Computational Models - Lecture 9[1]

Handout Mode

Iftach Haitner and Yishay Mansour.

Tel Aviv University.

April 28/30, 2014

[1] Based on slides by Benny Chor, Tel Aviv University, modifying slides by Maurice Herlihy, Brown University.

Announcement

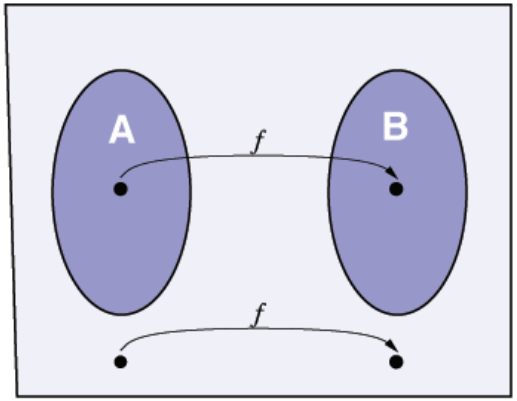
- Next week no classes
- Lectures (Monday and Wednesday) canceled
- Recitations (Wednesday and Thursday) canceled

Talk Outline

- $\mathcal{RE}$-Completeness
- Reductions via computational histories (CFG)
- Linear Bounded Automata
- Unrestricted Grammars

- Sipser's book, Chapter 5, Sections 5.1, 5.3

Mapping Reductions (Review)



A mapping reduction converts questions about membership in A to membership in B.

Theorem 1

If $A \leq_m B$ and B is decidable, then A is decidable.

Corollary 2

If $A \leq_m B$ and A is undecidable, then B is undecidable.

In fact, this has been our principal tool for proving undecidability of languages other than A_{TM}

Rice's Theorem

Non Trivial Properties of $\mathcal{RE}$ Languages:

- L is finite.
- L is infinite.
- L contains the empty string.
- L contains no prime number.
- L is co-finite.
- . . .

All these are non-trivial properties of enumerable languages, since for each of them there is $L_1, L_2 \in \mathcal{RE}$ such that L_1 satisfies the property but L_2 does not.

Theorem 3

Let $\emptyset \neq \mathcal{C} \subsetneq \mathcal{RE}$ and let $L_{\mathcal{C}} = \{\langle M \rangle : L(M) \in \mathcal{C}\}$. Then $L_{\mathcal{C}}$ is undecidable.

Rice's Theorem, Reflections

- Rice's theorem can be used to show undecidability of properties like
  - *Does* $\mathrm{L}(M)$ *contain infinitely many primes*
  - *Does* $\mathrm{L}(M)$ *contain an arithmetic progression of length 15*
  - *Is* $\mathrm{L}(M)$ *empty*
- Decidability of properties related to the encoding itself cannot be inferred from Rice.
  - The question *does* $\langle M \rangle$ *has an even number of states* is decidable.
  - The question *does* M *reaches state* q_6 *on the empty input string* is undecidable, but this does not follow from Rice's theorem.
- Rice does not say anything on membership in $\mathcal{RE}$.
- **Rice's Theorem is a powerful tool, but use it with care!**

Section 1

RE-Completeness

$\mathcal{RE}$-Completeness

Question 4

Is there a language L that is hardest in the class $\mathcal{RE}$?

Answer: Well, you have to define what you mean by "hardest language"...

Definition 5 ($\mathcal{RE}$-complete)

A language $L_0 \subseteq \Sigma^*$ is called $\mathcal{RE}$-complete, if the following holds

- $L_0 \in \mathcal{RE}$ (membership).
- for every $L \in \mathcal{RE}$ we have $L \leq_m L_0$ (hardness).

- The second item means that $\forall L \in \mathcal{RE}$, there is a mapping reduction f_L from L to L_0.
- The reduction f_L depends on L and will typically differ from one language to another.

A_{TM} is $\mathcal{RE}$-Complete.

Question 6

Are there $\mathcal{RE}$-complete languages?

Theorem 7

A_{TM} is $\mathcal{RE}$-Complete.

Proof:

- Clearly $A_{TM} \in \mathcal{RE}$.
- Let $L \in \mathcal{RE}$, and let M_L be a TM accepting it.
  Then $f_L(w) = \langle M_L, w \rangle$ is a mapping reduction from L to A_{TM} (why?).

♣

Other $\mathcal{RE}$-Complete problems

Question 8

Are there other $\mathcal{RE}$-complete languages?

Observations 9

Reductions are transitive:

$$A \leq_m B, \quad B \leq_m C \quad \Rightarrow \quad A \leq_m C$$

Theorem 10

H_{TM} *is* $\mathcal{RE}$*-Complete.*

Proof:

- Clearly $H_{TM} \in \mathcal{RE}$.
- We have $A_{TM} \leq_m H_{TM}$ and $L \leq_m A_{TM}$ for $L \in \mathcal{RE}$.

♣

Section 2

Controlled Executions

Bounded Acceptance – CET is Decidable

Definition 11

CET $:= \{\langle M, w, k\rangle : \ M$ accepts w within k steps$\}$.

Theorem 12

CET *is decidable.*

Proof?

What about space?

Definition 13

CES $:= \{\langle M, w, k\rangle : \ M$ accepts w using k cells$\}$.

Theorem 14

CES *is decidable.*

Bounded Acceptance – CET is Decidable, 2

Theorem 15

CES *is decidable.*

Proof: How to check that the computation will not terminate?

Three different proofs, involving TM configurations.
Let $m = |Q| \cdot |\Gamma|^k \cdot k$ be the number of configurations.

- Wait until a configuration repeats.
- Run for $m + 1$ steps.
- Build an automata with states as configurations, and an edge if M moves from one configuration to another. Check if there is a cycle reachable from the start state.

♣

Reductions via Controlled Executions

$L_\infty = \{\langle M \rangle : \ L(M) \text{ is infinite}\}$

- By Rice Theorem: $L_\infty \notin \mathcal{R}$ (why?)

Theorem 16

$L_\infty \notin \mathcal{RE}$.

Proof's idea:

Reduction from $\overline{H_{TM}}$.

- We are after a reduction $f(\langle M, w \rangle) = \langle B_{M,w} \rangle$ such that
  - If M halts on $w \implies L(B_{M,w})$ is finite.
  - If M does not halt on $w \implies L(B_{M,w})$ is infinite.
- It will follow that $x \in \overline{H_{TM}} \iff f(x) \in L_\infty$
- Hence, $\overline{H_{TM}} \leq_m L_\infty$.
- Since $\overline{H_{TM}} \notin \mathcal{RE}$, this will imply $L_\infty \notin \mathcal{RE}$.

The TM $B_{M,w}$

Definition 17 ($B_{M,w}$)

On input y

1. Emulate $M(w)$ for $|y|$ steps.
2. Accept, if $M(w)$ did not halt in that many steps; Otherwise, Reject.

- $M(w)$ does not halt $\Longrightarrow$ $B_{M,w}$ accepts all y's $\Longrightarrow$ $\mathsf{L}(B_{M,w}) = \Sigma^*$ $\Longrightarrow$ $\langle B_{M,w} \rangle \in \mathsf{L}_\infty$.
- $M(w)$ halts after k steps $\Longrightarrow$ $B_{M,w}$ accepts only y's of length smaller than k $\Longrightarrow$ $\mathsf{L}(B_{M,w})$ is finite $\Longrightarrow$ $\langle B_{M,w} \rangle \notin \mathsf{L}_\infty$.

  Hence, $x \in \overline{\mathsf{H}_{\mathsf{TM}}} \Longleftrightarrow f(x) \in \mathsf{L}_\infty \Longrightarrow \overline{\mathsf{H}_{\mathsf{TM}}} \leq_m \mathsf{L}_\infty \Longrightarrow \mathsf{L}_\infty \notin \mathcal{RE}$

Section 3

Computation Histories

Reduction via Computation Histories

Important technique for proving undecidability. Examples

- Basis for proof of undecidability in Hilbert's tenth problem (where "object" is integral root of polynomial).

Today's lecture:

- Does a context free grammar generate Σ^*?
- Does a linear bounded TM accept the empty language?

Reminder: Configurations

Configuration: $1011q_70111$, means:

- state is q_7
- LHS of tape is 1011
- RHS of tape is 0111
- head is on RHS 0

- (configuration) uaq_ibv yields (i.e., $\Longrightarrow$) uq_jacv, if $\delta(q_i, b) = (q_j, c, \mathrm{L})$
- uaq_ibv yields $uacq_jv$ if $\delta(q_i, b) = (q_j, c, \mathrm{R})$
- Special case (left end of tape): q_ibv yields q_jcv if $\delta(q_i, b) = (q_j, c, \mathrm{L})$.

Computation Histories

Let M be a TM and w an input string.

- An accepting computation is $\#C_1\#C_2\#\ldots\#C_\ell\#$, where
  1. C_1 is the starting configuration of M on w,
  2. C_ℓ is an accepting configuration of M,
  3. Each C_i yields C_{i+1} by transition function of M.
- A string is not an accepting computation history if it fails one or more of these conditions.
- A rejecting computation history for M on w is the same, except
  - C_ℓ is a rejecting configuration of M.

Remark 18

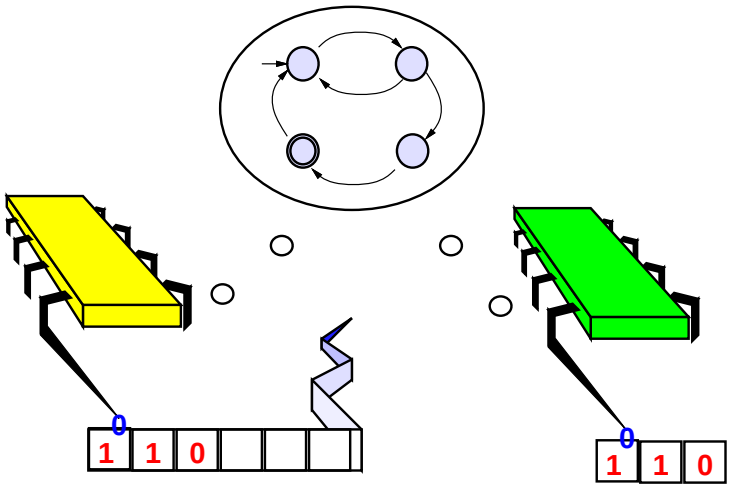
- Computation sequences are finite.
- If M does not halt on w, neither accepting nor rejecting computation history exist.
- Notion is useful for both deterministic (one history) and non-deterministic (many histories) TMs.

Section 4

Linear Bounded Automata

Linear Bounded Automata – LBA

- A restricted form of TM.
- Cannot move off portion of tape containing input (have no such instruction, for example, suppose $ signifies the end of input.)
- Size of input determines size of memory



turing machine linear bounded automaton

Question 19

Why is it called “linear"?

Answer: Using a tape alphabet larger than the input alphabet increases memory by a constant factor.

LBAs are Powerful!

- The deciders we seen for the following languages are all LBAs.
  - A_{DFA} (does a DFA accept a string?)
  - A_{CFG} (is string in a CFG?)
  - $EMPTY_{DFA}$ (is a DFA trivial?)
  - E_{CFG} (is a CFL empty?)
- Every CFL can be decided by an LBA.
- Not too easy to find a natural, decidable language that cannot be decided by an LBA.
- Almost all the algorithms in the data-structure and algorithm courses are linear space!

Acceptance for LBAs – A_{LBA}

$$A_{LBA} = \{\langle M, w\rangle : M \text{ accepts } w \text{ in linear space}\}$$

Question 20

Is A_{LBA} decidable?

YES, similar to controlled executions.

LBA are decidable

Theorem 21

A_{LBA} *is decidable.*

Proof's idea:

- Emulate $M(w)$, where if M tries to "exit" the input space, halt and reject.
- But what if M loops?
- M loops iff it repeats a configuration (Why?)

  By pigeon hole, if our LBA M runs long enough, it must repeat a configuration.

LBA's have Bounded Number of Configuration

Lemma 22

Let M be a LBA with q states, g symbols in tape alphabet. Then on input of size n, M has at most qng^n distinct configurations.

Proof: A configuration involves:

- control state (q possibilities)
- head position (n possibilities)
- tape contents (g^n possibilities)

♣

Decider for A_{LBA}

Algorithm 23

On input $\langle M, w \rangle$, where M is an LBA and $w \in \Sigma^*$,

1. Emulate $M(w)$ while maintaining a step counter
2. Counter incremented by 1 per each simulated step (of M).
3. Keep emulating M for $qng^n + 1$ steps, or until it halts (whichever comes first)
4. Accept if M has halted and accepted; otherwise, Reject

Emptiness for LBAs – All_{LBA}

$$\text{EMPTY}_{\text{LBA}} = \{\langle M \rangle : M \text{ is an LBA and } L(M) = \emptyset\}$$

Question 24

Is $\text{EMPTY}_{\text{LBA}}$ decidable?

Theorem 25

$\text{EMPTY}_{\text{LBA}}$ *is undecidable.*

Proof's idea:

- Given a TM M and input w, we construct an LBA $B_{M,w}$ such that if $\langle M, w \rangle \in A_{\text{TM}}$, then $L(B_{M,w})$ contains the accepting computation history for M on w
- Hence, M accepts w iff $L(B_{M,w}) \neq \emptyset$.

  $\Longrightarrow A_{\text{TM}} \leq_m \overline{\text{EMPTY}_{\text{LBA}}} \Longrightarrow \overline{\text{EMPTY}_{\text{LBA}}} \notin \mathcal{R} \Longrightarrow \text{EMPTY}_{\text{LBA}} \notin \mathcal{R}$.

The LBA $B_{M,w}$

Algorithm 26 (The LBA, $B_{M,w}$)

On input x

1. Split x according to the $\#$ delimiters into $C_1, C_2, \ldots, C_\ell$.
2. Check that all the following conditions hold:
   1. Each C_i is a configuration of M
   2. C_1 is the start configuration of M on w
   3. C_ℓ is an accepting configuration
   4. Every C_{i+1} follows from C_i according to M

- The only challenging task is to compute Step 4 in linear space

Algorithm 27 (Does $C_i \implies C_{i+1}$?)

1. Zig-zag between corresponding positions of C_i and C_{i+1}.
2. Use "dots" on tape to mark current position

This can be done inside space allocated by the input. Thus $B_{M,w}$ is indeed a LBA.

Putting It Together

- The LBA, $B_{M,w}$, accepts x iff x is an accepting computation history of M on w.
- Therefore $\mathsf{L}(B_{M,w})$ is either empty or a singleton $\{x\}$.

  $\Longrightarrow \langle M, w\rangle \in \mathsf{A}_{\mathsf{TM}} \Longleftrightarrow \langle B_{M,w}\rangle \in \overline{\mathsf{EMPTY}_{\mathsf{LBA}}}$.
- The reduction $\langle M, w\rangle \mapsto B_{M,w}$ is computable

  $\Longrightarrow \mathsf{A}_{\mathsf{TM}} \leq_m \overline{\mathsf{EMPTY}_{\mathsf{LBA}}} \Longrightarrow \overline{\mathsf{EMPTY}_{\mathsf{LBA}}} \notin \mathcal{R} \Longrightarrow \mathsf{EMPTY}_{\mathsf{LBA}} \notin \mathcal{R}$. ♠

Wholeness for LBAs – All_{LBA}

$$\text{All}_{\text{LBA}} = \{\langle M \rangle : M \text{ is an LBA and } L(M) = \Sigma^*\}$$

Question 28

Is All_{LBA} decidable?

Theorem 29

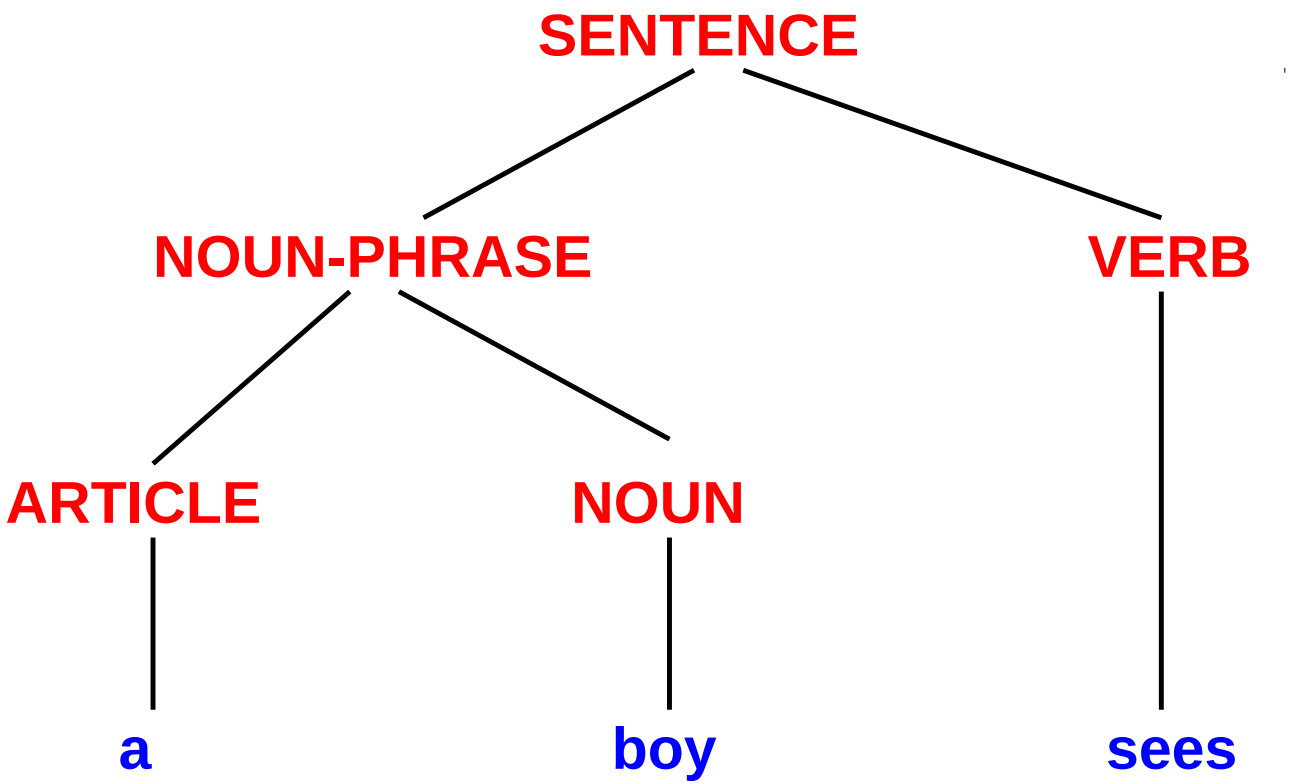
All_{LBA} *is undecidable.*

Proof's idea: Same as in $\text{EMPTY}_{\text{LBA}}$, flip accept and reject in $B_{M,w}$.

Question 30

Are $\text{EMPTY}_{\text{LBA}}, \text{All}_{\text{LBA}} \in \mathcal{RE}$? Are $\text{EMPTY}_{\text{LBA}}, \text{All}_{\text{LBA}} \in co\text{-}\mathcal{RE}$? Both?

Computation Histories and Emptiness of CFGs



Emptiness of CFGs

We have seen an algorithm to check whether a CFG is empty.

Algorithm 31

On input $\langle G \rangle$ (where G is a CFG):

1. Mark all terminal symbols in G.
2. Repeat until no new variables become marked:
3. Mark any A where $A \to U_1 U_2 \dots U_k$, and each U_i has already been marked.
4. Accept, if start symbol marked; otherwise Reject.

So E_{CFG} is decidable.

Question 32

What about the complementary question: does a CFG generate all strings?
Namely, does $All_{CFG} := \{\langle G \rangle : \ G \text{ is a CFG and } L(G) = \Sigma^*\} \in \mathcal{R}$

All_{CFG} is Undecidable

Theorem 33

All_{CFG} *is undecidable.*

Proof's idea: Reduction from A_{TM} to $\overline{All_{CFG}}$:

1. Given $\langle M, w \rangle$, construct a coding of a CFG, $\langle G \rangle$, that generates all strings that are not accepting computation histories for M on w
2. if M does not accept w, G generates all strings
3. if M does accept w, then G does not generate the accepting computation history.

The PDA

Instead of a CFG, we construct a PDA (recall equivalence) that "guesses" which condition is violated, and verifies the guessed violation.

Algorithm 34 (D)

On input $h = C_1 \# C_2 \ldots \# C_\ell$, check

1. Is there some C_i that is not a configuration of M (i.e., number of q symbols $\neq 1$)?
2. Is C_1 not the starting configuration of M on w?
3. Is C_ℓ not an accepting configuration of M?
4. $\exists i \in [\ell]$ s.t. $C_i \not\Rightarrow C_{i+1}$ according to δ – the transition function of M?

The last condition is the tricky one to check.

Checking $C_i \not\Rightarrow C_{i+1}$

Algorithm 35 (Checking $C_i \not\Rightarrow C_{i+1}$)

1. Push C_i onto the stack till $\#$.
2. Scan C_{i+1} and pop matching symbols of C_i

   Check if C_i and C_{i+1} match everywhere, except around the head position where difference dictated by transition function for M.

Problem

When C_i is popped from stack, it is in reverse order.

But we only trying to identify (ignoring the local changes around head position) the language $x\#y$, with $x \neq y$.

This can be done a PDA (see Lecture 5), but here we give a simpler solution.

Checking $C_i \not\Rightarrow C_{i+1}$, take 2

- So far, we used a "straight" notion of accepting computation histories

- But why not employ an alternative notion of accepting computation history, one that will make the life of our PDA much easier?

  A solution: write the accepting computation history so that every other configuration is in reverse order.

- This resolves the difficulty in the proof.

Putting It Together

- Given $\langle M, w \rangle$, we constructed (algorithmically) a PDA, D, that rejects the string z if and only if z equals an accepting computation history of M on w, written in the "alternating format".
- Therefore $\mathsf{L}(D)$ is either Σ^* or $\Sigma^* \setminus \{z\}$.
- This D has an equivalent (and efficiently described) CFG, G, namely $\mathsf{L}(D) = \mathsf{L}(G)$. So $\mathsf{L}(G)$ is either Σ^* or $\Sigma^* \setminus \{z\}$. The mapping $\langle M, w \rangle \mapsto \langle G \rangle$ is thus a reduction from A_{TM} to $\overline{\mathsf{All}_{\mathsf{CFG}}}$.
- (Since $\mathsf{A}_{\mathsf{TM}} \notin \mathcal{R}$) $\implies$ $\overline{\mathsf{All}_{\mathsf{CFG}}} \notin \mathcal{R}$.
- (Since $L \in \mathcal{R} \Leftrightarrow \overline{L} \in \mathcal{R}$) $\implies$ $\mathsf{All}_{\mathsf{CFG}} \notin \mathcal{R}$. ♠

Section 5

Unrestricted Grammars

Unrestricted Grammars

Unrestricted grammars (i.e., context dependant grammar) are similar to context free ones, except left hand side of rules can be strings of variables and terminal with at least one variable.

To non-deterministically generate a string according to a given unrestricted grammar:

1. Start with the initial symbol
2. While the string contains at least one non-terminal:
   1. Find (non deterministically) a substring that matches the LHS of some rule
   2. Replace that substring with the RHS of the rule

Unrestricted Grammar for $\{a^n b^n c^n\}$

Definition 36 (Unrestricted Grammar for $\{a^n b^n c^n\}$)

- Generate the variable sequence $L(ABC)^n$:
  $S \to LT|\epsilon$;
  $T \to ABCT|\epsilon$;
- Sort the $\{A, B, C\}$ and get $LA^k B^k C^k$.
  $BA \to AB$;
  $CB \to BC$;
  $CA \to AC$;
- Replace the variables by terminals.
  $LA \to a$;
  $aA \to aa$;
  $aB \to ab$;
  $bB \to bb$;
  $bC \to bc$;
  $cC \to cc$;

Formal Definition

An unrestricted grammar is a 4-tuple (V, Σ, R, S), where

- V is a finite set of variables
- Σ is a finite set of terminals
- R is a finite set of rules: each rule is has two finite strings of variables and terminals.
- S is the start symbol.
- If u and v are strings of variables and terminals, and $y \to w$ is a rule of the grammar, then uyv yields uwv, written $uyv \to uwv$.
- $u \stackrel{*}{\to} v$ if $u = v$, or $u \to u_1 \to \ldots \to u_k \to v$ for some sequence $u_1, u_2, \ldots, u_k$

Definition 37

The language of the grammar G, denoted $\mathcal{L}(G)$, is $\{w \in \Sigma^* : S \stackrel{*}{\to} w\}$

where $\stackrel{*}{\to}$ is determined by G.

The Class $\mathcal{UG}$

$\mathcal{UG} = \{L : \exists$ unrestricted grammar $G : L(G) = L\}$

I.e., the set of languages that can be described by an unrestricted grammar

Theorem 38

$\mathcal{UG} = \mathcal{RE}$

We show that:

- $\mathcal{UG} \subseteq \mathcal{RE}$
- $\mathcal{RE} \subseteq \mathcal{UG}$

Proving $\mathcal{UG} \subseteq \mathcal{RE}$

Given any unrestricted grammar G, we create a two-tape non-deterministic TM M that accepts $L(G)$.

Algorithm 39

Maintain input w on tape 1 and initialize tape 2 to the initial symbol S.

Do (until accept):

1. Move (non-deterministically) to some location on tape 2
2. Select (non-deterministically) a rule R and try to apply it to that location.
3. Accept if tape 1 and tape 2 are identical.

Idea of the proof $\mathcal{RE} \subseteq \mathcal{UG}$

- Let L $\in \mathcal{RE}$ and let M be a deterministic Turing Machine that accepts it. We create an unrestricted grammar G with $L(G) = L$
- Idea: variables of G are the states Q
- Assume the input starts with $[q_0 w]$,
- Simulate $M(w)$:
  $\delta(q, a) = (q', b, R) \Rightarrow qa \rightarrow bq'$
  $\delta(q, a) = (q', b, L) \Rightarrow cqa \rightarrow q'cb$; $[qa \rightarrow [q'b$;
  $\forall q \quad q] \rightarrow q\sqcup]$
- Two issues:
  (1) Need to initially generate $[q_0 w]$,
  (2) need to output w when reaching q_a.

Proving $\mathcal{RE} \subseteq \mathcal{UG}$

- Let L $\in \mathcal{RE}$ and let M be a deterministic Turing Machine that accepts it. We create an unrestricted grammar G with $L(G) = L$
- Idea: variables of G are the states Q
  - Maintain $w[c]$, where w is the input and c is the current configuration.
  - if c is an accepting configuration, replace $[c]$ by ϵ.

The Unrestricted Grammar for $L(M)$

Idea: maintain $w[c]$, where w is the input and c is the current configuration.

Definition 40 (Unrestricted Grammar G for $L(M)$)

- Generate the string $w[q_0 w]$:
  $S \to T[q_0]$
  For all $a \in \Sigma$:
  $T \to aTA_a | \epsilon$; $A_a[q_0 \to [q_0 A_a$; $A_a] \to a]$; $\forall b \in \Sigma \;\; A_a b \to b A_a$;
- Simulate $M(w)$:
  $\delta(q,a) = (q',b,R) \Rightarrow qa \to bq'$
  $\delta(q,a) = (q',b,L) \Rightarrow cqa \to q'cb$; $[qa \to [q'b$;
  $\forall q \quad q] \to q\sqcup]$
- Accepting – derive w from $w[uq_a v]$:
  $q_a \to E_L E_R$
  $aE_L \to E_L$; $[E_L \to \epsilon$
  $E_R a \to E_R$; $E_R] \to \epsilon$

Section 6

Primes, Yet Another Example

Primes

- $\text{Primes} = \{p \in \mathbb{N} : p \text{ is a prime}\}$
- $\text{Primes}_{\text{TM}} = \{\langle M \rangle : M \text{ is a TM and } \text{L}(M) = \text{Primes}\}$

Theorem 41

$\text{Primes}_{\text{TM}} \notin \mathcal{RE}$.

Proof's idea: We define a computable function f with

$$\langle M, w \rangle \in \overline{\text{A}_{\text{TM}}} \iff \langle B_{M,w} \rangle \in \text{Primes}_{\text{TM}}$$

Hence, $\overline{\text{A}_{\text{TM}}} \leq_m \text{Primes}_{\text{TM}}$

$\overline{A_{TM}} \leq_m \text{Primes}_{TM}$

Let P be a decider for Primes (how?).

Definition 42 ($B_{M,w}$)

On input x

- Emulate $P(x)$ and Accept if P accepts, otherwise continue
- Emulate $M(w)$ and Accept if M accepts, otherwise reject

- f is computable
- f does a correct mapping reduction:
  - $\langle M, w \rangle \in \overline{A_{TM}} \implies L(B_{M,w}) = \text{Primes}$
  - $\langle M, w \rangle \notin \overline{A_{TM}} \implies L(B_{M,w}) = \mathbb{N} \neq \text{Primes}$

  Hence $\overline{A_{TM}} \leq_m \text{Primes}_{TM} \implies \text{Primes}_{TM} \notin \mathcal{RE}$

Generalizing Primes

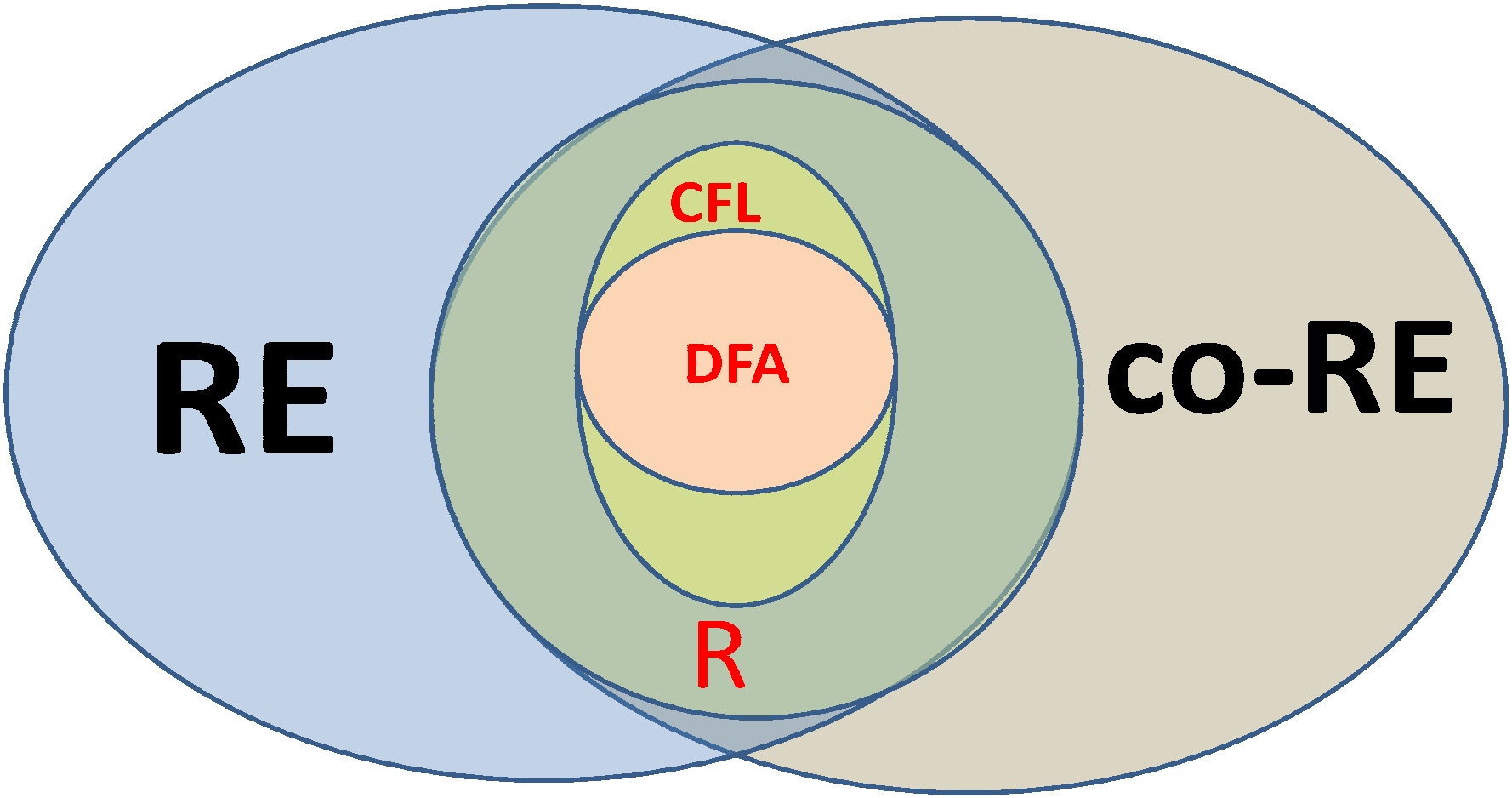
Question 43

What property of Primes have we used?

Theorem 44

Let $\Sigma^* \notin \mathcal{C} \subset \mathcal{R}$ *and* $\mathcal{C} \neq \emptyset$.
Let $\mathsf{L}_{\mathcal{C}} = \{\langle M \rangle : \mathsf{L}(M) \in \mathcal{C}\}$.
Then $\mathsf{L}_{\mathcal{C}} \notin \mathcal{RE}$.

Revised View of the World of Languages



Decidability, Summary

- Turing Machine - a universal computational model
- Language classes RE, co-RE, and R.
- None Decidability
  - Acceptance/Halting problem
  - Any non-trivial property of a program (Rice Theorem)
  - Questions with respect to Grammars.
  - much more exists ...
- Not in RE (what does it mean?)
- What does this imply to verification of software and hardware? (useless ???)