Self-reference + The Recursion Theorem

Final topic in computability: self-reference. Consider adding to TMs (or programs) a new, powerful capability to "know" and use their own descriptions.

The Recursion Theorem says that the apparent extra power does not add anything to the basic computability model! These self-referencing machines can be transformed into ordinary non-self-referencing TMs.

Topics:
1. Self-referencing machines/programs.
2. The Recursion Theorem (statement).
3. Applications of RT.
5. Proof of RT - general case.

1. Self-referencing machines/programs

Consider the program:

\[
P_1: \begin{cases} 
\text{Output } <P_1> \\
\end{cases} 
\]

This simply outputs its own description (encoding) as a string.

Simplest example of a program that uses its own description:

More interesting example:

\[
P_2: \begin{cases} 
\text{On input } W \\
\text{if } W = \epsilon \text{ then output } 0 \\
\text{else obtain } <P_2> \\
\text{run } P_2 \text{ on tail}(W) \\
\text{if } P_2 \text{ on tail}(W) \text{ outputs } n \text{ then output } n+1.
\end{cases} 
\]
P₂ computes |W|, the length of its input. It uses the recursive style common in Scheme and other recursive progbangs.

We assume that, once we have the encoding of a machine, we can emulate it on a given input.

E.g. if P₂ gets ⟨P₂⟩, it can emulate P₂ on any input (starting over with a new input).

Another example, a bit perplexing:

```
P₃: On input W:
   Obtain ⟨P₃⟩
   Run P₃ on W; if this outputs x₀ then output x₀.
```

A valid self-referencing program.

Sounds contradictory: if P₃ outputs x₀ then P₃ on W outputs x₀?

But note, according to the usual semantics of recursive calls, it never halts, so no contradiction.
2. The Recursion Theorem

Used to justify self-referential programs like $P_1, P_2, P_3$, by asserting that they have corresponding (equivalent) TMs.

Reursion Theorem: Suppose Theorem 6.3

Let $T$ be a TM that computes a (possibly partial) function

$$T : \Sigma^* \to \Sigma^*.$$ 

Then there exists another TM $R$ that computes the function

$$\tau : \Sigma^* \to \Sigma^*,$$

where for any $w$, $\tau(w) = t(\langle R \rangle, w)$.

Thus, $T$ is a TM that takes 2 inputs.

Think of the first input as the description of some arbitrary 1-input TM $M$:

Then $R$ behaves like $T$, but with

- the first input fixed to be $\langle R \rangle$,
- the description of $R$ itself:

Ex: $P_2$, revisited.

Computes length of input.

What are $T + R$?

Here is a version $T_2$ with a place for an extra input $\langle M \rangle$:
\[
T_2 : \text{ On inputs } <M> \text{ and } W : \\
\begin{align*}
\text{if } W = \epsilon & \text{ then output } 0. \\
\text{else run } M \text{ on } \text{tail}(W) & \text{ if } M \text{ on } \text{tail}(W) \text{ outputs } n \text{ then output } n+1.
\end{align*}
\]

Now, depending on what \(M\) is, \(T_2\) will produce different results.

\(t(<M>, W)\)

\textbf{Ex}: Suppose \(M\) always loops.
Then \(T_2\) on input \(\epsilon\) outputs 0.
\(T_2\) on any other input loops.

\textbf{Ex}: Suppose \(M\) always halts \& outputs 1.
Then\( \begin{align*}
T_2 \text{ on input } \epsilon & \text{ outputs } 0 \\
T_2 \text{ on any other input } & \text{ outputs } 2.
\end{align*}\)

The RT says there is some TM \(R\) computing \(t(<R>, W)\),
that is, the same as \(T_2\) but with the machine
input \(<M>\) set to \(<R>\), for the same machine \(T_2\) \& \(R\).

Plugging \(R\) into \(T_2\) for \(M\), we get \(R = P_2\) as given earlier.
3. Applications of RT
RT can be used to show various negative results, e.g. undecidability.

3.1 $A_{TM}$ is undecidable

We already knew this, but RT provides a new proof:

Suppose $D$ is a TM that decides $A_{TM}$ (for contradiction).

Construct another machine $R$ using self-reference (justified by RT):

\[
R : \text{On input $w$}
\]
\[
\text{Obtain } <R>. \quad \text{(Using RT)}
\]
\[
\text{Run } D \text{ on input } <R, w> \quad \text{(Assume we can construct for } <R>)
\]
\[
\text{Do the opposite of what } D \text{ does:}
\]
\[
\text{If } D \text{ accepts } <R, w> \text{ then reject.}
\]
\[
\text{If } D \text{ rejects } <R, w> \text{ then accept.}
\]

RT says there exists such a TM $R$ (assuming decision $D$ exists).

(Formally, to apply RT, use the 2-input TM $T$):

\[
T : \text{On inputs } <M> \text{ and } w:
\]
\[
\text{Run } D \text{ on } <M, w>
\]
\[
\text{Do the opposite ...}
\]

This yields a contradiction:

If $R$ accepts $w$ then $D$ accepts $<R, w>$ since $D$ is a decision
for $A_{TM}$
then $R$ rejects $w$ by def of $R$ above.

If $R$ does not accept $w$ then $D$ rejects $<R, w>$ since $D$ is a
decider for $A_{TM}$
then $R$ rejects $w$ by def of $R$.

Contradiction. So $D$ cannot exist, so $A_{TM}$ is undecidable.
3.2 A01 is undecidable

A similar example.

\[ A01 = \{ \langle M \rangle \mid M \text{ accepts } 01 \} \]

Proof:

Suppose D is a TM that decides A01.

Define R:

\[
R : \begin{cases} 
\text{On input } W & \text{(ignores input)} \\
\text{Obtain } \langle R \rangle & \text{(using RT)} \\
\text{Run } D \text{ on input } \langle R \rangle. & \\
\text{If } D \text{ accepts then reject} & \\
\text{If } D \text{ rejects then accept} & 
\end{cases}
\]

RT says R exists, assuming D exists.

Q: What does R do on input 01?

If R accepts 01 then D accepts \langle R \rangle since D is a decoder for A01

then R rejects 01 (and everything else)

by def. of R.

If R does not accept 01 then D rejects \langle R \rangle since D decides A01

then R accepts 01 by def. of R

Contrad, so D can't exist, so A01 is undecidable.
3.3 Using RT to prove Rice's Theorem

Recall Rice: Let $A_p = \{ < M > | L(M) \text{ satisfies } P \}$ where $P$ is a

nontrivial property of $\mathcal{R}$-recognizable languages.

(That is, $\exists \text{TM } M_1 \text{ with } L(M) \text{ satisfying } P$
$\exists \text{TM } M_2 \text{ not satisfying } P$.)

Then $A_p$ is undecidable.

We already proved this.

Now, new proof using RT:

Suppose $D$ is a decider for $A_p$.

Consider TM $R$:

\[
R: \text{on input } W: \quad \begin{cases} 
\text{Obtain } < R >. \quad \text{(using RT)} \\
\text{Run } D \text{ on input } < R >. \quad \text{If} \ \text{D accepts } < R > \text{ then run } M_2 \text{ on input } W + \text{ do the same thing.} \\
\text{If} \ \text{D rejects } < R > \text{ then run } M_1. \quad \ldots \quad \text{(M_1 + M_2 are as above, in nontriviality def.)}
\end{cases}
\]

$R$ exists, by RT.

Get contradiction by considering whether $L(R)$ satisfies $P$:

If $L(R)$ satisfies $P$,

- then $D$ accepts $< R >$ since $D$ decides $A_p$
- then $L(R) = L(M_2)$ by def. of $R$
- then $L(R)$ doesn't satisfy $P$.

If $L(R)$ doesn't satisfy $P$,

- then $D$ rejects $< R >$ since $D$ decides $A_p$
- then $L(R) = L(M_1)$
- then $L(R)$ satisfies $P$

Contradiction!
A non-T-recognizability result

Define \( \text{MIN}_{TM} = \{ <M> | M \text{ is a "minimal" } TM, \text{ that is, no } TM \text{ with a shorter encoding recognizes the same language} \} \)

Theorem: \( \text{MIN}_{TM} \) is not T-recognizable.

Note: This doesn't follow from Rice:
1. Requires "not T-recognizable" rather than "undecidable".
2. Not a language property.

Proof: Assume \( \text{MIN}_{TM} \) is T-recognizable.

Then it's enumerable, say by enumerator TM \( E \).

Consider TM \( R \):

- On input \( W \):
  - Obtain \( <R> \) (using \( RT \))
  - Run \( E \), producing list \( <M_1>, <M_2>, \ldots \) of all minimal TMs
    - Until you find some \( <M_c> \) with \( |<M_c>| > |<R>| \)
      (That is, until you find a machine with a representation bigger than yours.
      Then run \( M_c(w) \) and do the same thing.

To get contradiction:

\[ L(R) = L(M_c) \]

\( |<R>| < |<M_c>| \)

Therefore, \( <M_c> \) should not be in \( \text{MIN}_{TM} \), since \( \text{MIN}_{TM} \) contains only minimal TMs.

Contradiction.
4) Proof of RT—special case

Start with easier first step: Produce a TM corresponding to $P_1$:

\[
\begin{align*}
\left[ P_1 : & \text{ Obtain } < P_1 > \\
& \text{ Output } < P_1 > \\
\right]
\]

$P_1$ prints its own description.

Lemma: (Sipser 6.1)

There is a computable function $q : \Sigma^* \rightarrow \Sigma^*$ such that for any string $w$, $q(w)$ is the description of a TM $P_w$ that just prints out $w$ and halts.

\[
\begin{array}{c}
w \\
\downarrow \\
Q \\
\downarrow < P_w > \\
\end{array}
\]

$P_w$ has no input; just prints $w$.

Proof: Straightforward construction. Can hard-wire $w$ in the FSC of $P_w$.

Now, the machine that outputs its own description.

Consists of 2 sub-machines, called $A$ and $B$.

Output of $A$ feeds into $B$

Write as $A \circ B$. 

\[
A \xrightarrow{} B
\]
Construction of \( B \):

\[
\langle M \rangle \xrightarrow{B} \langle \langle P_{\langle M \rangle} \circ M \rangle \rangle
\]

B expects its input to be the representation of a 1-input \( TM \) (this \( M \) is a function-computer \( TM \), not a language recognizer).

If not, we don’t care what \( B \) does.

\( B \) generates (outputs) the encoding of the combination of two machines, \( P_{\langle M \rangle} \circ M \)

\[
\langle P_{\langle M \rangle} \rangle \xrightarrow{\langle M \rangle} M \xrightarrow{\text{some output}}
\]

The first is the machine \( P_{\langle M \rangle} \) which simply outputs \( \langle M \rangle \).

The second is \( M \) itself, again.

\text{Now:\quad} \begin{align*}
\text{B could generate a description of } \langle P_{\langle M \rangle} \rangle \langle P_{\langle M \rangle} \rangle, \text{ by} \\
\text{Lemma 6.1}.
\end{align*}

\begin{align*}
\text{B could generate a description of } \langle M \rangle \langle M \rangle, \text{ since it already} \\
\text{has } \langle M \rangle \text{ from its input.}
\end{align*}

\begin{align*}
\text{Once B has descriptions of } \langle P_{\langle M \rangle} \rangle \circ M, \text{ it could combine them} \\
\text{into a single description of the combined machine } P_{\langle M \rangle} \circ M,
\end{align*}

\text{which is } \langle P_{\langle M \rangle} \circ M \rangle.

Construction of \( A \):

\[
\text{Easy:\quad } \xrightarrow{A} \langle B \rangle
\]

\( A \) is just \( P_{\langle B \rangle} \), the machine that just outputs \( \langle B \rangle \), where \( B \) is the complicated machine constructed above.
5 Proof of general RT

So we have a machine that outputs its own description.

A curiosity - not the general RT that we need.

RT says not just: \( \exists \text{TM that outputs its own description}, \)
but: \( \exists \text{TMs that can use their own descriptions,} \)
in "arbitrary ways".

The "arbitrary ways" are captured by the \( T \) in the RT statement:

Recall \( \langle M \rangle \downarrow \downarrow w \)

\[ T \]

\[ t (\langle M \rangle, w) \]

Review RT statement

We must construct \( R \):

\[ R \]

\[ t (\langle R \rangle, w) \]

Construct \( R \) from \{ the given \( T \), and \}
\{ versions of \( A \) \& \( B \) from the special-case proof. \}

\( R \) will look like:

Numbers indicate which input it is, counting bottom-up.

Write this as \( A \odot (B \odot T) \) (not a complete notation system - just a convenience here)
**New A:** \( P \langle \text{BOT} \rangle \), where \( \text{BOT} \) means

\[ \begin{array}{c}
\text{B} \\
\text{T}
\end{array} \]

**New B:**

\[ \begin{array}{c}
\langle M \rangle \\
\text{B} \\
\langle P \langle M \rangle \rangle \\
\text{M}
\end{array} \]

\( M \) is a 2-input 1-output TM, as is \( \text{BOT} \).

This is:

\[ \begin{array}{c}
\langle M \rangle \\
\text{M}
\end{array} \]

a 1-input TM, which uses output of \( P \langle M \rangle \) as first input of \( M \).

Combine \( A, B, T \) as above:

\[ \begin{array}{c}
A \\
\text{B} \\
\text{T}
\end{array} \]

Trace:

\[ \begin{array}{c}
A = \langle P \langle \text{BOT} \rangle \rangle \\
\text{B} \\
\langle P \langle \text{BOT} \rangle \rangle \circ \langle \text{BOT} \rangle \\
\text{T}
\end{array} \]

Plug in \( \text{BOT} \) for \( M \) in \( B \) definition.

This = \( \langle A \circ \langle \text{BOT} \rangle \rangle \), which is just \( \langle R \rangle \).

Plug in \( P \langle \text{BOT} \rangle \circ \langle \text{BOT} \rangle \) for \( M \) in \( T \) definition.

This = \( T \langle \langle R \rangle, w \rangle \).

Thus, the entire system \( A \circ \langle \text{BOT} \rangle = R \) on input \( w \), produces \( T \langle \langle R \rangle, w \rangle \), as needed.