in $\mathcal{P}^*$ each atom $A \in B_{\mathcal{P}}$ is head of exactly one rule
- Herbrand Base of $\mathcal{P}$ is the set $B_{\mathcal{P}}$ of ground atoms
- Interpretation is a function $I : B_{\mathcal{P}} \rightarrow \{0, 1\}$.
- Model $I \models \mathcal{P}$ iff for all $r \in \mathcal{P}^*$ $I \models r$, where $I \models A \leftarrow \varphi$ iff $I(\varphi) \leq I(A)$

LP Query Answering

- Entailment: for a ground atom $p(\mathbf{c})$

$$\mathcal{P} \models p(\mathbf{c}) \text{ iff all models of } \mathcal{P} \text{ satisfy } p(\mathbf{c})$$

- Least model $M_{\mathcal{P}}$ of $\mathcal{P}$ exists and is least fixed-point of

$$T_{\mathcal{P}}(I)(A) = I(\varphi), \text{ for all } A \leftarrow \varphi \in \mathcal{P}^*$$

- M can be computed as the limit of

$$\begin{aligned} \mathbf{I}_0 &= \mathbf{0} \\ \mathbf{I}_{i+1} &= T_{\mathcal{P}}(\mathbf{I}_i) \, . \end{aligned}$$

Example

$$\mathcal{P} = \begin{cases} Q(x) & \leftarrow & B(x) \\ Q(x) & \leftarrow & C(x) \\ B(a) & & \\ C(b) & & \end{cases} \qquad \mathcal{P}^* = \begin{cases} Q(a) & \leftarrow & B(a) \vee C(a) \\ Q(b) & \leftarrow & B(b) \vee C(b) \\ B(a) & \leftarrow & 1 \\ C(b) & \leftarrow & 1 \end{cases}$$

$\mathbf{I}_i$	$Q(a)$	$Q(b)$	$B(a)$	$B(b)$	$C(a)$	$C(b)$
$\mathbf{I}_0$	0	0	0	0	0	0
$\mathbf{I}_1$	0	0	1	0	0	1
$\mathbf{I}_2$	1	1	1	0	0	1
$\mathbf{I}_3$	1	1	1	0	0	1

- $\mathbf{I}_2 = \mathbf{I}_3$, i.e. $T_{\mathcal{P}}(\mathbf{I}_2) = \mathbf{I}_3 = \mathbf{I}_2$
- $\mathbf{I}_2$ is least fixed-point and, thus, minimal model

LP Query Answering

Proposition

$\mathcal{P} \models p(t_1, ..., t_n)$ *iff* $M_{\mathcal{P}} \models p(t_1, ..., t_n)$.

- As a consequence, we may restrict our attention to minimal models only
- Query: is a rule of the form

$$q(\mathbf{x}) \leftarrow \varphi(\mathbf{x}, \mathbf{y})$$

- If $\mathcal{P} \models q(\mathbf{c})$ then $\mathbf{c}$ is called an answer to q
- The answer set of q w.r.t. $\mathcal{P}$ is defined as

$$ans(\mathcal{P}, q) = \{\mathbf{c} \mid \mathcal{P} \models q(\mathbf{c})\}$$

Toy Example

$$
\begin{aligned}
Q(x) &\leftarrow B(x) \\
Q(x) &\leftarrow C(x) \\
B(a) & \\
C(b) &
\end{aligned}
$$

$$\mathcal{P} \models Q(a) \quad \mathcal{P} \models Q(b) \qquad ans(\mathcal{P}, Q) = \{a, b\}$$

A general top-down query procedure for ground LPs

- Idea: use theory of fixed-point computation of equational systems over $\{0, 1\}$
- Assign a variable x_i to an atom $A_i \in B_{\mathcal{P}}$
- Map a rule $A \leftarrow f(A_1, \ldots, A_n) \in \mathcal{P}^*$ into the equation $x_A = f(x_{A_1}, \ldots, x_{A_n})$

$$p \leftarrow (q \vee r) \wedge t \text{ is mapped into } x_p = \min(\max(x_q, x_r), x_t)$$

- A LP $\mathcal{P}$ is thus mapped into the equational system, using $\mathcal{P}^*$

$$\begin{cases} x_1 = f_1(x_{1_1}, \ldots, x_{1_{a_1}}) \\ \vdots \\ x_n = f_n(x_{n_1}, \ldots, x_{n_{a_n}}) \end{cases}$$

- f_i is monotone and, thus, the system has least fixed-point, which is the limit of

$$\begin{aligned} \mathbf{y}_0 &= \mathbf{0} \\ \mathbf{y}_{i+1} &= \mathbf{f}(\mathbf{y}_i)\ . \end{aligned}$$

  where $\mathbf{f} = \langle f_1, \ldots, f_n \rangle$ and $\mathbf{f}(\mathbf{x}) = \langle f_1(x_1), \ldots, f_n(x_n) \rangle$
- The least-fixed point is the least model of $\mathcal{P}$

Example

$$\mathcal{P} = \begin{cases} Q(x) & \leftarrow & B(x) \\ Q(x) & \leftarrow & C(x) \\ B(a) & & \\ C(b) & & \end{cases} \qquad \mathcal{P}^* = \begin{cases} Q(a) & \leftarrow & B(a) \vee C(a) \\ Q(b) & \leftarrow & B(b) \vee C(b) \\ B(a) & \leftarrow & 1 \\ C(b) & \leftarrow & 1 \end{cases}$$

$$\begin{cases} x_{Q(a)} & = & \max(x_{B(a)}, x_{C(a)}) \\ x_{Q(b)} & = & \max(x_{B(b)}, x_{C(b)}) \\ x_{B(a)} & = & 1 \\ x_{C(b)} & = & 1 \end{cases}$$

$\mathbf{y}_i$	$x_{Q(a)}$	$x_{Q(b)}$	$x_{B(a)}$	$x_{B(b)}$	$x_{C(a)}$	$x_{C(b)}$
$\mathbf{y}_0$	0	0	0	0	0	0
$\mathbf{y}_1$	0	0	1	0	0	1
$\mathbf{y}_2$	1	1	1	0	0	1
$\mathbf{y}_3$	1	1	1	0	0	1

- $\mathbf{y}_2 = \mathbf{y}_3$, i.e. $\mathbf{f}(\mathbf{y}_2) = \mathbf{y}_3 = \mathbf{y}_2$
- $\mathbf{y}_2$ is least fixed-point and, thus, minimal model

- A simple query answering procedure to determine $ans(\mathcal{P}, q(\mathbf{x}))$:
  1. Convert $\mathcal{P}$ into $\mathcal{P}^*$
  2. Compute the minimal model $M_{\mathcal{P}}$ of $\mathcal{P}^*$, i.e. of $\mathcal{P}$
  3. Store the minimal model $M_{\mathcal{P}}$ of $\mathcal{P}^*$ in a database
  4. Translate $q(\mathbf{x})$ into a SQL statement
  5. Execute the SQL query over the relational database
- Problem: $M_{\mathcal{P}}$ may be huge (exponential in the size of $\mathcal{P}^*$)
- Possible solution: top-down query answering procedure
- First step: a top-down query answering procedure for ground queries
  - Given $q(\mathbf{c})$, check if $\mathcal{P} \models q(\mathbf{c})$ by computing just a fragment of $M_{\mathcal{P}}$ sufficient to answer the question
  - A top-down procedure exists for equational systems
  - Therefore, it works for LPs too

Procedure *Solve*($\mathcal{S}$, Q)
Input: monotonic system $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$, where $Q \subseteq V$ is the set of query variables;
Output: A set $B \subseteq V$, with $Q \subseteq B$ such that the mapping v equals lfp(f) on B.

1. $A := Q$, $dg := Q$, $in := \emptyset$, **for all** $x \in V$ **do** $v(x) = 0$, $exp(x) = 0$
2. **while** $A \neq \emptyset$ **do**
3. **select** $x_i \in A$, $A := A \setminus \{x_i\}$, $dg := dg \cup s(x_i)$
4. $r := f_i(v(x_{i_1}), ..., v(x_{i_{a_i}}))$
5. **if** $r \succ v(x_i)$ **then** $v(x_i) := r$, $A := A \cup (p(x_i) \cap dg)$ **fi**
6. **if not** $exp(x_i)$ **then** $exp(x_i) = 1$, $A := A \cup (s(x_i) \setminus in)$, $in := in \cup s(x_i)$ **fi**
7. **remove** x **from** A **if** $v(x) = \top$

od

$\mathcal{L}$ is complete lattice. For $q(\mathbf{x}) \leftarrow \phi \in \mathcal{P}$, with $\mathrm{s}(q)$ we denote the set of *sons* of q w.r.t. r, i.e. the set of intentional predicate symbols occurring in ϕ. With $\mathrm{p}(q)$ we denote the set of *parents* of q, i.e. the set $\mathrm{p}(q) = \{p_i : q \in \mathrm{s}(p_i, r)\}$ (the set of predicate symbols directly depending on q).

Example

$$\mathcal{P}^* = \left\{ \begin{array}{lcl} a & \leftarrow & b \wedge c \\ c & \leftarrow & a \vee d \\ b & \leftarrow & 1 \\ d & \leftarrow & 1 \end{array} \right. \quad \left\{ \begin{array}{lcl} x_a & = & \min(x_b, x_c) \\ x_c & = & \max(x_a, x_d) \\ x_b & = & 1 \\ x_d & = & 1 \end{array} \right.$$

$$\mathcal{P}^* \models a \quad ?$$

1. $\mathtt{A} = \{x_a\}, x_i = x_a, \mathtt{A} = \emptyset, \mathtt{dg} = \{x_a, x_b, x_c\}, r = 0, \mathtt{A} = \{x_b, x_c\}, \mathtt{exp}(x_a) = \mathtt{1}, \mathtt{in} = \{x_b, x_c\}$
2. $x_i = x_b, \mathtt{A} = \{x_c\}, r = 1, \mathtt{v}(x_b) = 1, \mathtt{A} = \{x_c, x_a\}, \mathtt{exp}(x_b) = \mathtt{1}$
3. $x_i = x_c, \mathtt{A} = \{x_a\}, \mathtt{dg} = \{x_a, x_b, x_c, x_d\}, r = 0, \mathtt{exp}(x_c) = \mathtt{1}, \mathtt{A} = \{x_a, x_d\}, \mathtt{in} = \{x_a, x_b, x_c, x_d\}$
4. $x_i = x_d, \mathtt{A} = \{x_a\}, r = 1, \mathtt{v}(x_d) = 1, \mathtt{exp}(x_d) = \mathtt{1}, \mathtt{A} = \{x_a, x_c\}$
5. $x_i = x_c, \mathtt{A} = \{x_a\}, r = 1, \mathtt{v}(x_c) = 1$
6. $x_i = x_a, \mathtt{A} = \emptyset, r = 1, \mathtt{v}(x_a) = 1$
7. `stop. return v` (in particular, $\mathtt{v}(x_a) = 1$)

- The fact that only a part of the model is computed becomes evident
  - the computation does not change if we add any program $\mathcal{P}'$ to $\mathcal{P}$ not containing atoms of $\mathcal{P}$
  - a bottom-up computation will consider $\mathcal{P}'$ as well
- Problem: we answer ground queries $q(\mathbf{c})$ only
  - There are too many $\mathbf{c}$ on which to test $q(\mathbf{c})$
- Solution: generalize $Solve(\mathcal{S}, Q)$ to compute ALL answers in one run only
  - Idea: the procedure is as for $Solve(\mathcal{S}, Q)$, but we compute answers incrementally

Computing All Answers

- A query is an atom Q (*query atom*) of the form $q(\mathbf{x})$
- For a given n-ary predicate p and a set of answers Δ_p of p, for convenience we represent Δ_p as an n-ary table tab_{Δ_p}, containing the records of the form $\langle c_1, \ldots, c_n \rangle$
- If Δ_p^1 and Δ_p^2 are two sets of answers for p, we write $\Delta_p^1 \preceq \Delta_p^2$ iff $\Delta_p^1 \subset \Delta_p^2$
- Our algorithm is an improved top-down query answering algorithm based on Semi Naive Evaluation for

  Datalog
  1. start by assuming all IDB (Intentional Database) relations empty;
  2. repeatedly evaluate the rules using the EDB (Extensional Database) and the previous IDB, to get a new IDB;
  3. end when no change to IDB.
- Consider a rule $p(\mathbf{x}) \leftarrow \varphi(\mathbf{x}, \mathbf{y})$ with predicates $p_1, \ldots, p_k$ in rule body $\varphi(\mathbf{x}, \mathbf{y})$
- Consider interpretation $\mathcal{I}$

$$\mathcal{I}(p_i(\mathbf{c})) = \begin{cases} 1, & \text{if } \mathbf{c} \in \Delta_{p_i} \\ 0, & \text{otherwise.} \end{cases}$$

- Assume

$$eval(p, \Delta_{p_1}, \ldots, \Delta_{p_k}) = \{\mathbf{c} \mid 1 = \max_{\mathbf{c}'} \mathcal{I}(\varphi(\mathbf{c}, \mathbf{c}'))\} ,$$

  where $\mathbf{c}'$ is a tuple of constants occurring in $\bigcup_i \Delta_{p_i}$
- *eval* can be implemented using SQL queries over relational tables $tab_{\Delta_{p_1}}, \ldots, tab_{\Delta_{p_k}}$
  - E.g.,

$$\texttt{path}(x, y) \quad \leftarrow \quad \texttt{edge}(x, y) \vee (\texttt{path}(x, z) \wedge \texttt{edge}(z, y))$$

  - $eval(path, \Delta_{edge}, \Delta_{path})$ is

$$\pi_{1,2}(tab_{\Delta_{\texttt{edge}}}) \cup \pi_{1,4}(tab_{\Delta_{\texttt{edge}}} \bowtie_{2=3} tab_{\Delta_{\texttt{path}}}) . \qquad (1)$$

Procedure *Answer*($\mathcal{L}$, $\mathcal{K}$, Q)
Input: Truth space $\mathcal{L} = \{0, 1\}$, knowledge base $\mathcal{K}$, set Q of query predicate symbols
Output: A mapping $\mathtt{v}$ such that it contains all answers of predicates in Q.

1. $\mathtt{A} := Q$, $\mathtt{dg} := Q$, $\mathtt{in} := \emptyset$, **for all** predicate symbols p in $\mathcal{P}$ **do** $\mathtt{v}(p) = \emptyset$, $\mathtt{exp}(p) = \mathtt{false}$
2. **while** $\mathtt{A} \neq \emptyset$ **do**
3. **select** $p_i \in \mathtt{A}$, $\mathtt{A} := \mathtt{A} \setminus \{p_i\}$, $\mathtt{dg} := \mathtt{dg} \cup \mathtt{s}(p_i)$
4. **if** (p_i extensional predicate) $\wedge$ ($\mathtt{v}(p_i) = \emptyset$) **then** $\mathtt{v}(p_i) := tab_{p_i}$
5. **if** (p_i intentional predicate) **then** $\Delta_{p_i} := eval(p_i, \mathtt{v}(p_{i_1}), \ldots, \mathtt{v}(p_{i_{k_i}}))$
6. **if** $\Delta_{p_i} \succ \mathtt{v}(p_i)$ **then** $\mathtt{v}(p_i) := \Delta_{p_i}$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{p}(p_i) \cap \mathtt{dg})$ **fi**
7. **if not** $\mathtt{exp}(p_i)$ **then** $\mathtt{exp}(p_i) = \mathtt{true}$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{s}(p_i) \setminus \mathtt{in})$, $\mathtt{in} := \mathtt{in} \cup \mathtt{s}(p_i)$ **fi**

od

- for predicate symbol p_i, $\mathtt{s}(p_i)$ is the set of predicate symbols occurring in the rule body of p_i, i.e. the *sons* of p_i;
- for predicate symbol p_i, $\mathtt{p}(p_i) = \{p_j : p_i \in \mathtt{s}(p_j)\}$, i.e. the *parents* of p_i;
- in step 5, $p_{i_1}, \ldots, p_{i_{k_i}}$ are all predicate symbols occurring in the rule body of p_i, i.e. the sons $\mathtt{s}(p_i) = \{p_{i_1}, \ldots, p_{i_{k_i}}\}$ of p_i.

Path Example

$$\texttt{path}(x, y) \leftarrow \texttt{edge}(x, y) \vee (\texttt{path}(x, z) \wedge \texttt{edge}(z, y))$$

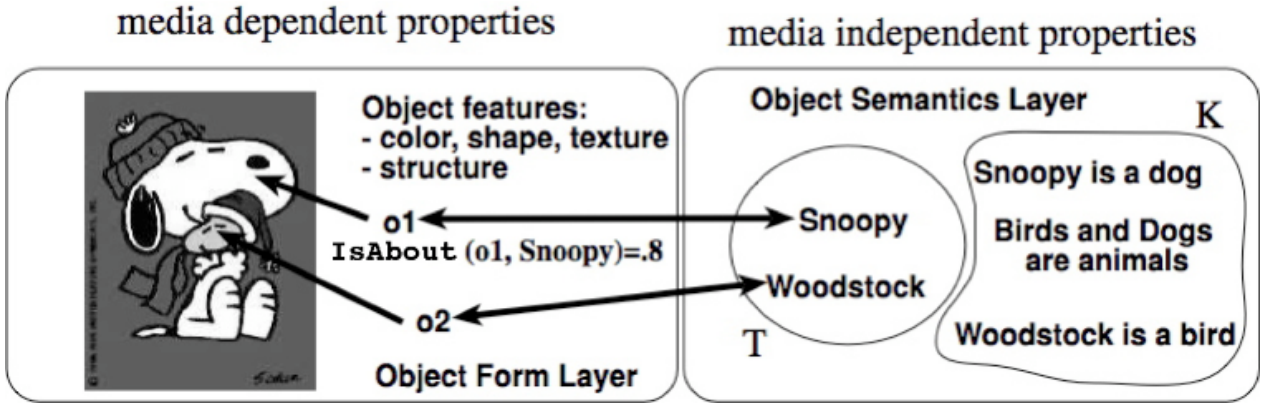
$tab_{\texttt{edge}}$	
c	b
a	c
b	a
a	b

1. $\texttt{A} := \{\texttt{path}\}, p_i := \texttt{path}, \texttt{A} := \emptyset, \texttt{dg} := \{\texttt{path}, \texttt{edge}\}, \Delta_{\texttt{path}} := \emptyset$
   $\texttt{exp}(\texttt{path}) := 1, \texttt{A} := \{\texttt{path}, \texttt{edge}\}, \texttt{in} := \{\texttt{path}, \texttt{edge}\}$
2. $p_i := \texttt{path}, \texttt{A} := \{\texttt{edge}\}, \Delta_{\texttt{path}} := \emptyset$
3. $p_i := \texttt{edge}, \texttt{A} := \emptyset, \Delta_{\texttt{edge}} \succ \texttt{v}(\texttt{edge}), \texttt{v}(\texttt{edge}) := \Delta_{\texttt{edge}}, \texttt{A} := \{\texttt{path}\}, \texttt{exp}(\texttt{edge}) := 1$
4. $p_i := \texttt{path}, \texttt{A} := \emptyset, \Delta_{\texttt{path}} \succ \texttt{v}(\texttt{path}), \texttt{v}(\texttt{path}) := \Delta_{\texttt{path}}, \texttt{A} := \{\texttt{path}\}$
5. $p_i := \texttt{path}, \texttt{A} := \emptyset, \Delta_{\texttt{path}} \succ \texttt{v}(\texttt{path}), \texttt{v}(\texttt{path}) := \Delta_{\texttt{path}}, \texttt{A} := \{\texttt{path}\}$
6. $p_i := \texttt{path}, \texttt{A} := \emptyset, \Delta_{\texttt{path}} \succ \texttt{v}(\texttt{path}), \texttt{v}(\texttt{path}) := \Delta_{\texttt{path}}, \texttt{A} := \{\texttt{path}\}$
7. $p_i := \texttt{path}, \texttt{A} := \emptyset, \Delta_{\texttt{path}} = \texttt{v}(\texttt{path})$
8. $\texttt{stop}.\ \texttt{return}\ \texttt{v}(\texttt{path})$

Iteri	Δ_{p_i}	$\texttt{v}(p_i)$
0.	—	$\texttt{v}(\texttt{edge}) = \texttt{v}(\texttt{path}) = \emptyset$
1.	$\Delta_{\texttt{path}} = \emptyset$	—
2.	$\Delta_{\texttt{path}} = \emptyset$	—
3.	$\Delta_{\texttt{edge}} = \{\langle a, b\rangle, \langle b, a\rangle, \langle a, c\rangle, \langle c, b\rangle\}$	$\texttt{v}(\texttt{edge}) = \Delta_{\texttt{edge}}$
4.	$\Delta_{\texttt{path}} = \{\langle a, b\rangle, \langle b, a\rangle, \langle a, c\rangle, \langle c, b\rangle\}$	$\texttt{v}(\texttt{path}) = \Delta_{\texttt{path}}$
5.	$\Delta_{\texttt{path}} = \{\langle a, a\rangle, \langle a, b\rangle, \langle a, c\rangle, \langle b, a\rangle, \langle b, b\rangle, \langle b, c\rangle, \langle c, a\rangle, \langle c, b\rangle\}$	$\texttt{v}(\texttt{path}) = \Delta_{\texttt{path}}$
6.	$\Delta_{\texttt{path}} = \{\langle a, a\rangle, \langle a, b\rangle, \langle a, c\rangle, \langle b, a\rangle, \langle b, b\rangle, \langle b, c\rangle, \langle c, a\rangle, \langle c, b\rangle, \langle c, c\rangle\}$	$\texttt{v}(\texttt{path}) = \Delta_{\texttt{path}}$
7.	$\Delta_{\texttt{path}} = \{\langle a, a\rangle, \langle a, b\rangle, \langle a, c\rangle, \langle b, a\rangle, \langle b, b\rangle, \langle b, c\rangle, \langle c, a\rangle, \langle c, b\rangle, \langle c, c\rangle\}$	—

Representing degrees in LPs

- How can we represent degrees of uncertainty and vagueness in LPs?
- Unfortunately, no standard exists yet
- However, as simple encoding is to make transform an n-ary predicate P into an $n+1$-ary predicate, where the additional argument stores the value:



IsAbout(o1,snoopy, 0.8)

- For instance, in LP systems we may write

$$q(h, s) \leftarrow hasLocation(h, hl), hasLocation(train, cl), close(hl, cl, s1), cheap(h, s2), s \text{ is } s1 \cdot s2$$

- But, then again such statements have to be appropriately be managed the system according to the underlying uncertainty or vagueness theory