The Class PSpace

We defined PSpace as:

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

and we observed that

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

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

**Definition 10.1:**

- A language \( H \) is **PSpace-hard**, if \( L \leq_p H \) for every language \( L \in \text{PSpace} \).
- A language \( C \) is **PSpace-complete**, if \( C \) is PSpace-hard and \( C \in \text{PSpace} \).

Quantified Boolean Formulae (QBF)

A QBF is a formula of the following form:

\[ \exists X_1 \cdot \cdot \cdot \exists X_\ell. \varphi[X_1, \ldots, X_\ell] \]

where \( \exists \in \{\exists, \forall\} \) are quantifiers, \( X_i \) are propositional logic variables, and \( \varphi \) 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

(\( \varphi[X/\top] \) is \( \varphi \) with \( X \) replaced by \( \top \), and similar for \( \bot \)).

Deciding QBF Validity

**True QBF**

- Input: A quantified Boolean formula \( \varphi \).
- Problem: Is \( \varphi \) 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 \( \varphi \) with variables \( X_1, \ldots, X_\ell \):

**Example 10.2:** The QBF \( \exists X_1. \cdot \cdot \cdot \exists X_\ell. \varphi \) is true if and only if \( \varphi \) is satisfiable.

**Example 10.3:** The QBF \( \forall X_1. \cdot \cdot \cdot \forall X_\ell. \varphi \) is true if and only if \( \varphi \) is a tautology.
The Power of QBF

Theorem 10.4: True QBF is PSpace-complete.

Proof:
(1) True QBF ∈ PSpace:
Give an algorithm that runs in polynomial space.

(2) True QBF is PSpace-hard:
Proof by reduction from the word problem for polynomially space-bounded TMs.

Markus Krötzsch, 21th Nov 2017 Complexity Theory slide 6 of 34

Solving True QBF in PSpace

01 TrueQBF(ϕ) {
02 if ϕ has no quantifiers :
03 return “evaluation of ϕ”
04 else if ϕ = ∃X. ψ :
05 return (TrueQBF(ψ[X/⊤]) OR TrueQBF(ψ[X/⊥]))
06 else if ϕ = ∀X. ψ :
07 return (TrueQBF(ψ[X/⊤]) AND TrueQBF(ψ[X/⊥]))
08 }

- Evaluation in line 03 can be done in polynomial space
- Recursions in lines 05 and 07 can be executed one after the other, reusing space
- Maximum depth of recursion = number of variables (linear)
- Store one variable assignment per recursive call

Markus Krötzsch, 21th Nov 2017 Complexity Theory slide 7 of 34

PSpace-Hardness of True QBF

Express TM computation in logic, similar to Cook-Levin

Given:
• a polynomial p
• a p-space bounded 1-tape NTM $M = (Q, Σ, Γ, δ, q_0, q_{accept})$
• a word w

Intended reduction
Define a QBF $ϕ_p, M, w$ such that $ϕ_p, M, w$ is true if and only if $M$ accepts $w$ in space $p(|w|)$.

Markus Krötzsch, 21th Nov 2017 Complexity Theory slide 8 of 34

Review: Encoding Configurations

Use propositional variables for describing configurations:
$Q_q$ for each $q ∈ Q$ means “$M$ is in state $q ∈ Q”
$P_i$ for each $0 ≤ i < p(n)$ means “the head is at Position $i”$
$S_{a,i}$ for each $a ∈ Γ$ and $0 ≤ i < p(n)$ means “tape cell $i$ contains Symbol $a”$

Represent configuration $(q, p, a_0, . . . , a_{p(n)})$ by assigning truth values to variables from the set

$C := \{Q_q, P_i, S_{a,i} | q ∈ Q, a ∈ Γ, 0 ≤ i < p(n)\}$

using the truth assignment $β$ defined as

$β(Q_q) := \begin{cases} 1 & q = s \\ 0 & q ≠ s \end{cases}$

$β(P_i) := \begin{cases} 1 & i = p \\ 0 & i ≠ p \end{cases}$

$β(S_{a,i}) := \begin{cases} 1 & a = a_i \\ 0 & a ≠ a_i \end{cases}$
Review: Validating Configurations

We define a formula $\text{Conf}(\mathcal{C})$ for a set of configuration variables $\mathcal{C} = \{Q_q, P_i, S_{a,i} \mid q \in Q, i \in \Gamma, 0 \leq i < p(n)\}$ as follows:

$$\text{Conf}(\mathcal{C}) := \bigwedge_{q \in Q} (Q_q \land \bigwedge_{q' < q} \neg Q_{q'}) \land \bigwedge_{p < q(n)} (P_p \land \bigwedge_{p' > p} \neg P_{p'}) \land \bigwedge_{0 \leq j < q(n)} \bigwedge_{k = a(j)} \neg S_{a,i}$$

"the assignment is a valid configuration": TM in exactly one state $q \in Q$" head in exactly one position $p < p(n)$" exactly one $a \in \Gamma$ in each cell"

For an assignment $\beta$ defined on variables in $\mathcal{C}$ define

$$\text{conf}(\mathcal{C}, \beta) := \begin{cases} (q, p, w_1 \ldots w_n) & \text{if } q \in Q, p < p(n), \text{ and } w_i \text{ is a potential configuration of } M \text{ for all } i \end{cases}$$

Note: $\beta$ may be defined on other variables besides those in $\mathcal{C}$.

**Lemma 10.5:** If $\beta$ satisfies $\text{Conf}(\mathcal{C})$ then $|\text{conf}(\mathcal{C}, \beta)| = 1$. We can therefore write $\text{conf}(\mathcal{C}, \beta) = (q, p, w)$ to simplify notation.

**Observations:**
- $\text{conf}(\mathcal{C}, \beta)$ is a potential configuration of $M$, but it may not be reachable from the start configuration of $M$ on input $w$.
- Conversely, every configuration $(q, p, w_1 \ldots w_{p(n)})$ induces a satisfying assignment $\beta$ or which $\text{conf}(\mathcal{C}, \beta) = (q, p, w_1 \ldots w_{p(n)})$.

Review: Transitions Between Configurations

Consider the following formula $\text{Next}(\mathcal{C}, \mathcal{C}')$ defined as

$$\text{Next}(\mathcal{C}, \mathcal{C}') := \text{Conf}(\mathcal{C}) \land \text{Conf}(\mathcal{C}') \land \text{NoChange}(\mathcal{C}, \mathcal{C}') \land \text{Change}(\mathcal{C}, \mathcal{C}')$$

$$\text{NoChange} := \bigwedge_{0 \leq j < p(n)} \bigwedge_{a(j)} (S_{a,i} \rightarrow S_{a,i})$$

$$\text{Change} := \bigwedge_{0 \leq j < p(n)} \bigwedge_{a(j)} \bigwedge_{(q', a, D) \notin \delta(q,a,p)} (Q_{q'} \land S_{a,i} \land P_{D(p)})$$

where $D(p)$ is the position reached by moving in direction $D$ from $p$.

**Lemma 10.6:** For any assignment $\beta$ defined on $\mathcal{C} \cup \mathcal{C}'$:

$\beta$ satisfies $\text{Next}(\mathcal{C}, \mathcal{C}')$ if and only if $\text{conf}(\mathcal{C}, \beta) \vdash_M \text{conf}(\mathcal{C}', \beta)$

Review: Start and End

**Defined so far:**
- $\text{Conf}(\mathcal{C})$: $\mathcal{C}$ describes a potential configuration
- $\text{Next}(\mathcal{C}, \mathcal{C}')$: $\text{conf}(\mathcal{C}, \beta) \vdash_M \text{conf}(\mathcal{C}', \beta)$

**Start configuration:** Let $w = w_0 \ldots w_{p(n)} \in \Sigma^*$ be the input word

$$\text{Start}_{M,w}(\mathcal{C}) := \text{Conf}(\mathcal{C}) \land Q_{q_0} \land P_0 \land \bigwedge_{j=n} S_{a,j} \land \bigwedge_{j=n} S_{a,j}$$

Then an assignment $\beta$ satisfies $\text{Start}_{M,w}(\mathcal{C})$ if and only if $\mathcal{C}$ represents the start configuration of $M$ on input $w$.

**Accepting stop configuration:**

$$\text{Acc-Conf}(\mathcal{C}) := \text{Conf}(\mathcal{C}) \land Q_{\text{accept}}$$

Then an assignment $\beta$ satisfies $\text{Acc-Conf}(\mathcal{C})$ if and only if $\mathcal{C}$ represents an accepting configuration of $M$. 

Markus Krötzsch, 21th Nov 2017 Complexity Theory slide 10 of 34

Markus Krötzsch, 21th Nov 2017 Complexity Theory slide 11 of 34

Markus Krötzsch, 21th Nov 2017 Complexity Theory slide 12 of 34

Markus Krötzsch, 21th Nov 2017 Complexity Theory slide 13 of 34
Simulating Polynomial Space Computations

For Cook-Levin, we used one set of configuration variables for every computing step: polynomially time $\sim$ polynomially many variables.

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

**What would Savitch do?**

Define a formula $\text{CanYield}(C_1, C_2)$ to state that $C_2$ is reachable from $C_1$ in at most $2^i$ steps:

\[
\text{CanYield}_i(C_1, C_2) := (C_1 = C_2) \lor \text{Next}(C_1, C_2)
\]

But what if $C_1 = C_2$? supposed to mean here. It is short for:

\[
\bigwedge_{q \in Q} Q_i^q \leftrightarrow Q_i^q \land \bigwedge_{0 \leq i < p(n)} P_i^1 \leftrightarrow P_i^1 \land \bigwedge_{a \in \Gamma, 0 \leq i < p(n)} S_{i,a}^1 \leftrightarrow S_{i,a}^2
\]

Putting Everything Together

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

\[
\phi_{p,M,w} := \exists C_1. \exists C_2. \text{Start}_M(w)(C_1) \land \text{Acc-Conf}(C_2) \land \text{CanYield}_{dp(n)}(C_1, C_2)
\]

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

**Lemma 10.7:** $\phi_{p,M,w}$ is satisfiable if and only if $M$ accepts $w$ in space $p(|w|)$.

Did we do it?

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

\[
\phi_{p,M,w} := \exists C_1. \exists C_2. \text{Start}_M(w)(C_1) \land \text{Acc-Conf}(C_2) \land \text{CanYield}_{dp(n)}(C_1, C_2)
\]

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 $\phi_{p,M,w}$?

\[
\phi_{p,M,w} := \exists C_1. \exists C_2. \text{Start}_M(w)(C_1) \land \text{Acc-Conf}(C_2) \land \text{CanYield}_{dp(n)}(C_1, C_2)
\]

Where $d$ is the least number such that $M$ has less than $2^{dp(n)}$ configurations in space $p(n)$.

**A correct reduction:** We redefine CanYield by setting

\[
\text{CanYield}_{dp(n)}(C_1, C_2) :=
\]

\[
\exists C. \text{Conf}(C) \land \forall Z_1, Z_2. ((Z_1 = C_1 \land Z_2 = C) \lor (Z_1 = C \land Z_2 = C_2)) \rightarrow \text{CanYield}(Z_1, Z_2)
\]
Size

Let’s analyse the size more carefully this time:

\[
\text{CanYield}_{i+1}(C_1, C_2) := \\
\exists C. \text{Conf}(C) \land \\
\forall Z_1. \forall Z_2. (((Z_1 = C_1 \land Z_2 = C_1) \lor (Z_1 = C_1 \land Z_2 = C_2)) \rightarrow \text{CanYield}(Z_1, Z_2))
\]

- \text{CanYield}_{i+1}(C_1, C_2) extends \text{CanYield}_{i}(C_1, C_2) by parts that are linear in the size of configurations \(\sim\) growth in \(O(p(n))\)
- Maximum index \(i\) used in \(\varphi_{p,M,w}\) is \(dp(n)\), that is in \(O(p(n))\)
- Therefore: \(\varphi_{p,M,w}\) 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 10.4: 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 for polynomially space-bounded TMs.

First-Order Logic is PSpace-complete

Theorem 10.8: 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.
Checking FOL Models in Polynomial Space (Sketch)

```plaintext
01 Eval(ϕ, I) {
02    switch (ϕ) :
03    case p(c₁, ..., cₙ) : return ⟨c₁, ..., cₙ⟩ ∈ p
04    case ¬ψ : return NOT Eval(ψ, I)
05    case ψ₁ ∧ ψ₂ : return Eval(ψ₁, I) AND Eval(ψ₂, I)
06    case ∃x.ψ :
07        for c ∈ ∆I :
08            if Eval(ψ[x ↦ c], I) : return TRUE
09        // eventually, if no success:
10           return FALSE
11    }
```

- We can assume ϕ only uses ¬, ∧ and ∃ (easy to get)
- We use ∆ to denote the (finite!) domain of I
- We allow domain elements to be used like constants in the formula

Hardness of FOL Model Checking

Given: a QBF ϕ = Q₁X₁. ··· QₗXₗ. ψ

FOL Model Checking Problem:

- Interpretation domain ∆ := {0, 1}
- Single predicate symbol true with interpretation true = {1}
- FOL formula ϕ' is obtained by replacing variables in input QBF with corresponding first-order expressions:
  O₁X₁. ··· OₗXₗ.ψ[X₁ ↦ true(x₁), ..., Xₗ ↦ true(xₗ)]

Lemma 10.9: ⟨I, ϕ’⟩ ∈ FOL Model Checking if and only if ϕ ∈ True QBF.

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 10.10: Answering SQL queries over a given database is PSpace-complete.
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 $\varphi$ 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 $\varphi$ true. Otherwise, Player 2 wins.

Deciding the Formula Game

**Theorem 10.11:** 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 for $X_3$, … such that $\varphi$ 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 10.12: Generalised Geography is PSpace-complete.

Proof:

1. Geography ∈ 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 ≤p Geography.

Geography is PSpace-hard: Example

We consider the formula ∃X. ∀Y. ∃Z. (X ∨ Z ∨ Y) ∧ (¬Y ∨ Z) ∧ (¬Z ∨ Y)

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

- Some more remarks on games
- Logarithmic space
- Complements of space classes