

# Foundations of Data and Knowledge Systems

## EPCL Basic Training Camp 2012

### Part Six

Thomas Eiter and Reinhard Pichler

Institut für Informationssysteme  
Technische Universität Wien

December 21, 2012

# Outline

## 7. Complexity and Expressive Power

7.1 Complexity Classes and Reductions

7.2 Propositional Logic Programming

7.3 Datalog Complexity

7.4 Complexity Stable Model

7.5 Expressive Power

# Outline

## 7. Complexity and Expressive Power

### 7.1 Complexity Classes and Reductions

### 7.2 Propositional Logic Programming

### 7.3 Datalog Complexity

### 7.4 Complexity Stable Model

### 7.5 Expressive Power

# 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?  $\Rightarrow$  **Complexity**
- What kind of properties can a given query language express?
- Is  $Q_1$  more expressive than  $Q_2$ ?  $\Rightarrow$  **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	$\Pi_2^P$ -complete	co-NEXPTIME <sup>NP</sup> -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	$\Pi_2^P$ -complete	co-NEXPTIME <sup>NP</sup> -complete

# Goal of this Lecture

- Recall basic concept of Turing machine
- Recall problem reduction (logspace reduction), completeness,
- Define data complexity and program complexity
- 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  $\Sigma$ .
  - 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

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME$$

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  $\text{co-}\mathcal{C}$ .
- For every language  $L \subseteq \Sigma^*$ , let  $\bar{L}$  denote its complement, i.e. the set  $\Sigma^* \setminus L$ . Then  $\text{co-}\mathcal{C}$  is  $\{\bar{L} \mid L \in \mathcal{C}\}$ .
- 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  $\Sigma$ ).
- $R: \Sigma^* \rightarrow \Sigma^*$  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,
- $\Sigma$  is a finite alphabet of symbols, which contains a special symbol  $\sqcup$  called the blank.
- $\delta$  is a transition function,
- and  $s_0 \in S$  is the initial state.

The transition function  $\delta$  is a map

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

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

# Turing machines

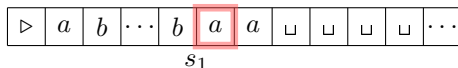
DTM quadruple:

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

Transition function:

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

The tape of the TM



# Turing machines

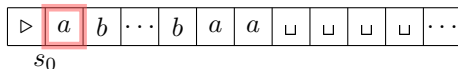
DTM quadruple:

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

Transition function:

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

The tape of the TM



# Turing machines

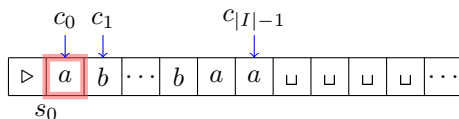
DTM quadruple:

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

Transition function:

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

The tape of the TM





# Turing machines

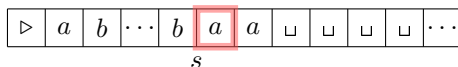
DTM quadruple:

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

Transition function:

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

The tape of the TM



Transition Function example:

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

# Turing machines

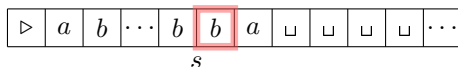
DTM quadruple:

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

Transition function:

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

The tape of the TM



Transition Function example:

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

# Turing machines

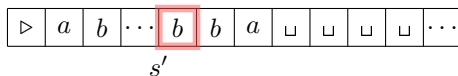
DTM quadruple:

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

Transition function:

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

The tape of the TM



Transition Function example:

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

# Turing machines

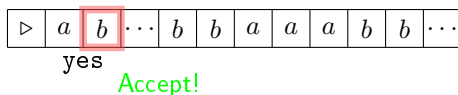
DTM quadruple:

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

Transition function:

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

The tape of the TM



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

# Turing machines

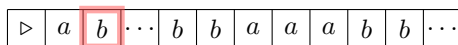
DTM quadruple:

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

Transition function:

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

The tape of the TM



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

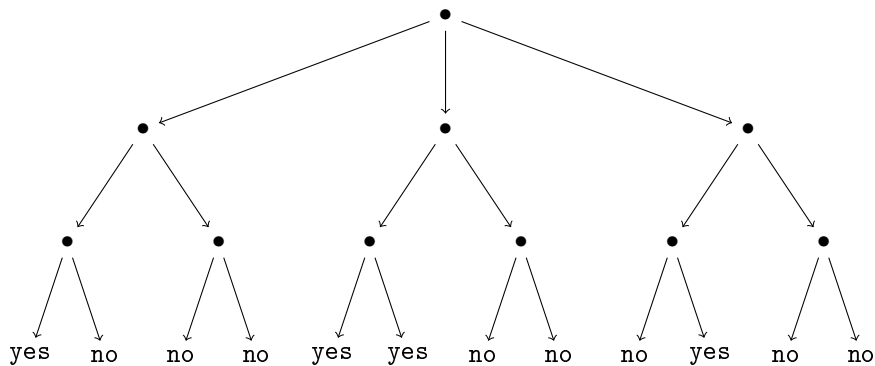
$$(S, \Sigma, \Delta, s_0)$$

- $S, \Sigma, s_0$  are the same as DTM
- $\Delta$  is no longer a function, but a relation:

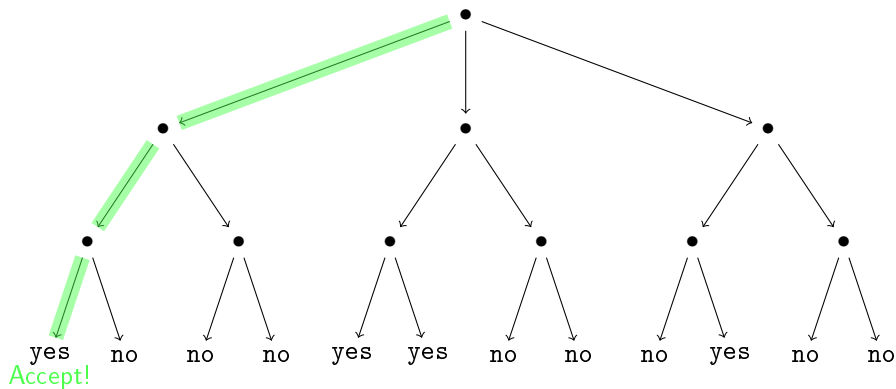
$$\Delta \subseteq (S \times \Sigma) \times (S \cup \{\text{yes, no}\}) \times \Sigma \times \{-1, 0, +1\}.$$

- A tuple with  $s$  and  $\sigma$ . 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)

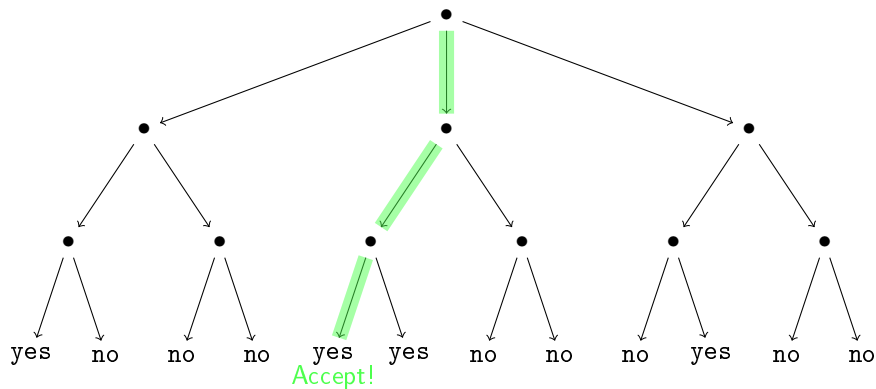


# Nondeterministic Computation (Accept)

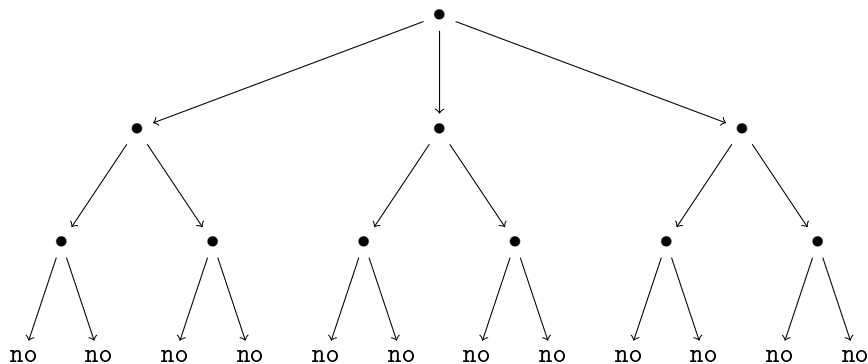




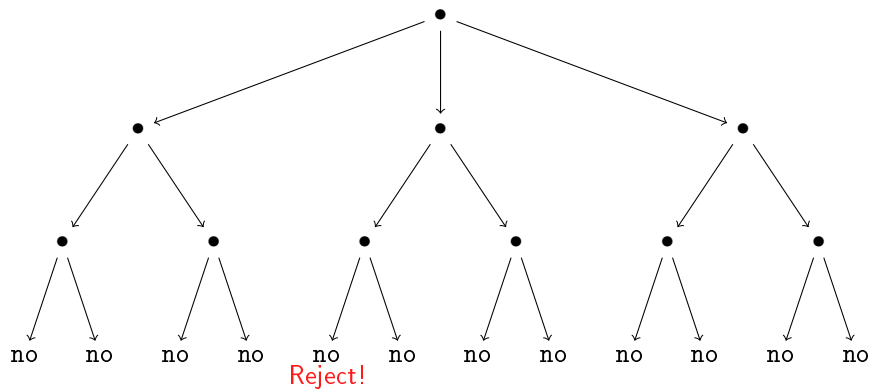
# Nondeterministic Computation (Accept)



# Nondeterministic Computation (Rejection)



# Nondeterministic Computation (Rejection)



# Outline

## 7. Complexity and Expressive Power

7.1 Complexity Classes and Reductions

**7.2 Propositional Logic Programming**

7.3 Datalog Complexity

7.4 Complexity Stable Model

7.5 Expressive Power

## 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		<b>P-complete</b>
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	$\Pi_2^P$ -complete	co-NEXPTIME <sup>NP</sup> -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  $lfp(\mathbf{T}_P)$  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 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  $\delta$  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  $\tau$  the DTM is in state  $s$ , the cursor points to cell number  $\pi$ , and this cell contains symbol  $\sigma$   
**then** at instant  $\tau + 1$  the DTM is in state  $s'$ , cell number  $\pi$  contains symbol  $\sigma'$ , 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* <sub>$\alpha$</sub>  $[\tau, \pi]$  for  $0 \leq \tau \leq N$ ,  $0 \leq \pi \leq N$  and  $\alpha \in \Sigma$ . Intuitive meaning: at instant  $\tau$  of the computation, cell number  $\pi$  contains symbol  $\alpha$ .

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

*state* <sub>$s$</sub>  $[\tau]$  for  $0 \leq \tau \leq N$  and  $s \in S$ . Intuitive meaning: at instant  $\tau$ , the DTM  $T$  is in state  $s$ .

*accept* Intuitive meaning:  $T$  has reached state yes.

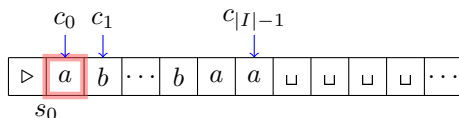


# Propositional LP P-hardness: the rules

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

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

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  $0 \leq \pi + d$ .

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

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

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

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

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

# 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.)
- $P(T, I, N) \models \text{accept}$  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. Complexity and Expressive Power

7.1 Complexity Classes and Reductions

7.2 Propositional Logic Programming

**7.3 Datalog Complexity**

7.4 Complexity Stable Model

7.5 Expressive Power

## 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	<b>P-complete</b>	<b>EXPTIME-complete</b>
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	$\Pi_2^P$ -complete	co-NEXPTIME <sup>NP</sup> -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*:

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

*ground(P, D)*:

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



## Grounding complexity

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

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

- $vmax(\geq 1)$  is the maximum number of different variables in any rule  $r \in P$
- $\#const(D) = |U_D|$  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  $LP(k)$ , where  $k$  is a constant.

- Represent rules  $A_0 \leftarrow A_1, \dots, A_i$ , where  $0 \leq i \leq k$ , by tuples  $\langle A_0, \dots, A_i \rangle$  in an  $(i + 1)$ -ary relation  $R_i$  on the propositional atoms.
- Then, a program  $P$  in  $LP(k)$  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

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

- $T(x)$  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  $\text{grounding}(P, D)$  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  $P_{\text{dat}}(T, I, N)$
  - 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  $\tau$  and tape positions  $\pi$  from 0 to  $N - 1$  are encoded in binary, i.e. by  $m$ -ary tuples  $t_\tau = \langle c_1, \dots, c_m \rangle$ ,  $c_i \in \{0, 1\}$ ,  $i = 1, \dots, m$ , such that  $0 = \langle 0, \dots, 0 \rangle$ ,  $1 = \langle 0, \dots, 1 \rangle$ ,  $N - 1 = \langle 1, \dots, 1 \rangle$ .
- 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**  $symbol_\sigma[0, \pi]$  are readily translated into the datalog rules

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

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

- 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

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

# 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

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



## 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

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

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

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

# 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  $S_1, \dots, S_n$  s.t. the semantics of  $P$  is computed by successively computing fixpoints of the immediate consequence operators  $\mathbf{T}_{S_1}, \dots, \mathbf{T}_{S_n}$ .
- 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:

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

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. Complexity and Expressive Power

7.1 Complexity Classes and Reductions

7.2 Propositional Logic Programming

7.3 Datalog Complexity

**7.4 Complexity Stable Model**

7.5 Expressive Power

## 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	$\Pi_2^P$ -complete	co-NEXPTIME <sup>NP</sup> -complete

## Recall Stable Model Semantics

Let  $S$  be a (possibly infinite) set of **ground** normal clauses, i.e., of formulas of the form  $A \leftarrow L_1 \wedge \dots \wedge L_n$  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  $0 \leq \pi + d$ .

$$\begin{aligned} symbol_{\sigma'_1}[\tau + 1, \pi] &\leftarrow state_s[\tau], symbol_{\sigma}[\tau, \pi], cursor[\tau, \pi] \\ cursor[\tau + 1, \pi + d_1] &\leftarrow state_s[\tau], symbol_{\sigma}[\tau, \pi], cursor[\tau, \pi] \\ state_{s_1}[\tau + 1] &\leftarrow state_s[\tau], symbol_{\sigma}[\tau, \pi], cursor[\tau, \pi] \end{aligned}$$

$$\begin{aligned} symbol_{\sigma'_2}[\tau + 1, \pi] &\leftarrow state_s[\tau], symbol_{\sigma}[\tau, \pi], cursor[\tau, \pi] \\ cursor[\tau + 1, \pi + d_2] &\leftarrow state_s[\tau], symbol_{\sigma}[\tau, \pi], cursor[\tau, \pi] \\ state_{s_2}[\tau + 1] &\leftarrow state_s[\tau], symbol_{\sigma}[\tau, \pi], cursor[\tau, \pi] \end{aligned}$$

What is wrong here?

Enforcement violated:

At any time instance  $\tau$ , 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  $\sigma$ , introduce atoms  $B_{s,\sigma,1}[\tau], \dots, B_{s,\sigma,k}[\tau]$  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

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

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

- Finally, add a rule

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

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

# Stable Model Prop. LP - Hardness

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

$$\begin{aligned}
 symbol_{\sigma'_1}[\tau + 1, \pi] &\leftarrow state_s[\tau], symbol_{\sigma}[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,1}[\tau] \\
 cursor[\tau + 1, \pi + d_1] &\leftarrow state_s[\tau], symbol_{\sigma}[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,1}[\tau] \\
 state_{s_1}[\tau + 1] &\leftarrow state_s[\tau], symbol_{\sigma}[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,1}[\tau] \\
 symbol_{\sigma'_2}[\tau + 1, \pi] &\leftarrow state_s[\tau], symbol_{\sigma}[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,2}[\tau] \\
 cursor[\tau + 1, \pi + d_2] &\leftarrow state_s[\tau], symbol_{\sigma}[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,2}[\tau] \\
 state_{s_2}[\tau + 1] &\leftarrow state_s[\tau], symbol_{\sigma}[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,2}[\tau]
 \end{aligned}$$

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

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:

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

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  $state_{yes}[\tau] \in I$  for some  $\tau$ ;  
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. Complexity and Expressive Power

7.1 Complexity Classes and Reductions

7.2 Propositional Logic Programming

7.3 Datalog Complexity

7.4 Complexity Stable Model

7.5 Expressive Power

## 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

$\text{datalog}^+$ : datalog with input negation.

## Theorem

$\text{datalog}^+ \subsetneq \text{P}$ .

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

## Theorem

*On ordered databases,  $\text{datalog}^+$  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.*

## From Expressive Power to Complexity Results

- If query language  $\mathcal{L}$  captures complexity class  $C$ , and  $C$  has complete problems, then  $\mathcal{L}$  has  $C$ -complete data complexity.

## From Expressive Power to Complexity Results

- If query language  $\mathcal{L}$  captures complexity class  $C$ , and  $C$  has complete problems, then  $\mathcal{L}$  has  $C$ -complete data complexity.
- Under certain conditions, results on expressive power allow one to derive complexity results about program complexity.



## From Expressive Power to Complexity Results

- If query language  $\mathcal{L}$  captures complexity class  $C$ , and  $C$  has complete problems, then  $\mathcal{L}$  has  $C$ -complete data complexity.
- Under certain conditions, results on expressive power allow one to derive complexity results about program complexity.
- This is linked to *succinct representation of problem inputs*
  - represent inputs  $w \in \{0, 1\}^*$  by **Boolean circuits**  $C_w$  with  $n = \log |w|$  input bits
  - $C_w(b_1, \dots, b_n) = 1 \Leftrightarrow w_i = 1$  where  $\text{bin}(i) = (b_1, \dots, b_n)$

## From Expressive Power to Complexity Results

- If query language  $\mathcal{L}$  captures complexity class  $C$ , and  $C$  has complete problems, then  $\mathcal{L}$  has  $C$ -complete data complexity.
- Under certain conditions, results on expressive power allow one to derive complexity results about program complexity.
- This is linked to *succinct representation of problem inputs*
  - represent inputs  $w \in \{0, 1\}^*$  by **Boolean circuits**  $C_w$  with  $n = \log |w|$  input bits
  - $C_w(b_1, \dots, b_n) = 1 \Leftrightarrow w_i = 1$  where  $\text{bin}(i) = (b_1, \dots, b_n)$
- “Complexity Upgrading” results (rooted in Papadimitriou/Yannakakis, 1985)
  - conclude from completeness of problem  $\Pi$  for class  $C_1$  under specific reductions, the hardness of its “succinct” variant  $s\Pi$  for an “exponentially” harder class  $C_2$

**Example:** Succinct 3-Colorability is NEXPTIME-complete

## From Expressive Power to Complexity Results, cont'd

- If  $\mathcal{L}$  is capable of simulating generic Boolean circuits (query  $Q_{BC}$ ), then a given succinct input  $C_w$  can be “uncompressed” to  $w$

**Example:** Datalog (with input negation)

## From Expressive Power to Complexity Results, cont'd

- If  $\mathcal{L}$  is capable of simulating generic Boolean circuits (query  $Q_{BC}$ ), then a given succinct input  $C_w$  can be “uncompressed” to  $w$

**Example:** Datalog (with input negation)

- If  $\mathcal{L}$  captures  $C_1$ , all database queries (thus also every query  $f$  hard under the specific reductions) can be expressed ( $Q_f$ )

## From Expressive Power to Complexity Results, cont'd

- If  $\mathcal{L}$  is capable of simulating generic Boolean circuits (query  $Q_{BC}$ ), then a given succinct input  $C_w$  can be “uncompressed” to  $w$

**Example:** Datalog (with input negation)

- If  $\mathcal{L}$  captures  $C_1$ , all database queries (thus also every query  $f$  hard under the specific reductions) can be expressed ( $Q_f$ )
- By composing  $Q_{BC}$  and  $Q_f$  (assuming certain modularity of  $\mathcal{L}$ ), some  $C_2$ -hard query is expressed.

**Example:** Datalog under stable semantics is NEXPTIME-complete

## From Expressive Power to Complexity Results, cont'd

- If  $\mathcal{L}$  is capable of simulating generic Boolean circuits (query  $Q_{BC}$ ), then a given succinct input  $C_w$  can be “uncompressed” to  $w$

**Example:** Datalog (with input negation)

- If  $\mathcal{L}$  captures  $C_1$ , all database queries (thus also every query  $f$  hard under the specific reductions) can be expressed ( $Q_f$ )
- By composing  $Q_{BC}$  and  $Q_f$  (assuming certain modularity of  $\mathcal{L}$ ), some  $C_2$ -hard query is expressed.

**Example:** Datalog under stable semantics is NEXPTIME-complete

- To this end, some abstract properties of  $\mathcal{L}$  are required

More details:

- E\_, Gottlob, Mannila: Disjunctive Datalog, ACM TODS 22:364-417, 1997
- G. Gottlob, N. Leone, H.Veith, Succinctness as a Source of Complexity in Logical Formalisms, APAL 97:231–260, 1999.