# Theory of Computation Assignment no. 11

Goktug Saatcioglu

**(1)**

a. If $A \preceq B$ then there exists a truth preserving reduction from $A$ to $B$. Furthermore, if $B \preceq C$ then there exists a truth preserving reduction from $B$ to $C$. Thus, we know that there is an algorithm $M_1$ (i.e. Turing machine) that takes an input word $x$ and outputs a word $y$ such that if $x \in A$ then $y \in B$ and if $x \notin A$ then $y \notin B$. We also know that there is an algorithm $M_2$ (i.e. Turing Machine) that takes an input word $y$ and outputs a word $z$ such that if $y \in B$ then $z \in C$ and if $y \notin B$ then $z \notin C$. Combining $M_1$ and $M_2$ together, we can create an algorithm $M_3$ (i.e. Turing machine) that takes an input $x$ and first simulates $M_1$ to get an output $y$. Then $M_3$ will simulate $M_2$ on $y$ to get $z$. Firstly, since $M_1$ and $M_2$ is computable we know that $M_3$ is computable. Secondly, we see that if $x \in A$ then $y \in B$ then $z \in C$ (or if $x \notin A$ then $y \notin B$ then $z \notin C$) which implies that $x \in A$ then $z \in C$ (or $x \notin A$ then $z \notin C$) which proves that $\preceq$ is transitive. Thus, we conclude if $A \preceq B$ and $B \preceq C$ then $A \preceq C$.

b. If $A \preceq_p B$ then there exists a truth preserving polynomial time reduction from $A$ to $B$. Furthermore, if $B \preceq_p C$ then there exists a truth preserving polynomial time reduction from $B$ to $C$. Thus, we know that there is an algorithm $M_1$ (i.e. Turing machine) that takes an input word $x$ and outputs a word $y$ such that if $x \in A$ then $y \in B$ and if $x \notin A$ then $y \notin B$ in some time polynomial in the length of the input $x$ (where $|x| = n$). We also know that there is an algorithm $M_2$ (i.e. Turing Machine) that takes an input word $y$ and outputs a word $z$ such that if $y \in B$ then $z \in C$ and if $y \notin B$ then $z \notin C$ in some time polynomial in length of the input $y$ (where $|x| = o$). Combining $M_1$ and $M_2$ together, we can create an algorithm $M_3$ (i.e. Turing machine) that takes an input $x$ and first simulates $M_1$ to get an output $y$. Then $M_3$ will simulate $M_2$ on $y$ to get $z$. Firstly, since $M_1$ and $M_2$ is computable we know that $M_3$ is computable. Secondly, we see that if $x \in A$ then $y \in B$ then $z \in C$ (or if $x \notin A$ then $y \notin B$ then $z \notin C$) which implies that $x \in A$ then $z \in C$ (or $x \notin A$ then $z \notin C$). Thirdly, since $M_1$ runs in polynomial time in the length of the input $x$ (where $|x| = n$) and since $M_2$ runs in polynomial time in the length of input $y$ where $|y| = o$) we know that $M_3$ also runs in polynomial time in the length of the input $x$. All three points combined proves that $\preceq_p$ is transitive and we conclude if $A \preceq_p B$ and $B \preceq_p C$ then $A \preceq_p C$.

c. We can use the same reduction as the one for all $L \in RE$, $L \preceq ACCEPT$. Let $M_f$ be a reduction algorithm from $L$ to $ACCEPT$ and $M$ be a Turing machine such that $L(M) = L$. We know that $M$ exists since we assume that $L \in RE$. Then, $M_f$, given an input $w$, creates the Turing machine $M$ described above and then outputs the descriptions of $M$ and $w$ (i.e. $(\langle M \rangle, \langle w \rangle)$). We see that $M_f$ is a truth-preserving reduction from $L$ to $ACCEPT$ since if a word $w \in L(M)$ then $\langle M, w \rangle \in ACCEPT$ and if $w \notin L(M)$ then $\langle M, w \rangle \notin ACCEPT$. Furthermore, $M_f$ is computable since we can encode both $M$ and $w$ according to the encoding scheme discussed in class. Finally, we know that the runtime of $M_f$ is polynomial in the length of $w$ (where $|w| = n$) since $M$ is given (i.e. a constant) meaning we can encode both $M$ and $w$ in polynomial time. Thus, we conclude that for all $L \in RE$, $L \preceq_p ACCEPT$.

d. Let $A \in R$ and $B \subseteq \Sigma^*$ such that $B \notin \{\emptyset, \Sigma^*\}$. Then there exists a Turing machine $M$ that decides $A$ because we assume that $A \in R$ and there exists two words $w_1, w_2 \in \Sigma^*$ such that $w_1 \in B$ and $w_2 \notin B$ because we assume that $B \notin \{\emptyset, \Sigma^*\}$. To get a truth preserving reduction from $A$ to $B$ consider the following algorithm $M_f$ (i.e. Turing machine). $M_f$ given an input $w$ will simulate $M$ on $w$ where if $M$ accepts then $M_f$ outputs $w_1$ and if $M$ rejects then $M_f$ outputs $w_2$. Firstly, $M_f$

is computable since by assumption we know that $M$ is computable (because $A \in R$). Secondly, if $w \in A$ then $M$ will accept and $M_f$ outputs $w_1 \in B$ and if $w \notin A$ then $M$ will reject and $M_f$ outputs $w_2 \notin B$ which means that $M_f$ is a truth-preserving reduction from $A$ to $B$. Thus, we conclude that $A \preceq B$ for every two languages $A, B \in R$, as long as $B$ is not the empty language nor the language containing all words.

e. Let $A \in P$ and $B \subseteq \Sigma^*$ such that $B \notin \{\emptyset, \Sigma^*\}$. Then there exists a Turing machine $M$ that computes $A$ in polynomial time in the length of some input $w$ (where $|w| = n$) because we assume that $A \in P$. Furthermore, we know that there exists two words $w_1, w_2 \in \Sigma^*$ such that $w_1 \in B$ and $w_2 \notin B$ because we assume that $B \notin \{\emptyset, \Sigma^*\}$. To get a truth preserving reduction from $A$ to $B$ consider the following algorithm $M_f$ (i.e. Turing machine). $M_f$ given an input $w$ will simulate $M$ on $w$ where if $M$ accepts then $M_f$ outputs $w_1$ and if $M$ rejects then $M_f$ outputs $w_2$. Firstly, $M_f$ is computable in polynomial time in the length of input $w$ since by assumption we know that $M$ is computable in polynomial time in the length of input $w$ (because $A \in P$). Secondly, if $w \in A$ then $M$ will accept and $M_f$ outputs $w_1 \in B$ and if $w \notin A$ then $M$ will reject and $M_f$ outputs $w_2 \notin B$ which means that $M_f$ is a truth-preserving polynomial time reduction from $A$ to $B$. Thus, we conclude that $A \preceq_p B$ for every two languages $A, B \in P$, as long as $B$ is not the empty language nor the language containing all words.

**(2)** a. We begin by defining a Turing machine $N$ such that $L(N) = \Sigma^*$. We know that such a machine exists and is easy to create since we just need to accept regardless of input. Then consider the following algorithm $M_f$ that takes as input descriptions of a Turing machine $\langle M' \rangle$ and a word $\langle w' \rangle$:

$M_f(\langle M' \rangle, \langle w' \rangle)$:

1. Construct a TM $M$ that ignores its input $w$ and simulates $M'$ on $\lambda$.
   $M$ accepts if $M$ halts on $\lambda$.
2. Construct a TM $N$ that always accepts its input (i.e. $L(N) = \Sigma^*$).
3. Output $(\langle N \rangle, \langle M \rangle)$.

Now, if $M'$ halts on $\lambda$ then the language of $M$ is given by $\Sigma^*$ since $M$ accepts all words $w$. In other words, if $M'$ halts on $\lambda$ then $L(M) = \Sigma^*$ which then means that $L(N) \subseteq L(M)$. Conversely, if $M'$ does not halt on $\lambda$ then $M$ runs (implicitly) infinitely for all words $w$ meaning $L(M) = \emptyset$ and $L(N) \not\subseteq L(M)$. Thus, $M_f$ is a truth-preserving reduction from $HALT_\lambda$ to $CONTAINED\_TM$ and we conclude $HALT_\lambda \preceq CONTAINED\_TM$.

b. Again, we define the Turing machine $N$ but this time we change $L(N)$. Consider the algorithm $M_f$ that takes as input descriptions of a Turing machine $\langle M' \rangle$ and a word $\langle w' \rangle$ of length $n$ (i.e. $|w'| = n$):

$M_f(\langle M' \rangle, \langle w' \rangle)$:

1. Compute $n = |w'|$.
   Construct a TM $M$ that ignores its input $w$ and simulates $M'$ on $\lambda$ for $n$ steps.
   $M$ enters an infinite loop if $M'$ halts at any point during the simulation,
   otherwise $M$ accepts.
2. Construct a TM $N$ that accepts inputs with length greater than or equal to $n$ (i.e. accept if $|w'| \geq n$).
3. Output $(\langle N \rangle, \langle M \rangle)$.

Similarly to a., if $M'$ does not halt on $\lambda$ in $n$ steps then $L(N) \subseteq L(M)$ because $M$ halts on all $w$ with length greater than or equal to $n$. Conversely, if $M'$ does halt on $\lambda$ in $n$ steps then it would enter an infinite loop for all $w$ meaning $L(M) = \emptyset$ and $L(N) \not\subseteq L(M)$. Thus, $M_f$ is a truth-preserving reduction from $(HALT_\lambda)^c$ to $CONTAINED\_TM$ and we conclude $(HALT_\lambda)^c \preceq CONTAINED\_TM$.

**(3)** a. Let $\Sigma$ be the alphabet of at least two letters such that it is the alphabet for an instance of $PCP$, say $P$. We know that the strings of $P$ are in the form $((s_1, t_1), \ldots, (s_n, t_n))$ and we seek to create two context free grammars $G$ and $H$ which respectively generate the $s$ strings and $t$ strings along with the information about the indices of the strings used. For our grammars we use the alphabet $\Sigma$ along with $n$ words that are not in $\Sigma$ (so that we can avoid conflicts) to denote indices of strings. Let these $n$ words be in the form $l_1, \ldots, l_n$ and assume that they are not in $\Sigma$. Now, we need $G$ to create the $s$ strings in the form $s_i G l_i$ so that we can generate a $s$ string along with its corresponding index (given by $l$). Thus, the CFG $G$ is given by

$$G \to s_1 G l_1 \mid \ldots \mid s_n G l_n \mid s_1 n_1 \mid \ldots \mid s_n l_n.$$

Similarly, for $H$ we need to create the $t$ strings in the form $t_i H l_i$ so that we can generate a $t$ string along with its corresponding index (given by $l$). Thus, the CFG $H$ is given by

$$H \to t_1 H l_1 \mid \ldots \mid t_n H l_n \mid t_1 n_1 \mid \ldots \mid t_n l_n.$$

Now let $w$ be a word that describes $((s_1, t_1), \ldots, (s_n, t_n))$. We notice that if $w$ is in $P$ (i.e. $w \in P$) then there also exists an intersection of the grammars $G$ and $H$ which is not empty (i.e. $G \cap H \neq \emptyset$). Conversely, if $w \notin P$ then $G \cap H = \emptyset$. Thus, given an instance of $PCP$, the algorithm that generates the grammars $G$ and $H$ as described above is a truth-preserving reduction from $PCP$ to $INTERSECT\_CFL$ and we conclude $PCP \preceq INTERSECT\_CFL$.

b. We know that if $A \preceq B$ and $A \notin R$, then $B \notin R$ (which has been shown in class). Furthermore, from a. (above) we know that $PCP \preceq INTERSECT\_CFL$ which means that if $PCP \notin R$, then $INTERSECT\_CFL \notin R$. In fact, since we know that $PCP \notin R$ (as given by the question) and $PCP \preceq INTERSECT\_CFL$, we know that $INTERSECT\_CFL \notin R$.

**(4)** a. We need to give an algorithm that takes a graph $G$ and gives a new graph $G'$ such that if $G \in 3COL$ then $G' \in 4COL$ and if $G \notin 3COL$ then $G' \notin 4COL$. Furthermore, this algorithm must run in polynomial time in length of the input $G$. Let $R$ be an algorithm that takes as input a graph $G = (V, E)$ with $V$ vertices and $E$ edges (where the edges are described as sets of 2 vertices). Then $R$ creates a new graph $G' = (V', E')$ where $V' = V \cup \{w\}$ and $w$ is a new vertex (i.e. $w \notin V$) and $E' = E \cup \{\{w, v\} \mid v \in V\}$. In other words, $R$ takes the graph $G$ and creates a new vertex $w$ which is then connected to all vertices in $G$ by edges which gives us the output graph $G'$. Now, if $G$ has some three coloring then $G'$ has the same exact three coloring by using the coloring of $G$ and then $G'$ is four colorable by coloring the introduced vertex $w$ the fourth color. Conversely, if $G$ does not have a three coloring then $G'$ will not have a four coloring since the addition of the introduced vertex $w$ can not lead to a four coloring. The algorithm $R$ runs takes linear time in length of the input $G$ since it creates a new vertex $w$ and then goes over all vertices of $G$ to create the new edges for $G'$. Thus, $R$ (as described above) is a polynomial time truth-preserving reduction from $3COL$ to $4COL$ and we conclude $3COL \preceq_p 4COL$.

b. We know that $3COL \preceq_p 4COL$ as this was shown above in a. and the algorithm that performs the reduction $R$ runs in linear time. Now, if $4COL \in P$ then there is exists some Turing machine $B$ that computes $4COL$ in polynomial time. To compute $3COL$ in polynomial time we can take an input graph $G = (V, E)$ with $V$ vertices and $E$ edges and first run it through our reduction algorithm $R$. This will give a resulting graph $G' = (V', E')$ which is in $4COL$ if $G$ was in $3COL$ and which is not in $4COL$ if $G$ was not in $3COL$. We then use $B$ to check if $G'$ is in $4COL$ where if $G' \in 4COL$ then we can conclude that $G \in 3COL$ and if $G' \notin 4COL$ then we can conclude that $G \notin 3COL$. We know that $R$ runs in linear time in input $G$, where $|G| = n$, and $B$ runs in polynomial time in input $G'$ which means that our new algorithm runs in $n^k$ which is polynomial time. Thus, we conclude that if $4COL \in P$ then $3COL \in P$ since $3COL \preceq_p 4COL$. (Note: It is also generally true that if $B \in P$ and $A \preceq_p B$ then $A \in P$.)

**(5)** a. We need to give an algorithm that takes an undirected graph $G$ and gives a new directed graph $G'$ such that if $G \in HAM\_CYCLE$ then $G' \in D\_HAM\_CYCLE$ and if $G \notin HAM\_CYCLE$ then

$G' \notin D\_HAM\_CYCLE$. Furthermore, this algorithm must run in polynomial time in length of the input $G$. Let $R$ be an algorithm that takes as input an undirected graph $G = (V, E)$ with $V$ vertices and $E$ edges (where the edges are described as sets of 2 vertices). Then $R$ creates a new directed graph $G' = (V', E')$ where $V' = V$ and $E' = \{\{u, v\} \mid \{u, v\} \in E\}$. In other words, $R$ takes the graph $G$ and creates a new directed graph $G'$ where for each undirected edge between $u$ and $v$ in $G$ there are two directed edges $u \to v$ and $v \to u$ in $G'$. Now, if the undirected graph $G$ has a Hamiltonian cycle $u_1 \ldots u_n$ then we know that $V$ has $n$ vertices where $\{u_1, \ldots, u_n\} = V$ and there exists at least an edge between each vertex in $V$ such that for each $1 \leq i < n : \{u_i, u_{i+1}\} \in E$ and $\{u_n, u_1\} \in E$. This then implies that for each $1 \leq i < n : \{u_i, u_{i+1}\} \in E'$ and $\{u_n, u_1\} \in E'$ (by construction of $R$) meaning $u_1 \ldots u_n$ is also an Hamiltonian cycle of the directed graph $G'$. Conversely, if $G$ does not have an Hamiltonian cycle then $G'$ cannot have an Hamiltonian cycle since $G'$ is essentially $G$ with bi-directional edges. The algorithm $R$ takes polynomial time in length of the input $G$ since we traverse over all edges once and create two new edge pairs for $G'$ for each edge we see. Thus, $R$ (as described above) is a polynomial time truth-preserving reduction from $HAM\_CYCLE$ to $D\_HAM\_CYCLE$ and we conclude $HAM\_CYCLE \preceq_p D\_HAM\_CYCLE$.

b. We need to give an algorithm that takes a directed graph $G$ and gives a new undirected graph $G'$ such that if $G \in D\_HAM\_CYCLE$ then $G' \in HAM\_CYCLE$ and if $G \notin D\_HAM\_CYCLE$ then $G' \notin HAM\_CYCLE$. Furthermore, this algorithm must run in polynomial time in length of the input $G$. Let $R$ be an algorithm that takes as input a directed graph $G = (V, E)$ with $V$ vertices and $E$ edges (where the edges are described as sets of 2 vertices). Then $R$ creates a new undirected directed graph $G' = (V', E')$ where $V' = \{\{v_{in}\}, \{v\}, \{v_{out}\} \mid \{v\} \in V\}$ and $E' = \{\{u_{out}, v_{in}\}, \{v_{in}, v\}, \{v, v_{out}\} \mid \{u, v\} \in E\}$. In other words, $R$ takes the graph the directed graph $G$ and creates a new undirected graph $G'$ where for each directed edge between $u$ and $v$ in $G$ there is now an undirected edge that goes from the newly created $u_{out}$ vertex into $v_{in}$, $v_{in}$ goes into $v$ and $v$ goes to $v_{out}$. Now, if the directed graph $G$ has a Hamiltonian cycle $u_1 \ldots u_n$ then we know that $V$ has $n$ vertices where $\{u_1, \ldots, u_n\} = V$ and there exists at least an edge between each vertex in $V$ such that for each $1 \leq i < n : \{u_i, u_{i+1}\} \in E$ and $\{u_n, u_1\} \in E$. This then implies that for each $1 \leq i < n : \{(u_i), u_{i+1}\} \in E'$ and $\{u_n, u_1\} \in E'$ (by construction of $R$) since we now have the Hamiltonian cycle $(u_1)_{in}(u_1)(u_1)_{out} \ldots (u_n)_{in}(u_n)(u_n)_{out}$ in $G'$. Conversely, if $G$ does not have an Hamiltonian cycle then $G'$ cannot have an Hamiltonian cycle since $G'$ is essentially $G$ with $3n$ vertices that has the same "structure" as $G$. The algorithm $R$ takes polynomial time in length of the input $G$ since we traverse over all vertices once to create the three new vertices and then traverse over all edges once to create the relevant edges in $G'$ for each edge we see. Thus, $R$ (as described above) is a polynomial time truth-preserving reduction from $D\_HAM\_CYCLE$ to $HAM\_CYCLE$ and we conclude $D\_HAM\_CYCLE \preceq_p HAM\_CYCLE$.