Complexity Theory
VU 181.142, SS 2015

8. PSPACE

Reinhard Pichler

Institut für Informationssysteme
Arbeitsbereich DBAI
Technische Universität Wien

16 June, 2015
Outline

8. PSPACE
8.1 QSAT (QBF)
8.2 Complexity of Query Evaluation
PSPACE

Motivation

PSPACE captures unrestricted alternation. Therefore, ...

- it generalizes the polynomial hierarchy,
- it is the class of many strategy games, decision making, etc.,
- it has QSAT (QBF) as natural complete problem.
QSAT (QBF)

INSTANCE: Boolean expression $\varphi$ in CNF with variables $x_1, \ldots, x_n$.

QUESTION: Is there a truth value for the variable $x_1$ such that for both truth values of $x_2$ there is a truth value for $x_3$ and so on up to $x_n$, such that $\varphi$ is satisfied by the overall truth assignment?
QSAT (QBF)

INSTANCE: Boolean expression \( \varphi \) in CNF with variables \( x_1, \ldots, x_n \).

QUESTION: Is there a truth value for the variable \( x_1 \) such that for both truth values of \( x_2 \) there is a truth value for \( x_3 \) and so on up to \( x_n \), such that \( \varphi \) is satisfied by the overall truth assignment?

Notation

An instance of QSAT is written as \( \exists x_1 \forall x_2 \exists x_3 \cdots Q x_n \varphi \), where \( Q \) is \( \forall \) if \( n \) is even and \( \exists \) if \( n \) is odd.
**QSAT (QBF)**

**INSTANCE:** Boolean expression \( \varphi \) in CNF with variables \( x_1, \ldots, x_n \).

**QUESTION:** Is there a truth value for the variable \( x_1 \) such that for both truth values of \( x_2 \) there is a truth value for \( x_3 \) and so on up to \( x_n \), such that \( \varphi \) is satisfied by the overall truth assignment?

**Notation**

An instance of QSAT is written as \( \exists x_1 \forall x_2 \exists x_3 \cdots Qx_n \varphi \), where \( Q \) is \( \forall \) if \( n \) is even and \( \exists \) if \( n \) is odd.

**Theorem**

QSAT is PSPACE-complete.
Proof of the PSPACE-Membership of QSAT

**Remark.** We only prove the PSPACE-membership here. The hardness will be proved below via the complexity of First-Order Logic.

Let an arbitrary QBF be given as $\psi \equiv \exists x_1 \forall x_2 \exists x_3 \cdots Q x_n \varphi$. All possible truth assignments of the variables can be represented by the leaves in a full binary tree of depth $n$ (= “semantic tree”):
Proof of the PSPACE-Membership of QSAT

Remark. We only prove the PSPACE-membership here. The hardness will be proved below via the complexity of First-Order Logic.

Let an arbitrary QBF be given as $\psi \equiv \exists x_1 \forall x_2 \exists x_3 \cdots Q x_n \varphi$. All possible truth assignments of the variables can be represented by the leaves in a full binary tree of depth $n$ (= “semantic tree”):

The left subtree of the root contains all truth assignments $T$ with $T(x_1) = \text{false}$, while the right subtree of the root contains all truth assignments $T$ with $T(x_1) = \text{true}$. 
Proof of the PSPACE-Membership of QSAT

Remark. We only prove the PSPACE-membership here. The hardness will be proved below via the complexity of First-Order Logic.

Let an arbitrary QBF be given as $\psi \equiv \exists x_1 \forall x_2 \exists x_3 \cdots Q x_n \varphi$. All possible truth assignments of the variables can be represented by the leaves in a full binary tree of depth $n$ (= “semantic tree”):

The left subtree of the root contains all truth assignments $T$ with $T(x_1) = \text{false}$, while the right subtree of the root contains all truth assignments $T$ with $T(x_1) = \text{true}$.

Analogously, for every $i \geq 1$, the subtrees at depth $i + 1$, whose root is the first child of its parent, contains all truth assignments $T$ with $T(x_{i+1}) = \text{false}$, while the subtrees at depth $i + 1$, whose root is the second child of its parent, contains all truth assignments $T$ with $T(x_{i+1}) = \text{true}$.
Proof of the PSPACE-Membership of QSAT (continued)

We can now turn this tree into a monotone Boolean circuit $C$ where all gates at the $i$-th level are
– OR-gates if $i$ is even (in particular, the root node at level 0) and
– AND-gates if $i$ is odd.
Proof of the PSPACE-Membership of QSAT (continued)

We can now turn this tree into a monotone Boolean circuit $C$ where all gates at the $i$-th level are
– OR-gates if $i$ is even (in particular, the root node at level 0) and
– AND-gates if $i$ is odd.

An input gate (i.e., a leaf node) is true if $\varphi$ evaluates to true in the truth assignment corresponding to this leaf node; and an input gate is false if $\varphi$ evaluates to false in this truth assignment.

Clearly, the QBF $\psi$ is true $\iff$ the Boolean circuit $C$ has the value true.
Proof of the PSPACE-Membership of QSAT (continued)

We can now turn this tree into a monotone Boolean circuit $C$ where all gates at the $i$-th level are
– OR-gates if $i$ is even (in particular, the root node at level 0) and
– AND-gates if $i$ is odd.

An input gate (i.e., a leaf node) is true if $\varphi$ evaluates to true in the truth assignment corresponding to this leaf node; and an input gate is false if $\varphi$ evaluates to false in this truth assignment.

Clearly, the QBF $\psi$ is true $\iff$ the Boolean circuit $C$ has the value true.

Notation. It will turn out to be convenient in the sequel to use strings over $\{0, 1\}$ (rather than natural numbers) as labels of the gates, i.e.: The output gate (= the root node) has as label the empty string $\epsilon$. Now suppose that some internal node $N$ has label $w \in \{0, 1\}^*$. Then the first child of $N$ has label $w0$ and the second child has label $w1.$
Proof of the PSPACE-Membership of QSAT (continued)

The Boolean circuit can be evaluated in space $O(n)$ by an algorithm which traverses the (tree-structured) circuit as follows:

- In order to evaluate an **AND-gate** $g$, we recursively evaluate its first child $g_0$. If $g_0$ is **false**, we know that $g$ is **false**. Otherwise, the evaluation continues with the second child $g_1$ of $g$.

- **OR-gates** are treated analogously – with **true** and **false** reversed.

- The evaluation of a **NOT-gate** is clear (namely by recursively evaluating its unique child and returning the opposite truth value) but not needed for the monotone Boolean circuit $C$.

- Once the evaluation of a gate $g$ is finished, the algorithm continues with the parent node of $g$ (whose label is obtained by simply omitting the last bit of $g$’s label).
Proof of the PSPACE-Membership of QSAT (continued)

The linear space bound on the evaluation of the Boolean circuit follows immediately from the following observation: At any time, the algorithm only needs to store (the label of) exactly 1 gate of the tree, namely the current gate $g$ of the evaluation.
Proof of the PSPACE-Membership of QSAT (continued)

The linear space bound on the evaluation of the Boolean circuit follows immediately from the following observation: At any time, the algorithm only needs to store (the label of) exactly 1 gate of the tree, namely the current gate \( g \) of the evaluation.

Implicitly, we thus have the entire path from \( g \) to the root. If the path contains a gate which is the first child of its parent \( h \), then it is clear that the second child of \( h \) has not been visited yet. If the path contains a gate which is the second child of \( h \), then it is clear that the value of the first child of \( h \) is \textbf{true} for an AND-gate \( h \) and \textbf{false} for an OR-gate \( h \).

The only difficulty remaining is that the circuit \( C \) has exponential size. Observe that both, the construction of \( C \) and the evaluation of \( C \) work in polynomial space. Hence, the combination of these two algorithms is feasible in PSPACE – by the same idea as in the proof that the composition of two log-space computations is feasible in log-space.
Proof of the PSPACE-Membership of QSAT (continued)

The linear space bound on the evaluation of the Boolean circuit follows immediately from the following observation: At any time, the algorithm only needs to store \text{(the label of)} exactly 1 gate of the tree, namely the current gate $g$ of the evaluation.

Implicitly, we thus have the entire path from $g$ to the root. If the path contains a gate which is the first child of its parent $h$, then it is clear that the second child of $h$ has not been visited yet. If the path contains a gate which is the second child of $h$, then it is clear that the value of the first child of $h$ is \textbf{true} for an AND-gate $h$ and \textbf{false} for an OR-gate $h$.

The only difficulty remaining is that the circuit $C$ has exponential size. Observe that both, the construction of $C$ and the evaluation of $C$ work in polynomial space. Hence, the combination of these two algorithms is feasible in PSPACE – by the same idea as in the proof that the composition of two log-space computations is feasible in log-space.
PSPACE vs. PH

**Proposition**

QSAT is a generalization of the $\Sigma_i P$-complete problem QSAT$_i$ for any value of $i$.

**Corollary**

$\mathsf{PH} \subseteq \mathsf{PSPACE}$
### Proposition

QSAT is a generalization of the $\Sigma_i \text{P}$-complete problem $\text{QSAT}_i$ for any value of $i$.

### Corollary

$\text{PH} \subseteq \text{PSPACE}$

### Remark

It is not known if PH is properly included in PSPACE. Most probably, $\text{PH} \subset \text{PSPACE}$ holds, because $\text{PH} = \text{PSPACE}$ would imply that the polynomial hierarchy collapses (since there exist PSPACE-complete problems).
Games

Observation

PSPACE is the class of many strategy games, decision making, etc.

QSAT can be considered as a two-person game:

- two players: $\exists$ and $\forall$
- players move alternatingly ($\exists$ first)
- a move: determining the truth value of a variable
- $\exists$ tries to make the formula $\varphi$ true while $\forall$ tries to make it false.
- after $n$ moves either $\exists$ or $\forall$ wins.

Decision making can sometimes be considered as a game against nature.
# Complexity of Query Evaluation

## Decision Problems

For (Boolean) queries of a certain query language (e.g., SQL, datalog, XPath, XQuery, etc.), there are three main kinds of decision problems:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data complexity</td>
<td>INSTANCE: An input database $D$. QUESTION: Does query $Q$ yield a non-empty result over $D$?</td>
</tr>
<tr>
<td>Query complexity</td>
<td>INSTANCE: A query $Q$. QUESTION: Does query $Q$ yield a non-empty result over $D$?</td>
</tr>
<tr>
<td>Combined complexity</td>
<td>INSTANCE: An input database $D$ and a query $Q$. QUESTION: Does query $Q$ yield a non-empty result over $D$?</td>
</tr>
</tbody>
</table>
Complexity of Query Evaluation

Decision Problems

For (Boolean) queries of a certain query language (e.g., SQL, datalog, XPath, XQuery, etc.), there are three main kinds of decision problems:

Data complexity refers to the following decision problem:
Let $Q$ be some *fixed* query.
INSTANCE: An input database $D$.
QUESTION: Does query $Q$ yield a non-empty result over the DB $D$?
Complexity of Query Evaluation

Decision Problems

For (Boolean) queries of a certain query language (e.g., SQL, datalog, XPath, XQuery, etc.), there are three main kinds of decision problems:

Data complexity refers to the following decision problem:
Let $Q$ be some fixed query.
INSTANCE: An input database $D$.
QUESTION: Does query $Q$ yield a non-empty result over the DB $D$?

Query complexity refers to the following decision problem:
Let $D$ be some fixed input database.
INSTANCE: A query $Q$.
QUESTION: Does query $Q$ yield a non-empty result over the DB $D$?
## Complexity of Query Evaluation

### Decision Problems

For (Boolean) queries of a certain query language (e.g., SQL, datalog, XPath, XQuery, etc.), there are three main kinds of decision problems:

**Data complexity** refers to the following decision problem:
- Let \( Q \) be some fixed query.
- **INSTANCE:** An input database \( D \).
- **QUESTION:** Does query \( Q \) yield a non-empty result over the DB \( D \)?

**Query complexity** refers to the following decision problem:
- Let \( D \) be some fixed input database.
- **INSTANCE:** A query \( Q \).
- **QUESTION:** Does query \( Q \) yield a non-empty result over the DB \( D \)?

**Combined complexity** refers to the following decision problem:
- **INSTANCE:** An input database \( D \) and a query \( Q \).
- **QUESTION:** Does query \( Q \) yield a non-empty result over the DB \( D \)?
First-Order Queries

Definition

A term is a constant or a variable.

For a given input schema \( \mathcal{R} = \{R_1, \ldots, R_n\} \), the base formulae are either equality atoms \( s = t \) or atoms of the form \( R(t_1, \ldots, t_\alpha) \), where the \( t_i \) are terms and \( \alpha \) is the arity of \( R \). A first-order query over \( \mathcal{R} \) is either a base formula or a formula of the following form:

1. \((\phi \land \psi)\), where \( \phi \) and \( \psi \) are formulae over \( \mathcal{R} \);
2. \((\phi \lor \psi)\), where \( \phi \) and \( \psi \) are formulae over \( \mathcal{R} \);
3. \(\neg\phi\), where \( \phi \) is a formula over \( \mathcal{R} \);
4. \(\exists x \phi\), where \( x \) is a variable and \( \phi \) is a formula over \( \mathcal{R} \);
5. \(\forall x \phi\), where \( x \) is a variable and \( \phi \) is a formula over \( \mathcal{R} \).

Remark. First-order queries essentially correspond to SQL without GROUP BY, (aggregate) functions and arithmetic.
First-Order Queries

Theorem

The query complexity and the combined complexity of first-order queries are PSPACE-complete (even if we disallow negation and equality atoms). The data complexity is in L (actually, even in a lower class).
First-Order Queries

**Theorem**

The query complexity and the combined complexity of first-order queries are PSPACE-complete (even if we disallow negation and equality atoms). The data complexity is in $L$ (actually, even in a lower class).

**Remark**

The decision problem for the query complexity is a special case of the decision problem for the combined complexity. Hence, it suffices to prove the following results:

- The combined complexity of first-order queries is in PSPACE.
- The query complexity of first-order queries is PSPACE-hard.
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness

We prove the hardness by reduction from an arbitrary language $L$ in PSPACE. To this end, we define a fixed database $D$. Moreover, we describe a reduction $R$ which, for every string $w$, constructs a First-Order sentence $R(w)$ such that $w \in L \iff R(w)$ evaluates to true over $D$. Let $T = (K, \Sigma, \delta, s)$ be a single-string Turing machine that decides $L$ in polynomial space. W.l.o.g., we assume that on any positive instance $w$, the TM $T$ has exactly one accepting configuration, say $(\text{"yes"}, \emptyset, \ldots)$. Assume that the computation on input $w$ requires at most $d \cdot n^k$ space with $n = |w|$ and constants $d$, $k$. Then the computation takes at most $N = c d \cdot n^k$ steps for some constant $c$. We first define the (fixed) input database $D$: it just contains two unary relations $K$ and $\Sigma$ with the states and symbols, respectively, of $T$. 

Reinhard Pichler
16 June, 2015
**PSPACE-Hardness of First-Order Queries**

### Proof of the PSPACE-Hardness

We prove the hardness by reduction from an arbitrary language $L$ in PSPACE. To this end, we define a fixed database $D$. Moreover, we describe a reduction $R$ which, for every string $w$, constructs a First-Order sentence $R(w)$ such that $w \in L \iff R(w)$ evaluates to true over $D$.

Let $T = (K, \Sigma, \delta, s)$ be a single-string Turing machine that decides $L$ in polynomial space. W.l.o.g., we assume that on any positive instance $w$, the TM $T$ has exactly one accepting configuration, say ("yes", $\triangleright$, $\square$ $\square$ ...). Assume that the computation on input $w$ requires at most $d \cdot n^k$ space with $n = |w|$ and constants $d, k$. Then the computation takes at most $N = c^{d \cdot n^k}$ steps for some constant $c$. 
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness

We prove the hardness by reduction from an arbitrary language $L$ in PSPACE. To this end, we define a fixed database $D$. Moreover, we describe a reduction $R$ which, for every string $w$, constructs a First-Order sentence $R(w)$ such that $w \in L \iff R(w)$ evaluates to true over $D$.

Let $T = (K, \Sigma, \delta, s)$ be a single-string Turing machine that decides $L$ in polynomial space. W.l.o.g., we assume that on any positive instance $w$, the TM $T$ has exactly one accepting configuration, say ("yes", $\triangleright$, $\square \square \ldots$). Assume that the computation on input $w$ requires at most $d \cdot n^k$ space with $n = |w|$ and constants $d, k$. Then the computation takes at most $N = c^{d \cdot n^k}$ steps for some constant $c$.

We first define the (fixed) input database $D$: it just contains two unary relations $K$ and $\Sigma$ with the states and symbols, respectively, of $T$. 
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness (continued)

Now let $w$ be an arbitrary instance of $L$. We have to construct an FO formula $R(w)$. This construction is based on well-known ideas.
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness (continued)

Now let $w$ be an arbitrary instance of $L$. We have to construct an FO formula $R(w)$. This construction is based on well-known ideas.

Idea 1. Recall the NL-completeness proof of REACHABILITY. Our PSPACE-hardness proof also makes use of the configuration graph $G(T, w)$ of TM $T$ on input $w$: The nodes are all possible configurations of $T$ (with space bound $d \cdot n^k$). There is an edge between two nodes (i.e., two configurations) $C_1$ and $C_2$ iff the TM $T$ has a transition in one step from $C_1$ to $C_2$. We have $w \in L$ iff there exists a path from the unique initial configuration $(s, \triangleright, w)$ to (“yes”, $\triangleright, \square \square \ldots$).
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness (continued)

Now let \( w \) be an arbitrary instance of \( L \). We have to construct an FO formula \( R(w) \). This construction is based on well-known ideas.

**Idea 1.** Recall the NL-completeness proof of REACHABILITY. Our PSPACE-hardness proof also makes use of the configuration graph \( G(T, w) \) of TM \( T \) on input \( w \): The nodes are all possible configurations of \( T \) (with space bound \( d \cdot n^k \)). There is an edge between two nodes (i.e., two configurations) \( C_1 \) and \( C_2 \) iff the TM \( T \) has a transition in one step from \( C_1 \) to \( C_2 \). We have \( w \in L \) iff there exists a path from the unique initial configuration \((s, \triangledown, w)\) to \( ("yes", \triangledown, \sqcup \sqcup \ldots) \).

**Idea 2.** Recall the proof of Savitch’s Theorem, where we search for a path between two nodes via middle-first search. The crucial idea of this proof was to define a predicate \( PATH(a, b, i) \) with the intended meaning that \( PATH(a, b, i) \) is true iff there is a path from \( a \) to \( b \) of length at most \( 2^i \). The main task of our PSPACE-hardness proof will be to encode predicates \( PATH(a, b, i) \) for \( i \in \{0, \ldots, \log N\} \) as FO formulas.
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness (continued)

**Configurations.** Every configuration can be represented by a vector of length $M = d \cdot n^k + 1$: we represent $(q, u, v)$ with $u = u_1, \ldots, u_\alpha$ and $v = v_1, \ldots, v_\beta$ as $(u_1, \ldots, u_\alpha, q, v_1, \ldots, v_\beta, \sqcap, \sqcap, \ldots)$. 

Reduction from $L$ to FO evaluation. Suppose that we have defined the predicates $\psi_i(x_1, \ldots, x_M, y_1, \ldots, y_M)$. Let $j = \log N$. Moreover, let $(a_1, \ldots, a_M)$ be the (representation of the) initial configuration $C_0$ on input $w$ and let $(b_1, \ldots, b_M)$ be the accepting configuration $C_{\text{"yes}}$. We define $\psi^* = \psi_j(a_1, \ldots, a_M, b_1, \ldots, b_M)$. Then we have $x \in L \iff \psi^*$ is true over $D$. 

Reinhard Pichler 16 June, 2015
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness (continued)

**Configurations.** Every configuration can be represented by a vector of length $M = d \cdot n^k + 1$: we represent $(q, u, v)$ with $u = u_1, \ldots, u_\alpha$ and $v = v_1, \ldots, v_\beta$ as $(u_1, \ldots, u_\alpha, q, v_1, \ldots, v_\beta, \sqcup, \sqcup, \ldots)$.

**Encoding of $\text{PATH}(a, b, i)$.** For every $i \in \{0, \ldots, \log N\}$ we define a formula $\psi_i(x_1, \ldots, x_M, y_1, \ldots, y_M)$ with free variables $x_1, \ldots, x_M, y_1, \ldots, y_M$, s.t. $\psi_i$ is true in $D$ iff $(x_1, \ldots, x_M)$ is instantiated to (the representation of) some configuration $C_1$, $(y_1, \ldots, y_M)$ is instantiated to (the representation of) some configuration $C_2$, and there is a path of length at most $2^i$ from $C_1$ to $C_2$ in the configuration graph $G(T, \ell)$. 
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness (continued)

Configurations. Every configuration can be represented by a vector of length \( M = d \cdot n^k + 1 \): we represent \((q, u, v)\) with \( u = u_1, \ldots, u_\alpha \) and \( v = v_1, \ldots, v_\beta \) as \((u_1, \ldots, u_\alpha, q, v_1, \ldots, v_\beta, \sqcup, \sqcup, \ldots)\).

Encoding of \( \text{PATH}(a, b, i) \). For every \( i \in \{0, \ldots, \log N\} \) we define a formula \( \psi_i(x_1, \ldots, x_M, y_1, \ldots, y_M) \) with free variables \( x_1, \ldots, x_M, y_1, \ldots, y_M \), s.t. \( \psi_i \) is true in \( D \) iff \((x_1, \ldots, x_M)\) is instantiated to (the representation of) some configuration \( C_1 \), \((y_1, \ldots, y_M)\) is instantiated to (the representation of) some configuration \( C_2 \), and there is a path of length at most \( 2^i \) from \( C_1 \) to \( C_2 \) in the configuration graph \( G(T, w) \).

Reduction from \( L \) to \( \text{FO} \) evaluation. Suppose that we have defined the predicates \( \psi_i(x_1, \ldots, x_M, y_1, \ldots, y_M) \). Let \( j = \log N \). Moreover, let \((a_1, \ldots, a_M)\) be the (representation of the) initial configuration \( C_0 \) on input \( w \) and let \((b_1, \ldots, b_M)\) be the accepting configuration \( C_{\text{"yes"}} \).

We define \( \psi^* = \psi_j(a_1, \ldots, a_M, b_1, \ldots, b_M) \).
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness (continued)

Configurations. Every configuration can be represented by a vector of length $M = d \cdot n^k + 1$: we represent $(q, u, v)$ with $u = u_1, \ldots, u_\alpha$ and $v = v_1, \ldots, v_\beta$ as $(u_1, \ldots, u_\alpha, q, v_1, \ldots, v_\beta, \sqcup, \sqcup, \ldots)$.

Encoding of $\text{PATH}(a, b, i)$. For every $i \in \{0, \ldots, \log N\}$ we define a formula $\psi_i(x_1, \ldots, x_M, y_1, \ldots, y_M)$ with free variables $x_1, \ldots, x_M$, $y_1, \ldots, y_M$, s.t. $\psi_i$ is true in $D$ iff $(x_1, \ldots, x_M)$ is instantiated to (the representation of) some configuration $C_1$, $(y_1, \ldots, y_M)$ is instantiated to (the representation of) some configuration $C_2$, and there is a path of length at most $2^i$ from $C_1$ to $C_2$ in the configuration graph $G(T, w)$.

Reduction from $L$ to FO evaluation. Suppose that we have defined the predicates $\psi_i(x_1, \ldots, x_M, y_1, \ldots, y_M)$. Let $j = \log N$. Moreover, let $(a_1, \ldots, a_M)$ be the (representation of the) initial configuration $C_0$ on input $w$ and let $(b_1, \ldots, b_M)$ be the accepting configuration $C_{\text{"yes"}}$. We define $\psi^* = \psi_j(a_1, \ldots, a_M, b_1, \ldots, b_M)$.

Then we have $x \in L \iff \psi^*$ is true over $D$. 
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness (continued)

**Base Case.** \( \psi_0(x_1, \ldots, x_M, y_1, \ldots, y_M) \) is defined as a big quantifier-free formula in DNF where each disjunct represents a valid combination of values for \((x_1, \ldots, x_M)\) and \((y_1, \ldots, y_M)\), i.e., either they represent the same configuration or they correspond to the transition of \( T \) in one step. For every \( \ell \in \{1, \ldots, M - 1\} \), \( \psi_0 \) thus contains disjuncts

\[
D = \Sigma(x_1) \land \cdots \land \Sigma(x_\ell) \land K(x_{\ell+1}) \land \Sigma(x_{\ell+2}) \land \cdots \land \Sigma(x_M) \land \\
\quad x_1 = y_1 \land \cdots \land x_M = y_M.
\]

For each transition \((q, a, q', b, \rightarrow)\) in \( \delta \), \( \psi_0 \) contains the following disjuncts (cursor movements \( \rightarrow \) and \( \leftarrow \) are treated analogously).

\[
D = \Sigma(x_1) \land x_1 = y_1 \land \cdots \land \Sigma(x_{\ell-1}) \land x_{\ell-1} = y_{\ell-1} \land \\
x_\ell = a \land x_{\ell+1} = q \land \Sigma(x_{\ell+2}) \land \\
y_\ell = b \land y_{\ell+1} = x_{\ell+2} \land y_{\ell+2} = q' \land \\
\Sigma(x_{\ell+3}) \land x_{\ell+3} = y_{\ell+3} \land \cdots \land \Sigma(x_M) \land x_M = y_M.
\]
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness (continued)

Notation. We use vector notation \( \vec{z} \) as a short-hand for \( (z_1, \ldots, z_M) \). We also write \( \vec{x} = \vec{y} \) for the conjunction \( x_1 = y_1 \land \cdots \land x_M = y_M \).
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness (continued)

Notation. We use vector notation $\vec{z}$ as a short-hand for $(z_1, \ldots, z_M)$. We also write $\vec{x} = \vec{y}$ for the conjunction $x_1 = y_1 \land \cdots \land x_M = y_M$.

Definition of $\psi_{i+1}$. We define $\psi_{i+1}$ inductively from $\psi_i$. It is tempting to define $\psi_{i+1}(\vec{x}, \vec{y})$ as $\psi_{i+1}(\vec{x}, \vec{y}) := (\exists \vec{z}) \psi_i(\vec{x}, \vec{z}) \land \psi_i(\vec{z}, \vec{y})$. However, this is not allowed since it would produce an exponentially big formula $\psi^*$. Instead, we have to "reuse" the definition of $\psi_i$ as follows.

$\psi_{i+1}(\vec{x}, \vec{y}) := (\exists \vec{z}) (\forall \vec{u}) (\forall \vec{v}) ((\vec{u} = \vec{x} \land \vec{v} = \vec{z}) \lor (\vec{u} = \vec{z} \land \vec{v} = \vec{y}) \rightarrow \psi_i(\vec{u}, \vec{v}))$.

It can be easily verified that this reduction works in polynomial time; actually even logarithmic space suffices. For the correctness of this reduction, we have to prove by induction on $i$ that $\psi_i$ has the intended meaning, i.e., $\psi_i(\vec{a}, \vec{b})$ is true over $D$ $\iff$ there is a path of length at most $2^i$ from configuration $\vec{a}$ to configuration $\vec{b}$ in $G(T,w)$.
PSPACE-Hardness of First-Order Queries

Proof of the PSPACE-Hardness (continued)

Notation. We use vector notation \( \vec{z} \) as a short-hand for \((z_1, \ldots, z_M)\). We also write \( \vec{x} = \vec{y} \) for the conjunction \( x_1 = y_1 \land \cdots \land x_M = y_M \).

Definition of \( \psi_{i+1} \). We define \( \psi_{i+1} \) inductively from \( \psi_i \). It is tempting to define \( \psi_{i+1}(\vec{x}, \vec{y}) \) as \( \psi_{i+1}(\vec{x}, \vec{y}) := (\exists \vec{z})\psi_i(\vec{x}, \vec{z}) \land \psi_i(\vec{z}, \vec{y}) \).

However, this is not allowed since it would produce an exponentially big formula \( \psi^* \). Instead, we have to “reuse” the definition of \( \psi_i \) as follows.

\[
\psi_{i+1}(\vec{x}, \vec{y}) := \\
(\exists \vec{z})(\forall \vec{u})(\forall \vec{v}) \left[ (\vec{u} = \vec{x} \land \vec{v} = \vec{z}) \lor (\vec{u} = \vec{z} \land \vec{v} = \vec{y}) \right] \rightarrow \psi_i(\vec{u}, \vec{v})
\]
## PSPACE-Hardness of First-Order Queries

### Proof of the PSPACE-Hardness (continued)

**Notation.** We use vector notation \( \vec{z} \) as a short-hand for \((z_1, \ldots, z_M)\). We also write \( \vec{x} = \vec{y} \) for the conjunction \( x_1 = y_1 \land \cdots \land x_M = y_M \).

**Definition of \( \psi_{i+1} \).** We define \( \psi_{i+1} \) inductively from \( \psi_i \). It is tempting to define \( \psi_{i+1}(\vec{x}, \vec{y}) \) as:

\[
\psi_{i+1}(\vec{x}, \vec{y}) := (\exists \vec{z}) \psi_i(\vec{x}, \vec{z}) \land \psi_i(\vec{z}, \vec{y}).
\]

However, this is not allowed since it would produce an exponentially big formula \( \psi^* \). Instead, we have to “reuse” the definition of \( \psi_i \) as follows.

\[
\psi_{i+1}(\vec{x}, \vec{y}) := (\exists \vec{z})(\forall \vec{u})(\forall \vec{v}) \left( [(\vec{u} = \vec{x} \land \vec{v} = \vec{z}) \lor (\vec{u} = \vec{z} \land \vec{v} = \vec{y})] \rightarrow \psi_i(\vec{u}, \vec{v}) \right)
\]

It can be easily verified that this reduction works in polynomial time; actually even logarithmic space suffices. For the correctness of this reduction, we have to prove by induction on \( i \) that \( \psi_i \) has the intended meaning, i.e., \( \psi_i(\vec{a}, \vec{b}) \) is true over \( D \iff \) there is a path of length at most \( 2^i \) from configuration \( \vec{a} \) to configuration \( \vec{b} \) in \( G(T, w) \).
PSPACE-Membership of First-Order Queries

Proof of the PSPACE-Membership

Let $D$ be an arbitrary input database and let $\varphi$ be an arbitrary first-order sentence. Moreover, let all constants in $\varphi$ and all elements in $D$ be from the domain $\text{dom}$. We prove the PSPACE-membership by reducing the problem of evaluating $\varphi$ over $D$ to the QSAT problem.

1. Restricting the domain to $\{0, 1\}$. Let $\text{dom} = \{a_1, \ldots, a_n\}$. Then these elements can be encoded by bit-vectors of size $m = \lceil \log(n) \rceil$. Let $\bar{b}_i$ denote the encoding of $a_i$. Then we transform $D$ into $D'$ by replacing any $\alpha$-ary relation $r$ by an $(\alpha \cdot m)$-ary relation $r'$. 
PSPACE-Membership of First-Order Queries

Proof of the PSPACE-Membership (continued)

Every tuple \((a_{i1}, \ldots, a_{i\alpha})\) in \(r\) is transformed into the tuple \((\bar{b}_{i1}, \ldots, \bar{b}_{i\alpha})\) in \(r'\). Likewise, we transform \(\varphi\) to \(\varphi'\) by replacing every constant \(a_i\) by its encoding \(\bar{b}_i\) and by replacing any variable \(x_j\) by a vector \((x_{j1}, \ldots, x_{jm})\) of fresh variables.
PSPACE-Membership of First-Order Queries

Proof of the PSPACE-Membership (continued)

Every tuple \((a_{i1}, \ldots, a_{i\alpha})\) in \(r\) is transformed into the tuple \((\bar{b}_{i1}, \ldots, \bar{b}_{i\alpha})\) in \(r'\). Likewise, we transform \(\varphi\) to \(\varphi'\) by replacing every constant \(a_i\) by its encoding \(\bar{b}_i\) and by replacing any variable \(x_j\) by a vector \((x_{j1}, \ldots, x_{jm})\) of fresh variables.

2. Eliminating all atoms \(R(t_1, \ldots, t_k)\) from \(\varphi'\). Let \(R\) be a \(k\)-ary relation symbol occurring in \(\varphi'\) and suppose that the corresponding relation in \(D'\) contains the tuples \((c_{11}, \ldots, c_{1k}), (c_{21}, \ldots, c_{2k}), \ldots, (c_{N1}, \ldots, c_{Nk})\). Then we transform \(\varphi'\) into the formula \(\varphi''\) by replacing all atoms of the form \(R(t_1, \ldots, t_k)\) by the following disjunction:

\[
\bigvee_{j=1}^{N} \left( t_1 = c_{j1} \land \cdots \land t_k = c_{jk} \right)
\]
PSPACE-Membership of First-Order Queries

Proof of the PSPACE-Membership (continued)

3. Replacing first-order variables by propositional variables. The only atoms occurring in $\varphi''$ are equality atoms $s = t$, where the terms $s, t$ are either variables (which can take the value 0 or 1) or the constants 0, 1. We identify 0 with the truth value false and 1 with the truth value true. Then we can transform $\varphi''$ into the QSAT formula $\psi$ by replacing the equality atoms by “equivalent” propositional formulae in the obvious way:

$$x = y \leadsto x \leftrightarrow y$$
$$x = 0, 0 = x \leadsto \neg x$$
$$0 = 1, 1 = 0 \leadsto \text{false} \ (\text{or } x \land \neg x)$$
$$x = 1, 1 = x \leadsto x$$
$$0 = 0, 1 = 1 \leadsto \text{true} \ (\text{or } x \lor \neg x)$$
PSPACE-Membership of First-Order Queries

Proof of the PSPACE-Membership (continued)

3. Replacing first-order variables by propositional variables. The only atoms occurring in $\varphi''$ are equality atoms $s = t$, where the terms $s, t$ are either variables (which can take the value 0 or 1) or the constants 0, 1. We identify 0 with the truth value false and 1 with the truth value true. Then we can transform $\varphi''$ into the QSAT formula $\psi$ by replacing the equality atoms by “equivalent” propositional formulae in the obvious way:

$$
\begin{align*}
  x = y & \iff x \leftrightarrow y \\
  x = 0, 0 = x & \iff \neg x \\
  0 = 1, 1 = 0 & \implies \text{false} \quad \text{(or } x \land \neg x) \\
  0 = 0, 1 = 1 & \implies \text{true} \quad \text{(or } x \lor \neg x) 
\end{align*}$$

Clearly, $\varphi$ evaluates to true over $D$ $\iff$ $\varphi'$ evaluates to true over $D'$ $\iff$ $\varphi''$ evaluates to true independently of any database $\iff$ $\psi$ is true.
Discussion

**Easy Consequences**

**PSPACE-hardness of QSAT.** The above proof of the PSPACE-hardness of FO evaluation together with the above reduction from FO evaluation to QSAT immediately yields the PSPACE-hardness of QSAT.
Discussion

Easy Consequences

**PSPACE-hardness of QSAT.** The above proof of the PSPACE-hardness of FO evaluation together with the above reduction from FO evaluation to QSAT immediately yields the PSPACE-hardness of QSAT.

**Narrowing FO evaluation and PSPACE-hardness.**

- The first 2 steps in the above reduction from FO evaluation to QSAT allowed us to transform an arbitrary FO formula $\varphi$ over a database with arbitrary finite domain into an FO formula $\psi$ over the domain $\{0, 1\}$, s.t. the atomic formulas of $\psi$ are equalities only. Moreover, negation can be shifted immediately in front of the equalities.

- Equalities and negated equalities over $\{0, 1\}$ can be represented by relations $\text{eq}$ and $\text{noteq}$ in the obvious way (this works for any finite domain), i.e., $\text{eq} = \{(0, 0), (1, 1)\}$ and $\text{noteq} = \{(0, 1), (1, 0)\}$.

- It follows that FO evaluation remains PSPACE-hard even if we disallow equalities and negation in the FO formulas.
Conjunctive Queries

**Definition**

**Conjunctive queries** (CQs) are a special case of first-order queries whose only connective is $\land$ and whose only quantifier is $\exists$ (i.e., $\lor$, $\neg$, and $\forall$ are excluded). Alternatively, CQs can be considered as a single datalog rule

$$Q : r(u) \leftarrow r_1(u_1) \land \ldots \land r_n(u_n)$$

where $n \geq 0$; $r_1, \ldots, r_n$ are (not necessarily distinct) extensional relation symbols and $u, u_1, \ldots, u_n$ are lists of terms of appropriate length. Moreover, all variables in $u$ occur in at least one $u_i$.

In a **Boolean conjunctive query**, the head of the rule $Q$ is the 0-ary intensional relation symbol $true()$ (rather than some arbitrary term $r(u)$).

**Remark.** Conjunctive queries correspond to select-project-join queries in the relational algebra, i.e., unnested select-from-where queries in SQL.
Conjunctive Queries

**Theorem**

The query complexity and the combined complexity of conjunctive queries are \( \text{NP-complete} \).

**Proof**

**NP-Membership** (of the combined complexity). For each variable \( u \) of the query, we guess a domain element to which \( u \) is instantiated. Then we check whether all the resulting ground atoms in the query body exist in \( D \). This check is obviously feasible in polynomial time.

**Hardness** (of the query complexity). We reduce the NP-complete 3-Colorability problem to our problem. For this purpose, we consider an input database over the binary relation symbol \( Edge \).
NP-Hardness of query complexity

Since we are considering the query complexity, the database $D$ is fixed (but arbitrarily chosen). We choose $D$ with a single relation $Edge = \{\langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 2, 1 \rangle, \langle 2, 3 \rangle, \langle 3, 1 \rangle, \langle 3, 2 \rangle\}$
NP-Hardness of query complexity

Since we are considering the query complexity, the database $D$ is fixed (but arbitrarily chosen). We choose $D$ with a single relation $Edge = \{\langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 2, 1 \rangle, \langle 2, 3 \rangle, \langle 3, 1 \rangle, \langle 3, 2 \rangle\}$

Now let $G = (V, E)$ be an arbitrary instance of the 3-Colorability problem. From this, we define the Boolean conjunctive query $Q$ as follows. $Q$ contains the variables $X = \{x_i | v_i \in V\}$. Moreover, we set

$$\text{ans}() \leftarrow \bigwedge_{[v_i, v_j] \in E} \text{Edge}(x_i, x_j)$$

Clearly, this reduction is feasible in logarithmic space. The correctness is seen as follows: $Q$ is true over the DB $D$ $\iff$ The variables in $X$ can be instantiated to values $\{1, 2, 3\}$, s.t. $Q$ contains only ground atoms occurring in $D$ $\iff$ The graph $G$ has a valid 3-coloring.
NP-Hardness of query complexity

Since we are considering the query complexity, the database $D$ is fixed (but arbitrarily chosen). We choose $D$ with a single relation $\text{Edge} = \{\langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 2, 1 \rangle, \langle 2, 3 \rangle, \langle 3, 1 \rangle, \langle 3, 2 \rangle\}$

Now let $G = (V, E)$ be an arbitrary instance of the 3-Colorability problem. From this, we define the Boolean conjunctive query $Q$ as follows. $Q$ contains the variables $X = \{x_i | v_i \in V\}$. Moreover, we set

$$\text{ans}() \leftarrow \bigwedge_{[v_i,v_j] \in E} \text{Edge}(x_i, x_j)$$

Clearly, this reduction is feasible in logarithmic space. The correctness is seen as follows: $Q$ is true over the DB $D$ $\iff$ The variables in $X$ can be instantiated to values $\{1, 2, 3\}$, s.t. $Q$ contains only ground atoms occurring in $D$ $\iff$ The graph $G$ has a valid 3-coloring.
Learning Objectives

- The power of unrestricted alternation (in QBF)
- PSPACE as the complexity class of many strategy games
- The relationship of PSPACE and PH
- Complexity of query evaluation, first-order queries