6.840 Recitation 5

Alexander Dimitrakakis

October 2020

1 Computation History Method

It is often the case that in order to prove that a specific language is undecidable it is hard to directly reduce A_{TM} to it. What we can often do however is consider the computation history of a specific string $w \in \Sigma^*$ on a specific Turing Machine M. The computation history is the sequence of C_i's: $C_1, C_2, ..., C_l$, where C_1 is the starting configuration and C_l is the ending configuration. We often want to check whether a specific computation history is a valid accepting computation history for TM M on input w. What we do is create some structure (be it a TM, a 2DIMDFA, a CFG or anything) that accepts a valid accepting computation history for M on w. We approach this problem by doing 3 things:

- Check that C_1 is a valid starting configuration for M on w
- Check that C_{i+1} is a valid configuration that can be derived from C_i in one step
- Check that C_l is a valid accepting configuration, so TM M must be in state q_{accept}

2 Dimensional DFA (2DIMDFA): The input is an $m \times n$ rectangle, for any $m, n \geq 2$. The squares along the boundary of the rectangle contain the symbol # and the internal squares contain symbols over the input alphabet Σ. The transition function $\delta : Q \times (\Sigma \cup \#) \to Q \times L, R, U, D$ indicates the next state and the new head position (Left, Right, Up, Down). The machine is read-only, meaning that it can never write to the 2-dimensional tape. The machine accepts when it enters one of the designated accept states. It rejects if it tries to move off the input rectangle or if it never halts.

$A_{2DIMDFA}$ is decidable

$$A_{2DIMDFA} = \{\langle D, w\rangle | D \text{ is a 2DIMDFA and } w \in \Sigma^*, w \in L(D)\}$$

.

This is a decidable language because we can upper bound the number of possible configurations it can be in. If we have a 2DIMDFA with q states and an $n \times m$ tape, then the upper bound on the number of configurations it can be in is qnm. Thus, we can create the following decider:

TM D = "On input $\langle D, w \rangle$:
1) Run w on D for qnm steps. If it accepts, accept. If it rejects, reject.
2) If it has not halted after qnm steps, reject."

This works because if it runs for more than qnm steps it is necessarily looping forever.

$E_{2DIMDFA}$ is undecidable

$$E_{2DIMDFA} = \{\langle D \rangle | D \text{ is a 2DIMDFA}, L(D) = \emptyset\}$$

.

Here we use the Computation history method. We reduce A_{TM} to 2DIMDFA in this way. We create a 2DIMDFA, say R, that recognizes all the valid, accepting computation histories of string w on TM M. Each row of the 2DIMDFA is a step in the computation history. So if the computation history is $C_1, C_2, ..., C_l$, the input to the 2DIMDFA R will be a 2-dimensional tape with l rows, where C_i is the entry to row i (I am ommitting the rows with # here for convenience. If we take them into account than C_i will be in row $i + 1$). The number of columns will be equal to the length of the longest computation history step (+2 accounting for the # at the beginning and the end). The machine R checks each of the conditions laid out in the bulletpoints in the explanation of the use of the computation history method. We first check that C_1 is a valid starting configuration for M on w. We then check each C_i and C_{i+1} to check that the transition was valid. The interesting thing here is that because we can go up and down there is no need to be able to mark symbols by writing on the tape to remember our position, as we had to do in the LBA example presented in class and in the textbook. The 2-dimensional structure allows us to easily compare C_i and C_{i+1} in a way not possible in the one-dimensional case.

Now, assume that we have a decider for $E_{2DIMDFA}$, say T. Now, we are ready to present the TM B that decides A_{TM} and thus arrive at a contradiction.

TM B = "On input $\langle M, w \rangle$:
1) Create the 2DIMDFA explained above that recognizes all valid, accepting computation histories of M on w.
2) Run the decider T for $E_{2DIMDFA}$ on the 2DIMDFA constructed.
3) If it accepts, reject. If it rejects, accept."

Hence, we have a decider for A_{TM}, a contradiction. So, $E_{2DIMDFA}$ is undecidable.

2 A problem on undecidability

Here we will review the undecidability of a seemingly simple language. More specifically consider the language:
$USE2TAPE_{TM} = \{\langle M\rangle | M$ is a 2-tape TM that uses its second tape on some input at some point$\}$.
We will reduce E_{TM} to it and, thus, prove that it is undecidable. Assume we have a decider for $USE2TAPE_{TM}$.

TM Decider for E_{TM} = "On input $\langle M\rangle$:
1) Create TM M' that is a copy of TM M with some tweaks:
- Add a second tape.
- Whenever M enters an accept state, force M' to first write a symbol on the second tape and then accept.
2) Run the decider for $USE2TAPE_{TM}$ on M'. If it accepts, reject. If it rejects, accept."

Clearly, since E_{TM} is undecidable, we have a contradiction and, hence, $USE2TAPE_{TM}$ is undecidable.

3 Mapping Reducibility

On the concept of mapping reducibility, in class we showed that if we can create a computable function f between language A and language B, such that:

$$w \in A \iff f(w) \in B$$

.

Then we said that $A \leq_m B$ and we have reduced language A to language B. It is important to note that a few things hold if this is true. First, if language B is decidable, then A is also decidable. If A is undecidable, then B is also undecidable. Similarly, if B is Turing-recognizable, then A is also Turing-recognizable. Finally, if A is not Turing recognizable, then neither is B. It is important to note that we usually use mapping reductions to prove undecidability or unrecognizability.

Also note that $A \leq_m B \iff \overline{A} \leq_m \overline{B}$. So in mapping reductions we can just prove that the complement of the language A reduces to the complement of B if this turns out to be an easier task. When would we use this? When we want to show that a language, say C, is not T-recognizable, we often reduce

$\overline{A_{TM}}$ to it. So we need to show that $\overline{A_{TM}} \leq_m C$, which is equivalent to showing that $A_{TM} \leq_m \overline{C}$, which is usually much easier to prove.

Let us consider the language $REG_{TM} = \{\langle M \rangle | L(M) \text{ is a regular language}\}$. We will prove that REG_{TM} is neither Turing-recognizable, nor coTuring-recognizable. Theorem 5.3 in the book shows that the language is undecidable. Here we go one step further.

Let us first prove that REG_{TM} is not T-recognizable. Corollary 5.29 in the book suggests that proving the following will suffice: $\overline{A_{TM}} \leq_m REG_{TM}$. This is equivalent to showing that $A_{TM} \leq_m \overline{REG_{TM}}$.

```
TM $R_1$ = " On input $\langle M, w \rangle$:
1) Construct TM $M_1$ = "On input $x$:
- Simulate $M$ on $w$.
- If $M$ accepts $w$, then accept $x$ if it is of the form $0^k 1^k$.
- If $M$ rejects $w$, reject."
2) Output $\langle M_1 \rangle$."
```

What we have done above is that we have given a mapping reduction between A_{TM} and $\overline{REG_{TM}}$. If M accepts w, then $L(M_1) = \{0^k 1^k | k \geq 0\}$, which is known to be non-regular. If M rejects w or loops forever, then $L(M_1) = \emptyset$, which is regular. Hence, REG_{TM} is not Turing-recognizable.

Now, we prove that REG_{TM} is not co-Turing-recognizable. Due to Corollary 5.29 in the textbook, we only need to prove that $\overline{A_{TM}} \leq_m \overline{REG_{TM}} \iff A_{TM} \leq_m REG_{TM}$. We provide the following mapping reduction:

```
TM $R_2$ = " On input $\langle M, w \rangle$:
1) Construct TM $M_2$ = "On input $x$:
- If $x$ is of the form $0^k 1^k$ with $k \geq 0$, then accept.
- Else run $M$ on $w$ and accept if it does."
2) Output $\langle M_2 \rangle$."
```

What we have done above is that we have given a mapping reduction between A_{TM} and REG_{TM}. $L(M_2) = \{0^k 1^k | k \geq 0\}$, which is a well-known non-regular language whenever M does not accept w. Furthermore, $L(M_2) = \Sigma^*$ is regular whenever M accepts w. Hence, we have proven that REG_{TM} is not co-Turing-recognizable.