# Exercise sessions #5
# Decidability and Complexity Theory

prof. B. Demoen W. Van Onsem

May 2015

**Exercise 1.** A *useless state for a Turing machine* $M$ is a state of $M$ such that for all input, the state doesn't play any part. Consider $U = \{\langle M, q\rangle \mid M \text{ is a TM and } q \text{ a useless state of } M\}$. Show that $U$ is undecidable.

*Answer.* One can construct a reduction from the *emptiness problem* $E_{\text{TM}}$ to the useless state problem: for a string $\langle M\rangle$ with $M$ a Turing machine, if $U$ was decidable, we could decide it by extracting the reject state $q_r$ from $M$ and the decide $\langle M, q_a\rangle$ for $U$. If $U$ rejects, we reject and vice versa.

This reduction is correct because if $q_a$ is a useless, it means that for no string, $M$ will accept that string. This means that the language $L_M$ decide by $M$ is empty and thus we should accept $\langle M\rangle$ as an element of the emptiness language. The same holds vice versa.

Since the reduction is correct, but the emptiness problem is undecidable, the only possible explanation is that $U_{\text{TM}}$ is undecidable as well. $\diamond$

**Exercise 2.** Show that $R_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA that accepts } w^{\mathcal{R}} \text{ iff the DFA accepts } w \text{ as well for all input } w\}$ is decidable. Is $R_{\text{TM}} = \{\langle M\rangle \mid M \text{ is a Turing machine such that for every input } w, M \text{ accepts } w \text{ iff } M \text{ accepts } w^{\mathcal{R}}\}$ decidable as well? Prove your answer.

*Answer.* We've seen in exercise session #3 a way to construct a DFA $A^{\mathcal{R}}$ for a given DFA $A$ that decides the reverse language[1]. Furthermore in the course text, we've seen an algorithm that construct the minimum DFA $M'$ for a given DFA $M$, such DFA is unique except for state isomorphism. One can thus construct the reverse DFA and minimize both DFA's. If the two DFA's decide the same language, the minimized DFA's should be equal (except for isomorphism). Now we can check whether the two DFA's are isomorphic. If that is the case, we accept, otherwise we reject.

We can check isomorphism exhaustively by enumerating over all possible mapping between a state of the first DFA and a state of the second. But a smarter algorithm can take the fact that DFA's are deterministic into account: for every character there is exactly one next state. One can thus construct a mapping straightforward and efficient.

$R_{\text{TM}}$ is undecidable. This is a consequence of *Rice's theorem.* The language is non-trivial since it is possible to construct a Turing machine (for instance deciding the language $\{ab\}$) where the property doesn't hold, and construct Turing machine (for instance a machine deciding the language $\{a\}$) where the property does hold. Furthermore the language is language-invariant: only properties of the language itself are constrained: the concrete machine (implementation of the language) does not play a part in $R_{\text{TM}}$. $\diamond$

**Exercise 3.** Show that the 'Post correspondence problem' is decidable for an alphabet $\Sigma = \{a\}$. (Or in other words: show that for a collection of domino pieces of the form $\left[\frac{a^m}{a^n}\right]$ with $n, m \in \mathbb{N}$, it is decidable whether there exists a correspondence.)

[1] Not to be confused with the complement language.

*Answer.* One can decide this as follows: there should exist a domino piece such that $m \geq n$ and a domino piece - possible the same - with $m \leq n$.

If the two conditions hold for the same domino piece, evidently one can repeat that domino piece an arbitrary number of times, for each repetition, the number of $a$'s of the upper part will be the same as the number of $a$'s of the lower part.

If the conditions correspond to two different pieces, we will denote the difference between $m$ and $n$ for the first piece as $\Delta$. The difference between $n$ and $m$ for the first piece will be denoted with $\Gamma$. Evidently both numbers are strictly positive. In that case we can repeat the first piece $\Gamma$ times and the second piece $\Delta$ times. In that case the total difference between the upper and the lower part will be zero. $\diamond$

**Exercise 4.** Show that there exists an undecidable language that is a subset of $1^\star$.

*Answer.* Since every Turing machine can be *serialized* by a sequence of characters, and thus be mapped to a unique integer, the number of Turing machines is $\#\mathbb{N} = \aleph_0$. The number of strings contained in $1^\star$ is the same as the number of integers, since for every integer, we can construct the string with the same number of repetitions of ones and vice versa. Thus $\#1^\star = \#\mathbb{N} = \aleph_0$. The number of subsets is thus equal to $\#\mathcal{P}(1^\star) = \aleph_1$. Since the number of subsets is greater than the number of Turing machines and every Turing machine decides at most one language, there exists at least one subset for which we cannot construct a Turing machine. $\diamond$

**Exercise 5.** Show that it is decidable for a given context-free grammar $G$ to determine whether that grammar can generate all strings of the form $1^*$.

*Answer.* Every context-free grammar has a *pumping length* $p$ with as upper bound the number of non-terminals. If every string $\left\{\epsilon, 1, 1^2, 1^3, \ldots, 1^{p+p!}\right\}$ is part of the context-free grammar, then every element of $1^\star$ is contained in the language described by the context-free grammar. The algorithm thus first counts the number of non-terminals of the grammar, then determines if all strings of ones up to length $p+p!$ are elements of the language described by the context-free grammar, then accept, otherwise reject. We now still need to prove that this holds, we can prove this by slightly modifying the pumping lemma. We prove that given for a context-free language $L$ with pumping length $p$ and the strings $\left\{\epsilon, 1, 1^2, 1^3, \ldots, 1^{p+p!}\right\}$ are all elements of $L$, then every string $1^l$ with $l > p + p!$ are elements of $L$ as well.

*Proof.* For $l$, we define two additional variables: $m = l - p \mod p!$, and $q = l - p - m$. As a result $l = p + m + q$. As a result $q$ is divisible by $p!$ and therefore divisible by every integer between 1 and $p$: since $m$ is the result of the modulo operation of $p!$ and $q$ is $l - p - m$, the modulo is subtracted. Furthermore $p \leq p + m \leq p + p!$: this holds since $m$ is the result of a modulo-operation of two positive numbers and since $m$ is modulo $p!$, $m$ cannot be greater than $p!$. Now we take a string $s = 1^{p+m}$, the length of this string is evidently greater than the pumping length. Now we consider every subdivision $s = uvxyz$ evidently $u, v, x, y, z \in 1^\star$. For these subdivision it holds that $0 < |vy|$ and $|vxy| \leq p$. We now define the length of $c = |vy|$. Now we prove that there exists an $i \in \mathbb{N}$ such that $uv^ixy^iz = 1^{p+m+i\cdot c} = 1^l$ (the string we wish to generate). As a result this means that $i \cdot c = q$ and $c \leq p$. Now since $q$ is divisible by $p!$, it is definitely divisible by $k$ with $k < p$. Thus we know such $i$ exists and therefore we can pump the string $1^l$. $\square$

$\diamond$

**Exercise 6.** Is a language $L$ that can be decided in *time complexity* $\mathcal{O}(\log_2 n)$ useful? Why (not)? Is a language with *memory complexity* $\mathcal{O}(\log_2 n)$ useful? Why (not)?

*Answer.* By looking at the definition of big-oh. A function $f(n) = \mathcal{O}(\log_2 n)$ means that $\exists c \in \mathbb{N}$ and $\exists n_0 \in \mathbb{N}$ such that $\forall n > n_0 : f(n) < c \cdot \log_2 n$. Since $f(n)$ is an upperbound on the number of steps the Turing machine can take to evaluate a string with length $n$, it means that from a certain length $n_0$, the Turing machine can take at most $c \cdot \log_2 n$ steps and thus read at most that many characters from the input tape. Since we know that there exists a number $n_1 \in \mathbb{N}$ for which it holds: $\forall n > n_1 : c \cdot \log_2 n < n$, it means that there is a point at which the Turing machine can no longer read the entire input. From that point, the additional characters are useless since the Turing machine cannot take a decision based

on these characters. Such languages thus don't make any sense since they contain information that is not useful.

The same doesn't hold for memory complexity $\mathcal{O}(\log_2 n)$: since memory can be reused, the Turing machine can take at most $\mathcal{O}\left(n^k\right)$ steps before getting stuck in an infinite loop. $\diamond$