

Random-Access Machine,

(9)

- formal model, but much closer to real hardware than the TM
- in fact, it's a family of related models, we will show the simplest of them
- RAM works with numbers (our version: the whole of $\mathbb{Z}$)
- memory: seq. of numbers, indexed by numbers (negative indices allowed)
- addressing of operands:
 - literal constant (embedded in an instruction)
 - $[n]$ - directly addressed memory cell
 - $[[n]]$ - indirectly — " — (read $[n]$ to obtain another cell address)
- instructions:
 - ① movement of data $X \leftarrow Y$
 $Y = \text{any}, X = \text{any except literal}$
 - ② arithmetics $X \leftarrow Y \oplus Z$
 $\oplus = +, -, *, \%$
bitwise &, or, xor
bitwise shift
 - ③ control
 - halt
 - jump PLACE
 - if $X < Y$ jump PLACE
 $\nwarrow <, >, =, \neq, \leq, \geq$
- instructions executed sequentially + jumps
- input is stored at agreed-upon locations in memory when the program starts
- output is found ———— " ———— when the program stops

Example: sum of N numbers

In: $[0] = N, [1] = x_1, \dots, [N] = x_N$

Out: $[0] = \text{sum}$

Temporary: $[-1] = \text{copy of } N, [-2] = \text{current index}$

Program:
 $[-1] \leftarrow [0]$ copy N
 $[0] \leftarrow 0$ initialize sum
 $[-2] \leftarrow 1$ start with x_1

Loop: if $[-2] > [-1]$ jump END
 $[0] \leftarrow [0] + [[-2]]$
 $[-2] \leftarrow [-2] + 1$
jump LOOP

END: halt

Complexity: time = # executed instructions

space = max (cell address used) - min (---)

← this varies between RAM versions,
e.g. we could define cost of an instruction as

$\max \log(1 + |x|)$

$x \in \{\text{operands, addresses, result}\}$

↓

or keep cost constant,
but restrict size of cells

Somehow...

• "TM is equivalent to RAM"

- what can this mean?

... they can simulate each other: for each
RAM program there is an equivalent TM
& vice versa

... but RAM crunches numbers,
while TM crunches strings

We will assume that the RAM gets a string $\in \Sigma^*$ as input:

$[0] = \text{length}$, $[1], [2], \dots = \text{symbols of the string (encoded as integers)}$

(this is UERG since both TM and RAM can convert between all reasonable input formats)

TM to RAM - wlog 1-tape TM with 1-way-infinite tape

- store the contents of the written-to part of the tape in $[1], [2], \dots$
- $[0]$ will specify how far the $\text{---} \text{ " " } \text{---}$ stretches.
- $[-1]$ = current position of head
- position in program represents machine state
- can simulate 1 step of the TM in constant time.

4. Using some numbering of the work alphabet

RAM to TM

- representation of numbers: binary + sign symbol
- TM subroutines for arithmetics (inputs/output on special tapes)
- tape M: memory of the RAM # cell -1 # cell 0 # cell 1 # ...
- tape A: address of memory cell in which the head on tape M is
 - ... can move 1 cell left/right, possibly extending M by empty cells at both ends
- memory read: given address on tape R, copy number read to tape D (data)
 - ... compare R with A, move across cells until ~~add~~ $R=A$, copy data from M to D
- memory write: similar, but need to expand cells if they are too small for new data
- every instruction can be composed of read/write/arithmetics
- keep position in RAM program inside state of the TM
- simulation works, but with significant slowdown (inevitable?)

Computability

We will study it only for languages (decision problems),
generalization to functions is straight-forward.

Df: • Turing machine M accepts word $\alpha \in \Sigma^*$ $\equiv$ computation on α ends in state q^+
• M rejects $\alpha \Leftrightarrow$ stops in q^- or runs forever (diverges)

- Language $L(M)$ accepted by $M \equiv \{\alpha \in \Sigma^* \mid M \text{ accepts } \alpha\}$
- Language L is decided by $M \equiv M$ always stops & $L = L(M)$.

Df: • Language L is computable (a.k.a. decidable/recursive) $\equiv$
 $\exists$ TM M : L is decided by M . ↑ refers to Church's formalism of recursive functions (equivalent to TM)

• Language L is partially computable (a.k.a. partially decidable/recursively enumerable)
 $\equiv \exists$ TM M : L is accepted by M (i.e., $L(M) = L$).

Df: $R := \{L \mid L \text{ is computable}\}$

$RE := \{L \mid L \text{ is partially computable}\}$

since elements of Σ can be arbitrary, these are proper classes.

WLOG we can fix $\Sigma = \{0, 1\}$

$R \subseteq RE \subseteq 2^{\{0,1\}^*}$ ← all languages over $\{0,1\}$ to make R and RE sets.
↑ are these strict? Watch out...

Enumeration (or: why "recursively enumerable"?)

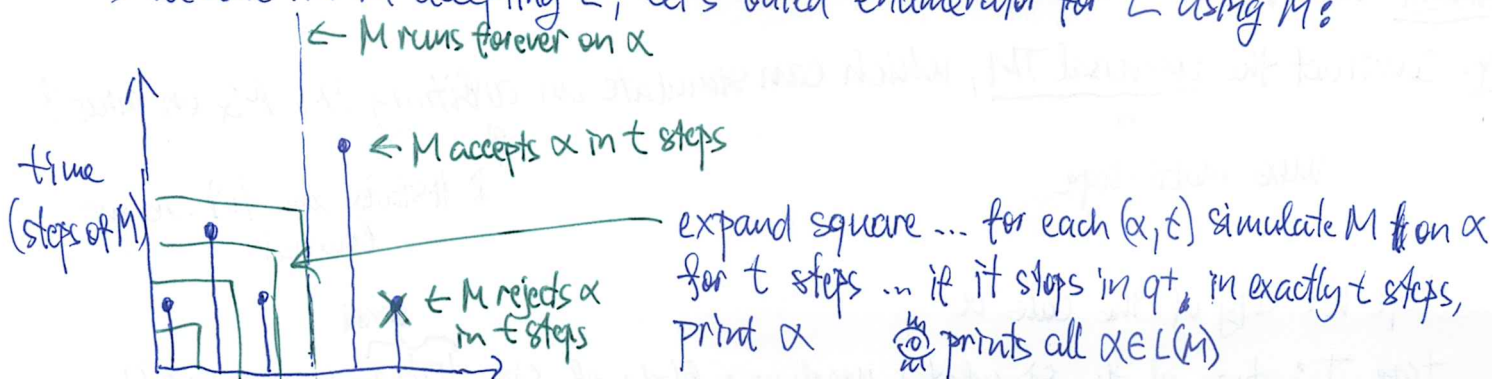
Df: Enumerator $\equiv$ TM with no input, potentially running forever, printing strings (formally: printer is an oracle) } → language enumerated by M

L is enumerable $\equiv \exists$ enumerator which prints exactly the words of L

Thm: $L \in RE \Leftrightarrow L$ is enumerable

Pf: $\Leftarrow$ we want to accept $\alpha \in L$... run enumerator, compare printed strings with α
YES $\Rightarrow$ stop in q^+ , NO $\Rightarrow$ continue
enumerator stops $\Rightarrow$ stop in q^-

$\Rightarrow$ we have TM M accepting L , let's build enumerator for L using M :
← M runs forever on α



Strings in length-lexicographic order ($\alpha \leq_L \beta \equiv |\alpha| < |\beta| \vee |\alpha| = |\beta| \wedge \alpha \leq_{\text{lex}} \beta$)

Homework: $L \in RE \Leftrightarrow L$ is enumerable in $\leq_{\text{lex}}$ order. [binary numbers with leading 1 removed.]

Universal TM (why we don't need TM program in modifiable memory)

(12)

Df: Encoding of TMs (a.k.a. Gödel numbering) ← but in our case, the codes are actually strings, not numbers
we define it for 1-tape machines with $\Sigma = \{0, 1\}$

alphabet: $\Gamma = \{x_0, x_1, x_2, \dots, x_m\}$
↑ ↑ ↑
0 1 2 other symbols in arbitrary order

directions: $\{d_0, d_1, d_2\}$
← → •

states: $Q = \{q_0, q_1, q_2, \dots, q_n\}$
↑ ↑ ↑
initial q^+ q^- other states

start code with $1^m 0 1^n 0$ to preserve $|\Gamma|$ and $|Q|$ even if symbols/states unused

transitions: $\delta(q_i, x_j) = (q_k, x_e, d_e) \rightarrow$ encode as $1^{i^m} 0 1^{j^n} 0 1^{k^m} 0 1^{e^n} 0 1^{d_e^m} 0$

↳ concatenate codes of all transitions $\rightarrow$ code of machine $\langle M \rangle$

Df: M_α := machine with code α (if α not a valid code $\Rightarrow$ machine which immediately halts in q^-)

⊗ $\forall TM M \exists \alpha : M \cong M_\alpha$

↑ isomorphism of TMs (defined in the obvious way)
↳ in fact, there are multiple such codes (we numbered Q, Γ arbitrarily etc.)

⊗ $L_\alpha := L(M_\alpha) \dots \forall \alpha L_\alpha \in RE$

⊗ $\forall L \in RE \exists \alpha : L = L_\alpha \dots$ infinitely many choices of α (we can add arbitrarily many unreachable states)

codes is countable $\Rightarrow RE$ is countable $\dots$ but $2^{\{0,1\}^*}$ uncountable

$\Rightarrow \exists L \notin RE$ (non-constructively)

Tool: Encoding of pairs $\langle \alpha, \beta \rangle$: $\langle x_1 \dots x_n, y_1 \dots y_m \rangle = x_1 0 x_2 0 \dots x_n 0 1 y_1 0 \dots y_m 0$

⊗ encoding & decoding is computable (& well-defined)

Df: Universal language $L_u := \{ \langle \alpha, \beta \rangle \mid \alpha, \beta \in \{0,1\}^* \text{ \& } \beta \in L_\alpha \}$
"contains all partially computable languages" (in a sense)

Lemma: $L_u \in RE$

Pf: Construct the universal TM, which can simulate an arbitrary TM M_α on input β
↑
w/o multi-tape
↑
#states and $|\Gamma|$ are not bounded

tape K: copy of the code α

tape T: tape of the simulated machine: blocks of size $|\Gamma|+1$, symbol $x_i \in \Gamma$
↑
stored as $1^i 0^{m-i}$

tape M: 1^m
↑
head on T encodes position of M_α 's head

tape S: current state of M_x stored as $1^i 0^{19-i}$

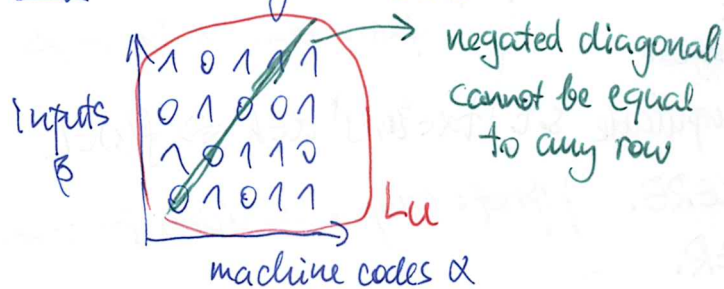
(13)

Init: Split $\langle \alpha, \beta \rangle$, copy α to tape K, encode β on tape T, initialize tape S
↳ & set tape M

Step: Read current symbol x from T, find entry for state s and symbol x on K, write new symbol & state, move head on T.

Lemma: $\overline{L_u} \notin RE$ ← We define $\overline{L} := \{0,1\}^* \setminus L$ for $L \subseteq \{0,1\}^*$

Proof: Use diagonalization



→ diagonal language

$L_d := \{ \alpha \in \{0,1\}^* \mid \alpha \notin L_\alpha \}$

$L_d \notin RE$... assume $L_d \in RE$

Then $\exists \alpha: L_d = L_\alpha$

but: $\alpha \in L_d \Leftrightarrow \alpha \notin L_\alpha \Leftrightarrow \alpha \notin L_d$ ↯

If $\overline{L_u}$ were partially decidable, we could modify the machine accepting $\overline{L_u}$ to a machine accepting L_d ↯

Corollaries:

- $L_u \notin R$ (R is closed under complement, so $L_u \in R$ would imply $\overline{L_u} \in R \subseteq RE$)

- $R \subsetneq RE \subsetneq 2^{\{0,1\}^*}$
↑ ↑
witnessed by L_u witnessed by $\overline{L_u}$

Exercise: Are R and RE closed under $\cap$ or $\cup$?

- RE is not closed under complement

Thm (Post's): $L \in R \Leftrightarrow L \in RE \ \& \ \overline{L} \in RE$. ← equivalently: $R = RE \cap \overline{co-RE}$

Pf: $\Rightarrow$ trivial, because $R \subseteq RE$ & R closed under complement. $\{ \overline{L} \mid L \in RE \}$

$\Leftarrow$ "run machines accepting L and $\overline{L}$ in parallel" (one step of each at a time)
One of them certainly stops.

Operations on machines codes

- swap q^+ with q^- : given M_x ^{deciding} ~~accepting~~ L , find M_y deciding $\overline{L}$
- compose two machines: find M_y , which runs first M_x and then M_y on its output
- substitute M_x for an oracle in M_y

} all these are computable functions

More decision problems regarding machine codes

(14)

$$L_{halt} := \{ \langle \alpha, \beta \rangle \mid M_\alpha \text{ halts on input } \beta \}$$

$$L_{empty} := \{ \alpha \mid L_\alpha = \emptyset \} \quad L_{total} := \{ \alpha \mid L_\alpha = \{0,1\}^* \}$$

$$L_{eq} := \{ \langle \alpha, \beta \rangle \mid L_\alpha = L_\beta \}$$

Exercise: which of these (& their complements) are in R and/or RE?

- Return to proof of $L_{eq} \notin RE$ via $L_d \notin RE$: "if we find a machine accepting L_{eq} , we can use it to accept L_d "

Let's generalize this.

Def: Many-to-one reduction between languages:

$$K \leq_m L \equiv \exists f: \{0,1\}^* \rightarrow \{0,1\}^* \text{ computable s.t. } \forall x \in \{0,1\}^* x \in K \Leftrightarrow f(x) \in L$$

$\leq_m$ is a partial quasi-order on languages

Lemma: If $K \leq_m L$ and $L \in RE$, then $K \in RE$.

If $K \leq_m L$ and $L \in R$, then $K \in R$.

proof: compose machines for f and L

Corollary: If $K \leq_m L$ and $K \notin RE$, then $L \notin RE$. (Similarly $K \notin R \Rightarrow L \notin R$.)

- Our original proof used $L_d \leq_m L_{eq}$ & $L_d \notin RE$ to show $L_{eq} \notin RE$.

Exercise: Find reductions between L_{eq} , L_{halt} , L_{empty} , L_{eq} & their complements.

Example: ③ $L_{halt} \xrightarrow{\leq_m} L_{empty}$: given $\langle \alpha, \beta \rangle$, construct TM M_x which ignores its input & runs M_α on input β

$\hookrightarrow L_x = \emptyset$ if $M_\alpha(\beta)$ diverges

$L_x = \{0,1\}^*$ otherwise

$$\Rightarrow (x \in L_{empty} \Leftrightarrow \langle \alpha, \beta \rangle \in L_{halt})$$

④ $L_{empty} \xrightarrow{\leq_m} L_{halt}$: for given α , construct M_x which ignores its input, simulates M_α on all inputs in parallel & stops if $M_\alpha(\beta)$ stops on some β ...

$$\textcircled{1} L \xrightarrow{\leq_m} M \Leftrightarrow \overline{L} \xrightarrow{\leq_m} \overline{M}$$

② Also, $L_{halt} \leq_m L_{eq} \leq_m L_{halt}$

$$\begin{aligned} \Leftrightarrow \langle \beta, \epsilon \rangle \in L_{halt} \\ \Leftrightarrow L_\beta = \emptyset \\ \Leftrightarrow \alpha \in L_{empty} \end{aligned}$$

Semantic properties of machines

Def: Property of languages: $P \subseteq RE$... P is non-trivial $\equiv P \neq \emptyset$ & $P \neq RE$.
(semantic)

$$L_P := \{ \alpha \in \{0,1\}^* \mid L_\alpha \in P \} \quad \dots \text{all machines whose languages have the property } P$$

Thm (Rice's): For every non-trivial property P , the language L_P is undecidable.

Proof idea: Show that $L_{halt} \rightarrow L_P$ for every non-trivial P .

Proof: Assume that $L_P \in R$ for some P .

WLOG $\emptyset \notin P$... otherwise use $\bar{P}$... $L_{\bar{P}} = \overline{L_P}$, so it's also in R .

Find $L_W \in P$... exists as P is non-trivial

Reduction: if we want to answer $\langle \alpha, \beta \rangle \in L_{\text{halt}}$, i.e. if $M_\alpha(\beta)$ halts

construct M_β which does on input δ :

this is
computable

- run M_α on β (1)
- run M_W on δ (2)

if $\langle \alpha, \beta \rangle \in L_{\text{halt}}$: (1) halts, (2) halts if $\delta \in M_W \Rightarrow L_\beta = L_W \in P$
 $\notin L_{\text{halt}}$: (1) diverges, (2) doesn't run $\Rightarrow L_\beta = \emptyset \notin P$

So this shows $L_{\text{halt}} \leq_m L_P$... but $L_{\text{halt}} \notin R$, so $L_P \notin R$.

What is the "hardest" language in a class?

- Let C be a set of languages.
- L is C -hard $\equiv \forall K \in C: K \leq_m L$
- L is C -complete $\equiv L$ is C -hard & $L \in C$

more precisely, it's
 C -m-complete
(complete wrt. $\leq_m$)

Thm: L_u is RE-complete.

Pf: ① $L_u \in RE$

② for $K \in RE$, we find $\alpha: L_\alpha = K$

Then β reducing K to L_u is $\beta \mapsto \langle \alpha, \beta \rangle$

Also: If K is C -complete
and $K \leq_m L$ for $L \in C$,
then L is C -complete, too.
Hence L_{halt} is also RE-complete.

"Natural" undecidable problems (not directly involving machines)

- given a set of axioms and a formula φ , is φ provable?
- given a system of multi-variate polynomial equations over $\mathbb{Z}$, does it have a solution in $\mathbb{R}$? $\rightarrow$ Matijasević theorem

both in
 $RE \setminus R$
(in suitable
encoding)

& many more (e.g., plane tiling)

Relative computability

- given any language $A \subseteq \{0,1\}^*$, we can define ~~oracle~~ TM with an oracle giving access to A (see section on TM extensions)
- we can define relative language classes $R[A]$ and $RE[A]$
- we also have $M_\alpha[A]$, $L_\alpha[A]$, $L_u[A]$

if $A \in R$, then $R[A] = R$ and $RE[A] = RE$ (in particular for $A = \emptyset$)

Previous arguments about plain TM can be trivially relativized,
so in particular: $R[A] \neq RE[A] \neq \Sigma_1^0$

$R[A] = RE[A] \cap \text{co-}RE[A] \leftarrow \text{co-}T = \{L \mid \bar{L} \in T\}$

And also
 $L_u[A]$ is $RE[A]$ -complete

Def: Aritmetical hierarchy: classes $\Sigma_n, \Pi_n, \Delta_n$ for $n \in \mathbb{N}$

(16)

- $\Sigma_0 = \Pi_0 = \Delta_0 = R$

- $\Sigma_{n+1} = RE[\Sigma_n]$

- $\Pi_{n+1} = co-RE[\Pi_n]$

- $\Delta_{n+1} = R[\Sigma_n]$

we have:

$$\Sigma_1 = RE$$

$$\Pi_1 = co-RE$$

$$\Delta_1 = R[R] = R$$

$$\Sigma_n \subseteq \Sigma_{n+1}$$

oracles from R
do not add
power to TM

this means:

$$RE[C] = \bigcup_{L \in C} RE[L]$$

this inclusion is strict

~~as $RE[\Sigma_n] \neq RE$~~

$$RE[L_n^u] \neq \Sigma_n$$

otherwise we would have

$$\Sigma_n \subseteq R[\Sigma_n] = R[L_n^u]$$

$$\neq RE[L_n^u] = \Sigma_{n+1}$$

- $L_u^1 = L_u$

$$L_u^{n+1} = L_u[L_u^n]$$

- $\Pi_{n+1} = co-RE[\Sigma_n]$

- $\Pi_{n+1} = co-\Sigma_{n+1}$

$\rightarrow L_u^n$ is Σ_n -complete

(by induction: $RE[\Sigma_n] = RE[L_u^n]$,

so $L_u[L_u^n]$ is Σ_{n+1} -complete)

Also: $\Sigma_n \subsetneq \Delta_{n+1} \dots$ and this is strict as Σ_n is not closed under complement, while Δ_{n+1} is
 $\Pi_n \subseteq \Delta_{n+1} \dots$ we can negate oracle's answer

$$\Delta_{n+1} = \Sigma_{n+1} \cap \Pi_{n+1} \dots \text{relative Post's thm.}$$

$$\Delta_{n+1} \subseteq \Sigma_{n+1} \dots \text{this is } R[L_u^n] \neq RE[L_u^n]$$

$$\Delta_{n+1} \subseteq \Pi_{n+1} \dots \text{analogous for co-RE}$$

Quantified formulas

- every language in R can be interpreted as a predicate $\varphi(x)$ with string parameter φ - decidable predicates

- $\varphi(\beta) \equiv \exists x \varphi(x, \beta)$ lies in RE

... and every $L \in RE$ can be written in this way
[$\alpha = \#$ steps after which a machine stops]

- $\varphi(\beta) \equiv \forall x \varphi(x, \beta) \dots$ this is $co-RE$ ($\neg \forall x \varphi(x, \beta) \Leftrightarrow \exists x \neg \varphi(x, \beta)$)

- ... $\exists x_1 \exists x_2 \varphi(x_1, x_2, \beta)$ is again RE ... we can say $\exists \alpha$ s.t. $\alpha = \langle x_1, x_2 \rangle$ & decode α inside φ

- $\exists x_1 \forall x_2 \varphi(x_1, x_2, \beta)$ is Σ_2

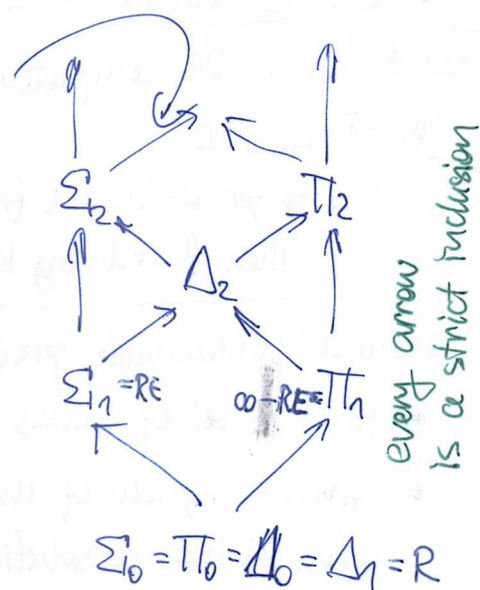
- in general: $\exists x_1 \forall x_2 \exists x_3 \dots Q_n x_n \varphi(x_1 \dots x_n, \beta)$ is Σ_n
 $\forall x_1 \exists x_2 \forall x_3 \dots Q_n x_n \varphi(x_1 \dots x_n, \beta)$ is Π_n

$$\subseteq \Sigma_2: \exists x_1 \forall x_2 \varphi(\dots) \Leftrightarrow \exists x_1 \neg (\exists x_2 \neg \varphi(\dots)) \leftarrow \text{so this is in } RE[\Sigma_1] = \Sigma_2$$

$\supseteq \Sigma_2$ consider $L \in \Sigma_2 = RE[\Sigma_1]$:

$\exists y$ computation of TM M_1 [L] $\exists \delta$ queries for L_u (δ consistent, & answers for δ true)

see next page



(continued) We want to show that $\forall L \in \Sigma_2$ there is an equivalent formula $\exists \exists \dots$ (17)

• let M be a $TM[\Sigma_1]$... that is $TM[L_u]$ accepting L

• formula: $\exists \delta$ (check that δ is consistent with input α , with oracle L_u , and with itself)

$\uparrow$ computation of M including all oracle queries & answers
 $\uparrow$ decidable
 $\uparrow$ decidable

for positive answers: $\exists \delta_1 - \delta_k$
 $\psi(\delta_1, \text{question 1})$
 $\vdots$
 $\psi(\delta_k, \text{question k})$
 Σ_1 -formula for L

for negative answers:
 $\forall \epsilon_1 - \epsilon_k \neg \psi(\epsilon_1, \dots)$
 $\& \dots$

together:
 $\exists \delta (\exists \gamma \forall \epsilon \varphi(\alpha, \delta, \gamma, \epsilon))$
 $\uparrow$ code of $\delta_1 - \delta_k$ $\uparrow$ code of $\epsilon_1 - \epsilon_k$

... but this is $\exists \langle \delta, \gamma \rangle \forall \epsilon \varphi(\dots)$ as we need.

COMPLEXITY

• For a multi-tape machine M , define run time $t_M(x)$ for input α as #steps before computation $M(x)$ halts.

- $t_M: \{0,1\}^* \rightarrow \mathbb{N}^*$
 generally Σ_1^*

sorry for confusing notation: this * is not iteration, but:
 $\mathbb{N} \cup \{\infty\}$ for divergent computations

• Time complexity of machine M : $T_M: \mathbb{N} \rightarrow \mathbb{N}^*$ s.t. $T_M(n) = \max_{\alpha \in \Sigma_1^n} t_M(\alpha)$.

👁 M always halts $\Leftrightarrow T_M(n)$ finite for all n .

Let's use $\alpha \in \Sigma_1^*$, $|\alpha| \leq n$ instead:
 this is slightly non-standard, but much more convenient. In particular, $T_M(n)$ will be non-decreasing.

Def: Asymptotic notation: for functions $f, g: \mathbb{N} \rightarrow \mathbb{R}$ define:

① $f \in O(g) \equiv \exists c \forall^* n \ f(n) \leq c \cdot g(n)$

$\uparrow$ for all but finitely many exceptions

← asymptotic upper bound

② $f \in \Omega(g) \equiv \exists c \forall^* n \ f(n) \geq c \cdot g(n)$

← asymp. lower bound

③ $f \in \Theta(g) \equiv f \in O(g) \ \& \ f \in \Omega(g)$

← both at once

... that is, $\Theta(g) = O(g) \cap \Omega(g)$

④ $f \in o(g) \equiv \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

← strict upper bound

⑤ $f \in \omega(g) \equiv g \in o(f)$

← strict lower bound

Examples: $f: n \mapsto 5n^3 - 7n^2 + 18$

$\in O(n^3), O(n^4), O(2^n) \in o(n^4)$

$\in \Omega(n^3), \Omega(n^2), \Omega(1) \in \omega(n^2)$

$O(1)$ = "bounded by constant"

$\in \Theta(n^3)$ "drop lower-order terms & multiplicative constant"