Reductions

Assignments

Assignment 6 and international editions
Assignment 7 due today
Assignment 8 out later today
    Last one!

Introduction to reductions

CS Dept Holiday Party

Friday, December 14, 1:30–3:30
College Center Multi Purpose Room (go up the stairs by the Express)
Who’s invited? You are!
RSVP: http://evite.me/SETx6EYASK
Reductions

To prove a problem $P_1$ to be hard in some sense (e.g., undecidable), we can reduce computing $P_2$, a known hard problem, to computing $P_1$.

We give a fixed algorithm that will take a string $w$ and:
- If $w$ is an instance of $P_2$, construct a string $x$ that is an instance of $P_1$.
- If $w$ is not an instance of $P_2$, construct a string $x$ that is not an instance of $P_1$.

• We then argue: If $P_1$ were decidable, we could:
  • Use the algorithm to transform $P_2$’s input $w$ to $P_1$’s input $x$.
  • Test $x$ for membership in $P_1$ as a way to decide $P_2$.
• Since $P_2$ is undecidable, we have a contradiction of the assumption that $P_1$ is decidable.
• The same idea works for showing $P_1$ not to be Turing-recognizable, but now $P_2$ must be non-Turing-recognizable, and the transformation from instances of $P_2$ to instances of $P_1$ may be a procedure, not necessarily an algorithm.

Common error: trying to do the reduction in the wrong direction (i.e., reducing the one we want to prove undecidable to the known hard problem)

Example reduction: Post’s Correspondence Problem

Post’s Correspondence Problem

An undecidable, but Turing-recognizable, problem about strings rather than Turing machines.

Given two lists of “corresponding” strings, $(w_1, w_2, ..., w_n)$ and $(x_1, x_2, ..., x_n)$, does there exist a nonempty sequence of integers $i_1, i_2, ..., i_k$ such that $w_{i_1}w_{i_2}...w_{i_k} = x_{i_1}x_{i_2}...x_{i_k}$?

(For each $i$, a pair $(w_i, x_i)$ is said to be a corresponding pair.)
PCP as a game

Given a set of cards:
- N card types (can use as many copies of each type as needed)
- Each card has a top string and a bottom string

Example 1:

Puzzle:
- Is it possible to arrange cards so that the top and bottom strings are the same?

Solution 1:

<table>
<thead>
<tr>
<th></th>
<th>BAB</th>
<th>A</th>
<th>AB</th>
<th>BA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>ABA</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>ABA</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>ABA</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>ABA</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>

N = 4

Yes!

Another example

Given a set of cards:
- N card types (can use as many copies of each type as needed)
- Each card has a top string and a bottom string

Example 2:

Puzzle:
- Is it possible to arrange cards so that the top and bottom strings are the same?

Solution 2:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>ABA</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B</td>
<td>BAB</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>ABA</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>

N = 4

No. The first card in the solution must contain the same letter in the leftmost position.

PCP is undecidable

We can try all lists \(i_1, i_2, \ldots, i_k\) in order of \(k\). If we find a solution, the answer is “yes”. But if we never find a solution, how can we be sure no longer solution exists?

So we can never say “no”

Argument by Reduction

We can show (see textbook or Appendix):
- \(A_{TM}\) can be reduced to an intermediate problem, MPCP (where M stands for “modified”)
- MPCP can be reduced to PCP
- So, if PCP is decidable, so is \(A_{TM}\)
- But we know \(A_{TM}\) is undecidable

Therefore, PCP is undecidable
- or, if PCP were decidable, \(A_{TM}\) would also be decidable (but it isn’t!)
Reducing One Undecidable Problem to Another

We say a problem $A$ is **reduced** to problem $B$ if the decidability of $A$ follows from the decidability of $B$. Then, if we know $A$ is undecidable, we can conclude that $B$ is also undecidable.

The typical case is to use $A_{TM}$, which we know to be undecidable.

**Important point:** The proof that this problem is undecidable does not preclude solving it for specific cases; it only states that this cannot always be done. I.e., there is no algorithm that can make a correct decision for all TM–input pairs.

Undecidable Problems

- We have already established that $A_{TM}$ (does machine $M$ accept $w$?) is undecidable.

- Related problem (Halting Problem):
  
  $HALT_{TM} = \{ \langle M, w \rangle \mid M$ is a TM and $M$ halts (by accepting or rejecting) on input $w\}$

$HALT_{TM}$ is undecidable

**Proof by contradiction:**

- Assume $HALT_{TM}$ is decidable.
- Let $R$ be a Turing machine that decides $HALT_{TM}$.
- Use $R$ to construct $S$, a Turing machine that decides $A_{TM}$.
- Deciding $A_{TM}$ is reduced to $HALT_{TM}$.
- Since $A_{TM}$ is undecidable, so is $HALT_{TM}$.
• Assume \( R \) decides \( \text{HALT}^{TM} \)
• Define \( S \) as follows:
  – On input \( \langle M, w \rangle \),
    • Run \( R \) on \( \langle M, w \rangle \)
    • If \( R \) rejects (i.e., \( R \) says \( M \) didn’t halt), reject
    • If \( R \) accepts, simulate \( M \) on \( w \) until it halts
      • If \( M \) accepted, accept; otherwise reject

**Undecidability of TM Behaviors**

• Such a problem involves a language \( L \), which is a set of all Turing machines with a specific behavior \( B \)
  – E.g., \( L = \{ \langle M \rangle | M \text{ prints three 1s in a row on blank input} \} \) or \( L = \{ \langle M \rangle | M \text{ has a state which is never entered on any input string} \} \)
• For this class of problems, reductions typically require modifying the input Turing machine code \( \langle M \rangle \) to create code for a new Turing machine \( \langle M' \rangle \) which does pretty much the same thing as \( M \) except that
  – When \( M \) would have halted, \( M' \) first does behavior \( B \)
  – \( M' \) never accidentally does behavior \( B \) while it’s computing

A typical example of this type of reduction is the state entry problem, where \( M' \) mimics \( M \) until \( M \) halts, then goes to state \( q \); and it is ensured that \( M' \) can never go into state \( q \) unless \( M \) halts
The State Entry Problem

Given any TM $M = (Q, \Sigma, \Gamma, \delta, B, F)$ and any $q \in Q, w \in \Sigma^+$, decide whether or not the state $q$ is ever entered when $M$ is applied to $w$.

- This problem is **undecidable**

Step 1:
Reduce $\text{HALT}_{TM}$ to the State-entry Problem

- Assume an algorithm $A$ exists to solve the state-entry problem. We could then use it to solve $\text{HALT}_{TM}$.
- E.g., given any $M$ and $w$, first modify $M$ to get $M_1$ in such a way that $M_1$ halts in state $q$ if and only if $M$ halts.

**Question**: Can a Turing machine construct $M_1$ from $M$?

**Answer**: Yes, because we need only add an extra state and new transitions

**Process**

Do this by looking at $\delta$ of $M$:

- If $M$ halts, it does so because some $\delta(q', a)$ is undefined.
- To get $M_1$, we change every such undefined $\delta$ to $\delta(q', a) = (q, a, R)$ where $q$ is a final state

Now, apply the state-entry algorithm $A$ to $(M_1, q, w)$

If $A$ answers **yes** (i.e., state $q$ is entered), then $(M, w)$ halts.
If $A$ answers **no**, $(M, w)$ does not halt.

Visualization of the process for the state-entry problem

The assumption that the state-entry problem is decidable gives us an algorithm to decide the Halting Problem

**But because the Halting Problem is undecidable, the state-entry problem must also be undecidable**

Example reduction:
The Blank-tape Halting Problem
The Blank-tape Halting Problem

Another problem to which the Halting Problem can be reduced:
- Given a TM $M$, determine whether or not $M$ halts if started with a blank tape
- This problem is undecidable

How to do the reduction:
- Assume we are given some $M$ and some $w$
- Construct from $M$ a new machine $M_w$ that starts with a blank tape, writes $w$ on it, then positions itself in a configuration $(q_0, w)$
- After that, $M_w$ acts like $M$

Clearly, $M_w$ will halt on a blank tape if and only if $M$ halts on $w$

KEY:

Suppose the blank-tape halting problem is decidable
- Given any $(M, w)$, construct $M_w$
- Apply the blank-tape halting problem algorithm to it
- The conclusion tells us whether or not $M$ applied to $w$ will halt
- Since this can be done for any $M$ and any $w$, an algorithm for the blank-tape halting problem can be converted into an algorithm to solve the halting problem

But the Halting Problem is undecidable, so the blank-tape halting problem must also be undecidable

Visualization of the process for the blank-tape halting problem

Method:
Reduction by code rewriting
Reduction by Code Rewriting

- These problems require $S$ to rewrite the code for $M$ before feeding it to $R$
  - This can be viewed as creating a new TM $M_2$ that behaves quite differently from $M$
- The new Turing machine often has $w$ “hardcoded” into it
- The new TM is usually designed so that it accepts one set of strings if $M$ accepts $w$, and a totally different set of strings if $M$ doesn’t accept $w$
  - One of these sets should have the property that $R$ tests for and the other should not
    - For example, $M_2$ accepts all input strings if $M$ accepts $w$, and accepts no input strings if $M$ does not accept $w$

- The code for $M_2$ typically involves simulating $M$ on $w$, plus some other tests on $x$ that can clearly be computed in a predictable amount of time
- For some reductions, we test $x$ first, e.g.,
  - input is a string $x$
  - If $x$ has some easily tested property $P$, accept
  - Otherwise, simulate $M$ on $w$ and accept if $M$ accepts $w$
- For either of these two steps, you could also take the opposite action, e.g., reject all strings $x$ that have property $P$

Examples of this pattern are $\text{REGULAR}_{\text{TM}}$ and $E_{\text{TM}}$

Example reduction: Does a TM accept a regular language?

$\text{REGULAR}_{\text{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$

Theorem: $\text{REGULAR}_{\text{TM}}$ is undecidable.

Proof by contradiction:
- Assume $\text{REGULAR}_{\text{TM}}$ is decidable
- Let $R$ be a TM that decides $\text{REGULAR}_{\text{TM}}$
- Use $R$ to construct $S$, a TM that decides $A_{\text{TM}}$

… but how?
• Modify $M$ so that the resulting machine $M_2$ accepts a regular language if and only if $M$ accepts $w$.

• Design $M_2$ so that
  – If $M$ does not accept $w$, then $M_2$ will accept $\{0^n1^n \mid n \geq 0\}$, which is not regular.
  – If $M$ accepts $w$, then $M_2$ accepts $\Sigma^*$, which is regular.

From $S$'s input $<M, w>$, we define $M_2$ that takes input $x$:

1. If $x$ is of form $0^n1^n$, $M_2$ accepts.
2. Otherwise, run $M$ on $w$ and
   1. If $M$ accepts $w$, then $M_2$ accepts any input $x$
   2. Otherwise, $M_2$ rejects $x$ either by halting or by running $M$ forever

✓ If $M$ accepts $w$, then $M_2$ is a TM that accepts $\Sigma^*$ (i.e., any string)
✓ If $M$ does not accept $w$, then $M_2$ is a TM that only accepts $\{0^n1^n \mid n \geq 0\}$

Now, feed $M_2$ to $R$
  – If $R$ accepts, then $S$ accepts
  – Otherwise, $S$ rejects

If $\text{REGULAR}_{\text{TM}}$ is decidable, we can use it to solve $\text{ATM}$
But we know $\text{ATM}$ is undecidable, and therefore $\text{REGULAR}_{\text{TM}}$ must be undecidable also

What is the language of $M_2$?

It depends on whether $M$ accepts $w$!
Reductions
Basic Template

- To prove that a language $L$ is undecidable, you will normally want to use a reduction from a language already known to be undecidable (e.g. $A_{TM}$)
- A reduction proof will look something like this:
  - Suppose that $L$ were decidable. Let $R$ be a Turing machine deciding $L$. We will now construct a Turing machine $S$ that decides $A_{TM}$.
  - $S$ is constructed as follows:
    - Input is $<M,w>$, where $M$ is the code for a Turing Machine and $w$ is a string.
    - [Pseudocode explaining how $S$ decides whether $M$ accepts $w$, using $R$ as a subroutine.]
  - But we know that $A_{TM}$ is undecidable. So $S$ can’t exist. Therefore we have a contradiction. So $L$ must have been undecidable.
- Sometimes you can write a shorter/simpler proof by replacing $A_{TM}$ with some other problem known to be undecidable
  - you may need to change the inputs to $S$
    - e.g. elements of the language $EQ_{TM}$ are pairs of Turing machines; a reduction from $EQ_{TM}$ requires constructing a Turing machine $S$ whose inputs are pairs $<M_1,M_2>$ of two Turing machines.
- If you aren’t sure what problem to reduce to $L$, $A_{TM}$ is normally a good choice

Using Reduction to Show Virus Detection is Undecidable

Virus Detection
Melissa Virus
Input: A Word macro (like a program, but embedded in an email message)
Output: true if the macro will forward the message to people in your address book; false otherwise.

How can we show it is undecidable?

Undecidability Proof

Suppose we could define $is\text{-}virus?$ that decides the Melissa problem.

\[
\begin{align*}
\text{(define (halts? P)} \\
\text{ (if (is-virus? '(begin P virus-code))} \\
\text{ \texttt{#t}} \\
\text{ \texttt{#f})})
\end{align*}
\]

Since it is a virus, we know virus-code was evaluated, and $P$ must halt (assuming $P$ wasn’t a virus).

It’s not a virus, so the virus-code never executed. Hence, $P$ must not halt.
Undecidability Proof

Suppose we could define is-virus? that decides the Melissa problem.

(define (halts? P)
  (is-virus? '(begin (vaccinate P)
                   virus-code))

Where (vaccinate P) evaluates to P with all mail commands replaced with print commands (to make sure (is-virus? P) is false.

Proof

• If we had is-virus? we could define halts?
• We know halts? is undecidable
• Hence, we can’t have is-virus?
• Thus, we know is-virus? is undecidable

PCP problem as dominoes

• Begin with a collection of dominoes, each containing two strings, one on each side:
  \[
  \{[[b], [a], [ca], [abc]], [[ca], [ab], [a], [abc]]\}
  \]

• Task: make a list of the dominoes (repetitions permitted) so that the string we get reading across the top is the same as the one across the bottom (call this a solution)
  \[
  \begin{array}{c}
  a \\
  \text{[ab]} \\
  a \\
  c \\
  \text{[abc]} \\
  \end{array}
  \]

• PCP is to determine whether a collection of dominoes has a solution.

Example

• (1, 0, 010, 11) and (10, 10, 01, 1).
• A solution is 1, 2, 1, 3, 3, 4.
• The constructed string from both lists is 10101001011.
  \[
  \begin{array}{cccccccc}
  1 & 0 & 0 & 1 & 0 & 1 & 1 \\
  10 & 10 & 01 & 01 & 1 & 1 \\
  \end{array}
  \]
Another Example

• From the book: (10, 011, 101) and (101, 11, 011).
  – Another argument why this instance of PCP has no solution:
    • The first index has to be 1 because only the pair 10 and 101 begin with the same symbol.
    • Then, whatever indexes we choose to continue, there will be more 1s in the string constructed from the first list than the second (because in each corresponding pair there are at least as many 1s in the second list). Thus, the two strings cannot be equal.

Plan to Show PCP is Undecidable

1. Introduce MPCP, where the first pair must be taken first in a solution.
2. Show how to reduce MPCP to PCP.
3. Show how to reduce $L_u$ to MPCP.
   – This is the only reason for MPCP: it makes the reduction from $L_u$ easier.
4. Conclude that if PCP is decidable, so is MPCP, and so is $L_u$ (which we know is false).

Reduction of MPCP to PCP

Trick:
• given an MPCP instance, introduce a new symbol *.
• In the first list, * appears after every symbol, but in the second list, the * appears before every symbol.

Example:
• the pair 10 and 011 becomes 1*0* and *0*1*1.
• Notice that no such pair can ever be the first in a solution.

• Take the first pair $w_1$ and $x_1$ from the MPCP instance (which must be chosen first in a MPCP solution) and add to the PCP instance another pair in which the “*”s are as always, but $w_1$ also gets an extra * at the beginning.
  – Referred to as “pair 0”.
  – Example: if 10 and 011 is the first pair, also add to the PCP instance the pair *1*0* and *0*1*1.
• Finally, since the strings from the first list will have an extra * at the end, add to the PCP instance the pair $\$*$ and *$.
  – $\$ is a new symbol, so this pair can be used only to complete a match.
  – Referred to as the "final pair."
### Proof the Reduction is Correct

- If the MPCP instance has a solution $i_1, i_2, \ldots, i_k$ followed by $i_1$, then the PCP instance has a solution, which is the same, using the first pair in place of pair 1, and terminating the list with the final pair.
- If the PCP instance has a solution, then it must begin with the “first pair”, because no other pair begins with the same symbol.
- Thus, removing the “*”s and deleting the last pair gives a solution to the MPCP instance.

### Reduction of $L_u$ to MPCP

- Intuition: The equal strings represent a computation of a TM $M$ on input $w$.
  - Sequence of IDs separated by a special marker #.
- First pair is # and $q_0 w$.
- String from first list is always one ID behind, unless an accepting state is reached, in which case the first string can “catch up”.

### Some Example Pairs

1. $X$ and $X$ for every tape symbol $X$. Allows copying of symbols that don’t change from one ID to the next.
2. If $\delta(q, X) = (p, Y, R)$, then $qX$ and $Yp$ is a pair. Simulates a move for the next ID.
3. If $q$ is an accepting state, then $XqY$ and $q$ is a pair for all $X$ and $Y$. Allows ID to “shrink to nothing” when an accepting state is reached.

### Constructing MPCP $P'$

1. Put $(\# , \# q_0 w_1 w_2 \ldots w_n \#)$ as the first pair.
2. For every $a, b \in \Gamma$ and every $q, r \in Q$, if $\delta(q, a) = (r, b, R)$ put $(qa, br)$ into $P'$.
3. For every $a, b, c \in \Gamma$ and every $q, r \in Q$, if $\delta(q, a) = (r, b, L)$ put $(cqa, rcb)$ into $P'$.
4. For every $a \in \Gamma$ put $(a, a)$ into $P'$.
5. Put $(\#, \#)$ (to separate configurations) and $(\#, B\#)$ (to simulate the infinite blanks to the right that are suppressed when the configuration is written) into $P'$.
What we have so far

- Suppose $\Gamma = \{0,1,2,B\}$, $w = 0100$, start state is $q_0$.
  $\delta(q_0,0) = (q_7,2,R)$

  \begin{itemize}
  \item \text{STEP 1:}
  \begin{verbatim}
  \top
  #
  q_0 0 1 0 0 #

  \bottom
  \end{verbatim}
  \item \text{STEP 2 adds $(q_0,0,2,q_7)$}
  \item \text{STEP 4 adds $(0,0), (1,1), (2,2), (B, B)$}
  \end{itemize}

- Continues until $M$ reaches a halting state – if also an accept state, let the top of the partial match “catch up”

- Add two more items to $P'$:
  - 6. For every $a \in \Gamma$, put $(aq_A, q_A)$ and $(q_Aa, q_A)$ into $P'$, where $q_A \in F$.

  \begin{itemize}
  \item \text{7. Put $(q_A\#\#, \#)$ into $P'$}
  \end{itemize}

- We have thus shown that $L_u$ can be reduced to MPCP.

- Last step: show that $M$ accepts $w$ if and only if the constructed MPCP instance has a solution.
  - \text{Only-if}: we have shown that the computation can be mimicked, providing the solution.
  - \text{If}: the MPCP will only provide a solution if we reach an accepting state, so that the two strings are of equal lengths. Otherwise, we never perform step 7, which enables the strings to become equal in length.