# Computational Models — Lecture 10[1]

## Handout Mode

Nachum Dershowitz & Yishay Mansour.

Tel Aviv University.

June 5–7, 2017

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

# Talk Outline

- Busy Beaver
- Kolmogorov Complexity
- Controlled executions
- Configuration histories
- $\mathcal{RE}$-Completeness

- Sipser's book, 6.4, 5.1, 5.3

Section 1

# Busy Beaver

# The *Busy Beaver*

(taken from `http://www.saltine.org/joebeaver1.jpg`)

## The *Busy Beaver*

We focus on one tape TMs, with $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \sqcup\}$.

**Definition 1 ($S_n$ and $BB(n)$)**

For $n \in \mathbb{N}$, let

- $S_n = \{$ all $n$-state TM's that halt on $\varepsilon\}$.
- $\mathrm{BB}(n) = \max_{M \in S_n}\{$ # of steps taken by $M$ on input $\varepsilon\}$.

- The set $S_n$ is finite (under standard encoding)
- Every $M \in S_n$ runs for finitely many steps on $\varepsilon$ .
- $\mathrm{BB}(n)$ is a total function from $\mathbb{N}$ to $\mathbb{N}$ (in particular, $\mathrm{BB}(n) \in \mathbb{N}$ for every $n \in \mathbb{N}$).

Values of BB (size not including accept and reject states):

| size | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| BB | 6 | 21 | 107 | $\geq 47,176,870$ | $\geq 7.4 \times 10^{36534}$ |

# The *Busy Beaver* function is not computable

**Theorem 2**

BB *is not computable.*

Proof: Assume BB is computable by the TM $R$ and consider the undecidable language $H_{TM,\varepsilon} = \{\langle M \rangle : M \text{ is a TM that halts on } \varepsilon\}$.

**Algorithm 3 ($S$)**

On input $\langle M \rangle$

1. Let $m$ be # of states in $M$, and let $n = R(m)$.
2. Emulate $M$ on $\varepsilon$ for $n+1$ steps.
3. Accept if $M$ halts; otherwise reject

Note that if $M$ did not halt in $n+1$ steps, then it will never halt!

**Claim 4**

$S$ decides $H_{TM,\varepsilon}$.

# The bounded *Busy Beaver* function is computable

## Definition 5

For $d \in \mathbb{N}$, define the function $\mathrm{BB}_d : \mathbb{N} \mapsto \mathbb{N}$ as

$$\mathrm{BB}_d(n) := \begin{cases} \mathrm{BB}(n), & n \leq d, \\ 0, & \text{otherwise.} \end{cases}$$

## Theorem 6

*The function $\mathrm{BB}_d$ is computable for every $d \in \mathbb{N}$.*

Proof's idea: "Hardwire" the values $\mathrm{BB}(1), \ldots, \mathrm{BB}(d)$ into a TM to compute $\mathrm{BB}_d$.

Section 2

# Kolmogorov Complexity

# Description Length and Information

Consider the two (equal length – 28 bits each) strings

```
0101010101010101010101010101
0010110011101010100110001111
```

Which of these two strings has more information?

This raises the difficult question of what information means, and how can it be measured.

Following Kolmogorov, we will measure the information of a string by means of its description length.

# Information and Description Length

The motivation for Kolmogorov complexity is that phenomena with shorter explanations are typically less complex than phenomena with longer explanations.

Consequently, we will say that strings with longer description length are more informative than those with shorter description.

Of course, we should still define what description length means.

An alternative route (not taken here) is to consider how much a string can be compressed.

## Kolmogorov Complexity

In this part of the lecture, we view all TMs as computing functions.

In particular, we can talk about the Universal TM for computing functions.

### Definition 7

Let $M$ by a TM, and $f_M$ be the function it computes.
The Kolmogorov Complexity of a string $x$ with respect to $M$, $K_M(x)$, is defined as the length of the shortest string $y$ satisfying $f_M(y) = x$.

If there is no such $y$, we define $K_M(x) = \infty$.

## Kolmogorov Complexity

Hey, this definition is no good. It is totally arbitrary and depends on the particular choice of machine $M$. Moreover, some strings may have $K_M(x) = \infty$, which is counter intuitive.

Well gidday, mates, and no worries . We will immediately show how this can be fixed.

**Theorem 8**

*Let $U$ be a universal Turing machine. For every Turing machine, $M$, there is a constant $c_M$ (depending on $M$ alone) such that for every $x \in \Sigma^*$, $K_U(x) \leq K_M(x) + c_M$.*

Proof: Let $y$ be a shortest string such that $f_M(y) = x$. Then for the universal TM, $U$,

$$f_U(\langle M, y \rangle) = f_M(y) = x \ .$$

Using prefix-free encodings for TMs, $\langle M, y \rangle$ is simply the concatenation of $\langle M \rangle$, followed by the string $y$. So we get

$$K_U(x) \leq |y| + |\langle M \rangle| = K_M(x) + |\langle M \rangle| \ .$$

So the theorem holds where $c_M = |\langle M \rangle|$. ♣

## Corollary 9

*If both $U_1$ and $U_2$ are universal Turing machines, then there is a constant $c$ such that for every string $x$,*

$$|K_{U_1}(x) - K_{U_2}(x)| < c \ .$$

So we can take any universal TM, $U$, define

$$K(x) = K_U(x) \ ,$$

and refer to this measure as "Kolmogorov complexity of the string $x$.

We now show that for every string $x$, $K(x)$ equals at most $x$'s length plus a constant.

**Theorem 10**

*There is a constant $c$ such that for every string $x$, $K(x) \leq |x| + c$.*

Pay attention to the order of quantifiers in the statement.

Proof: Let $M_{ID}$ be a TM computing the identity function $f(x) = x$ (*e.g.* a TM that halts immediately).
Obviously for any string $x$, $K_{M_{ID}}(x) = |x|$. By previous theorem, there is a constant $c$ such that for any string $x$,

$$K(x) = K_U(x) \leq K_{M_{ID}}(x) + c = |x| + c$$

♣

# Kolmogorov Complexity

Are there strings whose Kolmogorov complexity is substantially smaller than their own length?

- $K(xx) \leq K(x) + c$
- $K(1^n) \leq \log(n) + c$
- $K(1^{2^n}) \leq \log(n) + c$

But these strings with very concise description are rare.

# Kolmogorov Complexity

A simple counting argument gives

**Theorem 11**

*For every integer $c \geq 1$, the number of strings in $\{0,1\}^n$ for which $K(x) \leq n - c$ is at most $2^n/2^{c-1}$.*

Proof: In $\{0,1\}^*$ there is $1$ string of length $0$, $2$ string of length $1$, $\ldots$, $2^{n-c}$ string of length $n-c$. The total number of strings up to length $n-c$ is $2^{n+1-c} - 1 < 2^n/2^{c-1}$. So the number of possible descriptions $y$ of length $\leq n-c$ is no more than $2^n/2^{c-1}$. This implies that the number of length $n$ strings whose description length is $c$ shorter than their own length is at most $2^n/2^{c-1}$. ♣

## Kolmogorov Complexity Uncomputable

The function $K(\cdot)$ is total (defined for every string $x$) and unbounded. But is it computable?

**Theorem 12**

*The function $K(\cdot)$ is not computable.*

**Proof:** By contradiction. For every $n$ let $y_n$ be the lexicographically first string $y$ satisfying $K(y) > n$. Then the sequence $\{y_n\}_{n=1}^{\infty}$ is well defined.

Assume $K$ is computable. We'll show this implies the existance of a constant $c$ such that for every $n$, $K(y_n) < \log(n) + c$.

# Kolmogorov Complexity Uncomputable

Consider the following TM, $M$: On input $n$ (in binary), $M$ generates, one by one, all binary strings $x_0, x_1, x_2, \ldots$ in lexicographic order. For each $x_i$ it produces, $M$ computes $K(x_i)$.

If $K(x_i) > n$, the TM, $M$, outputs $y = x_i$ and halts. Otherwise, the TM, $M$, continues to examine the lexicographically next string, $x_{i+1}$.

Since the function $K$ is unbounded, it is guaranteed that $M$ will eventually reach a string $x$ satisfying $K(x) > n$.

# Kolmogorov Complexity is not computable

Conclusion: On input (in binary) $n$, the TM, $M$, outputs $y_n$ (the lexicographically first string whose Kolmogorov complexity exceeds $n$, $K(x) > n$).

Length of $n$ is $\log_2(n)$. So $K_M(y_n) \leq \log_2(n)$.

We saw that there is a constant $c_M$ such that
for every $y$, $K(y) \leq K_M(y) + c_M$,
so for every $n$, $K(y_n) \leq \log_2(n) + c_M$.

By definition, for every $n$, $\quad n < K(y_n)$. Combining the last two inequalities, we get, for every $n$,

$$n < \log_2(n) + c_M \ .$$

But asymptotically $n$ grows faster than $\log_2(n) + c_M$. Contradiction to $K(\cdot)$ computability.

# Intuition for the proof

- Consider the paradox:
  What is the smallers number that cannot be describe in twelve words
- Similar issue for $y_n$.

Section 3

# Controlled Executions

## Bounded time and space

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

**Definition 13**

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

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

**Theorem 14**

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

Proof?

**Definition 15**

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

**Theorem 16**

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 17**

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 $M$ accepts, and Reject if $M$ rejects.

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

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

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

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

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

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 $k$ $\implies$ $\mathsf{L}(B_{M,w})$ is finite $\implies$ $\mathsf{L}(B_{M,w}) \neq \Sigma^*$.
- $M(w)$ does not halt $\implies$ $B_{M,w}$ accepts all $y$'s $\implies$ $\mathsf{L}(B_{M,w}) = \Sigma^*$.

♣

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
  $\mathsf{All}_{\mathsf{CFG}} := \{\langle S \rangle : S \text{ is a CFG and } \mathsf{L}(S) = \Sigma^*\} \notin \mathcal{RE}$

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

# 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 $M$ on $w$
  3. $\forall i \in [\ell]$: $C_i \implies C_{i+1}$ according to $M$ on $w$

[2] In Lecture 7, 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 19 ($B_{M,w}$)**

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 $M$ on $w$
3. $\exists i \in [\ell - 1]$ s.t. $C_i \not\Rightarrow C_{i+1}$ according to $M$ on $w$

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 20 ($P_{M,w}$)**

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 $M$ on $w$
3. $\exists i \in [\ell - 1]$ s.t. $C_i \not\Rightarrow C_{i+1}$ according to $M$ on $w$

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 21 (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.

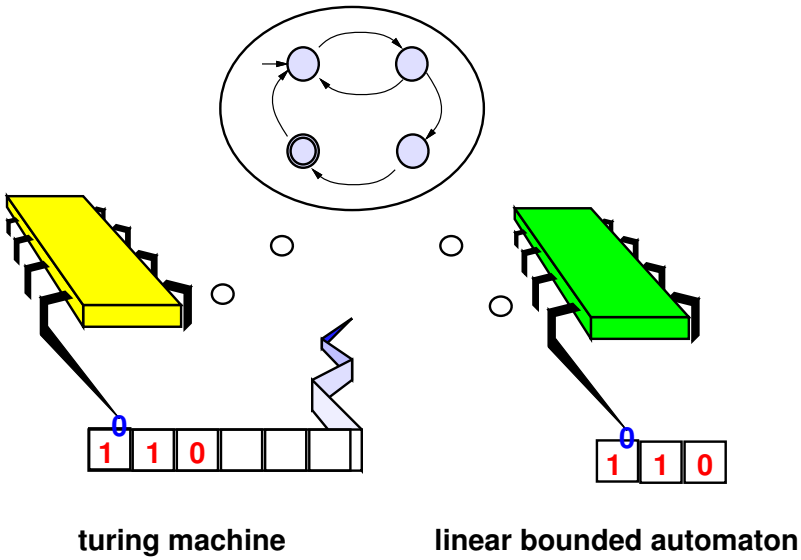
Section 5

# 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 22**

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 23**

Is $A_{LBA}$ decidable?

**Theorem 24**

$A_{LBA}$ *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 25**

*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 26

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

1. `Reject` if $M$ is not an LBA.
2. Emulate $M(w)$ while maintaining a step counter.
   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 – $\text{EMPTY}_{\text{LBA}}$

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

**Question 27**

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

**Theorem 28**

$\text{EMPTY}_{\text{LBA}}$ *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 $L(B_{M,w}) \neq \emptyset$.

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

**Algorithm 29 ($B_{M,w}$)**

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

1. $C_1$ is the starting configuration of $M$ on $w$
2. $C_\ell$ is an accepting configuration of $M$ on $w$
3. $\forall i \in [\ell - 1]$ $C_i \implies C_{i+1}$ according to $M$ on $w$

It is easy to see that $f$ is computable with LBA

If $M$ accepts $w$ then $L(B_{M,w}) = \{h\}$ where $h$ is the accepting computation of $M$ on $w$.
If $M$ does not accepts $w$ then $L(B_{M,w}) = \emptyset$.

Section 6

# RE-Completeness

# $\mathcal{RE}$-Completeness

**Question 30**

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 31 ($\mathcal{RE}$-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 32**

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

**Theorem 33**

$A_{TM}$ *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 34

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

## Observations 35

Reductions are transitive:

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

## Theorem 36

*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.

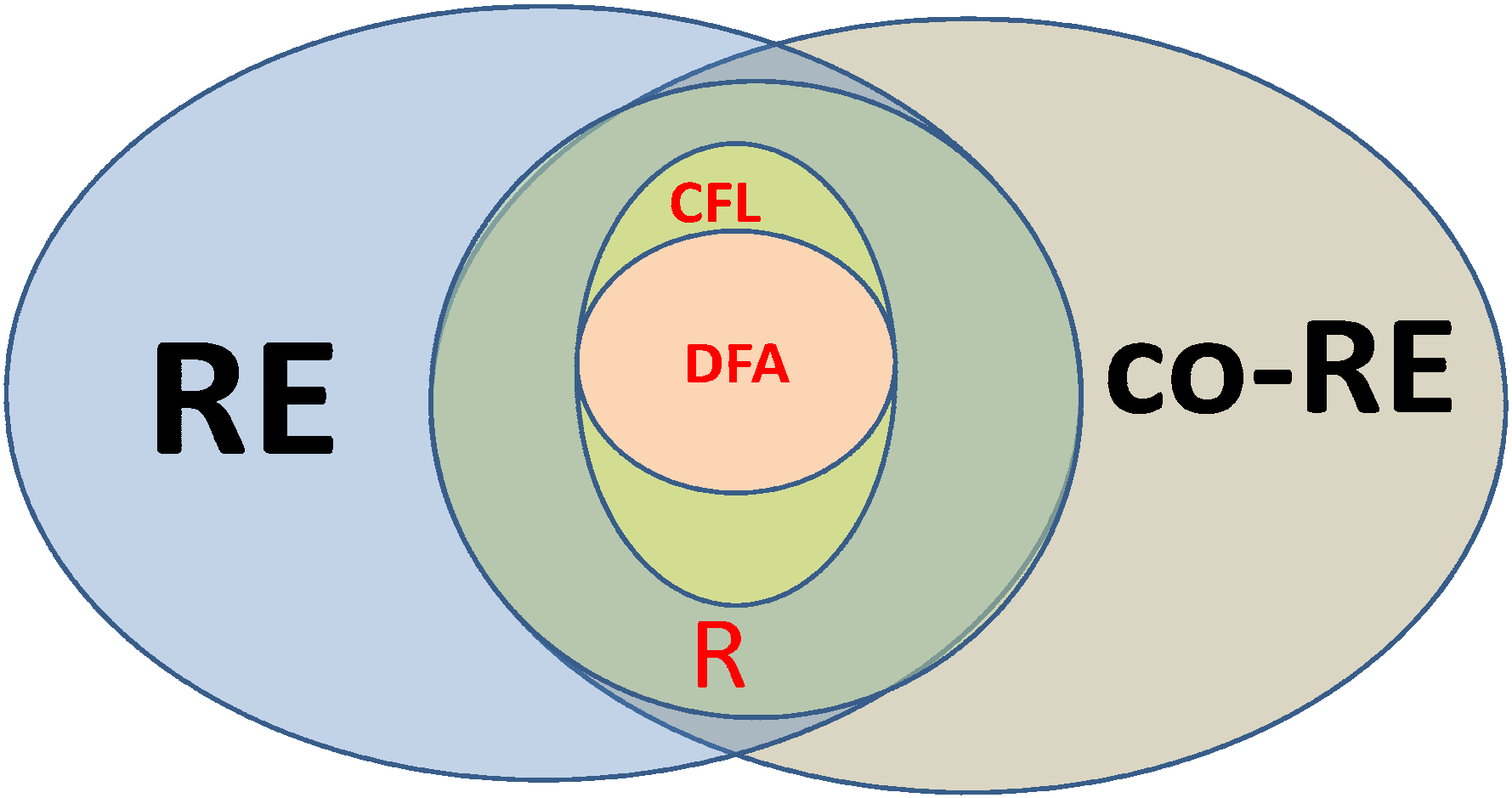
# Between $\mathcal{R}$ and $\mathcal{RE}$-complete

- 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...