

1 Decidability

$$L = \{\langle M, w \rangle \mid M \text{ is a TM and } w \text{ is a string} \\ \text{and } M \text{ never overwrites its tape on input } w\}$$

Here we consider TMs as defined in Sipser with tapes that are only infinite to the right. We will prove that the language L is decidable. Consider a TM M and an input w . When M is run on w , one of the following four cases must happen:

1. M will overwrite its tape.
2. M will not overwrite its tape and halt.
3. M will not overwrite its tape and will loop forever over a finite amount of tape.
4. M will not overwrite its tape and will loop forever while moving infinitely far to the right.

The first two cases are easy to detect. We may simply run M and see if it overwrites its tape. To detect the 3rd case, we note that since M is deterministic, if it is ever in the same state and position on its tape twice without overwriting its tape then it will loop forever. To detect the 4th case, if M is ever in the same state twice while only reading blank symbols and is further right the second time in the state than the first, then it will loop forever moving right.

Assume for the rest of this paragraph that M does not overwrite its tape when run on w . Let p denote the number of states in M . If M is ever more than p blanks to the right of the end of w , then by the pigeon hole principle, it must have been in the same state twice, and further right the second time. If this happens, we know it will loop forever moving right. If this doesn't happen, there are p possible states and $|w| + p$ possible positions for M . If M executes more than $p(|w| + p)$ steps, we know it must have either been in the same state and position twice or moved more than p blanks past the end of w . In either case, we know it is guaranteed to loop.

The following TM decides L : "On input $\langle M, w \rangle$:

1. Let p be the number of states in M .
2. Simulate M on input w until it either overwrites its tape or executes more than $p(|w| + p)$ steps.
3. If M ever overwrote its tape, reject. Otherwise, accept."

2 Undecidability

$$L = \{\langle M, w \rangle \mid M \text{ is a 2-tape TM and } w \text{ is a string} \\ \text{and } M \text{ never overwrites its } \textit{first} \text{ tape on input } w\}$$

We will prove that L is undecidable by giving a Turing reduction from $HALT$:

“On input $\langle M, w \rangle$:

1. Construct the following 2-tape TM M' :

“On input x :

- (a) Simulate M on input w using *only* the second tape.
- (b) Write x on the first tape.”

2. Query the oracle for L on input $\langle M', \epsilon \rangle$.

3. If the oracle for L accepts then reject. Otherwise, accept.”

If M halts on w , then M' will reach step b and overwrite its first tape. Thus, the oracle will reject so our TM will accept.

If M does not halt on input w , then M' will never reach step b. Thus, it will never overwrite its first tape so the oracle will accept and our TM will reject.

Thus, our TM with oracle access to L decides $HALT$, so L must be undecidable.

3 Reductions and regular languages

Disproof:

Let $A = \{a^n b^n \mid n \geq 0\}$ and let $B = L(0^* 1^*)$. Clearly, A is not regular and B is. However, $A \leq_m B$:

Let f be the TM “On input w :

1. If w is of the form $a^n b^n$ then output 01.
2. Otherwise output 10.”

For any string w , if $w \in A$ then $f(w) = 01 \in B$. If $w \notin A$ then $f(w) = 10 \notin B$. Finally, f is a computable functions since we may zig-zag between the a 's and b 's, crossing them off, to check to see if there is the same number of both in the input.

4 Map reductions

- $SUBSETEQ_{CFG} \leq_m EQ_{CFG}$:

For this reduction, we will use the idea that for any 2 sets A and B , $A \subseteq B$ iff $A \cup B = B$.

“On input $\langle G_1, G_2 \rangle$:

1. Let H be a grammar for $L(G_1) \cup L(G_2)$.
2. Output $\langle H, G_2 \rangle$.”

This is a computable function since CFGs are closed under union and there is a simple procedure (described in Sipser) to construct the union.

- $EQ_{CFG} \leq_m SUBSETEQ_{CFG}$:

For this reduction, we will use the idea that for any 2 sets A and B , $A = B$ iff $A \subseteq B$ and $B \subseteq A$, or equivalently $A \times B \subseteq B \times A$.

“On input $\langle G_1, G_2 \rangle$:

1. Let X be a grammar for the language consisting of only the string $\mathbf{x}$, where $\mathbf{x}$ is a symbol that is not generated by either G_1 or G_2 .
2. Let H_1 be a grammar for $L(G_1) \circ L(X) \circ L(G_2)$.
3. Let H_2 be a grammar for $L(G_2) \circ L(X) \circ L(G_1)$.
4. Output $\langle H_1, H_2 \rangle$.”

This is a computable function since CFGs are closed under concatenation and there is a simple procedure (described in Sipser) to construct the concatenation. Since $\mathbf{x}$ is not generated by either G_1 or G_2 , we may uniquely identify strings generated by H_1 with elements of $L(G_1) \times L(G_2)$ by splitting the string on $\mathbf{x}$. Similarly, we may uniquely identify strings generated by H_2 with elements of $L(G_2) \times L(G_1)$. Thus, $L(H_1) \subseteq L(H_2)$ iff $L(G_1) = L(G_2)$.