

Computational Models — Lecture 9¹

Handout Mode

Nachum Dershowitz & Yishay Mansour.

Tel Aviv University.

May 24–29, 2017

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

Talk Outline

- ▶ Computable functions
 - ▶ Reducibility
 - ▶ Mapping reductions
 - ▶ Rice Theorem
 - ▶ Busy Beaver
- Sipser's book, [5.1](#) – [5.2](#)

Reminder

We have already

- ▶ Established Turing Machines as the **gold standard** of computers and computability ...
- ▶ Seen examples of decidable problems ...
- ▶ Saw that

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

and

$$H_{\text{TM}} = \{ \langle M, w \rangle : M \text{ is a TM and } M \text{ halts on input } w \}$$

are **undecidable**.

Today, we look at other computationally undecidable problems **via reductions** and introduce the techniques of **mapping reductions**.

Section 1

Computable Functions

Computable functions

Definition 1 (total computable functions)

A TM M computes a **total** function $f: \Sigma^* \mapsto \Sigma^*$, if when starting with an input w , M halts with (only) $f(w)$ written on tape.^a

^aThe definition naturally extends to functions of **more than one** variable, where some special separator symbol indicates end of one variable and beginning of next.

Computable functions are also called **recursive** functions.

Example, 1

Claim 2

All the “usual” arithmetic functions on integers are computable.

These include addition, subtraction, multiplication, division (quotient and remainder), exponentiation, roots (to a specified precision), modular exponentiation, greatest common divisor.

Even **non-arithmetic** functions, like logarithms and trigonometric functions, can be computed (to a specified precision), using Taylor expansion or other numeric mathematic techniques.

Exercise 3

Design a TM that on input $\langle m, n \rangle$, halts with $\langle m + n \rangle$ on tape.

Example, 2

A useful class of functions **modifies TM descriptions**. For example:

Algorithm 4

On input w :

If $w = \langle M \rangle$ for some TM M , return $\langle \tilde{M} \rangle$, where \tilde{M} is TM s.t.,

- ▶ $L(\tilde{M}) = L(M)$, and
- ▶ \tilde{M} does not halt on $x \notin L(\tilde{M})$

Otherwise, return ε .

Question 5

Is the function defined above total? computable?

Are all total functions computable?

No, $f(\langle M, w \rangle)$ that returns 1 if M accepts w , and 0 otherwise, is **not** computable.

Definition of \tilde{M}

Given $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, TM $\tilde{M} = (Q', \Sigma, \Gamma, \delta', q_0, q_a, q_r)$ is defined by

▶ $Q' = Q \cup \{q^*\}$

▶

$$\delta'(q, \sigma) := \begin{cases} (q^*, \sigma, R), & \delta(q, \sigma) = (q_r, \cdot, \cdot) \\ (q^*, \sigma, R) & q = q^* \\ (q', \sigma', D), & \delta(q, \sigma) = (q', \sigma', D) \wedge q \notin \{q_r, q^*\} \end{cases}$$

Section 2

Reducibility

Reducibility

- ▶ Finding your way around a new city, reduces to . . . obtaining a city map.
- ▶ Finding the median in an array, reduces to . . . sorting an array
- ▶ The core idea behind procedures

Reducibility, in our context

Involves two languages A and B .

Desired property: If A reduces to B , then any solution of B can be used to find a solution of A .

This property says **nothing** about solving A by itself, or B by itself.

but if A reduces to B , then A cannot be **harder** than B :

- ▶ if B is decidable, so is A .
- ▶ if A is undecidable, then B is undecidable.

We next use reductions and the undecidability of A_{TM} , to show the undecidability of several problems.

Reminder, H_{TM} is undecidable

- ▶ $A_{TM} = \{\langle M, w \rangle : M \text{ is a TM that accepts } w\}$
- ▶ $H_{TM} = \{\langle M, w \rangle : M \text{ is a TM and } M \text{ halts on input } w\}$

Theorem 6

H_{TM} is undecidable.

Proof: Assume, by way of contradiction, that \exists TM R that decides H_{TM} .

Algorithm 7 (S)

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

Reject, if input is not well formatted.

1. Emulate R on $\langle M, w \rangle$.
2. If R rejects, **reject**.
3. If R accepts, emulate M on w until it halts.
4. **Accept** if M accepted; otherwise **reject**.

TM S decides A_{TM} , a contradiction ♣

What we actually did is a **reduction from A_{TM} to H_{TM}** .

EMPTY_{TM} is undecidable

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

Theorem 8

EMPTY_{TM} is undecidable.

Proof's idea: By contradiction:

- ▶ Assume EMPTY_{TM} is decidable and let R be a TM that decides EMPTY_{TM}.
- ▶ Use R to construct S , a TM that decides A_{TM} .

Algorithm 9 (S – first attempt)

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

Emulate $R(\langle M \rangle)$ and **reject** if R **accepts**.

But what if R **rejects**?

Solution? Modify M .

EMPTY_{TM} is undecidable, the TM M_w

Definition 10 (M_w)

For TM M and input w , the TM M_w is defined as follows:

On input x , emulate M on w , and **accept** if M does.

$$L(M_w) = \begin{cases} \Sigma^* & w \in L(M) \\ \emptyset & \text{otherwise} \end{cases}$$

Question 11

Can a TM construct M_w from $\langle M, w \rangle$?

Answer: Easily because we need only hardwire w , and add a few extra states to replace x with w .

EMPTY_{TM} is undecidable, the reduction

- ▶ $\text{EMPTY}_{\text{TM}} = \{\langle M \rangle : M \text{ is a TM and } L(M) = \emptyset\}$
- ▶ Assume EMPTY_{TM} is decidable and let R be a TM that decides EMPTY_{TM} .

Algorithm 12 (S)

On input $\langle M, w \rangle$. **Reject**, if input is not well formatted.

1. Construct M_w .
2. Emulate R on input $\langle M_w \rangle$.
3. **Accept** if R **Rejects**; **reject** if R **accepts**.

Claim 13

S decides A_{TM} .

Proof: Recall that $L(M_w) = \emptyset$ iff M does not accept w . \square

A **contradiction**.



EMPTY_{TM} is undecidable, building M_w

Given $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ and $w = w_1, \dots, w_n$ (assuming $n \geq 1$),

build $M_w = (Q', \Sigma, \Gamma, \delta', \vec{q}_1, q_a, q_r)$, for

$Q' = Q \cup \{ \vec{q}_1, \dots, \vec{q}_{n+1}, \overleftarrow{q}, \overleftarrow{q}_1 \}$, and δ' defined by

$$\delta'(q, \sigma) := \begin{cases} (\vec{q}_2, \$, R) & q = \vec{q}_1 \\ (\vec{q}_{i+1}, w_i, R) & q = \vec{q}_i, 2 \leq i \leq n \\ (\vec{q}_{n+1}, \sqcup, R) & q = \vec{q}_{n+1}, \sigma \neq \sqcup \\ (\overleftarrow{q}, \sigma, L) & q = \vec{q}_{n+1}, \sigma = \sqcup \\ (\overleftarrow{q}, \sigma, L) & q = \overleftarrow{q}, \sigma \neq \$ \\ (\overleftarrow{q}_1, w_1, R) & q = \overleftarrow{q}, \sigma = \$ \\ (q_0, \sigma, L) & q = \overleftarrow{q}_1 \\ (q', \sigma', D), & \delta(q, \sigma) = (q', \sigma', D) \\ & \text{and none of the above conditions hold} \end{cases}$$

REG_{TM} is undecidable

$$\text{REG}_{\text{TM}} = \{ \langle M \rangle : M \text{ is a TM and } L(M) \text{ is regular} \}$$

Theorem 14

REG_{TM} is undecidable.

Proof's idea: By contradiction.

- ▶ Assume REG_{TM} is decidable and let R be a TM that decides REG_{TM}.
- ▶ Use R to construct S – a TM that decides A_{TM} .

Question 15

But how we construct S ?

Intuition: On input $\langle M, w \rangle$, build M_w that accepts regular language iff M accepts w .

REG_{TM} is undecidable, the TM M_w

The TM M_w will have the following property.

- ▶ if M does not accept w , then $L(M_w) = \{0^n 1^n : n \geq 0\}$
- ▶ if M does accept w , then $L(M_w) = \Sigma^*$

Algorithm 16 (M_w)

On input x ,

1. Accept, if x is of the form $0^n 1^n$.
2. (otherwise) Emulate M on input w and accept if M does.

Claim 17

1. If M does not accept w , then $L(M_w)$ is not regular.
2. If M does accept w , then $L(M_w)$ is regular.
3. The function $f(\langle M, w \rangle) = \langle M_w \rangle$ is computable.

REG_{TM} is undecidable, the reduction

Algorithm 18 (S)

On input $\langle M, w \rangle$. **Reject**, if input is not well formatted.

1. Construct M_w .
2. Emulate R on input $\langle M_w \rangle$ (recall that R is a TM that decides REG_{TM}).
3. **Accept**, if R **accepts** ; **Reject**, if R **rejects**.

Claim 19

S decides A_{TM} .

A contradiction



EQ_{TM} is undecidable

$$EQ_{TM} = \{ \langle M_1, M_2 \rangle : M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \}$$

Theorem 20

EQ_{TM} is undecidable.

We are getting tired of reducing A_{TM} to everything.
Let's try instead a reduction from $EMPTY_{TM}$ to EQ_{TM} .

Proof's idea:

- ▶ $EMPTY_{TM}$ is the problem of testing whether a TM language is empty.
- ▶ EQ_{TM} is the problem of testing whether two TM languages are the same.
- ▶ If one of these two TM languages happens to be empty, then we are back to $EMPTY_{TM}$.
- ▶ So $EMPTY_{TM}$ is a special case of EQ_{TM} .

The rest is easy.

EQ_{TM} is undecidable, 2

- ▶ $EQ_{TM} = \{ \langle M_1, M_2 \rangle : M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \}$
- ▶ Assume EQ_{TM} is decidable and let R be a TM deciding EQ_{TM} .

Proof:

Algorithm 21 (M_{NO})

On input x , **reject**

Algorithm 22 (S)

On input $\langle M \rangle$. **Reject**, if input is not well formatted.

Emulate R on input $\langle M, M_{NO} \rangle$.

If R accepts, **accept**; if R rejects, **reject**.

Claim 23

S decides $EMPTY_{TM}$.

A contradiction ♣

Bucket of undecidable problems

Similar techniques can be used to prove undecidability of

- ▶ Does a TM accept a **decidable** language?
- ▶ Does a TM accept a **context-free** language?
- ▶ Does a TM accept a **finite** language?
- ▶ Does a TM halt on **all inputs**?
- ▶ Is there an input string that causes a TM to **traverse all its states**?

Section 3

Mapping Reductions

Motivation

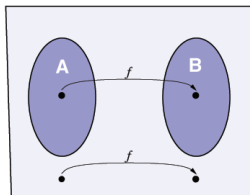
So far, we have seen many examples of **reductions** from one language to another, but the notion was neither defined nor treated formally.

Reductions play an important role in

- ▶ decidability theory (here and now)
- ▶ complexity theory (to come)

Time to **get formal**.

Mapping Reductions



Definition 24

A **total computable** function $f: \Sigma^* \mapsto \Sigma^*$ is a **mapping reduction** from language A to language B , if $x \in A \iff f(x) \in B$, for every $x \in \Sigma^*$.

If a mapping reduction from A to B exists, we say that A is **mapping reducible** to B , denoted by $A \leq_m B$.

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

Remark 25

Note that $A \leq_m B \iff \bar{A} \leq_m \bar{B}$

Applications

Theorem 26

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

Proof: Let M be the decider for B , and f a (mapping) reduction from A to B .

Algorithm 27 (N)

On input x

1. Compute $f(x)$
2. Emulate M on input $f(x)$. Accepts iff M does.



Corollary 28

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}

H_{TM} is undecidable — Revisited

- ▶ $A_{TM} = \{ \langle M, w \rangle : M \text{ is a TM that accepts } w \}$
- ▶ $H_{TM} = \{ \langle M, w \rangle : M \text{ is a TM and } M \text{ halts on input } w \}$

Claim 29

$$A_{TM} \leq_m H_{TM}$$

The following **computable** function f establishes
 $\langle M, w \rangle \in A_{TM} \iff f(\langle M, w \rangle) \in H_{TM}$.

Definition 30 (f)

On input $x = \langle M, w \rangle$. If x is not well formatted, return (some fixed) $x' \notin H_{TM}$.
Otherwise, return $\langle \tilde{M}, w \rangle$.

Definition 31 (TM \tilde{M})

On input y : emulate $M(y)$. **Accept**, if M accepts; **enter a loop**, if M rejects.

$H_{TM,\epsilon}$ is undecidable

Recall that

- ▶ $H_{TM} = \{\langle M, w \rangle : M \text{ is a TM that halts on input } w\}$
- ▶ $H_{TM,\epsilon} = \{\langle M \rangle : M \text{ is a TM that halts on input } \epsilon\}$

Claim 32

$$H_{TM} \leq_m H_{TM,\epsilon}$$

The following **computable** function f establishes

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

Definition 33 (f)

On input $x = \langle M, w \rangle$. If x is not well formatted, return $x' \notin H_{TM,\epsilon}$.
Otherwise, return $\langle M_w \rangle$.

(Recall M_w : on input x , emulates M on w).

Note that M_w halts on $\epsilon \iff M$ halts on w .

Therefore, $\langle M \rangle \in H_{TM,\epsilon} \iff \langle M, w \rangle \in H_{TM}$.

Mapping reducible relation is **not** symmetric

Claim 34

Let $\mathcal{A} \in \mathcal{R}$ and $\mathcal{B} \notin \mathcal{R}$, then $\mathcal{A} \leq_m \mathcal{B}$, but $\mathcal{B} \not\leq_m \mathcal{A}$.

Proof: It is clear that $\mathcal{B} \not\leq_m \mathcal{A}$ (?)

For proving $\mathcal{A} \leq_m \mathcal{B}$, define f as follows:

Definition 35 (f)

On input w , return $x_Y \in \mathcal{B}$ if $w \in \mathcal{A}$, and $x_N \notin \mathcal{B}$ otherwise.

- ▶ Do x_Y and x_N (always) exist?
- ▶ How can the TM implementing f find them?



Enumerability

Theorem 36

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

Proof is same as in the decidability case, using accepters instead of deciders.

Corollary 37

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

TM equality, revisited

$$EQ_{TM} = \{ \langle M_1, M_2 \rangle : M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \}$$

Theorem 38

Both EQ_{TM} and, its complement, $\overline{EQ_{TM}}$, are not enumerable.

Stated differently, EQ_{TM} is neither enumerable nor co-enumerable, or $EQ_{TM} \notin \mathcal{RE} \cup \text{co-}\mathcal{RE}$.

- ▶ We show (next slide) that $A_{TM} \leq_m EQ_{TM}$, and $A_{TM} \leq_m \overline{EQ_{TM}}$.
- ▶ Hence, $\overline{A_{TM}} \leq_m \overline{EQ_{TM}}$ and $\overline{A_{TM}} \leq_m EQ_{TM}$ (?)
- ▶ Hence, neither EQ_{TM} nor $\overline{EQ_{TM}}$ are enumerable.

$$A_{\text{TM}} \leq_m \text{EQ}_{\text{TM}}$$

Claim 39

$$A_{\text{TM}} \leq_m \text{EQ}_{\text{TM}}.$$

Definition 40 (f)

On input $x = \langle M, w \rangle$. If x is not well formatted, return $x' \notin \text{EQ}_{\text{TM}}$.

1. Construct the TM M_w (recall that M_w emulates M on w [while ignoring its input]), and the TM M_{all} that **accepts** Σ^* .
2. Return $\langle M_w, M_{\text{all}} \rangle$.

- ▶ If M accepts w , then M_w accepts **everything**. Otherwise, M_w accepts **nothing**.
- ▶ Hence, $\langle M, w \rangle \in A_{\text{TM}} \iff \langle M_w, M_{\text{all}} \rangle \in \text{EQ}_{\text{TM}}$.



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

Claim 41

$$A_{\text{TM}} \leq_m \overline{\text{EQ}_{\text{TM}}}.$$

Definition 42 (f)

On input $x = \langle M, w \rangle$. If x is not well formatted, return $x' \notin \overline{\text{EQ}_{\text{TM}}}$.

1. Construct the TM M_w , and the TM M_{NO} that **accepts** \emptyset .
2. Return $\langle M_w, M_{\text{NO}} \rangle$.

- ▶ If M accepts w , then M_w accepts **everything**. Otherwise, M_w accepts **nothing**.
- ▶ Hence, $\langle M, w \rangle \in A_{\text{TM}} \iff \langle M_w, M_{\text{NO}} \rangle \in \overline{\text{EQ}_{\text{TM}}}$



Section 4

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 43

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

Rice's Theorem

Theorem 44

For non-empty $\mathcal{C} \subsetneq \mathcal{RE}$, it holds that

$$L_{\mathcal{C}} = \{\langle M \rangle : M \text{ is a TM and } L(M) \in \mathcal{C}\} \notin \mathcal{R}.$$

Note that both L_{\emptyset} and $L_{\mathcal{RE}}$ are in \mathcal{R} .

Proof's idea: Mapping reduction from H_{TM} .

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

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

Proving Rice's Theorem

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

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

Algorithm 45 (M_w^C)

On input y :

1. Emulate $M(w)$.
2. Emulate $M_A(y)$:
 Accept, if M_A accepts; reject, if M_A rejects.

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

Claim 46

f is a mapping reduction from H_{TM} to L_C

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

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

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

Proof:

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



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

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

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

$\implies \bar{L}_{\mathcal{C}} \notin \mathcal{R}$ (?)

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

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

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

Theorem 47

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

Proof: (?)

Theorem 48

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

Proof: L_{Primes} is a non trivial subset of \mathcal{RE} .

Reflections on Rice theorem

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

Section 5

Proving Non-Enumerability

Proving non-enumerability

Theorem 49

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

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

Proof: We show that $\overline{A_{\text{TM}}} \leq_m L_{\mathcal{C}}$.

Let 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 D o/w.

Definition 50 ($B_{M,w}$)

On input x :

- ▶ Emulate $D(x)$ and **accept** if D does.
- ▶ Emulate $M(w)$ and **accept** if M does.

$$\text{▶ } \langle M, w \rangle \in \overline{A_{\text{TM}}} \implies L(B_{M,w}) = \mathcal{A} \in \mathcal{C}$$

$$\text{▶ } \langle M, w \rangle \notin \overline{A_{\text{TM}}} \implies L(B_{M,w}) = \Sigma^* \notin \mathcal{C}.$$

□

Rice theorem vs. Thm 49

Rice speaks about non-membership in \mathcal{R} where Thm 49 about non-membership in \mathcal{RE} .

But is it always the case that when Rice is applicable then so is Thm 49 ?

- ▶ Let $\mathcal{C}_a = \{\mathcal{L} \in \mathcal{RE} : a \in \mathcal{L}\}$.
- ▶ By definition, $\mathcal{L}_{\mathcal{C}_a} = \{\langle M \rangle : M \text{ is a TM and } L(M) \in \mathcal{C}_a\}$
- ▶ Note that $\mathcal{L}_{\mathcal{C}_a} = \{\langle M \rangle : a \in \mathcal{L}(M)\}$
- ▶ By Rice, $\mathcal{L}_{\mathcal{C}_a} \notin \mathcal{R}$
- ▶ But clearly, $\mathcal{L}_{\mathcal{C}_a} \in \mathcal{RE}$
- ▶ Thm 49 is not applicable here since $\mathcal{C}_a \notin \mathcal{R}$.

Section 6

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 51 (S_n and $BB(n)$)

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

- ▶ $S_n = \{ \text{all } n\text{-state TM's that halt on } \varepsilon \}$.
- ▶ $BB(n) = \max_{M \in S_n} \{ \# \text{ of steps taken by } M \text{ on input } \varepsilon \}$.
- ▶ The set S_n is finite (under standard encoding)
- ▶ Every $M \in S_n$ runs for **finitely** many steps on ε .
- ▶ $BB(n)$ is a total function from \mathbb{N} to \mathbb{N} (in particular, $BB(n) \in \mathbb{N}$ 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 52

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 53 (S)

On input $\langle M \rangle$

1. Let m be # of states in M , and let $n = R(m)$.
2. Emulate M on ε 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 54

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



The bounded *Busy Beaver* function is computable

Definition 55

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

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

Theorem 56

The function BB_d is computable for every $d \in \mathbb{N}$.

Proof's idea: "Hardwire" the values $BB(1) \dots, BB(d)$ into a TM to compute BB_d .