CSE 431

Larry Ruzzo, Spring 2010

http://www.cs.washington.edu/431

Initial Reading Assignment: Sipser Chapter 3

\[ x - 17 = 0 \]
\[ 2x - 17 = 0 \]
\[ 3x^2 - 17x + 52 = 0 \]
\[ x = \frac{b \pm \sqrt{b^2 - 4ac}}{2a} \]

\[ x^2 + y^2 = z^2 \quad z^2 = 10^2 \]

Diophantine equation

With basic 10x

\[ 19x^2 + 20x^2 + 10x^2 \quad \text{and...} \]

\[ \text{Quadratic Diophantine Eqns.} \]

Regular type: \( U \) is a complete \( = \phi \)

True \( > 2^2 + 2^2 + 2^2 \quad \text{any finite} \quad k \).
Algorithms

“An algorithm is a finite, precise set of instructions for performing a computation”

```
192 192
 7   9
199 201
```

“The Division Algorithm”: \( \forall a \in \mathbb{Z}, d \in \mathbb{Z}^+ \),
\( \exists \) unique \( q, r \) such that \( 0 \leq r < d \) and \( a = qr + d \)
Lecture 3

Example

\[ L = \{ w \# w \mid w \in \{0, 1\}^* \} \]

1. check that there's a single #
2. read | remember & cross off left-most letter
3. scan to # & compare next letter
4. if OK, cross it off
5. repeat
By definition, no transitions out of $q_{acc}, q_{rej}$:

- **M halts** if (and only if) it reaches either $q_{acc}$ or $q_{rej}$;
- **M loops** if it never halts ("loop" might suggest "simple", but non-halting computations may of course be arbitrarily complex);
- **M accepts** if it reaches $q_{acc}$,
- **M rejects** by halting in $q_{rej}$ or by looping.

The language recognized by $M$:

$L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$

L is *Turing recognizable* if $\exists M$ s.t. $L = L(M)$

L is *Turing decidable* if, furthermore, M halts on all inputs

* A key distinction!

---

**Example**

$L = \{ w \# w \mid w \in \{0,1\}^* \}$

1. check that there’s a single #
2. read, remember & cross off left-most letter
3. scan to # & compare next letter
4. if OK, cross it off
5. reject
Church-Turing Thesis

TM's formally capture the intuitive notion of “algorithmically solvable”

Not provable, since “intuitive” is necessarily fuzzy.

But, give support for it by showing that
   (a) other intuitively appealing (but formally defined)
       models are precisely equivalent (rest of lecture), and
   (b) models that are provably different are unappealing,
       either because they are too weak (e.g., DFA’s) or too
       powerful (e.g., a computer with a “solve-the-halting-problem”
       instruction).
**Announcements**

Late policy

eTurnin

Office hours M 2:30, W 12:30, Th 5:00

Midterm Fri 5/7, probably

---

**Lecture 4**

**Nondeterministic Turing Machines**

\[ \delta: Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L,R\}) \]

Accept if any path leads to \( q_{\text{accept}} \); reject otherwise, (i.e., all halting paths lead to \( q_{\text{reject}} \))
Simulating an NTM

Key issue: avoid getting lost on $\infty$ path
Key Idea: breadth-first search

A TM “Enumerator”

L Turing recognizable iff a TM enumerates it

$(\Leftarrow)$: Run enumerator, compare each “output” to input; accept if they match (reject by not halting if input never appears)

$(\Rightarrow)$: The “obvious” idea: enumerate $\Sigma^*$, run the recognizer on each, output those that are accepted.

L Turing recognizable iff a TM enumerates it

$(\Rightarrow)$: A better idea—“dovetailing”:
For $i = 0, 1, 2, 3, ...$

At stage $i$, run the recognizer for $i$ steps on each of the first $i$ strings in $\Sigma^*$, output any that are accepted.

[Oops, doesn’t work... may not halt...]
### Encoding things

\[
G = (V, \Sigma, R, S) ; \quad \langle G \rangle = \langle (S, A, B, \ldots), (a, b, \ldots), (S \rightarrow aA, S \rightarrow b, A \rightarrow cAb, \ldots), S \rangle
\]

or

\[
\langle G \rangle = \langle (A_0, A_1, \ldots), (a_0, a_1, \ldots), (A_0 \rightarrow a_0 A_1, A_0 \rightarrow a_1, A_1 \rightarrow a_2 A_1 a_1, \ldots), A_0 \rangle
\]

\[
\Sigma = ?
\]

### Decidability

Recall: \( L \) decidable means there is a TM recognizing \( L \) that always halts.

Example:

“The acceptance problem for DFAs”

\( A_{\text{DFA}} = \{ \langle D, w \rangle \mid D \text{ is a DFA} \text{ & } w \in L(D) \} \)

### Some Decidable Languages

The following are decidable:

\( A_{\text{DFA}} = \{ \langle D, w \rangle \mid D \text{ is a DFA} \text{ & } w \in L(D) \} \)

\( \text{pf: simulate } D \text{ on } w \)

\( A_{\text{NFA}} = \{ \langle N, w \rangle \mid N \text{ is an NFA} \text{ & } w \in L(N) \} \)

\( \text{pf: convert } N \text{ to a DFA, then use previous as a subroutine} \)

\( A_{\text{REX}} = \{ \langle R, w \rangle \mid R \text{ is a regular expr} \text{ & } w \in L(R) \} \)

\( \text{pf: convert } R \text{ to an NFA, then use previous as a subroutine} \)

\( \text{EMPTY}_{\text{DFA}} = \{ \langle D \rangle \mid D \text{ is a DFA} \text{ and } L(D) = \emptyset \} \)

\( \text{pf: is there no path from start state to any final state?} \)

\( \text{EQ}_{\text{DFA}} = \{ \langle A, B \rangle \mid A \text{ & } B \text{ are DFAs s.t. } L(A) = L(B) \} \)

\( \text{pf: equal iff } L(A) \oplus L(B) = \emptyset, \text{ and } x \oplus y = (x \cap y) \cup (x \cap \overline{y}) \text{, and regular sets are closed under } \cup, \cap, \overline{\cdot} \)

\( A_{\text{CFG}} = \{ \langle G, w \rangle \mid \ldots \} \)

\( \text{pf: see book} \)

\( \text{EMPTY}_{\text{CFG}} = \{ \langle G \rangle \mid \ldots \} \)

\( \text{pf: see book} \)
EQ_{\text{CFG}} = \{ <A,B> \mid A \text{ & } B \text{ are CFGs s.t. } L(A) = L(B) \}

This is **NOT** decidable

The Acceptance Problem for TMs

\[ A_{\text{TM}} = \{ <M,w> \mid M \text{ is a TM & } w \in L(M) \} \]

**Theorem**: \( A_{\text{TM}} \text{ is Turing recognizable} \)

**Pf**: It is recognized by a TM \( U \) that, on input \( <M,w> \), simulates \( M \) on \( w \) step by step. \( U \) accepts iff \( M \) does. 

\( U \) is called a **Universal Turing Machine**

(Ancestor of the stored-program computer)

Note that \( U \) is a recognizer, not a decider.
Cardinality

Two sets have equal cardinality if there is a bijection between them.

A set is countable if it is finite or has the same cardinality as the natural numbers.

Examples:
  - $\Sigma^*$ is countable (think of strings as base-$|\Sigma|$ numerals)
  - Even natural numbers are countable: $f(n) = 2n$
  - The Rationals are countable

More cardinality facts

If $f: A \to B$ in an injective function ("1-1", but not necessarily "onto"), then

$$|A| \leq |B|$$

(Intuitive: $f$ is a bijection from $A$ to its range, which is a subset of $B$, & $B$ can’t be smaller than a subset of itself.)

Theorem (Cantor-Schroeder-Bernstein):

If $|A| \leq |B|$ and $|B| \leq |A|$ then $|A| = |B|$
The Reals are Uncountable

Suppose they were
List them in order
Define X so that its $i^{th}$ digit $\neq i^{th}$ digit of $i^{th}$ real
Then X is not in the list
Contradiction

A detail: avoid .000..., .9999... in X

“Most” languages are neither Turing recognizable nor Turing decidable

Pf:
“< >” maps TMs into $\Sigma^*$, a countable set, so the set of TMs, and hence of Turing recognizable languages is also countable; Turing decidable is a subset of Turing recognizable, so also countable. But by the previous result, the set of all languages is uncountable.

Number of Languages in $\Sigma^*$ is Uncountable

Suppose they were
List them in order
Define L so that $w_i \in L \iff w_i \notin L$
Then L is not in the list
Contradiction

A specific non-Turing-recognizable language

Let $M_i$ be the TM encoded by $w_i$, i.e. $<M_i> = w_i$
(M_i = some default machine, if $w_i$ is an illegal code.)
i, j entry tells whether $M_i$ accepts $w_j$
Then $L_D$ is not recognized by any TM
Theorem: The class of Turing recognizable languages is not closed under complementation.

Proof:
The complement of $D$, is Turing recognizable:
On input $w_i$, run $<M_i>$ on $w_i$ ($= <M_i>$); accept if it does. E.g. use a universal TM on input $<M_i,<M_i>$. 

E.g., in previous example, $D^c$ might be $L(M_6)$

Theorem: The class of Turing decidable languages is closed under complementation.

Proof:
Flip $q_{accept}, q_{reject}$

Decidable $\subsetneq$ Recognizable

Lecture 6
The Acceptance Problem for TMs

\[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM} \land w \in L(M) \} \]

Theorem: \( A_{TM} \) is Turing recognizable

Proof: It is recognized by a TM \( U \) that, on input \( \langle M, w \rangle \), simulates \( M \) on \( w \) step by step. \( U \) accepts iff \( M \) does. □

\( U \) is called a Universal Turing Machine
(Ancestor of the stored-program computer)

Note that \( U \) is a recognizer, not a decider.

A specific non-Turing-recognizable language

<table>
<thead>
<tr>
<th>( w_1 )</th>
<th>( w_2 )</th>
<th>( w_3 )</th>
<th>( w_4 )</th>
<th>( w_5 )</th>
<th>( w_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \langle M_1 \rangle )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( \langle M_2 \rangle )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( \langle M_3 \rangle )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \langle M_4 \rangle )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \langle M_5 \rangle )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Then \( L_D \) is not recognized by any TM.

Note: The above TM \( D \), if it existed, would recognize exactly the language \( L_D \) defined in this diagonalization proof (which we already know is not recognizable).

\( L_D \) is not recognizable

Decidable \( \subset \neq \) Recognizable
Decidable = Rec \cap \text{co-Rec}

L decidable iff both L & L^c are recognizable

\[\begin{align*}
\text{recognizable} & \quad \text{co-recognizable} \\
\text{decidable} & \quad \\
\end{align*}\]

Pf:

(\Rightarrow) on any given input, dovetail a recognizer for L with one for L^c; one or the other must halt & accept, so you can halt & accept/reject appropriately.

(\Leftarrow): from last lecture, decidable languages are closed under complement (flip acc/rej)

The Halting Problem

HALT_{TM} = \{ <M,w> | \text{TM M halts on input w} \}

Theorem: The halting problem is undecidable

Proof:

A = A_{TM}, B = HALT_{TM} Suppose I can reduce A to B. We already know A is undecidable, so must be that B is, too.

Suppose TM R decides HALT_{TM}. Consider S:

On input <M,w>, run R on it. If it rejects, halt & reject; if it accepts, run M on w; accept/reject as it does.

Then S decides A_{TM}, which is impossible. R can’t exist.

Reduction

“A is reducible to B” means I could solve A if I had a subroutine for B

Ex:
Finding the max element in a list is reducible to sorting
pf: sort the list in increasing order, take the last element (A big hammer for a small problem, but never mind...)

Lecture 7
Reduction

"A is reducible to B" means I could solve A if I had a subroutine for B

Ex:
Finding the max element in a list is reducible to sorting
pf: sort the list in increasing order, take the last element
(A big hammer for a small problem, but never mind...)

The Halting Problem

\[ \text{HALT}_{TM} = \{ <M,w> \mid \text{TM } M \text{ halts on input } w \} \]

Theorem: The halting problem is undecidable
Proof:
\[ A = A_{TM}, B = \text{HALT}_{TM} \]
Suppose I can reduce A to B. We already know A is undecidable, so must be that B is, too.

Suppose TM R decides \( \text{HALT}_{TM} \). Consider S:
On input \(<M,w>\), run R on it. If it rejects, halt & reject; if it accepts, run M on w; accept/reject as it does.
Then S decides \( A_{TM} \), which is impossible. R can't exist.

Another Way

Rather than running R on \(<M,w>\), and manipulating that answer; manipulate the input to build a new \( M' \) so that R’s answer about \(<M',w>\) directly answers the question of interest.

Specifically, build \( M' \) as a clone of M, but modified so that if M halts-and-rejects, \( M' \) instead rejects by looping.
Then halt/not-halt for \( M' \) == accept/reject for M

Again, this reduces \( A_{TM} \) to \( \text{HALT}_{TM} \)
**Reduction**

Notation (not in book, but common):

\[ A \leq_T B \text{ means "A is Turing Reducible to B"} \]

i.e., if I had a TM deciding B, I could use it as a subroutine to solve A

Facts:

\[ A \leq_T B \text{ & } B \text{ decidable implies } A \text{ decidable} \]  
(definition)

\[ A \leq_T B \text{ & } A \text{ undecidable implies } B \text{ undecidable} \]  
(contrapositive)

\[ A \leq_T B \text{ & } B \leq_T C \text{ implies } A \leq_T C \]

---

**EMPTY\textsubscript{TM} is undecidable**

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

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

On input \(<M,w>\) build \( M' \):  
1. 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 \( A_{\text{TM}} \).  
2. And our hypothetical “EMPTY\textsubscript{TM}" subroutine applied to \( M' \) tells us just that. i.e., \( A_{\text{TM}} \leq_T \text{EMPTY}\textsubscript{TM} \)

**REGULAR\textsubscript{TM} is undecidable**

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

Pf: To show: \( A_{\text{TM}} \leq_T \text{REGULAR}\textsubscript{TM} \)

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

\[ L(M') = \begin{cases} 
\Sigma^*, \text{ if } M \text{ accepts } w \\
\emptyset, \text{ if } M \text{ rejects } w
\end{cases} \]
Announcements

re HW#1, Aeron says “If I made a comment, even if I didn’t take off points this time, people should pay attention because I will take off points for the same mistake in the future...”

Lecture 8

$EQ_{TM}$ is undecidable

$EQ_{TM} = \{ <M_1, M_2> | M_i \text{ are TMs s.t. } L(M_1) = L(M_2) \}$

Pf: Will show $EMPTY_{TM} \leq_T EQ_{TM}$

Suppose $EQ_{TM}$ were decidable. Let $M_\emptyset$ be a TM that accepts nothing, say one whose start state is $q_{\text{reject}}$.

Consider the TM $E$ that, given $<M>$, builds $<M, M_\emptyset>$, then calls the hypothetical subroutine for $EQ_{TM}$ on it, accepting/rejecting as it does. Now, $<M, M_\emptyset> \in EQ_{TM}$ if and only if $M$ accepts $\emptyset$, so, $E$ decides whether $M \in EMPTY_{TM}$, which we know to be impossible. Contradiction
Linear Bounded Automata

Like a (1-tape) TM, but tape only long enough for input
(head stays put if try to move off either end of tape)

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}) \]

\[ L(M) = \{ x \in \Sigma^* | M \text{ accepts } x \} \]

Finite state control

An Aside: The Chomsky Hierarchy

TM = phrase structure grammars \[ \alpha A \beta \rightarrow \alpha \gamma \beta \]
LBA = context-sensitive grammars \[ \alpha A \beta \rightarrow \alpha \gamma \beta, \ \gamma \neq \varepsilon \]
PDA = context-free grammars \[ A \rightarrow \gamma \]
DFA = regular grammars \[ A \rightarrow abcB \]

ALBA is decidable

\[ ALBA = \{ <M, w> | M \text{ is an LBA and } w \in L(M) \} \]

Key fact: the number of distinct configurations of an LBA on any input of length \( n \) is bounded, namely

\[ \leq n |Q| |\Gamma|^n \]

If \( M \) runs for more than that many steps, it is looping

Decision procedure for \( ALBA \):

Simulate \( M \) on \( w \) and count steps; if it halts and accepts/rejects, do the same; if it exceeds that time bound, halt and reject (it's looping).

EMPTY_LBA is undecidable

Why is this hard, when the acceptance problem is not?

Loosely, it's about infinitely many inputs, not just one
Can we exploit that, say to decide \( A_{TM} \)?
An idea. An LBA is a TM, so can it simulate \( M \) on \( w \)?
Only if \( M \) doesn't use too much tape.
What about simulating \( M \) on \( w########### \)?
Given \( M \), build LBA \( M' \) that, on input \( w \# \# \# \# \ldots \# \), simulates \( M \) on \( w \), treating \# as a blank. If \( M \) halts, do the same. If \( M \) tries to move off the right end of the tape, reject.

\[
L(M') = \{ w\#^k | M \text{ accepts } w \text{ using } \leq |w\#^k| \text{ tape cells} \}
\]

Key point:
- if \( M \) rejects \( w \), \( M' \) rejects \( w\#^k \) for all \( k \), \( \therefore L(M') = \emptyset \)
- if \( M \) accepts \( w \), some \( k \) will be big enough, \( \therefore L(M') \neq \emptyset \)
EMPTY\textsubscript{LBA} is undecidable

An alternate proof, using a new technique –

Computation Histories

\textit{Configuration:} state, head, tape

\textit{Encoding Configs:}\begin{align*}
0 & 1 & q & 0 & 1 & 1 \\
\end{align*}

A string in $\Gamma^* \mathcal{Q} \Gamma^*$ (trailing blanks optional)

Accepting (Rejecting) \textit{History:} $C_1, C_2, \ldots, C_n$ s.t.
1. $C_1$ is M’s initial configuration
2. $C_n$ is an accepting (rejecting, resp.) config, and
3. For each $1 \leq i < n$, $C_i$ moves to $C_{i+1}$ in one step
Checking Histories

Many proofs require checking that a string, say

\[ #C_1#C_2#\ldots#C_n# \text{ in } (\{\#\} \cup Q \cup \Gamma)^* \]

is/is not an accepting history:

1. \( C_1 \) is \( M \)'s initial configuration:
   \[ C_1 \in q_0 \Sigma^* \]
2. \( C_n \) is an accepting config:
   \[ \text{it contains } q_{\text{accept}} \]
3. For each \( 1 \leq i < n \), \( C_i \) moves to \( C_{i+1} \) in one step

\[ \text{“} C_i \text{ moves to } C_{i+1} \text{ in one step of } M \text{”} \]

\[ #a_1a_2\ldots a_ka_{k+1}a_{k+2}\ldots a_n#b_1b_2\ldots b_jb_{j+1}b_{j+2}\ldots b_m# \]

No change:
- \( a_i = b_i \in \Gamma \)
- \( p, q \in Q \)
- \( j = k \pm 1 \)
- \( n = m \)

Except for adjustments, all near the head, reflecting the move:
\[ \delta(p, a_{k+1}) = (q, b_{k+1}, L/R), \]
\[ j = k+1 \text{ if } R \text{ else } \max(k-1,0) \]
and injecting blanks on the right as needed:
- if \( n = k \), then “a_{k+1}” = blank
- \( m = n+1 \), ...

Aside: one reason TM's have been so useful for computation theory is that they make questions like this very simple; “config” and “move” are much messier for “real” computers.

A_{TM} \leq_T \text{EMPTY}_{LBA}

Given \( \langle M, w \rangle \), build an LBA \( L_{M,w} \) that recognizes

\[ \text{AH}_{M,w} = \{ x \mid x = #C_1#C_2#\ldots#C_n#, \text{ an Accepting computation History of } M \text{ on } w \} \]

Then pass \( \langle L_{M,w} \rangle \) to the hypothetical subr for EMPTY_{LBA}

Specifically, \( L_{M,w} \) operates by checking that:
1. Its input is of the form \( #C_1#C_2#\ldots#C_n# \)
2. \( C_1 \) is the initial config of \( M \) on \( w \)
3. \( C_n \) has \( M \)'s accept state, and
4. For each \( 1 \leq i < n \), \( C_i \) moves to \( C_{i+1} \) in one step of \( M \)
   (ziz-zag across adjacent pairs, checking as on prev slide)

Correctness

\[ L(L_{M,w}) = \text{AH}_{M,w} = \{ x \mid x = #C_1#C_2#\ldots#C_n#, \text{ an accepting computation history of } M \text{ on } w \} \]

Empty if \( M \) rejects \( w \) – no such \( x \)
Non-empty if \( M \) accepts \( w \) – there is one such history

So, “\( M \) accepts \( w \)” is equivalent to (non-)emptiness of \( \text{AH}_{M,w} \)

\[ \therefore A_{TM} \leq_T \text{EMPTY}_{LBA} \quad \Box \]
Notes

Similar ideas can be used to give reductions like

\[ A_{TM} \leq_T \text{EMPTY}_X \]

for any machine or language class \( X \) expressive enough that we can easily, given \( M \) & \( w \), represent \( \text{AH}_{M,w} \) in \( X \)

A nice thing about histories is that they are so transparent that this is easy, even for more restricted models than LBA's

(One example in homework; another below)

\[ A_{TM} \leq_T \text{ALL}_{CFG} \]

Given \( M, w \), build a PDA \( P \) that, on input \( x \), accepts if \( x \) does not start and end with \#; otherwise, let

\[ x = \# C_1 \# C_2^{R} \# C_3 \# C_4^{R} \ldots \# C_n^{(R)} \# \]

and nondeterministically do one of:

1. accept if \( C_1 \) is not \( M \)'s initial config on \( w \)
2. accept if \( C_n \) is not accepting, or
3. nondeterministically pick \( i \) and verify that \( C_i \) does not yield \( C_{i+1} \) in one step. (Push 1\(^{st} \); pop & compare to 2\(^{nd} \), with the necessary changes near the head.)

From \( P \), build equiv CFG \( G \); ask the hypothetical ALL\(_{CFG} \) subr if \( G \) generates all of \( (\{\#\} \cup Q \cup \Gamma)^* \)

ALL\(_{CFL} \) is Undecidable

\[ \text{ALL}_{CFG} = \{ <G> | G \text{ is a CFG with } L(G) = \Sigma^* \} \]

A variant on the above proof, but instead of using \( \text{AH}_{M,w} \), (the set of accepting histories of \( M \) on \( w \)), we use its complement:

\[ \text{NH}_{M,w} = \{ x | x \text{ is not an accepting computation history of } M \text{ on } w \} \]

* and change the representation of a history so that alternate configs are reversed:

\[ \# C_1 \# C_2^{R} \# C_3 \# C_4^{R} \ldots \# C_n^{(R)} \# \]

Computable Functions

In addition to language recognition, we are also interested in computable functions.

Defn: a function \( f : \Sigma^* \rightarrow \Sigma^* \) is computable if \( \exists \) a TM \( M \) s.t. given any input \( w \in \Sigma^* \), \( M \) halts with just \( f(w) \) on its tape.

(Note: domain(\( f \)) = \( \Sigma^* \); crucial that \( M \) always halt, else value undefined.)

Ex 1: \( f(n) = n^2 \) is computable

Ex 2: \( g(<M,w>) = <L_{M,w}> \) (as in the EMPTY\(_{LBA} \) pf) is computable

Ex 2: \( h(<M,w>) = "1 \) if \( M \) acc \( w \) else 0" is uncomputable

(Why? Reduce \( A_{TM} \) to it.)
Reducibility

“A reducible to B” means could solve A if had subr for B
Can use B in arbitrary ways–call it repeatedly, use its
answers to form new calls, etc. E.g.,

\[ \text{WHACKY} \leq_T \text{ATM} \]

where \( \text{WHACKY} = \{ <M,w_1, w_2, ..., w_n> \mid M \text{ accepts } \}
\)
a_1 \cdots a_n \text{ where } a_i = 0 \text{ if } M \text{ rejects } w_i \text{, 1 if accepts } w_i \}

BUT in “practice,” reductions rarely exploit this generality
and a more refined version is better for some purposes

Reduction

Notation (not in book, but common):

\[ A \leq_T B \text{ means “A is Turing Reducible to B”} \]

i.e., if I had a TM deciding B, I could use it as a
subroutine to solve A

Facts:

\[ A \leq_T B \& B \text{ decidable implies } A \text{ decidable } \quad \text{(definition)} \]
\[ A \leq_T B \& A \text{ undecidable implies } B \text{ undecidable} \quad \text{(contrapositive)} \]
\[ A \leq_T B \& B \leq_T C \implies A \leq_T C \]

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

Facts:

\[ A \leq_m B \& B \text{ decidable } \implies A \text{ too} \]
\[ A \leq_m B \& A \text{ undecidable } \implies B \text{ too} \]
\[ A \leq_m B \& B \leq_m C \implies A \leq_m C \]

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:

\[ A \leq_m B \& B \text{ decidable } \text{(recognizable)} \implies A \text{ too} \]
\[ A \leq_m B \& A \text{ undecidable } \text{(unrecognizable)} \implies B \text{ too} \]
\[ A \leq_m B \& B \leq_m C \implies A \leq_m C \]

*Most reductions we’ve seen were actually \( \leq_m \) reductions.*
Mapping Reducibility

Defn: A is mapping reducible to B (A ≤ \( m \) B) if there is computable function \( f \) such that \( w \in A \Leftrightarrow f(w) \in B \)

A special case of \( \leq_T \):
Call subr only once; its answer is the answer

Facts:

\[ A \leq_m B \land B \text{ decidable} \Rightarrow A \text{ is too} \]
\[ A \leq_m B \land A \text{ undecidable} \Rightarrow B \text{ is too} \]
\[ A \leq_m B \land B \leq_m C \Rightarrow A \leq_m C \]

Most reductions we’ve seen were actually \( \leq_m \) reductions.
Mapping Reducibility

Defn: A is *mapping reducible* to B (A ≤ₘ B) if there is computable function f such that w ∈ A ⇔ f(w) ∈ B

**Theorem:**

1) A ≤ₘ B & B decidable (recognizable) ⇒ A is too
2) A ≤ₘ B & A undecidable (unrecognizable) ⇒ B is too
3) A ≤ₘ B & B ≤ₘ C ⇒ A ≤ₘ C

**Proof:**

1) To decide (recognize) w in A compute f(w), then use decider (recognizer, resp) for B on f(w).
2) Contrapositive
3) Given f for A → B, g for B → C; then w ∈ A ⇔ g(f(w)) ∈ C

Other Examples of ≤ₘ

- \( A_{TM} \leq_m \text{REGULAR}_{TM} \)
- f(<M,w>) = <M₂>
  - Build M₂ so L(M₂) = \( \Sigma^* \setminus \{0^n1^n\} \), as M accept/rejects w
- \( \text{EMPTY}_{TM} \leq_m \text{EQ}_{TM} \)
- f(<M>) = <M, M rejects>
  - L(M rejects) = \( \emptyset \), so equiv to M iff L(M) = \( \emptyset \)
- \( A_{TM} \leq_m \text{MPCP} \)
- \( \text{MPCP} \leq_m \text{PCP} \)

Lecture 12

\( A_{TM} \leq_T \text{HALT}_{TM} \)

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

\[ S:\]
- M,w

\[ R:\]
- Halt?
  - Yes
  - Sim M on w
    - Yes
      - Build M'
  - rej
    - acc
    - rej
    - acc

\[ S':\]
- M,w

\[ R:\]
- Halt?
  - Yes
  - Build M'
  - M' = M, but replace q_{reject} by a loop
  - rej
  - acc

From Lecture 07
**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

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

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

... 

**$A_{\text{TM}} (\leq_T \text{ vs } \leq_m) \text{ HALT}_{\text{TM}}$**

$A_{\text{Prog}} \leq_m \text{ HALT}_{\text{Prog}}$

$f(<P,w>) = <P',w>$

```
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
}
```

From Lecture 07

**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 ex. shows some things are perhaps 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

Hopefully 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 ≤_m B) if there is a computable function f such that w ∈ A ⇔ f(w) ∈ B

A special case of ≤_T:
Call subr only once; its answer is the answer

Theorem:
A ≤_m B & B decidable (recognizable) ⇒ A is too
A ≤_m B & A undecidable (unrecognizable) ⇒ B is too
A ≤_m B & B ≤_m C ⇒ A ≤_m C

Most reductions we’ve seen were actually ≤_m reductions.

Other Examples of ≤_m

A<sub>TM</sub> ≤_m REGULAR<sub>TM</sub>
f(<M,w>) = <M<sub>2</sub>>

Build M<sub>2</sub> so L(M<sub>2</sub>) = Σ* \ { 0<sup>1</sup> n | n ≥ 0 }, as M accept/rejects w

EMPTY<sub>TM</sub> ≤_m EQ<sub>TM</sub>
f(<M>) = <M, M<sub>reject</sub>>

L(M<sub>reject</sub>) = Ø, so equiv to M iff L(M) = Ø

A<sub>TM</sub> ≤_m MPCP
MPCP ≤_m PCP

Exercise: Is it A<sub>TM</sub> ≤_m REGULAR<sub>TM</sub>? If not, could it be changed?
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 $x \in L$ & $f(x) \in L$

$(x \in L \iff f(x) \in \overline{L}, \text{so } x \notin L \iff f(x) \notin \overline{L} \iff f(x) \in L)$

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 $x$. $L(M_0) = \emptyset$

$M_1$: 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 $= \emptyset$ & $h$ computable

Then $\overline{A_{TM}} \leq_m EQ_{TM}$ via $g(<M,w>) = <M_0, M_2>$

And $\overline{A_{TM}} \leq_m \overline{EQ_{TM}}$ via $f(<M,w>) = <M_1, M_2>$ (& $A_{TM} \leq_m EQ_{TM}$)

Lecture 13

EQ$_{TM}$ is neither recognizable nor co-recognizable

$M_0$: on any input $x$, reject $x$. $L(M_0) = \emptyset$

$M_1$: 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 $= \emptyset$ & $h$ computable

Then $\overline{A_{TM}} \leq_m EQ_{TM}$ via $g(<M,w>) = <M_0, M_2>$

And $\overline{A_{TM}} \leq_m \overline{EQ_{TM}}$ via $f(<M,w>) = <M_1, M_2>$ (& $A_{TM} \leq_m EQ_{TM}$)
Defining Inequivalence

“If two TMs are not equivalent, there is some input w where they differ, and if they differ there is some time t such that one accepts within t steps, but the other will not accept no matter how long you run it.”

\[ \text{EQ}_{TM} = \{ x \mid \exists y \forall z \langle x,y,z \rangle \in D \} \]

where the decidable set \( D = \{ \text{triples} \langle x,y,z \rangle \text{ such that } x \text{ is a pair of TMs, } y \text{ is a pair } w,t, \text{ and one machine accepts } w \text{ within } t \text{ steps but the other has not accepted } w \text{ within } z \text{ steps} \} \)

“The human mind seems limited in its ability to understand and visualize beyond four or five alternations of quantifier. Indeed, it can be argued that the inventions, subtheories, and central lemmas of various parts of mathematics are devices for assisting the mind in dealing with one or two additional alternations of quantifier.”


The “Arithmetical Hierarchy”

\[ \Pi_2: \{ x \mid \forall y \exists z \langle x,y,z \rangle \in D \} \]

\[ \Sigma_2: \{ x \mid \exists y \forall z \langle x,y,z \rangle \in D \} \]

\[ \Delta_1: \text{decidable given } \text{ATM} \]

\[ \Pi_1 (\text{co-recognizable}): \{ x \mid \forall y \langle x,y \rangle \in D \} \]

\[ \Delta_0: \text{decidable} \]

\[ \Sigma_1 (\text{Turing recognizable}): \{ x \mid \exists y \langle x,y \rangle \in D \} \]

Potential Utility: It is often easy to give such a quantifier-based characterization of a language; doing so suggests (but doesn't prove) whether it is decidable, recognizable, etc. and suggests candidates for reducing to it.

Decidability Questions

Questions about a single TM:

Detail questions: about *operation* or *structure* of a TM

useless state, does head move left, does it take >100 steps, ...

Bottom-line questions: ask about a TM's *language*

Is \( L(M) \) empty? Infinite? Is 42 in \( L(M) \)? ...

About \( L(M) \), not \( M \), per se. Same answer for \( M' \) if \( L(M)=L(M') \)

Other: Questions about \( \langle M,w \rangle \), 2 TMs, grammars, ...


Language Properties

We formalize language properties simply as sets of languages. E.g., the "infiniteness" property is just the set of infinite languages.

A property is non-trivial if there is at least one language with the property and one without. E.g., "emptiness" is nontrivial: \( L_1 = \emptyset \) has it; \( L_2 = \{42\} \) doesn't. E.g., "countable" is trivial: every subset of \( \Sigma^* \) is countable.

Rice’s Theorem

Theorem:

For every nontrivial property \( \mathcal{P} \) of the Turing recognizable languages, it is undecidable whether a TM recognizes a language having property \( \mathcal{P} \). I.e., \( \mathcal{P}_{TM} = \{ \langle M \rangle | L(M) \in \mathcal{P} \} \) is undecidable.

Corr:

\( \text{EMPTY}_{TM}, \text{INFINITE}_{TM}, \text{REGULAR}_{TM}, \ldots \) all undecidable.
Rice’s Theorem

\[ \mathcal{P}_{TM} = \{ <M> \mid M \text{ is a TM s.t. } L(M) \in \mathcal{P} \} \]

Proof: To show: \( A_{TM} \leq_T \mathcal{P}_{TM} \). WLOG, \( \emptyset \notin \mathcal{P} \); \( M_1 \) is a TM s.t. \( L(M_1) \in \mathcal{P} \)

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 in \( \mathcal{P} \) exactly when \( M \) does/does not accept \( w \), so knowing whether \( L(M') \in \mathcal{P} \) answers whether \( <M,w> \) is in \( A_{TM} \).

I.e., \( A_{TM} \leq_T \text{EMPTY}_{TM} \)

NB: it shows \( A_{TM} \leq_m \mathcal{P}_{TM} \) or \( \overline{\mathcal{P}_{TM}} \)

Programs, in general, are opaque, inscrutable, confusing, complex, obscure, and generally yucky...

(If you’ve been a 142 TA, you might have observed this yourself...)

Decidability Questions

Questions about a single TM:

- **Detail questions:** about operation or structure of a TM
  - useless state, does head move left, does it take >100 steps, ...

- **Bottom-line questions:** ask about a TM’s language
  - Is \( L(M) \) empty? Infinite? Is 42 in \( L(M) \)? ...
  
  About \( L(M) \), not \( M \), per se. *Same answer for \( M' \) if \( L(M) = L(M') \)*

Other: Questions about \( <M,w> \), 2 TMs, grammars, ...

Rice’s theorem doesn’t *(directly) answer these* But it says all these are undecidable (or trivial)