1 Undecidability via Reductions

We turn to showing undecidability via reductions. The form of the argument will be as follows.

Suppose that we are given an algorithm (or program) $A_L$ which is claimed to decide set $L$. Suppose that using $A_L$ as a subroutine, we create another algorithm $A_J$ to decide set $J$. But suppose that we already know set $J$ to be undecidable, for example if $J = H$, the halting set. We can then conclude that $A_L$ does not decide set $L$, and in fact that there is no algorithm to decide set $L$. The latter claim follows by a proof by contradiction: assume that there is such an algorithm, and call it $A_L$; then the previous argument shows that $A_L$ does not decide $L$, a contradiction.

**Example 1** $W = \{\langle Q \rangle \mid Q$ on input string 0 eventually halts$\}$.

**Lemma 2** If there is an algorithm $A_W$ to decide $W$, then there is also an algorithm to decide $H$, the halting set.

**Proof:** Here is the algorithm $A_H$ deciding $H$. It will use $A_W$ as a subroutine.

On input $\langle P, w \rangle$:

$A_H$ will build (compute the encoding of) a one-input program $R_{P,w}$\(^1\) which it then inputs to the algorithm $A_W$. What we want is that:

$$A_W(\langle R_{P,w} \rangle) = \begin{cases} \text{“Recognize”} & \text{if } P \text{ eventually halts on input } w \\ \text{“Reject”} & \text{if } P \text{ does not halt on input } w \end{cases}$$

Now by the definition of $W$:

$$A_W(\langle R_{P,w} \rangle) = \begin{cases} \text{“Recognize”} & \text{if } R_{P,w}(0) \text{ halts} \\ \text{“Reject”} & \text{if } R_{P,w}(0) \text{ does not halt} \end{cases}$$

\(^1\)The notation $R_{P,w}$ is meant to indicate that program $R$ is determined in part by $P$ and $w$. (Thus $P$ might be a subroutine in $R$, and $w$ might be the initial value of one of $R$'s variables. $w$ is not an input to $R$. $x$ denotes $R$'s input.)
This means that we want:

\[ R_{P,w}(0) \text{ halts} \quad \text{if } P \text{ eventually halts on input } w \]
\[ R_{P,w}(0) \text{ does not halt} \quad \text{if } P \text{ does not halt on input } w. \]

Here is a program \( R_{P,w} \) that meets the above requirement:

\[ R_{P,w}(x) : \]
run \( P \) on input \( w \) (note that \( R_{P,w} \) ignores its input).

Clearly, for any \( x \), \( R_{P,w}(x) \) halts if \( P \) eventually halts on input \( w \), and \( R_{P,w}(x) \) does not halt if \( P \) does not halt on input \( w \), and hence it does so for \( x = 0 \) in particular.

Thus our algorithm for deciding \( H \) proceeds as follows:

**Step 1.** \( A_H \) computes \( \langle R_{P,w} \rangle \), the encoding of program \( R_{P,w} \).

**Step 2.** \( A_H \) simulates algorithm \( A_W \) on input \( \langle R_{P,w} \rangle \) and outputs the result of \( A_W(\langle R_{P,w} \rangle) \).

**Corollary 3** \( W \) is undecidable, that is there is no algorithm to decide \( W \).

**Example 4** Let \( x = \{ \langle Q \rangle \mid \text{either } Q \text{ on input } 0 \text{ eventually halts, or } Q \text{ on input } 1 \text{ eventually halts, or both} \} \).

**Lemma 5** If there is an algorithm \( A_X \) to decide \( X \), then there is also an algorithm \( A_H \) to decide \( H \).

**Proof:** This proof is very similar to the previous one. Again, \( A_H \) will build a program \( R'_{P,w} \) with the following characteristics:

\[ A_X(\langle R'_{P,w} \rangle) = \begin{cases} 
\text{“Recognize”} & \text{if } P \text{ eventually halts on input } w \\
\text{“Reject”} & \text{if } P \text{ does not halt on input } w
\end{cases} \]

This means that we want:

at least one of \( R'_{P,w}(\langle 0 \rangle) \) and \( R'_{P,w}(\langle 1 \rangle) \) halts if \( P \) eventually halts on input \( w \)
both \( R'_{P,w}(\langle 0 \rangle) \) and \( R'_{P,w}(\langle 1 \rangle) \) do not halt if \( P \) does not halt on input \( w \)

It is easy to check that \( R_{P,w} = R'_{P,w} \) meets this requirement. Thus we can use the following algorithm \( A_H \) to decide \( H \).

On input \( \langle P, w \rangle \):

**Step 1.** \( A_H \) constructs the encoding \( \langle R_{P,w} \rangle \).

**Step 2.** \( A_H \) simulates \( A_X(\langle R_{P,w} \rangle) \) and give its result as the output of \( A_H \).

**Corollary 6** \( X \) is undecidable.

**Example 7** \( Y = \{ \langle Q \rangle \mid \text{there is a variable } v \text{ in } Q \text{ that is never assigned a value when run on input } 1 \} \).
Lemma 8 If there is an algorithm $A_Y$ to decide $Y$, then there is also an algorithm $A_H$ to decide $H$.

Proof: Here is the algorithm $A_H$ to decide $H$.

On input $\langle P, w \rangle$:

$A_H$ will compute the encoding of a program $R''_{P,w}$ which it then inputs to $A_Y$. What we want is for:

$$A_Y(\langle R''_{P,w} \rangle) = \begin{cases} 
"Recognize" & \text{if } P \text{ eventually halts on input } w \\
"Reject" & \text{if } P \text{ does not halt on input } w
\end{cases}$$

This means that we want:

- If $P$ eventually halts on input $w$ then $R''_{P,w}$ has a variable that is never assigned a value when run on input 1, and
- If $P$ does not halt on input $w$ then every variable of $R''_{P,w}$, when run on input 1, is assigned a value.

Let’s try the following program for $R''_{P,w}$:

On input $x$:

Step 1. $R''_{P,w}$ simulates $P(w)$.

Step 2. For every variable $z$ appearing in $P$ do: $z \leftarrow 0$.

Step 3. $v \leftarrow 1$, where $v$ is a variable that does not appear in $P$.

When $R''_{P,w}$ is run on input $x$ and $x = 1$ in particular, if $P$ halts on input $w$, then every variable appearing in $R''_{P,w}$ is assigned a value, while if $P$ does not halt on input $w$, at the very least, variable $v$ in $R''_{P,w}$ is not assigned a value.

Oops, this is back to front. Unfortunately, this is unavoidable. So let’s change our goal for $A_Y$. Let’s require:

$$A_Y(\langle R''_{P,w} \rangle) = \begin{cases} 
"Reject" & \text{if } P \text{ eventually halts on input } w \\
"Recognize" & \text{if } P \text{ does not halt on input } w
\end{cases}$$

This means that we want:

- If $P$ eventually halts on input $w$ then every variable of $R''_{P,w}$, when run on input 1, is assigned a value.
- If $P$ does not halt on input $w$ then $R''_{P,w}$ has a variable that is never assigned a value when run on input 1.

But this is achieved by the above program $R''_{P,w}$.

Now, our algorithm $A_H$ for deciding $H$ simply reports the opposite answer to $A_Y(\langle R''_{P,w} \rangle)$. So the algorithm is the following:

Step 1. $A_H$ constructs the encoding $\langle R''_{P,w} \rangle$.

Step 2. $A_H$ simulates $A_Y(\langle R''_{P,w} \rangle)$.

Step 3. $A_H$ reports the opposite answer to that given by $A_Y$ in Step 2. □
Corollary 9 \( Y \) is undecidable.

Example 10 \( \text{Never-Halt} = \{\langle Q \rangle \mid Q \text{ does not halt on any input} \} \).

Lemma 11 If there is an algorithm \( \mathcal{A}_{\text{Never-Halt}} \) to decide \( \text{Never-Halt} \), then there is also an algorithm \( \mathcal{A}_H \) to decide \( H \).

Proof: The algorithm \( \mathcal{A}_H \) deciding \( H \) will use \( \mathcal{A}_{\text{Never-Halt}} \) as a subroutine. On input \( \langle P, w \rangle \), \( \mathcal{A}_H \) will compute the encoding of a one-input program \( R^3_{P,w} \) which it then inputs to the algorithm \( \mathcal{A}_{\text{Never-Halt}} \). What we want is that:

\[
\mathcal{A}_{\text{Never-Halt}}(\langle R^3_{P,w} \rangle) = \begin{cases} 
\text{"Recognize"} & \text{if } P \text{ eventually halts on input } w \\
\text{"Reject"} & \text{if } P \text{ does not halt on input } w 
\end{cases}
\]

Now by the definition of \( \text{Never-Halt} \):

\[
\mathcal{A}_{\text{Never-Halt}}(\langle R^3_{P,w} \rangle) = \begin{cases} 
\text{"Recognize"} & \text{if } R^3_{P,w} \text{ never halts} \\
\text{"Reject"} & \text{if } R^3_{P,w} \text{ halts on some input} 
\end{cases}
\]

This means that we want:

\[
R^3_{P,w} \text{ never halts} \quad \text{if } P \text{ eventually halts on input } w \\
R^3_{P,w} \text{ halts on some input} \quad \text{if } P \text{ does not halt on input } w.
\]

It does not seem possible to build such an \( R \). Let’s try switching the outputs given by \( \mathcal{A}_{\text{Never-Halt}} \) to be:

\[
\mathcal{A}_{\text{Never-Halt}}(\langle R^3_{P,w} \rangle) = \begin{cases} 
\text{"Reject"} & \text{if } P \text{ eventually halts on input } w \\
\text{"Recognize"} & \text{if } P \text{ does not halt on input } w 
\end{cases}
\]

And then we want:

\[
R^3_{P,w} \text{ never halts} \quad \text{if } P \text{ does not halt on input } w \\
R^3_{P,w} \text{ halts on some input} \quad \text{if } P \text{ eventually halts on input } w.
\]

The program \( R^3_{P,w} = R_{P,w} \) from Example 2 meets the above requirement.

Thus our algorithm for deciding \( H \) proceeds as follows:

Step 1. \( \mathcal{A}_H \) computes \( \langle R_{P,w} \rangle \), the encoding of program \( R_{P,w} \).

Step 2. \( \mathcal{A}_H \) simulates algorithm \( \mathcal{A}_{\text{Never-Halt}}(\langle R_{P,w} \rangle) \) and outputs the opposite of its result. \( \blacksquare \)

Corollary 12 \( \text{Never-Halt} \) is undecidable.

Example 13 \( \text{Equal-Prog} = \{\langle Q_1, Q_2 \rangle \mid Q_1 \text{ and } Q_2 \text{ halt on exactly the same inputs} \} \).

We write \( Q_1 = Q_2 \) if \( \langle Q_1, Q_2 \rangle \in \text{Equal-Prog} \) and \( Q_1 \neq Q_2 \) if \( \langle Q_1, Q_2 \rangle \notin \text{Equal-Prog} \), for short.
Lemma 14 If there is an algorithm $A_{\text{Equal-Prog}}$ to decide Equal-Prog, then there is also an algorithm $A_{\text{Never-Halt}}$ to decide Never-Halt.

Proof: The algorithm $A_{\text{Never-Halt}}$ deciding Never-Halt will use $A_{\text{Equal-Prog}}$ as a subroutine. On input $\langle P \rangle$, $A_{\text{Never-Halt}}$ will compute the encoding of programs $R^4_P$ and $R^5_P$ which it then inputs to the algorithm $A_{\text{Equal-Prog}}$. What we want is that:

$$A_{\text{Equal-Prog}}(\langle R^4_P, R^5_P \rangle) = \begin{cases} \text{"Recognize"} & \text{if } P \text{ does not halt on any input} \\ \text{"Reject"} & \text{if } P \text{ halts on some input} \end{cases}$$

Now by the definition of Equal-Prog:

$$A_{\text{Equal-Prog}}(\langle R^4_P, R^5_P \rangle) = \begin{cases} \text{"Recognize"} & \text{if } R^4_P = R^5_P \\ \text{"Reject"} & \text{if } R^4_P \neq R^5_P \end{cases}$$

This means that we want:

$$R^4_P = R^5_P \text{ if } P \text{ does not halt on any input}$$
$$R^4_P \neq R^5_P \text{ if } P \text{ halts on some input.}$$

This suggests the following choice for $R^4_P$ and $R^5_P$. Set $R^4_P = P$ and $R^5_P = N$, a program that never halts. Here is $N(x)$: loop forever. This pair does meet the above criteria. This yields the following algorithm $A_{\text{Never-Halt}}$ for deciding Never-Halt.

$$A_{\text{Never-Halt}} \text{ simulates algorithm } A_{\text{Equal-Prog}}(\langle P, N \rangle) \text{ and outputs its result. (Note that } \langle P \rangle \text{ is the input to } A_{\text{Never-Halt}}, \text{ and } \langle N \rangle \text{ can be stored as the initial value of one of the variables of } A_{\text{Never-Halt}.}$$

Corollary 15 Equal-Prog is undecidable.