Solutions of PS #3

4.10)

Let $INF_{PDA} = \{\langle M\rangle \mid M\ is\ a\ PDA\ and\ L(M)\ is\ an\ infinite\ language\}$. Following is a proof that INF_{PDA} is decidable:

- Convert the PDA M into a CFG G.
- Convert G into Chomsky normal form H.
- Generate every combination of derivations with $\leq 2^b$ steps, where b is the number of variables in H.
- If we find a derivation with 2^b steps, $L(M)$ is infinite. Otherwise it is finite.

4.12)

Let $A = \{\langle R,S\rangle \mid R\ and\ S\ are\ regular\ expressions, L(R) \subseteq L(S)\}$. Following is a proof that A is decidable:

If $L(R) \subseteq L(S)$ then $L(R) \cap L(S) = L(R)$. We apply EQ_{DFA} on both sides of this equation and return the result.

4.22)

Let $L = \{\langle M\rangle \mid M\ is\ a\ PDA\ that\ has\ a\ useless\ state\}$. Following is a proof that L is decidable:

Let M_E be a decider for E_{CFG}. We go through each state in M, marking it the only accepting state. For each such iteration we apply M_E on that machine, and it will be discovered as empty iff the state that is marked as accepting cannot be reached by any computation.

The Busy Beaver Problem

How many 1's can an n-state TM write on an initially empty tape (and then stop) [Tibor Rado, 1962]

This problem is originally defined for Turing machines with an infinite tape to both sides, $\Gamma = \{0,1\}$, tape initialized with only 0's, and n is the number of operational states (excluding halting states). Denote $\Sigma(k)$ the result of $BB(k)$.

$\underline{n = 1}$: $\Sigma(1) = 1$ since we have to halt after 1 state (thus the maximum #1's we can write is 1).

$\underline{n = 2}$:

	A	B
0	1RB	1LA
1	1LB	1RH

$0A0 \rightarrow 01B0 \rightarrow 0A110 \rightarrow 0B1110 \rightarrow 01H1110 \Rightarrow \Sigma(2) = 4$

$\underline{n = 3}$: …

Definitions:

- $\Sigma(n) =$ maximum #1's with n states
- $S(n) =$ maximum #shift operations with n steps, denoted "Step count"

	1	2	3	4	5	6
$\Sigma(n)$	1	4	6	13	≥ 4098	$\geq 3.5 \cdot 10^{18267}$
$S(n)$	1	6	21	107	≥ 47176870	?

Theorem

The function $\Sigma(n)$ is not computable.

Proof:

By contradiction, assume we have:

- $EVAL_\Sigma$ that computes Σ, so given n 1's on the input tape, it writes $\Sigma(n)$ on the input tape.
- $DOUBLE$ a TM that given n 1's, writes $2n$ 1's.
- INC a TM that given n 1's writes $n+1$ 1's.

Consider the concatenation of the TMs $INC \circ EVAL_\Sigma \circ DOUBLE$, and assume it has n_0 states. Let $CREATE_{n_0}$ be an n_0-state TM that writes n_0 1's (with the transitions $\delta(q_i, \_) = (q_{i+1}, 1, R)$, $\delta(q_{n_0}, \_) = (q_{accept}, 1, R)$).

Consider $INC \circ EVAL_\Sigma \circ DOUBLE \circ CREATE_{n_0}$, then this should have $N := n_0 + n_0$ states. This machine writes $\Sigma(N)+1$ 1's, a contradiction, since any machine with N states should be able to write only $\leq \Sigma(N)$ 1's. Therefore $EVAL_\Sigma$ does not exist, so Σ is not computable □

Note: we can standardize the TMs above to go back to the beginning of the input tape at the end of their computation, such that this way we can simply "glue" them together with no overhead states for the gluing (all overhead is covered with the "go back to square 1" part).

Theorem

The function $S(n)$ is not computable.

Proof:

Similarly to the proof above, only use: $CLEAN \circ EVAL_S \circ DOUBLE \circ CREATE_{n_0}$ where $CLEAN$ wipes all 1's from the input tape. As before it has $N := n_0 + n_0$ states and performs $> S(N)$ shift operations, since the CLEAN forces to perform $S(N)$ shifts only for the cleaning part – so we end up with more than that. □

Rado's Theorem:

Let $f: \mathbb{N} \to \mathbb{N}$ be a computable function, then $\exists n_0 \forall n \geq n_0 : f(n) < \Sigma(n) < S(n)$.

Proof: think about it…

Mapping Reducibility

Mapping-reducible: a language A is mapping-reducible to language B if there exists a function $f: \Sigma^* \to \Sigma^*$ such that $\forall w \in \Sigma^*: w \in A \Leftrightarrow f(w) \in B$ **and** f is computable (there exists a TM that computes it). Notation: $A \leq_m B$.

Conversely, if A is undecidable, B is undecidable.

In addition: $A \leq_m B \Leftrightarrow \bar{A} \leq_m \bar{B}$ (using the same reduction function).

Example:

$A_{TM} \leq_m HALT_{TM}$:

For a given input $\langle M, w \rangle$ we map it to a TM M' that halts iff $M(w)$ accepts. Define M' as follows: We simulate M on w. If it rejects, loop forever. If it accepts, accept.

If M accepts w, M' will accept, therefore halt. If M rejects w or loops forever, M' will loop forever. Therefore $\langle M, w \rangle \in A_{TM} \Leftrightarrow M' \in HALT_{TM}$. In the cases the input is not a proper pair of TM and a string, we map it to a TM M' that is not a proper encoding of a TM as well.

Review:

- We showed that A_{TM} is undecidable (using diagonalization): assume some TM M decides A_{TM}, so we build a machine M' that given a code of a machine N accepts if N accepts N and reject if N rejects N (using M to check if N halts on itself as input). Then we get a contradiction for the computation $M'(M')$, as it should do the opposite of what it does.
- $\overline{A_{TM}}$ is not recognizable.
- $HALT_{TM}$ is undecidable by a reduction from A_{TM}: $A_{TM} \leq_m HALT_{TM}$

Example 5.26:

$E_{TM} \leq_m EQ_{TM}$:

For a given TM M we build the pair M_1, M_2 where $M_1 = M$ and M_2 is a TM that rejects always. This way, if $L(M) = \emptyset$ then $L(M_1) = L(M_2) = \emptyset$, and the other way around.

Example 5.27:

The use of mapping reducibility to prove E_{TM} is undecidable:

Given $\langle M, w \rangle$ construct M' that does the following:

On input x, if $x \neq w$ then reject. Otherwise simulate M on w. If M accepts, accept. Otherwise, reject.

Therefore $L(M') = \emptyset \Leftrightarrow M$ does not accept w.

Note: this is a reduction $A_{TM} \leq_m \overline{E_{TM}}$.

Moreover it can be proven that there is no reduction $A_{TM} \leq_m E_{TM}$: if there was such a reduction, then there would be a reduction $\overline{A_{TM}} \leq_m \overline{E_{TM}}$, but $\overline{A_{TM}}$ is not recognizable, therefore it would mean $\overline{E_{TM}}$ is also not Turing-recognizable, but we know it is – a contradiction.

The recognizing TM M for $\overline{E_{TM}}$ simply enumerates over all possible inputs and simulates the given input TM (in "BFS": each step run i steps of the first i possible inputs).

Theorem 5.30

EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

Proof:

EQ_{TM} is not Turing-recognizable:

It suffices to show that $A_{TM} \leq_m \overline{EQ_{TM}}$, since then $\overline{A_{TM}} \leq_m \overline{\overline{EQ_{TM}}} = EQ_{TM}$, and $\overline{A_{TM}}$ is not Turing-recognizable. Given $\langle M, w\rangle$ we will create $\langle M_1, M_2\rangle$ such that M accepts $w \Leftrightarrow L(M_1) \neq L(M_2)$: Let M_1 s.t. $L(M_1) = \emptyset$ and M_2 simulates M on w.

$\overline{EQ_{TM}}$ is not Turing-recognizable:

It suffices to show that $A_{TM} \leq_m EQ_{TM}$ since then $\overline{A_{TM}} \leq_m \overline{EQ_{TM}}$, and $\overline{A_{TM}}$ is not Turing-recognizable. Given $\langle M, w\rangle$ we will create $\langle M_1, M_2\rangle$ such that M accepts $w \Leftrightarrow L(M_1) = L(M_2)$: Let M_1 s.t. $L(M_1) = \{w\}$ and M_2 rejects all $x \neq w$, and if $x = w$, simulates M on w and accepts iff it is accepted.

Recursion Theorem

The Turing machine $SELF$ prints its own code.

Lemma 6.1: There exists a computable function $q: \Sigma^* \to \Sigma^*$ such that $q(w)$ is the code of a TM P_w that prints w. □

From that we construct $SELF$ from two parts AB:

- A prints P_B
- B: on input M:
  - Print $q(M)$
  - Combine with M
  - Print the description