Reduction

Robert M. Keller
Harvey Mudd College
December 2013
Reading

- Sipser chapter 5
Why this stuff is trickier

• We usually use reduction to show that something is not possible ("prove a negative").

• It involves reasoning with hypothetical situations.
Wider Applicability

- The same kinds of construction are applied in algorithms.

- An example is the famous $P = NP$?

- See R.M. Karp, *Reducibility among combinatorial problems*
Review: Turing Machine Acceptance

- Acceptance as a property of a string $x$:
  
  \[ M \textbf{ accepts } x \iff \text{ when started on } x, \]
  \[ M \text{ eventually halts in an accepting state.} \]

- If $M$ does not accept $x$, there are two possibilities:
  - $M$ eventually halts in an \textbf{non-accepting} state.
  - $M$ \textbf{diverges} on $x$. 
Recap of Definitions

• L(M) is the *language recognized by* Turing machine M, meaning:
  \[ x \in L(M) \]
  iff
  when started on tape x,
  M will eventually halt in an *accepting* state.

• L(M) is additionally *decided by* M
  if M *always halts*. [Thus \( x \notin L(M) \) iff M halts in a non-accepting state when started on \( x \).]
Recognizability vs. Decidability

- A language $L$ is **recognizable** iff there is a Turing machine that **recognizes** $L$.

- A language $L$ is **decidable** iff there is a Turing machine that **decides** $L$.

- [We will avoid applying the term “accept” to languages in the case of Turing machines, and only apply it to strings.]
Possible Point of Confusion

Just because one TM M only recognizes L but does not decide L, does not mean there isn’t some other machine M’ that does decide L.

The same language might be decided by some machines, but only recognized by others.
Complementarity

- $L^c(M)$ is the language of strings over the tape alphabet of $M$ that are not accepted by $M$ [Sipsper uses an overbar instead of $^c$.]

- If $L$ is decided by $M$, then $L^c$ is accepted by some other machine.

- If $L$ is recognized by $M$, then $L^c$ might or might not be accepted by some other machine.

- If both $L$ and $L^c$ are recognizable, then $L$ is decidable.
Note

- \((L^c)^c = L\).
Recap of Basic Unrecognizability

- $K = \{<M> \mid (M \text{ is a Turing machine and}) \ <M> \notin L(M)\}$

  is not recognized by any Turing machine.

- Proof: Suppose $N$ recognizes $K$, i.e. $L(N) = K$.

  Then ask about $<N> \in K$:

  $<N> \in K$ iff

  $<N> \notin L(N)$ iff  (by definition of $K$)

  $<N> \notin K$  (by definition of $N$)
Recap of $A_{\text{TM}}$

- $A_{\text{TM}} = \{ <M, x> \mid (M \text{ is a Turing machine and) } x \in L(M) \}$
  
is not decided by any Turing machine.

- Proof using complementarity:
  
  $A_{\text{TM}}$ is the complement of $A_{\text{cTM}}$, which is shown on the next slide to be unrecognizable.

- **Note**: However, $A_{\text{TM}}$ is recognizable.
$A^c_{TM}$ (A for “accept”) generalizes K

- $A^c_{TM} = \{ <M, x> \mid (M \text{ is a Turing machine and}) \ x \notin L(M) \}$

is not recognized by any Turing machine.

- Proof: If were recognizable, then a related TM could recognize K as follows:
To determine whether $<M> \in K$, just note that

$<M> \in K$ iff
$<M> \notin L(M)$ iff
$<M, <M>> \in A^c_{TM}$.

But we previously showed that K is not recognizable.
Recap of $\text{Halt}_{TM}$

- $\text{Halt}_{TM} = \{<M, x> | M \text{ halts on input } x\}$

is not **decided** by any Turing machine.

- Proof by contradiction. Suppose we have an **algorithm H** that decides $\text{Halt}_{TM}$. We can use it to construct **algorithm A** that decides $A_{TM}$ as follows:

  Given input $<M, x>$ to A:
  - Call H on $<M, x>$.
    - If H says M halts on x, then run a universal Turing machine U on $<M, x>$ and the result.
    - If H says M does not halt on x, then return false.
  - As A cannot exist, neither can H. The supposition is false.
Example: $A_{TM} \leq Halt_{TM}$
(If were $Halt_{TM}$ decidable, $A_{TM}$ would be also.)
Example of Reduction

• The previous proof is an example of a reduction.

• We reduced the problem of deciding $A_{TM}$ to that of deciding $Halt_{TM}$.

• As $A_{TM}$ was earlier shown not to be decidable, $Halt_{TM}$ is not decidable either.
Reduction Example

- Having shown there is no algorithm for $A_{TM}$ we showed that:

  An algorithm for $\text{Halt}_{TM}$, if one existed, \textit{could} be used to construct an algorithm for $A_{TM}$

  thus

  $\text{Halt}_{TM}$ decidable $\rightarrow A_{TM}$ decidable

- Equivalently (\textit{logical contrapositive}):

  $A_{TM}$ \textit{undecidable} $\rightarrow \text{Halt}_{TM}$ \textit{undecidable}
Terminology

- We **reduced** $A_{TM}$ to $Halt_{TM}$

  in the sense that an **algorithm** for $Halt_{TM}$ could be used to construct an **algorithm** for $A_{TM}$.

- Note: It is **not correct** to say that this same argument reduces $A_{TM}$ to $Halt_{TM}$. That is backward usage.

- It is possible to reduce $A_{TM}$ to $Halt_{TM}$, but this would be a different argument.
Reduction used in the Positive Sense

A physicist and a mathematician are sitting in the cafe. Suddenly the coffee machine catches on fire. The physicist grabs a bucket, runs to the sink, fills the bucket with water and puts out the fire.

The next day, the same two are sitting in the cafe. Again, the coffee machine catches on fire. This time, the mathematician grabs a bucket and hands it to the physicist,

thus reducing the problem to one already solved.
Notation for Reduction

• If a problem (language) A can be reduced to a problem (language) B we write:

\[ A \leq B \] (A reduces to B)

suggesting that A is “no more difficult” than B.

If B is decidable, so is A.
If A is undecidable, so is B.

Also:
If B is recognizable, so is A.
If A is unrecognizable, so is B.
Proved Earlier

- $A_{TM} \leq Halt_{TM}$
- $K \leq A_{TM}^c$
Transitivity of Reduction

- If $A \leq B$ and $B \leq C$, then $A \leq C$.

- Example:

  $K \leq A_{TM}^c$ and $A_{TM}^c \leq Halt_{TM}^c$,

  therefore $K \leq Halt_{TM}^c$. 
Reduction Diagrams

- It is sometimes helpful to depict reduction arguments in diagrammatic form, to permit comparison of different reductions for example.

- The general form has:
  - An **outer box**, representing the problem **known** to be undecidable.
  - An **inner box**, representing the problem we are trying to show undecidable.
  - A input **transformation** step, representing the creation of a new input from an existing one.
  - An output **transformation** step, representing how the output of the inner box is interpreted.
Example: $K \leq A_{TM}^c$

$(K = \{<M> | <M> \not\in L(M)\}, \quad A_{TM}^c = \{<M, x> | x \not\in L(M)\})$

If were $A_{TM}^c$ computable, $K$ would be also.

\[<M> \xrightarrow{\text{xfm}} \langle M, <M>\rangle \xrightarrow{\text{effectively-computable transformation}} A_{TM}^c \xrightarrow{\text{accept/reject}} K\]
The transformation for $K \leq A_{TM}^{c}$

- All that this transformation needed to do was to make $<M, <M>>$ from $<M>$.

- Effectively it is making two copies of $<M>$ separated by a separator symbol.
Important Notes

- Reductions **analyze** the **source code** of the machine <M>, and use a transformation to **construct source code** of a new machine <M’>.

- **Analyzing** the machine, *might not entail running the machine*, although it could if need be.

- The $A_{TM} \leq Halt_{TM}$ example happens to run the code on a Universal TM.
Halt_{TM} \leq A_{TM}

- This is the “other direction”.
- If A_{TM} were decidable, then so Halt_{TM} would be.
- This problem is simpler to show.
- The reduction diagram is on the next page.
- But what goes in the box marked xfm?
Example: $\text{Halt}_{TM} \leq A_{TM}$
(If $A_{TM}$ were computable, $\text{Halt}_{TM}$ would be also.)
What transformation xfm?

• We want
  M halts on x
  iff
  M’ accepts x.

• So xfm has to transform M to M’ so that halting state of M is made into an accepting state of M’.
Example of Source $xfm$

Any state from which there is no transition is rejecting if it is not accepting.

<table>
<thead>
<tr>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reject</td>
</tr>
<tr>
<td>00</td>
<td>Accept</td>
</tr>
<tr>
<td>000</td>
<td>Reject</td>
</tr>
<tr>
<td>0000</td>
<td>Accept</td>
</tr>
<tr>
<td>00000</td>
<td>Reject</td>
</tr>
<tr>
<td>000000</td>
<td>Accept</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Accept</td>
</tr>
<tr>
<td>00</td>
<td>Accept</td>
</tr>
<tr>
<td>000</td>
<td>Accept</td>
</tr>
<tr>
<td>0000</td>
<td>Accept</td>
</tr>
<tr>
<td>00000</td>
<td>Accept</td>
</tr>
<tr>
<td>000000</td>
<td>Accept</td>
</tr>
</tbody>
</table>
Example of Source xfm for non-terminating inputs

Cancelled suggests divergence
Mutual Reductions

• We have proved both of these reductions:
  • $A_{\text{TM}} \leq \text{Halt}_{\text{TM}}$
  • $\text{Halt}_{\text{TM}} \leq A_{\text{TM}}$

• Thus we can say that $A_{\text{TM}}$ and $\text{Halt}_{\text{TM}}$ are of equivalent difficulty.

• This implies that if either is undecidable, so is the other.

• In the present case, we know that both are undecidable. (In some cases, we might not know.)
Example: Empty-Tape Decision Problem

- Is there an algorithm that will determine whether an arbitrary TM will accept an empty (i.e. all-blank) tape $\varepsilon$?

- This seems “easier” than $A_{TM}$ (because there is one less variable: the tape) but it’s actually just as hard.

- Call this problem $A_{\varepsilon_{TM}}$. 
$A_{TM} \leq A_{\varepsilon_{TM}}$

- Suppose we had an algorithm that would decide $A_{\varepsilon_{TM}}$. We can use it to decide $A_{TM}$, as follows.

- Let $<M, x>$ be an instance of the $A_{TM}$ problem.

- The transformation constructs $<M'>$, an instance of the $A_{\varepsilon_{TM}}$ problem.
\[ A_{TM} \leq A_{\varepsilon_{TM}} \] transformation

- From \(<M, x>\) construct \(M'\): \(M'\) first writes \(x\) on its tape, then behaves just as \(M\) would have on the result. (\(M'\) has the writing of \(x\) “wired into” its program.)

- Therefore \(M'\) on an empty tape will give the same result as \(M\) would on \(x\).

- So determining whether \(M'\) accepts \(\varepsilon\) is the same as determining whether \(M\) accepts \(x\).
\(A_{\text{TM}} \leq A_{\varepsilon \text{TM}}\) Diagram

M’ first writes x on its tape, then behaves just as M would have on the result.
Also $A_{\varepsilon TM} \leq A_{TM}$

- This is true because $\varepsilon$ is a special case of a general input.

- Neither problem is decidable.
A_{\varepsilon TM} \leq A_{TM} \text{ Diagram}
Example: Fixed-Tape Halting Problem

- Is there an algorithm that will decide whether an arbitrary TM will accept a fixed tape \( z \)? (such as \( z = 1101101 \))

- We showed this for the special case \( z = \varepsilon \).

- Call this problem \( A_{z_{TM}} \).
$A_{\text{TM}}^z$ is undecidable for any fixed $z$.

- Let $\langle M, x \rangle$ be an instance of $A_{\text{TM}}$.
- We show how to convert $\langle M, x \rangle$ to an equivalent instance $\langle M' \rangle$ of $A_{\text{TM}}$.

  - $M'$ with arbitrary input $y$:
    - Compare $y$ to $z$. If $y \neq z$, accept.
    - If $y = z$, then behave just as $M$ would have on $x$.
  
- So $M'$ accepts $z$ iff $M$ accepts $x$. 

$A_{TM} \leq A_{z_{TM}}$ Diagram

$A_{TM}$

$M'$ with arbitrary input $y$:
- Compare $y$ to $z$. If $y \neq z$, accept.
- If $y = z$, then behave just as $M$ would have on $x$.

$<M, x> \xrightarrow{xfm} <M'> \xrightarrow{A_{z_{TM}}} \begin{cases} yes & \text{if} \; A_{z_{TM}} \text{ accepts } z \\ no & \text{otherwise} \end{cases}$

$M'$ accepts $z$ iff $M$ accepts $x$. 
Emptiness Problems

- $E_{DFA} = \{ <B> \mid B \text{ is a DFA accepting no strings} \}$
- $E_{REX} = \{ <R> \mid R \text{ is a regular expression and } L(R) = \emptyset \}$
- $E_{CFG} = \{ <G> \mid G \text{ is a context-free grammar and } L(G) = \emptyset \}$
- $E_{CSG} = \{ <G> \mid G \text{ is a context-sensitive grammar and } L(G) = \emptyset \}$
- $E_{TM} = \{ <M> \mid M \text{ is a Turing machine and } L(M) = \emptyset \}$

Which of these are decidable? (Not necessarily obvious).
$E_{TM}$ is not decidable

- For problems that follow the usual pattern, we can just give the transformation.
- To reduce $A_{TM}$ to $E_{TM}$, the transformation is:
  Given $<M, x>$ as an instance of $A_{TM}$, construct instance $M'$ of $E_{TM}$.

  - $M'$ on input $w$ behaves as follows:
    - If $w \neq x$, then reject.
    - If $w = x$, then behave as $M$ on $x$.

  - Thus $L(M') \neq \emptyset$ iff $M$ accepts $x$. 
\( A_{TM} \leq E_{TM} \) Diagram

\( A_{TM} \)

\( M' \) on input \( w \) behaves as follows:
- If \( w \neq x \), then reject.
- If \( w = x \), then behave as \( M \) on \( x \).

\( \langle M, x \rangle \)

\( xfm \)

\( \langle M' \rangle \)

\( E_{TM} \)

\( L(M') \neq \emptyset \)

\( L(M') = \emptyset \)

\( M' \) accepts something iff \( M \) accepts \( x \).
Alternate Transformation for $A_{TM} \leq A_{z_{TM}}$, etc.

- This handles a number of problems with a single construction.
- Given $<M, x>$, an instance of $A_{TM}$, construct $M'$:
  - First **erases** whatever is on its tape.
  - Then writes $x$ on its tape.
  - Then behaves as $M$ would.
- So $M'$ accepts **all** tapes iff $M$ accepts $x$, otherwise $M'$ accepts no tape.
Advantage of the alternate xfm

- $M'$ shows that $A_{TM} \leq A_{Z_{TM}}$.
- But it can also be used in other reductions:
  - Emptiness: $A_{TM} \leq E_{TM}$  Does $M'$ accept anything?
  - Fullness: $A_{TM} \leq ALL_{TM}$  Does $M'$ accept everything?
  - Infinite: $A_{TM} \leq Infinite_{TM}$  Does $M'$ accept an infinite set?
- Where can't it be used?
  - Regularity: $A_{TM} \leq Regularity_{TM}$  Is $L(M')$ a regular language?
  - Why not? Because both $\emptyset$ and $\Sigma^*$ are regular.
Regular<sub>TM</sub> is not decidable (Sipser proof)

- Regular<sub>TM</sub> = \{<M> | L(M) is regular\}

- Given <M, x> as an instance of A<sub>TM</sub>, construct instance M’ of Regular<sub>TM</sub>.

- M’ on input w behaves as follows:
  - If w is of the form 0<sup>n</sup>1<sup>n</sup> for some n, then accept.
  - Otherwise, behave as M on x, and if M(x) accepts, accept w.

- Thus if M accepts x then L(M’) = Σ* (regular), but if M does not accept x, L(M’) = {0<sup>n</sup>1<sup>n</sup> | n ≥ 0} (not regular).

- So x ∈ L(M) iff M’ ∈ Regular<sub>TM</sub>. 
Different kinds of reduction

- Language $A \subseteq \Sigma^*$ is **Turing reducible** to $B \subseteq \Delta^*$, notated $A \leq_T B$ provided:

  An algorithm for deciding $A$ can be implemented by calling an algorithm for deciding $B$ (querying $B$ as if it were an “oracle”).

  Any number of such calls can be used in general.

Mapping Reducibility $\leq_m$

- Language $A \subseteq \Sigma^*$ is **mapping reducible** to $B \subseteq \Delta^*$, notated $A \leq_m B$ provided:

  there is some *computable* function $f: \Sigma^* \rightarrow \Delta^*$ such that

  $$\forall x \in \Sigma^* \quad x \in A \iff f(x) \in B.$$  

- In this case, function $f$ is called a **reduction** of $A$ to $B$.

- Mapping reductions are also called “many-to-one” reductions in the literature.

- They are a special case of Turing reductions, in which the oracle is only called once, to give the final answer (sort of like a **tail-recursive** version of $\leq_T$).
Reduction Diagram for Mapping Reductions

Mapping reductions are a simple specialized case of what we’ve seen before.

\[ x \in \Sigma^* \] 
\[ f \] 
\[ f(x) \in \Delta^* \] 
\[ f(x) \in B? \] 
\[ 'yes' \] 
\[ 'yes' \] 
\[ 'yes' \] 
\[ 'yes' \] 
\[ no \] 
\[ no \] 
\[ no \] 
\[ no \] 

\[ f \] must be computable, as always.
Example: \( \text{HALT}_{\text{TM}} \leq_m \text{A}_{\text{TM}} \)?

- What function maps \( \text{HALT}_{\text{TM}} \) into \( \text{A}_{\text{TM}} \).

- The mapping reduction \( f \) does this transformation:

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

where \( M' \) accepts \( x \) iff \( M \) halts on \( x \).

The transformation from \( M \) to \( M' \) changes all halting states into accepting states.
A \leq_m B and decidability

• As with any reduction, $A \leq_m B$ implies if $B$ were decidable, so would $A$ be:
  To determine whether $x \in A$: compute $f(x)$ by machine, determine whether $f(x) \in B$, to get your answer.

• It also means the **contrapositive**, that if $A$ is **undecidable**, so is $B$. 
A \leq_m B and recognizability

- A \leq_m B also implies if B were **recognizable**, so would A be:
  To determine whether \( x \in A \): compute \( f(x) \) by machine, determine whether \( f(x) \in B \), to get your answer. If the computation of \( f(x) \in B \) diverges, so does the computation of \( x \in A \).

- It also means the **contrapositive**, that if A is **unrecognizable**, so is B.
Example: Non-Recognizability of $A_{\text{TM}}^c$

- $K \leq_m A_{\text{TM}}^c$ (the non-acceptance language for TM)

- $K = \{<M> | <M> \notin L(M)\}$ is not recognizable

- $A_{\text{TM}}^c = \{<M, w> | w \notin L(M)\}$ is therefore not recognizable

- The reduction mapping in this case is:
  
  $f(<M>) = <M, <M>>$
A \leq_m B and complements

• A \leq_m B \iff A^c \leq_m B^c.

• Proof: Suppose A \leq_m B. Let f be the reduction function such that
  \[ x \in A \iff f(x) \in B \]

• Then also
  \[ x \notin A \iff f(x) \notin B \]

• That is:
  \[ x \in A^c \iff f(x) \in B^c \]
A \leq_m B and co-recognizability

- A \leq_m B similarly implies that if B were co-recognizable, so would A be.

- A \leq_m B implies if A is not co-recognizable, neither is B.
$\leq_m$ Mapping need not be 1-1
The sense of the answers matters with $\leq_m$

- With mapping reduction, a **yes** answer returned by the inner box must become the **yes** answer of the outer, not a **no** answer.

- In other words, $A \leq_m B$ is **not** interchangeable with $A \leq_m B^c$. 
Not a Mapping Reduction

\[ x \in \Sigma^* \quad \xrightarrow{f} \quad f(x) \in \Delta^* \quad \xrightarrow{f(x) \in B?} \]

- yes
- no

\[ x \in A \]

- yes
- no
$A_{TM} \leq E_{TM}$ Not a Mapping Reduction

M' on input $w$ behaves as follows:
- If $w \neq x$, then reject.
- If $w = x$, then behave as $M$ on $x$.

$M'$ accepts something iff $M$ accepts $x$. 

$L(M') \neq \emptyset$

$L(M') = \emptyset$
$A_{TM} \leq E_{TM}^c$ is a Mapping Reduction because complementing changes the sense.
So $A_{TM}^c \leq E_{TM}$, thus $E_{TM}$ is not recognizable.

$A_{TM}$

M’ on input w behaves as follows:
- If $w \neq x$, then reject.
- If $w = x$, then behave as M on x.

$<M, x> \xrightarrow{xfm} <M'> \xrightarrow{E_{TM}^c} \begin{cases} yes & L(M') \neq \emptyset \\ no & L(M') = \emptyset \end{cases}$

M’ accepts something iff M accepts x.
Another example where it matters

- It is not the case that $K \leq A_{TM}$, as $A_{TM}$ is recognizable whereas $K$ is not.

- However $K \leq A_{cTM}$, as can be seen by the function
  $$f(<M>) = <M, <M>>$$

  $$<M> \in K \iff <M> \notin L(M) \iff <M, <M>> \notin A_{TM} \iff <M, <M>> \in A_{cTM}$$
Is $E^c_{TM}$ recognizable?

• We just showed that $E_{TM}$, the emptiness problem for TM’s is not recognizable.

• But maybe the **non-emptiness** problem $E^c_{TM}$ is recognizable.

• What do you think?
Mixture: $A^c \leq_m B$

- $A^c \leq_m B$ means the *complement* of $A$ reduces to $B$.
- This is **equivalent** to $A \leq_m B^c$. 
Sipser Exercise 5.7

- **Show**: If $A$ is recognizable, and $A \leq_m A^c$, then $A$ is decidable.

- **Proof**: Suppose $A$ is recognizable and $A \leq_m A^c$.

  - By the previous slide, also $A^c \leq_m A$.

  - Thus $A^c$ is recognizable, from mapping reduction.

  - As both $A$ and are $A^c$ recognizable, $A$ is decidable by the complementarity lemma.
“ALL” Problems

- \( \text{ALL}_{\text{DFA}} = \{<B> \mid B \text{ is a DFA accepting all strings}\} \)
  
  (“all” means all of \( \Sigma^* \), where \( \Sigma \) is the alphabet of the DFA)

- \( \text{ALL}_{\text{REX}} = \{<R> \mid R \text{ is a regular expression and } L(R) = \Sigma^*}\} \)

- \( \text{ALL}_{\text{CFG}} = \{<G> \mid G \text{ is a context-free grammar and } L(G) = \Sigma^*}\} \)

- \( \text{ALL}_{\text{CSG}} = \{<G> \mid G \text{ is a context-sensitive grammar and } L(G) = \Sigma^*\} \)

- \( \text{ALL}_{\text{TM}} = \{<M> \mid M \text{ is a Turing machine and } L(M) = \Sigma^*}\} \)

- Which of these are decidable? (Not necessarily obvious).
\( \text{ALL}_{\text{TM}} \)

- Is \( \text{ALL}_{\text{TM}} \) decidable?
- What is your intuition?
- How would you prove it?
- What about recognizable or co-recognizable, if not decidable?
Languages neither recognizable nor co-recognizable.

- $\text{EQ}_{\text{TM}} = \{<M, N> | L(M) = L(N)\}$ is neither.

- To explore this kind of problem, try letting $N$ be a **fixed machine** and see if another unrecognizable language can be reduced to the problem with $<M>$ as a variable.

- Candidates for $N$:
  - $\Phi$, a machine that accepts nothing.
  - $\Omega$, a machine that accepts everything.
$E_{\text{EQ}_{\text{TM}}} = \{<M, N> \mid L(M) = L(N)\}$ is not recognizable

• Strategy: Try to reduce $A_{\text{c}_{\text{TM}}}$ to $E_{\text{EQ}_{\text{TM}}}$.

• Consider the mapping $f(<M, x>) = <M', \Phi>$, where $M'$ accepts every input iff $M$ accepts $x$.
  ($\Phi$ = a machine that accepts nothing)

• Then $<M, x> \in A_{\text{TM}}$ iff $<M', \Phi> \notin E_{\text{EQ}_{\text{TM}}}$
  (i.e. $M'$ accepts something, but $\Phi$ accepts nothing).

• Thus $A_{\text{TM}} \leq E_{\text{EQ}_{\text{c}_{\text{TM}}}}$, and therefore $A_{\text{c}_{\text{TM}}} \leq E_{\text{EQ}_{\text{TM}}}$. 
EQ_{TM} = \{<M, N> \mid L(M) = L(N)\} is not co-recognizable

- Strategy: Reduce A_{TM}, which is not co-recognizable, to EQ_{TM}.

- Consider the mapping f(<M, x>) = <M', \Omega>, where M' accepts all inputs iff M accepts x. (\Omega = a machine that accepts everything)

- Then <M, x> ∈ A_{TM} iff <M', \Omega> ∈ EQ_{TM}

- Therefore A_{TM} ≤ EQ_{TM}. 
Other languages neither recognizable nor co-recognizable.

- $\text{ALL}_{\text{TM}}$ is neither.

- To show $\text{ALL}_{\text{TM}}$ is not co-recognizable, give a mapping reduction from $\text{A}_{\text{TM}}$ (which is not co-recognizable), to $\text{ALL}_{\text{TM}}$.

  \[
  f(<M, w>) = M', \text{ where } M'(x) = M(w).
  \]

  Thus $M'$ accepts all iff $M$ accepts $w$.

- But how to show $\text{ALL}_{\text{TM}}$ is not recognizable??
ALL\textsubscript{TM} is not recognizable (New Technique)

- We prove this by \( \text{HALT}^{c}_{\text{TM}} \leq_{m} \text{ALL}_{\text{TM}} \).

- Let \(<M, x>\) be an arbitrary machine with input. The reduction mapping constructs \( M' \) to behave as follows on input \( w \):
  - \( M'(w) \) simulates \( M(x) \) for \(|w|\) steps.
  - If \( M(x) \) fails to halt within \(|w|\) steps, then \( M' \) accepts \( w \).
  - Otherwise \( M' \) rejects \( w \).

- Hence \( M' \) accepts all inputs iff \( M(x) \) does not halt.
Rice’s Theorem

- One pattern in proving languages undecidable has been generalized to a meta-theorem by H.G. Rice, 1953.

- We call it meta- because the properties it shows undecidable are very general.
Rice’s Theorem, Informally

- Practically **no** functional properties of recognizable languages are decidable.
- The caveat here is that the theorem only applies to **functional** properties, not structural ones.
- The concept of structural vs. functional is used in **software testing** (white-box vs. black-box testing).
Property = Set Membership

A TM having a property is just another way of saying its language is in the set of languages having that property.
Functional vs. Structural Properties

- We have been using TM’s to represent languages.

- When a property of a machine depends only on the language and not the specific TM used to represent the language, we call this a functional property.

- When a property depends on the specific details of a machine, it is called a structural property.
Functional vs. Structural Examples

- **Functional properties of the language of a machine M:**
  - \( L(M) = \emptyset \)
  - \( L(M) \) is regular
  - \( L(M) \) is infinite
  - \( L(M) \) is decidable
  - \( L(M) \) is recognizable (always true, by definition, since M is a TM)

- **Structural, not functional, properties of a machine M:**
  - M has more than 100 reachable control states.
  - M writes a non-blank character on its tape when started on an empty tape.
  - M reverses its direction of head travel at least once on every input.
**Trivial** Functional Properties

- A functional property of a recognizable language is called **trivial** if it is either:
  - true for *all* recognizable languages, or
  - true for *no* recognizable language.

- A **non-trivial property**, then, holds for some recognizable languages, but not all.
Rice’s Theorem

- Any non-trivial functional property of the recognizable languages is not decidable.

- Put another way:

  For any non-trivial functional property $P$, there is no algorithm that will determine whether or not the language recognized by a given TM has property $P$. 
Proof of Rice’s Theorem (1 of 4)

• Suppose $P$ is a non-trivial functional property that is decidable.

• **Critical assumption:** The empty language $\emptyset$ must not have property $P$.

  If the opposite is true, then interchange $P$ and $\neg P$ so that the assumption is then true.

  [This would be necessary in the case of $P = \text{Regular}$, for example, as $\emptyset$ is regular.]

• Let $A_P$ be a $P$-decider. We are going to draw a contradiction by reduction: $A_{TM} \leq_m A_P$. 
Proof of Rice’s Theorem (2 of 4)

• Note that $\emptyset$ is the language of a machine $\Phi$ that always diverges (and does not have property $P$).

• Let $L_p$ be some arbitrary recognizable language having property $P$ (which must exist, because $P$ is non-trivial).

• We know that $L_p \neq \emptyset$, by the assumption on the previous slide.

• Let $M_p$ be a machine recognizing the chosen language $L_p$. 
Strategy (3 of 4)

- $M_P \left( L_P \right)$ has property $P$.
- $\Phi \left( \emptyset \right)$ does not have property $P$.
- Construct and pass to the hypothetical $P$ decider a machine having language either $L_P$ or $\emptyset$, depending on whether $x \in L(M)$. 
Proof of Rice’s Theorem (4 of 4)

• Transform \(<M, x>\) to \(M’\).

• \(M’\): with input \(w\), temporarily set aside \(w\) and start **behaving as** \(M\) on \(x\).

• **If \(M\) on \(x\) accepts**, then continue by behaving as \(M_p\) on the original input \(w\). So in this case, \(L(M’)\) **has property** \(P\).

• **If \(M\) on \(x\) rejects or diverges**, purposely diverge. So in this case, \(L(M’)\) **does not have property** \(P\).

• Thus if there were an algorithm that tested for property \(P\) of an arbitrary machine, including \(M’\), there is also one that can test whether \(M\) accepts \(x\). But we know this is impossible.
Flowchart of $M'$ as constructed
(dashed lines = control, solid = data)
Note: This shows the transformation; this is **not** a reduction diagram.

M'

- **start here**
- run $M$ on $x$
- $M$ rejects $x$ or diverges: loop: diverge
- $M$ accepts $x$: Behave as $M_p$

$W$ → $M_p$ → Accept/Reject $w$ → Accept/Reject $w$
Note

- Although P was discussed as a property of language recognized by a Turing machine, essentially the same proof works in the case that:

  P is a property of the partial function computed by a Turing machine.

- This means, for example, there is no algorithm that will decide equivalence of an arbitrary machine’s partial function to that of a given machine.
Decidable vs. Undecidable Structural Properties

- M has more than 100 control states.
- M eventually reaches a specific control state when started on an empty tape.
- M writes a non-blank character on its tape when started on an empty tape.
- M uses more than a specified amount of tape when started on a certain input.

- Some of these properties might be decidable by determining **bounds on the number of steps** required to reach a certain configuration (cf. LBA discussion later).

- Some structural properties might shown undecidable by showing that they can be **transformed to functional properties**.
Post’s Correspondence Problem (PCP)

• This is another unsolvable problem that might be unexpected due to its simplicity.

• $\text{HALT}_{TM} \leq_m \text{PCP}$

• Please read about it in Sipser 5.2 and/or Huth&Ryan.

• Huth & Ryan give a proof of Church’s theorem using it.
Using Computation Histories

- A computation history (CH) is a recording of the history of a Turing-machine computation.
Of what use is a CH?

• Using CH’s we can demonstrate that various problems are undecidable.

• These problems may be difficult to show using other methods.
State of a Turing Machine

- The total **state** (often called a “configuration” to distinguish it from the control state) represents everything that needs to be known about the machine to continue the computation.
Capturing States as Strings

- A state can be represented as a string $xqy \in \Gamma^*$ where
  - $\Gamma$ is the tape alphabet
  - $x$ is the tape to the left of the head
  - $q$ is the control state
  - $y$ is the tape under and to the right of the head
Computation History

• As the TM computes, the states make transitions:
  \[ x_1q_1y_1 \Rightarrow x_2q_2y_2 \Rightarrow x_3q_3y_3 \Rightarrow \ldots \]

• A deterministic machine will eventually either:
  • **Halt**: Reach a state \( xq_hy \) where no further transition is defined, or

  • **Diverge**: Never reach such a state, i.e. it goes on forever.
Encoding An Entire History

- **In the halting case**, the entire history is encodable as a **single string**:

  \[x_1q_1y_1 \Rightarrow x_2q_2y_2 \Rightarrow x_3q_3y_3 \Rightarrow \ldots \Rightarrow x_hq_hy_h\]

  where here we consider \(\Rightarrow\) to be a symbol in a larger alphabet.

- Usually \# is used in place of \(\Rightarrow\), maybe because its easier to type.
Checking a History with a TM

- There is a Turing machine C (checker) that, with input \(<M, h>\), where \(<M>\) is an encoding of an arbitrary TM, can check whether \(h\) is a history of \(M\).

- All C needs to do is see if the symbols around the head of each state change correspond to the transition table of \(M\).

- Further more, C can check whether \(h\) is a halting (or an accepting) history by looking at the last control state.
Checking a History with a TM

- \( x_1 a_1 q_1 b_1 y_1 \Rightarrow x_2 a_2 q_2 b_2 y_2 \Rightarrow x_3 a_3 q_3 b_3 y_3 \Rightarrow \ldots \)

where each \( a_i, b_i \in \Gamma \)

Do the local transitions \( a_i q_i b_i \rightarrow a_{i+1} q_{i+1} b_{i+1} \) correspond to the rules in the description?

Do the other parts of the string match?

C can check all these things.
LBAs

- In fact, C can be a restricted TM called an **LBA**.
- LBA = “Linear Bounded Automaton”, a TM that can never exceed the amount of tape on which the input is written (storage is thus a constant times the length of the input).
- An LBA can have a tape alphabet that is bigger than the input alphabet, so it can effectively “mark” tape cells, etc. It just can’t **grow** its tape.
LBA Languages

Languages accepted by LBAs coincide with the languages generated by context-sensitive grammars (type 1 grammars).
LBA vs. DFA

- At first glance, it might appear that an LBA is no more powerful than a DFA, since it cannot add new states.

- The difference is, however, that an LBA’s total state set is a (linear) function of the input string size, which is not true for a DFA.

- Also, an LBA can move in both directions.
Aside: 2-way DFAs

- If the LBA \textbf{never writes} on its tape, then it becomes a 2-way DFA.

- In this case, the power is reduced to that of a DFA: 2-way DFA’s only accept regular languages.

- The proof of this is non-trivial, and surprising. It involves the Myhill-Nerode theorem.

- A 2-way DFA may be more efficient in its number of states, however.
$A_{LBA}$ is the Acceptance Language for LBAs

Define $A_{LBA} =$

$\{<M, w> \mid M \text{ is an LBA } \land M \text{ accepts } w\}$
Theorem: \( A_{LBA} \) is decidable (by a TM).

- Note that this differentiates \( A_{LBA} \) from \( A_{TM} \).

- Proof: Given an LBA encoding with tape \(<M, w>\), we can simulate \( M \) on \( w \).

- The maximum number of distinct tape states is \( n^{|w|} \), where
  \( n \) is tape alphabet size and \(|w|\) means the length of \( w \).

- The number of different head positions is \( 1+|w| \).

- The number of control states is \( m \), say.

- So the total number of different states of \( M \) for an input \( w \)
  is \( mn^{|w|}(1 + |w|) \).
Proof that $A_{LBA}$ is decidable (cont’d)

- For each step in the simulation of $M$ on $w$, we increment the count, having started at 0.

- If $M$ on $w$ is still computing after $mn|w|(1 + |w|)$ steps, we know that it is in a loop. So the simulating machine can reject $<M, w>$ if this happens.

- The simulating machine will otherwise accept or reject $<M, w>$, depending on what happens with $M$ on $w$.

- Hence there is a TM that can decide $A_{LBA}$. 
Theorem: $E_{LBA}$ is undecidable.

- Define the Emptiness Language $E_{LBA} = \{ <M> | M$ is an LBA $\land L(M) = \emptyset \}$

Proof: Show $A_{TM} \leq_m E_{LBA}^c$.

Define $f(<M, w>) = M'$, where $M'$ is an LBA that accepts the accepting computation histories for $M(w)$. Then $L(M') \neq \emptyset$ iff $<M, w> \in A_{TM}$. 
ALL\textsubscript{CFG} is undecidable

- Define ALL\textsubscript{CFG} =

\{<G>| G is a CFG \land L(G) = \Sigma^*\}

We know that for every CFG, there is a corresponding PDA that accepts the same language.

We show that PDA’s can also, in a sense, check computation histories.

The halting computation of a TM will be identified with the absence of a string accepted by the PDA.
Proof of ALL_{CFG} is undecidable

- For any TM and input <M, w> we construct a PDA that accepts all strings that are not valid computation histories.

- Thus, if the PDA accepts all strings, then M does not halt on w.
Proof of $\text{ALL}_{\text{CFG}}$ is undecidable

- We represent the computation histories slightly differently in this case: Every other state is reversed.

- Originally: $x_1a_1q_1b_1y_1 \Rightarrow x_2a_2q_2b_2y_2 \Rightarrow x_3a_3q_3b_3y_3 \Rightarrow \ldots$

- Now $\overline{x_1a_1q_1b_1y_1} \# \overline{x_2a_2q_2b_2y_2} \# \overline{x_3a_3q_3b_3y_3} \# \ldots$

- where overbar represents the reversal of the string below.
Why reverse?

- Reversing every other state enables two successive states in a history to be checked by a PDA.
- The PDA, given an input, checks one of these conditions (non-deterministically):
  - Does the input not begin with the initial state of <M, w>? If not, accept.
  - Does the input not end with an accepting state of <M, w>? If not, accept.
  - Starting at any one of the sections $C_i \# C_{i+1}$ or $\overline{C_i} \# C_{i+1}$, do the two sections not represent consecutive states. If not, accept.
Why reverse?

- So a string is **accepted** iff it does **not** represent a valid computation history.

- Thus the PDA **accepts** $\Sigma^*$ iff $M(w)$ does **not** have an accepting computation history.
Other Connections Between Computability & Logic

- Decidability of a certain logic theory \((N, +)\) is reducible to DFA equivalence.

- Other theories \((N, +, X)\) can be shown undecidable using Turing machines or their equivalent.

- Satisfiability of propositional logic plays a key role in polynomial-time reducibility (P vs. NP).