B501 Notes on Reduction

March 2012

1 Undecidability of HALT

1.1 Informal proof (Sipser book Sec 5.1)

Let's assume for the purposes of obtaining a contradiction that TM R decides $HALT$. We construct TM S to decide A_{TM}, with S operating as follows:

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

1.2 Formal proof

Suppose we have D_{HALT}, a decider for HALT, then we can construct D_{ATM}, a decider for A_{TM} as follows:

$$\begin{aligned} D_{\mathrm{ATM}}(\langle M, w\rangle) \;=\; & \text{if}\,(D_{\mathrm{HALT}}(\langle M, w\rangle)) \\ & \text{then eval}(\langle M, w\rangle)) \\ & \text{else } \textit{reject} \end{aligned}$$

Notice how this definition corresponds to the above informal definition.

Here the notation "eval($\langle M,\ w\rangle$)" means "the outcome of simulating TM D_{HALT} on input $\langle M,\, w\rangle$". This is analogous to the Scheme code `(eval '(M w))`. The notation $\langle M,\, w\rangle$ means a piece of *quoted code*, similar to the Scheme notation `'(M w)`. It corresponds to a string on the tape of a TM containing the description of a TM M and its input w.

The outcome of eval($\langle M, w\rangle$) may be *accept*, *reject*, but it also may *loop*. Since we first used D_{HALT} on $\langle M, w\rangle$ to determine whether M halts on w, we know that eval($\langle M, w\rangle$) will not loop, so the "then" branch will always produce *accept* or *reject*.

Thus we have defined a decider for A_{TM}, contradicting the fact that A_{TM} is undecidable.

1.3 Informal proof with mapping reduction (Sipser book Sec 5.3)

We demonstrate a mapping reducibility from A_{TM} to $\mathrm{HALT}_{\mathrm{TM}}$ as follows. To do so we must present a computable function f that takes input of the form $\langle M, w\rangle$ and returns output of the form $\langle M', w'\rangle$, where

$$\langle M, w\rangle \in A_{\mathrm{TM}} \text{ if and only if } \langle M', w'\rangle \in \mathrm{HALT}_{\mathrm{TM}}.$$

The following machine F computes a reduction f.

F = "On input $\langle M, w\rangle$:

1. Construct the following machine M'.
   M' = "On input x:
   1. Run M on x.
   2. If M accepts, *accept*.
   3. If M rejects, enter a loop."

2. Output $\langle M', w\rangle$."

1.4 Formal proof with mapping reduction

Suppose we have D_{HALT}, a decider for HALT. We define the mapping reduction as:

$$f(\langle M, w\rangle) \; = \; \langle M', w\rangle$$

where the TM M' is constructed as computing the function:

$$\begin{aligned} M'(x) \; &= \; \text{if}\,(\text{eval}(\langle M, x\rangle)) \\ &\quad\;\; \text{then } \textit{accept} \\ &\quad\;\; \text{else } \textit{loop} \end{aligned}$$

This means exactly (but concisely) what the above informal description says:

Construct the following machine M'.

"On input x:

1. Run M on x.
2. If M accepts, *accept*.
3. If M rejects, enter a loop."

Notice although M' takes input named "x", the actual input is w when it is simulated. This is because when $\langle M', w\rangle$ is passed as input to a decider of HALT, it uses the w part as the *actual argument* for M'. This is like the difference between a formal parameter and the actual argument in a function call when using a programming language such as Java or Python.

We can see the behavior of M, M' and D_{HALT} in the following table:

M accepts w	M' accepts w	D_{HALT} accepts $\langle M', w\rangle$
M rejects w	M' loops	D_{HALT} rejects $\langle M', w\rangle$
M loops on w	M' loops	D_{HALT} rejects $\langle M', w\rangle$

Table 1. Behavior table for M, M' and D_{HALT}

Notice that D_{HALT} accepts $\langle M', w\rangle$ when M accepts w, and reject otherwise. So if D_{HALT} exists, we can use the output of running it on $\langle M', w\rangle$ to decide A_{TM}. That is to say, we can define D_{ATM} as follows:

$$D_{\text{ATM}}(\langle M, w\rangle) = D_{\text{HALT}}(\langle M', w\rangle)$$

which contradicts the fact that D_{ATM} cannot exist.

2 Undecidability of E_{TM}

We reduce A_{TM} to $\overline{E_{\text{TM}}}$. Suppose we have a decider D_{ETM} for E_{TM}. We define the mapping reduction as:

$$f(\langle M, w\rangle) \; = \; \langle M_1\rangle$$

where M_1 is defined as

$$\begin{aligned} M_1(x) \; &= \; \text{if}\,(x = w) \\ &\quad\;\; \text{then eval}(\langle M, x\rangle) \\ &\quad\;\; \text{else } \textit{reject} \end{aligned}$$

Notice that f maps $\langle M, w\rangle$, the input for A_{TM} to $\langle M_1\rangle$, the input for D_{ETM}. There is no w part for the input of D_{ETM} because it determines property of a TM regarding *all inputs*.

We can see that if D_{ETM} exists, then we can decide A_{TM} by defining the decider for it as

$$D_{\mathrm{ATM}}(\langle M, w\rangle) = \mathrm{not}\, D_{\mathrm{ETM}}(\langle M_1\rangle).$$

This is because

- If $D_{\mathrm{ETM}}(\langle M_1\rangle)$ accepts, then M_1 is empty, i.e., it will reject *all* inputs. Looking at the definition of M_1, we can see that its input is x, and it runs M on x only when $x = w$, otherwise it rejects. That is, w is the only input M_1 can possibly accept. In order for M_1 to reject all input x, $\mathrm{eval}(\langle M, x\rangle)$ must reject, otherwise M_1 may accept or loop. But notice that we only execute $\mathrm{eval}(\langle M, x\rangle)$ when $x = w$, so M must reject w.
- On the other hand, if $D_{\mathrm{ETM}}(\langle M_1\rangle)$ rejects, then M_1 is not empty, i.e., it will accept *some* inputs. But notice from the definition of M_1 that w is the only input M_1 can possibly accept. In order for M_1 to accept some input x, $\mathrm{eval}(\langle M, w\rangle)$ must accept, otherwise M_1 will reject all inputs. That is to say, in order for M_1 to accept some input x, M must accept w.

We can see the behavior of D_{ETM} and M satisfies the following table:

M rejects w	D_{ETM} accepts $\langle M_1\rangle$
M accepts w	D_{ETM} rejects $\langle M_1\rangle$

Table 2. Behavior table for M and D_{ETM}.

So we have successfully mapping reduced A_{TM} to $\overline{E_{\mathrm{TM}}}$.

3 Undecidability of $\mathrm{EQ}_{\mathrm{TM}}$

We reduce E_{TM} to $\mathrm{EQ}_{\mathrm{TM}}$. We define the mapping reduction as:

$$f(\langle M\rangle) = \langle M, M_1\rangle$$

where M_1 is defined as

$$M_1(x) = \mathit{reject}.$$

That is to say M_1 rejects all input.

If D_{EQTM} is a decider for $\mathrm{EQ}_{\mathrm{TM}}$, then we can define a decider of E_{TM} as follows:

$$D_{\mathrm{ETM}}(\langle M\rangle) = D_{\mathrm{EQTM}}(\langle M, M_1\rangle).$$

contracting the fact that D_{ETM} cannot exist.

4 $\mathrm{EQ}_{\mathrm{TM}}$ is neither Turing-recoginizable nor co-Turing-recognizable

4.1 $\mathrm{EQ}_{\mathrm{TM}}$ is not Turing-recognizable

In order to prove that $\mathrm{EQ}_{\mathrm{TM}}$ is not Turing-recognizable, we reduce A_{TM} to $\overline{\mathrm{EQ}_{\mathrm{TM}}}$. We define the mapping reduction as

$$f(\langle M, w\rangle) = \langle M_1, M_2\rangle.$$

where M_1 and M_2 are two TMs defined as:

$$\begin{aligned} M_1(x) &= \mathit{reject} \\ M_2(x) &= \mathrm{eval}(\langle M, w\rangle). \end{aligned}$$

Notice that in the above definition of M_2, the body $\mathrm{eval}(\langle M, w\rangle)$ does not refer to the input x at all! That is, it ignores the input x. The input is x, but the body is $\mathrm{eval}(\langle M, w\rangle)$, which simulates M on w (and not x!). So $\mathrm{eval}(\langle M, w\rangle)$ solely decides the outcome *disregarding what input M_2 gets*. M_2 accepts *all* inputs if $\mathrm{eval}(\langle M, w\rangle)$ accepts; M_2 rejects *all* inputs if $\mathrm{eval}(\langle M, w\rangle)$ rejects.

So if decider for $\mathrm{EQ}_{\mathrm{TM}}$, D_{EQTM}, exists, then

- If D_{EQTM} accepts $\langle M_1, M_2\rangle$, then M_1 is equivalent to M_2, i.e., they accept and rejects the same inputs. Because M_1 rejects *all* inputs, M_2 must also reject *all* inputs x. In order to rejects all inputs x, $\mathrm{eval}(\langle M, w\rangle)$ must reject. That is to say, M must reject w.
- If D_{EQTM} rejects $\langle M_1, M_2\rangle$, then M_1 is not equivalent to M_2. Because M_1 rejects *all* inputs, M_2 must accept *some* inputs. In order to allow M_2 accept some inputs, $\mathrm{eval}(\langle M, w\rangle)$ must accept. That is to say, M must accept w.

Thus we have reduced A_{TM} to $\overline{\mathrm{EQ}_{\mathrm{TM}}}$.

4.2 $\mathrm{EQ}_{\mathrm{TM}}$ is not co-Turing-recognizable

Similarly, in order to prove that $\mathrm{EQ}_{\mathrm{TM}}$ is not co-Turing-recognizable, we reduce A_{TM} to $\mathrm{EQ}_{\mathrm{TM}}$. We define the mapping reduction as

$$f(\langle M, w\rangle) = \langle M_1, M_2\rangle.$$

where M_1 and M_2 are two TMs defined as:

$$\begin{aligned} M_1(x) &= \textit{accept} \\ M_2(x) &= \mathrm{eval}(\langle M, w\rangle). \end{aligned}$$

If decider for $\mathrm{EQ}_{\mathrm{TM}}$, D_{EQTM}, exists, then

- If D_{EQTM} accepts $\langle M_1, M_2\rangle$, then M_1 is equivalent to M_2, i.e., they accept and rejects the same inputs. Because M_1 accepts *all* inputs, M_2 must also accept *all* inputs. In order to accept all inputs, $\mathrm{eval}(\langle M, w\rangle)$ must accept. That is to say, M must accept w.
- If D_{EQTM} rejects $\langle M_1, M_2\rangle$, then M_1 is not equivalent to M_2. Because M_1 accept *all* inputs, M_2 must *not* accept all inputs. That is, M_2 must rejects *some* input. In order to allow M_2 reject some inputs, $\mathrm{eval}(\langle M, w\rangle)$ must rejct, otherwise if $\mathrm{eval}(\langle M, w\rangle)$ accepts, M_2 will accept *all* inputs. So M must reject w.

Thus we have reduced A_{TM} to $\mathrm{EQ}_{\mathrm{TM}}$.