Technische Universität München (I7) **Summer term 2019**
Prof. J. Křetínský / Dr. M.Raskin **08.05.2019**

Solution

Computational Complexity – Homework 5

Discussed on 14.05.2019.

Exercise 5.1

(a) Show that for any $L \in \mathbf{PSPACE}$ there is single-tape TM M (which may also write on its input tape) which decides L also in polynomial space.

(b) Show that it is **PSPACE**-complete to decide if a given word w can be derived by a given context-sensitive grammar G, i.e.,

$$\textsc{ConSens} := \{\langle G, w\rangle \mid \text{if } G \text{ is a context-sensitive grammar and } w \in L(G)\}.$$

Solution:

(a) As we are allowed to used polynomial space, we can compress all k tapes into a single tape using a vector alphabet $(\Gamma \cup \hat{\Gamma})^k$ where $\hat{\Gamma}$ is used to encode the positions of the original heads. We then can simulate a single step of the original machine within a bounded number of "oblivious" macro-steps: scan the single tape from left to right and back again and remember the symbols necessary for determining the next step of the original machine. Then change in a second scan from left to right and back again the tape content. The new machine will use at most the space used by the original machine.

(b) We first show that $\textsc{ConSens}$ is in **PSPACE**:

Let $G = (\Sigma, V, P, S)$ be a context-sensitive grammar with Σ the alphabet/terminals, V the set of variables/nonterminals, P the set of productions, and $S \in V$ the start symbol. By definition of context-sensitive grammar, every rule is of the form $\alpha A\beta \to \alpha\gamma\beta$ with $\alpha, \beta \in (\Sigma \cup V)^*$, $A \in V$, and $\gamma \in (\Sigma \cup V)^+$, i.e., $|\alpha A\beta| \leq |\alpha\gamma\beta|$.

A derivation in G is any finite sequence $\omega_1\omega_2 \ldots \omega_l$ such that $\omega_1 = S$ and ω_i can be rewritten to ω_{i+1} by means of some production of P. Then $L(G)$ is the set of all words $x \in \Sigma^*$ for which there exists a derivation ending with x. Note that the length of the ω_i is monotonocially increasing, i.e., $|\omega_i| \leq |\omega_{i+1}|$. This means that in linear space we can nondeterministically guess a derivation of x as every ω_i has length at most $|x|$: given ω_i construct some ω_{i+1} by nondeterministically applying a production; if $|\omega_{i+1}| > |x|$, reject $|x|$; otherwise go on until $\omega_i = x$. Note that this NDTM might not terminate. This is not a problem as there are only exponentially many different configurations, so we can add some counter (which needs space polynomial in $|x|$) for forcing termination the NDTM if too many steps have been made.

PSPACE-completeness:

Let M be a TM deciding some language L in space $s(n)$ where s is some polynomial. (Note that by Savitch's theorem we may indeed assume that M_L is deterministic.) By (a) we may also assume that M has a single tape. As M decides L every computation ends in either $(q_{\text{halt}}, \rhd 0)$ or $(q_{\text{halt}}, \rhd 1)$ with wlog. the head on the start symbol.

Given M and x we want to construct in polynomial time a context-sensitive grammar $G_{M;x}$ and word w_x such that

$$M \text{ accepts } x \text{ iff } w_x \in L(G_{M;x}).$$

We define $G_{M;x}$ as follows:

Every transition of M is of the form $\delta(q, a) = (q', b, \rightarrow)$. We translate this into the rule $ub(v, q') \rightarrow u(a, q)v$ for every possible $u, v \in \Gamma$ where Γ is the tape alphabet of M, i.e., a production corresponds to undoing a transition of M where we remember in the nonterminals the state, head position and symbol read by the head. These rules can be written in time polynomial in the description of M. Additionally, we add rules $S \rightarrow \rhd_{q_{\text{halt}}} 1B$ and $B \rightarrow \Box | \Box B$. A derivation of the grammar then obviously corresponds to the reverse of an accepting run of M. The grammar then has as terminals the band alphabet Γ of M. The nonterminals are given by $\Gamma \times Q$.

(In order to handle boundary cases one need to add additional left and right end symbols $ and # which never are overwritten.)

We therefore set $w_x = \rhd x \Box^{s(|x|)}$. As the computation of M on x needs at most $s(|x|)$ space, we have $x \in L$ iff $w_x \in L(G_{M;x})$.

Exercise 5.2

Prove that **EXPTIME** = **APSPACE**.

[*Hint:* For the $\subseteq$ direction consider breaking the work tape(s) into exponentially many segments which are then independently simulated in polynomial space. Use alternatation to coordinate these simulations.]

Remark: We can also show that **P** = **AL** (alternating logarithmic space).

Solution: In order to get **APSPACE** $\subseteq$ **EXPTIME** we note that a polynomial-space bounded machine has at ost exponentially many configurations and that its run-tree can thus be exhaustively explored (via a depth-first search) in exponential time. Without loss of generality we may assume that M has only one work-tape.

We thus concentrate on the harder direction: **EXPTIME** $\subseteq$ **APSPACE**. Suppose that we have a TM M that terminates in at most $c.2^{n^p}$-steps on an input x of length n.

In particular this means that M must also work in space bounded by $c.2^{n^p}$. We can thus think of the tape of M as being divided into 2^{n^p} many *segments* of constant length c. Each of these segments can be assigned an *address* in $[1, 2^{n^p}]$ from left-to-right.

We construct an alternating machine $\hat{M}$ whose configurations (plus some extras implicit in the description of existential and universal branching of the machine) have one of the two following forms:

$$(q, P, A, w) \quad \text{and} \quad (P', A', w', m \,|\, q, P, A, w)$$

which we refer to as *main* configurations and *check* configurations respectively.

The main configurations should be viewed as simulating a configuration of M. They consist of the following data:

(a) A control-state q of M,

(b) A program counter P that keeps track of how many steps in the simulation have so far been made,

(c) A 'segment pointer' A that indicates the address of the segment of M's tape current being simulated,

(d) A word w of constant length c indicating the contents of the segment of M's tape (including the head position) at address A at step P.

A *check* configuration contains the following data:

(a) The data to the right of $|$ is the same as in a main configuration.

(b) P', A' and w' are respectively (binary representations of) two numbers bounded by $c.2^{n^p}$ and a word of length c over the tape alphabet of M (plus head position marker). These should be interpreted as specifying an assertion that needs to be checked: "Is it the case that at step P' the segment with address A' has content w'"?

(c) The element m is a Boolean value that is set to true when the address A' current contains content w' and to false otherwise.

A main configuration can clearly simulate M faithfully until such a point that it must simulate moving the head either to the left or to the right of A (i.e. to $A' = A - 1$ or $A' = A + 1$). In this case it must *guess* the content w' of the tape at address A'. (Such a guess can be made with an existential ($\exists$) transition).

This guess w' needs to be verified. In order to do this, M makes a $\forall$ transition spawning the next main configuration as well as a 'check' configuration to verify the guess:

$$(q', P+1, A', w') \quad \text{and} \quad (P, A', w', m \,|\, q_0, 1, 1, \Box^c)$$

where m is set to true iff $w' = \Box^c$, otherwise it is set to false, and where q_0 is the initial state of M. Note that we are querying the content of the segment with address A' at step P. This is OK because its content must be the same at both step $P+1$ and step P because at step P the head of M was in a different segment.

A check configuration works in the same way to a main configuration on the right hand-side of the $|$ symbol. In particular it spawns new check configurations when it needs to simulate entering a new address (of course this time two check configurations will be spawned instead of a main and a check configuration).

The difference is that it must maintain the expected invariant for m. This is, however, trivial. The value of m should not change when $A \neq A'$. When $A = A'$, after each modification of w, w can be changed to w' and m set to true if $w = w'$ and false otherwise.

A check configuration halts when $P = P'$ and accepts if m is true, otherwise it rejects. Note that when a check configuration whose first component is P' spawns a check configuration to verify a guess, this new configuration will have first component $P'' < P'$. This ensures that the run tree is indeed well-founded (checking terminates).

Termination of the branch of the run tree consisting of main configurations can be ensured by checking that the P counters never exceed $c.2^{n^p}$, and this can also be done in polynomial space.

Note that $\hat{M}$ operates in polynomial space since all of the counters consume only $c.n^k$ space and all other components of configurations use only constant space.

Exercise 5.3

We will revisit two-player graph games, but this time we will not bound the number of moves in a play, and even allow the number of moves to be infinite.

A *game graph* is a structure $\langle V, E, V_0, V_1, v\rangle$ where $\langle V, E\rangle$ is a finite directed graph, and V_0, V_1 is a partition of the vertices V. Moreover $v \in V$ is the *initial node*.

Consider a sequence of nodes $(u)_{u\in I}$ where $I \subseteq \mathbb{N}$ is a downward closed index set (which may or may not be infinite) for the sequence. Such a sequence is called a *partial play* if (i) $u_0 = v$, and (ii) $(u_i, u_{i+1}) \in E$ for all $i+1 \in I$. A partial play is called a *play* if either $I = \mathbb{N}$, or it is a finitely long play $u_0, \dots, u_k$ such that there is no edge $(u_k, u) \in E$ for any $u \in V$.

Two players (player 0 and player 1) between them construct a partial play. The partial play begins with v. If a partial play $v_0, \dots, v_i$ has been constructed, and $v_i \in V_j$, and there exists $u \in V$ such that $(v_i, v) \in E$, then player j *must* choose the next node v_{i+1} in the partial play such that $(v_i, v_{i+1}) \in E$. The partial play is extended no further if no such move exists.

Thus after either finitely many or infinitely many moves the two players will have constructed a partial play that is a play.

We consider three different types of game that are distinguished by their *winning conditions* W. Given a play σ, we write $Occ(\sigma)$ for the set of nodes occurring at least once in σ, and $Inf(\sigma)$ for the set of nodes occurring infinitely often in σ (which will in particular be empty if σ is only finitely long).

- In a *reachability game* $W \subseteq V$ and player 0 wins a play σ if $W \cap Occ(\sigma) \neq \emptyset$.
- In a *Rabin game*, W is a set of pairs of the form (F, I) where $F, I \subseteq V$. Player 0 wins the play σ if there exists $(F, I) \in W$ such that $F \cap Inf(\sigma) = \emptyset$ and $I \cap Inf(\sigma) \neq \emptyset$.
- In a *Müller game*, $W = \langle C, \mathcal{C}, \chi \rangle$ where C is a finite set of colours, $\mathcal{C} \subseteq 2^C$, and $\chi : V \to C$. Player 0 wins a play σ if $\chi(Inf(\sigma)) \in \mathcal{C}$.

The decision problem associated with a particular type of game is the set containing elements $\langle \mathcal{G}, W \rangle$ where $\mathcal{G}$ is a game graph, W is an appropriate winning condition, and Player 0 can play in such a way that a play winning for Player 0 always results regardless of how Player 1 moves.

(a) Prove that the decision problem for reachability games is **P**-hard. (Remember that logarithmic space reductions must be used for this). For this take it as given that $\mathbf{AL} = \mathbf{P}$.

[*Remark:* It is possible to see that the version of reachability games defined in the previous problem sheet are equivalent to those defined above. Thus in fact reachability games are **P**-complete.]

(b) Prove that the decision problem for Rabin games is **NP**-complete.

[*Hint:* For hardness reduce from 3-SAT. Make Player 0 'prove' that they know some satisfying assignment. Allow Player 1 to 'interrogate' player 0's knowledge of such an assignment. Using the winning condition to ensure that for *some* literal player 0 is *eventually* consistent should suffice to allow Player 1 to successfully catch out Player 0 if no satisfying assignment exists.]

(c) Prove that the decision problem for Müller games is **PSPACE**-complete.

[*Hint:* For hardness reduce from QBF. Observe that Rabin conditions can be (in polynomial time) translated into Müller conditions. Note further that the *complement* of a Rabin condition can also be so translated. You might also find it helpful to work with a slight generalisation of Müller games allowing one to have a Müller game equivalent of adding quantifiers to the front of a propositional formula.]

<u>Exercise 5.4</u>

You have seen that 2SAT is in **NL**. Show that 2SAT is also **NL**-hard.

Solution: Since *REACHABILITY* is NL-hard and we know that NL is closed under complement, it suffices to show that there exists a logspace reduction from $\overline{REACHABILITY}$ to 2SAT. Suppose that we are given a graph $\mathcal{G} = \langle V, E \rangle$, an initial vertex v_0 and a target vertex v_f. From this we assign a variable x_v to each node in V and then construct $\phi_{\mathcal{G}} := \bigwedge_{(v_1,v_2)\in E}(x_{v_1} \to x_{v_2})$ (where $x_{v_1} \to x_{v_2}$ is $\neg x_{v_1} \vee x_{v_2}$). Finally we take the result of the reduction to be $\psi_{\mathcal{G}} := x_{v_0} \wedge x_{v_f} \wedge \phi_{\mathcal{G}}$.

$\psi_{\mathcal{G}}$ is a 2SAT instance and can be constructed in logspace (in the size of the reachability problem instance). Indeed the construction can be carried out in constant space: we can reuse the node IDs as variable IDs and in particular $\phi_{\mathcal{G}}$ is just a rewriting of E (copying node IDs from a pairs (v_1, v_2) and adding the appropriate Boolean operators.

It just remains to check that v_f is NOT reachable from v_0 iff $\psi_{\mathcal{G}}$ is SAT. For this it suffices to show that (i) if a valuation satisfies $x_{v_0} \wedge \phi_{\mathcal{G}}$ it must set x_v to true for all v reachable from v_0, and (ii) if a node v is unreachable from v_0, then there exists a valuation satisfying $x_{v_0} \wedge \phi_{\mathcal{G}}$ that sets x_v to false for every unreachable node v.

To prove (i) argue by induction on the number of steps to reach v from v_0. To prove (ii) take the valuation that sets x_v to true if v is reachable and false otherwise. Assume for contradiction that this is not a satisfying valuation. Since v_0 is trivially reachable it follows that there is a clause $x_{v_1} \rightarrow x_{v_2}$ in $\phi_{\mathcal{G}}$ such that x_{v_1} is set to true but x_{v_2} is set to false. But if this clause exists, $(v_1, v_2) \in E$ and by the definition of valuation v_1 is reachable whilst v_2 is not, which is a contradiction.

Exercise 5.5

Show that deciding the inequivalence of context-free grammars over one-letter terminal alphabet is Σ_2^p-hard. You can make use of Σ_2^p-hardness of integer expression inequivalence.

What does it imply for the equivalence problem?

Exercise 5.6

Under the assumption that $3\text{SAT} \leq_p \overline{3\text{SAT}}$ show that $\mathbf{NP} = \mathbf{PH}$.

Solution: If $3\text{SAT} \leq_p \overline{3\text{SAT}}$, then $\mathbf{NP} = \text{co}\mathbf{NP}$, i.e., $\mathbf{\Sigma}_1^p = \mathbf{\Pi}_1^p$. Consider now any $L \in \mathbf{\Sigma}_2^p$. We have

$$x \in L \text{ iff } \exists u \in \{0,1\}^{p(|x|)} \forall u \in \{0,1\}^{q(|x|)} : M(x, u, v) = 1.$$

The language

$$L_1\{(x, u) \mid \forall v : M(x, u, v) = 1\}$$

is then in co**NP** and, thus, in **NP**, i.e., we find a TM M' and a polynomial r, s.t.,

$$(x, u) \in L_1 \text{ iff } \exists v \in \{0,1\}^{r(|x|+|u|)} : M'(x, u, v) = 1.$$

As $|u| = p(|x|)$, we may assume that $|v| = r(|x|)$ by adjusting r.

Hence,

$$x \in L \text{ iff } \exists uv \in \{0,1\}^{p(|x|)+r(|x|)} : M'(x, uv) = 1,$$

i.e., $L \in \mathbf{NP}$.

So, $\mathbf{\Sigma}_2^p \subseteq \mathbf{NP} = \text{co}\mathbf{NP}$. Similarly, $\mathbf{\Pi}_2^p \subseteq \mathbf{NP} = \text{co}\mathbf{NP}$.

Using induction, one now shows that $\mathbf{NP} = \mathbf{PH}$.

Exercise 5.7

Apart from the certificate definition and the alternative bounded alternating Turing machine characterization, there is one more standard characterization of the polynomial hierarchy via *oracles*.

For a language L, an oracle machine M^L is a Turing machine which can moreover do the following kind of computation steps. It can write down a word w on a special tape and ask whether $w \in L$ and it immediately receives the correct answer. One can also talk about this machine even when the oracle is not specified, then we write $M^?$.

Example: In Exercise 3.4 (a), you have constructed an example of M^{SAT} where $M^?$ is a polynomial time TM.

- Prove or disprove: for every $M^?$, if $A \subseteq B$ then $\mathcal{L}(M^A) \subseteq \mathcal{L}(M^B)$.
- Prove or disprove: if $A \subseteq B$ then $\mathbf{P}^A \subseteq \mathbf{P}^B$ (as classes).

The polynomial hierarchy can be defined inductively setting $\Sigma_0^p = \Pi_0^p = \mathbf{P}$ and

$$\Sigma_{i+1}^p = \mathbf{NP}^{\Sigma_i^p}$$

$$\Pi_{i+1}^p = \textbf{co-NP}^{\Sigma_i^p}$$

where A^B is the set of decision problems solvable by a Turing machine in class A with an oracle for some complete problem in class B.

- Show this yields the same hierarchy as the original definition.

One can also define $\Delta_{i+1}^p = \mathbf{P}^{\Sigma_i^p}$ and show that $\Delta_{i+1}^p \subseteq \Sigma_{i+1}^p \cap \Pi_{i+1}^p$ and it contains all languages expressible as Boolean combinations (unions, intersections, complements) of languages of Σ_i^p and Π_i^p.

- What is the relationship of these classes to $\mathbf{DP} = \{L \mid \exists M, N \in \mathbf{NP} : L = M \setminus N\}$?