Technische Universität Dresden

International Center for Computational Logic

COMPLEXITY THEORY

Lecture 10: Polynomial Space and Games

Stephan Mennicke

Knowledge-Based Systems

TU Dresden, 11 Nov 2025

Consequences of Savitch's Theorem

Theorem 10.1 (Savitch's Theorem, 1970): For any function $f : \mathbb{N} \to \mathbb{R}^+$ with $f(n) \geq \log n$:

$$\text{NSpace}(f(n)) \subseteq \text{DSpace}(f^2(n)).$$

Corollary 10.2: PSpace = NPSpace.

Proof: PSpace $\subseteq$ NPSpace is clear. The converse follows since the square of a polynomial is still a polynomial. □

Similarly for "bigger" classes, e.g., ExpSpace = NExpSpace.

Corollary 10.3: NL $\subseteq$ DSpace$(O(\log^2 n))$.

Note that $\log^2(n) \notin O(\log n)$, so we do not obtain NL = L from this.

A Note on DSpace($O(\log^2 n)$) – Steve's Class SC

Stephen Cook studied the class of problems in polylogarithmic space polynomial-time, called **Steve's Class** (SC) in his honor.

$$\text{PolyLog} = \bigcup_{d \geq 1} \text{DSpace}(\log^k n)$$

The class CS $\subseteq$ PolyLog $\cap$ P.Because of Savitch,

$$\text{L} \subseteq \text{NL} \subseteq \text{SC} \subseteq \text{P}$$

It is open whether L $\subsetneq$ NL and NL $\subsetneq$ SC.

Another candidate in these lower realms is SL (for symmetric logarithmic space). The key problem there is **USTCONN**, which has been known to be SL-complete, yet was shown to be in L 10th November 2004 (**yesterday!**) by Omer Reingold.

The Class PSpace

We defined PSpace as:

$$\text{PSpace} = \bigcup_{d \geq 1} \text{DSpace}(n^d)$$

and we observed that

$$\text{P} \subseteq \text{NP} \subseteq \text{PSpace} = \text{NPSpace} \subseteq \text{ExpTime}.$$

We can also define a corresponding notion of PSpace-hardness:

Definition 10.4:

- A language **H** is PSpace-hard, if **L** $\leq_p$ **H** for every language **L** $\in$ PSpace.
- A language **C** is PSpace-complete, if **C** is PSpace-hard and **C** $\in$ PSpace.

Quantified Boolean Formulae (QBF)

A QBF is a formula of the following form:

$$\mathsf{Q}_1 X_1.\mathsf{Q}_2 X_2.\cdots\mathsf{Q}_\ell X_\ell.\varphi[X_1,\ldots,X_\ell]$$

where $\mathsf{Q}_i \in \{\exists, \forall\}$ are quantifiers, X_i are propositional logic variables, and φ is a propositional logic formula with variables $X_1,\ldots,X_\ell$ and constants $\top$ (true) and $\bot$ (false)

Semantics:

- Propositional formulae without variables (only constants $\top$ and $\bot$) are evaluated as usual
- $\exists X.\varphi[X]$ is true if either $\varphi[X/\top]$ or $\varphi[X/\bot]$ are true
- $\forall X.\varphi[X]$ is true if both $\varphi[X/\top]$ and $\varphi[X/\bot]$ are true

  (where $\varphi[X/\top]$ is "φ with X replaced by $\top$, and similar for $\bot$)

Deciding QBF Validity

TRUE QBF

Input: A quantified Boolean formula φ.

Problem: Is φ true (valid)?

Observation: We can assume that the quantified formula is in CNF or 3-CNF (same transformations possible as for propositional logic formulae)

Consider a propositional logic formula φ with variables $X_1, \ldots, X_\ell$:

Example 10.5: The QBF $\exists X_1. \cdots \exists X_\ell.\varphi$ is true if and only if φ is satisfiable.

Example 10.6: The QBF $\forall X_1. \cdots \forall X_\ell.\varphi$ is true if and only if φ is a tautology.

Theorem 10.7: TRUE QBF is PSpace-complete.

Proof:

(1) **TRUE QBF** $\in$ PSpace:
Give an algorithm that runs in polynomial space.

(2) **TRUE QBF** is PSpace-hard:
Proof by reduction from the word problem of any polynomially space-bounded TM.

□

PSpace-Hardness of True QBF

Express TM computation in logic, similar to Cook-Levin

Given:

An arbitrary polynomially space-bounded NTM, that is:

- a polynomial p
- a p-space bounded 1-tape NTM $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$

Intended reduction

Given a word w, define a QBF $\varphi_{p,\mathcal{M},w}$ such that
$\varphi_{p,\mathcal{M},w}$ is true if and only if $\mathcal{M}$ accepts w in space $p(|w|)$.

Notes

- We show the reduction for NTMs, which is more than needed, but makes little difference in logic and allows us to reuse our previous formulae from Cook-Levin
- The proof actually shows many reductions, one for every polyspace NTM, showing PSpace-hardness from first principles

Simulating Polynomial Space Computations

For Cook-Levin, we used one set of configuration variables for every computating step:
polynomial time $\rightsquigarrow$ polynomially many variables

Problem: For polynomial space, we have $2^{O(p(n))}$ possible steps ...

What would Savitch do?

Define a formula $\text{CanYield}_i(\overline{C}_1, \overline{C}_2)$ to state that $\overline{C}_2$ is reachable from $\overline{C}_1$ in at most 2^i steps:

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}.\text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

But what is $\overline{C}_1 = \overline{C}_2$ supposed to mean here? It is short for:

$$\bigwedge_{q \in Q} Q_q^1 \leftrightarrow Q_q^2 \wedge \bigwedge_{0 \leq i < p(n)} P_i^1 \leftrightarrow P_i^2 \wedge \bigwedge_{a \in \Gamma, 0 \leq i < p(n)} S_{a,i}^1 \leftrightarrow S_{a,i}^2$$

Putting Everything Together

We define the formula $\varphi_{p,\mathcal{M},w}$ as follows:

$$\varphi_{p,\mathcal{M},w} := \exists\overline{C}_1.\exists\overline{C}_2.\mathsf{Start}_{\mathcal{M},w}(\overline{C}_1) \wedge \mathsf{Acc\text{-}Conf}(\overline{C}_2) \wedge \mathsf{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

where we select d to be the least number such that $\mathcal{M}$ has less than $2^{dp(n)}$ configurations in space $p(n)$.

Lemma 10.8: $\varphi_{p,\mathcal{M},w}$ is satisfiable if and only if $\mathcal{M}$ accepts w in space $p(|w|)$.

Did we do it?

Note: we used only existential quantifiers when defining $\varphi_{p,\mathcal{M},w}$:

$$\mathsf{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \mathsf{Next}(\overline{C}_1, \overline{C}_2)$$

$$\mathsf{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}.\mathsf{Conf}(\overline{C}) \wedge \mathsf{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \mathsf{CanYield}_i(\overline{C}, \overline{C}_2)$$

$$\varphi_{p,\mathcal{M},w} := \exists \overline{C}_1.\exists \overline{C}_2.\mathsf{Start}_{\mathcal{M},w}(\overline{C}_1) \wedge \mathsf{Acc\text{-}Conf}(\overline{C}_2) \wedge \mathsf{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

Now that's quite interesting . . .

- With only (non-negated) $\exists$ quantifiers, **True QBF** coincides with **Sat**
- **Sat** is in NP
- So we showed that the word problem for PSpace NTMs to be in NP

So we found that NP = PSpace!

Strangely, most textbooks claim that this is not known to be true . . .
Are we up for the next Turing Award, or did we make a mistake?

Size

How big is $\varphi_{p,\mathcal{M},w}$?

$$\mathsf{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \mathsf{Next}(\overline{C}_1, \overline{C}_2)$$

$$\mathsf{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}.\mathsf{Conf}(\overline{C}) \wedge \mathsf{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \mathsf{CanYield}_i(\overline{C}, \overline{C}_2)$$

$$\varphi_{p,\mathcal{M},w} := \exists \overline{C}_1.\exists \overline{C}_2.\mathsf{Start}_{\mathcal{M},w}(\overline{C}_1) \wedge \mathsf{Acc\text{-}Conf}(\overline{C}_2) \wedge \mathsf{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

Size of $\mathsf{CanYield}_{i+1}$ is more than twice the size of $\mathsf{CanYield}_i$

$\rightsquigarrow$ Size of $\varphi_{p,\mathcal{M},w}$ is in $2^{O(p(n))}$. Oops.

A correct reduction: We redefine CanYield by setting

$$\mathsf{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) :=$$
$$\exists \overline{C}.\mathsf{Conf}(\overline{C}) \wedge$$
$$\forall \overline{Z}_1.\forall \overline{Z}_2.(((\overline{Z}_1 = \overline{C}_1 \wedge \overline{Z}_2 = \overline{C}) \vee (\overline{Z}_1 = \overline{C} \wedge \overline{Z}_2 = \overline{C}_2)) \rightarrow \mathsf{CanYield}_i(\overline{Z}_1, \overline{Z}_2))$$

Size

Let's analyse the size more carefully this time:

$$\mathsf{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) :=$$
$$\exists\overline{C}.\mathsf{Conf}(\overline{C}) \land$$
$$\forall\overline{Z}_1.\forall\overline{Z}_2.(((\overline{Z}_1 = \overline{C}_1 \land \overline{Z}_2 = \overline{C}) \lor (\overline{Z}_1 = \overline{C} \land \overline{Z}_2 = \overline{C}_2)) \to \mathsf{CanYield}_i(\overline{Z}_1, \overline{Z}_2))$$

- $\mathsf{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2)$ extends $\mathsf{CanYield}_i(\overline{C}_1, \overline{C}_2)$ by parts that are linear in the size of configurations $\rightsquigarrow$ growth in $O(p(n))$
- Maximum index i used in $\varphi_{p,\mathcal{M},w}$ is $dp(n)$, that is in $O(p(n))$
- Therefore: $\varphi_{p,\mathcal{M},w}$ has size $O(p^2(n))$ – and thus can be computed in polynomial time

Exercise:

Why can we just use $dp(n)$ in the reduction? Don't we have to compute it somehow? Maybe even in polynomial time?

The Power of QBF

Theorem 10.7: True QBF is PSpace-complete.

Proof:

(1) **True QBF** $\in$ PSpace:
Give an algorithm that runs in polynomial space.

(2) **True QBF** is PSpace-hard:
Proof by reduction from the word problem of any polynomially space-bounded TM.

$\square$

A More Common Logical Problem in PSpace

Recall standard first-order logic:

- Instead of propositional variables, we have atoms (predicates with constants and variables)
- Instead of propositional evaluations we have first-order structures (or interpretations)
- First-order quantifiers can be used on variables
- Sentences are formulae where all variables are quantified
- A sentence can be satisfied or not by a given first-order structure

FOL Model Checking

Input: A first-order sentence φ and a finite first-order structure $\mathcal{I}$.

Problem: Is φ satisfied by $\mathcal{I}$?

First-Order Logic is PSpace-complete

Theorem 10.9: FOL Model Checking is PSpace-complete.

Proof:

(1) **FOL Model Checking** $\in$ PSpace:
Give algorithm that runs in polynomial space.

(2) **FOL Model Checking** is PSpace-hard:
Proof by reduction **True QBF** $\leq_p$ **FOL Model Checking.**

$\square$

Checking FOL Models in Polynomial Space (Sketch)

```
Eval(φ, 𝓘) {
  switch (φ) :
    case p(c_1, ..., c_n) :  return ⟨c_1, ..., c_n⟩ ∈ p^𝓘
    case ¬ψ :  return NOT Eval(ψ, 𝓘)
    case ψ_1 ∧ ψ_2 :  return Eval(ψ_1, 𝓘) AND Eval(ψ_2, 𝓘)
    case ∃x.ψ :
      for c ∈ Δ^𝓘 :
        if Eval(ψ[x ↦ c], 𝓘) :  return TRUE
      // eventually, if no success:
      return FALSE
}
```

- We can assume φ only uses $\neg$, $\wedge$ and $\exists$ (easy to get)
- We use $\Delta^{\mathcal{I}}$ to denote the (finite!) domain of $\mathcal{I}$
- We allow domain elements to be used like constants in the formula

Hardness of FOL Model Checking

Given: a QBF $\varphi = \mathsf{Q}_1 X_1.\cdots\mathsf{Q}_\ell X_\ell.\psi$

FOL Model Checking Problem:

- Interpretation domain $\Delta^{\mathcal{I}} := \{0, 1\}$
- Single predicate symbol true with interpretation $\mathrm{true}^{\mathcal{I}} = \{\langle 1\rangle\}$
- FOL formula φ' is obtained by replacing variables in input QBF with corresponding first-order expressions:

$$\mathsf{Q}_1 x_1.\cdots\mathsf{Q}_\ell x_\ell.\psi[X_1 \mapsto \mathrm{true}(x_1), \ldots, X_\ell \mapsto \mathrm{true}(x_\ell)]$$

Lemma 10.10: $\langle \mathcal{I}, \varphi'\rangle \in$ **FOL Model Checking** if and only if $\varphi \in$ **True QBF**.

First-Order Logic is PSpace-complete

Theorem 10.9: FOL MODEL CHECKING is PSpace-complete.

Proof:

(1) **FOL MODEL CHECKING** $\in$ PSpace:
Give algorithm that runs in polynomial space.

(2) **FOL MODEL CHECKING** is PSpace-hard:
Proof by reduction **TRUE QBF** $\leq_p$ **FOL MODEL CHECKING**.

$\square$

FOL Model Checking: Practical Significance

Why is **FOL Model Checking** a relevant problem?

Correspondence with database query answering:

- Finite first-order interpretation = database
- First-order logic formula = database query
- Satisfying assignments (for non-sentences) = query results

Known correspondence:
As a query language, FOL has the same expressive power as (basic) SQL (relational algebra).

Corollary 10.11: Answering SQL queries over a given database is PSpace-complete.

But why should we do it after all? Do database engineers have better ideas to tackle hard problems? Answering database queries is fast!

Review: Checking FOL Models in Time and Space (Sketch)

```
Eval(φ, I) {
  switch (φ) :
    case p(c_1, ..., c_n) :  return ⟨c_1, ..., c_n⟩ ∈ p^I
    case ¬ψ :  return NOT Eval(ψ, I)
    case ψ_1 ∧ ψ_2 :  return Eval(ψ_1, I) AND Eval(ψ_2, I)
    case ∃x.ψ :
      for c ∈ Δ^I :
        if Eval(ψ[x ↦ c], I) :  return TRUE
      // eventually, if no success:
      return FALSE
}
```

- Let φ be a formula of length m and $|\mathcal{I}| = n$.
- Runtime is bounded by $O((n+2)^{m+2})$.
- Memory usage bounded by $O(m \log m + (m+1) \log n)$.
- **Data complexity** vs. **combined Complexity** (or query complexity).

Games

Games as Computational Problems

Many single-player games relate to NP-complete problems:

- Sudoku
- Minesweeper
- Tetris
- ...

Decision problem: Is there a solution?

(For Tetris: is it possible to clear all blocks?)

What about two-player games?

- Two players take moves in turns
- The players have different goals
- The game ends if a player wins

Decision problem: Does Player 1 have a winning strategy?

In other words: can Player 1 enforce winning, whatever Player 2 does?

Example: The Formula Game

A contrived game, to illustrate the idea:

- Given: a propositional logic formula φ with consecutively numbered variables $X_1, \ldots X_\ell$.
- Two players take turns in selecting values for the next variable:
  - Player 1 sets X_1 to true or false
  - Player 2 sets X_2 to true or false
  - Player 1 sets X_3 to true or false
  - …

  until all variables are set.
- Player 1 wins if the assignment makes φ true.
  Otherwise, Player 2 wins.

Deciding the Formula Game

FORMULA GAME

Input: A formula φ.

Problem: Does Player 1 have a winning strategy on φ?

Theorem 10.12: FORMULA GAME is PSpace-complete.

Proof sketch: FORMULA GAME is essentially the same as **TRUE QBF**.

Having a winning strategy means: there is a truth value for X_1, such that, for all truth values of X_2, there is a truth value of X_3, $\ldots$ such that φ becomes true.

If we have a QBF where quantifiers do not alternate, we can add dummy quantifiers and variables that do not change the semantics to get the same alternating form as for the Formula Game. □

Example: The Geography Game

A children's game:

- Two players are taking turns naming cities.
- Each city must start with the last letter of the previous.
- Repetitions are not allowed.
- The first player who cannot name a new city looses.

A mathematicians' game:

- Two players are marking nodes on a directed graph.
- Each node must be a successor of the previous one.
- Repetitions are not allowed.
- The first player who cannot mark a new node looses.

Decision problem **(Generalised) Geography**:
given a graph and start node, does Player 1 have a winning strategy?

GEOGRAPHY is PSpace-complete

Theorem 10.13: GENERALISED GEOGRAPHY is PSpace-complete.

Proof:

(1) **GEOGRAPHY** $\in$ PSpace:
Give algorithm that runs in polynomial space.
It is not difficult to provide a recursive algorithm similar to the one for **TRUE QBF** or **FOL MODEL CHECKING.**

(2) **GEOGRAPHY** is PSpace-hard:
Proof by reduction **FORMULA GAME** $\leq_p$ **GEOGRAPHY.**

□

Geography is PSpace-hard

Let φ with variables $X_1, \ldots, X_\ell$ be an instance of **Formula Game.**
Without loss of generality, we assume:

- ℓ is odd (Player 1 gets the first and last turn)
- φ is in CNF

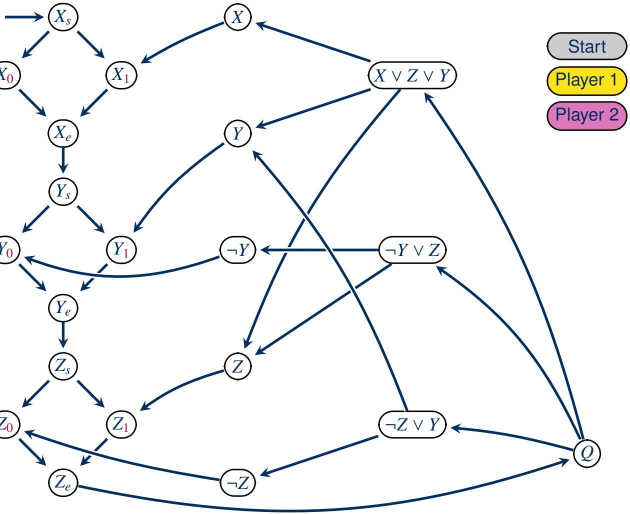
We now build a graph that encodes **Formula Game** in terms of **Geography**

- The left-hand side of the graph is a chain of diamond structures that represent the choices that players have when assigning truth values
- The right-hand side of the graph encodes the structure of φ: Player 2 may choose a clause (trying to find one that is not true under the assignment); Player 1 may choose a literal (trying to find one that is true under the assignment).

(see board or [Sipser, Theorem 8.14]) □

GEOGRAPHY is PSpace-hard: Example

We consider the formula $\exists X.\forall Y.\exists Z.(X \vee Z \vee Y) \wedge (\neg Y \vee Z) \wedge (\neg Z \vee Y)$



More Games

The characteristic of PSpace is quantifier alternation

This is closely related to taking turns in 2-player games.

Are many games PSpace-complete?

- Issue 1: many games are finite – that is: computationally trivial
  $\rightsquigarrow$ generalise games to arbitrarily large boards
  - generalised Tic-Tac-Toe is PSpace-complete
  - generalised Reversi is PSpace-complete
  - it is not always clear how to generalise a game (Generalised Backgammon?)
- Issue 2: (generalised) games where moves can be reversed may require very long matches
  $\rightsquigarrow$ such games often are even harder
  - generalised Go with Japanese ko rule is ExpTime-complete
  - generalised Draughts (Checkers) is ExpTime-complete
  - generalised Chess (without 50-move no-capture draw rule) is ExpTime-complete

Surprisingly, some of these games, e.g. Chess, are known to become even harder – namely ExpSpace-complete – if the exact same board position is not allowed to re-occur in a match. For Go, this case is open (link).

Summary and Outlook

True QBF is PSpace-complete

FOL Model Checking and the related problem of SQL query answering are PSpace-complete

Some games are PSpace-complete

What's next?

- Logarithmic space
- Complements of space classes