Lecture 12
Why TM’s?
Programs are OK too

Fix $\Sigma = \text{printable ASCII}$

Programming language with ints, strings & function calls

“Computable function” = always returns something

“Decider” = computable function always returning 0 / 1

“Acceptor” = accept if return 1; reject if $\neq 1$ or loop

$A_{\text{Prog}} = \{ <P,w> | \text{ program } P \text{ returns 1 on input } w \}$

$\text{HALT}_{\text{Prog}} = \{ <P,w> | \text{ prog } P \text{ returns something on } w \}$

...
\( A_{TM} (\leq_T v s \leq_m) \text{ HALT}_{TM} \)

\[
f(<M, w>) = <M', w>
\]

From Lecture 07
\[ A_{\text{Prog}} \leq_m \text{HALT}_{\text{Prog}} \]

\[ f(<P,w>) = <P',w> \]

```
sub f(P,w){
    // build P'
    pn = ...(find P's name)
    pp = "sub " + pn + "prime(x){"
    pp += P
    pp += "if " + pn + "(x) return 1;"
    pp += "while True {;;}";
    val = "<" + pp + "," + w + ">
    return val
}
```

```
sub Pprime(x){
    sub P(y){
        ...
        (copy of P)
    }
    if P(x) return 1;
    while True { ; ; }
}
```
Programs vs TMs

Everything we’ve done re TMs can be rephrased re programs. From the Church-Turing thesis (hopefully made concrete in earlier HW) we know they are equivalent.

Above example shows some things are easier with programs. Others get harder (e.g., “Universal TM” is a Java interpreter written in Java; “configurations” and “computation histories” are much messier).

TMs are convenient to use here since they strike a good balance. But I hope you can mentally translate between the two; decidability/undecidability of various properties of programs are obviously more directly relevant.
Mapping Reducibility

Defn: A is *mapping reducible* to B (A \( \leq_m B \)) if there is computable function \( f \) such that \( w \in A \iff f(w) \in B \)

A special case of \( \leq_T \):

Call subr only once; its answer is *the* answer

**Theorem:**

\( A \leq_m B \) & B *decidable* (recognizable) \( \Rightarrow \) A is too

\( A \leq_m B \) & A *undecidable* (*un*)recognizable) \( \Rightarrow \) B is too

\( A \leq_m B \) & B \( \leq_m C \) \( \Rightarrow \) A \( \leq_m C \)

*Most reductions we’ve seen were actually \( \leq_m \) reductions.*
Other Examples of $\leq_m$

$A_{TM} \leq_m REGULAR_{TM}$

$$f(<M,w>) = <M_2>$$

Build $M_2$ so $L(M_2) = \Sigma^* / \{ 0^n | n \}$, as $M$ accept/rejects $w$

$EMPTY_{TM} \leq_m EQ_{TM}$

$$f(<M>) = <M, M_{\text{reject}}>$$

$L(M_{\text{reject}}) = \emptyset$, so equiv to $M$ iff $L(M) = \emptyset$

$A_{TM} \leq_m MPCP$

$MPCP \leq_m PCP$

$A_{TM} \leq_m \underline{EMPTY}_{TM}$

$$f(<M,w>) = <M_1>$$

Build $M_1$ so $L(M_1) = \{w\} / \emptyset$, as $M$ accept/rejects $w$
**EMPTY\textsubscript{TM} is undecidable**

\[
\text{EMPTY}\textsubscript{TM} = \{ <M> | M \text{ is a TM s.t. } L(M) = \emptyset \}
\]

**Pf:** To show: \( A\textsubscript{TM} \leq_T \text{EMPTY}\textsubscript{TM} \)

On input \(<M,w>\) build \(M'\):

Do not run \(M\) or \(M'\). (That whole “halting thing” means we might not learn much if we did.) But note that \(L(M')\) is/is not empty exactly when \(M\) does not/does accept \(w\), so knowing whether \(L(M') = \emptyset\) answers whether \(<M,w>\) is in \(A\textsubscript{TM}\).

And our hypothetical “\(\text{EMPTY}\textsubscript{TM}\)” subroutine applied to \(M'\) tells us just that. I.e., \(A\textsubscript{TM} \leq_T \text{EMPTY}\textsubscript{TM}\)

**NB:** it shows \(A\textsubscript{TM} \leq_m (\text{EMPTY}\textsubscript{TM})^c\)

---

From Lecture 07

---

\[
M' \text{ on input } x:
\begin{align*}
1. & \text{ erase } x \\
2. & \text{ write } w \\
3. & \text{ run } M \text{ on } w \\
4. & \text{ if } M \text{ accepts } w, \text{ then accept } x \\
5. & \text{ otherwise, reject } x
\end{align*}
\]

\[
L(M') = \begin{cases} 
\Sigma^*, & \text{if } M \text{ accepts } w \\
\emptyset, & \text{if } M \text{ rejects } w
\end{cases}
\]
REGULAR$_{TM}$ is undecidable

REGULAR$_{TM} = \{ <M> | \text{ M is a TM s.t. } L(M) \text{ is regular } \}$

Pf: To show: $A_{TM} \leq_T \text{REGULAR}_{TM}$

On input $<M,w>$ build $M'$:
Do not run $M$ or $M'$. (That whole “halting thing” ...) But note that $L(M')$ is/is not regular exactly when $M$ does/does not accept $w$, so knowing whether $L(M')$ is regular answers whether $<M,w>$ is in $A_{TM}$. The hypothetical “REGULAR$_{TM}$” subroutine applied to $M'$ tells us just that. I.e., $A_{TM} \leq_T \text{REGULAR}_{TM}$

M’ on input $x$:
1. if $x \in \{0^n1^n | n \geq 0 \}$, accept $x$
2. otherwise, erase $x$
3. write $w$
4. run $M$ on $w$
5. if $M$ accepts $w$, then accept $x$
6. otherwise, reject $x$

$L(M') = \begin{cases} \Sigma^*, & \text{if } M \text{ accepts } w \\ \{0^n1^n | n \geq 0 \}, & \text{otherwise} \end{cases}$

Exercise: Is it $A_{TM} \leq_m \text{REGULAR}_{TM}$? If not, could it be changed?

From Lecture 07
More on $\leq_T$ vs $\leq_m$

Theorem: For any $L$, $L \leq_T \overline{L}$

*The same is not true of* $\leq_m$:

Theorem: $L$ recognizable and $L \leq_m \overline{L} \Rightarrow L$ is decidable.

Proof: on input $x$, dovetail recognizers for $x \in L$ & $f(x) \in L$

Corr: $A_{TM} \leq_T \overline{A}_{TM}$ but *not* $A_{TM} \leq_m \overline{A}_{TM}$

Theorem: $A \leq_m B$ iff $\overline{A} \leq_m \overline{B}$

Theorem: If $L$ is not recognizable and both $L \leq_m B$ and $L \leq_m \overline{B}$, then neither $B$ nor $\overline{B}$ are recognizable
EQ$_{TM}$ is neither recognizable nor co-recognizable

M$_0$: on any input x, reject x. L(M$_0$) = ∅

M$_1$: on any input x, accept x. L(M$_1$) = $\Sigma^*$

For any <M,w>, let h(<M,w>) = M$_2$ be the TM that, on input x,

1. runs M on w
2. if M accepts w, then accept x.

Claim: L(M$_2$) = $\Sigma^*$ (if <M,w> $\in$ A$_{TM}$), else = ∅ & h computable

Then A$_{TM}$ $\leq_m$ EQ$_{TM}$ via g(<M,w>) = <M$_0$,M$_2$>

And A$_{TM}$ $\leq_m$ EQ$_{TM}$ via f(<M,w>) = <M$_1$,M$_2$> (& A$_{TM}$ $\leq_m$ EQ$_{TM}$)