T065001: Introduction to Formal Languages

Lecture 12: Reducibility (1)

Chapter 5.1 in Sipser's textbook

2025-07-07

(Lecture slides by Yih-Kuen Tsay)

Introduction

- A *reduction* is a way of converting one problem into another problem in such a way that a solution to the second problem can be used to solve the first problem.
- If a problem A reduces (is reducible) to another problem B, we can use a solution to B to solve A.

Introduction

- A *reduction* is a way of converting one problem into another problem in such a way that a solution to the second problem can be used to solve the first problem.
- If a problem A reduces (is reducible) to another problem B, we can use a solution to B to solve A.
- *Reducibility* says nothing about solving A or B alone, but only about the solvability of A in the presence of a solution to B.
- Reducibility is the primary method for proving that problems are computationally unsolvable.

Introduction

- A *reduction* is a way of converting one problem into another problem in such a way that a solution to the second problem can be used to solve the first problem.
- If a problem A reduces (is reducible) to another problem B, we can use a solution to B to solve A.
- *Reducibility* says nothing about solving A or B alone, but only about the solvability of A in the presence of a solution to B.
- Reducibility is the primary method for proving that problems are computationally unsolvable.
- Suppose that A is reducible to B. If B is decidable, then A is decidable; equivalently, if A is undecidable, then B is undecidable.

The Acceptance Problem

From the previous lecture:

- $A_{\mathrm{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$.

Theorem (4.11)

A_{TM} is undecidable.

The Acceptance Problem

From the previous lecture:

- $A_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$.

Theorem (4.11)

A_{TM} is undecidable.

Now define:

- $HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on } w\}$.

The Acceptance Problem

From the previous lecture:

- $A_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$.

Theorem (4.11)

A_{TM} is undecidable.

Now define:

- $HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on } w\}$.

Note the difference in the definitions of A_{TM} and $HALT_{\mathsf{TM}}$:

- If M accepts w then $\langle M, w\rangle \in A_{\mathsf{TM}}$ and $\langle M, w\rangle \in HALT_{\mathsf{TM}}$.
- If M rejects w then $\langle M, w\rangle \notin A_{\mathsf{TM}}$ but $\langle M, w\rangle \in HALT_{\mathsf{TM}}$.
- If M **loops** on w then $\langle M, w\rangle \notin A_{\mathsf{TM}}$ and $\langle M, w\rangle \notin HALT_{\mathsf{TM}}$.

The Halting Problem

- $A_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$.
- $HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on } w\}$.

The Halting Problem

- $A_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$.
- $HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on } w\}$.

Theorem (5.1)

$HALT_{\mathrm{TM}}$ is undecidable.

- The idea is to reduce the acceptance problem A_{TM} (known to be undecidable) to $HALT_{\mathrm{TM}}$.
- Assume toward a contradiction that a TM R decides $HALT_{\mathrm{TM}}$.
- We could then construct a decider S for A_{TM} as follows.

(We need to explain how S will work when it receives the input $\langle M, w\rangle$.)

The Halting Problem (cont.)

$S =$ "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w \rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; it M has rejected, *reject*."

The Halting Problem (cont.)

$S =$ "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w \rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; it M has rejected, *reject*."

The TM S defined above decides A_{TM}.
But this contradicts Theorem 4.11.
Thus, R cannot exist, so $HALT_{\mathrm{TM}}$ is undecidable.

More undecidable languages related to TMs

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

More undecidable languages related to TMs

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

To prove that E_{TM} is undecidable, we'll reduce from A_{TM} again.

Theorem (5.2)

E_{TM} is undecidable.

- Assuming that a TM R decides E_{TM}, we construct a decider S for A_{TM} as follows.

(We need to explain how S will work when it receives the input $\langle M, w\rangle$.)

More undecidable languages related to TMs (cont.)

One idea might be to let S run R on input $\langle M \rangle$:

- If R accepts $\langle M \rangle$, then S knows that M does not accept any strings at all, and hence that M rejects w.
- Unfortunately, this idea doesn't work because in case R rejects $\langle M \rangle$ then S only knows that M *accepts at least one string*...
  S still doesn't know whether M accepts the particular string w or not.

More undecidable languages related to TMs (cont.)

One idea might be to let S run R on input $\langle M \rangle$:

- If R accepts $\langle M \rangle$, then S knows that M does not accept any strings at all, and hence that M rejects w.
- Unfortunately, this idea doesn't work because in case R rejects $\langle M \rangle$ then S only knows that M *accepts at least one string*...
  S still doesn't know whether M accepts the particular string w or not.

Better idea: Let S run R on a modified version of $\langle M \rangle$ instead.

Define the modified machine, called M_1, so that it checks if its input is the string w, and if so, behaves exactly like M on w, but it rejects all other strings automatically.

Then, we can let S can run R on $\langle M_1 \rangle$.

See the next slide for details...

More undecidable languages related to TMs (cont.)

$S =$ "On input $\langle M, w \rangle$:

1. Construct the following TM M_1.
   $M_1 =$ "On input x:
   1.1 If $x \neq w$, *reject*.
   1.2 If $x = w$, run M on input w and *accept* if M accepts w."
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, *reject*; if R rejects, *accept*."

More undecidable languages related to TMs (cont.)

$S =$ "On input $\langle M, w \rangle$:

1. Construct the following TM M_1.
   $M_1 =$ "On input x:
   1.1 If $x \neq w$, *reject*.
   1.2 If $x = w$, run M on input w and *accept* if M accepts w."
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, *reject*; if R rejects, *accept*."

Then we get: $L(M_1) = \begin{cases} \{w\}, & \text{if } M \text{ accepts } w \\ \emptyset, & \text{if } M \text{ rejects } w \\ \emptyset, & \text{if } M \text{ loops on input } w \end{cases}$

More undecidable languages related to TMs (cont.)

$S =$ "On input $\langle M, w \rangle$:

1. Construct the following TM M_1.
   $M_1 =$ "On input x:
   1.1 If $x \neq w$, *reject*.
   1.2 If $x = w$, run M on input w and *accept* if M accepts w."
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, *reject*; if R rejects, *accept*."

Then we get: $L(M_1) = \begin{cases} \{w\}, & \text{if } M \text{ accepts } w \\ \emptyset, & \text{if } M \text{ rejects } w \\ \emptyset, & \text{if } M \text{ loops on input } w \end{cases}$

so applying R to distinguish between the case $L(M_1) \neq \emptyset$ and the case $L(M_1) = \emptyset$ allows us to determine whether or not M accepts w.

More undecidable languages related to TMs (cont.)

$S =$ "On input $\langle M, w \rangle$:

1. Construct the following TM M_1.
   $M_1 =$ "On input x:
   1.1 If $x \neq w$, *reject*.
   1.2 If $x = w$, run M on input w and *accept* if M accepts w."
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, *reject*; if R rejects, *accept*."

Then we get: $L(M_1) = \begin{cases} \{w\}, & \text{if } M \text{ accepts } w \\ \emptyset, & \text{if } M \text{ rejects } w \\ \emptyset, & \text{if } M \text{ loops on input } w \end{cases}$

so applying R to distinguish between the case $L(M_1) \neq \emptyset$ and the case $L(M_1) = \emptyset$ allows us to determine whether or not M accepts w.

In conclusion, the TM S defined above decides A_{TM}. However, this contradicts Theorem 4.11. Thus, R cannot exist, so E_{TM} is undecidable.

More undecidable languages related to TMs (cont.)

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

More undecidable languages related to TMs (cont.)

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

Below, we give a reduction from A_{TM} to $\mathit{REGULAR}_{\mathsf{TM}}$.

Theorem (5.3)

$\mathit{REGULAR}_{\mathrm{TM}}$ is undecidable.

- Assuming that a TM R decides $\mathit{REGULAR}_{\mathrm{TM}}$, we construct a decider S for A_{TM} as follows.

(We need to explain how S will work when it receives the input $\langle M, w \rangle$.)

More undecidable languages related to TMs (cont.)

$S =$ "On input $\langle M, w \rangle$:

1. Construct the following TM M_2.
   $M_2 =$ "On input x:
   1.1 If x has the form $0^n 1^n$, *accept*.
   1.2 If x does not have this form, run M on input w and *accept* if M accepts w."
2. Run R on input $\langle M_2 \rangle$.
3. If R accepts, *accept*; if R rejects, *reject*."

More undecidable languages related to TMs (cont.)

$S =$ "On input $\langle M, w \rangle$:

1. Construct the following TM M_2.
   $M_2 =$ "On input x:
   1.1 If x has the form $0^n 1^n$, *accept*.
   1.2 If x does not have this form, run M on input w and *accept* if M accepts w."
2. Run R on input $\langle M_2 \rangle$.
3. If R accepts, *accept*; if R rejects, *reject*."

Note: if M does not accept w, then $L(M_2) = \{0^n 1^n \mid n \geq 0\}$, which is not regular; if M accepts w, then $L(M_2) = \{0, 1\}^*$, which is regular.

More undecidable languages related to TMs (cont.)

$S =$ "On input $\langle M, w \rangle$:

1. Construct the following TM M_2.
   $M_2 =$ "On input x:
   1.1 If x has the form $0^n 1^n$, *accept*.
   1.2 If x does not have this form, run M on input w and *accept* if M accepts w."
2. Run R on input $\langle M_2 \rangle$.
3. If R accepts, *accept*; if R rejects, *reject*."

Note: if M does not accept w, then $L(M_2) = \{0^n 1^n \mid n \geq 0\}$, which is not regular; if M accepts w, then $L(M_2) = \{0, 1\}^*$, which is regular.

This means that the TM S defined above decides A_{TM}, contradicting Theorem 4.11. Thus, R cannot exist, so $REGULAR_{\mathsf{TM}}$ is undecidable.

More undecidable languages related to TMs (cont.)

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

More undecidable languages related to TMs (cont.)

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

This time, we'll give a reduction from E_{TM} (and not A_{TM} as before!).
Recall the definition of E_{TM} from earlier today:

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

More undecidable languages related to TMs (cont.)

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

This time, we'll give a reduction from E_{TM} (and not A_{TM} as before!). Recall the definition of E_{TM} from earlier today:

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

Theorem (5.4)

EQ_{TM} is undecidable.

- Assume that a TM R decides EQ_{TM}.
- We construct a decider S for E_{TM} as follows.

(We need to explain how S will work when it receives the input $\langle M\rangle$.)

More undecidable languages related to TMs (cont.)

- $S =$ "On input $\langle M \rangle$:
  1. Run R on input $\langle M, M_1 \rangle$, where M_1 is a TM that rejects all inputs.
  2. If R accepts, *accept*; if R rejects, *reject*."

More undecidable languages related to TMs (cont.)

- $S =$ "On input $\langle M \rangle$:
  1. Run R on input $\langle M, M_1 \rangle$, where M_1 is a TM that rejects all inputs.
  2. If R accepts, *accept*; if R rejects, *reject*."

Here, the TM S uses R to check if $L(M) = L(M_1)$, where $L(M_1) = \emptyset$.

More undecidable languages related to TMs (cont.)

- $S =$ "On input $\langle M \rangle$:
  1. Run R on input $\langle M, M_1 \rangle$, where M_1 is a TM that rejects all inputs.
  2. If R accepts, *accept*; if R rejects, *reject*."

Here, the TM S uses R to check if $L(M) = L(M_1)$, where $L(M_1) = \emptyset$. In other words, S decides E_{TM}. However, this contradicts Theorem 5.2. Thus, R cannot exist, so EQ_{TM} is undecidable.

Undecidable languages related to CFGs?

In the previous lecture and weekly exercises, we showed that many languages related to DFAs are decidable:

- A_{DFA}, A_{NFA}, A_{REX}, E_{DFA}, EQ_{DFA}, $EQ_{\text{DFA,REX}}$, ALL_{DFA}, etc.

Undecidable languages related to CFGs?

In the previous lecture and weekly exercises, we showed that many languages related to DFAs are decidable:

- A_{DFA}, A_{NFA}, A_{REX}, E_{DFA}, EQ_{DFA}, $EQ_{\text{DFA,REX}}$, ALL_{DFA}, etc.

In contrast, many languages related to TMs are undecidable:

- A_{TM}, $HALT_{\text{TM}}$, E_{TM}, $REGULAR_{\text{TM}}$, EQ_{TM}

Undecidable languages related to CFGs?

In the previous lecture and weekly exercises, we showed that many languages related to DFAs are decidable:

- A_{DFA}, A_{NFA}, A_{REX}, E_{DFA}, EQ_{DFA}, $EQ_{\mathsf{DFA,REX}}$, ALL_{DFA}, etc.

In contrast, many languages related to TMs are undecidable:

- A_{TM}, $HALT_{\mathsf{TM}}$, E_{TM}, $REGULAR_{\mathsf{TM}}$, EQ_{TM}

What about languages related to CFGs? We already showed that the following are decidable:

- A_{CFG}, E_{CFG}

Undecidable languages related to CFGs?

In the previous lecture and weekly exercises, we showed that many languages related to DFAs are decidable:

- A_{DFA}, A_{NFA}, A_{REX}, E_{DFA}, EQ_{DFA}, $EQ_{\mathrm{DFA,REX}}$, ALL_{DFA}, etc.

In contrast, many languages related to TMs are undecidable:

- A_{TM}, $HALT_{\mathrm{TM}}$, E_{TM}, $REGULAR_{\mathrm{TM}}$, EQ_{TM}

What about languages related to CFGs? We already showed that the following are decidable:

- A_{CFG}, E_{CFG}

and we will now give an example of an undecidable language for CFGs:

- ALL_{CFG}

(One of this week's exercises is to prove that EQ_{CFG} is undecidable, too.)

Undecidable languages related to CFGs

- $ALL_{\mathrm{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$.

Theorem (5.13)

ALL_{CFG} is undecidable.

(From Chapter 3.1)

- As a TM computes, changes occur in
  1. the current state,
  2. the current tape contents, and
  3. the current head location.
- A setting of these three items is called a **configuration** of the TM.
- We write uqv to denote the configuration where
  1. the current state is q,
  2. the current tape contents is uv, and
  3. the current head location is the first symbol of v.

  (The tape contains only blanks following the last symbol of v.)

(From Chapter 3.1)

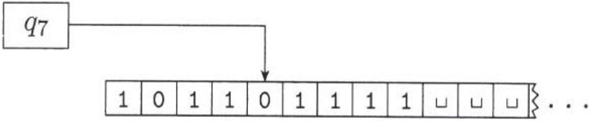

FIGURE 3.4
A Turing machine with configuration $1011q_701111$

Computation Histories

Definition (5.5)

An *accepting computation history* for M on w is a sequence of configurations $C_1, C_2, \cdots, C_l$, where

1. C_1 is the start configuration,
2. C_l is an accepting configuration, and
3. C_i yields C_{i+1}, $1 \leq i \leq l - 1$.

Computation Histories

Definition (5.5)

An *accepting computation history* for M on w is a sequence of configurations $C_1, C_2, \cdots, C_l$, where

1. C_1 is the start configuration,
2. C_l is an accepting configuration, and
3. C_i yields C_{i+1}, $1 \leq i \leq l-1$.

Note that any accepting computation history is a *finite sequence*.
$\Rightarrow$ We can represent accepting computation histories by *strings*.

Undecidable languages related to CFGs

- $ALL_{\mathrm{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$.

Theorem (5.13)

ALL_{CFG} is undecidable.

- For a TM M and an input w, we construct a CFG G (by first constructing a PDA) to generate those strings that don't represent accepting computation histories for M on w.

Undecidable languages related to CFGs

- $ALL_{\mathrm{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$.

Theorem (5.13)

ALL_{CFG} is undecidable.

- For a TM M and an input w, we construct a CFG G (by first constructing a PDA) to generate those strings that don't represent accepting computation histories for M on w.

There is some string that G won't generate if and only if M accepts w.

- That is, G generates all strings if and only if M does not accept w.
- If ALL_{CFG} were decidable, then A_{TM} would be decidable.
  But this would contradict Theorem 4.11, so ALL_{CFG} is undecidable.

Undecidable languages related to CFGs (cont.)

PDA for recognizing strings that aren't accepting computation histories:

- The input is regarded as a computation history of the form:

  $$\#C_1\#C_2^R\#C_3\#C_4^R\#\cdots\#C_l\#$$

  where C_i^R denotes the reverse of C_i.

  If it doesn't start and end with $\#$, the PDA accepts it immediately.

Undecidable languages related to CFGs (cont.)

$$\# \underbrace{\longrightarrow}_{C_1} \# \underbrace{\longleftarrow}_{C_2^{\mathcal{R}}} \# \underbrace{\longrightarrow}_{C_3} \# \underbrace{\longleftarrow}_{C_4^{\mathcal{R}}} \# \quad \cdots \quad \# \underbrace{\qquad\quad}_{C_l} \#$$

FIGURE **5.14**
Every other configuration written in reverse order

Undecidable languages related to CFGs (cont.)

PDA for recognizing strings that aren't accepting computation histories:

- The input is regarded as a computation history of the form:

  $$\#C_1\#C_2^R\#C_3\#C_4^R\#\cdots\#C_l\#$$

  where C_i^R denotes the reverse of C_i.

  If it doesn't start and end with $\#$, the PDA accepts it immediately.

- The PDA nondeterministically chooses to check if one of the following conditions holds for the input:
  - C_1 is not the start configuration.
  - C_l is not an accepting configuration.
  - C_i does not yield C_{i+1}, for some i, $1 \leq i < l$.

Undecidable languages related to CFGs (cont.)

PDA for recognizing strings that aren't accepting computation histories:

- The input is regarded as a computation history of the form:

  $$\#C_1\#C_2^R\#C_3\#C_4^R\#\cdots\#C_l\#$$

  where C_i^R denotes the reverse of C_i.

  If it doesn't start and end with $\#$, the PDA accepts it immediately.

- The PDA nondeterministically chooses to check if one of the following conditions holds for the input:
  - C_1 is not the start configuration.
  - C_l is not an accepting configuration.
  - C_i does not yield C_{i+1}, for some i, $1 \leq i < l$.

The third branch scans the input until it nondeterministically decides it has reached configuration C_i. Next, it checks if C_i does not yield C_{i+1} by first pushing that C_i/C_i^R onto the stack, and then comparing symbols from C_{i+1}^R/C_{i+1} and symbols popped from the stack; they must match except around the head position (how they may differ depends on M's δ).