# Computational Models — Lecture 10[1]

## Handout Mode

Iftach Haitner.

Tel Aviv University.

January 2, 2017

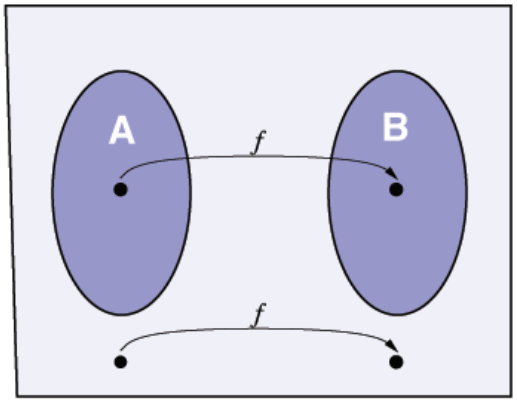
[1] Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.

# Talk Outline

- Rice's theorem and friends
- Controlled executions
- Configuartion histories
- $\mathcal{RE}$-Completeness

- Sipser's book, 5.1, 5.3

# Mapping reductions (review)




$$x \in \mathcal{A} \iff f(x) \in \mathcal{B}$$

Converts questions about membership in $\mathcal{A}$ to membership in $\mathcal{B}$.

**Theorem 1**

*If $\mathcal{A} \leq_m \mathcal{B}$ and $\mathcal{B} \in \mathcal{R}$, then $\mathcal{A} \in \mathcal{R}$.*

**Theorem 2**

*If $\mathcal{A} \leq_m \mathcal{B}$ and $\mathcal{B} \in \mathcal{RE}$, then $\mathcal{A} \in \mathcal{RE}$.*

**Corollary 3**

*If $\mathcal{A} \leq_m \mathcal{B}$ and $\mathcal{A} \notin \mathcal{R}$ [resp., $\mathcal{A} \notin \mathcal{RE}$], then $\mathcal{B} \notin \mathcal{R}$ [resp., $\mathcal{B} \notin \mathcal{RE}$] .*

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 $\mathcal{RE}$ – 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 4**

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

# Rice's Theorem

**Theorem 5**

*For non-empty $\mathcal{C} \subsetneq \mathcal{RE}$, it holds that*
$\mathsf{L}_{\mathcal{C}} = \{\langle M \rangle : \ M \text{ is a TM and } \mathsf{L}(M) \in \mathcal{C}\} \notin \mathcal{R}$.

Note that both $\mathsf{L}_{\emptyset}$ and $\mathsf{L}_{\mathcal{RE}}$ are in $\mathcal{R}$.

Proof's idea: Mapping reduction from $\mathsf{H}_{\mathsf{TM}}$.

Given $M$ and $w$, we construct $M_w^{\mathcal{C}}$ such that:

- If $M$ halts on $w$, then $\langle M_w^{\mathcal{C}} \rangle \in \mathsf{L}_{\mathcal{C}}$.
- If $M$ does not halt on $w$, then $\langle M_w^{\mathcal{C}} \rangle \notin \mathsf{L}_{\mathcal{C}}$.

## Proving Rice's Theorem

Assume $\emptyset \notin \mathcal{C}$ (Will take care of the other case later).

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

**Algorithm 6 ($M_w^{\mathcal{C}}$)**

On input $y$:

1. Emulate $M(w)$.
2. Emulate $M_{\mathcal{A}}(y)$:

   Accept, if $M_{\mathcal{A}}$ accepts; reject, if $M_{\mathcal{A}}$ rejects.

Let $f(\langle M, w \rangle) := \langle M_w^{\mathcal{C}} \rangle$

**Claim 7**

$f$ is a mapping reduction from $\mathrm{H}_{\mathrm{TM}}$ to $\mathrm{L}_{\mathcal{C}}$

Since $f$ is clearly computable (?), it is left to show that

$$\langle M, w \rangle \in \mathrm{H}_{\mathrm{TM}} \iff f(\langle M, w \rangle) \in \mathrm{L}_{\mathcal{C}}$$

$\langle M, w \rangle \in \mathrm{H_{TM}} \iff f(\langle M, w \rangle) \in \mathrm{L}_{\mathcal{C}}$

Proof:

- If $\langle M, w \rangle \in \mathrm{H_{TM}}$, then $M_w^{\mathcal{C}}$ gets to Step 2, and emulates $M_{\mathcal{A}}(y)$. Hence $\mathrm{L}(M_w^{\mathcal{C}}) = \mathcal{A} \in \mathcal{C} \Longrightarrow M_w^{\mathcal{C}} \in \mathrm{L}_{\mathcal{C}}$.
- Otherwise, $M_w^{\mathcal{C}}$ never gets to Step 2. Hence $\mathrm{L}(M_w^{\mathcal{C}}) = \emptyset \notin \mathcal{C} \Longrightarrow M_w^{\mathcal{C}} \notin \mathrm{L}_{\mathcal{C}}$.
- Thus, $\langle M, w \rangle \in \mathrm{H_{TM}}$ iff $\langle M_w^{\mathcal{C}} \rangle \in \mathrm{L}_{\mathcal{C}}$.

♣

## Completing the proof: The case $\emptyset \in \mathcal{C}$

Since $\mathcal{C} \subsetneq \mathcal{RE}$ and $\mathcal{C}$ is non empty, it follows that $\emptyset \notin \overline{\mathcal{C}} \subsetneq \mathcal{RE}$, and $\overline{\mathcal{C}}$ is non-empty.

$\Longrightarrow$ (by the first part of the proof) $L_{\overline{\mathcal{C}}} \notin \mathcal{R}$

$\Longrightarrow$ $\overline{L_{\mathcal{C}}} \notin \mathcal{R}$ (?)

$\Longrightarrow$ $L_{\mathcal{C}} \notin \mathcal{R}$

## Example: $\text{Primes}_{\text{TM}} \notin \mathcal{R}$

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

**Theorem 8**

$\text{Primes} \in \mathcal{R}$

Proof: (?)

**Theorem 9**

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

Proof: $\text{L}_{\text{Primes}}$ is a non triviall subset of $\mathcal{RE}$.

# Reflections on Rice theorem

- 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 says nothing on membership in $\mathcal{RE}$
- **Rice's Theorem is a powerful tool, but use it with care!**

Section 2

# Proving Non-Enumarability

## Proving non-enumarability

### Theorem 10

*For non-empty* $\mathcal{C} \subseteq (\mathcal{R} \setminus \{\Sigma^*\})$*, it holds that* $\mathsf{L}_{\mathcal{C}} = \{\langle M \rangle : M \text{ is a TM and } \mathsf{L}(M) \in \mathcal{C}\} \notin \mathcal{RE}$ .

Corollary: $\mathsf{Primes}_{\mathsf{TM}} \notin \mathcal{RE}$.

Proof: We show that $\overline{\mathsf{A}_{\mathsf{TM}}} \leq_m \mathsf{L}_{\mathcal{C}}$.

Let $\mathsf{D}$ be a decider for some $\mathcal{A} \in \mathcal{C}$.

Define $f(x = \langle M, w \rangle)$ to return $B_{M,w}$ if $x$ is well formatted, and $\mathsf{D}$ o/w.

### Definition 11 ($B_{M,w}$)

On input $x$:

- Emulate $\mathsf{D}(x)$ and accept if $\mathsf{D}$ does.
- Emulate $M(w)$ and accept if $M$ does.

- $\langle M, w \rangle \in \overline{\mathsf{A}_{\mathsf{TM}}} \implies \mathsf{L}(B_{M,w}) = \mathcal{A} \in \mathcal{C}$
- $\langle M, w \rangle \notin \overline{\mathsf{A}_{\mathsf{TM}}} \implies \mathsf{L}(B_{M,w}) = \Sigma^* \notin \mathcal{C}$. □

What about $\mathsf{All}_{\mathsf{TM}} := \mathcal{L}_{\Sigma^*}$?

## Rice theorem vs. Thm 10

Rice speaks about non-membership in $\mathcal{R}$ where Thm 10 about non-membership in $\mathcal{RE}$.

But is it always the case that when Rice is applicable then so is Thm 10 ?

- Let $C_a = \{\mathcal{L} \in \mathcal{RE} : a \in \mathcal{L}\}$.
- By definition, $\mathcal{L}_{C_a} = \{\langle M \rangle : M \text{ is a TM and } \mathrm{L}(M) \in C_a\}$
- Note that $\mathcal{L}_{C_a} = \{\langle M \rangle : a \in \mathcal{L}(M)\}$
- By Rice, $\mathcal{L}_{C_a} \notin \mathcal{R}$
- But clearly, $\mathcal{L}_{C_a} \in \mathcal{RE}$
- Thm 10 is not applicable here since $C_a \notin \mathcal{R}$.

Section 3

# Controlled Executions

## Bounded time and space

In the following a TM stands for a *single-tape deterministic* TM.

**Definition 12**

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

Is $\text{CET} \in \mathcal{R}$?

**Theorem 13**

$\text{CET} \in \mathcal{R}$.

Proof?

**Definition 14**

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

**Theorem 15**

CES *is decidable.*

## Proving CES $\in \mathcal{R}$

How to check that the computation will not terminate?

Proof: $m = |Q| \cdot |\Gamma|^k \cdot k$ is a bound on the number of $k$-cell configurations of $M$.

**Algorithm 16**

On input $\langle M, w, k \rangle$.

1. Emulate $M(w)$ while maintaining a step counter.
   Counter is incremented by 1 per each simulated step (of $M$).
2. Reject if counter reaches $m + 1$ or $M$ uses more than $k$ cells.
3. Accept if $M$ does.

Correctness follows since if $M$ does not accept $w$ within $m$ steps (using $k$ cells), then it will never halt ♣

## Proving $\text{All}_{\text{TM}} = \mathcal{L}_{\Sigma^*} = \{\langle M\rangle : M \text{ is a TM and } \text{L}(M) = \Sigma^*\} \notin \mathcal{RE}$

Proof: We show that $\overline{\text{H}_{\text{TM}}} \leq_m \text{All}_{\text{TM}}$. Namely, we define a computabale $f$ with $f(\langle M, w\rangle) = \langle B_{M,w}\rangle$ such that

- $M(w)$ halts $\implies$ $\text{L}(B_{M,w}) \neq \Sigma^*$.
- $M(w)$ does not halt $\implies$ $\text{L}(B_{M,w}) = \Sigma^*$.

**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.

- $M(w)$ halts after $k$ steps $\implies$ $B_{M,w}$ accepts only $y$'s of length smaller than $k$ $\implies$ $\text{L}(B_{M,w})$ is finite $\implies$ $\text{L}(B_{M,w}) \neq \Sigma^*$.
- $M(w)$ does not halt $\implies$ $B_{M,w}$ accepts all $y$'s $\implies$ $\text{L}(B_{M,w}) = \Sigma^*$.

♣

Section 4

# Computation Histories

# Reduction via Computation Histories

Important technique for proving undecidability. Examples

- Basis for proof of undecidability in Hilbert's tenth problem (given a polynomial with integer coefficients, does it have a solution over the integers?).
- Enables showing that
  $\text{All}_{\text{CFG}} := \{\langle S \rangle : S \text{ is a CFG and } \text{L}(S) = \Sigma^*\} \notin \mathcal{RE}$

  Recall that $\text{EMPTY}_{\text{CFG}} := \{\langle S \rangle : S \text{ is a CFG and } \text{L}(S) = \emptyset\} \in \mathcal{R}$.
- Proof: we use computation histories to show that
  $\overline{\text{A}_{\text{TM}}} \leq_m \text{All}_{\text{PDA}} := \{\langle P \rangle : P \text{ is a PDA and } \text{L}(P) = \Sigma^*\}$.
- Hence, $\text{All}_{\text{PDA}} \notin \mathcal{RE}$.

# Reminder: Configurations

- Configuration: $1011q_70111$, means:
  - state is $q_7$
  - LHS of tape is $1011$
  - RHS of tape is $0111$
  - head is on RHS $0$
- Yield relation
  - $uaq_ibv \implies uq_jacv$, if $\delta(q_i, b) = (q_j, c, \mathrm{L})$
  - $uaq_ibv \implies uacq_jv$, if $\delta(q_i, b) = (q_j, c, \mathrm{R})$
  - Special cases: $q_ibv$ and $uaq_i$
- Special type of configurations: starting, accepting, rejecting, halting
- $h = C_1 \# C_2 \ldots \# C_\ell$ is accepting configuration history of $M$ on $w$,[2] if
  1. $C_1$ is the starting configuration of $M$ on $w$
  2. $C_\ell$ is an accepting configuration of $M$
  3. $\forall i \in [\ell]$: $C_i \implies C_{i+1}$ according to $M$

[2] In Lecture 1, we called such $C_1, \ldots, C_\ell$, an *accepting valid sequence of configurations with respect to $M$ and $w$*.

## Warmup: $\overline{A_{TM}} \leq_m All_{TM}$ (reproving that $All_{TM} \notin \mathcal{RE}$)

Proof: We define computable $f$ such that the TM $B_{M,w} = f(\langle M, w \rangle)$ has the following properties:

1. if $M$ does not accept $w$, then $L(B_{M.w}) = \Sigma^*$
2. if $M$ does accept $w$, then $L(B_{M.w}) \neq \Sigma^*$

$B_{M,w}$ accepts all strings but the *accepting configuration history* of $M$ on $w$. (accepts all strings if $w \notin L(M)$)

**Algorithm 18 ($B_{M,w}$)**

Accepts input $h = C_1 \# C_2 \ldots \# C_\ell$, if one of the followings holds:

1. $C_1$ not the starting configuration of $M$ on $w$
2. $C_\ell$ is not an accepting configuration of $M$
3. $\exists i \in [\ell - 1]$ s.t. $C_i \not\Rightarrow C_{i+1}$ according to $M$

It is easy to see that $f$ is computable

## $\overline{A_{TM}} \leq_m All_{PDA}$

Proof: We define computable $f$ such that **PDA** the $P_{M,w} = f(\langle M, w \rangle)$ has the following properties:

1. if $M$ does not accept $w$, then $L(P_{M.w}) = \Sigma^*$
2. if $M$ does accept $w$, then $L(P_{M.w}) \neq \Sigma^*$

$P_{M,w}$ accepts all strings but the accepting configuation history of $M$ on $w$.

**Algorithm 19 ($P_{M,w}$)**

Accepts input $h = C_1 \# C_2 \ldots \# C_\ell$, f one of the followings holds

1. $C_1$ not the starting configuration of $M$ on $w$
2. $C_\ell$ is not an accepting configuration of $M$
3. $\exists i \in [\ell - 1]$ s.t. $C_i \not\Rightarrow C_{i+1}$ according to $M$

The (only) hard part is checking $C_i \not\Rightarrow C_{i+1}$ (the PDA will "guess" the right $i$ if such exists)

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

**Algorithm 20 (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 next slide 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.

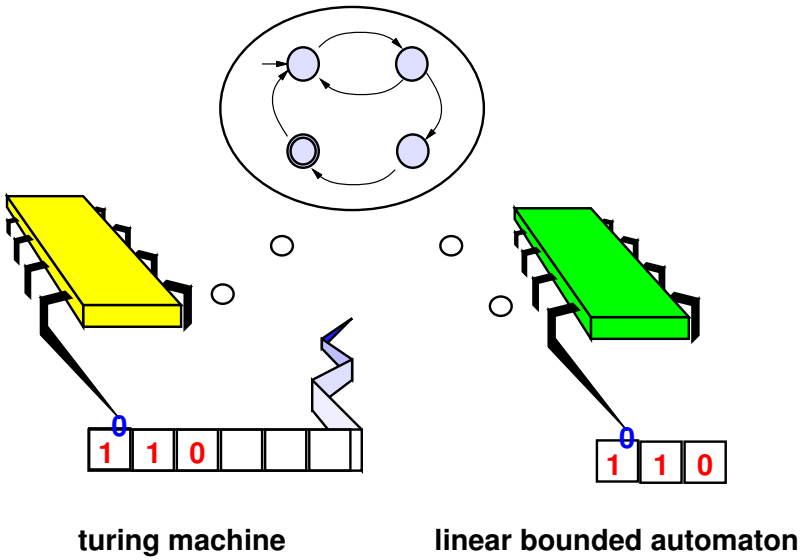
Section 5

# Linear Bounded Automaton

# Linear Bounded Automaton – LBA

- A restricted form of TM, that on input $w$ uses space (at most) $|w|$: it never changes the content of th first blank ($\sqcup$) cell, and never moves right to that cell.

  More formally, $\delta(\sqcup, q) = (\cdot, \sqcup, L)$ for any $q \in Q$.
- Size of input determines size of memory.



# Why “linear"?

**Question 21**

Why such machines 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?)
  - $EMPTY_{CFG}$ (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 decided by LBAs!

# Acceptance for LBAs – $A_{LBA}$

$$A_{LBA} = \{\langle M, w\rangle : M \text{ is an LBA } \wedge w \in \mathcal{L}(M)\}$$

**Question 22**

Is $A_{LBA}$ decidable?

**Theorem 23**

$A_{LBA}$ *is decidable.*

Proof's idea: Similar to controlled executions.

- `Reject` if $M$ is not an LBA. (?)
- Emulate $M(w)$ for limited number of steps (number depends on $M$ and $|w|$), accepts if $M$ does.

# LBAs have bounded number of configurations

**Lemma 24**

*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 25

On input $\langle M, w \rangle$.

1. `Reject` if $M$ is not an LBA.
2. Emulate $M(w)$ while maintaining a step counter.
3. Counter incremented by 1 per each simulated step (of $M$).
4. Keep emulating $M$ for $qng^n + 1$ steps, or until it halts (whichever comes first)
5. Accept if $M$ has halted and accepted; otherwise, Reject

# Emptiness for LBAs – $\text{EMPTY}_{\text{LBA}}$

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

**Question 26**

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

**Theorem 27**

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

Proof's idea: Show that $A_{\text{TM}} \leq_m \overline{\text{EMPTY}_{\text{LBA}}} \implies \overline{\text{EMPTY}_{\text{LBA}}} \notin \mathcal{R} \implies \text{EMPTY}_{\text{LBA}} \notin \mathcal{R}$.

Proof uses computation histories, similar to proving that $\overline{A_{\text{TM}}} \leq_m \text{All}_{\text{PDA}}$.

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

Section 6

# RE-Completeness

# $\mathcal{RE}$-Completeness

**Question 28**

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 29 ($\mathcal{RE}$-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.

# An $\mathcal{RE}$-complete language

**Question 30**

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

**Theorem 31**

$A_{TM}$ *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 $A_{TM}$.

♣

# Other $\mathcal{RE}$-Complete problems

## Question 32

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

## Observations 33

Reductions are transitive:

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

## Theorem 34

*Let* L *be a language such*

1. $L \in \mathcal{RE}$
2. $A_{TM} \leq_m L$

*then L is* $\mathcal{RE}$*-complete.*

Hence, $H_{TM}$ and $H_{TM,\varepsilon}$ and ..., are all $\mathcal{RE}$-complete.

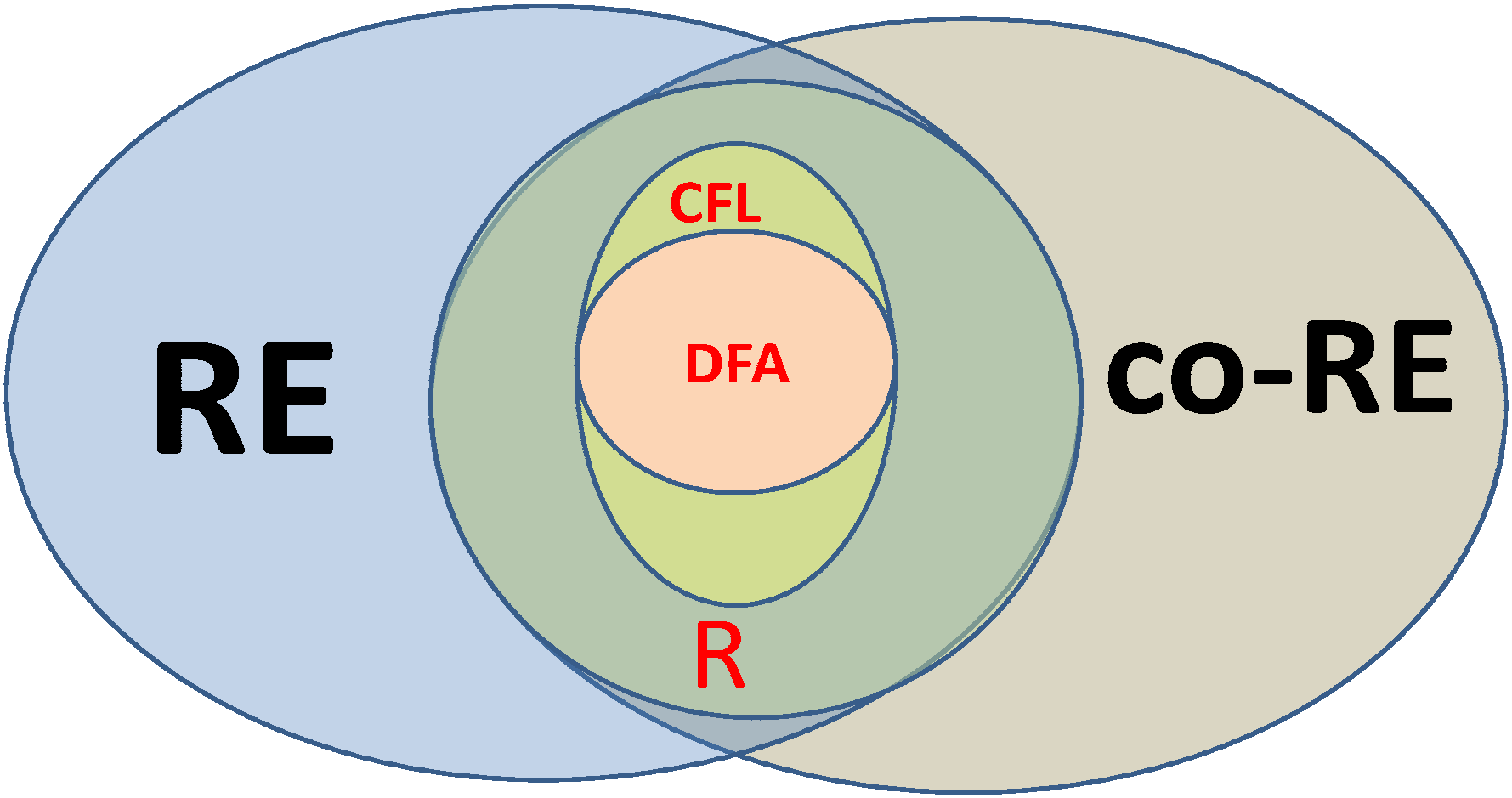
# Between $\mathcal{R}$ and $\mathcal{RE}$-complete

- Are there languages in $\mathcal{RE} \setminus \mathcal{R}$ that are not $\mathcal{RE}$-complete?
- Yes, but not natural ones. See work of Emil Post.

Section 7

# Computability, Summary

# World of Languages



# Summary

- Turing Machine - a universal computational model
- Language classes we considered: $\mathcal{R}$, $\mathcal{RE}$ and $co-\mathcal{RE}$.
- Undecidable languages (not in $\mathcal{R}$)
  - Acceptance/Halting problem
  - Any non-trivial property of a program (Rice Theorem)
  - Questions with respect to Grammars.
  - Many more exists ...
- Non-enumerable languages (not in $\mathcal{RE}$)
  - Some non-trivial property of a program
  - Much more exists ...

- What does this imply to verification of software and hardware?
- Well...