The Recursion Theorem

The *recursion theorem* basically states that, in designing a TM, it is possible to assume that the TM has access to its own description.

- More generally, it is possible to assume that any program can have access to its own source code.
- This is not quite trivial: consider, *e.g.*, how you might write a program that can print its own source code, (or that can recognize its own source code).
- With access to its own description, it is possible for the TM to simulate itself (*e.g.* call itself recursively).

The statement of the recursion theorem uses the notion of a *computable function*:

Def. *(Sipser 5.17):* A function $f : \Sigma^* \to \Sigma^*$ is a *computable function* if there is a Turing machine M, such that when started on any input w, machine M eventually halts with just $f(w)$ on its tape.

We can extend this definition to cover, *e.g.* *computable functions of two arguments*, by adopting some convention as to how the two arguments are given as input (*e.g.* $w_1 \# w_2$).

Theorem 6.3 (Recursion Theorem): Let T be a TM that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$. Then there is a TM R that computes a function $r : \Sigma^* \to \Sigma^*$, where for every $w \in \Sigma^*$:

$$r(w) = t(\langle R \rangle, w).$$

Moreover, the mapping from $\langle T \rangle$ to $\langle R \rangle$ is computable.

Interpretation:

- To construct a TM R that can compute with its own description, we need only construct a TM T that expects a TM description as an extra argument.
- It is simply a matter of "turning the crank" to obtain the description $\langle R \rangle$ from the description $\langle T \rangle$.

Proof Sketch (following Sipser)

Construct R in three parts (uses "pipeline notation" to suggest the construction):

- A is print "$\langle B \mid T\rangle$". The description $\langle B \mid T\rangle$ is "hard-coded" into A's control, so we need B to obtain A.
- B inputs a TM description $\langle X\rangle$ and prints TM description $\langle$print "$\langle X\rangle$" $\mid X\rangle$. So B is: read$\langle X\rangle$; print $\langle$print "$\langle X\rangle$" $\mid X\rangle$
- R is the TM $A \mid B \mid T$.

Note: If B inputs $\langle B \mid T\rangle$, then it prints $\langle$print "$\langle B \mid T\rangle$" $\mid B \mid T\rangle$ (*i.e.* $\langle A \mid B \mid T\rangle$, which is $\langle R\rangle$). So R sends $\langle R\rangle$ to T.

Illustration in BASH

The BASH function `REC` below computes code for a command R from code for a command T.

```
REC()
{
    T="$(cat)"
    B='(X="$(cat)"; echo "echo $(printf %q "$X") | $X")'
    A="echo $(printf %q "$B | $T")"
    echo "$A | $B | $T"
}
```

- T reads std. input, writes to std. output.
- B expects code `X` on std. input, emits `A | X` on std. output. Assuming `X` is `B | T`, then it outputs `A | B | T`.
- A prints `B | T`, which has been hard-coded. The code for A is constructed from that for B and T (*cf.* Lemma 6.1).

```
-----------------------------------
Print own code
-----------------------------------
T:       cat

R:       echo \(X=\"\$\(cat\)\"\;\ echo\ \"echo\ \$\(printf\ %q\ \"\$X\"\)\
         \|\ \$X\"\)\ \|\ cat | (X="$(cat)"; echo "echo $(printf %q "$X") | $X")
         | cat

eval R:  echo \(X=\"\$\(cat\)\"\;\ echo\ \"echo\ \$\(printf\ %q\ \"\$X\"\)\
         \|\ \$X\"\)\ \|\ cat | (X="$(cat)"; echo "echo $(printf %q "$X") | $X")
         | cat
-----------------------------------
```

```
-----------------------------------
Print own code, reversed
-----------------------------------
T:       rev

R:       echo \(X=\"\$\(cat\)\"\;\ echo\ \"echo\ \$\(printf\ %q\ \"\$X\"\)\
         \|\ \$X\"\)\ \|\ rev | (X="$(cat)"; echo "echo $(printf %q "$X") | $X")
         | rev

eval R:  ver | )"X$ | )"X$" q% ftnirp($ ohce" ohce ;")tac($"=X( | ver \|\ \)
         \"\X$\ \|\ \)\"\X$\"\ \q% \ftnirp(\$\ \ohce"\ \ohce \;\"\)\tac(\$\"\=X(
         \ ohce
-----------------------------------
```

```
-----------------------------------
Print own code, uuencoded
-----------------------------------
T:       uuencode -

R:       echo \(X=\"\$\(cat\)\"\;\ echo\ \"echo\ \$\(printf\ %q\ \"\$X\"\)\
         \|\ \$X\"\)\ \|\ uuencode\ - | (X="$(cat)"; echo "echo $(printf %q "$X"
         | uuencode -

W:

eval R:  begin 664 -
M96-H;R!<*%@]7")<)%PH8V%T7"E<(EP[7"!E8VA07"!<(F5C:&]<(%PD7"AP
M<FEN=&9<("5Q7"!<(EPD6%PB7"E<(%Q\7"!<)%A<(EPI7"!<?%P@=75E;F-O
M9&5<("T@?"`H6#TB)"AC870I(CL@96-H;R`B96-H;R`D*'!R:6YT9B`E<2`B
9)%@B*2!\("18(BD@?"!U=65N8V]D92`M"@``
`
end
-----------------------------------
```

```
-----------------------------------
Recognizes own code (positive test)
-----------------------------------
T:       (R=$(cat); if [ X"$R" = X"$W" ]; then echo YES; else echo NO; fi)

R:       echo \(X=\"\$\(cat\)\"\;\ echo\ \"echo\ \$\(printf\ %q\ \"\$X\"\)\
         \|\ \$X\"\)\ \|\ \(R=\$\(cat\)\;\ if\ \[\ X\"\$R\"\ =\ X\"\$W\"\ \]\;\
         then\ echo\ YES\;\ else\ echo\ NO\;\ fi\) | (X="$(cat)"; echo "echo
         $(printf %q "$X") | $X") | (R=$(cat); if [ X"$R" = X"$W" ];
         then echo YES; else echo NO; fi)

W:       echo \(X=\"\$\(cat\)\"\;\ echo\ \"echo\ \$\(printf\ %q\ \"\$X\"\)\
         \|\ \$X\"\)\ \|\ \(R=\$\(cat\)\;\ if\ \[\ X\"\$R\"\ =\ X\"\$W\"\ \]\;\
         then\ echo\ YES\;\ else\ echo\ NO\;\ fi\) | (X="$(cat)"; echo "echo
         $(printf %q "$X") | $X") | (R=$(cat); if [ X"$R" = X"$W" ];
         then echo YES; else echo NO; fi)

eval R:  YES
-----------------------------------
```

```
-----------------------------------
Recognizes own code (negative test)
-----------------------------------
T:        (R=$(cat); if [ X"$R" = X"$W" ]; then echo YES; else echo NO; fi)

R:        echo \(X=\"\$\(cat\)\"\;\ echo\ \"echo\ \$\(printf\ %q\ \"\$X\"\)\
          \|\ \$X\"\)\ \|\ \(R=\$\(cat\)\;\ if\ \[\ X\"\$R\"\ =\ X\"\$W\"\ \]\;\
          then\ echo\ YES\;\ else\ echo\ NO\;\ fi\) | (X="$(cat)"; echo "echo
          $(printf %q "$X") | $X") | (R=$(cat); if [ X"$R" = X"$W" ];
          then echo YES; else echo NO; fi)

W:        Not my code

eval R:   NO
-----------------------------------
```

Applications of the Recursion Theorem

The Recursion Theorem allows us to construct Turing machines that are able to obtain and use their own descriptions, *e.g.*:

$SELF =$ "*On any input:*

1. Obtain, via the recursion theorem, own description $\langle SELF \rangle$.
2. Print $\langle SELF \rangle$."

We could replace (2) by *Print* $f(\langle SELF \rangle)$, where f is any computable function.

Def. *(Sipser 6.5):* A_{TM} is undecidable.

We already have proved this, but we can get a simpler proof using the Recursion Theorem.

Proof: Assume some H decides A_{TM}. Construct TM B as follows:

B = "*On input w:*

- Obtain, via the Recursion Theorem, own description $\langle B \rangle$.
- Run H on input $\langle B, w \rangle$.
- Do the opposite of what H says (*i.e.* accept if H rejects and reject if H accepts).

But then running B on input w does the opposite of what H says, so H cannot be deciding A_{TM}. Contradiction! cannot be

Def. *(Sipser 6.6):* A TM M is *minimal* if there is no TM N equivalent to M such that the length of $\langle N \rangle$ is less than $\langle M \rangle$.

$$MIN_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a minimal TM}\}$$

Thm. *(Sipser 6.7):* MIN_{TM} is not Turing-recognizable.

Proof: Assume some TM E enumerates MIN_{TM}. Construct TM C as follows:

C = "*On input w:*

- Obtain, via the Recursion Theorem, own description $\langle C \rangle$.
- Run E until it outputs $\langle D \rangle$ where $\langle D \rangle$ is longer than $\langle C \rangle$.
- Simulate D on input w."

Then C is equivalent to D, but has a shorter description. Contradiction!