

# Computational Models - Lecture 9<sup>1</sup>

## Handout Mode

Iftach Haitner and Yishay Mansur.

Tel Aviv University.

May 7/9, 2012

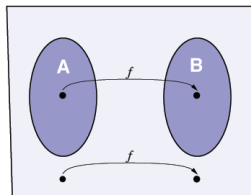
---

<sup>1</sup>Based on slides by Benny Chor, Tel Aviv University, modifying slides by Maurice Herlihy, Brown University.

## Talk Outline

- Undecidability by Rice Theorem
- $\mathcal{RE}$ -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  $A_{TM}$

# Section 1

## **Rice's Theorem**

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

A few examples

- $L$  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.

### Question 3

Are there **trivial** properties of  $\mathcal{RE}$  languages?

# Rice's Theorem

## Theorem 4

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.

Proof's idea: Reduction from  $H_{\text{TM}}$ : Given  $M$  and  $w$ , we construct a TM  $B_{M,w}$  such that:

- If  $M$  halts on  $w$ , then  $\langle B_{M,w} \rangle \in L_{\mathcal{C}}$ .
- If  $M$  does not halt on  $w$ , then  $\langle B_{M,w} \rangle \notin L_{\mathcal{C}}$ .

It will follow that  $f(\langle M, w \rangle) := \langle B_{M,w} \rangle$  is a **mapping reduction** from  $H_{\text{TM}}$  to  $L_{\mathcal{C}} \implies H_{\text{TM}} \leq_m L_{\mathcal{C}} \implies L_{\mathcal{C}}$  is undecidable.

## Proving Rice's Theorem

We assume wlg. that  $\emptyset \notin \mathcal{C}$  (otherwise, look at  $\bar{\mathcal{C}}$ , also proper and non-empty).

Fix  $L \in \mathcal{C}$  and let  $M_L$  be a TM accepting it (recall  $\mathcal{C} \subseteq \mathcal{RE}$ ).

Given a pair  $\langle M, w \rangle$  of TM and a string, define

### Algorithm 5 ( $B_{M,w}$ )

On input  $y$ :

1 Emulate  $M(w)$ .

2 Emulate  $M_L(y)$ :

Accept if  $M_L$  accepts; Reject if  $M_L$  rejects.

Let  $f(\langle M, w \rangle) := \langle B_{M,w} \rangle$ , and let  $f(x) = \emptyset$  if  $x$  is not of the form  $\langle M, w \rangle$ .

### Claim 6

$f$  is a mapping reduction from  $H_{TM}$  to  $L_{\mathcal{C}}$

## $f$ is Computable

### Claim 7

$f$  is computable.

Proof: On a valid pair  $\langle M, w \rangle$ , the TM  $B_{M,w} = f(\langle M, w \rangle)$  is simply a concatenation of two known TMs: the **universal machine** and  $M_L$ . ♣



$$\langle M, w \rangle \in H_{TM} \iff f(\langle M, w \rangle) \in L_C$$

### Claim 8

$$\langle M, w \rangle \in H_{TM} \iff f(\langle M, w \rangle) \in L_C$$

Proof:

If  $\langle M, w \rangle \in H_{TM}$ , then  $B_{M,w}$  gets to **Step ??**, and emulates  $M_L(y)$ .

Hence  $L(B_{M,w}) = L \in C$ .

Otherwise (i.e.,  $\langle M, w \rangle \notin H_{TM}$ ),  $B_{M,w}$  never gets to **Step ??**.

Hence  $L(B_{M,w}) = \emptyset \notin C$ .

- Thus,  $\langle M, w \rangle \in H_{TM}$  iff  $\langle B_{M,w} \rangle \in L_C$ .



We proved that  $H_{TM} \leq_m L_C$ , thus  $L_C$  is undecidable.

## Reflections

- Rice's theorem can be used to show **undecidability** of properties like
  - ▶ Does  $L(M)$  contain infinitely many primes
  - ▶ Does  $L(M)$  contain an arithmetic progression of length 15
  - ▶ Is  $L(M)$  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 2

# Controlled Executions

## Bounded Acceptance – CET is Decidable

### Definition 9

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

### Theorem 10

$\text{CET}$  is decidable.

Proof?

What about space?

### Definition 11

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

### Theorem 12

$\text{CES}$  is decidable.

### Theorem 13

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:  $L_\infty \notin \mathcal{R}$ .
- We want to show that  $L_\infty \notin \mathcal{RE}$ .

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 14 ( $B_{M,w}$ )

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**  $\implies B_{M,w}$  accepts all  $y$ 's  $\implies L(B_{M,w}) = \Sigma^*$   
 $\implies \langle B_{M,w} \rangle \in L_\infty$ .
  - $M(w)$  **halts** after  $k$  steps  $\implies B_{M,w}$  accepts only  $y$ 's of length **smaller** than  $k$   $\implies L(B_{M,w})$  is **finite**  $\implies \langle B_{M,w} \rangle \notin L_\infty$ .
- Hence,  $x \in \overline{H_{TM}} \iff f(x) \in L_\infty \implies \overline{H_{TM}} \leq_m L_\infty \implies L_\infty \notin \mathcal{RE}$

## Section 3

# RE-Completeness



## $\mathcal{RE}$ -Completeness

### Question 15

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 16 ( $\mathcal{RE}$ -complete)

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

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

### Question 17

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

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

### Theorem 18

$A_{TM}$  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?).



## Section 4

# 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).
- Does a **context free grammar** generate  $\Sigma^*$ ?
- 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)  $uaq_i b v$  yields (i.e.,  $\implies$ )  $uq_j a c v$ , if  $\delta(q_i, b) = (q_j, c, L)$
- $uaq_i b v$  yields  $uacq_j v$  if  $\delta(q_i, b) = (q_j, c, R)$
- **Special case** (left end of tape):  $q_i b v$  yields  $q_j c v$  if  $\delta(q_i, b) = (q_j, c, L)$ .

## Computation Histories

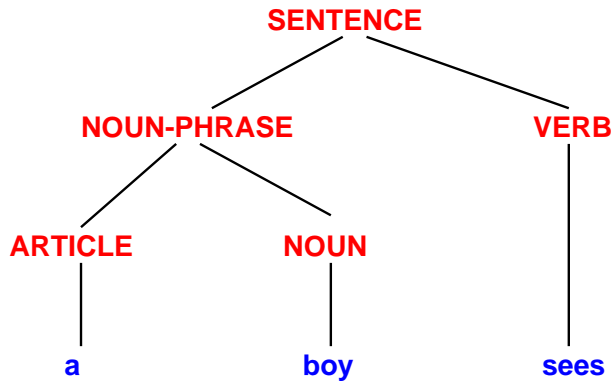
Let  $M$  be a TM and  $w$  an input string.

- An **accepting** computation is  $\#C_1\#C_2\#\dots\#C_\ell\#$ , where
  - 1  $C_1$  is the starting configuration of  $M$  on  $w$ ,
  - 2  $C_\ell$  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
  - ▶  $C_\ell$  is a rejecting configuration of  $M$ .

### Remark 19

- 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.

# Computation Histories and Emptiness of CFGs



## Emptiness of CFGs

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

### Algorithm 20

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  $A \rightarrow U_1 U_2 \dots U_k$ , and each  $U_i$  has already been marked.
- 4 **Accept**, if start symbol marked; otherwise **Reject**.

So  $E_{\text{CFG}}$  is decidable.

### Question 21

What about the complementary question: does a CFG generate **all** strings?

Namely, does  $\text{All}_{\text{CFG}} := \{ \langle G \rangle : G \text{ is a CFG and } L(G) = \Sigma^* \} \in \mathcal{R}$



# All<sub>CFG</sub> is Undecidable

## Theorem 22

All<sub>CFG</sub> is undecidable.

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

- 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) the “guesses” which condition is **violated**, and **verifies** the guessed violation.

### Algorithm 23 (D)

On input  $h = C_1 \# C_2 \dots \# C_\ell$ , 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  $C_\ell$  **not** an accepting configuration of  $M$ ?
- 4  $\exists i \in [\ell]$  s.t.  $C_i \not\Rightarrow C_{i+1}$  according to  $\delta$  – the transition function of  $M$ ?

The last condition is the tricky one to check.

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

### Algorithm 24 (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

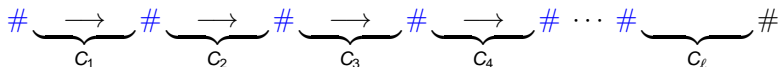
When  $C_i$  is popped from stack, it is in **reverse order**.

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.

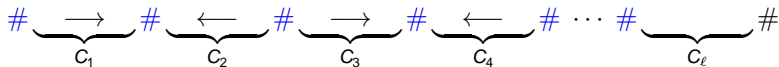
## Checking $C_i \Rightarrow C_{i+1}$ , take 2

- 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  $L(D)$  is either  $\Sigma^*$  or  $\Sigma^* \setminus \{z\}$ .
- This  $D$  has an equivalent (and efficiently described) CFG,  $G$ , namely  $L(D) = L(G)$ . So  $L(G)$  is either  $\Sigma^*$  or  $\Sigma^* \setminus \{z\}$ . The mapping  $\langle M, w \rangle \mapsto \langle G \rangle$  is thus a reduction from  $A_{\text{TM}}$  to  $\overline{\text{All}_{\text{CFG}}}$ .
- (Since  $A_{\text{TM}} \notin \mathcal{R}$ )  $\implies \overline{\text{All}_{\text{CFG}}} \notin \mathcal{R}$ .
- (Since  $\mathcal{R} = \overline{\mathcal{R}}$ )  $\implies \text{All}_{\text{CFG}} \notin \mathcal{R}$ .

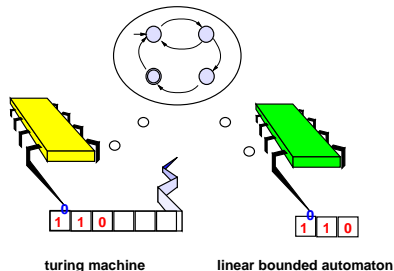


## Section 5

# 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



### Question 25

Why is it called “linear”?

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

## LBA's 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**.



## Acceptance for LBAs – $A_{LBA}$

$$A_{LBA} = \{ \langle M, w \rangle : M \text{ is an LBA that accepts } w \}$$

### Question 26

Is  $A_{LBA}$  decidable?

### Theorem 27

$A_{LBA}$  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 28

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)



### Algorithm 29

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  $qng^n$  steps, or until it halts (whichever comes first)
- 4 **Accept** if  $M$  has halted and **accepted**; otherwise, **Reject**

## Wholeness for LBAs – $All_{LBA}$

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

### Question 30

Is  $All_{LBA}$  decidable?

### Theorem 31

$All_{LBA}$  is undecidable.

Proof's idea: Same as in  $All_{CFG}$ : the computation we did using a PDA can be done **deterministically** by an **LBA**.

## Emptiness for LBAs – All<sub>LBA</sub>

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

### Question 32

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

### Theorem 33

$\text{EMPTY}_{\text{LBA}}$  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  $L(B_{M,w}) \neq \emptyset$ .  
 $\implies A_{\text{TM}} \leq_m \overline{\text{EMPTY}_{\text{LBA}}} \implies \overline{\text{EMPTY}_{\text{LBA}}} \notin \mathcal{R} \implies \text{EMPTY}_{\text{LBA}} \notin \mathcal{R}.$

## The LBA $B_{M,w}$

### Algorithm 34 (The LBA, $B_{M,w}$ )

On input  $x$

- 1 Split  $x$  according to the  $\#$  delimiters into  $C_1, C_2, \dots, C_\ell$ .
- 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  $C_\ell$  is an **accepting configuration**
  - 4 Every  $C_{i+1}$  **follows** from  $C_i$  according to  $M$

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

### Algorithm 35 (Does $C_i \Rightarrow C_{i+1}$ ?)

- 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  $L(B_{M,w})$  is either **empty** or a singleton  $\{x\}$ .  
 $\implies \langle M, w \rangle \in A_{\text{TM}} \iff \langle B_{M,w} \rangle \in \overline{\text{EMPTY}_{\text{LBA}}}$ .
- The reduction  $\langle M, w \rangle \mapsto B_{M,w}$  is **computable**  
 $\implies A_{\text{TM}} \leq_m \overline{\text{EMPTY}_{\text{LBA}}} \implies \overline{\text{EMPTY}_{\text{LBA}}} \notin \mathcal{R} \implies \text{EMPTY}_{\text{LBA}} \notin \mathcal{R}$ .
- $B_{M,w}$  is the **mirror** of the machine to that we used for proving  $A_{\text{TM}} \leq_m \text{All}_{\text{LBA}}$ .

### Question 36

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

## Section 6

# 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 37 (Unrestricted Grammar for $\{a^n b^n c^n\}$ )

- Generate the variable sequence  $L(ABC)^n$ :

$$S \rightarrow LT|\epsilon;$$

$$T \rightarrow ABCT|\epsilon;$$

- Sort the  $\{A, B, C\}$  and get  $LA^k B^k C^k$ .

$$BA \rightarrow AB;$$

$$CB \rightarrow BC;$$

$$CA \rightarrow AC;$$

- Replace the variables by terminals.

$$LA \rightarrow a;$$

$$aA \rightarrow aa;$$

$$aB \rightarrow ab;$$

$$bB \rightarrow bb;$$

$$bC \rightarrow bc;$$

$$cC \rightarrow cc;$$

## The Class $UG$

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

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

### Theorem 38

$$UG = RE$$

We show that:

- $UG \subseteq RE$
- $RE \subseteq UG$

## Proving $UG \subseteq RE$

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.

## 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  $\epsilon$ .

## 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  $w[q_0w]$ :

$$S \rightarrow T[q_0]$$

For all  $a \in \Sigma$ :

$$T \rightarrow aTA_a | \epsilon; A_a[q_0 \rightarrow [q_0A_a; A_ab \rightarrow bA_a; A_a] \rightarrow a]$$

- Simulate  $M(w)$ :

$$\delta(q, a) = (q', b, R) \Rightarrow qa \rightarrow bq'$$

$$\delta(q, a) = (q', b, L) \Rightarrow cqa \rightarrow q'cb; [qa \rightarrow [q'b; q] \rightarrow q_⊥]$$

- Accepting – derive  $w$  from  $w[uq_av]$ :

$$q_a \rightarrow E_L E_R$$

$$aE_L \rightarrow E_L; [E_L \rightarrow \epsilon$$

$$E_R a \rightarrow E_R; E_R] \rightarrow \epsilon$$

## Section 7

# Primes, Yet Another Example

# Primes

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

## Theorem 41

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

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

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

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



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

Let  $P$  be a decider for  $\text{Primes}$  (?). Let  $f$  be the function computed by the following TM

### Algorithm 42 ( $F$ )

on input  $\langle M, w \rangle$ , output  $B_{M,w}$

### Definition 43 ( $B_{M,w}$ )

On input  $x$

- Emulate  $P(x)$  and **Accept** if  $P$  accepts
- Emulate  $M(w)$  and **Accept** if  $M$  accepts

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

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

### Question 44

What property of  $\text{Primes}$  have we used?