CoCo - Assignment 4

Johannes Agerskov, Mads Friis Frand-Madsen, and Sifan Huang

Dated: March 7, 2021

6.23

Let t be the computable function that, in a TM describtion interchange the accept and reject states. We give a concrete example of a fixed point of t. We notice that the proof fixed point version of the recursion theorem (Theorem 6.8) gives a construction of a fixed point, F, which obtain its own describtion and then simulate the TM described by $t(\langle F\rangle)$. However, in this specific example of t, migth actually conclude that F is the Turring machine that neither accepts nor rejects any string. This is easily seen by the fact that if w is string such that F accepts w, then the TM, G, described by $T(\langle F\rangle)$ will reject w. But since F simulates exactly G, we conclude that F also rejects w which is a contradiction. Thus F cannot accept any string. Swapping *accept* and *reject* in the previous argument shows the F also cannot reject any string. Conversely, if F is a TM such that it neither accept nor reject any state, $t(\langle text\rangle)$ obviously describes an equivalent TM. Thus any TM, F, which never halts on any string is a fixed point of t.

Therefore we describe F by $F = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accepts}}, q_{\text{reject}})$, where $Q = \{q_0, q_{\text{accepts}}, q_{\text{reject}}\}$. $\Sigma = \{1\}$, $\Gamma = \{\sqcup, 1\}$, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is defined by $\delta(q_0, 1) = (q_0, 1, R)$ and $\delta(q_0, \sqcup) = (q_0, \sqcup, L)$, and δ need not be defined, or rather could be defined to be anything, on the accept and reject state, since the machine halts there anyway. Thus, F loops forever by going to the last 1 in any input string and then going back and forth between $\sqcup$ and 1 at the end of the string. On the empty string it loops by trying to move left on the tape, but stays at the first tape position. This TM is a fixed point of t be the above argument.

6.6

For each $m > 1$ let $\mathcal{Z}_m = \{0, 1, 2, ..., m-1\}$, and let $\mathcal{F}_m = (\mathcal{Z}_m, +, \times)$ be the model whose universe is $\mathcal{Z}_m$ and that has relations corresponding to the $+$ and $\times$ relations modulo m. We show in the following that $\text{Th}(\mathcal{F}_m)$ is decidable.

Proof. We construct an TM that decides $\text{Th}(\mathcal{F}_m)$. We know from Theorem 6.12 that $\text{Th}(\mathcal{N}, +)$ is decidable. Let S be the TM that decides $\text{Th}(\mathcal{N}, +)$. We can clearly use this TM to show that $\text{Th}(\mathcal{Z}_m, +)$ is decidable. We can design, M_1, to be a TM that has S as a subroutine, such that M_1 can add modulo m. For any formula ϕ, M_1 simply does the following:

1. identify all substrings of ϕ of the form $x_1 + ... + x_n = y$.
2. Generate a new substring $\tilde{\phi}$ by replacing every substring of ϕ of the form $x_1+...+x_n = y$, by $\bigvee_{i=1}^{n}(x_1 + ... + x_n = y + \underbrace{m + m + ... + m}_{i \text{ terms}})$, where $\bigvee_{i=1}^{n} \varphi(i) = \varphi(1) \vee ... \vee \varphi(n)$, for any formula $\varphi(i)$ depending on i.
3. Run S on $\tilde{\phi}$, if true *accept.* If false *reject*

Thus M_1 accepts a formula if and only if it is true modulo m. Clearly we can never have $z - m \geq x_1 + ... + x_n \geq z + \underbrace{m + m + ... + m}_{n \text{ terms}}$ for $x, y \in \mathcal{Z}_m$, so this TM desides $\mathrm{Th}(\mathcal{Z}_m, +)$. Thus we have shown that $\mathrm{Th}(\mathcal{Z}_m, +)$ is reducible to $\mathrm{Th}(\mathcal{N}, +)$, so from Theorem 6.12 it follows that $\mathrm{Th}(\mathcal{Z}_m, +)$ is decidable. Now that $\mathrm{Th}(\mathcal{Z}_m, +, \times)$ is decidable can be shown by the fact that $\mathrm{Th}(\mathcal{Z}_m, +, \times)$ reduces to $\mathrm{Th}(\mathcal{Z}_m, +)$. Given a decider M_1 for $\mathrm{Th}(\mathcal{Z}_m, +)$, we design the TM, M that desides $\mathrm{Th}(\mathcal{Z}_m, +, \times)$ it can be described as: On formula φ follows

1. Identify all substrings of φ of the form, $x \times y$ where y is a free variable, and rewrite the substring to $\underbrace{x + x + x + ...}_{y \text{ times}}$.
2. Indentify any formula that has a substring of the form, $x \times y$ where y is not a free variable. Then we consider two cases:
   (a) φ contains a substring of the form $\forall y$, *i.e.* can be written $\forall y[\phi(y)]\Phi$, for some formula $\phi(y)$ depending on y and some string Φ, which is just the rest of φ. In this case M discards the quantifier of y and generate the formula

   $$\bigwedge_{i=0}^{m-1} \phi(i)\Phi, \tag{0.1}$$

   . where $\bigwedge_{i=1}^{N} \phi(i) = \phi(1) \wedge \phi(2) \wedge ... \wedge \phi(N)$

   (b) φ contains a substring of the form $\exists y$, *i.e.* can be written $\exists y[\phi(y)]\Phi$, for some formula $\phi(y)$ depending on y and some string Φ, which is just the rest of φ, in which case M discards the quantifier of y and generate the formula

   $$\bigvee_{i=0}^{m-1} \phi(i)\Phi, \tag{0.2}$$

   . where $\bigvee_{i=1}^{N} \phi(i) = \phi(1) \vee \phi(2) \vee ... \vee \phi(N)$

   this process is repeated iteratively til no products are left.
3. Run M_1 on the generated formula, if true *accept*, else *reject.*

Now since $\mathcal{Z}_m$ is finite, there are always only finitely many terms in the above description. Thus M will always halt, making it a decider. Furhtermore, it is obvious by construction, that

M accepts any formula if and only if it is true in the usual model of $\mathcal{Z}_m$ with addition and multiplication modulo m. $\square$

Proof. From Theorem 6.12 we may infer the existence of a Turing machine T deciding Th$(\mathcal{N}, +)$. Define a Turing machine $T_{m,+}$ for deciding Th$(\mathcal{Z}_m, +)$ as follows:

1) Given a formula $\psi = Q_1x_1 \dots Q_kx_kR_1y_1 \dots R_ly_l[x_1 + \dots + x_n = y_1 + \dots y_l]$ construct the formula

$$\widetilde{\phi} = \left(\bigvee_{i=0}^{m-1}\left(x_1 + \dots + x_n + \underbrace{m + \dots + m}_{i \text{ terms}} = y_1 + \dots y_l\right)\right) \vee \left(\bigvee_{i=0}^{m-1}\left(x_1 + \dots + x_n = y_1 + \dots y_l + \underbrace{m + \dots + m}_{i \text{ terms}}\right)\right)$$

2) Run T on $Q_1x_1 \dots Q_kx_kR_1y_1 \dots R_ly_l[\widetilde{\phi}]$ and return the outcome of T.

Evidently, the Turing machine above decides Th$(\mathcal{Z}_m, +)$, and so we move on to constructing a decider for Th$(\mathcal{Z}_m, +, \times)$.

1) Given a formula $Q_1x_1 \dots Q_kx_kR_1y_1 \dots R_ly_l[\psi]$, identify a substring of the form $x_i \times y_j$. If there are no such substrings, go to 4)

2) If $Q_i = \forall$, replace $x_i \times y_j$ by
$$\bigwedge_{i=0}^{m-1} \underbrace{(y_j + \dots y_j)}_{i \text{ terms}},$$
and return to 1).

3) If $Q_i = \exists$, replace $x_i \times y_j$ by
$$\bigvee_{i=0}^{m-1} \underbrace{(y_j + \dots y_j)}_{i \text{ terms}},$$
and return to 1).

4) Run $T_{m,+}$ on $R_1y_1 \dots R_ly_l[\psi]$ and return the outome of $T_{m,+}$.

It is clear the Turing machine T_m halts, and likewise it is constructed to accept exactly Th$(\mathcal{Z}_m, +, \times)$. $\square$

6.7

Let A and B be any two languages, we show that then there exist a language J such that $A \leq_T J$ and $B \leq_T J$.

Proof. Let $\aleph$ be a special symbol that is not in the alphabet of A or B. Construct the the language $\aleph A$ *i.e.* all string of the form $\aleph w$ for $w \in A$. Define the language $J = \aleph A \cup B$. We then construct an oracle TM, M_A^J with an oracle for J that desides A. M_A^J acts as follows in input w.

1. Construct the string $\aleph w$.

2. Query the oracle for J if $\aleph w \in J$, if true, *accept*, if false, *reject*.

Clearly M_A^J accepts w if and only if $\aleph w \in J$ but since $\aleph w \notin B$, we conclude that $\aleph w \in J$ if and only if $\aleph w \in \aleph A$ which can be true if and only if $w \in A$. Thus M_A^J decides A, and A is reducible relative to J, *i.e.* $A \leq_T J$.
We now proceed by also constructing and oracle TM, M_B^J, that decides B. M_B^J acts as follows on input $w = w_1 \ldots$.

1. Check if $w_1 = \aleph$, if true, *reject*.

2. Query the oracle for J if $w \in J$, if true, *accept*, if false, *reject*.

Clearly M_B^J accept if and only if $w \in J$ but since $w \notin \aleph A$ we conclude that $w \in J$ if and only if $w \in B$. So M_B^J decides B, and B is reducible relative to J, *i.e.* $B \leq_T J$. This concludes the proof. $\square$

6.11

We show that the complement of EQ_{TM} is recognizable by a TM with an oracle for A_{TM}.

Proof. First we define the oracle TM, $S_{\langle G,F\rangle}^{A_{TM}}$, that, given two TMs G and F, decides the language $L(G)\Delta L(F) = (L(G) \cap L(F)^{\complement}) \cup (L(G)^{\complement} \cap L(F))$, *i.e.* the symmetric difference of the languages. Let $S_{\langle G,F\rangle}^{A_{TM}}$ be the oracle TM that acts a follows: On input w

1. Query the oracle of A_{TM} if G accepts w, and store the result.

2. Query the oracle of A_{TM} if F accepts w, and store the result.

3. If exactly one of step 1 and step 2 returned True, *accept*. If both or neither of step 1 and step 2 returned True, *reject*.

Clearly $S_{\langle G,F\rangle}^{A_{TM}}$ is a decider, and accept w if and only if $w \in L(G)\Delta L(F)$. Now we define the oracle TM $M^{A_{TM}}$ which recognizes EQ_{TM}. On input $\langle G, F\rangle$, where G and F are TMs, $M^{A_{TM}}$ acts as follows:

1. Check that F and G have the same input alpabet alphabet, Σ, if not *accept*.

2. Construct the oracle TM $S_{\langle G,F\rangle}^{A_{TM}}$ using the oracle of A_{TM}, as described above.

3. Run $S_{\langle G,F\rangle}^{A_{TM}}$ on w for each string in $w \in \Sigma^*$, say in lexicographical ordering. If for some string $\tilde{w} \in \Sigma^*$, $S_{\langle F,G\rangle}^{A_{TM}}$ accepts, *accept*.

It is easily seen that $M^{A_{TM}}$ accepts if and only if $L(G)$ and $L(F)$ does not have the same alphabet, *or* they have the same alphabet Σ and there exist a string w in Σ^* such that $w \in L(G)\Delta(F)$ which is again true if and only if $L(G) \neq L(F)$. Thus $M^{A_{TM}}$ accepts input $\langle G, F\rangle$ if and only if $L(G) \neq L(F)$ which is equaivalent to $L(M^{A_{TM}}) = EQ_{TM}^{\complement}$, as desired. □

Proof. We construct a Turing machine T with access to an oracle deciding A_{TM} as follows.

1) On input $\langle M_1, M_2\rangle$, accept if the input alphabets are distinct.

2) Query the oracle for A_{TM} with $\langle M_1\rangle w$ and $\langle M_2\rangle w$ for all strings $w \in \Sigma^*$ in lexicographical order.

3) If the query results for some string w are distinct, then accept.

□

6.14

Given a string x, we show how to compute the descriptive complexity $K(x)$, given an oracle for A_{TM}. Given a string, x, over the alphabet $\Sigma = \{0, 1\}$ we simply check all strings in Σ^* one by one, in lexocographical ordering. Given a string $v \in \Sigma^*$ we split it in all possible ways $v = v_1 v_2$ and query the oracle if v_1 as a descriptions of a TM accepts v_2, if so, we run the TM that v_1 decribes on v_2 and read of the tape after acceptance. We then check wether the read off tape matches x if so, $K(x) = |v|$, where v is the first string in lexicographilcal ordering that succesfully could be split in a description of a TM and an input such that the TM run on the input printed x. Clearly this can be done, since we never encounter any loops, so we are guarenteed to finish this proceedure after finitely many steps, especially beacuse of Theorem 6.24 that bounds $K(x) \leq |x| + b$.
Notice that there is a subtlety in the above argument, namely that in a general encoding $\langle M\rangle\, w$ of x, M need not *accept* w, but only *halt*. However, if there is a TM M that *rejects* w with x on its tape, we can describe the correpsonding TM, $\bar{M}$ with q_{accepts} and q_{reject} by a string $\langle \bar{M}\rangle$ such that $\left|\langle \bar{M}\rangle\right| = |\langle M\rangle|$, and thus we can restrict to encodings TMs that accept the input, without increasing $K(x)$. It is clear by definition of $K(x)$ that we are also *not* decreasing $K(x)$ by restricting the TM.
Alternatively one migth argue that an oracel for A_{TM} gives us an oracle for $HALT_{TM}$, by reducibility, and then use this oracle instead.

PCP and E_{TM}

We show that $PCP \leq_T E_{TM}$ and that $E_{TM} \leq_T PCP$. We use in the followsing the proof of Theorem 5.15 (that PCP is undecidable) and Example 6.19 i the book. By the proof of Theorem 5.15, we have that $A_{TM} \leq_m MPCP \leq_m PCP$, (as also noted in example 5.25) which implies $A_{TM} \leq_T PCP$ by transitivity of $\leq_m$ and by the fact that $A \leq_m B$ implies $A \leq_T B$. By

Example 6.19 we have that $E_{TM} \leq_T A_{TM}$. By transitivity of $\leq_T$ we thus only need to show $A_{TM} \leq_T PCP$ and $E_{TM} \leq_T A_{TM}$...

We show that $PCP \leq_T E_{TM}$ and that $E_{TM} \leq_T PCP$.

Proof. We start by showing that $PCP \leq_T E_{TM}$. Given an oracle for E_{TM} and an instance of the PCP, we simply construct a TM, S, such that S has non-empty language only if there is a match in the given instance of the PCP. Let $P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], ..., \left[\frac{t_k}{b_k}\right] \right\}$ be an instance of the PCP. We design S_P to have an input alphabet $a_1, ..., a_k$. S_P also contain the PCP instance, P, in the sense, that it associates to every a_i the domino $\left[\frac{t_i}{b_i}\right]$. We may decribe S_P in the following way: On input w, S_P does the following

1. Check that $w \in \{a_1, a_2, ..., a_k\}^+$ *i.e.* $w = a_{\sigma(1)}...a_{\sigma(m)}$ for some function $\sigma : \{1, ..., m\} \to \{1, ..., k\}$ and some $m \geq 1$. If not, S_P *rejects*.
2. Rewrite a_i on the tape as $[t_i/b_i]$, for every $i = 1, ..., k$, where $[$, $/$, and $]$ are tape letters. (This will require to rearrange the tape in order to make room for the extra letters).
3. Check if $t_{\sigma(1)}...t_{\sigma(m)} = b_{\sigma(1)}...b_{\sigma(m)}$, if true, S_P *accepts*, if false, S_P *rejects*.

Clearly S accepts w if and only if w corresponds via the bijective map $a_i \mapsto \left[\frac{t_i}{b_i}\right]$, to a match of P.
Now given an oracle for E_{TM} is clear that we can decide PCP defined by
$PCP = \{\langle P \rangle \mid P \text{ is an instance of } PCP \text{ with a match}\}$ with the following oracle TM, $M^{E_{TM}}$. On input $\langle P \rangle$, $M^{E_{TM}}$ does the following

1. Check that $\langle P \rangle$ encodes an instance of the PCP.
2. Construct S_P as described above.
3. Query the oracle for E_{TM}, wether $L(S_P) = \emptyset$, if true $M^{E_{TM}}$ *rejects*, if false, $M^{E_{TM}}$ *accepts*.

Thus $M^{E_{TM}}$ accepts input $\langle P \rangle$ if and only if P is an instance of PCP and S_P has non-empty language, which is again true if and only if P has a match. This shows that PCP is Turing reducible to E_{TM} *i.e.* $PCP \leq_T E_{TM}$.

We show now that $E_{TM} \leq_T PCP$, this can be seen in the following way. We take inspiration from the proof of Theorem 5.15, and modify it slightly. Given a TM describtion $\langle M \rangle$, we construct an instance of the PCP, P_M, such that P_M has a match if and only if $\langle M \rangle \in E_{TM}$. Let P_M have all the same dominos as P in the proof of theorem 5.15, except for the first domino. instead we include the dominos $\left[\frac{**\aleph}{**\aleph**\aleph**}\right]$, $\left[\frac{**\aleph}{\aleph**\aleph**}\right]$, $\left[\frac{**\aleph}{a*}\right]$, $\left[\frac{**\aleph*}{\#*}\right]$ for every a in the alpabet of M. It is clear that any match must start with $\left[\frac{**\aleph*}{**\aleph**\aleph**}\right]$, since this is the only domino that has upper string matches the first part of the lower string. Also since we have more $\aleph$s in the lower string of the first domino by construction, the upper string can catch up in two ways: It can catch up by simply using $\left[\frac{**\aleph*}{\#*}\right]$ at which case we have $\left[\frac{**\aleph**\aleph*}{**\aleph**\aleph*\#*}\right]$, thus we are set to simulate the empty string, by use of the dominos from the proof of Theorem 5.15. Alternatively it catches up by first using dominos of the form $\left[\frac{**\aleph}{\aleph**\aleph**}\right]$ *i.e.* it actually falls behind in the upper string, but then catches back up by using first dominos of the form $\left[\frac{**\aleph}{a*}\right]$ until it is forced to use the domino $\left[\frac{**\aleph*}{\#*}\right]$, as an example we might have $\left[\frac{**\aleph*}{\#*}\right]$ at which case we have $\left[\frac{**\aleph**\aleph**\aleph**\aleph**\aleph**\aleph*}{**\aleph**\aleph**\aleph**\aleph**\aleph**\aleph**a_1*a_2*\#*}\right]$, where we used domino $\left[\frac{**\aleph}{\aleph**\aleph**}\right]$ twice, followed by $\left[\frac{**\aleph}{a_1*}\right]$, $\left[\frac{**\aleph}{a_2*}\right]$ and $\left[\frac{**\aleph*}{\#*}\right]$. It is clear that by doing this we have set up the P_M instance to simulate M on input a_1a_2 by constructing any match, if it exists. By this last method we can clearly generate strings of arbitrary length. Thus we see that the constructed PCP instance, P_M, will have a match if and only if there *exist* a string w such that M accepts w or equivalently P_M has a match if and only if $L(M)$ is non-empty. Thus we see that we may construct an oracle TM M^{PCP} with an oracle for PCP such that M^{PCP} decides E_{TM}. M^{PCP} acts as follows on input $\langle K\rangle$ which is a TM describtion:

1. Construct P_K as decribed above.

2. Query the oracle for PCP, if $P_K \in PCP$, if true *reject*, if false, *accept*.

Thus by the arguments above we clearly have that M^{PCP} accepts $\langle K\rangle$ if and only if $L(K) = \emptyset$. This shows that $E_{TM} \leq_T PCP$. $\square$