CS:4330 Theory of Computation

Spring 2018

Computability Theory

Reducibility

Haniel Barbosa

The University of Iowa

Readings for this lecture

Chapter 5 of [Sipser 1996], 3rd edition. Section 5.1.

Reduction: Examples

- ▷ Suppose that one wants to find their way in a city. This would be easy if one had a city map. Hence, the previous problem is reduced to the problem of finding a city map.

- ▷ Problem A: measuring the area of a circle, reduces to problem B: measuring r, the circle radius, which reduces to problem C: performing πr^2

- ▷ Problem A: solving a system of linear equations, reduces to problem B: triangulating a matrix

- ▷ Problem A: proving a set is uncountable, reduces to problem B: establishing a correspondence between the set and the set of reals.

Observations

- ▷ Reduction is a terminating process.
- ▷ When problem A is reduced to problem B, solving A cannot be harder than the sum of reduction and solving B, because a solution to B gives a solution to A.
- ▷ If A is reduced to B and B is decidable, then A is decidable.
- ▷ If A is undecidable and reducible to B then B is also undecidable.

Decidable problems: Methodology

Fer proving that a problem Q is decidable by reduction method, proceeding as follows:

1. Find a problem P known to be decidable
2. Assume that P is decided by a TM M_P
3. Use the TM M_P to construct a TM M_Q that solves Q:
   (a) Encode every instance q of the problem Q as an instance q_P of problem P
   (b) Use M_P to solve q_P and return the result

"Decidable" can be replaced by "Turing-recognizable" to show some problems are Turing-recognizable.

Undecidable problems: Methodology

A common strategy for proving that a problem P is undecidable is by reduction method, proceeding as follows:

1. Find a problem Q known to be undecidable
2. Assume that P is decided by a TM M_P
3. Use the TM M_P to construct a TM M_Q that solves Q:
   (a) Encode every instance q of the problem Q as an instance q_P of problem P
   (b) Use M_P to solve q_P
4. Since it is known that Q is undecidable, M_Q cannot exist. Hence, M_P cannot exist either and P is undecidable.

"Undecidable" can be replaced by "not Turing-recognizable" to show some problems are not Turing-recognizable.

Emptiness problem for TM

Theorem

The language $E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } \mathscr{L}(M) = \varnothing\}$ *is undecidable.*

Proof idea: reduction from A_{TM} to E_{TM}

- ▷ Assume that E_{TM} is decidable and let R be its TM decider.
- ▷ Show that a TM S can be constructed using R that decides A_{TM}

Emptiness problem for TM

Theorem

The language $E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } \mathscr{L}(M) = \varnothing\}$ *is undecidable.*

Proof idea: reduction from A_{TM} to E_{TM}

- ▷ Assume that E_{TM} is decidable and let R be its TM decider.
- ▷ Show that a TM S can be constructed using R that decides A_{TM}
- ▷ *Bad idea*: Run R on $\langle M \rangle$. If it accepts then $\mathscr{L}(M) = \varnothing$, so M does not accept w; otherwise, $\mathscr{L}(M) \neq \varnothing$, which does not entail that M accepts w

Emptiness problem for TM

Theorem

The language $E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } \mathscr{L}(M) = \varnothing\}$ *is undecidable.*

Proof idea: reduction from A_{TM} to E_{TM}

- ▷ Assume that E_{TM} is decidable and let R be its TM decider.
- ▷ Show that a TM S can be constructed using R that decides A_{TM}
- ▷ *Bad idea*: Run R on $\langle M \rangle$. If it accepts then $\mathscr{L}(M) = \varnothing$, so M does not accept w; otherwise, $\mathscr{L}(M) \neq \varnothing$, which does not entail that M accepts w
- ▷ *Good idea*: run R on a modification $\langle M_1 \rangle$ of $\langle M \rangle$ that guarantees that M_1 rejects all strings except w. That is

$$\mathscr{L}(M_1) = \begin{cases} \{w\} & \text{if } w \in \mathscr{L}(M) \\ \varnothing & \text{otherwise} \end{cases}$$

R can test then if $\mathscr{L}(M_1) = \varnothing$ to determine whether $w \in \mathscr{L}(M)$.

Proving E_{TM} is undecidable

The modified machine M_1

$M_1 =$"On input x:

1. If $x \neq w$, *reject*
2. If $x = w$, run M on input w and if M accepts, *accept*."

▷ Note that M_1 has w "hardcoded" as part of its description.

Proving E_{TM} is undecidable

The modified machine M_1

M_1 ="On input x:

1. If $x \neq w$, *reject*
2. If $x = w$, run M on input w and if M accepts, *accept*."

▷ Note that M_1 has w "hardcoded" as part of its description.

The machine S

S ="On input $\langle M, w \rangle$, in which M is a TM and w a string:

1. Construct M_1 from $\langle M, w \rangle$
2. Run R on input $\langle M_1 \rangle$
3. If R accepts, *reject*; if R rejects, *accept*."

▷ If E_{TM} was decidable then A_{TM} would be decidable. Since A_{TM} is undecidable, so is E_{TM}.

TM and regular languages

- Can a TM recognize a language recognized by a simpler computational model, such as a regular language?

- For example, $REGULAR_{\text{TM}}$ is the problem of testing whether a given TM has an equivalent finite automaton.

- This is the same as testing whether a TM recognizes a regular language:

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

TM and regular languages

Theorem

$REGULAR_{\text{TM}}$ is undecidable.

Proof idea: reduction to decidability of A_{TM}

Let R be a decider for $REGULAR_{\text{TM}}$. Build decider S for A_{TM} which leverages R.

TM and regular languages

Theorem

$REGULAR_{\text{TM}}$ is undecidable.

Proof idea: reduction to decidability of A_{TM}

Let R be a decider for $REGULAR_{\text{TM}}$. Build decider S for A_{TM} which leverages R.

- ▷ On input $\langle M, w \rangle$, S modifies M so that the resulting TM M' recognizes a regular language if and only if M accepts w:

TM and regular languages

Theorem

$REGULAR_{\text{TM}}$ is undecidable.

Proof idea: reduction to decidability of A_{TM}

Let R be a decider for $REGULAR_{\text{TM}}$. Build decider S for A_{TM} which leverages R.

- ▷ On input $\langle M, w \rangle$, S modifies M so that the resulting TM M' recognizes a regular language if and only if M accepts w:
  - M' recognizes the non-regular language $\{0^n 1^n \mid n \geq 0\}$ (chosen arbitrarily) if M does not accept w
  - M' recognizes the regular language Σ^* (chosen arbitrarily) if M accepts w.

TM and regular languages

A decider S for A_{TM} based on a decider R for $REGULAR_{\text{TM}}$ is as follows:

$S =$"On input $\langle M, w \rangle$ in which a M is a TM and w a string:

1. Build the folowing TM M':
   $M' =$"On input x:
   1. If x has the form $0^n 1^n$, *accept*
   2. Otherwise, run M on w and if M accepts w, *accept*."
2. Run R on $\langle M' \rangle$.
3. If R accepts, *accept*; if R rejects, *reject*."

TM and regular languages

A decider S for A_{TM} based on a decider R for $REGULAR_{\text{TM}}$ is as follows:

$S =$"On input $\langle M, w \rangle$ in which a M is a TM and w a string:

1. Build the folowing TM M':
   $M' =$"On input x:
   1. If x has the form $0^n 1^n$, *accept*
   2. Otherwise, run M on w and if M accepts w, *accept*."
2. Run R on $\langle M' \rangle$.
3. If R accepts, *accept*; if R rejects, *reject*."

Since A_{TM} is undecidable, R cannot exist, therefore $REGULAR_{\text{TM}}$ is undecidable.

TM and regular languages

A decider S for A_{TM} based on a decider R for $REGULAR_{\text{TM}}$ is as follows:

$S =$"On input $\langle M, w \rangle$ in which a M is a TM and w a string:

1. Build the folowing TM M':
   $M' =$"On input x:
   1. If x has the form $0^n 1^n$, *accept*
   2. Otherwise, run M on w and if M accepts w, *accept*."
2. Run R on $\langle M' \rangle$.
3. If R accepts, *accept*; if R rejects, *reject*."

Since A_{TM} is undecidable, R cannot exist, therefore $REGULAR_{\text{TM}}$ is undecidable.

Moreover, the same holds for CFL_{TM}, $DECIDABLE_{\text{TM}}$, etc.

Rice's Theorem

Determining any property of the languages recognized by Turing machines is undecidable.

Theorem

Let P be the language of TMs descriptions where P fulfills the following conditions:

1. *P is nontrivial, i.e. it contains some, but not all, TM descriptions. Formally: there exist TMs M_1 and M_2 s.t. $\langle M_1 \rangle \in P$ and $\langle M_2 \rangle \notin P$.*
2. *P is a property of the TMs language. Formally: for any TMs M_1 and M_2, if $\mathscr{L}(M_1) = \mathscr{L}(M_2)$ then $\langle M_1 \rangle \in P$ iff $\langle M_2 \rangle \in P$, i.e. membership of a TM M in P depends only on the language of M.*

P is undecidable.

Other reductions

Sometimes reducing from other undecidable languages other than A_{TM}, such as E_{TM}, may be more convenient:

Theorem

$EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } \mathscr{L}(M_1) = \mathscr{L}(M_2)\}$ *is undecidable.*

Proof idea: reduction from E_{TM} to EQ_{TM}

Other reductions

Sometimes reducing from other undecidable languages other than A_{TM}, such as E_{TM}, may be more convenient:

Theorem

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

Proof idea: reduction from E_{TM} to EQ_{TM}

Let R be the decider of EQ_{TM}. A decider S E_{TM} based on R is as follows:
$S =$"On input $\langle M \rangle$ in which a M is a TM:

1. Run R on input $\langle M, M' \rangle$, in which M' is a TM that rejects all inputs
2. If R accepts, *accept*; if R rejects, *reject*."

Since E_{TM} is undecidable, so must be EQ_{TM}.