Today:

Finish discussing reductions

Then:

Graded exam 2
Review for final exam
World domination

Example reduction:

Empty language?

$E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \}$

**Theorem: $E_{TM}$ is undecidable**

Proof idea:

- By contradiction
- Assume $E_{TM}$ is decidable
- Let $R$ be a TM that decides $E_{TM}$
- Use $R$ to construct $S$, a TM that decides $A_{TM}$
• When $S$ receives input $\langle M, w \rangle$, it calls $R$ with input $M$
• If $R$ accepts, then reject, because $M$ does not accept $w$
  – i.e., $R$ says $L(M)$ is empty, so it must not accept $w$
• But what if $R$ rejects?
  – Then $L(M)$ is not empty
  – So, we don’t know if $M$ accepts $w$ or not
  – **Solution: Modify $M$**

Define $M_2$, a modified version of $M$:
• On input $x$:
  – Run $M$ on $w$
    1. If $M$ rejects, **reject**. (We want $M_2$ to be a TM that accepts the empty language.)
    2. Otherwise, if $M$ accepts $w$, then we have to set things up so that $M_2$ accepts something other than the empty language. As a convenience, have it accept a single string, $w$:
      – If $x = w$, **accept**
      – Else, **reject**

• For some reductions, we simulate $M$ on $w$ first, before we test $x$
• Code for $M_w$ looks like:
  – Simulate $M$ on $w$
  – If $M$ rejects, **reject**
  – Otherwise, **accept** exactly when $x$ has some easily tested property $P$

An example of this pattern is the proof of Rice’s theorem; see Sipser’s solved exercise 5.28
Tips

- Note that the string $w$ is an input to the machine $S$, i.e., the machine that decides $A_{TM}$, and the string $x$ is an input to the Turing machine $M_2$.
- The input to the Turing machine $R$ is usually the code for the machine $M_2$, but $w$ and $x$ are not inputs to $R$:
  - The value of $w$ may be hardcoded into the code for $M_2$.
  - The value of $x$ is **not specified**. You can imagine that $M_2$ asks the user to input a value for $x$.

Reductions Involving Languages

Is an arbitrary Turing-recognizable language empty?

- Reduce $A_{TM}$ to this problem
- Suppose we are given a TM $M$ and a string $w$
- Modify $M$ such that
  1. $M$ first saves its input (say, $x$) on some special part of its tape
  2. Whenever $M$ enters a final state, it checks the saved input and accepts **if and only if** it is $w$
- Do this by changing $\delta$ in a simple way:
  - Create a machine $M_w$ such that $L(M_w) = L(M) \cap \{w\}$

- Then construct a corresponding grammar $G_w$
- Clear that $L(G_w)$ is nonempty if and only if $w \in L(M)$
- Assume there exists an algorithm $A$ for deciding whether or not $L(G) = \emptyset$
- Let $T$ denote an algorithm by which we generate $G_w$
- Put $T$ and $A$ together to create a TM that for any $M$ and $w$, tells us whether or not $w \in L(M)$:
Conclusion

• If such a Turing machine existed, we would have a membership algorithm for RE languages
• But we know membership for REs is undecidable
• So, \( L(G) = \emptyset \) is not decidable

Is \( L(M) \) finite?

• Use \( HALT_{TM} \) again
• From \( M \) construct another TM \( M' \) in which
  – Halting states of \( M \) are changed so that if any one is reached, all input is accepted by \( M' \)
  – Achieve this by having any halting configuration go to a final state
  – \( M \) is modified so that \( M' \) first generates \( w \) on its tape, then performs the same computations as \( M \) using the newly created \( w \) and some other unused space
  – Moves of \( M' \) after writing \( w \) on its tape are the same as those performed by \( M \), had it started in the original configuration \((q_0, w)\)
• If \( M \) halts on any configuration, then \( M' \) will halt in a final state

More reductions

• If \((M, w)\) halts, \( M' \) will halt in a final state for all input
• If \((M, w)\) does not halt, \( M' \) will not halt either and so will accept nothing
• In other words: \( M' \) either accepts the infinite language \( \Sigma^* \) or the finite language \( \emptyset \)
• If we assume the existence of an algorithm \( A \) that tells whether \( L(M') \) is finite, then can solve the halting problem:

\[
\begin{array}{c}
M, w \quad \text{GENERATE} \quad M' \\
\text{FINITENESS ALGORITHM} \quad A \\
L(M') \text{ finite} \\
\text{Does not halt} \\
L(M') \text{ not finite} \\
\text{Halts}
\end{array}
\]
Does $L(M)$ contain two strings of the same length?

- Use the same approach as in the previous example, except when $M'$ reaches a halting configuration, it is modified to accept two strings $a$ and $b$ of equal length.
- To do this:
  1. Initial input is saved and at the end of the computation compared to $a$ and $b$.
  2. Accepts only these two strings.
- Therefore, if $(M, w)$ halts, $M'$ will accept two strings of equal length.
- Otherwise $M'$ accepts nothing.
- Finish up the argument as in the previous problem.

Are two TMs equivalent?

$EQ_{\text{TM}} = \{\langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

Theorem: $EQ_{\text{TM}}$ is undecidable

- We are getting tired of reducing everything to $A_{\text{TM}}$ or $HALT_{\text{TM}}$.
- Let’s try reduction to $E_{\text{TM}}$.

Theorem: $EQ_{\text{TM}}$ is undecidable

Proof Idea:

- $E_{\text{TM}}$ is the problem of testing whether a TM’s language is empty.
- $EQ_{\text{TM}}$ 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 $E_{\text{TM}}$.
- So $E_{\text{TM}}$ is a special case of $EQ_{\text{TM}}$.
- The rest is easy.

Let $M_{\text{NO}}$ be the Turing machine:

- **1. reject**

Let $R$ decide $EQ_{\text{TM}}$

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.
Bucket of Undecidable Problems

Same techniques prove undecidability of
- Does a TM accept a decidable language?
- Does a TM accept a context-free language?
- Is there an input string that causes a TM to traverse all its states?
- Does $L(M)$ contain any string of length 5?

*These are (creatively) called “similar questions”*

Rice’s Theorem

- By now, some of you may have become cynical and embittered
- Like, been there, done that, bought the T-shirt

Rice’s Theorem

- By now, some of you may have become cynical and embittered
- Like, been there, done that, bought the T-shirt
  - $500 Raf Simons sweatshirt
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**

If $C$ is a proper non-empty subset of the set of enumerable languages, then it is undecidable whether for a given Turing machine $M$, $L(M)$ is in $C$

Undecidable Problems About CFLs

PCP is a convenient tool for studying undecidable questions about CFLs

Example:

- You have been asked to convert regular expressions to DFAs, but there’s an algorithm to convert REs to DFAs, so it’s possible for you to succeed
- Suppose you are asked to take a CFG and tell whether it is ambiguous. You can’t do it because the problem is undecidable!
Reduction of PCP to CFG Ambiguity Problem

Given lists $A$ and $B$, construct grammar as follows:

- $S \to A \mid B$.
- $A$ is the start symbol for a grammar from list $A$; $B$ is the same for list $B$.
- If there is a solution to the PCP instance, then the same string can be derived starting $S \Rightarrow A$ and $S \Rightarrow B$.
  
  Conversely, the only way a string can have two leftmost derivations is if they begin in these two ways, because the grammar of each list is unambiguous.

Example

Use the lists $\langle 1, 0, 010, 11 \rangle$ and $\langle 10, 10, 01, 1 \rangle$

Let $a, b, c, d$ stand for the four index integers

- That is, we have

The grammar is:

\[
S \to A \mid B \\
A \to 1Aa \mid 0Ab \mid 010Ac \mid 11Ad \mid \varepsilon \\
B \to 10Ba \mid 10Bb \mid 01Bc \mid 1Bd \mid \varepsilon
\]

Each string has a unique derivation from $A$ and $B$.

Ambiguity can only come from $S$.

A string with two leftmost derivations:

10101001011 dccaba

CONCLUSION:

The grammar constructed from $(A, B)$ is ambiguous if and only if there is a solution to instance $(A, B)$ of PCP.

- In this case, there is a way to get a corresponding pair of strings.

Language theory and computation
A Hierarchy of Formal Languages and Automata

- We have already seen the following:
  - A language is **recursively enumerable** if there is a TM that accepts it
  - A language is **recursive** if there is a TM that accepts it and halts on every $w$ in $\Sigma^*$
  - There are languages that are **not recursively enumerable** (i.e., the ones we showed are not countable using the diagonalization argument)
  - There are RE languages whose complement is not RE
  - If both a language and its complement are RE then both are recursive
  - There are RE languages that are not recursive – i.e., the family of recursive languages is a proper subset of the family of RE languages

Unrestricted Grammars

A grammar $(V,T,S,P)$ is **unrestricted** if all the productions are of the form $u \to v$, where $u$ is in $(V \cup T)^+$ and $v$ is in $(V \cup T)^*$

- Basically, no restrictions imposed on productions
  - Any number of variables on the left and right-hand sides
  - Can occur in any order
  - Only restriction is that $\varepsilon$ cannot appear on the left side of a production

Any language generated by an unrestricted grammar is **recursively enumerable**

- **Proof**: the grammar in effect defines a procedure for enumerating all the strings in the language systematically
- Since the set of productions is finite, we can simulate the derivations on a TM to enumerate every string in the language

The Chomsky Hierarchy

- **Recursively Enumerable Languages**
  - Turing machine
- **Context-Sensitive Languages**
  - Linear bounded automata
- **Context-Free Languages**
  - Pushdown automata
- **Regular Languages**
  - Finite automata

Expanded Hierarchy

- **Non-RE Languages**
- **Recursive Languages**
  - Turing Machine that halts
- **Context-Sensitive Languages**
  - Linear Bounded Automata
- **Context-Free Languages**
  - Pushdown Automata
- **Deterministic CFL**
  - Deterministic PDA
- **Regular Languages**
  - Finite Automata
Congratulations!
You Now Know The Whole Story!

Appendix

Undecidable Problem
Is the Intersection of Two CFLs Empty?

• Consider the two list languages from a PCP instance. They have an empty intersection if and only if the PCP instance has no solution.
Complements of List Languages

- We can get other undecidability results about CFLs if we first establish that the complement of a list language is a CFL.
- PDA is easier approach.
- Accept all ill-formed input (not a sequence of symbols followed by indexes) using the state.
- For inputs that begin with symbols from the alphabet of the PCP instance, store them on the stack, accepting as we go.
- When index symbols start, pop the stack, making sure that the right strings were found on top of the stack; again, keep accepting until…
- When we expose the bottom-of-stack marker, we have found a sequence of strings from the PCP list and their matching indexes. This string is not in the complement of the list language, so don’t accept.
- If more index symbols come in, then we have a mismatch, so start accepting again and keep on accepting.

Undecidable Problem: Is a CFL Equal to $\Sigma^*$?

- Take an instance of PCP, say lists $A$ and $B$.
- The union of the complements of their two list languages is $\Sigma^*$ if the instance has no solution, and something less if there is a solution.

Undecidable Problem
Is the Intersection of Two CFLs Regular?

- Key idea: the intersection of list languages is regular if and only if it is empty. Thus, PCP reduces to regularity of intersection for CFLs.
- Obviously, if empty, it is regular.
- Suppose the intersection of two list languages, for $A$ and $B$, $L_A \cap L_B$ is nonempty. Then there is a solution to this instance of PCP, say string $w$ and string of index symbols $i$.
  - Example: for the running PCP instance, $w = 10101001011$ and $i = abaccd$. 
• Then $i^k$ is an index sequence that yields solution $w^k$ for all $k$.
  – General principle: concatenation of PCP solutions is a solution.
• Consider homomorphism $h(0) = w$ and $h(1) = i^R$.
• $h^{-1}(L_A \cap L_B)$ is $\{0^n1^n \mid n \geq 1\}$
• Since regular languages are closed under inverse homomorphism, if the intersection were regular, so would $h^{-1}(L_A \cap L_B)$ be.
• Since we know this language is not regular, we conclude that $L_A \cap L_B$ is not regular.