Mapping reductions
More undecidable languages
Undecidability by Rice Theorem
$\mathcal{RE}$-Completeness
Reductions by computational histories

Sipser’s book, Chapter 5, Sections 5.1, 5.3
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 $B$. 
A mapping reduction converts questions about membership in $A$ to membership in $B$

Notice that $A \leq_m B$ implies $\overline{A} \leq_m \overline{B}$. 
Mapping Reductions: Reminders

Theorem 1:
If $A \leq_mB$ and $B$ is decidable, then $A$ is decidable.

Theorem 2:
If $A \leq_mB$ and $B$ is recursively enumerable, then $A$ is recursively enumerable.
Corollary 1: If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable.

Corollary 2: If $A \leq_m B$ and $A$ is not in $\mathcal{RE}$, then $B$ is not in $\mathcal{RE}$.

Corollary 3: If $A \leq_m B$ and $A$ is not in $\text{co}\mathcal{RE}$, then $B$ is not in $\text{co}\mathcal{RE}$.
Mapping Reductions in General

Mapping reductions are applicable to wide areas in mathematics, not only computing. For example, consider the following two sets:

1. The set of equations of the form $Ax^2 + By + C$ where the coefficients are integers, that have a root consisting of positive integers.

2. The set of knots that can be untied (without tearing or breaking the rope) leaving at most $\ell$ loops.

Even though these two sets have very different nature, they are reducible to each other (by mapping reductions).

In this course we concentrate mainly on computing related problems, but reductions are relevant in much wider scopes.
Bucket of Undecidable Problems

Same techniques prove undecidability of

- Does a TM accept a **decidable** language?
- Does a TM accept a **regular** language?
- Does a TM accept a **context-free** language?
- Does a TM accept a **finite** language?
- Does a TM accept a language that contains **all prime numbers**?
- Does a TM accept a language that contains **all quartets of positive integers** \( > 2 \) satisfying \( x^n + y^n = z^n \)?
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 TMs is undecidable.

That is correct.
Rice’s Theorem – Restatement

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

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

**Theorem** Let $C$ be a proper non-empty subset of the set of enumerable languages. Denote by $L_C$ the set of all TMs encodings, $\langle M \rangle$, such that $L(M)$ is in $C$. Then $L_C$ is undecidable.

Proof by reduction from $A_{TM}$.

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

- If $M$ accepts $w$, then $\langle M_0 \rangle \in L_C$.
- If $M$ does not accept $w$, then $\langle M_0 \rangle \notin L_C$. 
Rice’s Theorem

- Without loss of generality, $\emptyset \not\in C$.
- (Otherwise, look at $\overline{C}$, also proper and non-empty.)
- Since $C$ is not empty, there exists some language $L \in C$. Let $M_L$ be a TM accepting this language (recall $C$ contains only recursively enumerable languages).
- continued . . .
Rice’s Theorem

Given $\langle M, w \rangle$, construct $M_0$ such that:

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

$M_0$ on input $y$:

1. Run $M$ on $w$.

2. If $M$ accepts $w$, run $M_L$ on $y$.
   a. if $M_L$ accepts, accept, and
   b. if $M_L$ rejects, reject.

Claim: The transformation $\langle M, w \rangle \rightarrow \langle M_0 \rangle$ is a mapping reduction from $A_{TM}$ to $L_C$. 
Rice’s Theorem

Proof: $M_0$ on input $y$:

1. Run $M$ on $w$.
2. If $M$ accepts, run $M_L$ on $y$.
   a. if $M_L$ accepts, accept, and
   b. if $M_L$ rejects, reject.

The machine $M_0$ is simply a concatenation of two known TMs – the universal machine, and $M_L$.

Therefore the transformation $\langle M, w \rangle \rightarrow \langle M_0 \rangle$ is a computable function, defined for all strings in $\Sigma^*$.

(But what do we actually do with strings not of the form $\langle M, w \rangle$?)
Rice’s Proof (Concluded)

- If $\langle M, w \rangle \in A_{TM}$ then $M_0$ gets to step 2, and runs $M_L$ on $y$.
- In this case, $L(M_0) = L$, so $L(M_0) \in C$.
- On the other hand, if $\langle M, w \rangle \not\in A_{TM}$ then $M_0$ never gets to step 2.
- In this case, $L(M_0) = \emptyset$, so $L(M_0) \not\in C$.
- This establishes the fact that $\langle M, w \rangle \in A_{TM}$ iff $\langle M_0 \rangle \in L_C$. So we have $A_{TM} \leq_m L_C$, thus $L_C$ is undecidable.

♣

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Rice’s Theorem (Reflections)

Rice’s theorem can be used to show undecidability of properties like

- “does $L(M)$ contain infinitely many primes”
- “does $L(M)$ contain an arithmetic progression of length 15”
- “is $L(M)$ empty”

Decidability of properties related to the encoding itself cannot be inferred from Rice. For example “does $\langle M \rangle$ has an even number of states” is decidable.

Properties like “does $M$ reaches state $q_6$ on the empty input string” are undecidable, but this does not follow from Rice’s theorem.

Rice does not say anything on membership in $\mathcal{RE}$ of languages like “is $L(M)$ finite”.

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Consider the language $L_{\text{infinite}} = \{\langle M \rangle \mid L(M) \text{ is infinite}\}$. By Rice Theorem, this language is not in $\mathcal{R}$. We want to show that $L_{\text{infinite}} \notin RE$.

**Idea:** Reduction from $H_{\text{TM}}$. So we are after a reduction $f : \langle M, w \rangle \mapsto \langle M_0 \rangle$ such that

- If $M$ halts on $w$ then $L(M_0)$ is finite.
- If $M$ does not halts on $w$ then $L(M_0)$ is infinite.

This looks a bit tricky…
Reductions via Controlled Executions (2)

We are after a reduction $f : \langle M, w \rangle \mapsto \langle M_0 \rangle$ such that

- If $M$ halts on $w$ then $L(M_0)$ is finite.
- If $M$ does not halts on $w$ then $L(M_0)$ is infinite.

Given $\langle M, w \rangle$, construct the TM $M_0$ as following:

- $M_0$ on input $y$
- Runs the universal machine $U$ on $\langle M, w \rangle$ for $|y|$ steps.
- If $U$ did not halt in that many steps, $M_0$ accepts $y$.
- If $U$ did halt in that many steps, $M_0$ rejects $y$.

$f(\langle M, w \rangle) = M_0$. Let us examine $L(M_0)$.

(Remark: $M_0$ halts on all inputs.)
Reductions via Controlled Executions (3)

\[ f(\langle M, w \rangle) = M_0. \] Let us examine \( L(M_0) \).

- If \( M \) does not halt on \( w \), then \( M_0 \) accepts all \( y \), so \( L(M_0) = \Sigma^* \), and thus \( \langle M_0 \rangle \in L_{\text{infinite}} \).

- If \( M \) does halt on \( w \) after \( k \) simulation steps, then \( M_0 \) accepts only \( y \)s of length smaller than \( k \), so \( L(M_0) \) is finite, and thus \( \langle M_0 \rangle \notin L_{\text{infinite}} \).

We have shown that \( \overline{H_{\text{TM}}} \leq_m L_{\text{infinite}} \).

Since \( \overline{H_{\text{TM}}} \notin \mathcal{RE} \), this implies \( L_{\text{infinite}} \notin \mathcal{RE} \).  ♠
**RE-Completeness**

**Question:** Is there a language $L$ that is hardest in the class $\text{RE}$ of enumerable languages (languages accepted by some TM)?

**Answer:** Well, you have to define what you mean by “hardest language”.

**Definition:** A language $L_0 \subseteq \Sigma^*$ is called $\text{RE}$-complete if the following holds

- $L_0 \in \text{RE}$ (membership).
- For every $L \in \text{RE}$, $L \leq_m L_0$ (hardness).
**RE-Completeness**

Definition A language $L_0 \subseteq \Sigma^*$ is called $\mathcal{RE}$-complete if the following holds

- $L_0 \in \mathcal{RE}$ (membership).
- For every $L \in \mathcal{RE}$, $L \leq_m L_0$ (hardness).

The second item means that for every enumerable $L$ there is a mapping reduction $f_L$ from $L$ to $L_0$. The reduction $f_L$ depends on $L$ and will typically differ from one language to another.
**RE-Completeness**

**Question:** Having defined a reasonable notion, we should make sure it is not vacuous, namely verify there is at least one language satisfying it.

**Theorem** The language $A_{TM}$ is RE-Complete.

**Proof:**

- The universal machine $U$ accepts the language $A_{TM}$, so $A_{TM} \in \mathcal{RE}$.
- Suppose $L$ is in $\mathcal{RE}$, and let $M$ be a TM accepting it. Then $f_L(w) = \langle M, w \rangle$ is a mapping reduction from $L$ to $A_{TM}$ (why?).
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 $\delta(q_i, b) = (q_j, c, L)$. 
Computation Histories

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.

- A **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.
A CFG Question

SENTENCE

NOUN-PHRASE

ARTICLE

a

NOUN

boy

VERB

sees

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Emptiness of CFGs

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.
Using Computation Histories for CFGs

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

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

$$\text{All}_{\text{CFG}} = \{ ⟨G⟩ | G \text{ is a CFL and } L(G) = \Sigma^* \}$$
Does a CFG Generate All Strings?

Theorem: $\text{All}_{\text{CFG}}$ is undecidable.

Proof by reduction from $A_{\text{TM}}$ to $\overline{\text{All}_{\text{CFG}}}$:

- Given $\langle M, w \rangle$, construct a coding of a CFG, $\langle G \rangle$
- $G$ 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.
Does a CFG Generate All Strings?

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, $D$ (recall equivalence). $D$ 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 $M$?

- The last condition is the tricky one to check.
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.
Does a CFG Generate All Strings?

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

\[
\begin{align*}
&\# \to \# \to \# \to \# \to \# \cdots \to \# \\
&C_1 &C_2 &C_3 &C_4 &C_\ell
\end{align*}
\]

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.

\[
\begin{align*}
&\# \leftrightarrow \# \leftrightarrow \# \leftrightarrow \# \leftrightarrow \# \\
&C_1 &C_2 &C_3 &C_4 &C_\ell
\end{align*}
\]

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

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

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

This $D$ has an equivalent (and efficiently described) CFG, $G$, namely $L(D) = L(G)$. So $L(G)$ is either $\Sigma^*$ or $\Sigma^* \setminus \{z\}$.

The mapping $\langle M, w \rangle \mapsto \langle G \rangle$ is thus a reduction from $A_{TM}$ to $\overline{All_{CFG}}$.

Since $A_{TM} \notin \mathcal{R}$ we get $\overline{All_{CFG}} \notin \mathcal{R}$.

As $\mathcal{R}$ is closed under complement, we conclude that $\overline{All_{CFG}} \notin \mathcal{R}$.
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
Linear Bounded Automata

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

$A_{\text{DFA}}$ (does DFA accept?)

$A_{\text{CFG}}$ (is string in CFG?)

EMPTY$_{\text{DFA}}$ (is DFA trivial?)

EMPTY$_{\text{CFG}}$ (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.
A Language

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 \( A_{LBA} \) 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

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 was accepted by $M$, and reject $w$ if it was rejected by $M$.
6. reject $w$ if counter limit reached ($M$ has not halted).
More LBAs

Here is a related problem.

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

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:

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

with descriptions of configurations separated by # delimiters.
The LBA

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

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
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 be updated according to the 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 \( x \)
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 $\text{EMPTY}_{LBA}$ (which we assume to exist).

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

Important!
The Proof

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

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

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 $\langle M, w \rangle$
The Proof (cont.)

If $R$ rejects $⟨B⟩$

- 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 $⟨M, w⟩$.

To conclude, $S$ decides $A_{TM}$, a contradiction. ♣