VO Deductive Databases

WS 2014/2015

Stefan Woltran

Institut für Informationssysteme
Arbeitsbereich DBAI

Overview

- Introduction;
- Background (Propositional Logic, Complexity Theory);
- Propositional Answer-Set Programming;
- Comparing Propositional Logic Programs;
- Non-Ground Answer-Set Programming;
- Comparing Non-Ground Logic Programs;
- Program Transformations.

What are Deductive Databases?

"*The area of Deductive Databases originates from the fusion of database technology and logic programming*".

Abiteboul, Hull, Vianu: *Foundations of Databases*, Addison-Wesley, 1995.

- Common aspects of databases and logic programs?
- What are the conceptual differences?

Common Aspects of Databases and Logic Programs

- *Declarative* methodology:
  - Order of “statements” does not matter:
  - neither of data nor of program rules,
  - neither within queries nor within rules.
  - In reality: indexing, prolog (SLD-resolution).
- Both support some *Closed-World Assumption* (CWA).

Specific Issues in Databases

- DBMS: database management system: organizes physical data and its access;
  - DDL (data definition language), DML (data manipulation language), query languages;
- concurrency, security issues;
- recovery.

- *Database theory* focuses on the description of data and querying facilities.

Specific Issues in Logic Programming

- Usually, logic programs are understood as a set of Horn clauses in first-order logic:

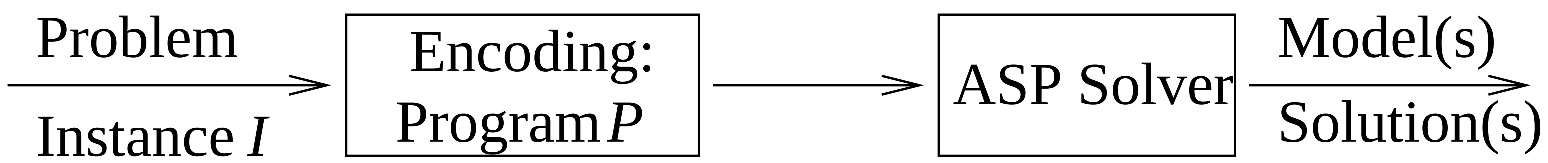
$$\forall X \left(h(\vec{t}) \leftarrow b_1(\vec{t_1}), \ldots, b_n(\vec{t_n}) \right)$$

- Typical questions
  - Given logic program P, goal A; is there some substitution θ such that $P \models A\theta$.
  - Given logic program P, compute Herbrand-models of P.
- SLD (Selection Rule Driven) Resolution.

The Answer-Set Programming Paradigm

- Compared to prolog: no function symbols.
- Focus of interest: Models.
- Models should be used to describe the solutions of a given problem.
- Typically, models are not unique.

The Answer-Set Programming Paradigm



- Variant 1: Recompile P for each instance I;
- Variant 2: Fixed encoding P for problem; instance I is added to P as input.

Querying Databases

- Central theoretical model: Relational model/calculus
  - introduced by E. F. Codd in 1970 (since then: several variants).
- Most important practical query language: SQL
  - since 1974 (IBM), standardized in 1986/87.

Querying Databases (Relational Model)

- Example. Graph with some designated vertices.
- Relations:

$$e(a, b), e(b, c), e(a, d), e(d, f).$$
$$v(a), v(d).$$

- Query: “Neighborhood” of designated vertices.

$$\pi_3(\sigma_{1=2}(v \times e)).$$

Querying Databases (SQL)

- Example. Graph with some designated vertices.
- Tables:

$$\texttt{table}\, e(x : \texttt{string},\, y : \texttt{string}).$$

$$\texttt{table}\, v(z : \texttt{string}).$$

- Query: “Neighborhood” of designated vertices.

$$\texttt{select}\ y\ \texttt{from}\ e, v\ \texttt{where}\ z = x.$$

DATALOG

- DATALOG stems from extending (rule-based variants of) relational calculus.
- Answer-Set Programming can be understood as DATALOG (without an explicit distinction between data and query).
- Example from above with relations:

$$e(a,b), e(b,c), e(a,d), e(d,f).$$
$$v(a), v(d).$$

- Query: “Neighborhood” of designated vertices.

$$out(Y) \leftarrow v(X), e(X,Y).$$

DATALOG (ctd.)

- We can do much more now, e.g. compute *all* nodes accessible from the designated vertices;

$$out(Y) \leftarrow v(X), e(X, Y).$$
$$out(Y) \leftarrow out(X), e(X, Y).$$

- Remark: Not possible in (traditional) SQL.

Ultimate Goals in this Lecture

- How to decide whether two logic programs (resp. queries) are doing the same job?
  - What means doing the same job?
- Benefits:
  - Deeper understanding of Answer-Set Programming.
  - Theoretical foundation of program optimization (this calls for understanding the computational complexity, however).

Equivalence Notions — Motivating Example

$\left\{ \texttt{edge(a,b). edge(b,c). ...} \right\}$ KB

$$\left\{ \begin{array}{l} \texttt{path(X,Y) :- edge(X,Y).} \\ \texttt{path(X,Z) :- path(X,Y), edge(Y,Z).} \end{array} \right\} Q1$$

$$\left\{ \begin{array}{l} \texttt{path(X,Y) :- edge(X,Y).} \\ \texttt{path(X,Z) :- path(X,Y), path(Y,Z).} \end{array} \right\} Q2$$

- Ordinary Equivalence (OE):

  Do $Q1 \cup KB$ and $Q2 \cup KB$ have the same output?

- More interesting problem: Query Equivalence (QE):

  Do $Q1 \cup KB$ and $Q2 \cup KB$ have the same output, for each KB (i.e., for any set of edges)?

Equivalence Notions — Motivating Example (ctd.)

$\left\{ \texttt{edge(a,b). edge(b,c). path(c,d). ...} \right\}$ KB

```
path(X,Y) :- edge(X,Y).
path(X,Z) :- path(X,Y), edge(Y,Z).
```
$Q1$

```
path(X,Y) :- edge(X,Y).
path(X,Z) :- path(X,Y), path(Y,Z).
```
$Q2$

- Query Equivalence (QE):

  Do $Q1 \cup KB$ and $Q2 \cup KB$ have the same output, for each KB (i.e., for any set of edges)?

- Different problem: Uniform Equivalence (UE)

  Do $Q1 \cup I$ and $Q2 \cup I$ have the same output, for any input I (i.e., also paths may be part of the input)?

Equivalence Notions — Motivating Example (ctd.)

P:
```
edge(a,b). edge(b,c). ...
...   :- path(X,Y).
...   :- ...
```

$M1$:
```
path(X,Y) :- edge(X,Y).
path(X,Z) :- path(X,Y), edge(Y,Z).
```

$M2$:
```
path(X,Y) :- edge(X,Y).
path(X,Z) :- path(X,Y), path(Y,Z).
```

- Strong Equivalence (SE):

  Do $M1 \cup P$ and $M2 \cup P$ have the same output for any program P?

Equivalence Notions — Motivating Example (ctd.)

$$\left\{\begin{array}{l}\texttt{edge(a,b). edge(b,c). ...}\\ \texttt{... :- path(X,Y).}\\ \texttt{... :- ...}\end{array}\right\} \quad P$$

$$\left\{\begin{array}{l}\texttt{path(X,Y) :- edge(X,Y).}\\ \texttt{path(X,Z) :- path(X,Y), edge(Y,Z).}\end{array}\right\} \quad M1$$

$$\left\{\begin{array}{l}\texttt{path(X,Y) :- edge(X,Y).}\\ \texttt{path(X,Z) :- path(X,Y), path(Y,Z).}\end{array}\right\} \quad M2$$

- Better: Application Specific Equivalence

  Do $M1 \cup P$ and $M2 \cup P$ have the same output for any program P where `edge` appears only in rule heads and `path` only in rule bodies?

Background—Roadmap

- Propositional Logic (PL);
- Quantified Propositional Logic (QBFs);
- Complexity Theory (Basic Aspects).

Why using Propositional Logic?

- Semantics of DATALOG is given by grounding; (and propositional logic makes life easier...)
- Example from above (simplified):

$$e(a,b).\ \ e(b,c).\ \ v(a).$$
$$out(Y) \leftarrow v(X), e(X,Y).$$

amounts to

$$e(a,b).\ \ e(b,c).\ \ v(a).$$
$$out(a) \leftarrow v(a), e(a,a). \quad out(a) \leftarrow v(b), e(b,a). \quad out(a) \leftarrow v(c), e(c,a).$$
$$out(b) \leftarrow v(a), e(a,b). \quad out(b) \leftarrow v(b), e(b,b). \quad out(b) \leftarrow v(c), e(c,b).$$
$$out(c) \leftarrow v(a), e(a,c). \quad out(c) \leftarrow v(b), e(b,c). \quad out(c) \leftarrow v(c), e(c,c).$$

PL—Syntax

- The *alphabet* of propositional logic is given by
  - (primitive) logical connectives $\neg$, $\wedge$, $\vee$, $\supset$;
  - a countable set of *propositional atoms* $\mathcal{A} = \{p, q, r, \ldots\}$;
  - *propositional constants* $\top$ and $\bot$; and
  - auxiliary symbols (,).

- A *(propositional) formula (over $\mathcal{A}$)* is defined as follows:

  P_1 : Each propositional atom and constant is a formula;

  P_2 : If ϕ, ψ are formulas, then also $(\neg\phi)$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, and $(\phi \supset \psi)$ are formulas.

  P_3 : Formulas are solely given by P_1 and P_2.

☞ For the sake of readability, we omit parentheses if not ambiguous; e.g. $p \vee q \wedge \neg r \supset s$ amounts to $((p \vee (q \wedge (\neg r))) \supset s)$.

PL—Semantics

- A *(propositional) interpretation* (over $\mathcal{A}$) is a function $m : \mathcal{A} \to \{0, 1\}$.

- The *truth-value*, $V^m(\cdot)$, of a formula under an interpretation m is defined as follows:

  $W_0 :$ $V^m(\top) = 1$; $V^m(\bot) = 0$;

  $W_1 :$ $V^m(p) = m(p)$, for any $p \in \mathcal{A}$;

  $W_2 :$ $V^m(\neg\phi) = 1 - V^m(\phi)$;

  $W_3 :$ $V^m(\phi \wedge \psi) = V^m(\phi) * V^m(\psi)$;

  $W_4 :$ $V^m(\phi \vee \psi) = 1$, if $V^m(\phi) + V^m(\psi) \geq 1$, otherwise $V^m(\phi \vee \psi) = 0$;

  $W_5 :$ $V^m(\phi \supset \psi) = 1$, if $V^m(\phi) \leq V^m(\psi)$, otherwise $V^m(\phi \supset \psi) = 0$.

PL—Semantics (ctd.)

- We also consider interpretations as sets $I \subseteq \mathcal{A}$.
- Given interpretations $m : \mathcal{A} \to \{0, 1\}$ and $I \subseteq \mathcal{A}$. We have the following correspondences:

$$
\begin{aligned}
I_m &= \{p \in \mathcal{A} : m(p) = 1\}; \\
m_I(p) &= \begin{cases} 1 & \text{if } p \in I; \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}
$$

- We write $I \models \phi$ iff $V^{m_I}(\phi) = 1$.

PL—Semantics (ctd.)

Some important concepts:

- ϕ is *true* under m if $V^m(\phi) = 1$.
- ϕ is *false* under m if $V^m(\phi) = 0$.
- ϕ is *satisfiable* if there is some m such that $V^m(\phi) = 1$.
- ϕ is *valid* if $V^m(\phi) = 1$, for any m.
- m is a *model* of ϕ if $V^m(\phi) = 1$.
- ϕ are ψ are *(logically, classically) equivalent* iff $V^m(\phi) = V^m(\psi)$ for any m.
- The set of models of a formula ϕ, is denoted by $Mod(\phi)$.

PL—Designated Models

- Considering interpretations as sets $I \subseteq \mathcal{A}$, the following concepts are natural and important later: A model I of a formula ϕ is called
  - *minimal* iff, there is no model $J \subset I$ of ϕ;
  - *maximal* iff, there is no model $J \supset I$ of ϕ.

- Example: The formula $(p \supset q)$ has three models (over $\{p, q\}$):

$$I_1 = \emptyset,\ I_2 = \{q\},\ \text{and}\ I_3 = \{p, q\};$$

I_1 is the minimal model of $(p \supset q)$; I_3 its maximal model.

Replacement Property of Classical Logic

- Let $\theta[\phi/\psi]$ denote the formula resulting from θ by replacing an occurrence of ϕ in θ by formula ψ.

  Then, θ and $\theta[\phi/\psi]$ are logically equivalent, whenever ϕ and ψ are logically equivalent.

Normalforms

- A formula is in *conjunctive normalform* (CNF) if it is of the form

$$\bigwedge_{i=1}^{n}\Big(\bigvee_{j=1}^{m(i)} L_{ij}\Big),$$

where L_{ij} is a *literal*, i.e., either an atom or a negated atom.

- A formula in CNF is *positive* iff no negation occurs in it.
- A formula in CNF is *Horn* iff each $\bigvee_{j=1}^{m(i)} L_{ij}$ contains at most one unnegated atom.
- A formula in CNF is *definite* iff each $\bigvee_{j=1}^{m(i)} L_{ij}$ contains at least one unnegated atom.
- Observations:
  - For each formula, there exists an equivalent formula in CNF;
  - a definite Horn formula is always satisfiable;
  - a positive formula is always satisfiable.

Theories

- A (*propositional*) *theory* is a set of formulas.
- Let T, T' theories.
  - An interpretation m is a *model* of T iff $V^m(\phi) = 1$, for all $\phi \in T$.
  - T is *satisfiable* iff there is a model for T.
  - T and T' are *equivalent* iff T and T' have the same models.
  - We usually identify a theory T as the conjunction of its elements, i.e., $T = \bigwedge_{\phi \in T} \phi$.

Renaming Concepts

- It is sometimes convenient to use a "copy" of the alphabet.
- For instance, by using a function $(\cdot)'$ mapping each atom p to a globally new one p'.
- For a formula ϕ, ϕ' results from ϕ by replacing any occurrence of any atom p by p'.
- For any set of atoms A, A' is defined as the set $\{p' \mid p \in A\}$.
- Important building blocks used later, given a set of atoms A:

$$
\begin{aligned}
(A \leq A') \quad &:= \quad \bigwedge_{p \in A} (p \supset p'); \\
(A < A') \quad &:= \quad (A \leq A') \land \neg(A' \leq A).
\end{aligned}
$$

- This allows to compare interpretations (blackboard!)

Renaming Concepts (ctd.)

- **Proposition:** Let $A \subseteq \mathcal{A}$ be a set of atoms, $X, Y \subseteq A$, and I an interpretation, such that $(I \cap A) = X$ and $(I \cap A') = Y'$. Then,
  1. I is a model of $A \leq A'$ iff $X \subseteq Y$;
  2. I is a model of $A < A'$ iff $X \subset Y$.

- Let $A = \mathcal{A} = \{a, b\}$ and $\phi = a \wedge b$. Then,
  1. $\phi \wedge \phi' \wedge (A \leq A')$ has a model $I = \{a, b, a', b'\}$;
  2. $\phi \wedge \phi' \wedge (A < A')$ has no model.

Quantified Propositional Logic—Introduction

- Basic idea of quantified propositional logic (QPL):
  - extend syntax by unary connectives $\exists p$, $\forall p$, for any atom p.
  - Intuitive semantics:

$$\exists p \phi \iff \text{there is truth assignment to } p\text{, s.t. } \phi \text{ becomes true;}$$
$$\forall p \phi \iff \text{for any truth assignment to } p\text{, } \phi \text{ becomes true.}$$

- This allows for propositions over semantical concepts of propositional logic within the language.
- Yields some form of "second-order propositional-logic".
- Formulas of QPL are often called *quantified Boolean formulas* (QBFs).

QPL—Introduction (ctd.)

- Example: Consider the propositional formula

$$\phi = (p \supset q) \wedge (q \supset p);$$

  - ϕ is true under interpretations $m(p) = m(q)$.

- Now consider the following QBFs:
  - $\exists p \exists q \phi$ is true (since ϕ is satisfiable);
  - $\forall p \forall q \phi$ is false (since ϕ is not valid);
  - $\exists p \forall q \phi$ is false (see models of ϕ);
  - $\forall p \exists q \phi$ is true (see models of ϕ).

QPL—Syntax

- Extend alphabet of propositional logic by *quantifier symbols* $\exists$, $\forall$ (existential, resp. universal quantifier). We use Q to refer to any quantifier.

- A QBF (over $\mathcal{A}$) is defined as follows:

  $(1):$ Each propositional atom and constant is a QBF;

  $(2):$ If Φ, Ψ are QBFs, then also $(\neg\Phi)$, $(\Phi \wedge \Psi)$, $(\Phi \vee \Psi)$, and $(\Phi \supset \Psi)$ are QBFs.

  (3) If Φ is a QBF and $p \in \mathcal{A}$, then $(\exists p\, \Phi)$ and $(\forall p\, \Phi)$ are QBFs;

  (4) QBFs are solely given by $(1) - (3)$.

- Furthermore, we define
  - an occurrence of an atom p in QBF Φ as *bound* in Φ if it is in a subformula Q$p\Psi$ of Φ;
  - an occurrence of atom p as *free* in Φ iff it is not bound in Φ;
  - a QBF Φ as *closed*, if each atom occurrence is bound in Φ.

QPL—Syntax (ctd.)

- A sequence of quantifiers $\mathsf{Q}p_1 \ldots \mathsf{Q}p_n$ with $A = \{p_1, \ldots, p_n\}$, is abbreviated by $\mathsf{Q}A$.
- Let Φ be a QBF, p an atom, and ϕ a propositional formula, then $\Phi[p/\phi]$ denotes the QBF resulting from ϕ by replacing each *free* occurrence of p in Φ by ϕ.
- A QBF is in *prenex normal form* (PNF) iff it is of the form

$$\mathsf{Q}_1 A_1 \ldots \mathsf{Q}_n A_n \phi,$$

  where
  - ϕ is propositional formula
  - the sets A_i are pairwise disjoint;
  - $\mathsf{Q}_i \neq \mathsf{Q}_{i+1}$, for each $1 \leq i < n$.
- Unless stated otherwise PNF-QBFs are considered to be closed.
- A QBF as above is called (n, Q_1)-QBF.

QPL—Semantics

- As in propositional logic, we consider interpretations $m : \mathcal{A} \to \{0, 1\}$ (or $I \subseteq \mathcal{A}$) and define the truth-value of a QBF Φ, $V^m(\Phi)$ under m as:
  - $V^m(\exists p \Psi) = 1$, if $V^m(\Psi[p/\top]) = 1$ or $V^m(\Psi[p/\bot]) = 1$;
  - $V^m(\forall p \Psi) = 1$, if $V^m(\Psi[p/\top]) = 1$ and $V^m(\Psi[p/\bot]) = 1$;
  - all other cases are as in propositional logic.
- We use the termini *true*, *false*, *satisfiable*, *model*, etc. as in propositional logic.
- Note: Closed QBFs are either true (under any interpretation) or false (under any interpretation).

QPL—Semantics (ctd.)

- For each QBF Φ, we can construct a logically equivalent QBF in PNF by the following rewritings:

$$\begin{aligned}
\mathsf{Q}q\Psi &\Rightarrow \mathsf{Q}p\Psi[q/p] \\
\mathsf{Q}p\Psi &\Rightarrow \Psi \\
\neg\exists p\Phi &\Rightarrow \forall p\neg\Phi \\
\neg\forall p\Phi &\Rightarrow \exists p\neg\Phi \\
(\mathsf{Q}p\Phi)\circ\Psi &\Rightarrow \mathsf{Q}p(\Phi\circ\Psi) \\
\Psi\circ(\mathsf{Q}p\Phi) &\Rightarrow \mathsf{Q}p(\Psi\circ\Phi).
\end{aligned}$$

  where $\circ \in \{\wedge, \vee\}$, and p does not occur free in Ψ.

- Such a QBF in PNF can be obtained from any QBF in polynomial time.

Complexity—Introduction

- *Complexity theory* studies the difficulty of problems; difficulty is measured relative to some resources, usually time or space.

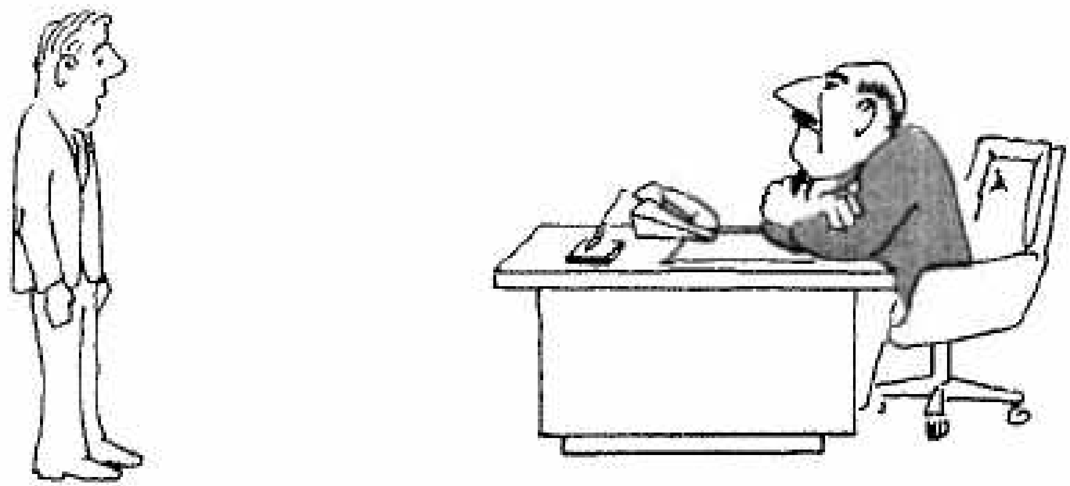
- Problems are located in particular *complexity classes*.

- One line of research studies properties of and relations between such classes.

- *Complexity analysis* addresses the classification of problems.
  - Having classified a problem, one gets numerous properties of that problem.

- Basic distinction: *tractable* (*feasible*) problems vs. *untractable* (*infeasible*) problems.

Complexity—Introduction (ctd.)



"I can't find an efficient algorithm, I guess I'm just too dumb."

Complexity—Introduction (ctd.)

"I can't find an efficient algorithm, because no such algorithm is possible!"

Complexity—Introduction (ctd.)

"I can't find an efficient algorithm, but neither can all these famous people."

Garey and Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

Complexity—Basic Concepts

- **Problem Description**: A language L and subset Y of (positive) instances of L.
- **Decision Problem**: Given instance $I \in L$. Does $I \in Y$ hold?

  Example: SAT (*Satisfiability*):

  Given.: Propositional formula A.

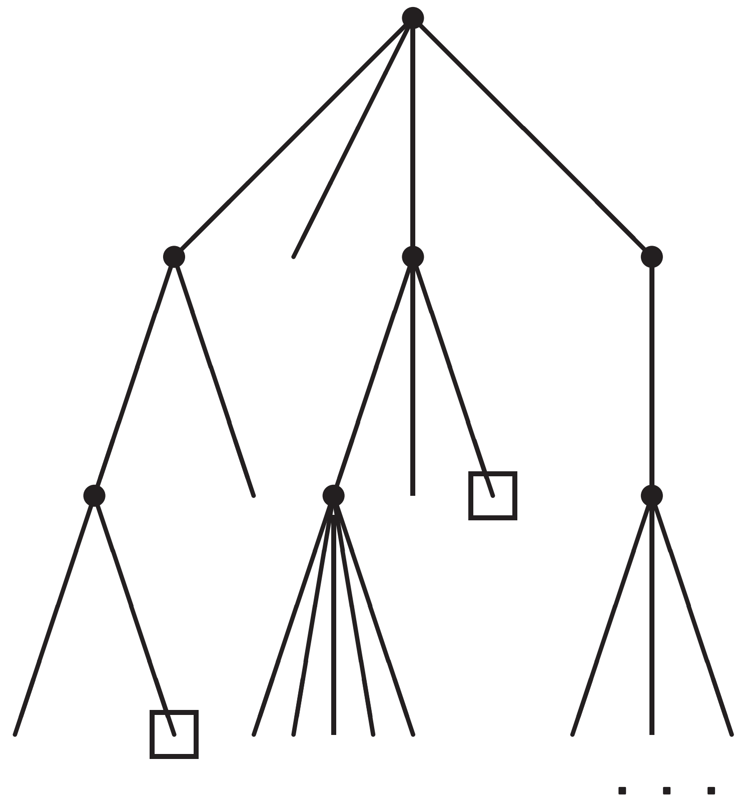
  Question.: Is A satisfiable?
- **Representation** has to be ”adequate”:
  - not too simple: e.g., unary representation of numbers;
  - not too complicated: representation must not be “encoded”.

Complexity—Basic Concepts (ctd.)

- Classical formalization of computation: *Turing Machine* (TM).
- A deterministic TM (DTM) M consists of a
  - a finite set S of states; with a designated start state and accepting states;
  - the transition function $\delta : S \times \Sigma \to S \times \Sigma \times \{l, r\}$.
- Intuitively, the input (a word over Σ) is written onto an infinite tape; a move of a TM consists of reading the current tape symbol, overwriting it, moving the tape head left or right, and changing state.
- If M reaches an accepting state, the input is accepted, otherwise it is rejected. $L(M)$ is the language of words accepted by M.
- *Nondeterministic TM* (NTM): δ maps $S \times \Sigma$ to $2^S \times \Sigma \times \{l, r\}$.
- A word is accepted by a NTM M if there is at least one computation ending in an accepting state.

Complexity—Basic Concepts (ctd.)

- nondeterministic computation-tree:



□ ... accept

Complexity Classes

- Informal definition of important classes:

class	model of computation	expense wrt resource
P	deterministic	polynomial time
NP	non deterministic	polynomial time
PSPACE	deterministic	polynomial space
NPSPACE	non deterministic	polynomial space
EXPTIME	deterministic	exponential time
NEXPTIME	non deterministic	exponential time

Complexity Classes

- Relations between complexity classes:
  - $\mathrm{P} \subseteq_{=?} \mathrm{NP} \subseteq_{=?} \mathrm{PSPACE}$
  - $\mathrm{PSPACE} = \mathrm{NPSPACE}$
  - $\mathrm{PSPACE} \subseteq_{=?} \mathrm{EXPTIME}$
  - $\mathrm{P} \subset \mathrm{EXPTIME}$
  - $\mathrm{NP} \subset \mathrm{NEXPTIME}$

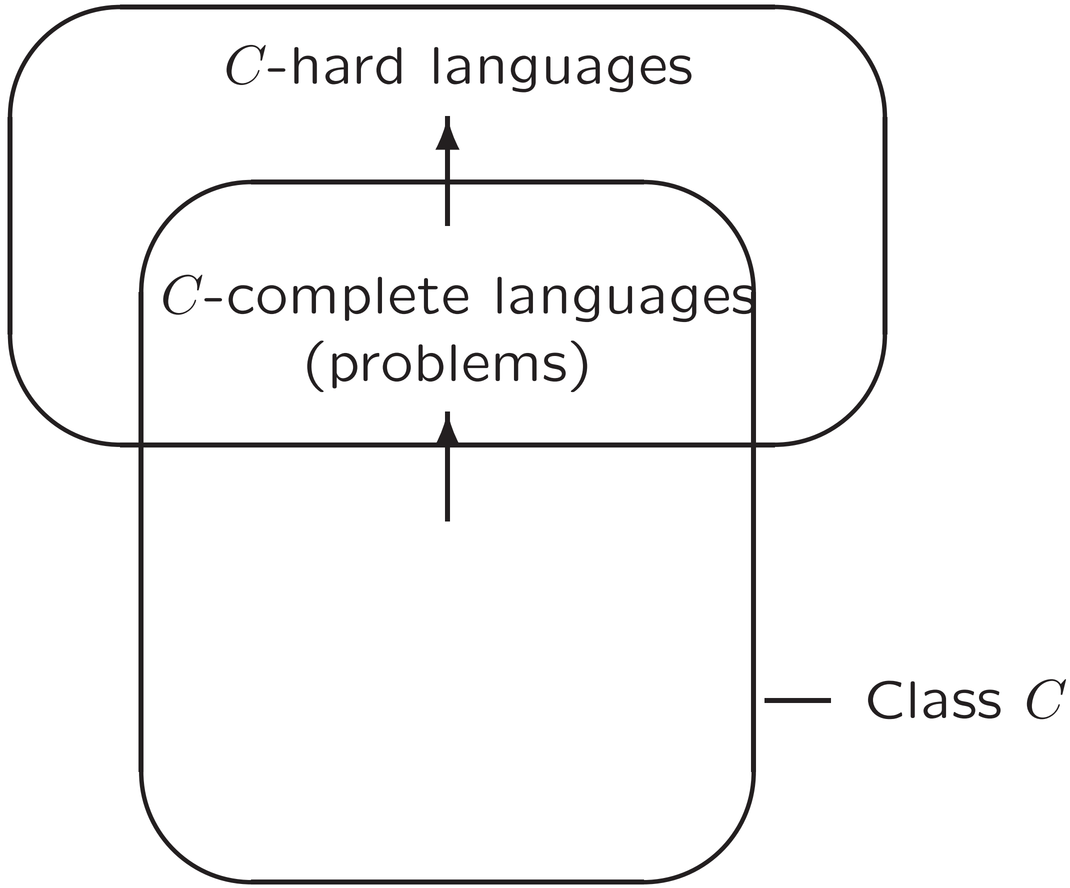
Complexity Classes (ctd.)

- Given $K \subseteq L$, define its complement as $\overline{K} = L \setminus K$.
  - Example (for L propositional logic): UNSAT $= \overline{\text{SAT}}$, i.e., the set of unsatisfiable formulas.
- Given complexity class C, then $\text{co-}C = \{\bar{A} \mid A \in C\}$.
- Det. classes are closed under complement (but this is unclear for nondet. classes).

Complexity—Completeness

- **Reduction:** Given two languages L, K. Language L is *reducible* to K iff there is a computable mapping f, such that, for each w, $w \in L$ iff $f(w) \in K$.
- To compare languages properly, it is sufficient in our context to consider reductions which are computable in polynomial time.
- We write $L \leq_P K$ to denote that L is *polynomially reducible* to K.
  - $L \leq_P K$ means, that deciding L is not harder than deciding K. An algorithm solving K solves L modulo an (ignorable) translation overhead.
- **Definition:** A problem K is
  - *hard* for a class C, if for each $L \in C$, $L \leq_P K$ holds.
  - *C-complete* iff $K \in C$ and K is C-hard.

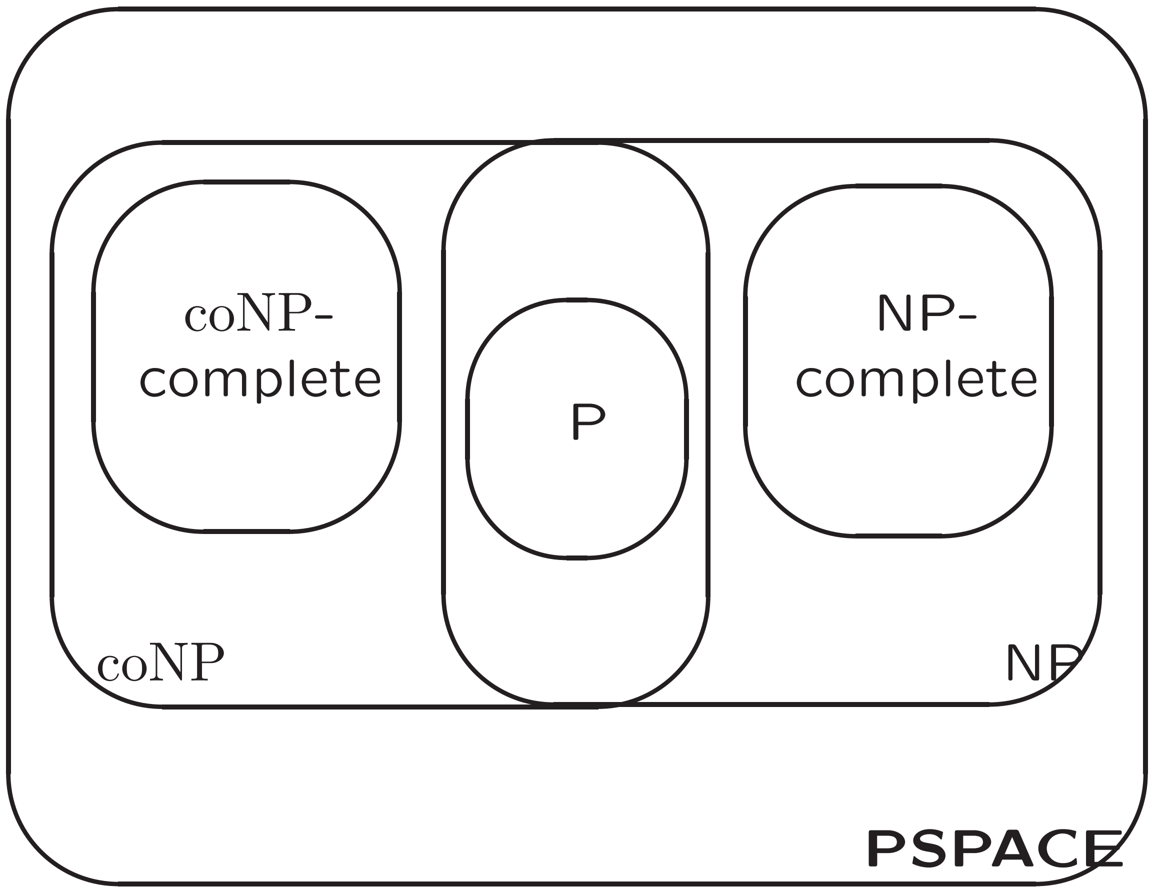
Complexity—Completeness (ctd.)



Complexity—Completeness (ctd.)

- We have the following important properties:
  - If L is C-hard and $L \leq_P K$, then K is also C-hard.
  - If L is C-complete, $K \in C$ and $L \leq_P K$, then K is C-complete.
  - If L is C-hard, then $\overline{L}$ is co-C-hard.
  - If L is C-complete, then $\overline{L}$ is co-C-complete.
- Strategy to show C-completeness for a language L:
  1. show $L \in C$;
  2. show $K \leq_P L$ for a C-complete problem K.

Complexity Classes (ctd.)



(Assuming $\mathrm{P} \neq \mathrm{NP}$ and $\mathrm{NP} \neq \mathrm{coNP}$)

Complexity Classes (ctd.)

- Equivalent model for nondeterministic computation: *Guess & Check*.
  - for SAT: "Guess" an interpretation I; check whether I is model of the given formula.
  - ➥ SAT is in NP.
- NP-completeness of SAT (Cook, 1971): Encode the computation of any NTM M on input w in t steps as a prop. formula ϕ (which is obtained from M, w, t in polynomial time), such that ϕ is satisfiable iff M holds in less than t steps on input w.
- UNSAT is coNP-complete.
- HORNSAT is P-complete.

The Polynomial Hierarchy

- Computation with oracles: special move of a TM which amounts to a call of a subprocedure, but without counting the resources needed by the subprocedure.

- Given class C; P^C is then the class of languages, recognized by DTMs with the help of oracles for problems in C in polynomial time.

- Analogous definition for NP^C.

- Remark: "Complementary oracles" do not make any difference; we have e.g., $\mathrm{P}^C = \mathrm{P}^{\mathrm{co}-C}$.

The Polynomial Hierarchy (ctd.)

- Oracle-classes can be defined in a recursive way:
- The *polynomial hierarchy* consists of classes Σ_k^P, Π_k^P, and Δ_k^P, where

$$\Sigma_0^P = \Pi_0^P = \Delta_0^P = \mathrm{P};$$

and for $k \geq 1$:

$$\begin{aligned} \Delta_{k+1}^P &= \mathrm{P}^{\Sigma_k^P}; \\ \Sigma_{k+1}^P &= \mathrm{NP}^{\Sigma_k^P}; \\ \Pi_{k+1}^P &= \mathrm{co} - \Sigma_{k+1}^P. \end{aligned}$$

The Polynomial Hierarchy (ctd.)

- In particular, we get:

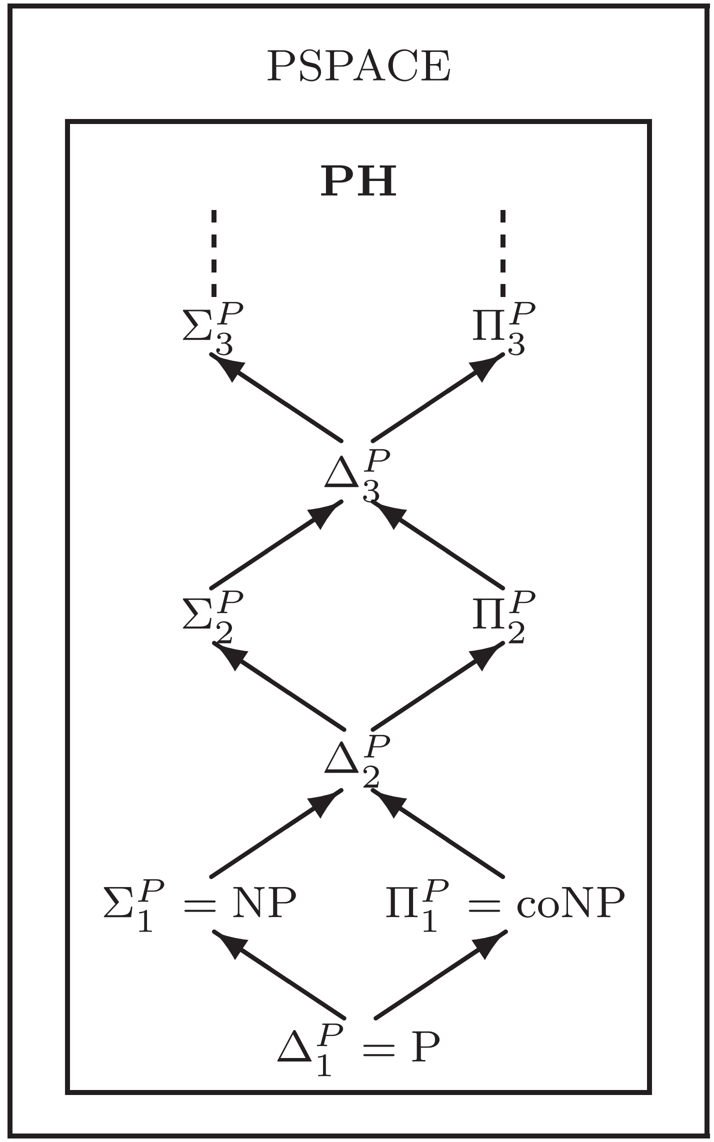
$$\begin{array}{rclrcl}\Delta_1^P & = & \mathrm{P}; & \Delta_2^P & = & \mathrm{P}^{\mathrm{NP}};\\ \Sigma_1^P & = & \mathrm{NP}; & \Sigma_2^P & = & \mathrm{NP}^{\mathrm{NP}};\\ \Pi_1^P & = & \mathrm{coNP}; & \Pi_2^P & = & \mathrm{coNP}^{\mathrm{NP}}.\end{array}$$

- Relations:

$$\Delta_k^P \subseteq \left(\Sigma_k^P \cap \Pi_k^P\right); \quad \left(\Sigma_k^P \cup \Pi_k^P\right) \subseteq \Delta_{k+1}^P;$$

$$\bigcup_{k=0}^{\infty} \Sigma_k^P \subseteq \mathrm{PSPACE}.$$

The Polynomial Hierarchy (ctd.)



The Polynomial Hierarchy (ctd.)

- Problem QSAT:

  Given: Closed QBF Φ;

  Quest.: Is Φ true?

  is PSPACE-complete.

- Problem $(k, \exists)$-QSAT:

  Given.: $(k, \exists)$-QBF Φ;

  Quest.: Is Φ true?

  is Σ_k^P-complete.

- Problem $(k, \forall)$-QSAT:

  Given.: $(k, \forall)$-QBF Φ;

  Quest.: Is Φ true?

  is Π_k^P-complete.

Exercises:

- Construct a function $\mathcal{S}$ which maps every pair (ϕ, ψ) of propositional formulas over atoms V, into a closed QBF $\mathcal{S}(\phi, \psi)$ over V, such that $\mathcal{S}(\phi, \psi)$ is true iff ϕ is *satisfiable* and ψ is *unsatisfiable*.

  In a second step try to give $\mathcal{S}(\phi, \psi)$ in PNF. What are your observations?

- Construct a function $\mathcal{T}$ mapping every propositional formula ϕ over atoms V to an open QBF $\mathcal{T}(\phi)$ over $V \cup V'$ (with atoms V being free), such that the models of the QBF $\mathcal{T}(\phi)$ are exactly the *maximal* models of ϕ.