Foundations of Query Languages

Dr. Fang Wei

Lehrstuhl für Datenbanken und Informationssysteme
Universität Freiburg

SS 2011

Positive Propositional Logic Programs

- A Horn clause is a rule of the form

$$A_0 \leftarrow A_1, \ldots, A_m \quad (m \geq 0)$$

  where each A_i is a propositional atom.
- A rule r of the form $A_0 \leftarrow$ is called a fact.
- A logic program is a finite set of Horn clauses.
- A atom A is true w.r.t. program P (denoted $P \models A$), if A is a classical consequence of P.

Positive Propositional Logic Programs

Example

$$
\begin{array}{lcl}
a & \leftarrow & b \\
a & \leftarrow & c \\
c & \leftarrow & d, e \\
d & \leftarrow & f \\
e & \leftarrow & g \\
b & \leftarrow & h \\
f & \leftarrow & \\
g & \leftarrow &
\end{array}
$$

$P \models f, P \models g, P \models d, \ldots, P \models c$

Relationship to SAT Problem

- Each program P can be viewed as a classical CNF $\phi(P)$
- Each rule r corresponds to a clause $\phi(r)$:

$$A_0 \leftarrow A_1, \ldots, A_m \rightleftharpoons A_0 \vee \neg A_1 \ldots \neg A_m$$

- $\phi(P) = \bigwedge_{r \in P} \phi(r)$.

Theorem

$P \models A$ holds if and only if $\phi(P) \wedge \neg A$ is unsatisfiable.

Positive Propositional Logic Programs

- Let $\mathcal{A}$ be all the atoms occurs in P. A model of P is the set $\mathcal{M} \subseteq \mathcal{A}$ which satisfies every rule $A_0 \leftarrow A_1, \ldots, A_m$ in P, i.e., $A_0 \in \mathcal{M}$ whenever $\{A_1, \ldots, A_m\} \subseteq \mathcal{M}$.
- The semantics of P is given by the *least model* of P, denoted $lm(P)$, i.e., the unique minimal model of P.

Positive Propositional Logic Programs

Example

$$
\begin{array}{lcl}
a & \leftarrow & b \\
a & \leftarrow & c \\
c & \leftarrow & d, e \\
d & \leftarrow & f \\
e & \leftarrow & g \\
b & \leftarrow & h \\
f & \leftarrow & \\
g & \leftarrow &
\end{array}
$$

$Im(P)$?

Propositional LP

- Existence of $lm(P)$ is trivial (it always exists)
- Reasoning: given a program P and an atom A, decide whether $A \in lm(P)$

Theorem

Propositional logic programming is P*-complete.*

Proof: (Membership)

- The semantics of a given program P can be defined as the least fixpoint of the immediate consequence operator $\mathbf{T}_P$
- This least fixpoint $lfp(\mathbf{T}_P)$ can be computed in polynomial time even if the "naive" evaluation algorithm is applied.
- The number of iterations is bounded by the number of rules plus 1.
- Each iteration step is clearly feasible in polynomial time.

Propositional LP P-hardness Proof

Proof: (Hardness)

- Encoding of a a deterministic Turing machine (DTM) T. Given a DTM T, an input string I and a number of steps N, where N is a polynomial of $|I|$, construct in logspace a program $P = P(T, I, N)$. An atom A such as $P \models A$ iff T accepts I in N steps.
- The transition function δ of a DTM with a single tape can be represented by a table whose rows are tuples $t = \langle s, \sigma, s', \sigma', d \rangle$. Such a tuple t expresses the following if-then-rule:

  if at some time instant τ the DTM is in state s, the cursor points to cell number π, and this cell contains symbol σ
  then at instant $\tau + 1$ the DTM is in state s', cell number π contains symbol σ', and the cursor points to cell number $\pi + d$.

Propositional LP P-hardness Proof

Proof: (Hardness)

- Encoding of a a deterministic Turing machine (DTM) T. Given a DTM T, an input string I and a number of steps N, where N is a polynomial of $|I|$, construct in logspace a program $P = P(T, I, N)$. An atom A such as $P \models A$ iff T accepts I in N steps.
- The transition function δ of a DTM with a single tape can be represented by a table whose rows are tuples $t = \langle s, \sigma, s', \sigma', d \rangle$. Such a tuple t expresses the following if-then-rule:

  if at some time instant τ the DTM is in state s, the cursor points to cell number π, and this cell contains symbol σ

  then at instant $\tau + 1$ the DTM is in state s', cell number π contains symbol σ', and the cursor points to cell number $\pi + d$.

Propositional LP P-hardness Proof

Proof: (Hardness)

- Encoding of a a deterministic Turing machine (DTM) T. Given a DTM T, an input string I and a number of steps N, where N is a polynomial of $|I|$, construct in logspace a program $P = P(T, I, N)$. An atom A such as $P \models A$ iff T accepts I in N steps.
- The transition function δ of a DTM with a single tape can be represented by a table whose rows are tuples $t = \langle s, \sigma, s', \sigma', d \rangle$. Such a tuple t expresses the following if-then-rule:

  if at some time instant τ the DTM is in state s, the cursor points to cell number π, and this cell contains symbol σ
  then at instant $\tau + 1$ the DTM is in state s', cell number π contains symbol σ', and the cursor points to cell number $\pi + d$.

Propositional LP P-hardness: the atoms

The propositional atoms in $P(T, I, N)$.
(there are many, but only polynomially many...)

symbol$_\alpha[\tau, \pi]$ for $0 \leq \tau \leq N$, $0 \leq \pi \leq N$ and $\alpha \in \Sigma$. Intuitive meaning: at instant τ of the computation, cell number π contains symbol α.

cursor$[\tau, \pi]$ for $0 \leq \tau \leq N$ and $0 \leq \pi \leq N$. Intuitive meaning: at instant τ, the cursor points to cell number π.

state$_s[\tau]$ for $0 \leq \tau \leq N$ and $s \in S$. Intuitive meaning: at instant τ, the DTM T is in state s.

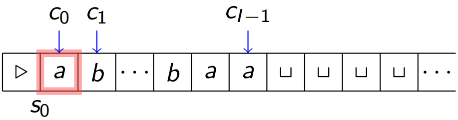
accept Intuitive meaning: T has reached state yes.

Propositional LP P-hardness: the rules

initialization facts: in $P(T, I, N)$:

$$\begin{array}{rcl} symbol_\sigma[0,\pi] & \leftarrow & \quad \text{for } 0 \leq \pi < |I|, \text{ where } I_\pi = \sigma \\ symbol_\sqcup[0,\pi] & \leftarrow & \quad \text{for } |I| \leq \pi \leq N \\ cursor[0,0] & \leftarrow & \\ state_{s_0}[0] & \leftarrow & \end{array}$$

The tape of the TM



Propositional LP P-hardness: the rules

- transition rules: for each entry $\langle s, \sigma, s', \sigma', d \rangle$, $0 \leq \tau < N$, $0 \leq \pi < N$, and $0 \leq \pi + d$.

$$\begin{array}{rcl} symbol_{\sigma'}[\tau+1,\pi] & \leftarrow & state_s[\tau], symbol_\sigma[\tau,\pi], cursor[\tau,\pi] \\ cursor[\tau+1,\pi+d] & \leftarrow & state_s[\tau], symbol_\sigma[\tau,\pi], cursor[\tau,\pi] \\ state_{s'}[\tau+1] & \leftarrow & state_s[\tau], symbol_\sigma[\tau,\pi], cursor[\tau,\pi] \end{array}$$

- inertia rules: where $0 \leq \tau < N$, $0 \leq \pi < \pi' \leq N$

$$\begin{array}{rcl} symbol_\sigma[\tau+1,\pi] & \leftarrow & symbol_\sigma[\tau,\pi], cursor[\tau,\pi'] \\ symbol_\sigma[\tau+1,\pi'] & \leftarrow & symbol_\sigma[\tau,\pi'], cursor[\tau,\pi] \end{array}$$

- accept rules: for $0 \leq \tau \leq N$

$$accept \leftarrow state_{\mathrm{yes}}[\tau]$$

Propositional LP P-hardness

- The encoding precisely simulates the behaviour machine T on input I up to N steps. (This can be formally shown by induction on the time steps.)
- $P(T, I, N) \models \mathit{accept}$ iff the DTM T accepts the input string I within N steps.
- The construction is feasible in Logspace

Horn clause inference is P-complete

Datalog Complexity

Query	Data Complexity	Program Complexity
Conjunctive query	AC_0	NP-complete
FO	AC_0	PSPACE-complete
Prop. LP		P-complete
Datalog	P-complete	EXPTIME-complete
Stratified Datalog	P-complete	EXPTIME-complete
Datalog(WFM)	P-complete	EXPTIME-complete
Datalog(INF)	P-complete	EXPTIME-complete
Datalog(Stable Model)	co-NP-complete	co-NEXPTIME-complete
Disjun. Datalog	Π_2^p-complete	co-NEXPTIME$^{\mathrm{NP}}$-complete

Complexity of Datalog Programs – Data complexity

Theorem

Datalog is data complete for P*.*

Proof: (Membership)

Effective reduction to Propositional Logic Programming is possible. Given P, D, A:

- Generate $\mathit{ground}(P, D)$
- Decide whether $\mathit{ground}(P, D) \models A$

Grounding of Datalog Rules

- Let U_D be the universe of D (usually the active universe (domain), i.e., the set of all domain elements present in D).
- The grounding of a rule r, denoted $ground(r, D)$, is the set of all rules obtained from r by all possible uniform substitutions of elements of U_D for the variables in r.

For any datalog program P and database D,

$$ground(P, D) = \bigcup_{r \in P} ground(r, D).$$

Grounding example

P and D:

$parent(X, Y) \leftarrow father(X, Y) \quad parent(X, Y) \leftarrow mother(X, Y)$
$ancestor(X, Y) \leftarrow parent(X, Y)$
$ancestor(X, Y) \leftarrow parent(X, Z), ancestor(Z, Y)$
$father(john, mary), father(joe, kurt), mother(mary, joe), mother(tina, kurt)$

$ground(P, D)$:

$parent(john, john) \leftarrow father(john, john)$
$parent(john, john) \leftarrow father(john, marry)$
$\ldots$
$parent(john, john) \leftarrow mother(john, john)$
$parent(john, marry) \leftarrow mother(john, marry)$
$\ldots$
$ancestor(john, john) \leftarrow parent(john, john)$
$\ldots$

Grounding complexity

Given P, D, the number of rules in $ground(P, D)$ is bounded by

$$|P| * \#consts(D)^{vmax}$$

- $vmax (\geq 1)$ is the maximum number of different variables in any rule $r \in P$
- $\#consts(D) = |U_D|$ is the number of constants in D (ass.: $|U_D| > 0$).

- $ground(P \cup D)$ can be exponential in the size of P.
- $ground(P \cup D)$ is polynomial in the size of D.

hence, the complexity of propositional logic programming is an upper bound for the data complexity.

Datalog data complexity: hardness

Proof: Hardness The P-hardness can be shown by writing a simple datalog *meta-interpreter* for propositional $\mathrm{LP}(k)$, where k is a constant.

- Represent rules $A_0 \leftarrow A_1, \ldots, A_i$, where $0 \leq i \leq k$, by tuples $\langle A_0, \ldots, A_i \rangle$ in an $(i+1)$-ary relation R_i on the propositional atoms.
- Then, a program P in $\mathrm{LP}(k)$ which is stored this way in a database $D(P)$ can be evaluated by a fixed datalog program $P_{MI}(k)$ which contains for each relation R_i, $0 \leq i \leq k$, a rule

$$T(X_0) \leftarrow T(X_1), \ldots, T(X_i), R_i(X_0, \ldots, X_i).$$

- $T(x)$ intuitively means that atom x is true. Then, $P \models A$ just if $P_{MI} \cup P(D) \models T(A)$. P-hardness of the data complexity of datalog is then immediately obtained.

Program Complexity Datalog

Theorem

Datalog is program complete for EXPTIME.

- **Membership**. Grounding P on D leads to a propositional program $grounding(P, D)$ whose size is exponential in the size of the fixed input database D. Hence, the program complexity is in EXPTIME.
- **Hardness**.
  - Adapt the propositional program $P(T, I, N)$ deciding acceptance of input I for T within N steps, where $N = 2^m$, $m = n^k(n = |I|)$ to a datalog program $P_{dat}(T, I, N)$
  - Note: We can not simply generate $P(T, I, N)$, since this program is exponentially large (and thus the reduction would not be polynomial!)

Datalog Program Complexity: Hardness

Main ideas for lifting $P(T, I, N)$ to $P_{dat}(T, I, N)$:

- use the predicates $symbol_\sigma(X, Y)$, $cursor(X, Y)$ and $state_s(X)$ instead of the propositional letters $symbol_\sigma[X, Y]$, $cursor[X, Y]$ and $state_s[X]$ respectively.
- The time points τ and tape positions π from 0 to $N-1$ are encoded in binary, i.e. by m-ary tuples $t_\tau = \langle c_1, \ldots, c_m \rangle$, $c_i \in \{0, 1\}$, $i = 1, \ldots, m$, such that $0 = \langle 0, \ldots, 0 \rangle$, $1 = \langle 0, \ldots, 1 \rangle$, $N - 1 = \langle 1, \ldots, 1 \rangle$.
- The functions $\tau + 1$ and $\pi + d$ are realized by means of the successor $Succ^m$ from a linear order $\leq^m$ on U^m.

Datalog Program Complexity: Hardness

The ground facts $Succ^1(0,1)$, $First^1(0)$, and $Last^1(1)$ are provided.

- The initialization facts $symbol_\sigma[0,\pi]$ are readily translated into the datalog rules

$$symbol_\sigma(\mathbf{X},\mathbf{t}) \leftarrow First^m(\mathbf{X}),$$

where $\mathbf{t}$ represents the position π,

- Similarly the facts $cursor[0,0]$ and $state_{s_0}[0]$.
- Initialization facts $symbol_\sqcup[0,\pi]$, where $|I| \leq \pi \leq N$, are translated to the rule

$$symbol_\sqcup(\mathbf{X},\mathbf{Y}) \leftarrow First^m(\mathbf{X}),\ \leq^m(\mathbf{t},\mathbf{Y})$$

where $\mathbf{t}$ represents the number $|I|$.

Datalog Program Complexity: Hardness

- Transition and inertia rules: for realizing $\tau + 1$ and $\pi + d$, use in the body atoms $Succ^m(\mathbf{X}, \mathbf{X}')$. For example, the clause

  $$symbol_{\sigma'}[\tau + 1, \pi] \leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi]$$

  is translated into

  $$symbol_{\sigma'}(\mathbf{X}', \mathbf{Y}) \leftarrow state_s(\mathbf{X}), symbol_\sigma(\mathbf{X}, \mathbf{Y}), cursor(\mathbf{X}, \mathbf{Y}), Succ^m(\mathbf{X}, \mathbf{X}').$$

- The translation of the accept rules is straightforward.

Defining $Succ^m(\mathbf{X}, \mathbf{X}')$ and $\leq^m$

- The ground facts $Succ^1(0, 1)$, $First^1(0)$, and $Last^1(1)$ are provided.
- For an inductive definition, suppose $Succ^i(\mathbf{X}, \mathbf{Y})$, $First^i(\mathbf{X})$, and $Last^i(\mathbf{X})$ tell the successor, the first, and the last element from a linear order $\leq^i$ on U^i, where $\mathbf{X}$ and $\mathbf{Y}$ have arity i. Then, use rules

$$
\begin{array}{rcl}
Succ^{i+1}(Z, \mathbf{X}, Z, \mathbf{Y}) & \leftarrow & Succ^i(\mathbf{X}, \mathbf{Y}) \\
Succ^{i+1}(Z, \mathbf{X}, Z', \mathbf{Y}) & \leftarrow & Succ^1(Z, Z'), Last^i(\mathbf{X}), First^i(\mathbf{Y}) \\
First^{i+1}(Z, \mathbf{X}) & \leftarrow & First^1(Z), First^i(\mathbf{X}) \\
Last^{i+1}(Z, \mathbf{X}) & \leftarrow & Last^1(Z), Last^i(\mathbf{X})
\end{array}
$$

Defining $Succ^m(\mathbf{X}, \mathbf{X}')$ and $\leq^m$

- The ground facts $Succ^1(0, 1)$, $First^1(0)$, and $Last^1(1)$ are provided.
- For an inductive definition, suppose $Succ^i(\mathbf{X}, \mathbf{Y})$, $First^i(\mathbf{X})$, and $Last^i(\mathbf{X})$ tell the successor, the first, and the last element from a linear order $\leq^i$ on U^i, where $\mathbf{X}$ and $\mathbf{Y}$ have arity i. Then, use rules

$$
\begin{array}{rcl}
Succ^{i+1}(0, \mathbf{X}, 0, \mathbf{Y}) & \leftarrow & Succ^i(\mathbf{X}, \mathbf{Y}) \\
Succ^{i+1}(1, \mathbf{X}, 1, \mathbf{Y}) & \leftarrow & Succ^i(\mathbf{X}, \mathbf{Y}) \\
Succ^{i+1}(0, \mathbf{X}, 1, \mathbf{Y}) & \leftarrow & Last^i(\mathbf{X}), First^i(\mathbf{Y}) \\
First^{i+1}(0, \mathbf{X}) & \leftarrow & First^i(\mathbf{X}) \\
Last^{i+1}(1, \mathbf{X}) & \leftarrow & Last^i(\mathbf{X})
\end{array}
$$

Defining $Succ^m(\mathbf{X}, \mathbf{X}')$ and $\leq^m$

- The ground facts $Succ^1(0,1)$, $First^1(0)$, and $Last^1(1)$ are provided.
- For an inductive definition, suppose $Succ^i(\mathbf{X}, \mathbf{Y})$, $First^i(\mathbf{X})$, and $Last^i(\mathbf{X})$ tell the successor, the first, and the last element from a linear order $\leq^i$ on U^i, where $\mathbf{X}$ and $\mathbf{Y}$ have arity i. Then, use rules

$$\begin{array}{rcl}
Succ^{i+1}(0, \mathbf{X}, 0, \mathbf{Y}) & \leftarrow & Succ^i(\mathbf{X}, \mathbf{Y}) \\
Succ^{i+1}(1, \mathbf{X}, 1, \mathbf{Y}) & \leftarrow & Succ^i(\mathbf{X}, \mathbf{Y}) \\
Succ^{i+1}(0, \mathbf{X}, 1, \mathbf{Y}) & \leftarrow & Last^i(\mathbf{X}), First^i(\mathbf{Y}) \\
First^{i+1}(0, \mathbf{X}) & \leftarrow & First^i(\mathbf{X}) \\
Last^{i+1}(1, \mathbf{X}) & \leftarrow & Last^i(\mathbf{X})
\end{array}$$

- The order $\leq^m$ is easily defined from $Succ^m$ by two clauses

$$\begin{array}{rcl}
\leq^m(\mathbf{X}, \mathbf{X}) & \leftarrow & \\
\leq^m(\mathbf{X}, \mathbf{Y}) & \leftarrow & Succ^m(\mathbf{X}, \mathbf{Z}),\ \leq^m(\mathbf{Z}, \mathbf{Y})
\end{array}$$

Datalog Program Complexity Conclusion

- Let $P_{dat}(T, I, N)$ denote the datalog program with empty *edb* described for T, I, and $N = 2^m$, $m = n^k$ (where $n = |I|$)
- $P_{dat}(T, I, N)$ is constructible from T and I in polynomial time (in fact, careful analysis shows feasibility in logarithmic space).
- $P_{dat}(T, I, N)$ has *accept* in its least model $\Leftrightarrow$ T accepts input I within N steps.
- Thus, the decision problem for any language in EXPTIME is reducible to deciding $P \models A$ for datalog program P and fact A.
- Consequently, deciding $P \models A$ for a given datalog program P and fact A is EXPTIME-hard.