CS1010: Theory of Computation

Lecture 13: Other approaches to reduction

Lorenzo De Stefani

Fall 2020

Outline

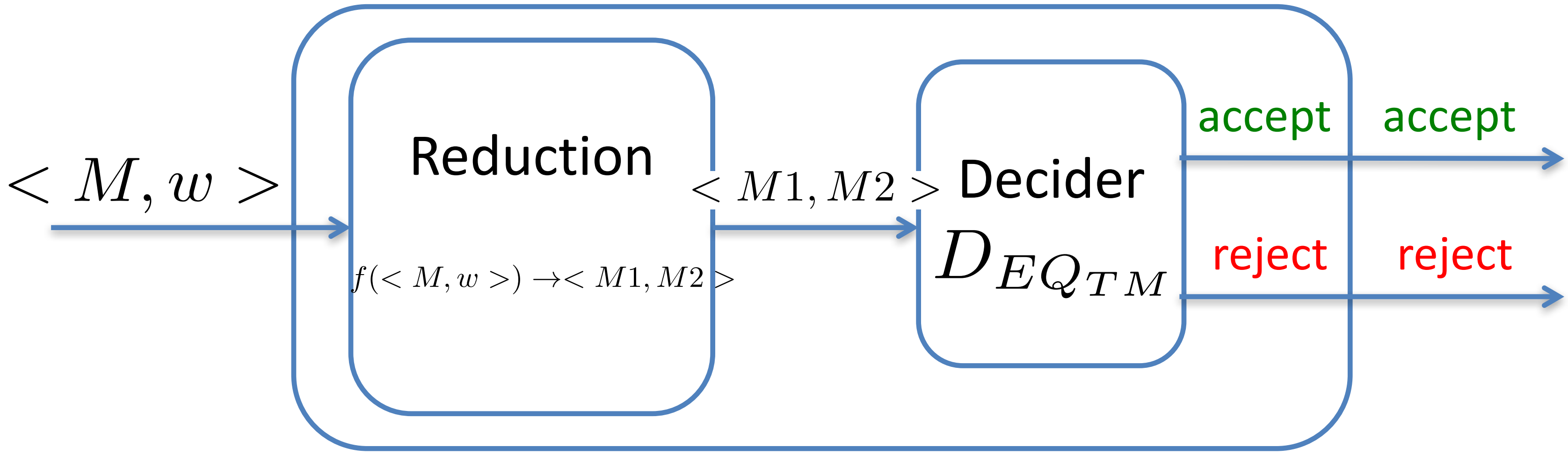
- Mapping Reducibilty
- Non Turing-recognizable languages
- Reductions via computation histories
- The PCP problem (at a glance)

From Sipser Chapter 5.1-5.3

EQ_{TM}

EQ_{TM} = {<M1,M2>| L(M1)=L(M2)}

- it is undecidable! We saw reduction from E_{TM}
- we could reduce from A_{TM} as well



  - M1 is a TM that accepts Σ^*
  - M2 is a TM that accepts Σ^* if M accepts w (simulate M on w)
- f is the reduction function

Computable functions

A function $f : \Sigma^* \rightarrow \Sigma^*$ is computable if there exists a TM M such that for every $x \in \Sigma^*$, M halts with just $f(x)$ on its tape

- Example: Let Σ be a fixed alphabet and define $f(x)$ as:
  - If $w = < M >$ for some TM, then $f(w) = < M' >$ where M' is M with q_{accept} and q_{reject} swapped;
  - Otherwise, $f(w) = w$
- f is computable!

Mapping reducibility

A language $A \subseteq \Sigma^*$ is mapping reducible to $B \subseteq \Sigma^*$ (write $A \leq_m B$) if there exists a computable function $f : \Sigma^* \to \Sigma^*$ such that for every $x \in \Sigma^*$

$$x \in A \textbf{ if and only if } f(x) \in B$$

That is, the function f maps members of A to members of B and non-members of A to non-members of B

- Example: $A_{TM} \leq_m EQ_{TM}$. The computable function is $f(< M, w >) =< M1, M2 >$ such that M1 and M2 are as defined in the previous example
- Then, $< M, w > \in A_{TM} \iff f(< M, w >) \in EQ_{TM}$

Properties of Mapping Reducibility

Theorem 5.22: If $A \leq_m B$ and B is decidable, then so is A

Proof:

- Since $A \leq_m B$ there exists a computable function f that realizes the reduction from A to B
- Since f is computable, there exists M_f that computes the reduction
- Since B is decidable there exists a decider M_B for it.
- We construct a decider M_A for A:
  1. On input x for M_A, simulate M_f on x to compute $f(x)$
  2. Simulate M_B on $f(x)$. M_A accepts x iff M_B accepts $f(x)$
- Since $A \leq_m B$ then M_A decides A
  1. By mapping reducibility if $f(x) \in B$ then $x \in A$
  2. If M_B accepts then M_A accepts as well

Properties of Mapping Reducibility

Corollary 5.23: If $A \leq_m B$ and A is undecidable, then so is B

Proof:

- There exists a computable function f that realizes the reduction from A to B
- Since f is computable, there exists M_f that computes the reduction
- Assume towards contradiction that there exists a decider M_B for B, construct a decider M_A for A:
  1. On input x for M_A, simulate M_f on x to compute $f(x)$
  2. Simulate M_B on $f(x)$. M_A accepts x iff M_B accepts $f(x)$
- Since $A \leq_m B$ then M_A decides A → contradiction!

Example: $\text{DECIDER}_{\text{TM}}$ is Undecidable

- $DECIDER_{TM} = \{< M > |\text{TM } M \text{ is a decider}\}$
- Show that $A_{TM} \leq_m DECIDER_{TM}$
- Show that there exists a computable function mapping the two languages
  - On input <M, w>, f outputs <M'> such that: M' on input x simulates M on w and
    - M' accepts x if M accepts w
    - Otherwise M' enters an infinite loop

$$< M, w > \in A_{TM} \iff < M' > \in DECIDER_{TM}$$

  - To compute f (build M') we modify M
  - By construction, this function is computable

Mapping Reducibility and Recognizability

Theorem 5.28: If $A \leq_m B$ and B is recognizable, then so is A

Proof:

- Since $A \leq_m B$ there exists a computable function f that realizes the reduction from A to B
- Since f is computable, there exists M_f that computes the reduction
- Let M_B be a recognizer for B, construct a recognizer M_A for A:
  1. On input x for M_A, simulate M_f on input x to compute $f(x)$
  2. Simulate M_B on $f(x)$:
     1. M_A accepts (resp., rejects) x, if M_B accepts (resp., rejects) f(x)
     2. M_A loops on x, if M_B loops on f(x)
- Since $A \leq_m B$ then M_A recognizes A

Properties of Mapping Reducibility

Corollary 5.29: If $A \leq_m B$ and A is not Turing-recognizable, then neither is B

Proof:

- Since $A \leq_m B$, there exists a computable function f that realizes the reduction from A to B
- Since f is computable, there exists M_f that computes the reduction
- Assume towards contradiction that there exists a recognizer M_B for B, construct a recognizer M_A for A:
  1. On input x for M_A, simulate M_f on x to compute $f(x)$
  2. Simulate M_B on $f(x)$ so that
     1. M_A accepts (resp., rejects) x if M_B accepts (resp., reject) $f(x)$
     2. M_A loops forever on x if M_B loops on $f(x)$
- Since $A \leq_m B$ then M_A recognizes A → contradiction!

EQ_{TM} is not TM recognizable

- We would like to use Corollary 5.29.
- Which non Turing Recognizable language L which is mapping reducible EQ_{TM} to can we use?
  - We know A_{TM} is undecidable.
  - We know A_{TM} is Turing recognizable.
  - This implies A^c_{TM} is not Turing-recognizable
- We need to show $A^c_{TM} \leq_m EQ_{TM}$

$A_{TM}^c \leq_m EQ_{TM}$

We need to construct a computable function f which realizes the reduction from to A_{TM}^c to EQ_{TM}:

- On input <M,w>, f returns <M1,M2> which on input x
  - M1 rejects x
  - M2 runs w on M and accepts (rejects, loop), if M does
- f is computable
  - $< M, w > \in A_{TM} \rightarrow L(M1) = L(M2) = \emptyset \rightarrow < M1, M2 > \in EQ_{TM}$
  - $< M, w > \notin A_{TM} \rightarrow L(M1) \neq L(M2) \rightarrow < M1, M2 > \notin EQ_{TM}$
- EQ_{TM} is not Turing-recognizable

Mapping reducibility and complement

Theorem : If $A \leq_m B$ then $A^c \leq_m B^c$

Proof:

- Since $A \leq_m B$, by definition there exists a computable function $f : \Sigma^* \to \Sigma^*$ such that for every $x \in \Sigma^*$, $x \in A$ if and only if $f(x) \in B$
- But then the same function must also be such that for every $x \in \Sigma^*$, $x \in A^c$ if and only if $f(x) \in B^c$

$$A_{TM} \leq_m DECIDER_{TM} \quad \longrightarrow \quad A_{TM}^c \leq_m DECIDER_{TM}^c$$

Example:

- Since A_{TM}^c is non Turing-recognizable, neither is $DECIDER_{TM}^c$

Reminder of configurations of a TM

- At any step a TM is in a certain configuration which is specified by:
  - the state
  - the current reader head location
  - the symbol at the current head location
- We say that a configuration C1 yields C2 if there is a transition which allows to go from C1 to C2

Types of configurations

- Starting configuration: in starting state, head position at the beginning of the input
  - Leftmost position of the tape occupied by the input
- Accepting configuration: in accepting state
- Rejecting configuration: in rejecting state
- Halting configuration: either accepting or rejecting configurations

Computation histories

- An accepting computation history for a TM is a sequence of a configurations $C_1C_2...C_j$ such that:
  - C_1 is the start configuration for input w
  - C_j is an accepting configuration, and
  - Each C_i follows legally from C_{i-1}
- Analogous definition for rejecting computation history
- Computation histories are finite – if M does not halt on a given input there is no history
- For Deterministic TM: any accepting or rejecting computation histories for a single input
- Non-Deterministic TMs: multiple possible histories corresponding to the possible execution branches

Linear Bounded Automaton

- Suppose we reduce the power of a TM so that the head never moves outside the boundaries of the input string
- Such a TM is called Linear Bounded Automaton

Lemma 5.8: Let M be a LBA with q states and g symbols in the tape alphabet. There are exactly qng^n distinct configurations of a tape of length n

Proof: M can be in one of q states, the head can be on one of n cells, there are g^n possible strings on the tape at any time

A_{LBA} is decidable

Theorem 5.9:

$$A_{LBA} = \{< M, w > | \quad \text{M is an LBA that accepts string } w\}$$

Is decidable.

- We need to build a decider D for A_{LBA}
- We simulate M on input w
  - If M accepts and halts/rejects D halts/rejects accordingly
- How do we handle loops?
  - Consider the sequence of configuration of M on input w $C_1, C_2, \ldots, C_j, \ldots C_i$
  - If there exist j,i such that $C_j = C_i$ we have a cycle!
  - The computation can continue on that loop forever!
  - Can we detect the loop? From Lemma 5.8 there are a finite number of possible configurations! If there is a loop we will detect in finite time
  - If loop is detected the decide D rejects w!
  - M loops if and only if it does not accept w → D is decider!

Computation over Computation Histories

- Consider an accepting computation history of a TM $C_1C_2\ldots C_i$
- Each configuration C_i can be codified as a string $<C_i>$!
- Consider the following string

$$\#\underbrace{\qquad\qquad}_{C_1}\#\underbrace{\qquad\qquad}_{C_2}\#\underbrace{\qquad\qquad}_{C_3}\#\cdots\#\underbrace{\qquad\qquad}_{C_l}\#$$

- The set of all valid accepting histories is also a language!
- Such strings have finite lengths!
  - No infinite loop repetitions if accepting history!
- An LBA B can check if a given string is a valid accepting computation history for a TM M accepting w
  - Check that C_1 is a valid starting configuration for M
  - Check that C_i is a valid accepting configuration for M
  - Check that C_{j+1} follows legally from C_j for j=1,2,..i-1
- If $L(B) \neq \emptyset$ then M accepts w!

$E_{LBA} = \{< M, w > | M \text{ is an LBA and } L(B) = \emptyset\}$

- It is undecidable!
- Proof idea: reduction from A_{TM}
  - Assume towards contradiction that R decides E_{LBA}
  - Show how to build a decider D for A_{TM}
  - Use the construction previously seen to obtain an LBA B such that

$$L(B) \neq \emptyset \iff w \in L(M)$$

  - Given B as input to R, then we have

$$R \text{ rejects } B \iff L(B) \neq \emptyset$$

  - Thus,

$$R \text{ rejects } B \iff w \in L(M)$$

  - D accepts/rejects if R rejects/accepts B → D decides A_{TM}!
  - Contradiction!

The Post-Correspondence Problem

- Are issues of undecidability confined to problems concerning automata and languages?
- No! There are other *algorithmic*, *natural* undecidable problems
- The Post Correspondence Problem (PCP) is a *tiling problem* over strings:
  - A tile, or domino, contains two strings t (top) and b (bottom) $\left[\frac{t}{b}\right] = \left[\frac{ca}{a}\right]$
- Consider a set of given dominos

$$\left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\}$$

- A match is a list of these dominos so that when concatenated the top and the bottom strings are identical

$$\left[\frac{a}{ab}\right]\left[\frac{b}{ca}\right]\left[\frac{ca}{a}\right]\left[\frac{a}{ab}\right]\left[\frac{abc}{c}\right] = \frac{abcaaabc}{abcaaabc}$$

- Some sets have no match!

The Post-Correspondence Problem

- Given a set of dominos, or an instance of the PCP problems we would like to be able to decide whether there exists a match!
- Can we rephrase this is terms of languages?

$$L_{PCP} = \{< P > | P \text{ is an instance of PCP and it has a match}\}$$

- Can we decide the language L_{PCP}?
- Theorem 5.15: L_{PCP} is undecidable!
- Proof idea:
  - Reduction from A_{TM} using computation histories approach!
  - We show the contradiction: if L_{PCP} is decidable so would A_{TM}
  - We will reduce an input <M,w> to a PCP instance that has a mathc if and only if M accepts w!
- If interested in the details check section 5.2