Computable functions

Reductions

Reducing $A$ to $B$ by Mapping reductions

More undecidable languages

Undecidability by Rice Theorem

Reductions by computational histories (maybe)

Sipser’s book, Chapter 5, Sections 5.1, 5.3
More Undecidable Problems

We have already

- Established Turing Machines as the gold standard of computers and computability . . .
- seen examples of solvable problems . . .
- and saw one problem, $A_{TM}$, that is computationally unsolvable.

Today, we look at other computationally unsolvable problems. We introduce the techniques of mapping reductions, of Rice Theorem, and of reductions by computational histories (time permitting) for proving that languages are undecidable/non-enumerable.
Computable Functions

A TM computes a total function

\[ f : \Sigma^* \rightarrow \Sigma^* \]

if the TM

- when starting with an input \( w \),
- it halts with only \( f(w) \) written on tape.

The definition can be extended to functions of more than one variable, where some special separator symbol indicates end of one variable and beginning of next.
Computable Functions

A TM computes a **partial** function

\[ f : \Sigma^* \rightarrow (\Sigma^* \cup \bot) \]

if the TM

- when starting with an input \( w \),
- if \( f(w) \) is defined, TM halts with only \( f(w) \) on tape,
- if \( f(w) \) is undefined, TM **does not halt**.

Computable functions are also called (**total** or **partial**) recursive functions.
Claim: All the “usual” arithmetic functions on integers are computable.

These include addition, subtraction, multiplication, division (quotient and remainder), exponentiation, roots (to a specified precision), modular exponentiation, greatest common divisor.

Even non-arithmetic functions, like logarithms and trigonometric functions, can be computed (to a specified precision), using Taylor expansion or other numeric mathematic techniques.

Exercise: Design a TM that on input \( \langle m, n \rangle \), halts with \( \langle m + n \rangle \) on tape.
Computable Functions

A useful class of functions modifies TM descriptions. For example:

On input $w$:

- if $w = \langle M \rangle$ for some TM,
  - construct $\langle M' \rangle$, where
    - $L(M') = L(M)$, but
    - $M'$ never tries to move off LHS of tape.
- otherwise write $\varepsilon$ and halt.
Reducibility

Example:

- Finding your way around a new city
- reduces to . . .
- obtaining a city map.
Reducibility, In Our Context

Always involves two problems, $A$ and $B$.

Desired Property: If $A$ reduces to $B$, then any solution of $B$ can be used to find a solution of $A$.

Remark: This property says nothing about solving $A$ by itself or $B$ by itself.
Examples

Reductions:
- Traveling from Boshton to Paris . . .
- reduces to buying plane ticket . . .
- which reduces to earning the money for that ticket . . .
- which reduces to finding a job
  (or getting the $s from mom and dad. . .)
Examples

Reductions:
- Measuring area of rectangle . . .
- reduces to measuring lengths of sides.

Also:
- Solving a system of linear equations . . .
- reduces to inverting a matrix.
If $A$ is reducible to $B$, then

- $A$ cannot be harder than $B$
- if $B$ is decidable, so is $A$.
- if $A$ is undecidable and reducible to $B$, then $B$ is undecidable.
Additional Undecidable Problems

We have already established that $A_{TM}$ is undecidable.

Here is a related problem – the original halting problem (of Shoshana and Uri :-).

$$H_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \}$$

Clarification: How does $H_{TM}$ differ from $A_{TM}$?
Undecidable Problems

\[ \mathcal{H}_{\text{TM}} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \} \]

**Theorem:** \( \mathcal{H}_{\text{TM}} \) is undecidable.

**Proof idea:**
- By contradiction.
- Assume \( \mathcal{H}_{\text{TM}} \) is decidable.
- Let \( R \) be a TM that decides \( \mathcal{H}_{\text{TM}} \).
- Use \( R \) to construct \( S \), a TM that decides \( \mathcal{A}_{\text{TM}} \).
- So \( \mathcal{A}_{\text{TM}} \) is reduced to \( \mathcal{H}_{\text{TM}} \).
- Since \( \mathcal{A}_{\text{TM}} \) is undecidable, so is \( \mathcal{H}_{\text{TM}} \).
Undecidable Problems

Theorem: $H_{TM}$ is undecidable.

Proof: Assume, by way of contradiction, that TM $R$ decides $H_{TM}$. Define a new TM, $S$, as follows:

- On input $⟨M, w⟩$,
- run $R$ on $⟨M, w⟩$.
- If $R$ rejects, reject.
- If $R$ accepts (meaning $M$ halts on $w$), simulate $M$ on $w$ until it halts (namely run $U$ on $⟨M, w⟩$).
- If $M$ accepted, accept; otherwise reject.

TM $S$ decides $A_{TM}$, a contradiction
Undecidable Problems

\[ H_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \} \]

Theorem: \( H_{TM} \) is undecidable.

What we actually did was a reduction from \( A_{TM} \) to \( H_{TM} \).

This will be formalized later.
Does a TM accept any string at all?

\[ \text{EMPTY}_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

**Theorem:** \( \text{EMPTY}_{TM} \) is undecidable.

**Proof structure:**

- By contradiction.
- Assume \( \text{EMPTY}_{TM} \) is decidable.
- Let \( R \) be a TM that decides \( \text{EMPTY}_{TM} \).
- Use \( R \) to construct \( S \), a TM that decides \( A_{TM} \).
Undecidable Problems (2)

\[ \text{EMPTY}_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

First attempt: When \( S \) receives input \( \langle M, w \rangle \), it calls \( R \) with input \( \langle M \rangle \).

- If \( R \) accepts, then reject, because \( M \) does not accept any string, let alone \( w \).
- But what if \( R \) rejects?

Second attempt: Let’s modify \( M \).
Undecidable Problems (2)

\[ \text{EMPTY}_{\text{TM}} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

Define \( M_1 \): on input \( x \),

1. if \( x \neq w \), reject.
2. if \( x = w \), run \( M \) on \( w \) and accept if \( M \) does.

\( M_1 \) either

- accepts just \( w \), or
- accepts nothing.
Machine $M_1$: on input $x$,

1. if $x \neq w$, reject.
2. if $x = w$, run $M$ on $w$ and accept if $M$ does.

Question:
Can a TM construct $M_1$ from $M$?

Answer:
Easily, because we need only hardwire $w$, and add a few extra states to perform the "$x = w?$" test.
Undecidable Problems (2)

\[
\text{EMPTY}_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \}.
\]

**Theorem:** \( \text{EMPTY}_{TM} \) is undecidable.

Define \( S \) as follows:

On input \( \langle M, w \rangle \), where \( M \) is a TM and \( w \) a string,

- Construct \( M_1 \) from \( M \) and \( w \).
- Run \( R \) on input \( \langle M_1 \rangle \),
- if \( R \) accepts, reject; if \( R \) rejects, accept.

\( \text{TM } S \) decides \( A_{TM} \), a contradiction.
Undecidable Problems (3)

Does a TM accept a regular language?

\[ \text{REG}_{\text{TM}} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular} \} \]

**Theorem:**
\( \text{REG}_{\text{TM}} \) is undecidable.

**Skeleton of Proof:**
- By contradiction.
- Assume \( \text{REG}_{\text{TM}} \) is decidable.
- Let \( R \) be a TM that decides \( \text{REG}_{\text{TM}} \).
- Use \( R \) to construct \( S \), a TM that decides \( A_{\text{TM}} \).

But how?
Undecidable Problems (3)

\[ \text{REG}_{\text{TM}} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular} \} \]

**Intuition:** Modify \( M \) so that the resulting TM accepts a regular language if and only if \( M \) accepts \( w \).

Design \( M_2 \) so that

- if \( M \) does not accept \( w \), then \( M_2 \) accepts \( \{0^n1^n | n \geq 0\} \) (non-regular)

- if \( M \) accepts \( w \), then \( M_2 \) accepts \( \Sigma^* \) (regular).
Given $M$ and $w$, construct $M_2$:

On input $x$,
1. If $x$ has the form $0^n1^n$, accept it.
2. Otherwise, run $M$ on input $w$ and accept $x$ if $M$ accepts $w$.

Claim:
- If $M$ does not accept $w$, then $M_2$ accepts $\{0^n1^n | n \geq 0\}$.
- If $M$ accepts $w$, then $M_2$ accepts $\Sigma^*$.
- The function: On input $\langle M, w \rangle$, output $\langle M_2 \rangle$, is computable.
Undecidable Problems (3)

\[
\text{REG}_{\text{TM}} = \{\langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular} \}
\]

Theorem: \( \text{REG}_{\text{TM}} \) is undecidable.

Define \( S \):

On input \( \langle M, w \rangle \),

1. Construct \( M_2 \) from \( M \) and \( w \).
2. Run \( R \) on input \( \langle M_2 \rangle \).
3. If \( R \) accepts, accept; if \( R \) rejects, reject.

\( \text{TM } S \) decides \( A_{\text{TM}} \), a contradiction
Are two TMs equivalent?

\[ \text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \} \]

**Theorem:** \( \text{EQ}_{\text{TM}} \) is undecidable.

We are getting tired of reducing \( A_{\text{TM}} \) to everything.
Let’s try instead a reduction from \( \text{EMPTY}_{\text{TM}} \) to \( \text{EQ}_{\text{TM}} \).
EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } \ L(M_1) = L(M_2) \} \\

Theorem: \ EQ_{TM} \text{ is undecidable.} \\
Idea:

- \ EMPTY_{TM} \text{ is the problem of testing whether a TM language is empty.}
- \ EQ_{TM} \text{ is the problem of testing whether two TM languages are the same.}
- If one of these two TM languages happens to be empty, then we are back to \ EMPTY_{TM}.
- So \ EMPTY_{TM} \text{ is a special case of } \ EQ_{TM}.

The rest is easy.
Undecidable Problems (4)

\[ \text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \} \]

**Theorem:** \( \text{EQ}_{\text{TM}} \) is undecidable.

Let \( M_{\text{NO}} \) be this TM: On input \( x \), reject.

Let \( R \) decide \( \text{EQ}_{\text{TM}} \).

Let \( S \) be: On input \( \langle M \rangle \):

1. Run \( R \) on input \( \langle M, M_{\text{NO}} \rangle \).
2. If \( R \) accepts, accept; if \( R \) rejects, reject.

If \( R \) decides \( \text{EQ}_{\text{TM}} \), then \( S \) decides \( \text{EMPTY}_{\text{TM}} \).
Bucket of Undecidable Problems

Same techniques prove undecidability of

- Does a TM accept a **decidable** language?
- Does a TM accept a **context-free** language?
- Does a TM accept a **finite** language?
- Does a TM halt on all inputs?
- Is there an input string that causes a TM to **traverse all its states**?
Reducibility

So far, we have seen many examples of reductions from one language to another, but the notion was neither defined nor treated formally.

Reductions play an important role in

- decidability theory (here and now)
- complexity theory (to come)

Time to get formal.
Mapping Reductions

**Definition:** Let $A$ and $B$ be two languages. We say that there is a **mapping reduction** from $A$ to $B$, and denote

$$A \leq_m B$$

if there is a **computable function**

$$f : \Sigma^* \rightarrow \Sigma^*$$

such that, for every $w$,

$$w \in A \iff f(w) \in B.$$ 

The function $f$ is called the **reduction** from $A$ to $B$. 

---

*Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.*
A mapping reduction converts questions about membership in $A$ to membership in $B$.
Mapping Reductions

Theorem:
If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.

Proof: Let
- $M$ be the decider for $B$, and
- $f$ the reduction from $A$ to $B$.

Define $N$: On input $w$
1. compute $f(w)$
2. run $M$ on input $f(w)$ and output whatever $M$ outputs.
Corollary: If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable.

In fact, this has been our principal tool for proving undecidability of languages other than $A_{TM}$.
Example: Halting

Recall that

\[
A_{TM} = \{\langle M, w \rangle | \text{TM } M \text{ accepts input } w \}\n\]

\[
H_{TM} = \{\langle M, w \rangle | \text{TM } M \text{ halts on input } w \}\n\]

Earlier we proved that

- \(H_{TM}\) undecidable
- by (de facto) reduction from \(A_{TM}\).

Let's reformulate this.
Define a **computable function**, $f$:

- input of form $\langle M, w \rangle$
- output of form $\langle M', w' \rangle$
- where $\langle M, w \rangle \in A_{TM} \iff \langle M', w' \rangle \in H_{TM}$. 

Example: Halting
Example: Halting

The following machine computes this function $f$.

$F =$ on input $\langle M, w \rangle$:

- Construct the following machine $M'$.
  - $M'$: on input $x$
    - run $M$ on $x$
    - If $M$ accepts, accept.
    - if $M$ rejects, enter a loop.
- output $\langle M', w \rangle$
Enumerability

**Theorem:**
If $A \leq_m B$ and $B$ is enumerable, then $A$ is enumerable.

Proof is same as before, using accepters instead of deciders.
Corollary: If $A \leq_m B$ and $A$ is not enumerable, then $B$ is not enumerable.
**Theorem:** Both $EQ_{TM}$ and its complement, $\overline{EQ}_{TM}$, are not enumerable. Stated differently, $EQ_{TM}$ is neither enumerable nor co-enumerable, or $EQ_{TM} \notin RE \cup coRE$.

- We first show that $A_{TM}$ is reducible to $EQ_{TM}$. The same function is also a mapping reduction from $A_{TM}$ to $\overline{EQ}_{TM}$, and thus $\overline{EQ}_{TM}$ is not enumerable.

- We then show that $A_{TM}$ is reducible to $\overline{EQ}_{TM}$. The new function is also a mapping reduction from $A_{TM}$ to $EQ_{TM}$, and thus $EQ_{TM}$ is not enumerable.
TM Equality

Claim: \( A_{TM} \) is reducible to \( EQ_{TM} \).

\[ f : A_{TM} \rightarrow EQ_{TM} \] works as follows:

\( F \): On input \( \langle M, w \rangle \)
- Construct machine \( M_1 \): on any input, reject.
- Construct machine \( M_2 \): on input \( x \), run \( M \) on \( w \). If it accepts, accept.
- Output \( \langle M_1, M_2 \rangle \).
TM Equality

\[ F: \text{On input } \langle M, w \rangle \]

- Construct machine \( M_1 \): on any input, reject.
- Construct machine \( M_2 \): on any input \( x \), run \( M \) on \( w \).
  
  If it accepts, accept \( x \).
- Output \( \langle M_1, M_2 \rangle \).

Note

- \( M_1 \) accepts nothing
- if \( M \) accepts \( w \) then \( M_2 \) accepts everything, and otherwise nothing.

- so \( \langle M, w \rangle \in A_{TM} \iff \langle M_1, M_2 \rangle \in \overline{EQ_{TM}} \).
**Claim:** $A_{TM}$ is reducible to $EQ_{TM}$.

$f : A_{TM} \rightarrow EQ_{TM}$ works as follows:

$F$: On input $\langle M, w \rangle$

- Construct machine $M_1$: on any input, accept.
- Construct machine $M_2$: on any input $x$, run $M$ on $w$. If it accepts, accept.
- Output $\langle M_1, M_2 \rangle$. 
TM Equality

\(F\): On input \(\langle M, w \rangle\)

- Construct machine \(M_1\): on any input, accept.
- Construct machine \(M_2\): on any input \(x\), run \(M\) on \(w\).
  
  If it accepts, accept.

- Output \(\langle M_1, M_2 \rangle\).

Note

- \(M_1\) accepts everything
- if \(M\) accepts \(w\), then \(M_2\) accepts everything, and otherwise nothing.
- \(\langle M, w \rangle \in A_{\text{TM}} \iff \langle M_1, M_2 \rangle \in \text{EQ}_{\text{TM}}.\)
Rice’s Theorem

By now, some of you may have become cynical and embittered.

- Like, been there, done that, bought the T-shirt.
- Looks like any non-trivial property of enumerable languages is undecidable.

That is correct.
Non Trivial Properties

**Theorem:** If $C$ is a proper, non-empty subset of the set of enumerable languages, then it is undecidable whether for a given TM $M$, $L(M)$ is in $C$.

(See problem 5.22 in Sipser’s book)
Rice’s Theorem

Theorem: If \( C \) is a proper non-empty subset of the set of enumerable languages, then it is undecidable whether for a given TM, \( M \), \( L(M) \) is in \( C \).

Proof by reduction from \( H_{TM} \) (does \( M \) halt on input \( x \)?).
Rice’s Theorem (proof - 2)

- Assume WLOG that $\emptyset \not\in C$.
- Otherwise, look at $\overline{C}$, also proper and non-empty.
- Also, there exists some $L \in C$ accepted by some $M_L$.
- continued . . .
Rice's Theorem (proof - 3)

Given $M$ and $x$, we will construct $M_0$ such that:

- If $M$ accepts $x$, then $L(M_0) \in C$.
- If $M$ does not accept $x$, then $L(M_0) \notin C$.

Deciding whether $M_0 \in C$ is the same as deciding whether $\langle M, x \rangle \in H_{TM}$!
Rice’s Theorem

Given $M$ and $x$, construct $M_0$ such that:

- If $M$ accepts $x$, then $L(M_0) \in C$.
- If $M$ does not accept $x$, then $L(M_0) \notin C$.

On input $y$:

1. Run $M$ on input $x$.
2. If $M$ halts, then run $M_L$ on $y$.
   - 1. if $M_L$ accepts, accept, and
   - 2. if $M_L$ rejects, reject.

Recall that $M_L$ accepts some language in $C$. 
Rice’s Theorem

**Theorem:** If $C$ is a proper non-empty subset of the set of enumerable languages, then it is undecidable whether for a given TM $M$, $L(M) \in C$.

Assume that TM $R$ decides $C$. Construct $S$ that decides $H_{TM}$.

On input $\langle M, x \rangle$,

1. Construct $M_0$ from $M$ and $x$, as above.
2. Run $R$ on input $M_0$.
3. If $R$ accepts, accept; if $R$ rejects, reject.
Reduction via Computation Histories

Important technique for proving undecidability.

- Useful for testing existence of some objects.
- For example, basis for proof of undecidability in Hilbert’s tenth problem,
  where "object" is integral root of polynomial.
- Other examples: Does a linear bounded TM accept the empty language?
- Does a context free grammar generate $\Sigma^*$?
Reminder: Configurations

Configuration:

\[1011q_70111\]

means:
- state is \(q_7\)
- LHS of tape is 1011
- RHS of tape is 0111
- head is on RHS 0
Configurations

- configuration $uaq_i bv$ yields $uq_j acv$ if $\delta(q_i, b) = (q_j, c, L)$
- Of course, $uaq_i bv$ yields $uacq_j v$ if $\delta(q_i, b) = (q_j, c, R)$
- Special case (left end of tape): $q_i bv$ yields $q_j cv$ if $\delta(q_i, b) = (q_j, c, blueL)$. 

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Let $M$ be a TM and $w$ an input string.

An **accepting** computation history for $M$ on $w$ is a sequence $C_1, C_2, \ldots, C_\ell$, where

- $C_1$ is the starting configuration of $M$ on $w$
- $C_\ell$ is an accepting configuration of $M$,
- each $C_i$ yields $C_{i+1}$ according to the transition function.

An **rejecting** computation history for $M$ on $w$ is the same, except

- $C_\ell$ is a rejecting configuration of $M$. 
Remarks

- Computation sequences are finite.
- If $M$ does not halt on $w$, no accepting or rejecting computation history exists.
- Notion is useful for both deterministic (one history) and non-deterministic (many histories) TMs.
Linear Bounded Automata

- A restricted form of TM.
- Cannot move off portion of tape containing input
- Rejects attempts to move head beyond input
- Size of input determines size of memory

![Turing machine vs Linear Bounded Automaton](image-url)
Question: Why linear?

Answer: Using a tape alphabet larger than the input alphabet increases memory by a constant factor.
Believe it or not, LBAs are quite powerful. The **deciders** for

- $A_{DFA}$ (does DFA accept?)
- $A_{CFG}$ (is string in CFG?)
- $\text{EMPTY}_{DFA}$ (is DFA trivial?)
- $\text{EMPTY}_{CFG}$ (is CFG empty?)

are all LBAs.

Every CFL can be decided by a LBA.

Not easy to find a **natural, decidable language** that **cannot be decided** by an LBA.
Define

\[ A_{LBA} = \{ \langle M, w \rangle | M \text{ is an LBA that accepts } w \} \]

**Question:** Is \( A_{LBA} \) decidable?
Lemma: Let $M$ be a LBA with

- $q$ states
- $g$ symbols in tape alphabet

On an input of size $n$, LBA has exactly $qng^n$ distinct configurations, because a configuration involves:

- control state ($q$ possibilities)
- head position ($n$ possibilities)
- tape contents ($g^n$ possibilities)
Theorem

Theorem: $A_{\text{LBA}}$ is decidable.

Idea:

- Simulate $M$ on $w$.
- But what do we do if $M$ loops?
- Must detect looping and reject.
- $M$ loops if and only if it repeats a configuration.
- Why? And is this also true of “regular” TMs?
- By pigeon hole, if our LBA $M$ runs long enough, it must repeat a configuration!
Theorem:

A_{LBA} is decidable.

On input \langle M, w \rangle, where \( M \) is an LBA and \( w \in \Sigma^* \)

1. Simulate \( M \) on \( w \),

2. While maintaining a counter.

3. Counter incremented by 1 per each simulated step (of \( M \)).

4. Keep simulating \( M \) for \( qng^n \) steps, or until it halts (whichever comes first)

5. If \( M \) has halted, accept \( w \) if it accepted by \( M \), and reject \( w \) if it rejected by \( M \).

6. reject \( w \) if counter limit reached (\( M \) has not halted).
More LBAs

Here is a related problem.

\[ \text{EMPTY}_{\text{LBA}} = \{ \langle M \rangle | \text{M is an LBA and } L(M) = \emptyset \} \]

**Question:** Is \( \text{EMPTY}_{\text{LBA}} \) decidable?

Surprisingly though, LBAs do have undecidable problems too!
More LBAs

$\text{EMPTY}_{\text{LBA}} = \{ \langle M \rangle | M \text{ is an LBA and } L(M) = \emptyset \}$

Theorem: $\text{EMPTY}_{\text{LBA}}$ is undecidable.

Proof by reduction using computation histories.
More LBAs

\[ \text{EMPTY}_{\text{LBA}} = \{ \langle M \rangle | M \text{ is an LBA and } L(M) = \emptyset \} \]

**Theorem** \( \text{EMPTY}_{\text{LBA}} \) is undecidable.

**Proof by reduction from** \( A_{\text{TM}} \).

If \( \text{EMPTY}_{\text{LBA}} \) were decidable, then \( A_{\text{TM}} \) would also be.

**Question:** Suppose that \( \text{EMPTY}_{\text{LBA}} \) is decidable. How can we use this supposition to decide \( A_{\text{TM}} \)?

Let \( R \) be a decider for the language \( \text{EMPTY}_{\text{LBA}} \).
More LBAs

Given $M$ and $w$, we will construct an LBA, $B$.

- $L(B)$ will contain exactly all accepting computation histories for $M$ on $w$
- $M$ accepts $w$ iff $L(B) \neq \emptyset$.
- Will use $R$ to decide whether $L(B) = \emptyset$.
- Then we can decide whether $M$ accepts $w$. 
More LBAs

It is not enough to show that $B$ exists.

We must show that a TM can construct $\langle B \rangle$ from $\langle M, w \rangle$.

Assume an accepting computation history is presented as a string:

$$
\# C_1 \# C_2 \# C_3 \# \cdots \# C_\ell \# ,
$$

with descriptions of configurations separated by $\#$ delimiters.
The LBA $B$ works as follows:

On input $x$, the LBA $B$:

- breaks $x$ according to the $\#$ delimiters
- identifies strings $C_1, C_2, \ldots, C_\ell$.
- then checks that following conditions hold:
  - Each $C_i$ are a configuration of $M$
  - $C_1$ is the start configuration of $M$ on $w$
  - Every $C_{i+1}$ follows from $C_i$ according to $M$
  - $C_\ell$ is an accepting configuration
The LBA

- Checking that each $C_i$ is a configuration of $M$ is easy: All it means is that $C_i$ includes exactly one $q$ symbols.
- Checking that $C_1$ is the start configuration on $w$, $q_0w_1w_2\cdots w_n$, is easy, because the string $w$ was “wired into” $B$.
- Checking that $C_\ell$ is an accepting configuration is easy, because $C_\ell$ must include the accepting state $q_a$.
- The only hard part is checking that each $C_{i+1}$ follows from $C_i$ by $M$’s transition function.
The Hard Part

Checking that for all \( i \), \( C_{i+1} \) follows from \( C_i \) by \( M \)'s transition function.

- \( C_i \) and \( C_{i+1} \) almost identical, except for positions under head and adjacent to head.
- These positions should updated according to transition function.

Do this verification by

- zig-zagging between corresponding positions of \( C_i \) and \( C_{i+1} \).
- use “dots” on tape to mark current position
- all this can be done in space allocated by input \( x \)
Important!

The LBA, $B$, accepts the string $x$ if and only if $x$ equals an accepting computation history of $M$ on $w$.

Therefore $L(B)$ is either empty or a singleton $\{x\}$.

We construct $B$ in order to feed it to the claimed decider, $R$, of $\text{EMPTY}_{\text{LBA}}$ (which we assume to exist).

Once this decider returns its answer, we invert this answer to decide whether $M$ accepts $w$. 
The Proof

Suppose TM $R$ decides $\text{EMPTY}_{\text{LBA}}$.

Define TM $S$ that decides $A_{\text{TM}}$:

On input $\langle M, w \rangle$

1. Construct LBA, $B$, from $M$ and $w$ as described above.
2. Run $R$ on $\langle B \rangle$.
3. if $R$ rejects, accept; if $R$ accepts, reject.

If $R$ accepts $\langle B \rangle$

- $M$ has no accepting computation history on $w$
- $M$ does not accept $w$
- So $S$ rejects $\langle M, w \rangle$
If $R$ rejects $\langle B \rangle$

- the language of $B$ is non-empty
- the only string $B$ can accept is an accepting computation of $M$ on $w$
- thus $M$ accepts $w$
- So $S$ accepts $\langle M, w \rangle$.

To conclude, $S$ decides $A_{TM}$, a contradiction.
A CFG Question

SENTENCE

NOUN-PHRASE

ARTICLE   NOUN

a  boy

VERB

sees
Another Use for Computation Histories

We have already seen an algorithm to check whether a context-free grammar is empty. On input \(\langle G \rangle\) where \(G\) is a CFG:

1. Mark all terminal symbols in \(G\).
2. Repeat until no new variables become marked:
   3. Mark any \(A\) where
      \[
      A \rightarrow U_1 U_2 \ldots U_k
      \]
      and each \(U_i\) has already been marked.
   4. If start symbol marked, accept, otherwise reject.
Another Use for Computation Histories

So the language $\text{EMPTY}_{\text{CFG}}$ is decidable.

**Question:** What about the complementary question: Does a CFG generate all strings?

$$\text{All}_{\text{CFG}} = \{\langle G \rangle | G \text{ is a CFL and } L(G) = \Sigma^* \}$$
Theorem: All_{CFG} is undecidable.

Proof by reduction and contradiction:

- Assume All_{CFG} is decidable.
- show that A_{TM} is then decidable.
- for a TM, M, and input, w, construct a CFG, G
- G generates all strings that are not accepting computation histories for M on w
- if M does not accept w, G generates all strings
- if M does accept w, G does not generate the accepting computation history.
An accepting computation history appears as
\[#C_1#C_2# \ldots #C_\ell#,\] where
- $C_1$ is the starting configuration of $M$ on $w$,
- $C_\ell$ is an accepting configuration of $M$,
- Each $C_i$ yields $C_{i+1}$ by transition function of $M$.

A string is not an accepting computation history if it fails one or more of these conditions.
Does a CFG Generate All Strings?

Instead of the CFG, $G$, we construct a PDA, $D$. $D$ non-deterministically “guesses” which condition is violated.

- then verifies the guessed violation:
  - Is there some $C_i$ that is not a configuration of $M$ (number of $q$ symbols $\neq 1$)?
  - Is $C_1$ not the starting configuration of $M$ on $w$?
  - Is $C_\ell$ not an accepting configuration of $M$?
  - Does $C_i$ not yield $C_{i+1}$ by the transition function of $M$?
- Like before, last condition is the tricky one to check.

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Does a CFG Generate All Strings?

Does \( C_i \) not yield \( C_{i+1} \)?

Idea:

- Scan input. Nondeterministically decide "violating configuration" \( C_i \) was reached.
- Push \( C_i \) onto the stack till \#.
- Scan \( C_{i+1} \) and pop matching symbols of \( C_i \)
  - check if \( C_i \) and \( C_{i+1} \) match everywhere, except . . .
  - around the head position,
  - where difference dictated by transition function for \( M \).
Problem: When $D$ pops $C_i$ from stack, $C_i$ is in reverse order. Ignoring the local changes around head position, what we were trying to identify the language $x \# y$, with $x \neq y$. While this can be done in principle by a non deterministic PDA (see problem 2.26 in Sipser), there is a simpler way. So far, we used a “straight” notion of accepting computation histories.

\[
\begin{array}{ccccccc}
\# & \rightarrow & \# & \rightarrow & \# & \rightarrow & \# & \rightarrow & \# & \cdots & \# & \rightarrow & \# \\
C_1 & & C_2 & & C_3 & & C_4 & & C_\ell & & & & & \\
\end{array}
\]
Does a CFG Generate All Strings?

So far, we used a “straight” notion of accepting computation histories

\[
\# \rightarrow # \rightarrow # \rightarrow # \rightarrow \# \cdots \# \rightarrow \#
\]

But in this modern age, why not employ an alternative notion of accepting computation history, one that will make the life of our PDA much easier?

**Solution:** Write the accepting computation history so that every other configuration is in reverse order.

\[
\# \leftarrow # \leftarrow # \leftarrow # \leftarrow \# \cdots \# \leftarrow \#
\]

This takes care of difficulty in the proof.
Wrapping Things Up

Given \( \langle M, w \rangle \), we construct (algorithmically) a PDA, \( D \), which rejects the string \( x \) if and only if \( x \) equals an accepting computation history of \( M \) on \( w \), written in the "alternating format".

Therefore \( L(D) \) is either \( \Sigma^* \) or \( \Sigma^* \setminus \{ x \} \).

We construct \( D \) in order to feed it to the claimed decider, \( R \), of \( \text{All}_{\text{CFG}} \) (which we assume to exist).

Once this decider returns its answer, we invert this answer to decide whether \( M \) accepts \( w \).

But then we can use \( R \) to decide \( A_{\text{TM}} \), a contradiction. ♣