Foundations of Data and Knowledge Systems

VU 181.212, WS 2010

7. Complexity and Expressive Power

Thomas Eiter and Reinhard Pichler

Institut für Informationssysteme
Technische Universität Wien

December 14, 2010

Outline

Outline

The Story So far

- Query languages with the form of logics
- Syntax, declarative and operational semantics

- How much resource (time, space) do we need for the computation of these semantics? ⇒ Complexity

- What kind of properties can a given query language express?
- Is Q_1 more expressive than Q_2? ⇒ Expressive power

A dream query language should have:

- lower complexity, and
- more expressive power

The Results Overview

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$^{\mathsf{NP}}$-complete

The Results Overview

Today we shall concentrate on

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$^{\text{NP}}$-complete

The Goal of this Lecture

- Basic concept of Turing machine, reduction, data complexity and program complexity
- How to prove completeness, Logspace reduction
- Get a taste of the hardness proofs of logic programming via nice encoding of a Turing machine
- Learn basics about expressive power

Decision Problems

- Problems where the answer is “yes” or “no”
- Formally,
  - A language L over some alphabet Σ.
  - An *instance* is given as a word $x \in \Sigma^*$.
  - Question: whether $x \in L$ holds
- The resources (i.e., either time or space) required in the worst case to find the correct answer for any instance x of a problem L is referred to as the *complexity* of the problem L

Complexities

Let P be a program with some query language, D_{in} input database and A a ground atom.

- **data complexity**
  Let P be *fixed*

  Instance. D_{in} and A.
  Question. Does $D_{in} \cup P \models A$ hold?
- **program complexity** (a.k.a. **expression complexity**)
  Let D_{in} be *fixed*.

  Instance. P and A.
  Question. Does $D_{in} \cup P \models A$ hold?
- **combined complexity**

  Instance. P, D_{in} and A.
  Question. Does $D_{in} \cup P \models A$ hold?

Complexity classes

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME} \subseteq \mathsf{NEXPTIME}$$

These are the classes of problems which can be solved in

- logarithmic space (L),
- non-deterministic logarithmic space (NL),
- polynomial time (P),
- non-deterministic polynomial time (NP),
- polynomial space (PSPACE),
- exponential time (EXPTIME), and
- non-deterministic exponential time (NEXPTIME).

we shall encounter in this course: P, NP, PSPACE, EXPTIME

Complexity classes – co Problems

- Any complexity class $\mathcal{C}$ has its complementary class denoted by co-$\mathcal{C}$.
- For every language $L \subseteq \Sigma^*$, let $\overline{L}$ denote its complement, i.e. the set $\Sigma^* \setminus L$. Then co-$\mathcal{C}$ is $\{\overline{L} \mid L \in \mathcal{C}\}$.
- Every deterministic complexity class is closed under complement, because one can simply add a last step to the algorithm which reverses the answer. (co-P?)

Complexity classes – Reductions

- **Logspace Reduction**
  - Let L_1 and L_2 be decision problems (languages over some alphabet Σ).
  - $R : \Sigma^* \to \Sigma^*$ be a function which can be computed in logarithmic space
  - The following property holds: for every $x \in \Sigma^*$, $x \in L_1$ iff $R(x) \in L_2$.
  - Then R is called a logarithmic-space reduction from L_1 to L_2 and we say that L_1 is reducible to L_2.
- **Hardness, Completeness**
  Let $\mathcal{C}$ be a set of languages. A language L is called $\mathcal{C}$-hard if any language L' in $\mathcal{C}$ is reducible to L. If L is $\mathcal{C}$-hard and $L \in \mathcal{C}$ then L is called *complete for* $\mathcal{C}$ or simply $\mathcal{C}$-complete.

Turing machines

A deterministic Turing machine (DTM) is defined as a quadruple

$$(S, \Sigma, \delta, s_0)$$

- S is a finite set of states,
- Σ is a finite alphabet of symbols, which contains a special symbol ␣ called the blank.
- δ is a transition function,
- and $s_0 \in S$ is the initial state.

The transition function δ is a map

$$\delta : \quad S \times \Sigma \quad \rightarrow \quad (S \cup \{\mathtt{yes}, \mathtt{no}\}) \times \Sigma \times \{\mathtt{-1,\ 0,\ +1}\},$$

where `yes`, and `no` denote two additional states not occurring in S, and `-1, 0, +1` denote motion directions.

Turing machines

DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM

$\triangleright$	a	b	$\cdots$	b	$\boxed{a}$	a	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\cdots$
					s_1						

Turing machines

DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c}
\hline
\triangleright & \boxed{a} & b & \cdots & b & a & a & \sqcup & \sqcup & \sqcup & \sqcup & \cdots \\
\hline
\multicolumn{2}{c}{s_0} & & & & & & & & & & \\
\end{array}$$

Turing machines

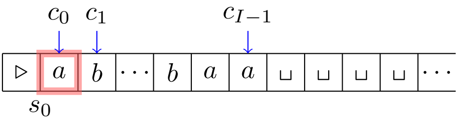
DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM



Turing machines

DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM

$\triangleright$	a	b	$\cdots$	b	$\boxed{a}$	a	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\cdots$
					s						

Transition Function example:

$$\delta(s, a) = (s', b, -1)$$

Turing machines

DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM

$\triangleright$	a	b	$\cdots$	b	$\boxed{b}$	a	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\cdots$

s

Transition Function example:

$$\delta(s, a) = (s', b, -1)$$

Turing machines

DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c}
\hline
\triangleright & a & b & \cdots & \boxed{b} & b & a & \sqcup & \sqcup & \sqcup & \sqcup & \cdots \\
\hline
\multicolumn{4}{c}{} & s' & \multicolumn{7}{c}{}
\end{array}$$

Transition Function example:

$$\delta(s, a) = (s', b, -1)$$

Turing machines

DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM

$\triangleright$	a	$\boxed{b}$	$\cdots$	b	b	a	a	a	b	b	$\cdots$

$\texttt{yes}$

Accept!

T halts, when any of the states $\texttt{yes}$ or $\texttt{no}$ is reached

Turing machines

DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s,\sigma) = (s', \sigma', d).$$

The tape of the TM

$\triangleright$	a	$\boxed{b}$	$\cdots$	b	b	a	a	a	b	b	$\cdots$

no

Reject!

T halts, when any of the states yes or no is reached

NDTM

A non-deterministic Turing machine (NDTM) is defined as a quadruple

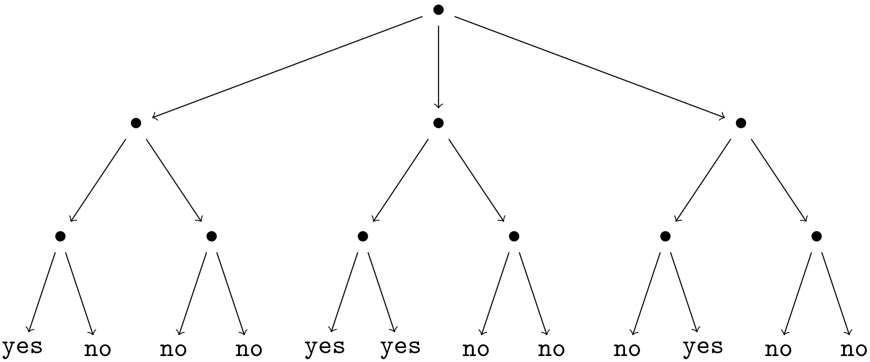
$$(S, \Sigma, \Delta, s_0)$$

- S, Σ, s_0 are the same as DTM
- Δ is no longer a function, but a relation:

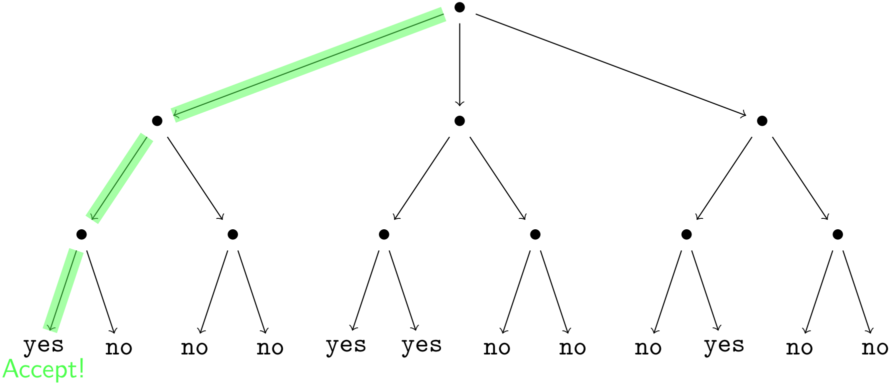
$$\Delta \ \subseteq \ (S \times \Sigma) \times (S \cup \{\texttt{yes}, \texttt{no}\}) \times \Sigma \times \{\texttt{-1, 0, +1}\}.$$

- A tuple with s and σ. If the number of such tuples is greater than one, the NDTM non-deterministically chooses any of them and operates accordingly.
- Unlike the case of a DTM, the definition of acceptance and rejection by a NDTM is asymmetric.

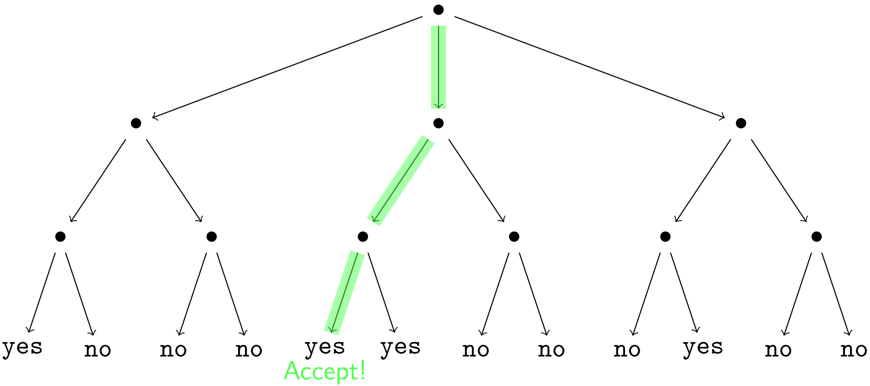
Nondeterministic Computation (Accept)



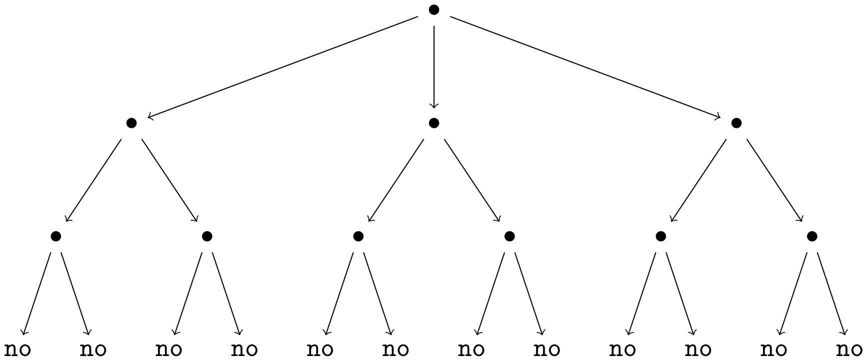
Nondeterministic Computation (Accept)



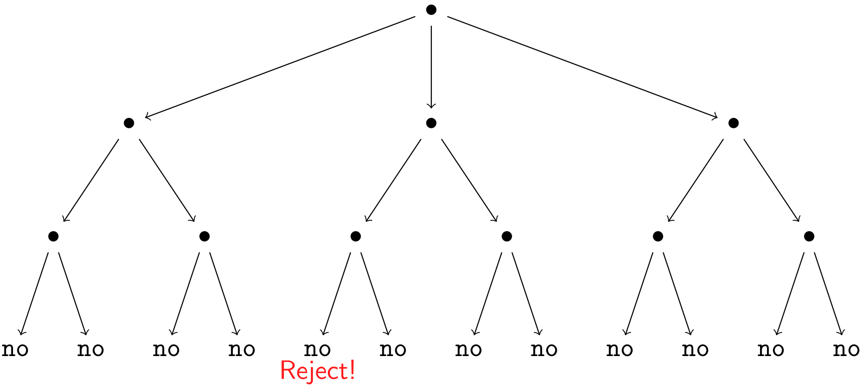
Nondeterministic Computation (Accept)



Nondeterministic Computation (Rejection)



Nondeterministic Computation (Rejection)



Outline

7.2 Propositional Logic Programming

Today we shall concentrate on

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$^{\text{NP}}$-complete

Propositional LP

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 $\mathit{lfp}(\mathbf{T}_P)$ can be computed in polynomial time even if the "naive" evaluation algorithm is applied.
- The number of iterations (i.e. applications of $\mathbf{T}_P$) 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: the atoms

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

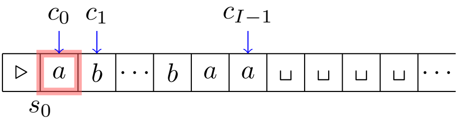
- *symbol*$_\alpha[\tau, \pi]$ for $0 \leq \tau \leq N$, $0 \leq \pi \leq N$ and $\alpha \in \Sigma$. Intuitive meaning: at instant τ of the computation, cell number π contains symbol α.
- *cursor*$[\tau, \pi]$ for $0 \leq \tau \leq N$ and $0 \leq \pi \leq N$. Intuitive meaning: at instant τ, the cursor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

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

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 $0 \leq \pi + d$.

$$\begin{array}{rcl} \mathit{symbol}_{\sigma'}[\tau+1,\pi] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi] \\ \mathit{cursor}[\tau+1,\pi+d] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi] \\ \mathit{state}_{s'}[\tau+1] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi] \end{array}$$

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

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

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

$$\mathit{accept} \leftarrow \mathit{state}_{\mathsf{yes}}[\tau]$$

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.)
- $P(T, I, N) \models$ *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

Outline

7.3 Datalog Complexity

Today we shall concentrate on

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 $ground(P, D)$
- Decide whether $ground(P, D) \models A$

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 predicates Succ^m, First^m, and Last^m are provided.

- The initialization facts $\mathit{symbol}_\sigma[0,\pi]$ are readily translated into the datalog rules

$$\mathit{symbol}_\sigma(\mathbf{X},\mathbf{t}) \leftarrow \mathit{First}^m(\mathbf{X}),$$

  where $\mathbf{t}$ represents the position π,
- Similarly the facts $\mathit{cursor}[0,0]$ and $\mathit{state}_{s_0}[0]$.
- Initialization facts $\mathit{symbol}_\sqcup[0,\pi]$, where $|I| \leq \pi \leq N$, are translated to the rule

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

  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 $\mathit{Succ}^m(\mathbf{X}, \mathbf{X}')$. For example, the clause

  $$\mathit{symbol}_{\sigma'}[\tau + 1, \pi] \leftarrow \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau, \pi], \mathit{cursor}[\tau, \pi]$$

  is translated into

  $$\mathit{symbol}_{\sigma'}(\mathbf{X}', \mathbf{Y}) \leftarrow \mathit{state}_s(\mathbf{X}), \mathit{symbol}_\sigma(\mathbf{X}, \mathbf{Y}), \mathit{cursor}(\mathbf{X}, \mathbf{Y}), \mathit{Succ}^m(\mathbf{X}, \mathbf{X}').$$

- The translation of the accept rules is straightforward.

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

- The ground facts $\mathit{Succ}^1(0,1)$, $\mathit{First}^1(0)$, and $\mathit{Last}^1(1)$ are provided.
- For an inductive definition, suppose $\mathit{Succ}^i(\mathbf{X}, \mathbf{Y})$, $\mathit{First}^i(\mathbf{X})$, and $\mathit{Last}^i(\mathbf{X})$ 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

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

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

- The ground facts $\mathit{Succ}^1(0, 1)$, $\mathit{First}^1(0)$, and $\mathit{Last}^1(1)$ are provided.
- For an inductive definition, suppose $\mathit{Succ}^i(\mathbf{X}, \mathbf{Y})$, $\mathit{First}^i(\mathbf{X})$, and $\mathit{Last}^i(\mathbf{X})$ 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

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

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

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

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.

Complexity of Datalog with Stratified Negation

Theorem

Stratified propositional logic programming with negation is P*-complete. Stratified datalog with negation is data complete for* P *and program complete for* EXPTIME*.*

- stratified P can be partitioned into disjoint sets $S_1, \ldots, S_n$ s.t. the semantics of P is computed by successively computing fixpoints of the immediate consequence operators $\mathbf{T}_{S_1}, \ldots, \mathbf{T}_{S_n}$.
- Let $\mathbf{I}_0$ be the initial instance over the extensional predicate symbols of P and let $\mathbf{I}_i$ (with $1 \leq i \leq n$) be defined as follows:

$$\mathbf{I}_1 := \mathbf{T}^{\omega}_{S_1}(\mathbf{I}_0),\ \ \mathbf{I}_2 := \mathbf{T}^{\omega}_{S_2}(\mathbf{I}_1),\ \ \ldots,\ \ \mathbf{I}_n := \mathbf{T}^{\omega}_{S_n}(\mathbf{I}_{n-1})$$

  Then the semantics of program P is given through the set $\mathbf{I}_n$.
- In the propositional case, $\mathbf{I}_n$ is clearly polynomially computable. Hence, stratified negation does not increase the complexity.

Outline

7.4 Complexity Stable Model

Today we shall concentrate on

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

Recall Stable Model Semantics

Let S be a (possibly infinite) set of ground normal clauses, i.e., of formulas of the form $A \leftarrow L_1 \wedge \ldots \wedge L_n$ where $n \geq 0$ and A is a ground atom and the L_i for $1 \leq i \leq n$ are ground literals.

Gelfond-Lifschitz Transformation

Let $B \subseteq HB$. The Gelfond-Lifschitz transform $GL_B(S)$ of S with respect to B is obtained from S as follows:

1. remove each clause whose antecedent contains a literal $\neg A$ with $A \in B$.
2. remove from the antecedents of the remaining clauses all negative literals.

Stable Model

An Herbrand interpretation $HI(B)$ is a stable model of S iff it is the unique minimal Herbrand model of $GL_B(S)$.

Complexity Prop. LP Stable model

Theorem

Given a propositional normal logic program P, deciding whether P has a stable model is NP*-complete.*

Membership. Clearly, P^I is polynomial time computable from P and I. Hence, a stable model M of P can be guessed and checked in polynomial time.

Stable Model Prop. LP - Hardness

Proof hardness

- Encoding of a non-deterministic Turing machine (NDTM) T.
  - Given a NDTM T, an input string I and a number of steps N, where N is a polynomial of $|I|$, construct in logspace a program $P = P(T, I, N)$.
  - P has a stable model iff T accepts I in non-deterministically N steps.
- Much similar to the encoding of DTM with propositional LP. Modification on deterministic property.

Stable Model Prop. LP - Hardness

Example: $\langle s, \sigma, s_1, \sigma'_1, d_1\rangle$, $\langle s, \sigma, s_2, \sigma'_2, d_2\rangle$
Transition rules $0 \leq \tau < N$, $0 \leq \pi < N$, and $0 \leq \pi + d$.

$$\begin{array}{rcl}
\mathit{symbol}_{\sigma'_1}[\tau+1,\pi] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi] \\
\mathit{cursor}[\tau+1,\pi+d_1] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi] \\
\mathit{state}_{s_1}[\tau+1] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi]
\end{array}$$

$$\begin{array}{rcl}
\mathit{symbol}_{\sigma'_2}[\tau+1,\pi] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi] \\
\mathit{cursor}[\tau+1,\pi+d_2] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi] \\
\mathit{state}_{s_2}[\tau+1] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi]
\end{array}$$

What is wrong here?
Enforcement violated:
At any time instance τ, there is exactly one cursor; each cell of the tape contains exactly one element; in exactly one state.

Stable Model Prop. LP - Hardness

- For each state s and symbol σ, introduce atoms $B_{s,\sigma,1}[\tau]$,..., $B_{s,\sigma,k}[\tau]$ for all $1 \leq \tau < N$ and for all transitions $\langle s, \sigma, s_i, \sigma'_i, d_i \rangle$, where $1 \leq i \leq k$.
- Add $B_{s,\sigma,i}[\tau]$ in the bodies of the transition rules for $\langle s, \sigma, s_i, \sigma'_i, d_i \rangle$.
- Add the rule

$$B_{s,\sigma,i}[\tau] \quad \leftarrow \quad \neg B_{s,\sigma,1}[\tau], \ldots, \neg B_{s,\sigma,i-1}[\tau], \neg B_{s,\sigma,i+1}[\tau], \ldots, \neg B_{s,\sigma,k}[\tau].$$

  Intuitively, these rules non-deterministically select precisely one of the possible transitions for s and σ at time instant τ, whose transition rules are enabled via $B_{s,\sigma,i}[\tau]$.
- Finally, add a rule

$$\mathit{accept} \leftarrow \neg \mathit{accept}.$$

  It ensures that *accept* is true in every stable model.

Stable Model Prop. LP - Hardness

Example: $\langle s, \sigma, s_1, \sigma'_1, d_1\rangle$, $\langle s, \sigma, s_2, \sigma'_2, d_2\rangle$

$$\begin{array}{rcl}
\mathit{symbol}_{\sigma'_1}[\tau+1,\pi] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi], {\color{red}B_{s,\sigma,1}[\tau]} \\
\mathit{cursor}[\tau+1,\pi+d_1] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi], {\color{red}B_{s,\sigma,1}[\tau]} \\
\mathit{state}_{s_1}[\tau+1] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi], {\color{red}B_{s,\sigma,1}[\tau]} \\
\mathit{symbol}_{\sigma'_2}[\tau+1,\pi] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi], {\color{red}B_{s,\sigma,2}[\tau]} \\
\mathit{cursor}[\tau+1,\pi+d_2] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi], {\color{red}B_{s,\sigma,2}[\tau]} \\
\mathit{state}_{s_2}[\tau+1] & \leftarrow & \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau,\pi], \mathit{cursor}[\tau,\pi], {\color{red}B_{s,\sigma,2}[\tau]}
\end{array}$$

$$\begin{array}{rcl}
B_{s,\sigma,1}[\tau] & \leftarrow & \neg B_{s,\sigma,2}[\tau] \\
B_{s,\sigma,2}[\tau] & \leftarrow & \neg B_{s,\sigma,1}[\tau]
\end{array}$$

One and only one atom from $B_{s,\sigma,1}[\tau]$ and $B_{s,\sigma,2}[\tau]$ is true. Which one?
Non-deterministic

Stable Model Prop. LP - Hardness

Proof.

- Assume there is a sequence of choices leading to the state yes.
  Let I be the set of the propositional atoms along the computation path reaching the state *accept*.
  Then *accept* $\in I$ due to the rule:

  $$\mathit{accept} \quad \leftarrow \quad \mathit{state}_{\mathrm{yes}}[\tau]$$

  Clearly I is a stable model of P.
- Assume there exists no sequence of choices leading to the state yes in the computation tree. Suppose I is a stable model of P and *accept* $\in I$.
  By minimality of I for P^I, it follows that $\mathit{state}_{\mathrm{yes}}[\tau] \in I$ for some τ;
  moreover, this means that a sequence of choices leads to yes.
  Contradiction.

Further Complexity Results

Theorem

Propositional logic programming with negation under well-founded semantics is P*-complete. Datalog with negation under well-founded semantics is data complete for* P *and program complete for* EXPTIME*.*

Theorem

Propositional logic programming with negation under inflationary semantics is P*-complete. Datalog with negation under inflationary semantics is data complete for* P *and program complete for* EXPTIME*.*

Further Complexity Results

Theorem

Propositional logic programming with negation under stable model semantics is co-NP*-complete. Datalog with negation under stable model semantics is data complete for* co-NP *and program complete for* co-NEXPTIME*.*

Note that the decision problem here is whether an atom is true in all stable models.

Further Complexity Results

Theorem

The program complexity of conjunctive queries is NP*-complete.*

Theorem

First-order queries are program-complete for PSPACE*. Their data complexity is in the class* AC^0*, which contains the languages recognized by unbounded fan-in circuits of polynomial size and constant depth.*

Further Complexity Results

Theorem

Logic programming is r.e.-complete.

Theorem

Nonrecursive logic programming is NEXPTIME*-complete.*

Outline

7.5 Expressive Power

- A query q defines a mapping $\mathcal{M}_q$ that assigns to each suitable input database D_{in} (over a fixed input schema) a result database $D_{out} = \mathcal{M}_q(D_{in})$ (over a fixed output schema)
- Formally, the expressive power of a query language Q is the set of mappings $\mathcal{M}_q$ for all queries q expressible in the language Q by some query expression (program) E
- Research tasks concerning expressive power:
  - Comparing two query languages Q_1 and Q_2 in their relative expressive power (e.g. FO vs. SQL vs. Datalog).
    This is important for designing and analysing a query language.
  - determining the absolute expressive power of a query language, e.g. proving that a given query language Q is able to express exactly all queries whose evaluation complexity is in a complexity class $\mathcal{C}$.
    We say Q captures $\mathcal{C}$ and write simply $Q = \mathcal{C}$.

Expressive Power

There is a substantial difference between showing that the query evaluation problem for a certain query language Q is $\mathcal{C}$-complete and showing that Q captures $\mathcal{C}$.

- If the evaluation problem for Q is $\mathcal{C}$-complete, then at least one $\mathcal{C}$-hard query is expressible in Q.
- If Q captures $\mathcal{C}$, then Q expresses all queries evaluable in $\mathcal{C}$ (including, of course, all $\mathcal{C}$-hard queries).
- Example: Evaluating Datalog is P hard (data complexity), but positive Datalog can only express monotone properties, however, there are of course problems in P which are non-monotonic.

Expressive Power: Ordered Structures

- To prove that a query language Q captures a machine-based complexity class $\mathcal{C}$, one usually shows that each $\mathcal{C}$-machine with (encodings of) finite structures as inputs that computes a generic query can be represented by an expression in language Q.
- A Turing machine works on a string encoding of the input database D. Such an encoding provides an implicit *linear order* on D, in particular, on all elements of the universe U_D
- Therefore, one often assumes that a linear ordering of the universe elements is predefined
- We consider here ordered databases whose schemas contain special relation symbols *Succ*, *First*, and *Last*

Expressive Power: Datalog

Theorem

datalog$^+$ $\subsetneq$ P.

Show that there exists no datalog$^+$ program P that can tell whether the universe U of the input database has an even number of elements.

Theorem

On ordered databases, datalog$^+$ *captures* P.

Expressive Power: More Results

Theorem

Nonrecursive range-restricted datalog with negation
= relational algebra
= domain-independent relational calculus
= first-order logic (without function symbols).

Theorem

On ordered databases, the following query languages capture P*:*

- *stratified datalog,*
- *datalog under well-founded semantics,*
- *datalog under inflationary semantics.*

Theorem

Datalog under stable model semantics captures co-NP*.*