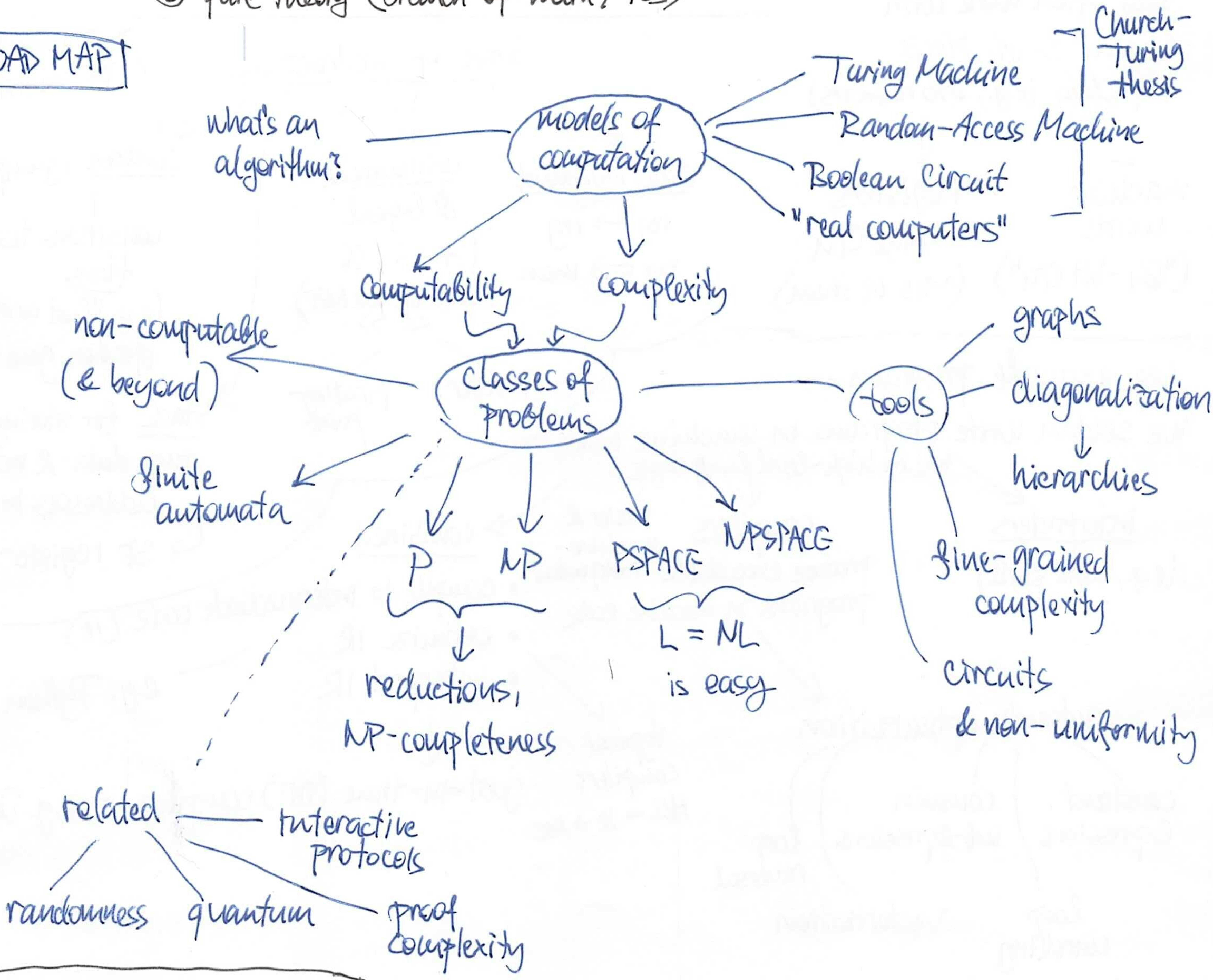
Automata & Complexity Theory

winter 2022

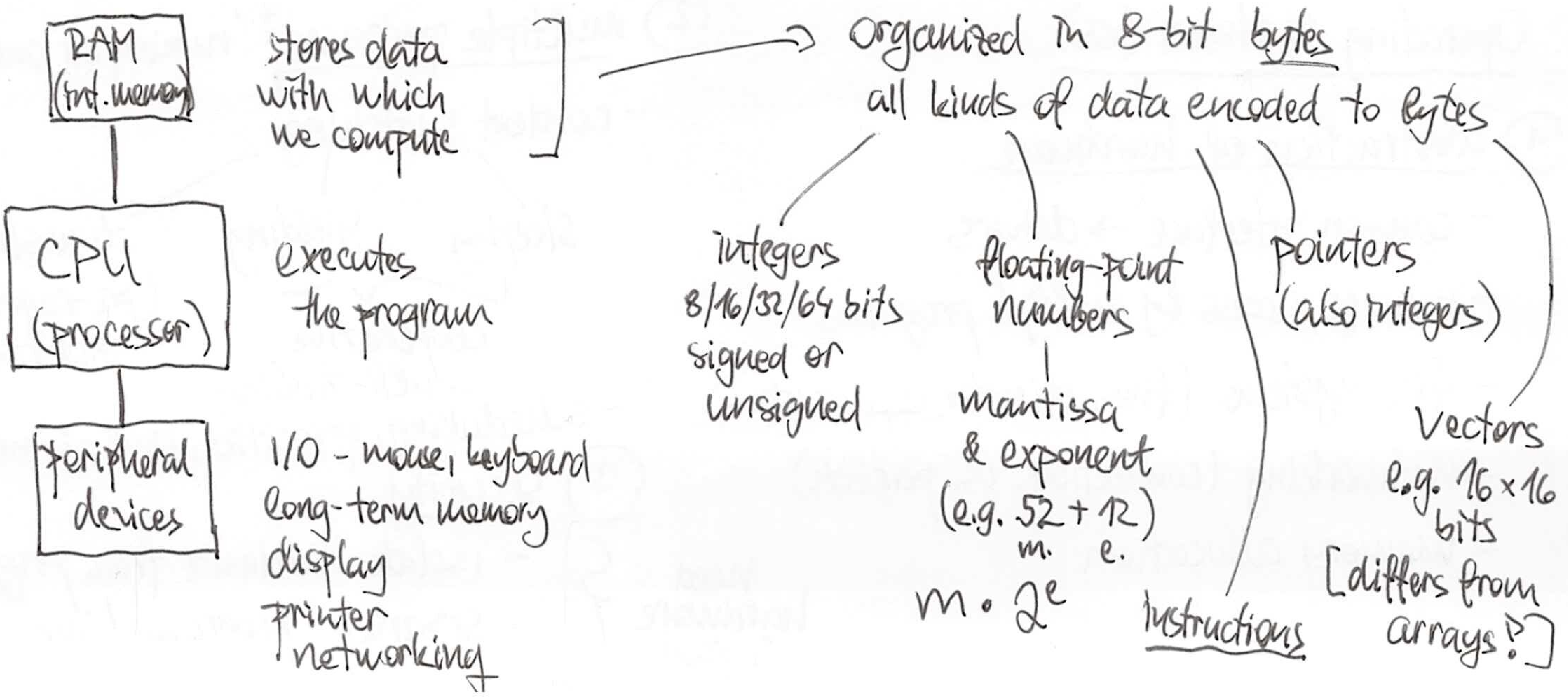
goal: build theory of computation & hardness of problems

two views: ① an applied theory to help us use machines more efficiently
② pure theory (branch of math / TCS)

ROAD MAP



PHYSICAL COMPUTERS & their architecture (typical case in 2022, just sketching)



machine instructions (program)
- stored in the same memory as data (Von Neumann architecture)
- can modify itself
- just another interpretation of bytes

↓

they often work with several simple pieces of data (e.g., 64b numbers)
- machine words ("64-bit CPU")
- registers inside CPU (~10s of them)

---

types of instructions
- **data movement**
  - reg → reg
  - reg ↔ mem
- **arithmetical & logical** (+ - * / % AND OR XOR NOT << >>)
  - integer
  - floating-point
- **control** (jumps) → program flow, PC register
  - conditions test flags (e.g. "last arithmetica operation gave zero")
- **stack** for storing tmp. data & return addresses in RAM
  - ↳ SP register

---

see example program...

We seldom write programs in machine insn's but in high-level languages
- **interpreters** (e.g., UNIX shell)
- **compilers** produce executable programs in machine code
  - easier & machine-independent
  - **automatic optimization**
    - constant expressions
    - common sub-expressions
    - loop reversal
    - loop unrolling
    - vectorization
    - & many others
- **combined**
  - • compile to intermediate code (IR)
  - • optimize IR
  - • interpret IR — e.g. Python
  - typical compiler: HLL → IR → MC
  - ↓ just-in-time (JIT) compilers — e.g. Java

Operating systems (OS)

① **abstraction of hardware**
- common interface → drivers
- manage access by multiple programs
- file systems (files, directories, mounts...)
- networking (connections vs. packets)
- memory allocation

② **multiple processes** "running at once"
- context switching
  - sleeping, yielding → cooperative multi-tasking
  - timeslices (pre-emptive multitasking)
- scheduling, priorities, real-time

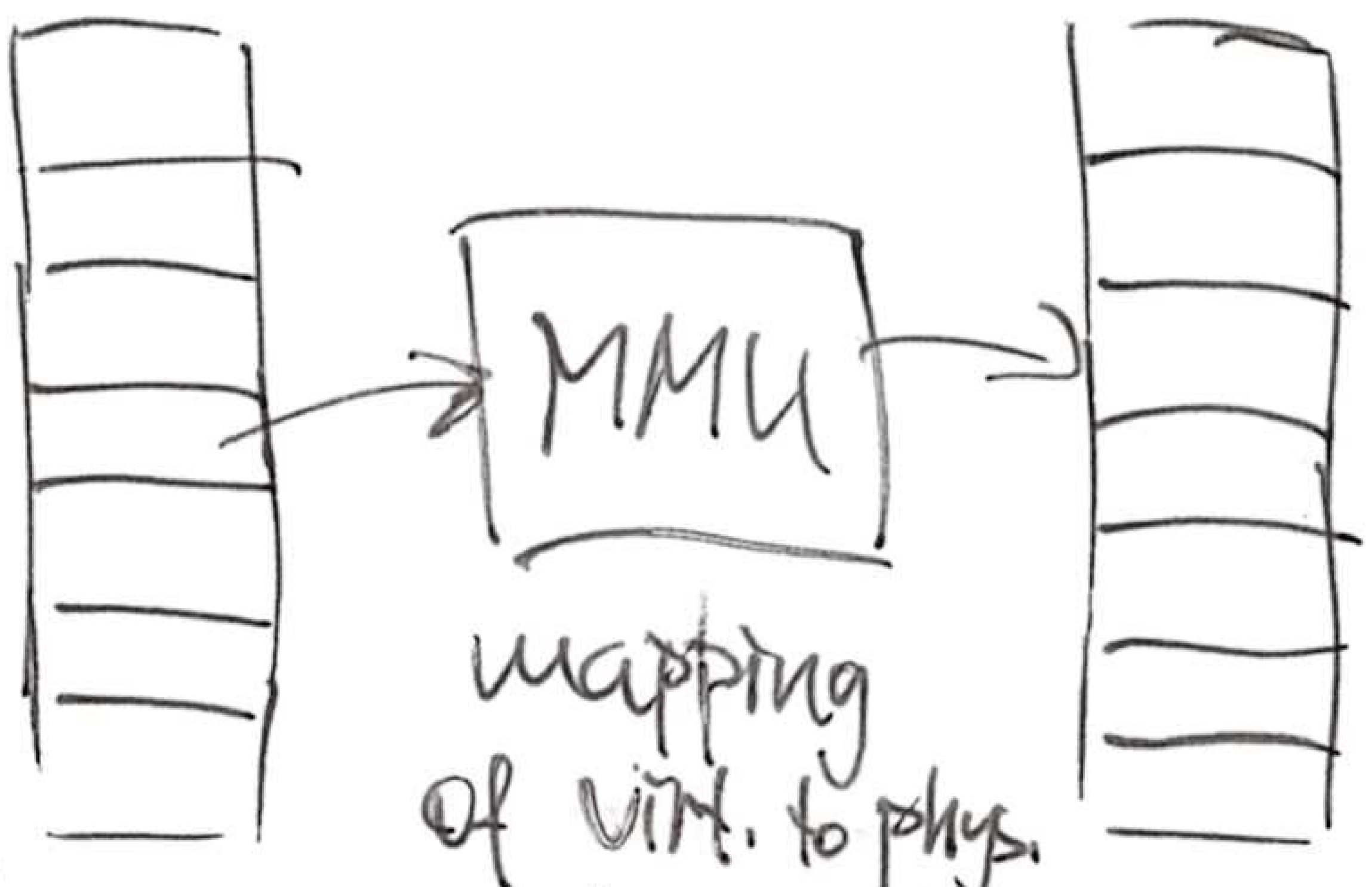
③ **security**

need hardware support {
- isolate hardware from programs
- separate programs/users

- HW features for security – privilege levels (supervisor/user mode) → system call instructions

↓

Virtual memory management



Virtual memory (4kB pages)

mapping of virt. to phys. page numbers — includes access rights
- read R
- write W
- execute code X
- supervisor only (or switching tables on mode changes...)

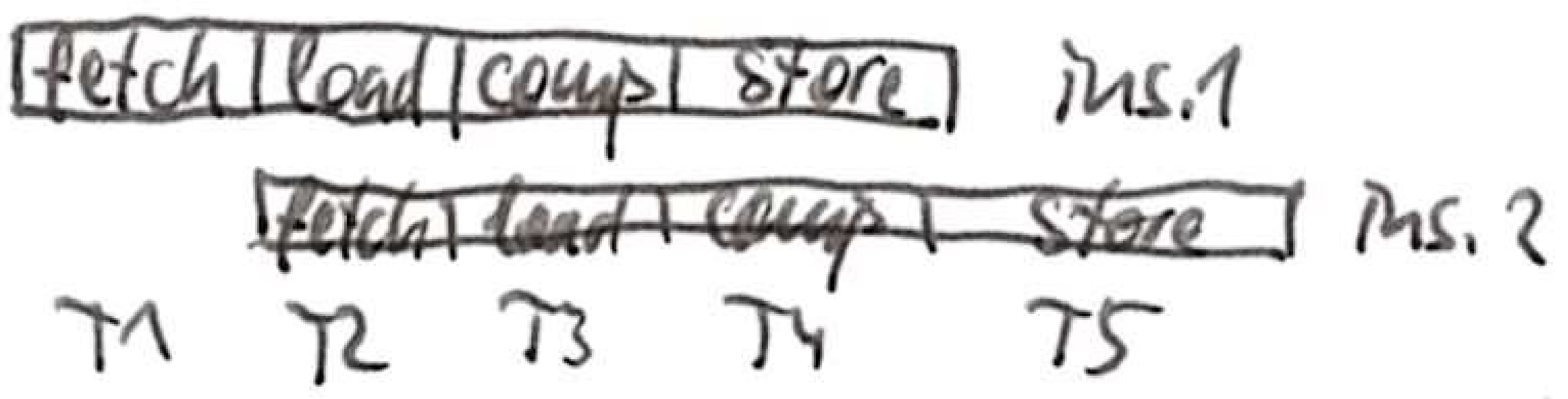
Physical memory (same-size pages) → page fault exception

Uses:
- protection of processes (private memory) ... RW(X) for 1 process
- shared memory ... RW(X) in multiple processes
- shared library ... R in multiple processes
- lazy allocation ... read-only shared zeroes, copy on write
- fork ... using copy-on-write mapping
- swapping ... store seldom-used pages to disk, read back on access

- caching – RAM is slow (CPU executes ~ 10^9 instructions/second, RAM latency is tens of ns) ↓ can be better only for small memory (speed of light etc.)
  - idea: small, very fast memory inside the CPU which remembers frequently used data] called a cache
  - caches 64B chunks of data (cache lines)
  - strategy:
    - write-through vs. write-back
    - when cache fills up: evict least-recently used item (LRU)
  - real caches have limited associativity → cache aliasing
  - multiple levels of caches
  - example: accessing a matrix row by row vs. column by column
    - row by row ↳ sequential in memory
    - column by column ↳ every access is a cache miss → very slow

  → modelling of caches, cache-oblivious algorithms (not at this lecture)

- improving execution of instructions
  - CPU works in cycles, historically 1 instruction took multiple cycles ↩ e.g.:
    - ① fetch & decode
    - ② load operands
    - ③ compute result
    - ④ store result
  - pipelining



all units of CPU always busy

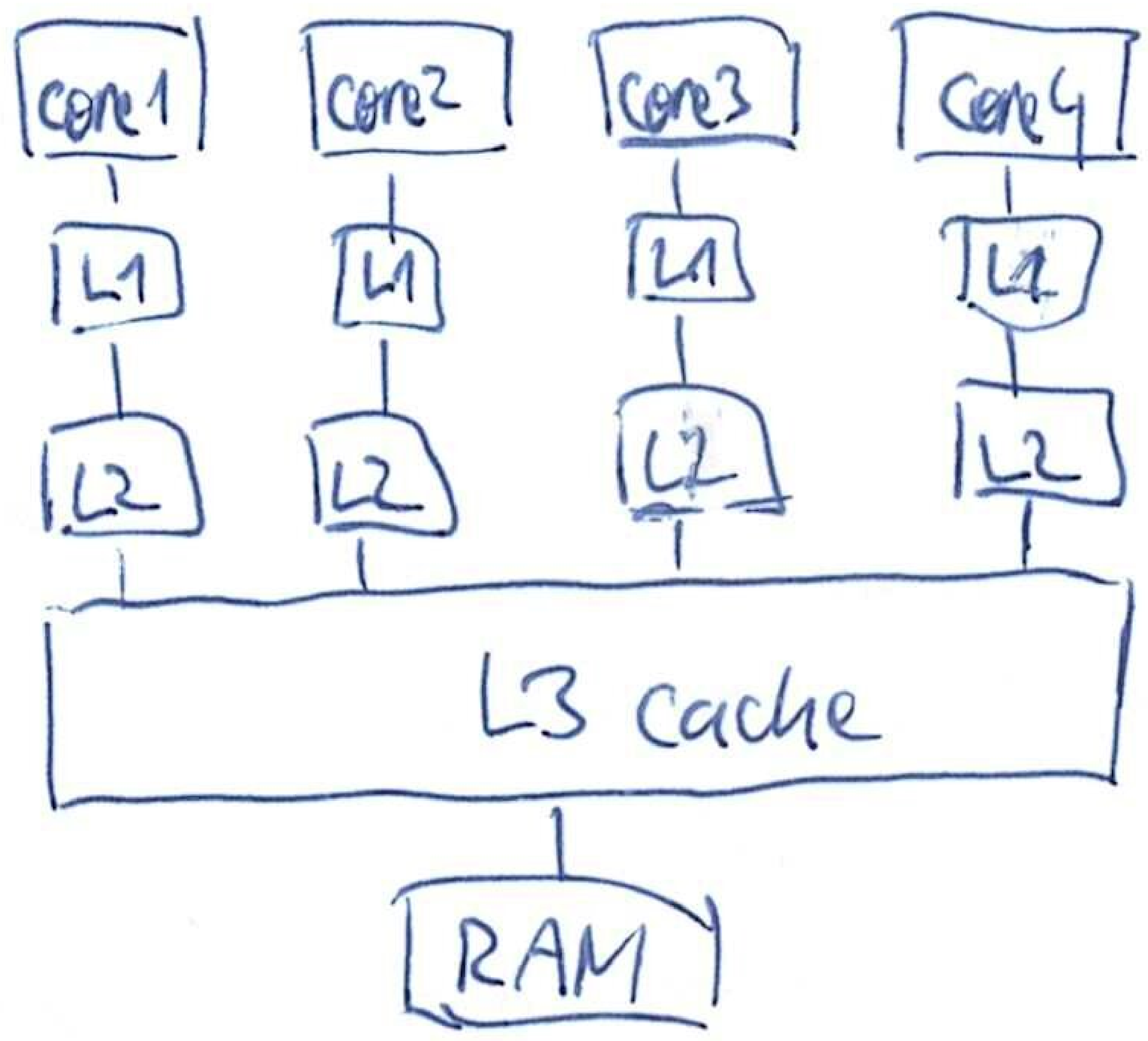
problems: dependencies & (conditional) jumps

  - superscalar CPU: multiple units for different types of instructions, can run in parallel
    → scheduling instructions to units
  - jump prediction

all this is transparent to SW (well, almost: Meltdown & other bugs)

- multiple processors sharing memory (SMP = Symmetric Multi-Processing)
  - OS schedules processes on processors – real parallelism
  - hard to get right: locking in SW, cache coherency protocols in HW
- multi-core processors: SMP on a single chip

For example:



typical sizes

32 KB code + 32 KB data

256 KB unified

8 MB unified

16 GB

- multi-threaded cores: two cores sharing their execution units & caches
  - unclear benefits (can even make things worse!)
- virtual machines: simulating a whole machine within a process
  - including supervisor mode → the VM can run its own OS
  - including virtual peripherals
  - CPUs have special support for VMs (e.g., nested paging in MMU)

Relationship with theory

- will ignore most machine-dependent constants
- concentrate on asymptotics ⇒ all machines (roughly) equal
- use simple mathematical machines instead
- I/O and caches need special treatment

} rest of the semester

Models of Computation

history: beginning of 20th century: people asking for "mechanical procedures" for solving math problems – e.g., solving integer polynomial equations

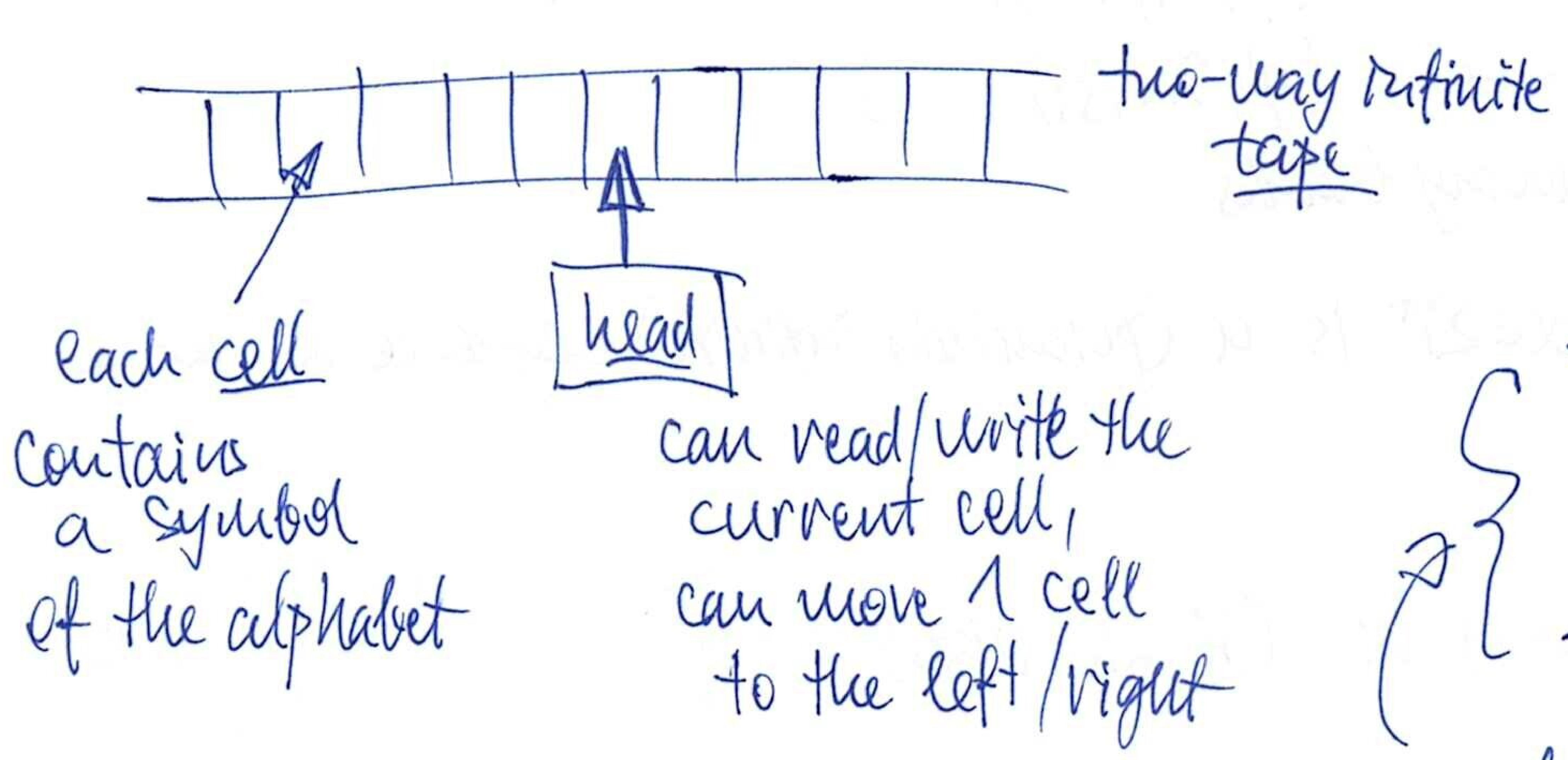
193x: Gödel, Church, Kleene, Turing: formal definitions of computation (all of them equivalent, yet different)

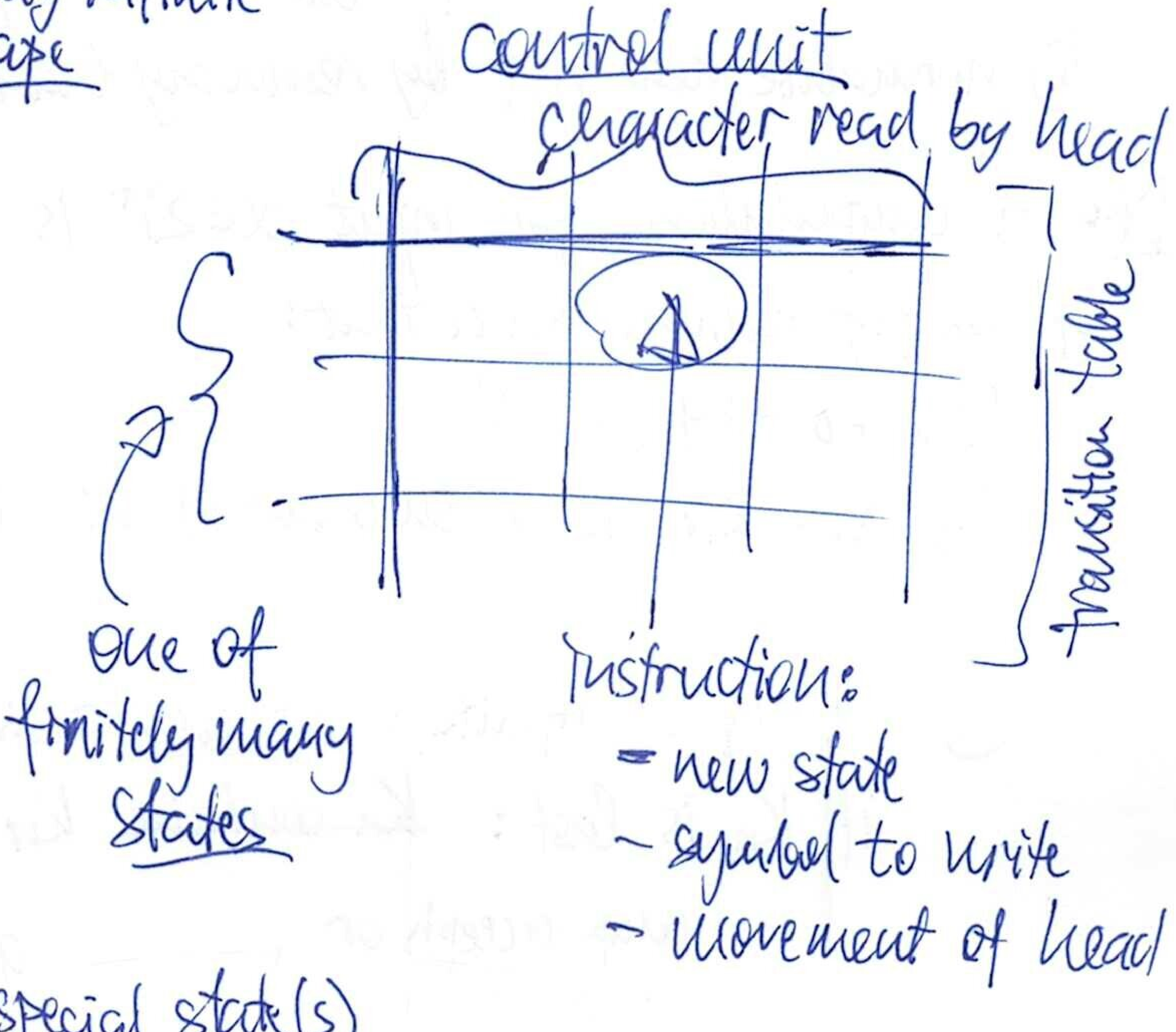
What problems we want to solve?
↳ fix "language"

- Σ_i – finite alphabet of symbols (characters) – examples: $\{0,1\}$, $\{a \dots z\}$, math symbols
  ↓ every alphabet can be encoded in every other (assuming $|\Sigma| \geq 2$)
- Σ_i^* – set of all words (finite sequences) over Σ_i
  - ε … empty word
  - $|\alpha|$ … length
  - $\alpha\beta$ … concatenation
  - symbol ≈ 1-symbol word
  - $\alpha[i]$ … i-th symbol (starting with 0)
  - $\alpha[i:j] = \alpha[i] \dots \alpha[j-1]$ … subword
  - $\alpha[:j] = \alpha[0:j]$ … prefix
  - $\alpha[i:] = \alpha[i:|\alpha|]$ … suffix
- problem: function from Σ_i^* to Σ_i^*
- decision problem: $f: \Sigma_i^* \to \{0,1\}$
  ↳ also viewed as language $L \subseteq \Sigma_i^*$: $\alpha \in L \Leftrightarrow f(\alpha) = 1$ (characteristic function of a set)
- usually we find encoding of inputs (e.g. polynomials) to strings
  - concrete encoding doesn't matter (they can be converted algorithmically)
  - what happens if the input string is not a valid encoding?
    ↳ suppose we always answer ε or 0 in such cases.

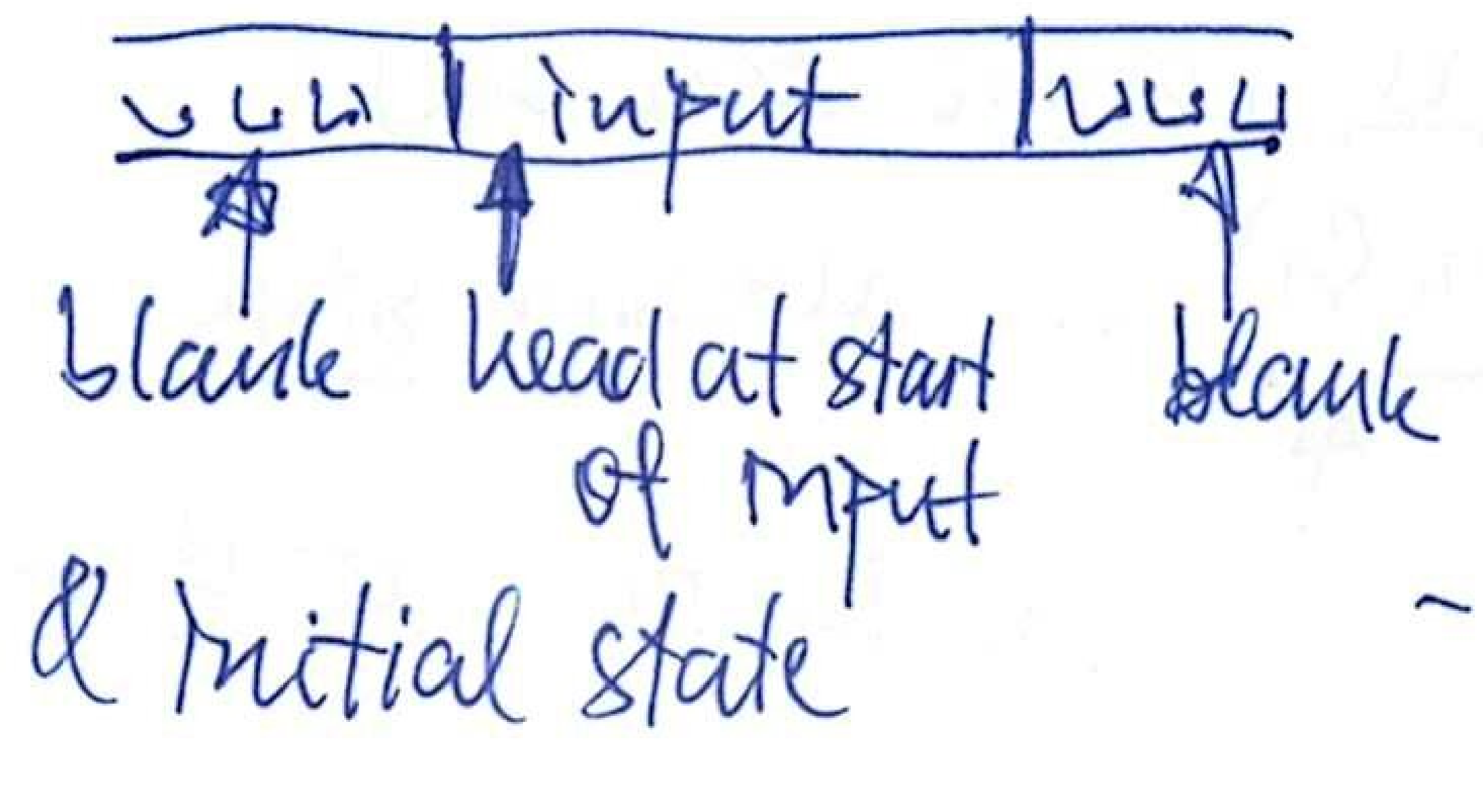
Turing Machines

motivation: a mathematician with finite mind working on an infinite blackboard





– start:



– finish: special state(s)

Now formally... ⑥

Df: A Turing machine consists of:

- Q ... a finite set of states
- $q_0 \in Q$ ... initial state
- $q_+, q_- \in Q$ ... final states (accepting & rejecting) $\}$ q_0, q_+, q_- all distinct
- Σ_I ... non-empty finite input alphabet
- $\Gamma \supseteq \Sigma_I$ ... finite work alphabet
- $\sqcup \in \Gamma \setminus \Sigma_I$ ... blank symbol
- $\delta : (Q \setminus \{q_+, q_-\}) \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \bullet, \rightarrow\}$ ... transition function

Df: Configuration of the TM: $(q, \lambda, \varrho) \in Q \times \Gamma^* \times \Gamma^*$

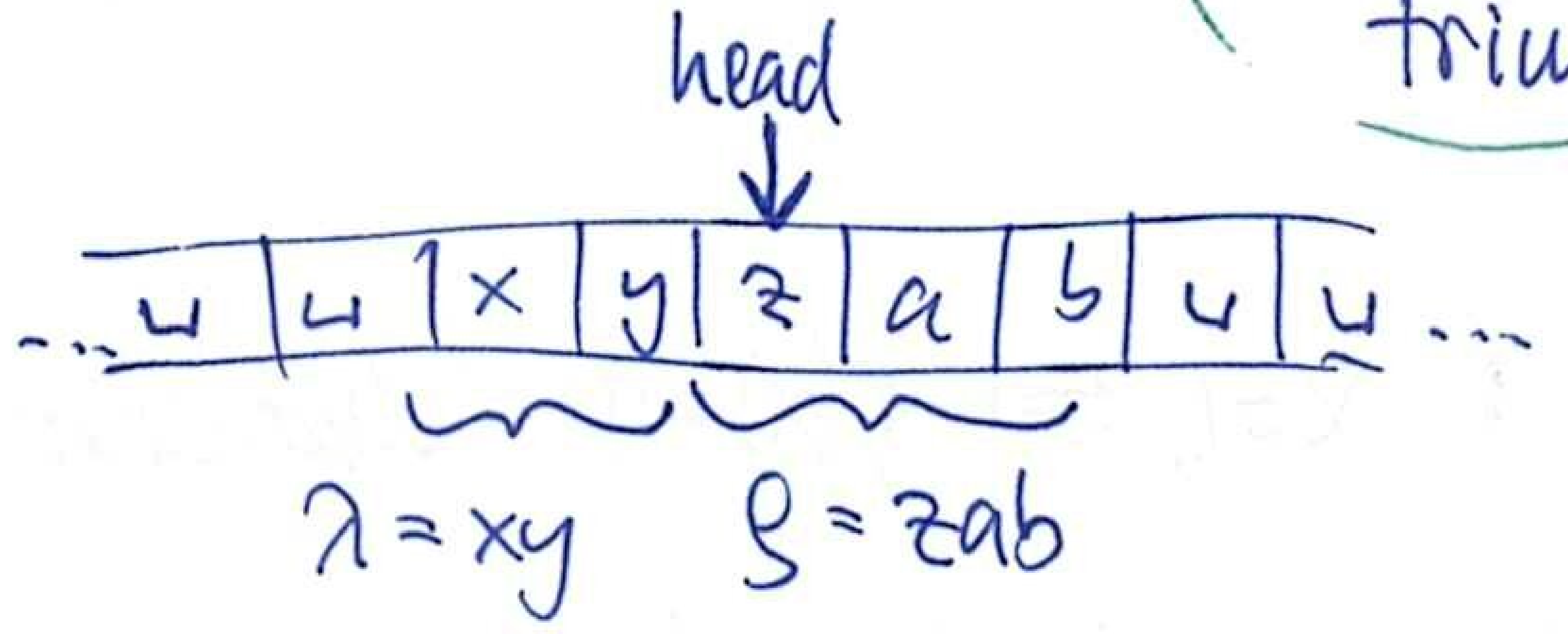
q: current state

λ: left part of tape before the head with leading blanks trimmed

ϱ: right part of tape from the head onwards with trailing blanks trimmed

normalization

Example:



$\lambda = xy \quad \varrho = zab$

👁 $\lambda\varrho$ = non-blank part of tape

on empty tape, all positions of the head are the same configuration

Df: A configuration (q, λ, ϱ), $q \neq q^+, q^-$ has a successor defined in this way:

① extend λ by a leading $\sqcup$, ϱ by a trailing $\sqcup$

② now, $\lambda = \lambda_1 y$, $\varrho = x\varrho_1$ for some $x, y, \lambda_1, \varrho_1$

③ evaluate $\delta(q, x)$ ... get (q', x', dir)

④ execute instruction: if $dir = \bullet$ ... $(q', \lambda, x'\varrho_1)$
if $dir = \leftarrow$ ... $(q', \lambda_1, yx'\varrho_1)$
if $dir = \rightarrow$ ... $(q', \lambda x', \varrho_1)$
$\}$ new config.

⑤ normalize new λ, ϱ by removing blanks

Df: A computation for input $\alpha \in \Sigma_I^*$ is a (potentially infinite) sequence $K_0, K_1, \dots$ of configurations such that:

① $K_0 = (q_0, \varepsilon, \alpha)$

② $\forall i$: K_{i+1} is a successor of K_i (if K_{i+1} exists) ← therefore state q^+ or q^- never occurs except for the last config of a finite seq.

③ if seq. is infinite: the computation diverges (doesn't terminate)

if K_n is last: ~~Kn contains~~ $K_n = (q_n, \lambda_n, \varrho_n)$ ... machine stops

comp. accepts or rejects the input ← q_+ or q_-

$\lambda_n \varrho_n$ is the output of computation

Df: Computability:

Function $f: \Sigma_i^* \to \Sigma_i^*$ is computable
also general recursive
$\equiv \exists M$ Turing machine s.t.
$\forall \alpha \in \Sigma_i^*$ $M(\alpha)$ halts and outputs $f(\alpha)$
M on input α (& its computation)

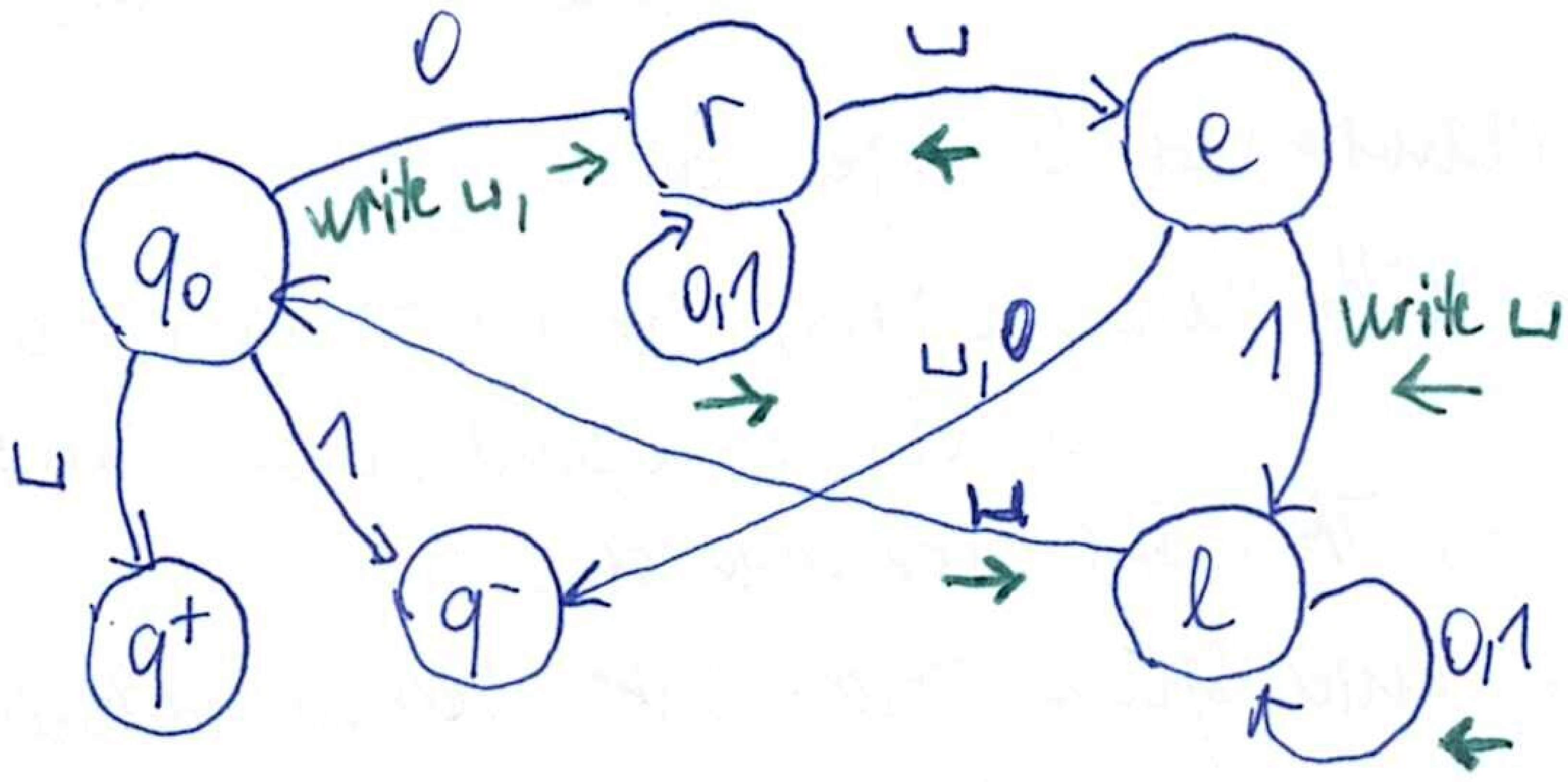
Function $f: \Sigma_i^* \to \Sigma_i^* \cup \{\uparrow\}$ $(\uparrow \notin \Sigma_i)$
divergence
is partially computable $\equiv \exists M$ T.m.
also partially recursive
s.t. $\forall \alpha \in \Sigma_i^*$: if $f(\alpha) = \uparrow$: $M(\alpha)$ diverges
else $M(\alpha)$ halts & outputs $f(\alpha)$

Language $L \subseteq \Sigma_i^*$ is computable
also recursive
$\equiv \exists M$ T.m. s.t.
$\forall \alpha \in \Sigma_i^*$ $M(\alpha)$ always halts
& ($\underbrace{\text{accepts}}_{\text{ends in } q^+}$ $\alpha \iff \alpha \in L$)
→ equivalent to char. fn of L computable

Language $L \subseteq \Sigma_i^*$ is partially computable
also recursively enumerable
$\equiv \exists M$ T.m. s.t.
$\forall \alpha \in \Sigma_i^*$ $M(\alpha)$ halts in state q^+ $\iff \alpha \in L$.
→ equivalent to $c_L(\alpha) = \begin{cases} 1 & \text{if } \alpha \in L \\ \uparrow & \text{if } \alpha \notin L \end{cases}$ partially computable

Idea: Time & Space spent by computation
Time: # configurations visited $\cong$ # instructions executed
Space: # cells visited by the head
} will serve as basis for complexity theory (later)

Example: Recognizing $\{0^n 1^n\} \subseteq \{0,1\}^*$ by accepting / rejecting.
Idea: erase first 0 & final 1, repeat.

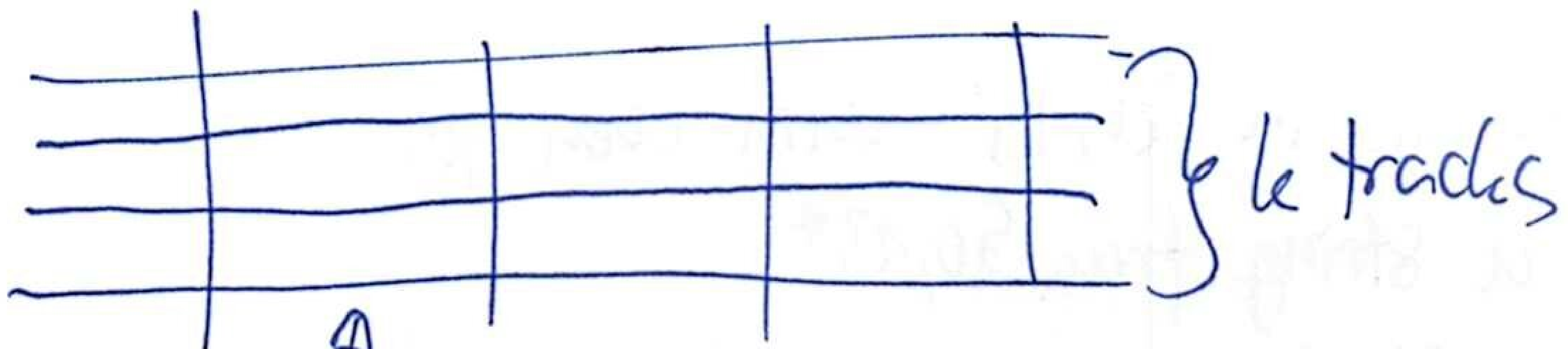


$\Sigma_i = \{0,1\}$
$\Gamma = \{0, 1, \sqcup\}$
$Q = \{q^0, q^+, q^-, r, e, l\}$

doesn't matter

δ	0	1	$\sqcup$
q_0	$(r, \sqcup, \rightarrow)$	$(q^-, ?, ?)$	$(q^+, ?, ?)$
r	$(r, 0, \rightarrow)$	$(r, 1, \rightarrow)$	$(e, \sqcup, \leftarrow)$
e	$(q^-, ?, ?)$	$(l, \sqcup, \leftarrow)$	$(q^-, ?, ?)$
l	$(l, 0, \leftarrow)$	$(l, 1, \leftarrow)$	$(q_0, \sqcup, \rightarrow)$

Ideas:
- Encode multiple variables with finite domains in the state – state is a tuple
- Multi-track tape



each cell contains k-tuple

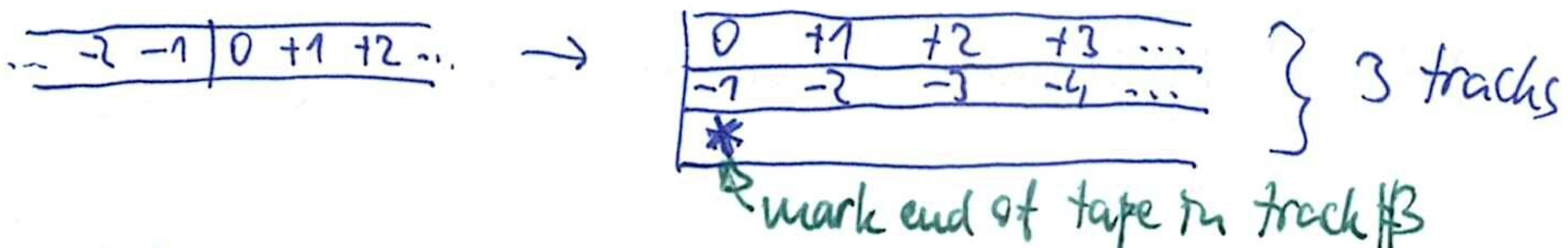
head reads/writes k-tuples
but all tracks share head position

Remember to en/decode tape at start/end of computation.

Variants of the TM (robustness of definition)

① **One-way infinite tape** ... 1-way → 2-way trivial
2-way → 1-way "fold tape in half" simulation

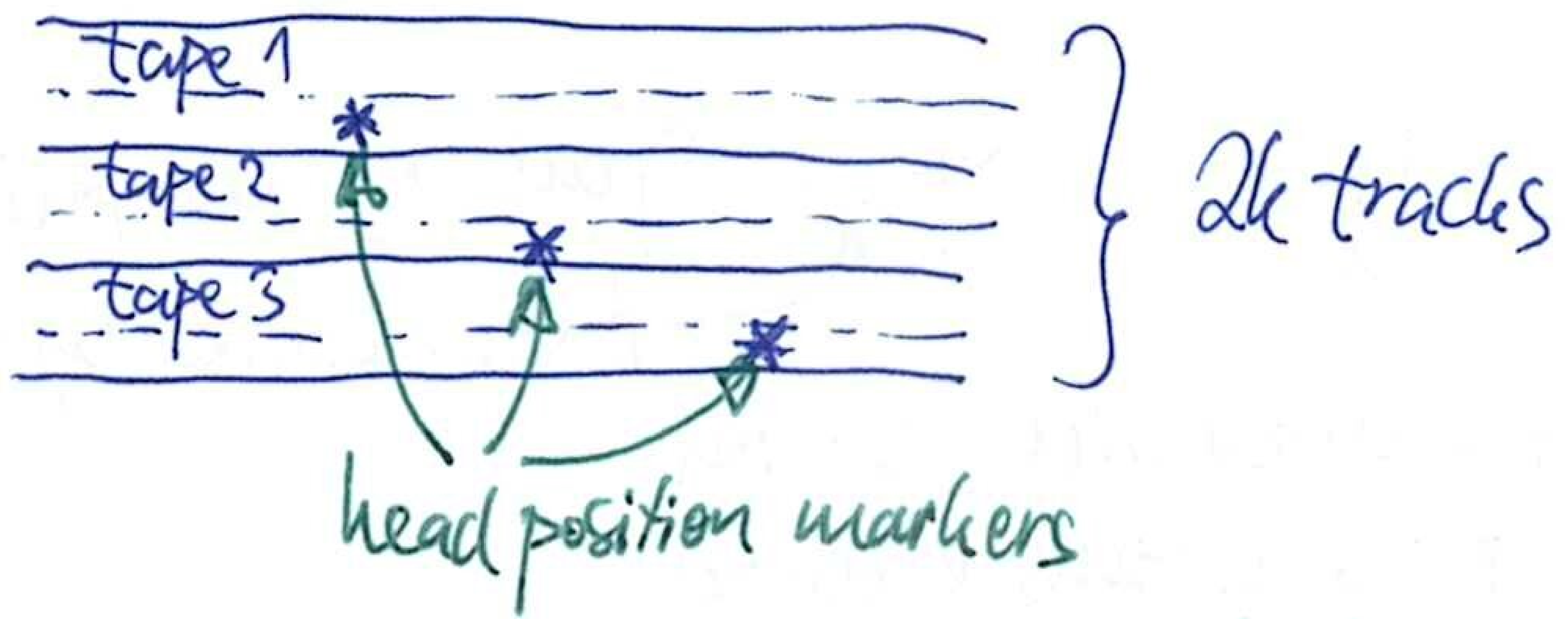
↳ equally powerful
(set of computable functions remains unchanged...)



state contains "positive half-tape" switch.

② **k tapes** with independent heads (but sharing a common work alphabet (WLOG))

- transition function: $(Q \setminus \{q^+, q^-\}) \times \Gamma^k \to Q \times \Gamma^k \times \{\leftarrow, \rightarrow, \bullet\}^k$
- configuration, successor, computation easy to extend
  - start: tape #1 contains input, other tapes empty
  - end: tape #1 contains output
- Equally powerful ... k-tape → 1-tape:



1 step of orig. machine can be simulated by scanning the whole tape 2 times:
  - find all heads, record symbols they see in state
  - write symbols back & move heads

- But complexity changes: $\{0^n 1^n\}$ requires super-linear time with 1 tape but can be solved in $O(n)$ time with 2 tapes

actually, you can do it in $\Theta(n \log n)$ time (exercise)

- Later: there is a more efficient reduction of k tapes to 2.

③ **Randomized TM** ... read-only tape with random bits, moving to the right only
↳ what is a computation then? ...

④ **Oracles** (functions defined outside the TM, but accessible to it)
- oracle tape: write query there, enter special state, tape changes contents to the answer

⑤ **Interactive TM** ... outside world can be modelled as an oracle ☺

⑥ **Exercise: 2-Dimensional tape** ... head moves ←, →, ↑, ↓, •

... in some sense, the most powerfull physically feasible computer is a TM with 3-D tape (or maybe 2-D only to allow heat spreading...)

Exercises:
- accept strings in $\{0,1\}^*$ with even #1
- reverse a string from $\{0,1\}^*$
- add / multiply numbers written in binary ... what's the complexity?
  ↑ non-negative integers

Random-Access Machine

- 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 = any, X = 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
      <, >, =, ≠, ≤, ≥
- 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, …, [N] = x_N

Out: [0] = sum

Temporary: [-1] = copy of N, [-2] = current index

Program:

```
       [-1] ← [0]            copy N
       [0] ← 0               initialize sum
       [-2] ← 1              start with x1
LOOP:  if [-2] > [-1] jump END
       [0] ← [0] + [[-2]]
       [-2] ← [-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 $\max \log(1+|x|)$, $x \in \{$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

⑩

We will assume that the RAM gets a string $\in \Sigma^*$ as input:
[0] = length, [1], [2], … = symbols of the string (encoded as integers)
(this is WLOG 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], … using some numbering of the work alphabet
- [0] will specify how far the — " — stretches.
- [-1] = current position of head
- position in program represents machine state
- can simulate 1 step of the TM in constant time.

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 R=A, copy data from M to D
- memory write: similar, but need to expand cells if they are to 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 $\alpha \in \Sigma_i^*$ ≡ computation on α ends in state q^+

👁 rejects α ⟺ stops in q^- or runs forever (diverges)

↳ • Language $L(M)$ accepted by M $:= \{\alpha \in \Sigma_i^* \mid M \text{ accepts } \alpha\}$

• Language L is decided by M ≡ M always stops & $L = L(M)$.

Df: • Language L is computable (a.k.a. decidable / recursive) ≡ $\exists$ 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) ≡ $\exists$ 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_i can be arbitrary, these are proper classes. WLOG we can fix $\Sigma_i = \{0,1\}$ to make R and RE sets.

👁 $R \subseteq RE \subseteq 2^{\{0,1\}^*}$ ← all languages over $\{0,1\}$

↑ ↑ are these strict? Watch out...

Enumeration (or: why "recursively enumerable"?)

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

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

Thm: $L \in RE$ ⟺ L is enumerable

Pf: ⟸ we want to accept $\alpha \in L$ ... run enumerator, compare printed strings with α

YES ⇒ stop in q^+, NO ⇒ continue

enumerator stops ⇒ stop in q^-

⟹ we have TM M accepting L, let's build enumerator for L using M:

expand square ... for each (α, t) simulate M on α for t steps ... if it stops in q^+, in exactly t steps, print α 👁 prints all $\alpha \in L(M)$

strings in length-lexicographic order ($\alpha \leq_{LL} \beta \equiv |\alpha| < |\beta| \lor |\alpha| = |\beta| \;\&\; \alpha \leq_{Lex} \beta$)

[binary numbers with leading 1 removed]

Homework: $L \in R$ ⟺ L is enumerable in $\leq_{Lex}$ order.

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

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 $\Sigma_I = \{0,1\}$

alphabet: $\Gamma = \{x_0, x_1, x_2, \ldots, x_m\}$

$\uparrow$ 0, $\uparrow$ 1, $\uparrow$ 2, $\underbrace{\ldots, x_m}$ other symbols in arbitrary order

directions: $\{d_0, d_1, d_2\}$

$\leftarrow$ $\rightarrow$ $\bullet$

states: $Q = \{q_0, q_1, q_2, \ldots, q_n\}$

$\uparrow$ initial, $\uparrow$ q^+, $\uparrow$ q^-, $\underbrace{\ldots q_n}$ other states

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

transitions: $\delta(q_i, x_j) = (q_k, x_l, d_t) \rightarrow$ encode as $1^i 0 1^j 0 1^k 0 1^l 0 1^t 0$

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

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

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

$\uparrow$ isomorphism of TMs (defined in the obvious way)

↳ in fact, there are multiple such codes (we numbered Q, Γ arbitrarily &c.)

● $L_\alpha := L(M_\alpha)$ … $\forall \alpha$ $L_\alpha \in RE$

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

\# codes is countable ⇒ RE is countable … but $2^{\{0,1\}^*}$ uncountable

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

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

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

Df: Universal language $L_u := \{ \langle \alpha, \beta \rangle \mid \alpha, \beta \in \{0,1\}^* \;\&\; \beta \in L_\alpha \}$

"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

$\uparrow$ WLOG multi-tape

$\uparrow$ #states and $|\Gamma|$ are not bounded

tape K: copy of the code α

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

tape M: 1^m

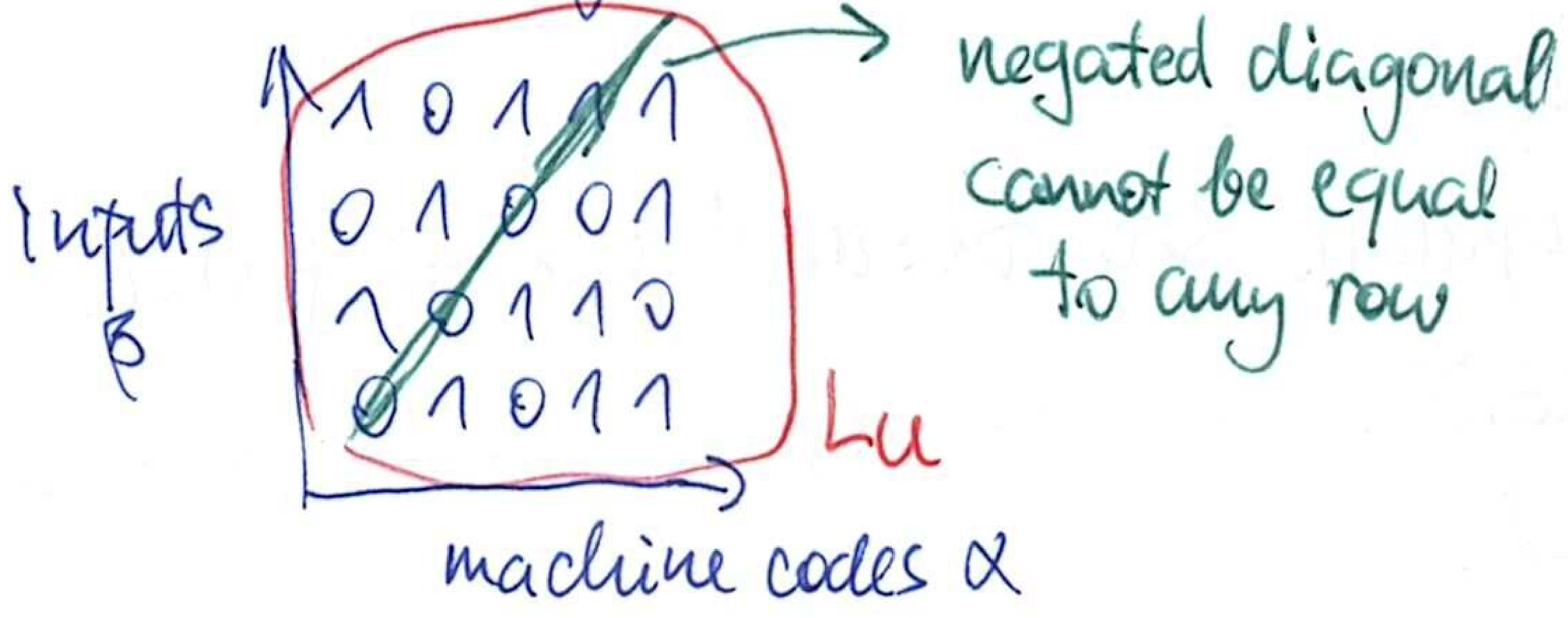
$\uparrow$ head on T encodes position of M_α's head tape S: current state of M_α stored as $1^j 0^{|S|-j}$

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 (for $R \subsetneq RE$), witnessed by $\overline{L_u}$ (for $RE \subsetneq 2^{\{0,1\}^*}$)
- RE is not closed under complement

Exercise: Are R and RE closed under ∩ or ∪?

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

Pf: ⇒ trivial, because $R \subseteq RE$ & R closed under complement.

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

Operations on machine codes

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

} all these are computable functions

More decision problems regarding machine codes

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

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

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

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

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

↳ let's generalize this.

Df: Many-to-one reduction between languages:

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

↑ sometimes denoted $K \to L$

$\leq_m$ 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 $L_d \leq_m \overline{L_u}$ & $L_d \notin RE$ to show $\overline{L_u} \notin RE$.

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

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

↳ $L_\gamma = \emptyset$ if $M_\alpha(\beta)$ diverges
$L_\gamma = \{0,1\}^*$ otherwise

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

④ $L_{empty} \xrightarrow{\leq_m} \overline{L_{halt}}$: for given α, construct M_β which ignores its input, simulates M_α on all inputs in parallel & stops if $M_\alpha(\gamma)$ stops on some γ ... $\langle \beta, \varepsilon \rangle \in \overline{L_{halt}}$
$\Leftrightarrow L_\alpha = \emptyset$.
$\Leftrightarrow \alpha \in L_{empty}$.

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

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

- Semantic properties of machines

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

$L_P := \{ \alpha \in \{0,1\}^* \mid L_\alpha \in P \}$ ... 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 $L_{halt} \to L_P$ for every non-trivial P.

Proof: Assume that $L_P \in R$ for some P.
WLOG $\emptyset \notin P$ ... otherwise use $\overline{P}$ ... $L_{\overline{P}} = \overline{L_P}$, so it's also in R.
Find $L_\omega \in P$ ... exists as P is non-trivial
Reduction: if we want to answer $\langle \alpha, \beta \rangle \in L_{halt}$, i.e. if $M_\alpha(\beta)$ halts
construct M_γ which does on input δ:

this is computable

- run M_α on β (1)
- run M_ω on δ (2)

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

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

What is the "hardest" language in a class?

- Let $\mathcal{C}$ be a set of languages.
- L is <u>$\mathcal{C}$-hard</u> $\equiv \forall K \in \mathcal{C} : K \leq_m L$
- L is <u>$\mathcal{C}$-complete</u> $\equiv L$ is $\mathcal{C}$-hard & $L \in \mathcal{C}$

more precisely, it's $\mathcal{C}$-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 α: $L_\alpha = K$
Then f reducing K to L_u is $\beta \mapsto \langle \alpha, \beta \rangle$

Also: If K is $\mathcal{C}$-complete and $K \leq_m L$ for $L \in \mathcal{C}$, then L is $\mathcal{C}$-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 $\mathbb{Z}$? → Matijasević theorem

both in RE \ 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 <u>relative</u> 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: $R[A] \subsetneq RE[A] \subsetneq 2^{\{0,1\}^*}$

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

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

Df: **Arithmetical hierarchy:** classes $\Sigma_n, \Pi_n, \Delta_n$ for $n \in \mathbb{N}$

- $\Sigma_0 = \Pi_0 = \Delta_0 = R$
- $\Sigma_{n+1} = RE[\Sigma_n]$
- $\Pi_{n+1} = co\text{-}RE[\Pi_n]$
- $\Delta_{n+1} = R[\Sigma_n]$

this means: $RE[\mathcal{C}] = \bigcup_{L \in \mathcal{C}} RE[L]$

→ We have:

$\Sigma_1 = RE$
$\Pi_1 = co\text{-}RE$
$\Delta_1 = R[R] = R$

oracles from R do not add power to TM

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

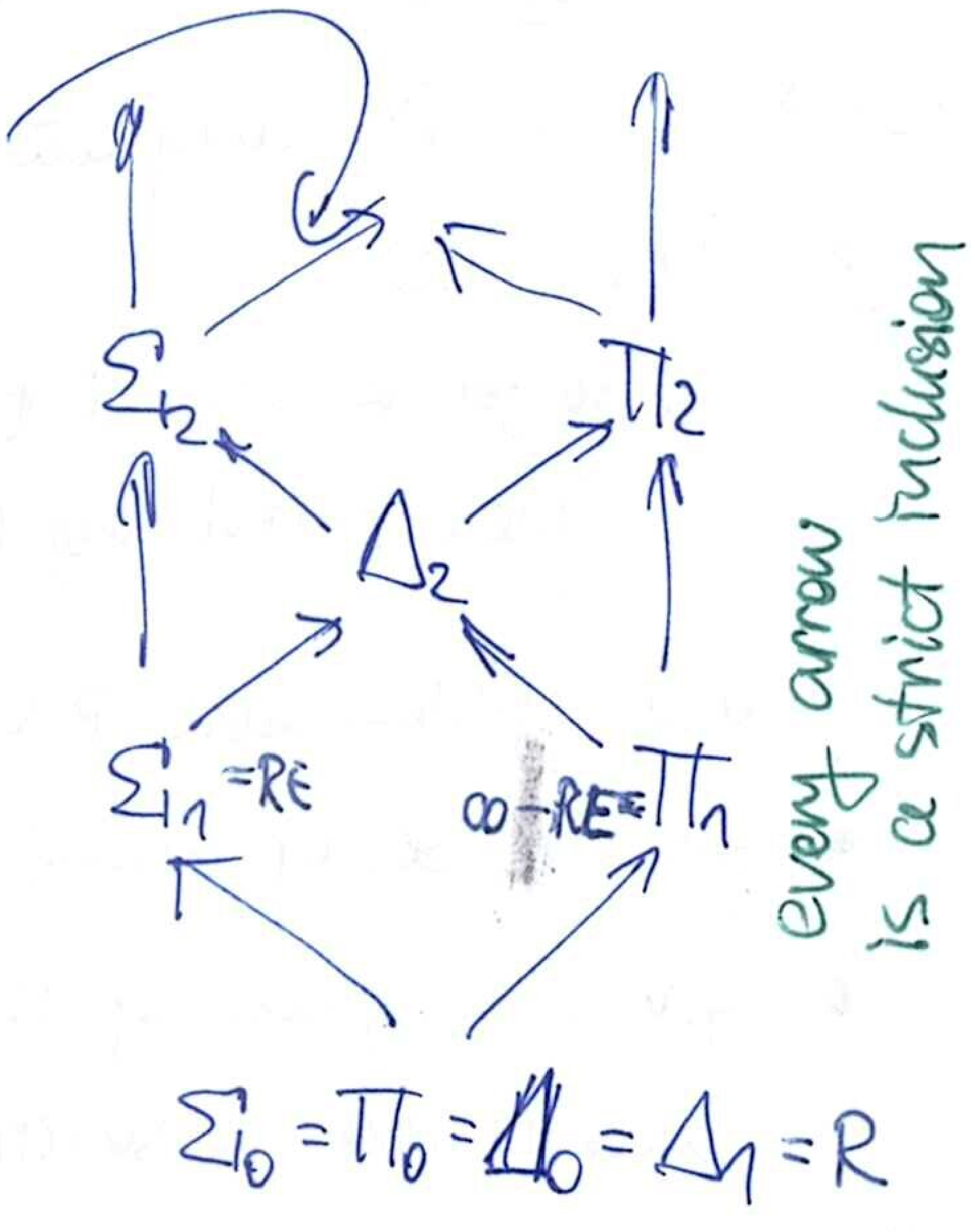
this inclusion is strict ~~as $L_u[\Sigma_n]$ is RE~~

~~$RE[L_u^n] \not\subseteq \Sigma_n$, otherwise we would have~~

$\Sigma_n \subseteq R[\Sigma_n] = R[L_u^n] \subsetneq RE[L_u^n] = \Sigma_{n+1}$

- $L_u^1 = L_u$
  $L_u^{n+1} = L_u[L_u^n]$ → 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)
- $\Pi_{n+1} = co\text{-}RE[\Sigma_n]$
- $\Pi_{n+1} = co\text{-}\Sigma_{n+1}$

Also:
$\Sigma_n \subseteq \Delta_{n+1}$ … and this is strict as Σ_n is not closed under complement, while Δ_{n+1} is
$\Pi_n \subseteq \Delta_{n+1}$ … we can negate oracle's answer
$\Delta_{n+1} = \Sigma_{n+1} \cap \Pi_{n+1}$ … relative Post's thm.
$\Delta_{n+1} \subseteq \Sigma_{n+1}$ … this is $R[L_u^n] \subsetneq RE[L_u^n]$
$\Delta_{n+1} \subseteq \Pi_{n+1}$ … analogous for co-RE



Quantified formulas

- every language in R can be interpreted as a predicate $\varphi(\alpha)$ with string parameter φ – decidable predicates
- $\psi(\beta) \equiv \exists\alpha\, \varphi(\alpha, \beta)$ lies in RE
  … and every $L \in RE$ can be written in this way
  [α = # steps after which a machine stops]
- $\psi(\beta) \equiv \forall\alpha\, \varphi(\alpha, \beta)$ … this is co-RE $\left(\neg\forall\alpha\, \varphi(\alpha,\beta) \Leftrightarrow \exists\alpha\, \underbrace{\neg\varphi(\alpha,\beta)}_{\text{decidable}}\right)$
- … $\exists\alpha_1 \exists\alpha_2\, \varphi(\alpha_1, \alpha_2, \beta)$ is again RE … we can say $\exists\alpha$ s.t. $\alpha = \langle\alpha_1, \alpha_2\rangle$ & decode α inside φ
- $\exists\alpha_1 \forall\alpha_2\, \varphi(\alpha_1, \alpha_2, \beta)$ is Σ_2
- in general: $\exists\alpha_1 \forall\alpha_2 \exists\alpha_3 \dots Q_n\alpha_n\, \varphi(\alpha_1 \_ \alpha_n, \beta)$ is Σ_n
  $\forall\alpha_1 \exists\alpha_2 \forall\alpha_3 \dots Q_n\alpha_n\, \varphi(\alpha_1 \_ \alpha_n, \beta)$ is Π_n

$\subseteq \Sigma_2$: $\exists\alpha_1 \forall\alpha_2\, \varphi(\dots) \Leftrightarrow \exists\alpha_1\, \neg\underbrace{(\exists\alpha_2\, \varphi(\dots))}_{\text{can be answered by oracle } L_u \in \Sigma_1}$ ← so this is in $RE[\Sigma_1] = \Sigma_2$

$\supseteq \Sigma_2$ consider $L \in \Sigma_2 = RE[\Sigma_1]$:
$\exists\gamma$ computation of TM ~~the Σ_1~~ $[L_u]$ $\exists\delta$ queries for L_u (γ, 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 $\exists\forall\ldots$

- 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)

δ ↑ computation of M including all oracle queries & answers

input α ↑ decidable

itself ↑ decidable

oracle L_u ↑ for positive answers: $\exists\gamma_1 - \gamma_k$
$\psi(\gamma_1, \text{question}1)$
$\vdots$
$\psi(\gamma_k, \text{question}k)$

Σ_1-formula for L_u

for negative answers: $\forall\varepsilon_1 - \varepsilon_l \neg\psi(\varepsilon_1, \ldots)$
& …

together:
$$\exists\delta\big(\underset{\substack{\uparrow\\ \text{code of}\\ \gamma_1-\gamma_k}}{\exists\gamma}\ \underset{\substack{\uparrow\\ \text{code of}\\ \varepsilon_1-\varepsilon_l}}{\forall\varepsilon}\ \varphi(\alpha,\delta,\gamma,\varepsilon)\big)$$

… but this is $\exists\langle\delta,\gamma\rangle\ \forall\varepsilon\ \varphi(\ldots)$ as we need.

COMPLEXITY

Df:

- For a multi-tape machine M, define run time $t_M(\alpha)$ for input α as # steps before computation $M(\alpha)$ halts.
  - $t_M : \underbrace{\{0,1\}^*}_{\text{generally } \Sigma_i^*} \to \mathbb{N}^*$ ← $\mathbb{N} \cup \{\infty\}$ for divergent computations

sorry for confusing notation: this * is not iteration, but:

- Time complexity of machine M: $T_M : \mathbb{N} \to \mathbb{N}^*$ s.t. $T_M(n) = \max\limits_{\alpha \in \Sigma_i^n} t_M(\alpha)$.

M always halts ⇔ $T_M(n)$ finite for all n.

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

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

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

$\forall^*$ ↑ for all but finitely many exceptions

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 $f \in o(g) \equiv \lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$ ← strict upper bound

⑤ $f \in \omega(g) \equiv g \in o(f)$ ← strict lower bound

Examples: $f : n \mapsto 5n^3 - 7n^2 + 18$ $\in O(n^3), O(n^5), O(2^n)$ $\in o(n^4)$
$\in \Omega(n^3), \Omega(n^2), \Omega(1)$ $\in \omega(n^2)$
$\in \Theta(n^3)$ "drop lower-order terms & multiplicative constant"

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

Dependence of complexity on # tapes

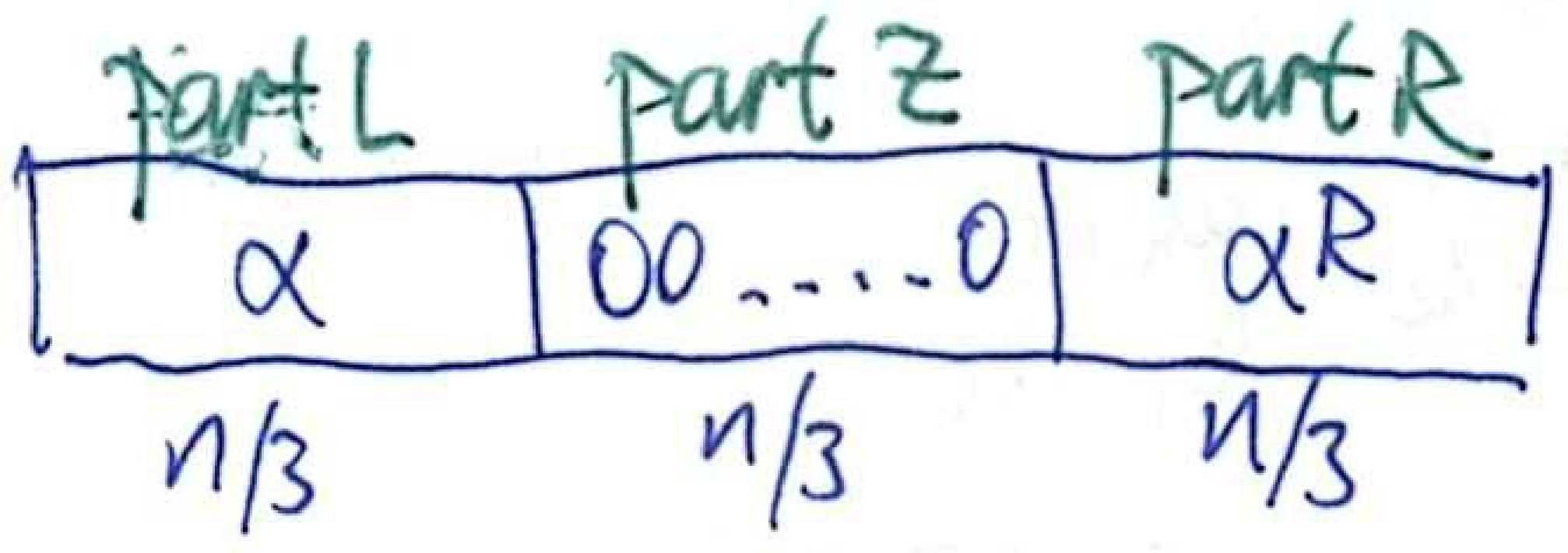
Df: $L_{PAL} = \{ \alpha\underbrace{\alpha^R}_{\text{reversed}} \mid \alpha \in \{0,1\}^* \}$ "even palindromes"

- trivial 2-tape TM deciding L_{PAL} in $\Theta(n)$ time.
- but all machines we found with 1 tape run in $\Theta(n^2)$ time!

Thm: Every 1-tape machine deciding L_{PAL} runs in time $\Omega(n^2)$.

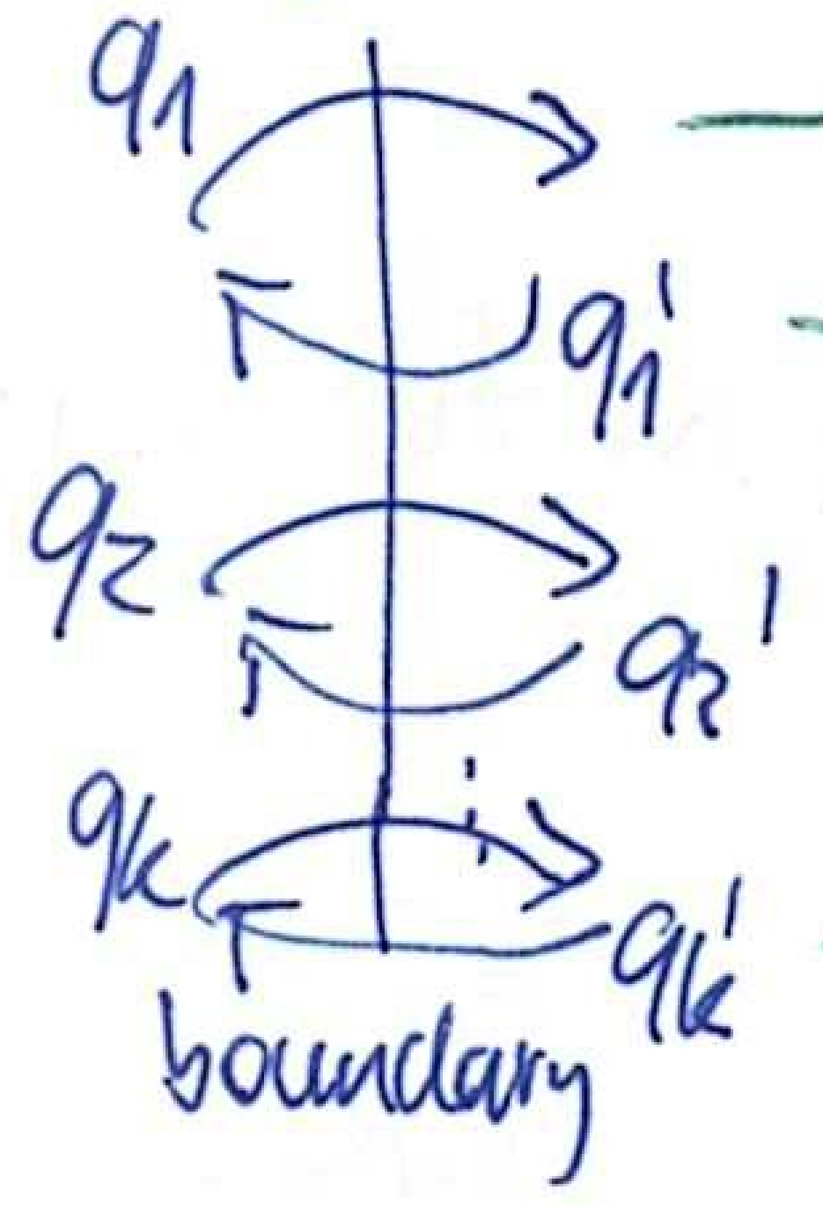
Proof: Assume there is M deciding L_{PAL} s.t. $T_M(n) \in o(n^2)$. ← that is: $\forall \varepsilon > 0\ \forall^* n : T_M(n) < \varepsilon n^2$.

- Consider inputs of type: for $6 \backslash n$ (n will be chosen later)



← answer is YES for every such string

- Consider boundary between 2 zeros in part Z & how computation of M crosses it:



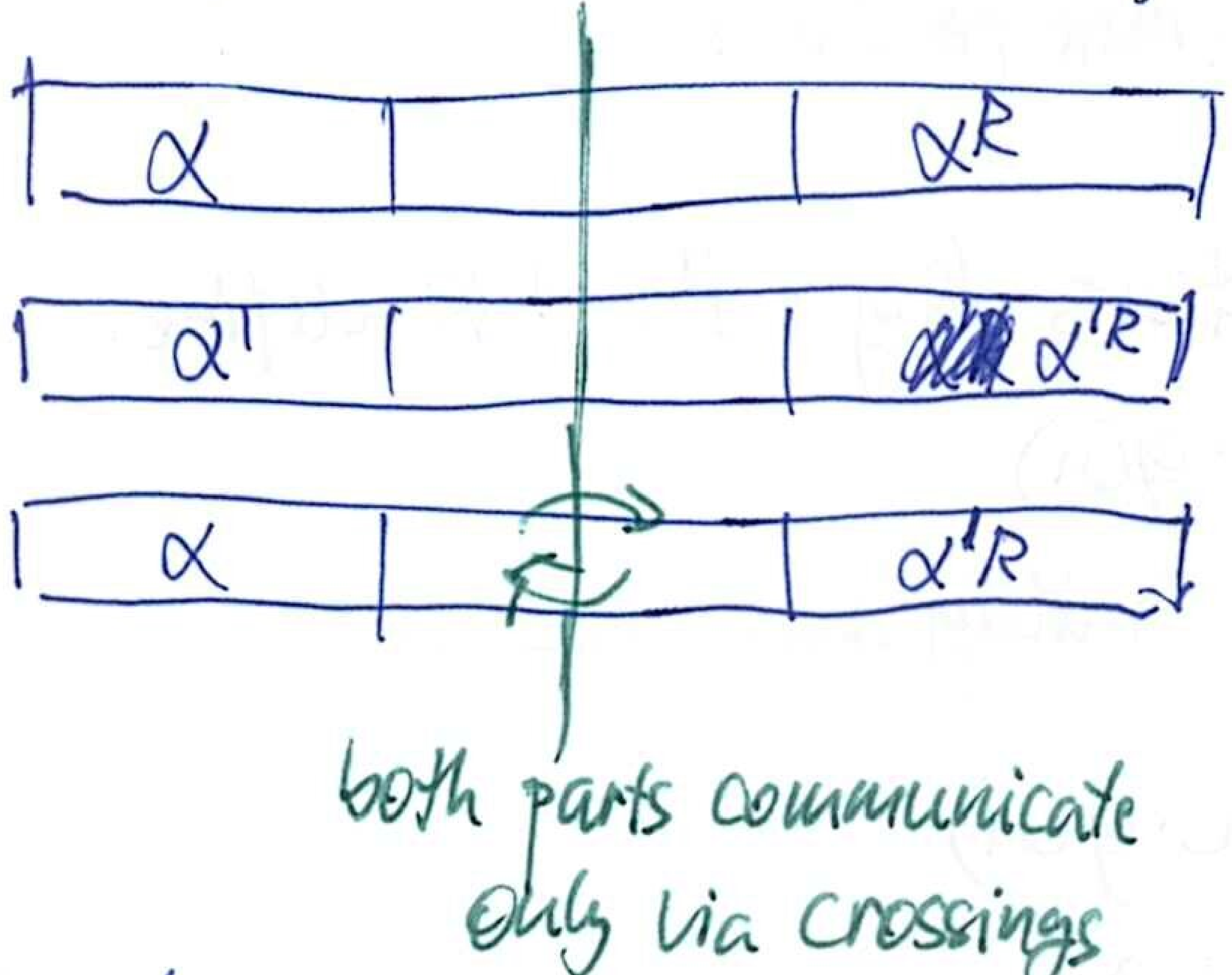
— first, we cross from left to right in some state q_1

— then we return in state q_1'

— WLOG computation ends in L (we can modify M to "rewind" without affecting $o(n^2)$)

↳ crossing sequence $(q_1, q_1'), \ldots, (q_k, q_k')$

- If two inputs with α, α' have the same C.S. for the same boundary:



answers YES

answers YES

also answers YES, although the answer should be NO ↯

- Similarly if they have same C.S. for different boundaries. (part Z can have odd length, but that implies NO anyway)
- Let's use P.H.P. (Pigeon-hole principle) to show that such α, α' exist:

$$\#\text{ c.s. of length } k = |Q|^{2k}$$

$$\#\text{ c.s. of length at most } k \le c \cdot |Q|^{2k}$$ for some constant c (via sum of geom. series)

If # c.s. < # inputs, then ∃ two inputs with the same C.S.

↳ so we want $c \cdot |Q|^{2k} < 2^{n/3}$ ... $2^{\underbrace{\log c + 2k \log|Q|}_{\le 3k \log|Q|}} < 2^{\frac{n}{3}}$ ... $k < \frac{n}{9\cancel{k} \log|Q|}$

- P.H.P. once again (for every input we can find boundary with a small # crossings):
  - we have n/3 boundaries
  - Σ of lengths of their C.S. $\leq T_M(n) < \varepsilon n^2$
  - $\Rightarrow \exists$ boundary with at most $\frac{\varepsilon n^2}{n/3} = 3\varepsilon n$ crossings
- now set ε such that we have:

$$\text{min. \# crossings} \leq 3\varepsilon n < \frac{n}{9 \log|Q|}$$

such ε exists & inequality satisfied for n large enough

- so there are 2 inputs with the same C.S. ⇒ mixing produces contradiction.

Thm: For every multi-tape TM M there is 1-tape TM M' s.t. $L(M) = L(M')$ & $T_{M'}(n) \in O(T_M(n)^2)$.

Proof: Analyze reduction from previous lectures. [hint: at most $T_M(n)$ tape cells used on each tape]

Thm: For every multi-tape TM M there is 2-tape TM M' s.t. $L(M) = L(M')$ & $T_{M'}(n) \in O(T_M(n) \cdot \log T_M(n))$.

(proof omitted)

Time complexity classes

- $\text{DTIME}(f) := \{ L \subseteq \{0,1\}^* \mid \exists M$ multi-tape TM deciding L $\overbrace{\text{in time } O(f)}^{T_M \in O(f)} \}$

$f: \mathbb{N} \to \mathbb{R}$

sometimes just TIME(f) ["D" is for "deterministic"]

every finite Σ s.t. $|\Sigma| \geq 2$ would work

this will be the default

makes things easier, but for TMs it's not necessary because of time compression thm.

- We will require f to have these properties:
  ① non-decreasing
  ② $\forall n\ f(n) \geq n$
  ③ time-constructible ≡ ∃ TM M_f which for input 1^n produces output $1^{f(n)}$ in time $O(f(n))$

"proper time complexity function"

- $P := \bigcup_{i \geq 1} \text{DTIME}(n^i)$ ← polynomial-time decidable languages

Why we like P:
- Corresponds (roughly) to "efficiently solvable"
- indepent of model of computation (RAM gives the same P as TM)
- polynomials are the smallest set of functions containing constants & identity and closed under addition, multiplication and composition.

Examples:
- reachability in graphs
- evaluation of Boolean formulas

Classes of functions

$\{0,1\}^*$

- $\mathrm{DTIMEF}(f) := \{g : \Sigma_1^* \to \Sigma_1^* \mid \exists \text{ TM } M \text{ computing } g \text{ in time } O(f)\}$
- $\mathrm{PF} := \bigcup_{i \geq 1} \mathrm{DTIME}(n^i)$

encoded in binary

if $|f(n)| \in \mathrm{poly}(n)$: $L_g := \{\langle \alpha, i \rangle \mid i\text{-th bit of } g(\alpha) \text{ is } 1\}$

$O(n^k)$ for some fixed k

$L_g \in P \iff g \in \mathrm{PF}$.

So studying only languages in P is WLOG.

Consider these problems:

as usually: suitably encoded

path vs. walk ← can repeat

doesn't repeat vertices

① HAMILTONPATH Input: undirected graph G, vertices u,v
Question: $\exists$ path in G with endpoints u,v containing all vertices (exactly) once.

② 3-COLORING Input: undirected graph G
Q: $\exists f : V(G) \to \{1,2,3\}$ s.t. $\forall \{u,v\} \in E(G) : f(u) \neq f(v)$
coloring of G with 3 colors

③ INDEPENDENTSET Input: undirected graph G, $k \in \mathbb{N}$
Q: $\exists A \subseteq V(G) : |A| \geq k$ & $\forall u,v \in A : \{u,v\} \notin E(G)$

④ CLIQUE Input: G, $k \in \mathbb{N}$
Q: $\exists A \subseteq V(G) : |A| \geq k$ & $\forall u,v \in A : (u = v \lor \{u,v\} \in E(G))$

⑤ 0,1-EQuations a.k.a. ZOE Input: matrix A, vector b with 0/1 entries
Q: $\exists x \in \{0,1\}^n : Ax = b$ (evaluated in integers, not $\mathbb{Z}_2$)
WLOG $b = \mathbf{1}$

⑥ SAT (Boolean satisfiability): Input: Boolean formula $\varphi(x_1 — x_m)$ in CNF
Q: $\exists x_1 — x_m \in \{0,1\}$ s.t. $\varphi(\vec{x})$ is true.

For all these: we are looking for something we can easily recognize (poly-time), but we don't know how to find it in poly time.

Clause

$\varphi \equiv (x_1 \lor \neg x_2 \lor x_3) \land (\ldots) \land (\ldots)$

literals: either x_i or $\neg x_i$

(restriction to CNF is WLOG, see later)

(we cannot use thm. from Logic about equivalent formulas in CNF, because it blows up size exponentially)

Reductions will again prove themselves useful:

Df: For languages K, L: $K \leq^P_m L \equiv \exists f \in \mathrm{PF}$ s.t. $\forall \alpha \in \Sigma_1^*$ $\alpha \in K \iff f(\alpha) \in L$.

polynomial-time many-to-one reduction

Properties of $\le_m^P$:

- reflexive & transitive (quasi-order)
- ∃ incomparable languages (exercise)
- $K \le_m^P L$ and $L \in P \Rightarrow K \in P$ [composition of 2 algorithms running in poly. time is again poly-time]

That is, P is closed under reductions.

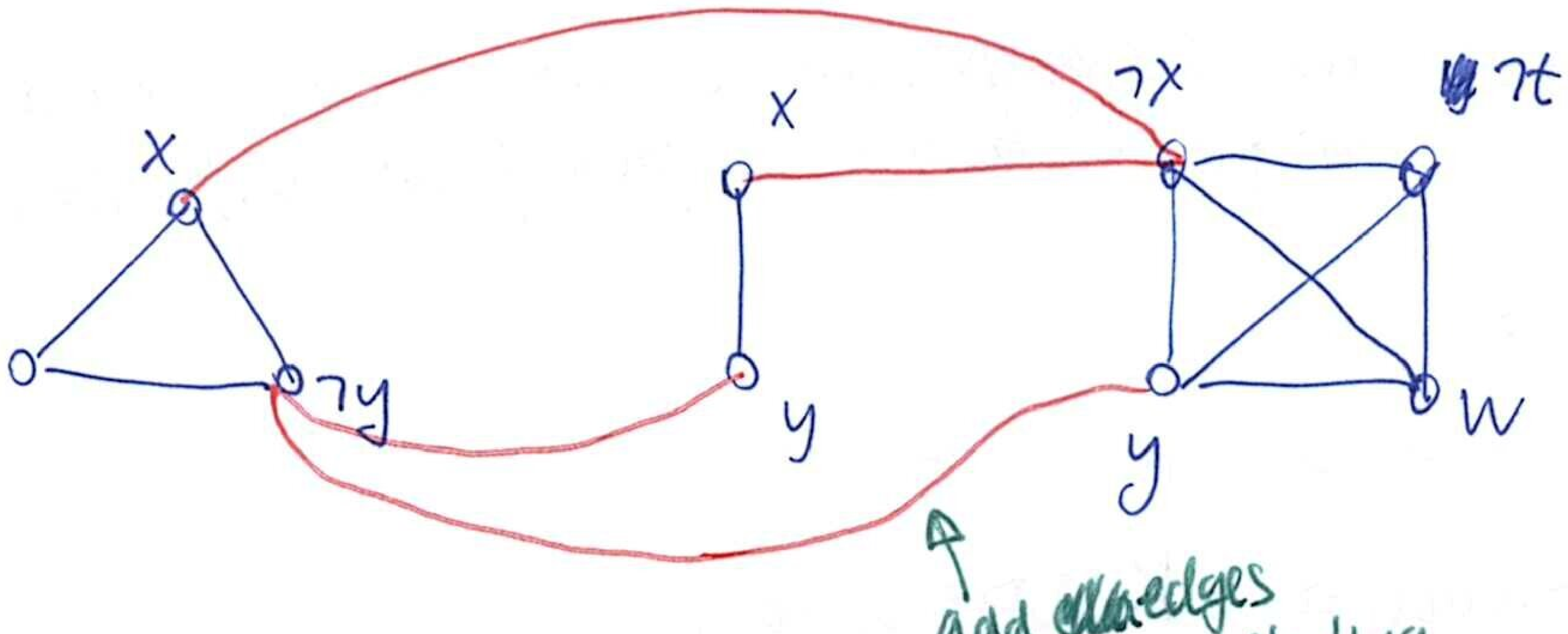
size of input for ALG2 is at most time complexity of ALG1

Example: SAT $\le_m^P$ INDEPSET

$$(x \vee \neg y \vee z) \wedge (x \vee y) \wedge (\neg x \vee y \vee \neg t \vee w)$$

given a formula ↓ produce a graph, $k := \#$clauses ⇓ from each subgraph we must select exactly 1 vertex

each clause gets complete subgraph labelled with literals of the clause



add edges between conflicting literals

- ∃ satisfying assignment: for each clause, pick 1 satisfied literal, put its vertex to the indep. set } → got I.S. of size k
- ∃ indep. set of size k: each vertex ~~selected~~ in I.S. selects a variable which will be set to 0/1 to satisfy the corresponding clause, red edges guarantee that we won't set var to both 0 and 1
  - remaining variables set arbitrarily

  ↳ got satisfying assignment
- the reduction runs in poly. time

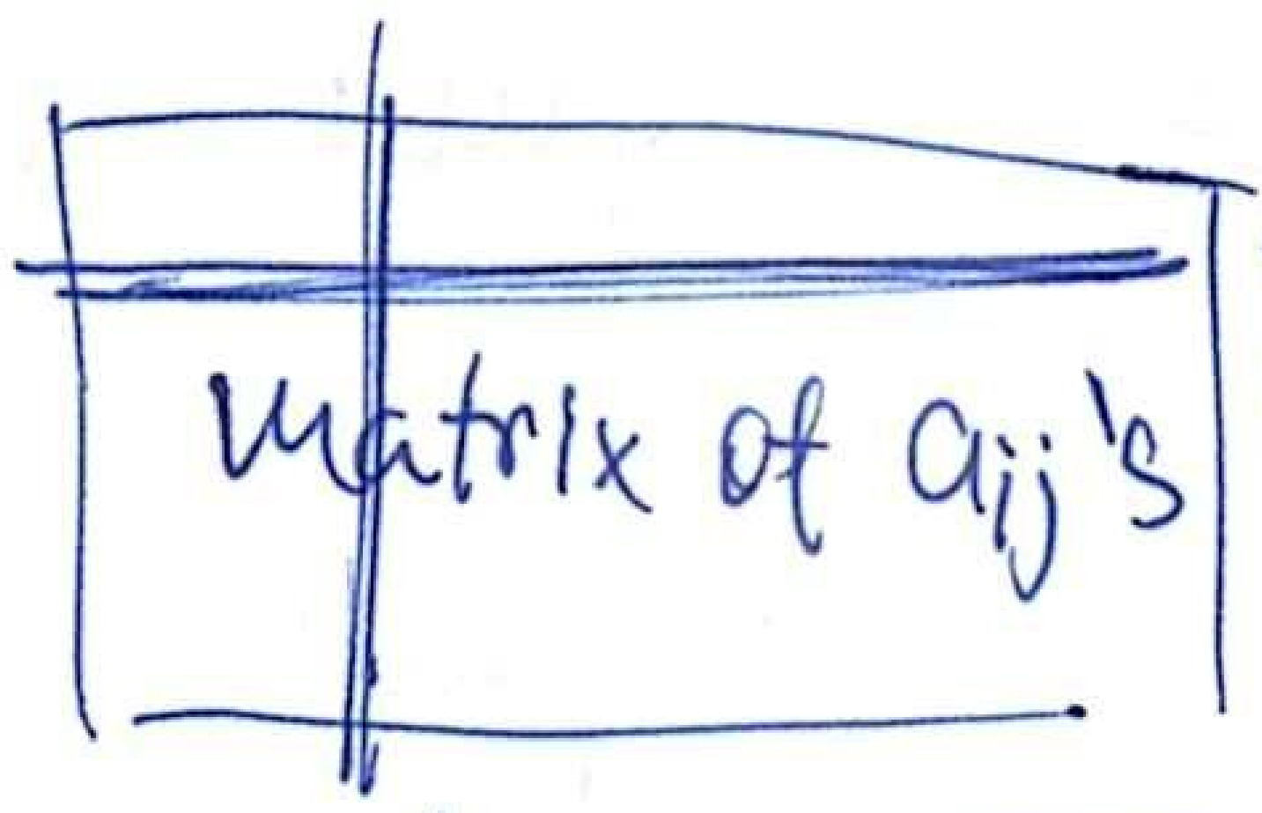
Example: INDEPSET $\le_m^P$ SAT ... given G, k, construct formula φ s.t. φ is satisfiable $\Leftrightarrow$ G has ind. set of size k

Variables: x_1 — x_n : vertex v_i selected to ind. set

a_{ij} for $1 \le i \le k$, $1 \le j \le n$: vertex j is i-th in the ind. set ← order on vertices of the set

Clauses: $\forall \{v_i, v_j\} \in E(G)$: $\neg x_i \vee \neg x_j$

$\forall i,j\ a_{ij} \Rightarrow x_j$ ← order describes the set (we allow unordered elements of set, which doesn't break anything)



$a_{ij} \Rightarrow \neg a_{ij'}$ no number used multiple times

$a_{ij} \Rightarrow a_{i'j}$ no vertex used twice or more

$a_{i1} \vee \ldots \vee a_{in}$ each number used at least once

implication $x \Rightarrow y$ is a clause $\neg x \vee y$ → so we get CNF

Exercises: ① INDEP SET $\lesseqgtr^P_m$ CLIQUE

② 3-COLORING $\leq^P_m$ SAT

Formalization of "search problems":

Df: Class of languages NP:

∃ certificate — of polynomial size — which is accepted by the verifier

$L \in NP \equiv \exists V \in P$ (verifier)

$\forall \alpha \in \Sigma^* : \alpha \in L \Leftrightarrow \left(\exists \beta \in \Sigma^* : |\beta| \in poly(|\alpha|) \ \& \ V(\alpha, \beta) \right)$

$P \subseteq NP$ ... verifier does all the work & ignores β

resembles proofs in logic: true statements have a proof, which is easy to verify
for false statements, no proof passes verification

Big question: Is $P = NP$?

↑ 1M$ price by Clay Mathematical Institute (waits for you ☺)

Df: Language L is NP-hard $\equiv \forall K \in NP : K \leq^P_m L$

L is NP-complete $\equiv$ furthermore, $L \in NP$

Lemma: Let $K \leq^P_m L$. Then:

NP is closed under reductions

① if $L \in NP$, then $K \in NP$ (just compose verifier with reduction)

② if K is NP-hard, then L is NP-hard. ($\forall M \in NP \ M \leq K \leq L \Rightarrow M \leq L$)

(2') if K is NP-complete & $L \in NP$, then L is NP-complete.

makes it easy to prove NP-completeness once we have one NP-comp. problem

Lemma: If $L \in P$ is NP-complete, then $P = NP$.

Proof: $P \subseteq NP$ is trivial, will prove $NP \subseteq P$:

Let $K \in NP$. Then $K \leq L$, which implies $K \in P$.

Thm (Cook-Levin): SAT is NP-complete.

↳ will be proven later

MORE REDUCTIONS

3D MATCHING Input: sets B (boys), G (girls), C (cats)

$J \subseteq B \times G \times C$ (triples)

Output: $\exists J' \subseteq J$ s.t. each element of $B \,\dot\cup\, G \,\dot\cup\, C$ is contained in exactly 1 triple in J'

(generalizes bipartite matching, which is in P)

3-SAT: SAT, but all clauses contain at most 3 literals (generally: k-SAT)

3,3-SAT: Furthermore, every variable occurs in at most 3 clauses. (generally: k,l-SAT)
[Extension: every literal occurs at most 2 times – i.e., the 3 occur's of a variable aren't all positive nor all negative.]

Reduction: SAT $\leq^P_m$ 3-SAT

$$\underbrace{(l_1 \vee l_2 \vee \_\_ \vee l_k)}_{\text{"long" clause with } k>3 \text{ literals}} \xrightarrow{} \underbrace{(l_1 \vee l_2 \vee t)}_{3 \text{ literals}} \,\&\, (\underbrace{l_3 \vee \_ \vee l_k \vee \neg t}_{k-1 \text{ literals}})$$

replace by

t ↑ new variable

New formula is satisfiable ⇔ the old one was.

Iterate until all long clauses are broken.

Reduction: 3-SAT $\leq^P_m$ 3,3-SAT

Replace variable x with $k>3$ occurrences by new variables x_1 — x_k.

Add clauses $(x_1 \Rightarrow x_2), (x_2 \Rightarrow x_3), \ldots, (x_{k-1} \Rightarrow x_k), (x_k \Rightarrow x_1)$

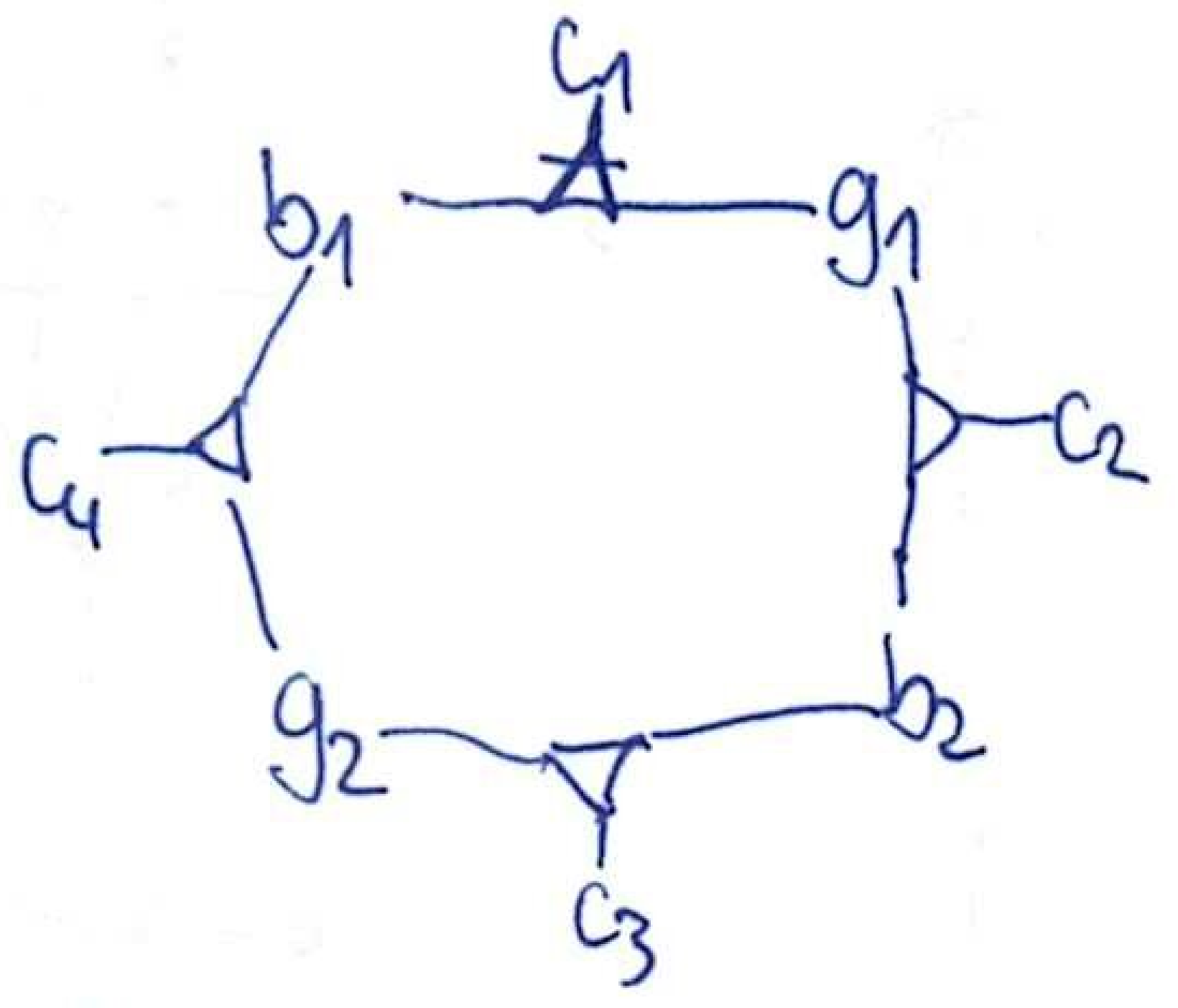
↑ this is $\neg x_2 \vee x_3$

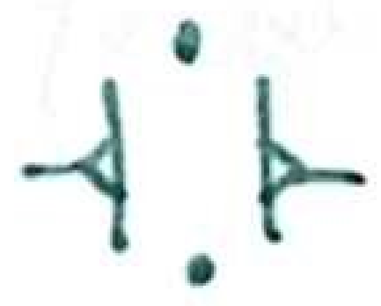
Preserves satisfiability.

Each new variable has at most 2 positive & at most 2 negative occurrences.
Can apply the transform for $k=3$, too.

Reduction: 3,3-SAT $\leq^P_m$ 3D-MATCHING

Choice gadget (for each variable)



2 states:  c_1, c_3 free ← logical 0

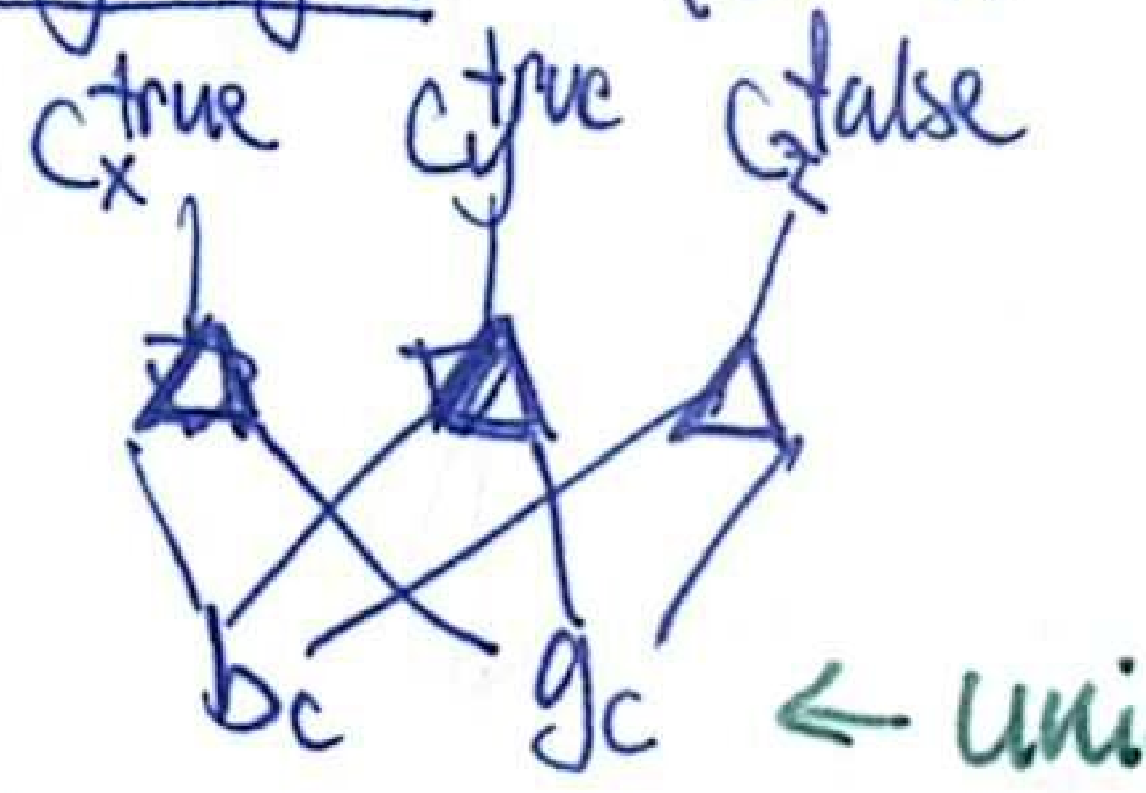
and  c_2, c_4 free ← logical 1

b_1, b_2, g_1, g_2 unique for this gadget

$c_1 \ldots c_4$ shared with clause gadgets

(also called consistency gadget)

Clause gadget for $x \vee y \vee \neg z$



← unique for this gadget

1 of the cats which are free if x is true (that is c_2 or c_4)

← each literal occurs at most 2 times in 3,3-SAT formulas, so we have enough cats for all literals

we have (4·#variables − Σ of clause sizes) free cats ⇒ add this many pairs of "universal cat lovers" which have triples for every cat

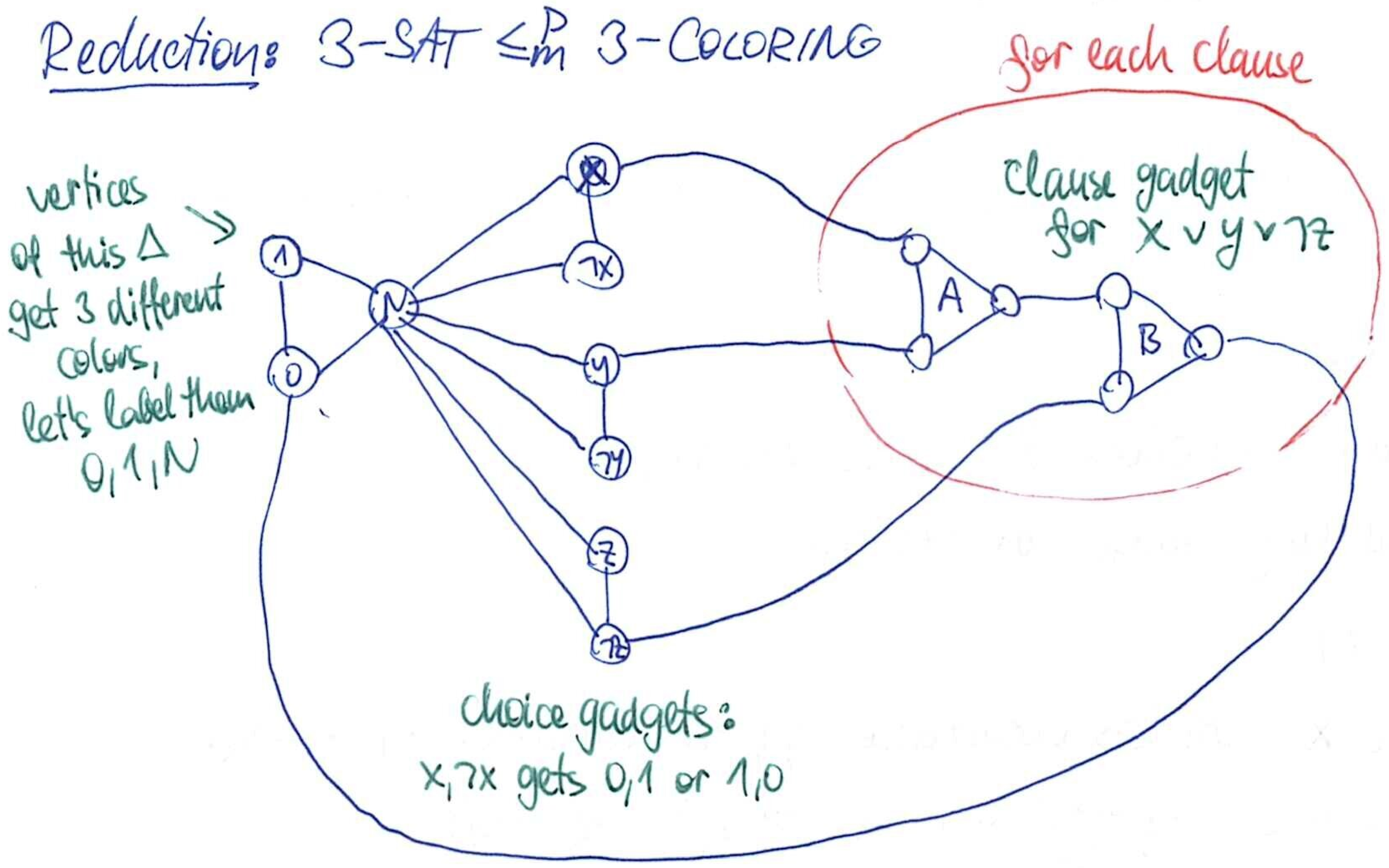
∃ matching ⇔ ∃ satisfying assignment

Exercise: 3D-MATCHING $\le^P_m$ ZOE (zero-one equations)

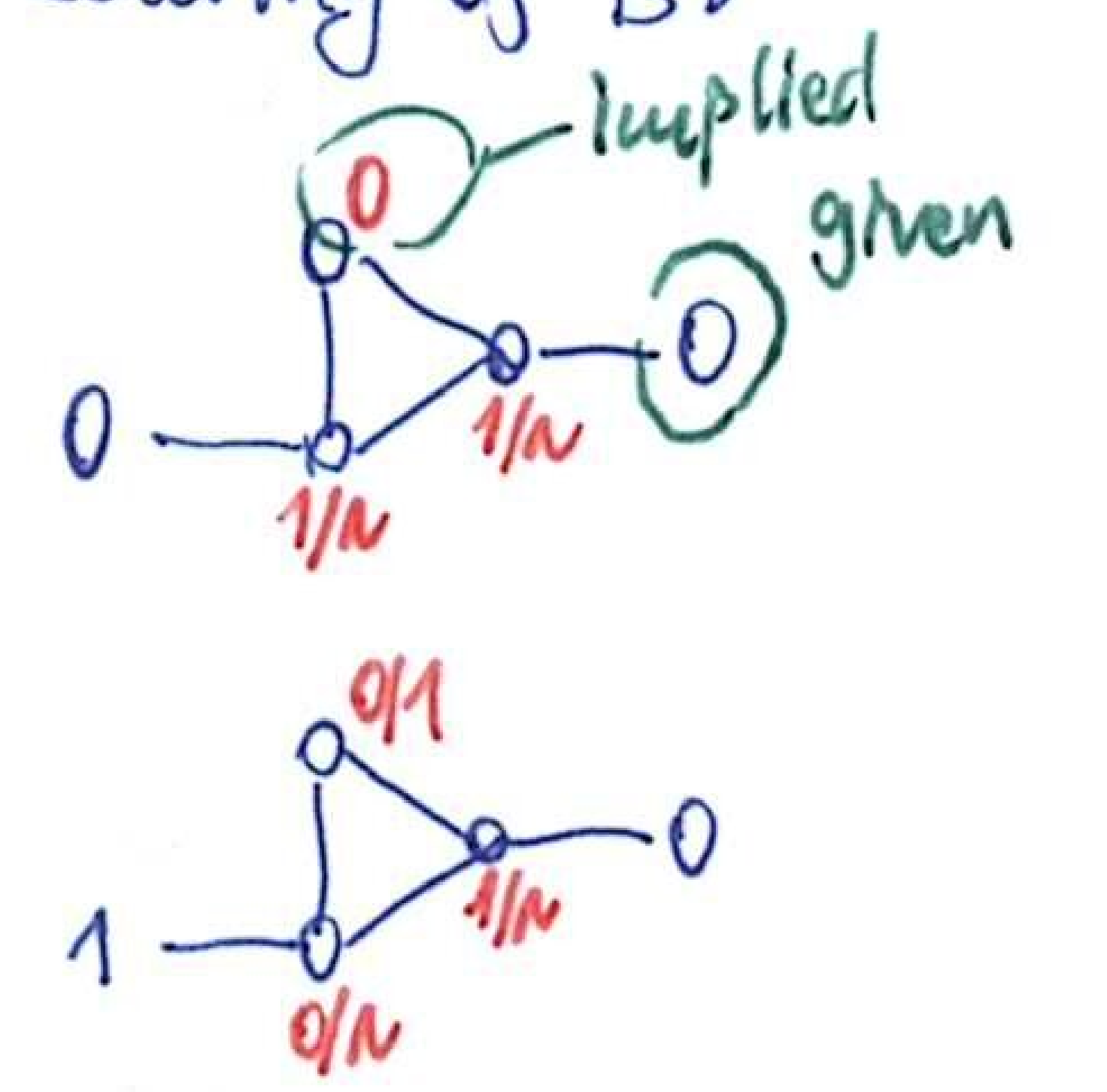
↳ & show that restriction of ZOE to equations with exactly 3 variables stays NP-complete.

[This is sometimes called 1-in-3-SAT: exactly 1 literal must be true ... no negations are needed. There also exists a direct reduction from 3-SAT to this problem.]

Reduction: 3-SAT $\le^P_m$ 3-COLORING

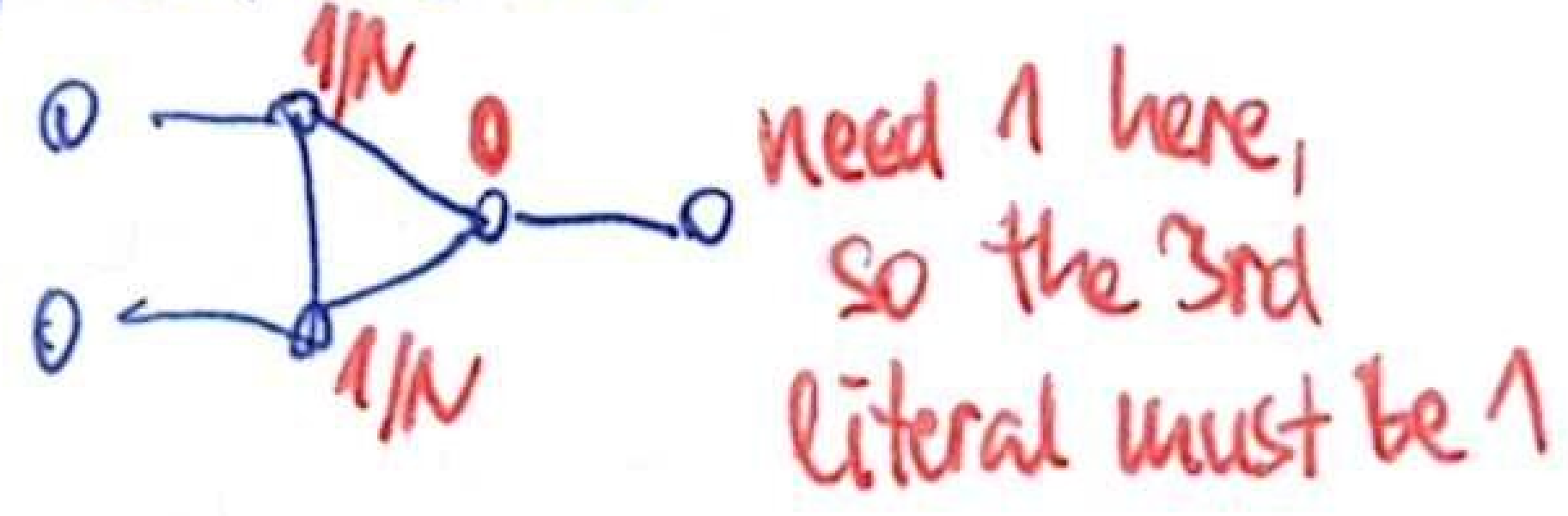


= coloring of B:



= coloring of A:

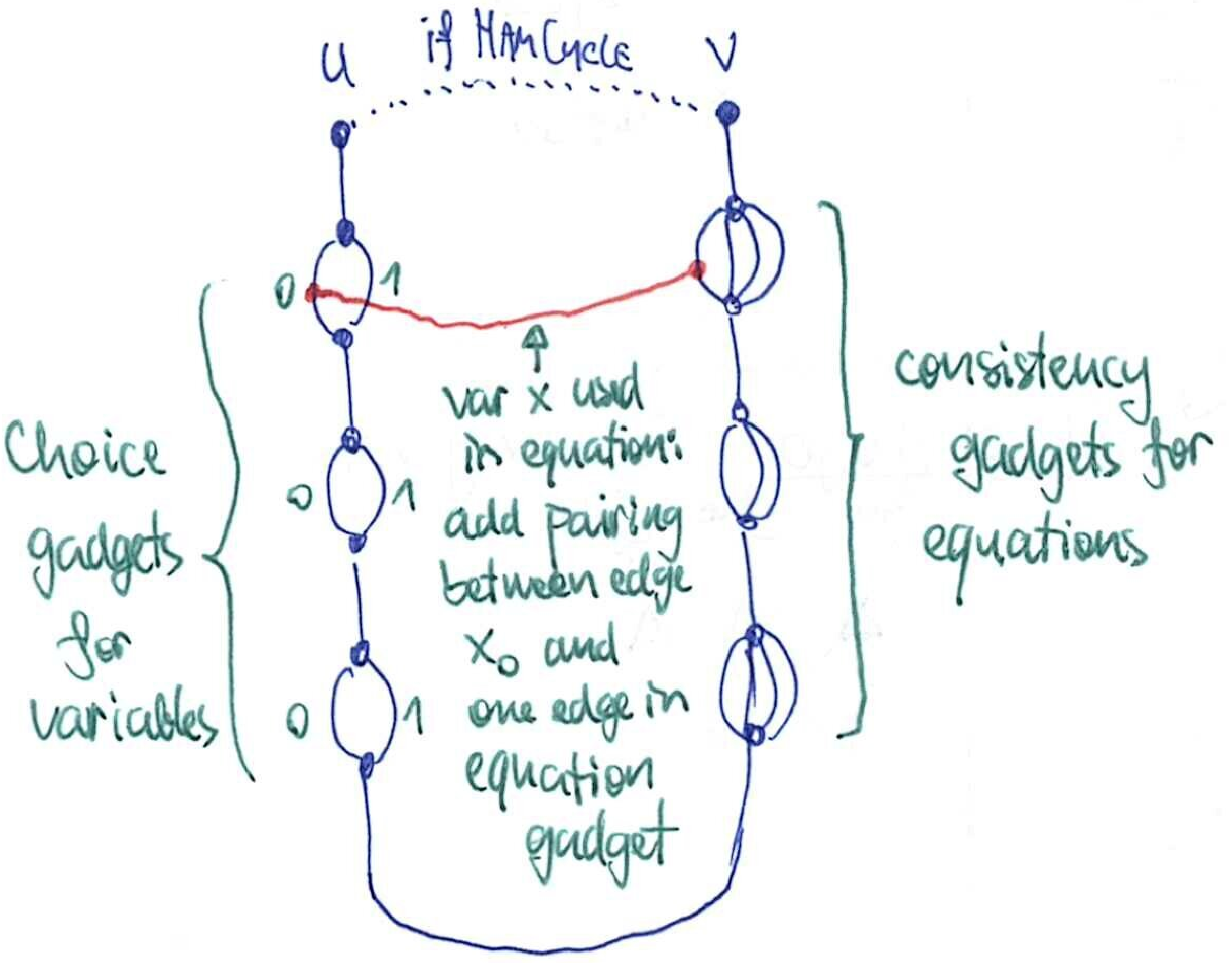
- if inputs contain 1: use coloring of B which passes 0
- if both are 0:

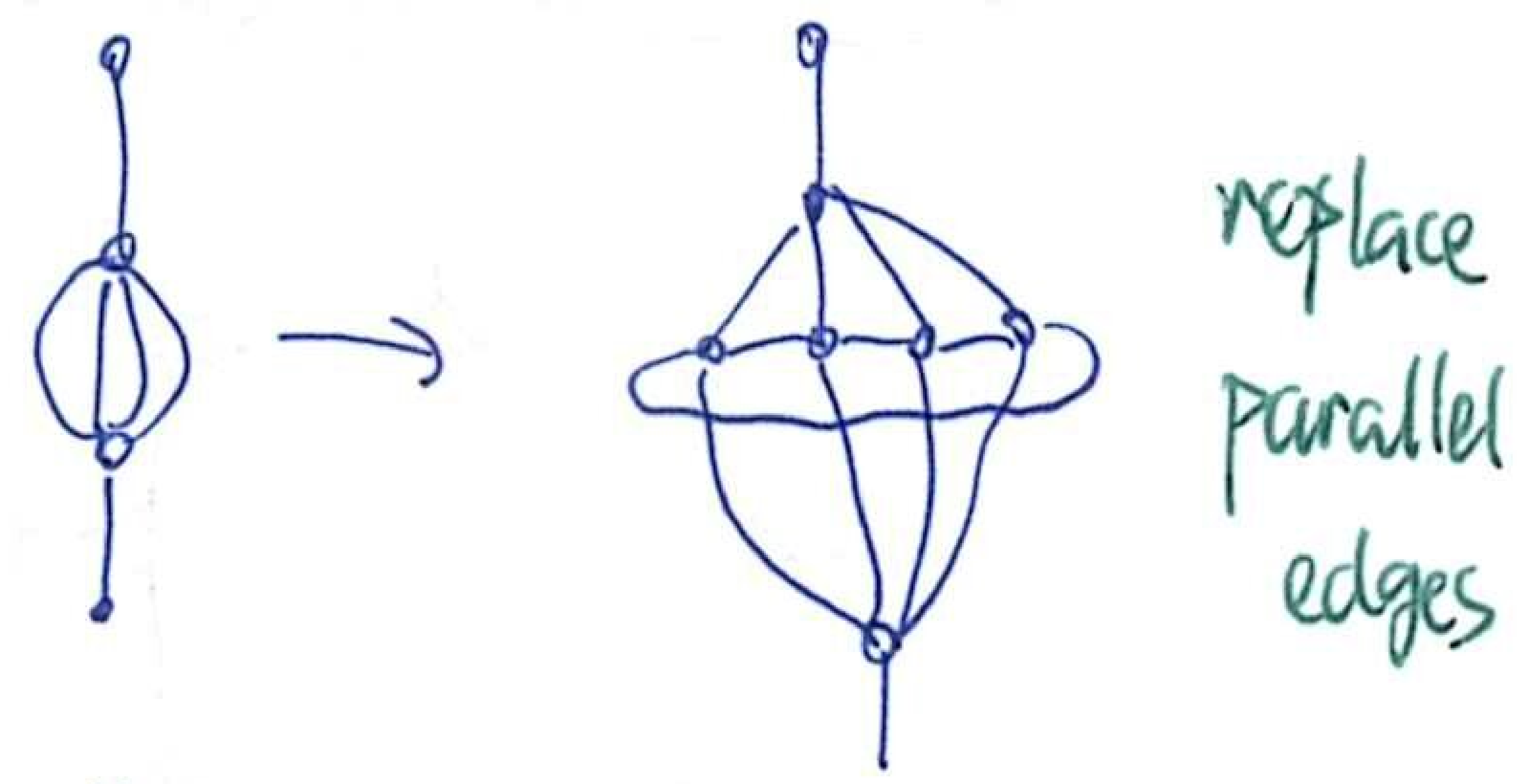


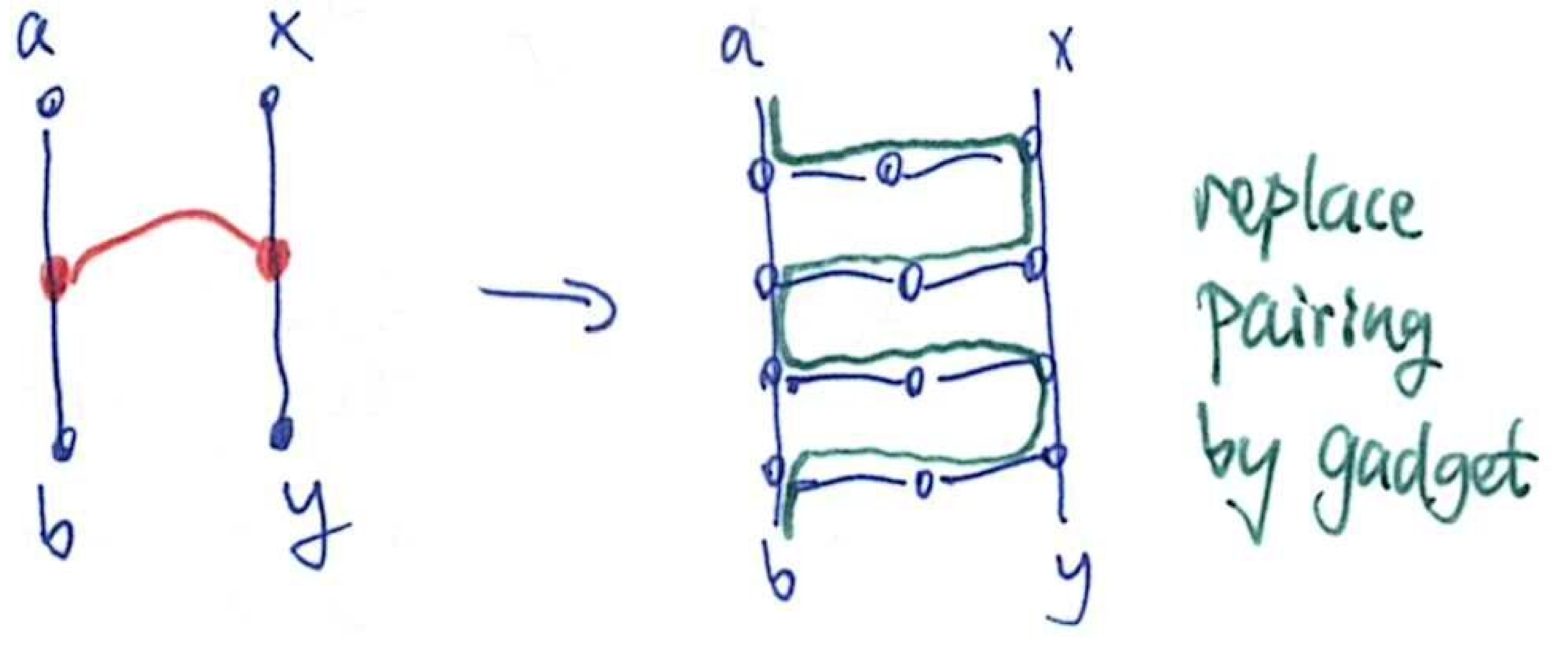
Reduction: ZOE $\le^P_m$ HAMILTON ~~Cycle~~ or PATH

First consider problem HAMPATH* which allows:
- parallel edges
- pairing: e ~ f ≡ must use exactly 1 edge of e, f

reduce to plain HAMILTON PATH







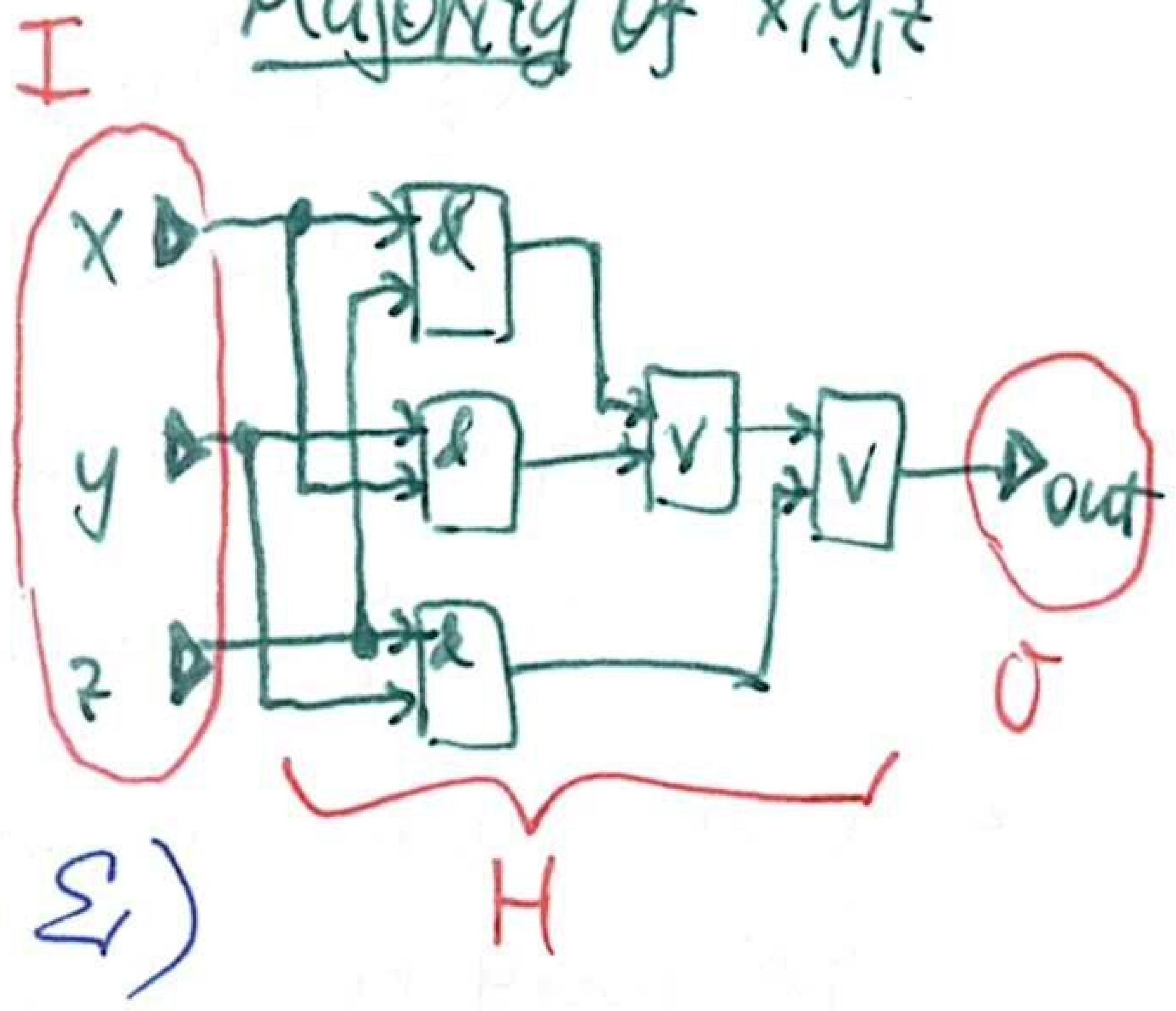
Exercises:
- SUBSET SUM problem is NP-complete: Given a finite set $X \subseteq \mathbb{N}$, $s \in \mathbb{N}$, is there $X' \subseteq X$ s.t. $\sum_{a \in X'} a = s$? (Hint: reduce from ZOE)
- 2 BANDITS: given finite $X \subseteq \mathbb{N}$, is there $X' \subseteq X$ s.t. $\sum X' = \sum (X \setminus X')$? Also NP-complete.

Brief detour: From formulas to Boolean circuits...

Df: A Combinatorial Circuit consists of:

- a finite alphabet Σ,
- finite sets I (input terminals), $= \{i_1, \dots, i_{|I|}\}$
  O (output terminals) $= \{o_1, \dots, o_{|O|}\}$
  H (gates) $= \{h_1, \dots, h_{|H|}\}$ } pairwise disjoint
- directed acyclic multigraph $(I \cup O \cup H, E)$
- arity: $a : H \to \mathbb{N}$
- assignment of functions to gates $F : h \mapsto (f_h : \Sigma^{a(h)} \to \Sigma)$
- assignment of gate inputs to incoming edges: $\mathcal{Z} : (u,v) \in E \mapsto i \in \{1 \_ a(v)\}$

Example:
Majority of x,y,z



Where:

- $\forall i \in I \; \deg^{in}(i) = 0$
- $\forall o \in O \; \deg^{in}(o) = 1, \; \deg^{out}(o) = 0$
- $\forall h \in H \; \deg^{in}(h) = a(h) \;\&\; \forall i \in \{1 \_ a(h)\} \; \exists! (x,h) \in E : \mathcal{Z}((x,h)) = i$

Df: Boolean Circuit: Comb. circuit with $\Sigma = \{0,1\}$

Df: Computation of a circuit proceeds in steps.

↑ Sketch of

Step 0: input terminals and arity-0 gates (constants) have defined values.

Step i+1: gates & output terminals whose input is defined in step at most i produce output.

As the graph is acyclic, gate outputs never change and every gate/terminal is defined within finite # steps.

⇒ the circuit computes a function from $\Sigma^{|I|}$ to $\Sigma^{|O|}$.

Bounding arity: Since a single gate of high arity can compute anything in 1 step, we will bound arity by 2. (Actually, any fixed constant > 1 would work.)

Circuit complexity: Time ≈ # layers (# steps of computation)
Space ≈ # gates
} BTW circuits are an interesting model of parallel computing

Boolean formulas ≈ circuits with tree structure (except for inputs)

Lemma: Every function $f : \{0,1\}^k \to \{0,1\}^l$ can be computed by a Boolean circuit consisting only of AND, OR and NOT gates and constants. (OR can be replaced by $\overline{\bar{x} \,\&\, \bar{y}}$) — in fact it's a formula in DNF

Proof: ① n-input AND/OR can be computed by a tree of 2-input ANDs/ORs.

② Function with multiple-bit output: replace by l single-bit functions.

③ Function whose truth table contains exactly one 1:
e.g. $\neg x_1 \,\&\, \neg x_2 \,\&\, \neg x_3 \,\&\, x_4$ ... 1 at position 0001

④ Truth table with multiple 1's: OR functions for each 1.

⑤ Otherwise it's constant 0.

} produces circuits of exponential size, but good enough for k, l constant

Corollaries: ① can simulate arbitrary gates of fixed arity with $O(1)$ space/time overhead.
② can simulate arbitrary comb. circuit by a Boolean circuit (binary-encoded Σ)

Problem: A circuit handles inputs of constant size only.
↳ a "program" is a family of circuits $C_0, C_1, \ldots$
where C_n solves the problem for inputs of size n.

But: If we allow arbitrary sequences, we can compute undecidable problems:

$$L = \{\alpha \mid \underbrace{|\alpha| \text{ written in binary}}_{\text{with leading 1 removed}} \in L_u\}$$

arbitrary enumeration of binary strings

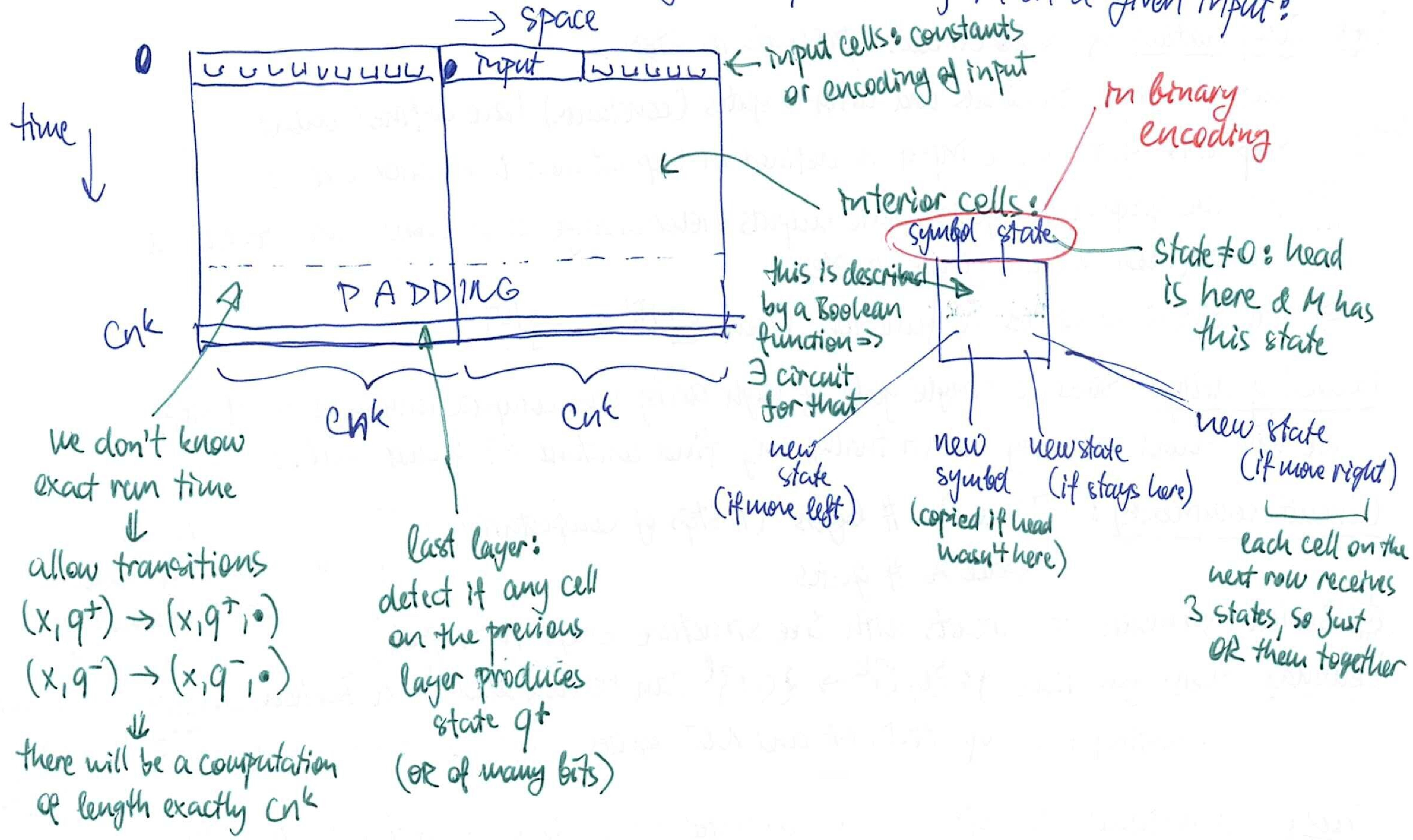
- So we usually require the family to be uniform: there is an algorithm which for every n produces C_n in time poly(n).

$\boxed{\text{So languages decidable by uniform circuit families} = P}$

Theorem: For every $L \in P$ there is $f \in PF$ s.t. for every n, $f(n)$ is an (encoding of) Boolean circuit with n inputs and 1 output which decides L for strings of length n.

↑ with obvious meaning (computes char. function of L)

Proof: Let M be a 1-tape TM deciding L in time at most $c \cdot n^k$ for some $c, k \in \mathbb{N}$.
(WLOG)
We will build a circuit producing a computation of M on a given input:



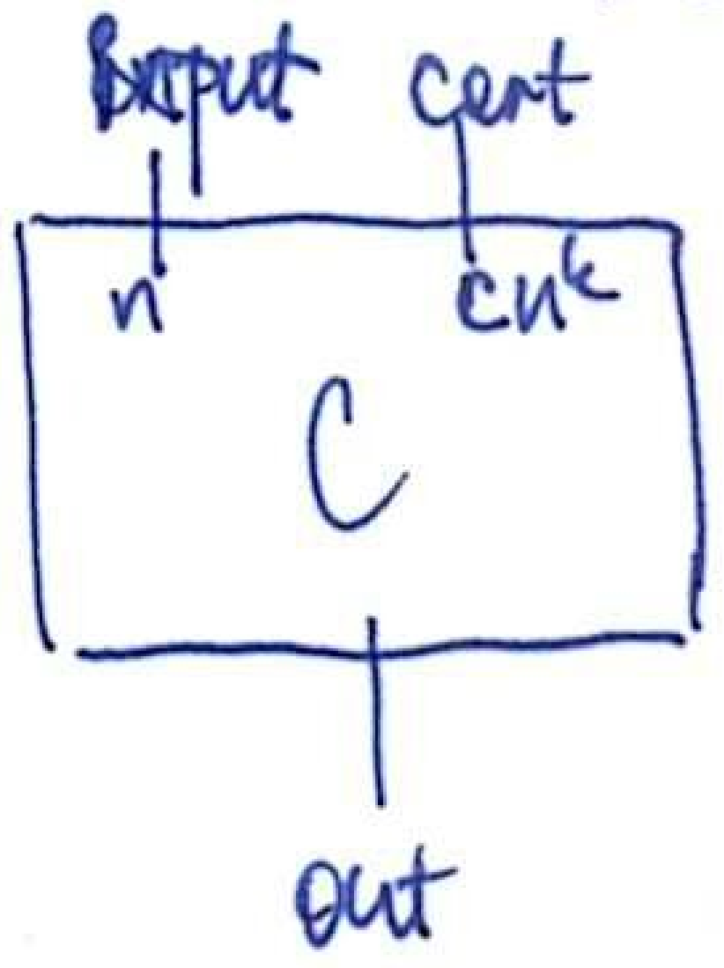
Df: CIRCUIT-SAT: given a Boolean circuit with 1 output, is there an input for which the output is true?

↳ Obviously, this is in NP.

Thm: CIRCUIT-SAT is NP-complete.

Proof: When reducing from $L \in NP$ to C.-SAT: consider verifier $V \in P$ & upper bound cn^k for certificate size.

- Adapt verifier to accept certificates of size exactly cn^k (using reversible padding like 10^*)
- Find Boolean circuit for V on inputs of size $n + cn^k$:



& fix input terminals to input α

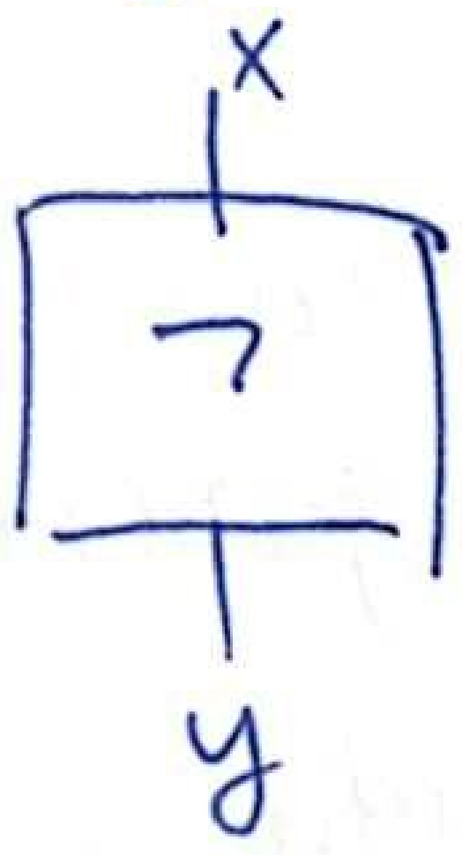
↓

SAT for $C_\alpha(\text{cert})$ computes $\alpha \in L$

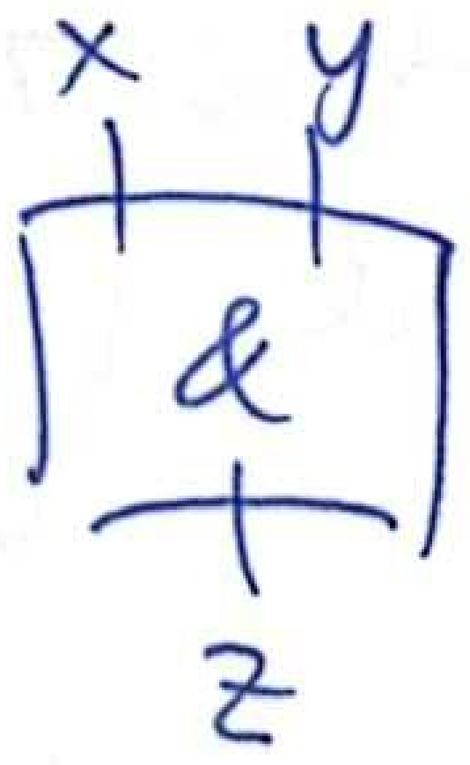
done inside the reduction when receiving input α of size n

Lemma: CIRCUIT-SAT $\leq_m^P$ SAT.

Proof: Assume WLOG that all gates are AND and NOT.
Introduce new variables for gate outputs.
Add consistency-checking clauses:



$x \Rightarrow \neg y$
$\neg x \Rightarrow y$



$x \& y \Rightarrow z$
$\neg x \Rightarrow \neg z$
$\neg y \Rightarrow \neg z$

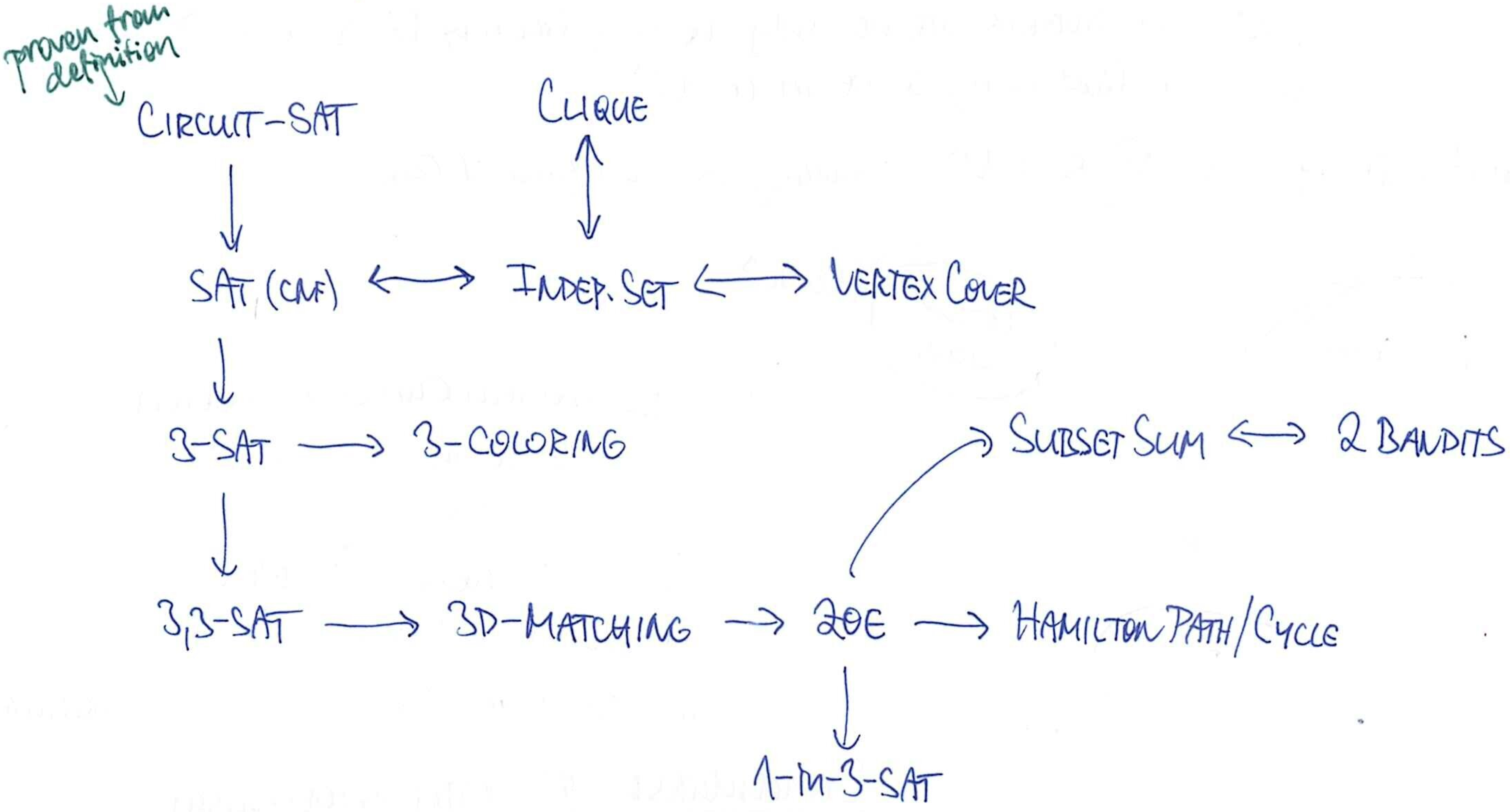
this is $\neg x \vee \neg y \vee z$ in CNF

this also means that SAT for general (non-CNF) formulas is reducible to SAT

BTW we produced an instance of 3-SAT :)

Corollary: SAT is NP-complete. [This is Cook-Levin theorem!]

Map of NP-complete problems we encountered until now:



Further SAT variants: 2-SAT is in P

E3,E3-SAT (clauses of size exactly 3, vars have exactly 3 occurrences)
surprise: all instances satisfiable!

The class co-NP

Df: For a language $L \subseteq \{0,1\}^*$ we define it's complement $\overline{L} := \{0,1\}^* \setminus L$

Df: For a class $\mathcal{C}$ of languages: $\text{co-}\mathcal{C} := \{\overline{L} \mid L \in \mathcal{C}\}$ → co-P = P

$A = B \Leftrightarrow \text{co-}A = \text{co-}B$
$A \subseteq B \Leftrightarrow \text{co-}A \subseteq \text{co-}B$
$\text{co-co-}A = A$

Let's study co-NP...

- $P \subseteq NP \cap \text{co-}NP$ ... open if the inclusion is strict
- if $P = NP$, then $NP = \text{co-}NP$ } contrapositive
- if $NP \neq \text{co-}NP$, then $P \neq NP$ (~~because P = co-P~~)
- as $K \leq^P_m L \Leftrightarrow \overline{K} \leq^P_m \overline{L}$, we have: L is NP-complete ⇔ $\overline{L}$ is co-NP-complete
- certificate-based def.: $L \in \text{co-}NP \equiv \exists V \in P : (x \in L \Leftrightarrow \forall \beta \in \{0,1\}, |\beta| \leq poly(|x|)\ V(x,\beta))$

↳ so $\overline{SAT}$ is co-NP-complete

↑ this is not UNSAT (unsatisfiability for CNF), because for strings which do not encode a formula, we still have to answer 1 in $\overline{SAT}$ but 0 in UNSAT

↳ but $\overline{SAT} \lesseqgtr^P_m$ UNSAT, so UNSAT is co-NP-comp.

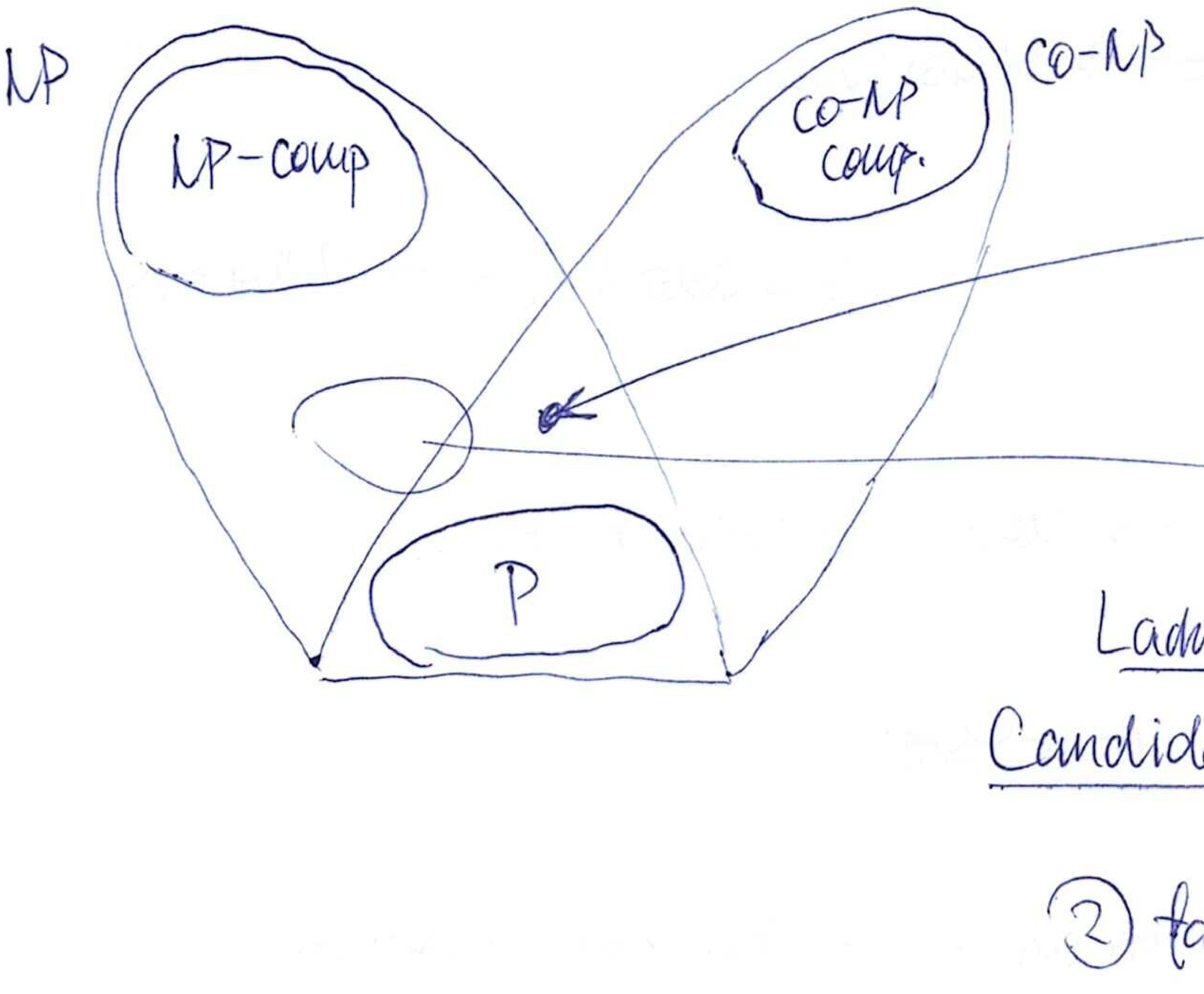
↳ $\neg \exists \vec{x}\, \varphi(\vec{x}) \Leftrightarrow \forall \vec{x}\, \neg\varphi(x)$ → so $\neg\varphi$ is a tautology & if φ is in CNF, $\neg\varphi$ can be written in DNF by propagating negation

So: TAUTOLOGY $:= \{\alpha \mid \alpha$ is (encoding of) DNF formula which is tautological$\}$ is also co-NP-complete (this is the most standard co-NP-c. problem)

Formally: TAUTOLOGY $\lesseqgtr^P_m$ UNSAT $\lesseqgtr^P_m$ $\overline{SAT}$

Exercise: If $L \in \text{co-}NP$ is NP-complete, then $NP = \text{co-}NP$.
(so NP-comp. problems are not only the least likely of NP to be in P, but also least likely to be in co-NP).

Landscape of P vs. NP vs. co-NP (assuming the most general case)



no problem currently known to be in $(NP \cap \text{co-}NP) \setminus P$
(former candidate: primality testing, but it's known to be in P now)

NP-intermediate problems in $NP \setminus (NP\text{-comp.} \cup P)$

Ladner's thm: if $P \neq NP$, $\exists L$ NP-intermediate.

Candidates: ① graph isomorphism (known to be in $DTIME(n^{\log n})$)
② factoring (given x, a, b: does x have a factor in $[a,b]$?)

Non-deterministic TM (NTM)

extend $\delta : Q \times \Gamma^k \to \mathcal{P}(Q \times \Gamma^k \times \{\leftarrow, \rightarrow, \circ\}^k)$ … choice of instruction from the set

- successor relation is not a function → multiple computations for a given output
- if $\delta(q,x) = \emptyset$, we assume rejection.
- input α accepted ≡ ∃ at least one accepting computation
- halting ≡ all computations halt
- time & space: maximum over all computations

👁 WLOG $|\delta(q,x)| \leq 2$

👁 enumeration works again (NM_α = NTM with code α), we have an universal NTM &c.

Exercise: k-tape → 2-tape NTM with only constant-factor slowdown

Df: Non-deterministic complexity classes: NTIME(f), NTIMEF(f) for functions,

└ all accepting computations must agree on result, ∃ at least one accepting comp.

Theorem: $NP = NTIME(poly(n)) := \bigcup_{k \geq 0} NTIME(n^k)$

Proof: ⊆ the NTM guesses the certificate using non-determinism & then it runs the verifier

⊇ the certificate encodes non-deterministic choices

Example: The following problem is NP-complete:

$\{ \langle \alpha, \beta, 1^t \rangle \mid NM_\alpha \text{ accepts input } \beta \text{ within } t \text{ steps} \}$

↗ ∈NP: simulate $NM_\alpha(\beta)$ using universal NTM with an "alarm clock" (reject after t steps)

↓ reduction: calculate $t \in poly(n)$, α := code of NTM solving source problem, pass β

Space Complexity

We want to count only "work space" of the TM.

3 types of tapes:

- input tape: read-only, head doesn't move more than 1 cell before/after input string
- k work tapes: read-write
- output tape: write-only, head cannot move left

} space used by computation := # visited cells on the work tapes (for NTM: max over computations)

👁 This doesn't change time complexity classes: we can copy input → work → output with constant slowdown

👁 We can encode information in position of head on input tape.

- this makes a difference if work space ∈ $o(\log n)$
- otherwise we can keep track of the head position in binary

Space classes: (defined using machines which always halt)

DSPACE(f)
NSPACE(f) } decision problems

& DSPACEF(f)
NSPACEF(f) } functions

PSPACE = DSPACE(poly(n))
NPSPACE = NSPACE(poly(n))

We want f to be:

① non-decreasing
② space-constructible
 ⤷ f(n) can be computed from 1^n, result in binary, in space $O(f(n))$
③ usually $f(n) \geq \log n$

} proper space-complexity function

Basic inclusions: DTIME$(f) \subseteq$ NTIME$(f) \underset{\text{can try all certificates in space } O(f)}{\subseteq}$ DSPACE$(f) \subseteq$ NSPACE(f)

So: P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ NPSPACE

Thm: DSPACE$(f) \subseteq$ DTIME$(2^{O(f)})$ for every $f \geq \log n$.

Proof: First, let's bound # reachable configurations:

$$\underbrace{|Q|}_{\text{state}} \cdot \underbrace{(n+2)}_{\text{pos. of head on input tape}} \cdot \underbrace{(|\Gamma|+1)^{f(n)}}_{\text{contents of work tapes, extra character for "end of tape"}} \cdot \underbrace{f(n)^{\overset{\text{\# tapes}}{k}}}_{\text{pos. of heads on work tapes}}$$

... this is $O(2^{O(f(n))})$ → so if it halts, it must do so within $2^{O(f(n))}$ steps

If a configuration repeats, the whole computation loops. (⚠ this requires deterministic TM!) ⊛

⇒ add a binary counter of $O(f(n))$ bits, use it as alarm clock (increment in every step of the original TM). Alarm expires ⇒ reject.

} in space-bounded computation with space $\geq \log n$, we can always make sure than the machine halts.

Corollary: ~~N~~PSPACE $\subseteq$ EXPTIME := DTIME$(2^{\text{poly}(n)})$] this is called EXPTIME or EXP

We want to prove the same for NSPACE(f), but ⊛ makes it more complicated.

Reachability method

Df: Configuration graph of a given NTM (within space $f(n)$) on a given input is a directed graph with:

- V := set of configurations (for input tape, consider only head position) ← limited by available space
- E := successor relation (depends on x)
- start $\in V$ ... initial config
- accept $\in V$ ... modify the TM to clean up before accepting < clear working tapes, rewind input tape } unique accepting config

👁 $|V| \in O(2^{O(f)})$, $|E| \in O(|V|)$, graph can be generated in $O(\text{poly}(|V|))$ time & $O(f)$ space.

👁 Machine accepts ⟺ graph contains a (directed) path from start to accept.

Thm: NSPACE$(f) \subseteq$ DTIME$(2^{O(f)})$ for every $f \geq \log n$. [therefore NPSPACE $\subseteq$ EXPTIME]

Proof: Construct the reachability graph & run BFS on it.

↑ time $O(\text{poly}(|V|, |E|))$ ← which is $O(2^{O(f)})$

↑ also time $O(\text{poly}(|V|, |E|))$

Generally: Time-/space-efficient algorithms for REACH translate to inclusions of complexity classes.

↑ $\{\langle G, s, t\rangle \mid \exists$ path from s to t in $G\}$

Thm (Savitch's): NSPACE$(f) \subseteq$ DSPACE(f^2) for every $f \geq \log n$.

Corollary: NPSPACE $\subseteq$ PSPACE, so NPSPACE = PSPACE.

↳ will be proven soon...

31

Lemma: REACH $\in O(\log^2 n)$

Proof: Use "middle-first search".

Recursive function $D_k(x,y)$ computing "$\exists$ walk from x to y with at most 2^k edges".

We have: $D_0(x,y) = (x=y) \vee ((x,y)\in E)$

$D_k(x,y) = \bigvee_{z\in V}(D_{k-1}(x,z) \,\&\, D_{k-1}(z,y))$ … FOR loop & recursion

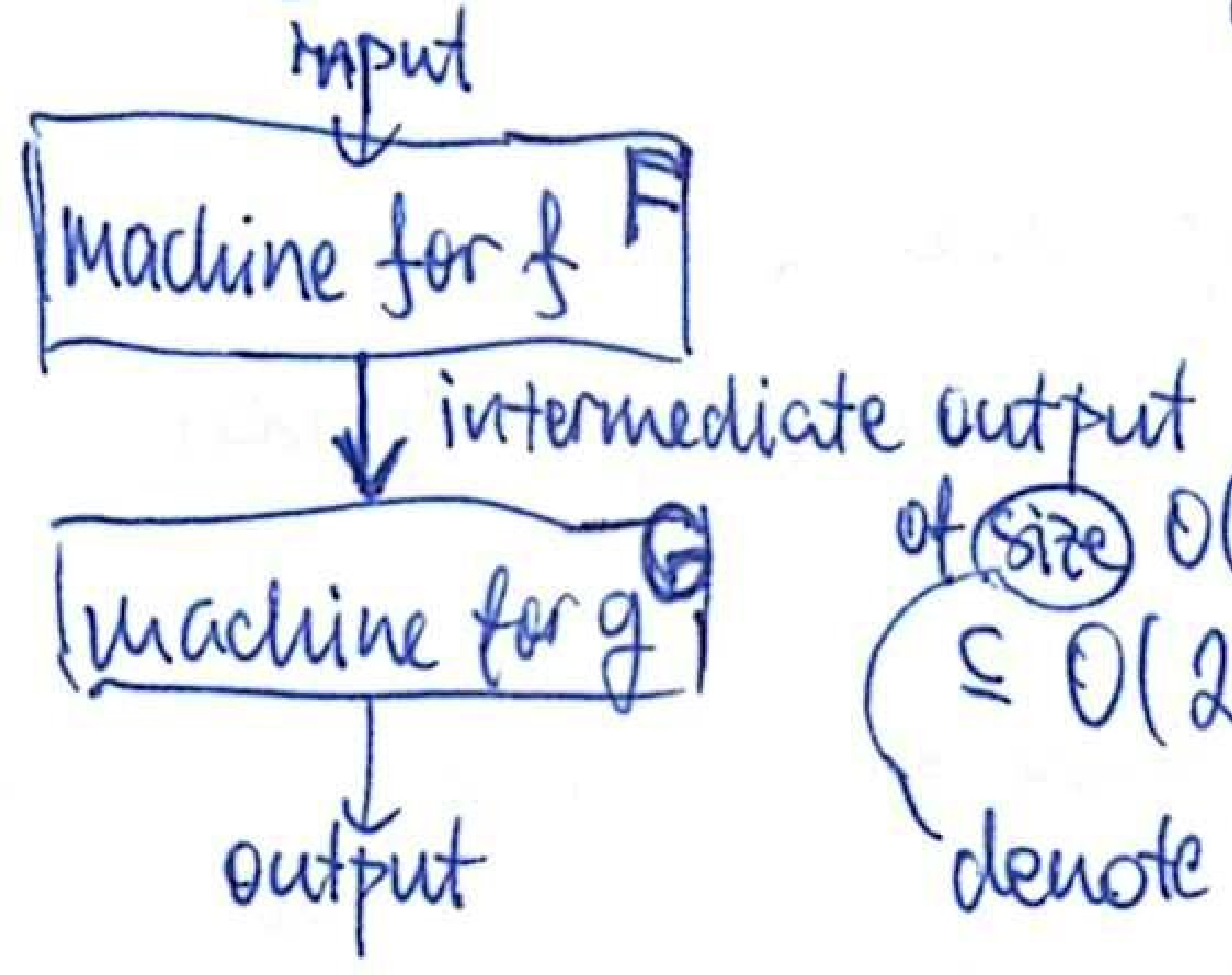
Every level of recursion requires $O(\log n)$ space for local variables, $\overset{\log n}{\cancel{\#}}$ levels suffice to find a path.

Now, we want to combine generator of config graph with this algorithm, but we don't have space to store the graph.

Lemma: If f can be computed in space $s(n)$
and g —— " ——— $t(m)$, } $s,t \geq \log$
Then $g(f(\dots))$ can be computed in space $O(s(n) + \overset{t(2^{O(s(n))})}{\cancel{t(m)}})$

} also applies to composition of a language with a function (that is, a reduction)

Proof:



of size $O(\text{time}(F)) \subseteq O(2^{O(s(n))})$
denote by m

Start G & keep track of position of head on G's input tape. $O(\log m)$ space

Whenever G moves its input head (& at the start of computation), re-run F to get the corresponding symbol of its output.

↳ modify F: reset work tapes on startup
reset input head — " —
keep track of output head position] $O(\log m)$ space
write to output tape: compare with G's input head pos
= remember char in state
≠ discard written char (won't be read by F)

Total space:

$s(n)$ for F

$O(\log m)$ for head positions … this is $O(\overset{s(n)}{\cancel{t(m)}})$

$t(m)$ for G

Corollary: Savitch's thm.

essentially, we combined a reachability alg. with an oracle for edges

↳ If $L \in$ NSPACE(f): graph generation requires $O(f)$ space,
reachability needs $O(\cancel{\log}(2^{O(f)})^2) = O(\overset{f^2}{\cancel{2^{O(f)}}})$] combined by the lemma to $O(\underset{f^2}{\cancel{2^{O(f)}}})$

Corollary: DTIMEF$(\log n)$ is closed under composition of functions.

Remark: REACH $\in O(\log n)$ would imply NSPACE(f) = DSPACE(f) … but this is long open.

It's known that undirected UREACH $\in O(\log n)$ [Reingold 2004, non-trivial]

↳ this implies only SSPACE(f) = DSPACE(f)
↑ symmetric non-determinism (successor relation symmetric)

So we have: DTIME$(f) \subseteq$ NTIME$(f) \subseteq$ DSPACE$(f) \subseteq$ NSPACE$(f) \subseteq$ DTIME$(2^{O(f)})$
└ $\subseteq$ DSPACE(f^2)

and: NSPACE$(\log n) \subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE = NPSPACE $\subseteq$ EXPTIME

↑ this is also known as NL

↑ also = co-NPSPACE as PSPACE is closed under complements

More about PSPACE

Thm: QBF is PSPACE-complete. ← with respect to $\le_m^P$ (QBF is sometimes called QSAT)

↑ the language of all true <u>quantified Boolean formulas</u> (all variables bound by quantifiers)

Proof: ① QBF ∈ PSPACE by the following recursive algorithm:

- $QBF(\forall x\, \varphi(x)) = QBF(\varphi(0))\ \&\ QBF(\varphi(1))$
- $QBF(\exists x\, \varphi(x)) = QBF(\varphi(0)) \lor QBF(\varphi(1))$
- $QBF(\varphi \lor \psi) = QBF(\varphi) \lor QBF(\psi)$
- $QBF(\varphi\ \&\ \psi) = QBF(\varphi)\ \&\ QBF(\psi)$
- $QBF(\neg\varphi) = \neg QBF(\varphi)$

} $O(n)$ levels of recursion, $O(n)$ space per level. } $O(n^2)$ space

② QBF is PSPACE-hard: consider $L \in$ PSPACE, TM M deciding L and its config. graph G.

- vectors of variables $\bar{x}$ encoding vertices … $O(poly(n))$ bits [log of $|G|$]
- formula $\varphi(\bar{x},\bar{y}) \equiv (\bar{x}=\bar{y}) \lor (x,y) \in E(G)$
  - can construct poly-sized circuit as in proof of Cook-Levin thm. & then reduce the circuit to an existentially-quantified formula as in Circuit-SAT $\le_m^P$ SAT.
- mimic proof of Savitch's thm: $\varphi_k(\bar{x},\bar{y}) \equiv \bar{y}$ is reachable from $\bar{x}$ within 2^k steps ($\wedge$ max.)

Failed attempt: $\varphi_k(\bar{x},\bar{y}) \equiv \exists \bar{z} \left(\varphi_{k-1}(\bar{x},\bar{z})\ \&\ \varphi_{k-1}(\bar{z},\bar{y})\right)$

Double recursion ⇒ formula size grows exponentially!

Better: $\varphi_k(\bar{x},\bar{y}) \equiv \exists \bar{z}\, \forall \bar{a}\, \forall \bar{b} \left(\left((\bar{a}=\bar{x}\ \&\ \bar{b}=\bar{z}) \lor (\bar{a}=\bar{z}\ \&\ \bar{b}=\bar{y})\right) \Rightarrow \varphi_{k-1}(\bar{a},\bar{b})\right)$

- G has size $O(2^{poly(n)})$ ⇒ log (path len) ∈ poly(n) ⇒ recursion has poly(n) levels, formula size grows to poly(n).

Intuition: PSPACE is the class of strategies for 2-player games with perfect information:

(∃ player 1 opening move) (∀ player 2's ~~resp~~ response) (∃ player 1's counter-response) (∀ …) …

Examples:

- graph coloring game: undirected graph, finite set of k colors each player colors an uncolored vertex, ~~every~~ no edge must have both ends of the same color
- graph path game: building path edge by edge, ~~first~~ ~~each~~ pl~~ayer has his target~~, ~~2nd player~~ vertices must not repeat
- variants of Go, Checkers &c. (generalized to N×N boards)
- Sokoban (1-player, but enough internal state, which restricts future moves, but can be modified)

these are known to be PSPACE-complete

Alternating Turing Machine (ATM)

3 kinds of states:

- deterministic: config is accepting ⇔ next config is accepting
- existential: config is accepting ~~accepts~~ ⇔ ∃ non-det. choice ~~leading~~ ~~s.t. the rest~~ leading to accepting config.
- universal: is accepting ⇔ ∀ non.-det. choice leads to accepting config.

We will require all computations to halt.

ATM → classes ATIME(f), AP ... [we won't define space-bounded classes as we require all computations to halt & alarm clocks don't help]

<u>Theorem:</u> AP = PSPACE.

<u>Proof:</u> $\supseteq$ is easy: QBF $\in$ AP since we can execute quantifiers using corresponding state types.
So if $L \le_m^P$ QBF, we can first compute the reduction and then solve QBF.

$\subseteq$: simulate the ATM recursively as in QBF $\in$ PSPACE.

- configurations take $O(\text{poly}(n))$ space – all computations are poly-time, so they are poly-space, too.
- recursion depth is bounded by time of the ATM.

Polynomial Hierarchy

Consider following restrictions of QBF:

- Σ_k-formulas: $\underbrace{(\exists x_1 \dots \exists x_k)(\forall \dots)(\exists \dots) \dots}_{k \text{ groups of quantifiers, starting with } \exists} \underbrace{\varphi(-)}_{\text{quantifier-free formula}}$
  (with no free variables)
- Π_k-formulas: similar, but starting with $(\forall \dots)$

negation of a Σ_k-formula can be written as a Π_k-formula & vice versa

- Σ_k-SAT $:= \{ \langle\varphi\rangle \mid \varphi$ is a true Σ_k-formula$\}$ ... similarly Π_k-SAT.
  $\langle\varphi\rangle$ ↑ encoding of
- Σ_1-SAT is SAT (for general formulas, not only CNF), Π_1-SAT is TAUT.

We can use this to define new classes which generalize NP:

- $\Sigma_k^P := \{ L \mid L \le_m^P \Sigma_k\text{-SAT} \}$ ... $\Sigma_1^P = NP$, $\Sigma_0^P = P$
- $\Pi_k^P := \{ L \mid L \le_m^P \Pi_k\text{-SAT} \}$ ... $\Pi_1^P = \text{co-}NP$, $\Pi_0^P = P$, $\Pi_k^P = \text{co-}\Sigma_k^P$

we defined classes using a problem complete for them

- generally, the following inclusions hold:

$$\begin{tikzcd}
 & \Sigma_1^P = NP \arrow[r] \arrow[rdd] & \Sigma_2^P \arrow[r] \arrow[rdd] & \Sigma_3^P \arrow[r] \arrow[rdd] & \dots \\
\Sigma_0^P = \Pi_0^P = P \arrow[ru] \arrow[rd] & & & & \\
 & \Pi_1^P = \text{co-}NP \arrow[r] \arrow[ruu] & \Pi_2^P \arrow[r] \arrow[ruu] & \Pi_3^P \arrow[r] \arrow[ruu] & \dots
\end{tikzcd}$$

This is akin to the arithmetical hierarchy, but the inclusions are not known to be strict.

- $PH := \bigcup_k \Sigma_k^P = \bigcup_k \Pi_k^P$
- since every Σ_k / Π_k-SAT reduces trivially to QBF, we have $PH \subseteq PSPACE$ (not known to be strict)

<u>Example:</u> "given Bool. formula φ, find a shortest ψ s.t. $\forall \bar{x}\ \varphi(\bar{x}) \Leftrightarrow \psi(\bar{x})$" $\in \Sigma_2^P$
(in fact, it's Σ_2^P-complete)

<u>Remark:</u> Definition using oracle machines also works:

$$\Sigma_{k+1}^P := NP[\Sigma_k^P] = NP[\Pi_k^P]$$

$$\Pi_{k+1}^P := \text{co-}NP[\Sigma_k^P] = \text{co-}NP[\Pi_k^P]$$

we can add $\Delta_{k+1}^P := P[\Sigma_k^P] = P[\Pi_k^P]$ ...

$$\begin{tikzcd}
\Sigma_k \arrow[rd] \arrow[rr, bend left] & & \Sigma_{k+1} \\
 & \Delta_{k+1} \arrow[ru] \arrow[rd] & \\
\Pi_k \arrow[ru] \arrow[rr, bend right] & & \Pi_{k+1}
\end{tikzcd}$$

this leads to the same hierarchy

Remark: We can also define Σ_k^P & Π_k^P using alternative TMs:

- Σ_k-TIME$(f) := \{ L \mid$ L can be decided by an ATM running in time $\in O(f(|input|))$ which performs at most $k-1$ quantifier changes, starting with $\exists \}$
- $\Sigma_k^P = \Sigma_k$-TIME$(poly(n))$

Collapse of PH

- $P = NP \Leftrightarrow P = PH$
- $NP = co\text{-}NP \Leftrightarrow NP = PH$
- if $\Sigma_j^P = \Sigma_{j+1}^P$, then $\Sigma_j^P = \Sigma_k^P$ for all $k > j$] we say that PH collapsed to the j-th level
  ... so $PH = \Sigma_j^P$
- if $\Sigma_j^P = \Pi_j^P$, then $\Sigma_{j+1}^P = \Sigma_j^P$ (we can reduce $\exists \bar{x} \forall \bar{y}\, \varphi(\ldots)$ to $\exists \bar{x} \exists \bar{y}\, \varphi(\ldots)$)

this is weaker than $P = NP$, but still open

Note: If graph isomorphism is in P, then $PH = \Sigma_2^P$. [proof non-trivial]

SPACE CO-CLASSES

Unlike non-deterministic time classes, non-det. space classes are known to be closed under complement.

Theorem (Immerman–Szelepcsényi): $NSPACE(s(n)) = co\text{-}NSPACE(s(n))$ for all space-constructible functions $s(n) \geq \log n$.

Proof: We design a non-deterministic algorithm for non-reachability in config. graphs. More generally, we'll calculate $R_i :=$ #vertices reachable from source by walk of len $\leq i$.

$V_i :=$ set of these vertices

Then modify graph by adding edges from target to all vertices, so (tgt reachable from src) $\Leftrightarrow R_n = n$ for $n =$ #vertices.

```
• R_0 = 1
• R_{i-1} -> R_i: For all v ∈ V:
                      For all w ∈ V_{i-1}:
                          if (w,v) ∈ E or v=w: R_i <- R_i + 1
• Enumeration of V_i:
      t <- 0
      For all u ∈ V:
          If guess that u ∈ V_i:
              If ∄ walk src -> u of length ≤ i: REJECT
              t <- t+1
      If t ≠ R_{i-1}: REJECT
```

if we don't guess correctly, either the path doesn't exist or $t < R_{i-1}$ at the end

guess the path using non-determinism & check that it's valid

Space needed:
- $O(1)$ variables for vertices & counters
- R_i and R_{i-1}

} $O(\log |V|)$ space, where $|V| = 2^{O(s(n))}$, so this is in $NSPACE(s(n))$.

INSIDE P

👁 this is transitive (composition of space-bound functions)

We need to use log-space reductions $\leq_m^{log}$ (when using $\leq_m^P$, all problems in P except $\emptyset$ and $\{0,1\}^*$ are equivalent)

Important classes: $L := DSPACE(\log n)$, $NL := NSPACE(\log n)$

We know: $\underset{\substack{\uparrow \\ \text{trivial}}}{L \subseteq} \; \underset{\substack{\uparrow \\ \text{Imm.-Sz.}}}{NL = } \; co\text{-}NL \underset{\substack{\uparrow \\ \text{reachability } \& \ 2^{c \log n} = n^c}}{\subseteq} P$] inclusions not known to be strict

Theorem: CIRCUIT-EVAL is P-complete wrt. $\leq_m^{log}$.
given Boolean circuit & input, is the output true?

Proof: Verify that the circuit construction we used when proving Cook's thm can be carried out in log. space.
CIRCUIT-EVAL $\in$ P trivial: evaluate gates in topological order on the graph.

open: CIRCUIT-EVAL $\in$ NL would imply NL = P

Theorem: REACH is NL-complete wrt. $\leq_m^{log}$.

Proof: Configuration graph of a NTM can be constructed in log space.
REACH $\in$ NL trivial: guess the path using non-determinism, use binary counter to limit its length.

open: REACH $\in$ L would imply L = NL

2-SAT (CNF formulas, all clauses have ≤ 2 literals)
exactly 2 if we replace (x) by $(x \vee x)$

👁 $(\alpha \vee \beta)$ is an implication $\neg\alpha \Rightarrow \beta$, which is also $\neg\beta \Rightarrow \alpha$

For a 2-CNF formula φ, construct its implication graph: vertices = variables & their negations
edges = implications (clause produces two)

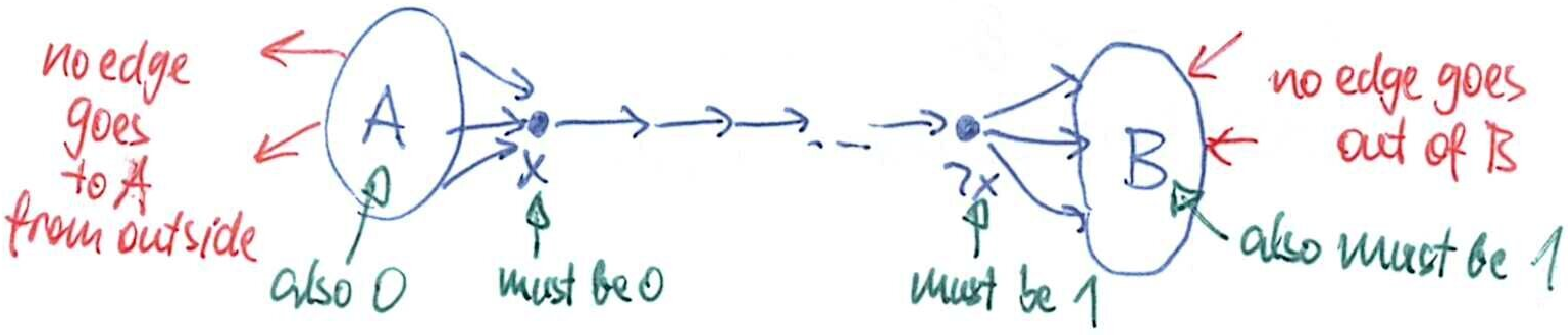
Lemma: φ is unsatisfiable $\Leftrightarrow$ $\exists$ var. v s.t. G_φ contains both a path $v \to \neg v$ and a path $\neg v \to v$] "contradictory cycle"

Proof: First observe: if literal α is set to 1, all literals reachable from α must be also 1.
... if v set to 0, all literals from which v is reachable must be also 0.

Hence $\Leftarrow$ is true.

$\Rightarrow$: prove contra-positive: If there $\nexists$ a contradictory cycle, we construct a satisfying assignment.

① If there exists a path $x \to \neg x$:



Also, A is the mirror image of B:
- literals negated
- edge directions flipped

If $A \cap B \neq \emptyset$: $\exists$ contradictory cycle.
Otherwise: remove $A, B, x, \neg x$ & continue (because of red note, the removed part cannot affect SAT'ability of the rest)

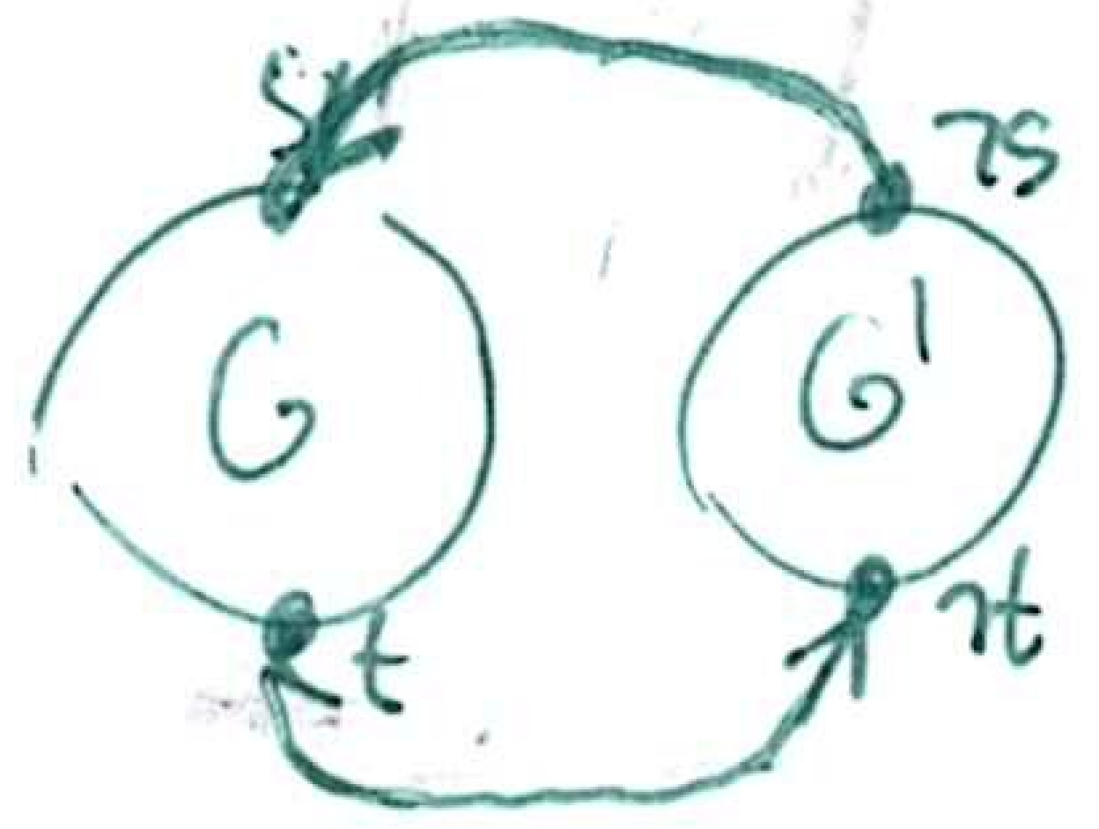
② $\exists$ path $\neg x \to x$: symetrically.

③ no such paths exist: add edge $x \to \neg x$ for some remaining variable x and continue (this couldn't have created a new contradictory cycle)
effectively setting $x=0$

Corollary: 2-SAT $\in$ P (in fact, there is an $O(n)$-time alg. on the RAM)

Thm: 2-SAT is NL-complete.

Proof: REACH ∈ NL, which is also co-NL, so $\overline{\text{REACH}}$ ∈ NL.
We can decide 2-SAT using a subroutine for $\overline{\text{REACH}}$, so 2-SAT ∈ NL.
NL-hardness: REACH is NL-complete, NL = co-NL, so $\overline{\text{REACH}}$ is also NL-complete.
Let's reduce $\overline{\text{REACH}}$ to 2-SAT in log space:



- given graph G and vertices s, t (if s=t, REJECT) ← by producing a constant non-SATable formula
- build a formula with implication graph G, containing only positive literals (a disjoint copy of G gets created with the mirror image...)
- add implications $t \Rightarrow \neg t$, $\neg s \Rightarrow s$ (~~this creates $s \Rightarrow \neg t$, $\neg s \Rightarrow t$~~)
- resulting formula is SATable ⟺ ∄ path $s \to t$ (no other contradictory cycle is possible)

HIERARCHY THEOREMS

Goal: Show that some classes are different ☺

Tools:
- time/space-constructibility of functions
- enumeration of machines M_α (we can also use integer codes instead of strings)
- Universal Turing Machine (UTM): given $\langle \alpha, \beta \rangle$, simulates M_α on input β.
  - complexity: if M_α runs in time T and space S, on input β, UTM(α, β) runs in:
    - space $\in O(S)$
    - time $\in O(T^2)$ or $O(T \log T)$

constants dependent on α (e.g., size of work alphabet)

$O(T^2)$: because of reduction k tapes → 1 tape

$O(T \log T)$: can use a better reduction k → 2 tapes (we haven't proven that)

- can extend the UTM to count space/time used
  - by the simulated machine
  - by the UTM itself

  & stop simulation if limit exceeded

Theorem (space hierarchy): If f, g are non-decreasing space-constructible functions, $f \in o(g)$ and $g(n) \geq \log n$, then
$$\text{DSPACE}(f(n)) \subsetneq \text{DSPACE}(g(n)).$$

Proof: ⊆ trivial, will construct a language $L \in \text{DSPACE}(g(n)) \setminus \text{DSPACE}(f(n))$.

Define machine M: Given input β:

1. Check that β has the form $\alpha 1 0^l$ for some α, l.
2. Write $g(|\beta|)$ 1s on a work tape X.
3. Simulate M_α on input β using an UTM.
   Stop if more than $g(|\beta|)$ cells are used by the UTM (& assume M_α rejected then)
4. If M_α accepted, reject.
   If rejected, accept.

then $L := L(M)$

We check that M runs in space $O(g(n))$, so $L \in DSPACE(g(n))$.
Let's show that $L \notin DSPACE(f(n))$... if it were true, there $\exists M_\alpha$ deciding L in space $f'(n) \in O(f(n))$.
So the UTM can simulate M_α in space $\underbrace{c \cdot f'(n)}$ for some c (depending on α).

this is in $o(g(n))$, so $cf'(n) < g(n)$ for n large enough

Construct input $\beta := \alpha 10^l$ for l large enough.
Then the UTM fits in the ~~time~~ space bound $g(|\beta|) \Rightarrow$ on this input, M_α doesn't agree with M ↯ ■

Notes: The trick with padding α by 10^l is actually not nesessary, because for every machine, there are infinitely many equivalent codes $\Rightarrow$ just pick code α large enough.
[this is more complicated! the constants in complexity of simulation generally depend on α ... but actually only on alphabet & states, which is harmless]

Corollaries: $DSPACE(n) \subsetneq DSPACE(n^2) \subsetneq DSPACE(n^3) \subsetneq \dots$, so $PSPACE \neq DSPACE(n^k)$ for every k.

$DSPACE(n) \subsetneq DSPACE(n \log\log n) \subsetneq DSPACE(n \log n) \subsetneq DSPACE(n^2)$

$\underset{\text{Savitch's thm.}}{NL \subseteq} DSPACE(\log^2 n) \subsetneq DSPACE(n) \subsetneq PSPACE$] so $NL \neq PSPACE$ and $QBF \notin NL$

$PSPACE \subseteq DSPACE(2^n) \subsetneq DSPACE(2^{n^2}) \subseteq EXPSPACE$] so $PSPACE \neq EXPSPACE$

Theorem (time hierarchy): If f, g are time-constructible non-decreasing functions such that $f \cdot \log f \in o(g)$, then $DTIME(f(n)) \subsetneq DTIME(g(n))$.

Proof: Almost identical, modify step 3 to stop the UTM after $g(n)$ steps.
If M_α decides M in time $f' \in O(f)$, then UTM simulates M_α in time at most $f'(n) \log f'(n) \in o(g($
So for large enough equivalent code α, UTM completes simulation of $M_\alpha(\alpha)$ in time $g(|\alpha|)$.
Therefore M_α disagrees with M on input α ↯

Corollaries: $DTIME(n) \subsetneq DTIME(n^2) \subsetneq \dots$ (But we cannot separate $DTIME(n \log n)$ from $DTIME(n)$ this way)

$DTIME(n^k) \subsetneq DTIME(n^{\log n}) \subsetneq EXP$ ... so $P \subsetneq EXP$.

$P \neq DTIME(n^k)$ for every k.

So we have: $$L \subseteq \underbrace{NL \subseteq \overbrace{P \subseteq NP \subseteq PSPACE}^{\neq}}_{\neq} = \underbrace{\overbrace{NPSPACE \subseteq EXP} \subseteq NEXP \subseteq EXPSPACE}_{\neq}.$$

Note: What about non-deterministic classes?

We have $NTIME(f(n)) \subsetneq NTIME(g(n))$
and $NSPACE(f(n)) \subsetneq NSPACE(g(n))$ } whenever $f \in o(g)$

- non-deterministic reduction of tapes can be done with constant overhead in both time & space
- we have non-deterministic UTM
- but how do we negate its output ???
  - for space-bounded classes, use Immerman-Szelepcsényi thm.
  - for time-bounded, a more involved proof is needed (not covered here)

RELATIVE CLASSES

We can define complexity classes for machines with an oracle.
For example P[A] a.k.a. P^A and NP[A] a.k.a. NP^A. ← called "relative" classes wrt. A
Many proofs apply to relativized statements of theorems, too.
But P vs. NP cannot be relativized: ← this limits proof techniques which could separate P from NP
(e.g., diagonalization as in proofs in hierarchy thm's doesn't work)

Theorem: There exist languages A, B s.t. $P[A] = NP[A]$, but $P[B] \neq NP[B]$.

Proof: Ⓐ Let A = QBF. Then $P[A] = PSPACE[A] = PSPACE$
$NP[A] \subseteq PSPACE[A] = PSPACE$.

Ⓑ For every language B, define $U_B := \{1^n \mid \exists \beta \in B \text{ with } |\beta| = n\}$. ← "shadow cast by the language B"
We have $U_B \in NP[B]$: just guess β and check it's in B.
Construct B s.t. $U_B \notin P[B]$: in step i, we make sure that $M_i[B]$ doesn't decide U_B within $2^n/10$ steps for inputs of size n. To achieve that, we put finitely many strings $\underbrace{\text{inside or forever outside } B}_{\text{we "decide their fate"}}$

Step i: Choose minimum n > lengths of all strings whose fate we already decided.
Run $M_i[B]$ on 1^n for $2^n/10$ steps.

- when it queries B for a string β:
  - if fate of β was already decided, answer consistently
  - if not, put β outside B and answer NO
- if it accepted 1^n, arrange $1^n \notin U_B$: so far, no string of length n is in B, put the remaining ones outside B
- if it rejected 1^n, add one string of length n to B (so $1^n \in U_B$)
  ↳ so far, we met at most $2^n/10$ such strings, so some undecided strings must remain.

Now if some machine $M[B]$ decides U_B in time $f(n) \in poly(n)$,
we have $f(n) < 2^n/10$ for n large enough.
For large enough i s.t. M_i is equivalent to M, n is also large enough
$\Rightarrow L(M)$ disagrees with U_B on input 1^n.

So $U_B \notin P[B]$.

CIRCUIT COMPLEXITY

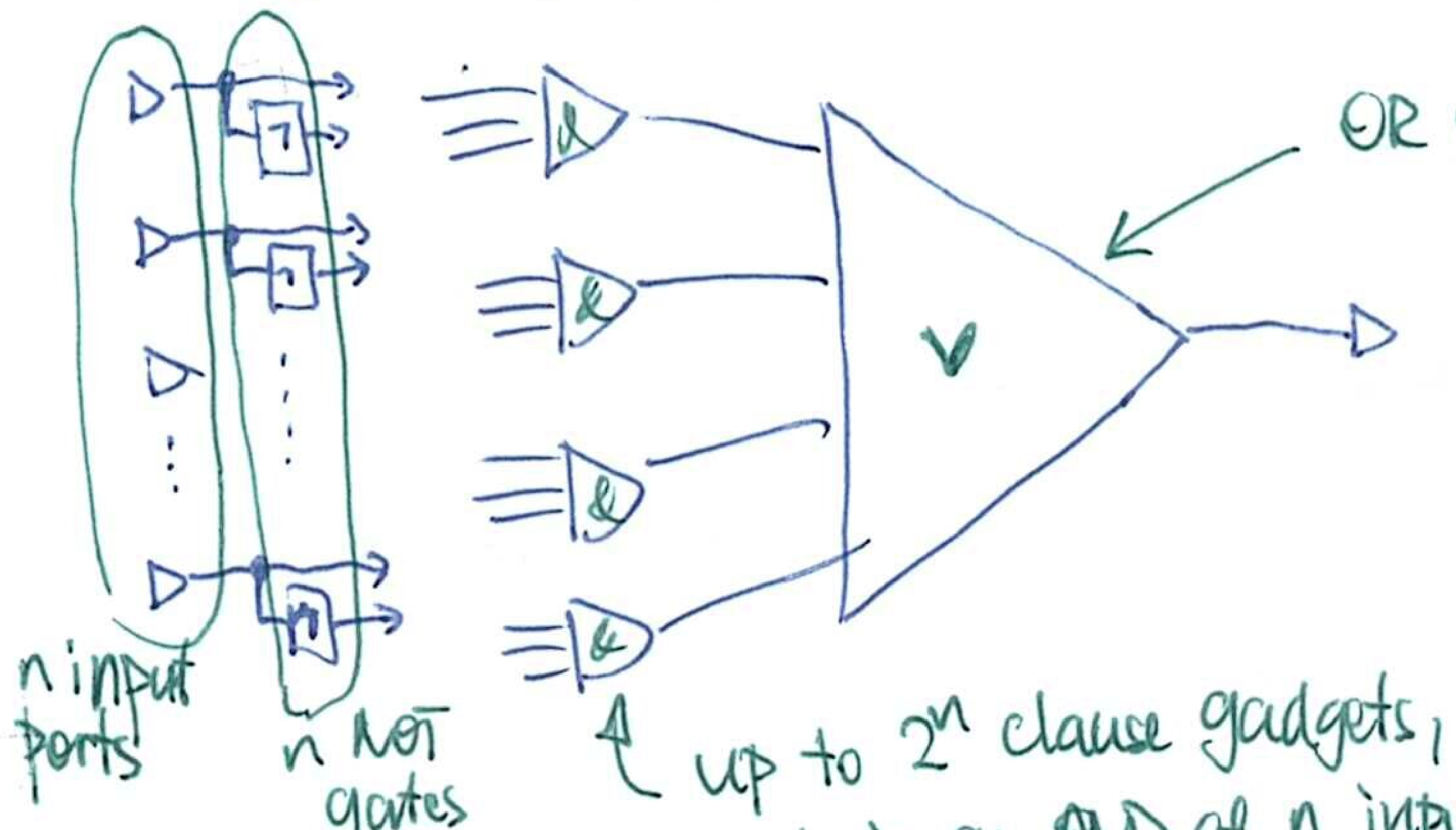
Back to (non-uniform) Boolean circuits.

We will use only AND, OR, NOT gates (other gates can be simulated with constant overhead).

<u>Circuit size</u> = #gates + #ports (input & output) ← size $\le S$ is equivalent with size $= S$ as we can pad circuits with redundant gates

<u>Thm:</u> For every function $f: \{0,1\}^n \to \{0,1\}$ there exists a circuit of size $\le 10n \cdot 2^n$ computing f.

<u>Proof:</u> Use DNF formula for f (see lecture on Cook-Levin thm.)



OR of at most 2^n inputs → 2^n gates

up to 2^n clause gadgets, each is an AND of n inputs → $n \cdot 2^n$ gates

<u>Note:</u> There is a better construction with $O(2^n/n)$ gates. ← surprisingly, this is asymptotically optimal

<u>Exercise:</u> Achieve $O(2^n)$ gates.

<u>Thm:</u> For all sufficiently large n, there is $f: \{0,1\}^n \to \{0,1\}$ which is computed by no circuit of size at most $2^n/10n$.

<u>Proof:</u> There are 2^{2^n} functions from $\{0,1\}^n$ to $\{0,1\}$.

Let's count circuits of a given size: s:

$$\#\text{circuits} \le 3^s \cdot s^{2s} = 2^{s \cdot \log_2 3} \cdot 2^{2s \log s} \le 2^{3s \log s}$$

↑ Except for ports, each gate can be AND, OR, NOT

↑ # interconnections: each gate has at most 2 inputs, which are connected to a port or output of another gate

Now for $s = 2^n/10n$:

$$\#\text{circuits} \le 2^{\frac{3 \cdot 2^n}{10n} \cdot n} < 2^{2^n} \text{ for } n \text{ large enough.}$$

← In fact, the majority of functions has no small circuits

<u>Notes:</u> All problems in P have polynomially large circuits.

Hypothesis (Kolmogorov): $O(n)$ is enough.

Surprisingly, the best lower bound so far is $5n$.

<u>Idea:</u> If we found $L \in NP$ with super-polynomial lower bound for circuit size, then $P \ne NP$.
But so far, we failed completely...

<u>Df:</u> For $S: \mathbb{N} \to \mathbb{N}$ we define $SIZE(S(n))$ as the class of languages, which are computable by a (non-uniform) family $\{C_n\}_{n=0}^{\infty}$ of circuits s.t. size of $C_n \le \boxed{S(n)}$. ← beware, no O here

<u>We know:</u> $P \subseteq \bigcup_k SIZE(n^k \boxed{+k})$ ← this is to overcome finitely many exceptions in O

Df: Computation with advice: the TM gets an extra input (advice), which depends only on the size of the main input.

$\underset{\text{time bound}}{\mathrm{DTIME}(f(n))}/\underset{\text{size of advice}}{g(n)}$: the class of languages L s.t. $\exists M$ Turing Machine and $\exists a: \mathbb{N} \to \{0,1\}^*$ where $\forall \alpha \in \{0,1\}^*$ with $n = |\alpha|$ $M(\langle \alpha, a(n) \rangle)$ halts within $O(f(n))$ steps, accepts iff $\alpha \in L$ and $|a(n)| \leq g(n)$.

Then: $P/g(n) := \mathrm{DTIME}(\mathrm{poly}(n))/g(n)$

- $P/0 = P$
- $P/1 \supsetneq P$, because $P/1$ contains the unary Halting problem!
- $P/\mathrm{poly} = \bigcup_k \mathrm{SIZE}(n^k + k)$ … $\supseteq$ the advice is the circuit, TM just evaluates the circuit; $\subseteq$ we translate the TM to a circuit & hard-wire the advice in it

Circuit lower bounds for NP are hard, but at least we can make one for EXP:

Theorem: $\forall k \geq 0\ \exists L \in \mathrm{EXP} \setminus \mathrm{SIZE}(n^k + k)$. ← beware, this doesn't imply $L \in \mathrm{EXP} \setminus P/\mathrm{poly}$

Proof: L is defined by the following algorithm:

```
1. Let $\alpha$ be an input of size $n$.
2. Let $\beta_0, \dots, \beta_{2^n-1}$ denote possible inputs for an n-input circuit: $\beta_j :=$ j written in binary.
3. $C_0 \leftarrow$ {all circuits of size $n^k+k$ with n inputs}
4. $i \leftarrow 0$
5. While $C_i \neq \emptyset$ & $i < 2^n$:
6.     Simulate all circuits in $C_i$ on input $\beta_i$, let $t_i$ be the minority answer.
7.     $C_{i+1} \leftarrow$ those circuits from $C_i$ which gave output $t_i$
8.     $i \leftarrow i+1$
9. If $\alpha = \beta_j$ for some $j < i$: answer $t_j$
   Else: answer NO ← arbitrary
```

Now: $C_{i+1} \leq \frac{1}{2}|C_i| \Rightarrow C_i \leq |C_0|/2^i$

$|C_0| \leq 2^{n^{k+1}} \Rightarrow$ after less than 2^n steps, we get $C_i = \emptyset$ — works for n large enough (i.e., we don't run out of β_j's)

↑ see calculations in circuit lower bound thm. (for size s, it was at most $2^{3s\log s}$)

So the whole algorithm runs in time $O(2^{n^{k+2}})$, so $L \in \mathrm{EXP}$.

But no circuit in $\mathrm{SIZE}(n^k+k)$ can agree with L on n large enough.

Improvement: Choose $k := \lfloor \log n \rfloor$ … then run time is in $O(\underset{2^{2^{\log^2 n+2}}}{2^{n^{\log n+2}}}) \subseteq \underbrace{O(2^{2^n})}$

this is called EEXP or 2-EXP (compare with $\mathrm{EEXP} = \mathrm{DTIME}(2^{2^{\mathrm{poly}(n)}})$)

So L is in EEXP, but not in $\mathrm{SIZE}(n^k+k)$ for any k.

So $L \in \mathrm{EEXP} \setminus P/\mathrm{poly}$.

Therefore $\mathrm{EEXP} \not\subseteq P/\mathrm{poly}$.

Theorem: If $NP \subseteq P/poly$, then $PH = \Sigma_2^P$. ← generally, it's believed that the PH does not collapse, so there should be languages in NP with no poly-size circuits

Proof: (not shown at the lecture)

- we want to show that $\Sigma_2^P = \Pi_2^P$
- so $\Pi_2^P \subseteq \Sigma_2^P$ suffices (the other inclusion by taking complements)
- so Π_2-SAT $\in \Sigma_2^P$ suffices
  - ↑ this is $\{ \langle\psi\rangle \mid \psi$ is a true formula of the form $\forall\alpha\in\{0,1\}^n\ \exists\beta\in\{0,1\}^n\ \varphi(\alpha,\beta) \}$
  - ↑ unquantified fla of size $O(n)$
- If $NP \subseteq P/poly$, there is a family of poly-size circuits $\{C_n\}_{n=0}^{\infty}$ solving: given $\langle\varphi\rangle$ and α, is there β s.t. $\varphi(\alpha,\beta)$ is true?
  - ↳ we can convert this to $\langle\varphi\rangle, \alpha \mapsto$ find that β ... still within polynomial size → circuits C'_n
  - ↑ the exercise with SAT oracle earlier ...
- We don't know how C'_n looks, but we can guess it and verify:

  $\exists \langle C'_n\rangle\ \forall\alpha\ \varphi(\alpha, C'_n(\alpha))$ ... this solves Π_2-SAT, but it is in Σ_2^P.

PROBABILISTIC ALGORITHMS (a.k.a. randomized)

Define the **Probabilistic TM** (PTM): random states & exactly 2 possible instructions, the TM decides by flipping a fair coin (i.e., generates an uniformly random bit, independent of all the other random bits)

← another form of non-determinism like ∃ & ∀ states

↓

We have a probability distribution on computations

↳ $P(M \text{ accepts } \alpha)$

Df:
- BPTIME$(f(n))$:= class of all languages L s.t. $\exists M$ PTM: all computations halt within $O(f(n))$ steps and $P(M(\alpha) = L(\alpha)) \geq 2/3$.
  - ↑ indicator of $\alpha \in L$
  - 2-sided error
  - ↓ BPP := BPTIME(poly(n))
- RTIME$(f(n))$:= class of all languages L s.t. $\exists M$ PTM: all computations halt within $O(f(n))$ steps,
  - if $\alpha\in L$: $P(M(\alpha) \text{ accepts}) \geq 2/3$
  - if $\alpha\notin L$: $P(M(\alpha) \text{ accepts}) = 0$
  - 1-sided error
  - ↓ RP := RTIME(poly(n)), co-RP (error at the opposite side)

Amplification of probability of success:

① For RP: Let $L\in RP$, M the corresponding PTM. Run $M(\alpha)$ t times independently, accept if at least one run accepted.

→ this is still poly-time ...

$\alpha\notin L$: always rejected

$\alpha\in L$: rejected with pr. $\leq (1-2/3)^t$

↓ symetrically for co-RP

Corollary: Iterating decreases pr. of error exponentially with # tries. This works whenever the original $P(\text{accepts}) \geq c$ for any $c>0$. So the definition is robust w.r.t. change of the (arbitrary) 2/3.

② For BPP: Run $\underset{\text{odd}}{t}$ times independently, use majority answer.

Analysis: Let random variable X_i := indicator of correct answer in try #i, $E[X_i] \geq \frac{2}{3}$

assume $= \frac{2}{3}$, more is obviously better

Let $X := \sum_i X_i$ (# successful tries), then $\mathbb{E}[X] = \frac{2}{3} \cdot t$

$P(\text{majority is wrong}) = P(X < \frac{1}{2} t)$

this is Chernoff with

$\mu = \frac{2}{3} t$, $(1-\delta) \cdot \frac{2}{3} = \frac{1}{2}$, so $\delta = \frac{1}{4}$.

Hence $P \leq e^{\frac{-(\frac{1}{4})^2 \cdot \frac{2}{3} \cdot t}{2}} = e^{-\frac{2 \cdot t}{4^2 \cdot 3 \cdot 2}} = e^{-\Omega(t)}$

Tool: Chernoff's bound for the left tail:
Let $X_1 - X_k$ be independent random vars with domain $\{0,1\}$, $X = \sum_i X_i$, $\mu = \mathbb{E}[X_i]$, $\delta \in (0,1)$. Then:

$$P(X < (1-\delta)\mu) \leq e^{-\frac{\delta^2 \mu}{2}}$$

So Pr. of error again decreases exponentially with # tries.
This works whenever the original machine answers correctly with $P \geq c$, where $c > \frac{1}{2}$.

Certificate-based definition: • BPP is the class of all languages L s.t. $\exists V \in P$ ← verifier and
$\forall \alpha \in \{0,1\}^*$ $P_{\beta \in \{0,1\}^{\text{poly}(n)}}(V(\langle \alpha, \beta \rangle) = L(\alpha)) \geq \frac{2}{3}$.
↑ input ↑ certificate

↳ Why this is the same BPP: old ⇒ this: the certificate is a sequence of all random bits generated by the TM, the rest can be simulated deterministically
padded to the same size for all computations
this ⇒ old: first generate random β, then run V.

- for RP: $L \in RP \Leftrightarrow \exists V \in P$ $\forall \alpha \in \{0,1\}^*$: $P_{\beta \in \{0,1\}^{\text{poly}(n)}}(V(\langle \alpha, \beta \rangle) = 1) \begin{cases} \geq 2/3 & \text{if } \alpha \in L \\ = 0 & \text{if } \alpha \notin L \end{cases}$

↳ this implies $RP \subseteq NP$

"Zero-sid~~bas~~ed errors": Two definitions: ① TM runs in expected time $O(t(n))$, always answers correctly
ZTIME(t(n))
↓
ZPP := ZTIME(poly(n))

② TM runs in worst-case time $O(t(n))$, can answer MAYBE.
If answer is not MAYBE, it's correct.
P(answers MAYBE) ≤ 1/3.

① ⇒ ② Run machine for $3 \cdot t(n)$ steps, if it times out, answer MAYBE.
$P(\text{MAYBE}) = P(\text{time} \geq 3 \cdot \mathbb{E}(\text{time})) \leq \frac{1}{3}$
↑ by Markov's inequality

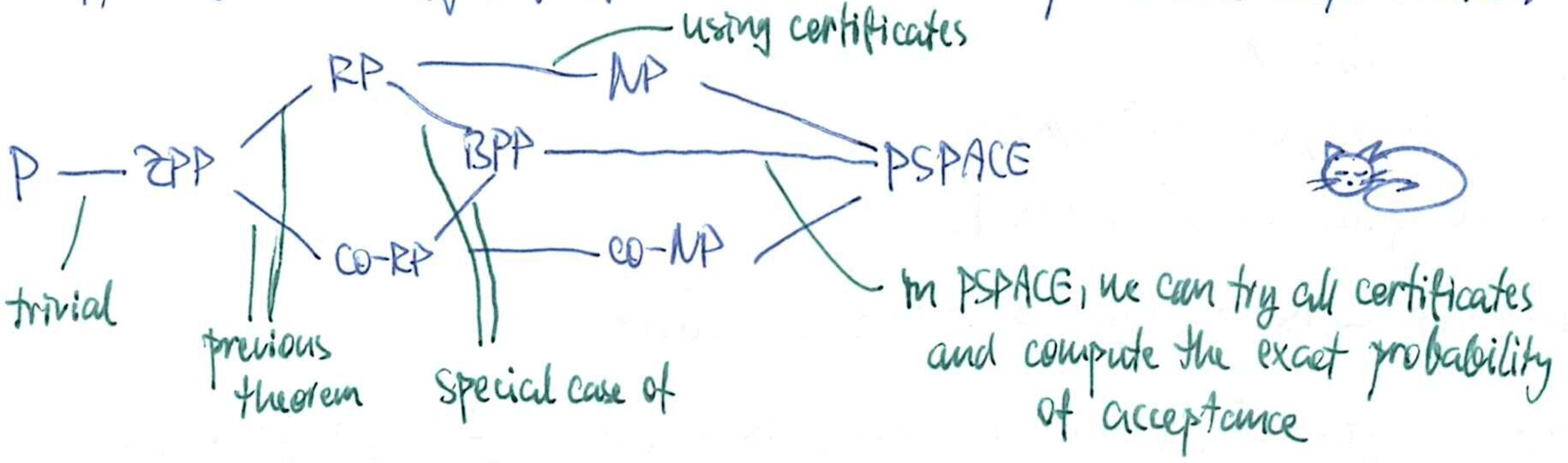
② ⇒ ① Run machine repeatedly as long as it returns MAYBE.
Tool: "Water jug lemma" (a.k.a. geometric distribution)
If P(1 try succeeds) = p, then $\mathbb{E}$(# tries until the 1st success) = $1/p$.
$\mathbb{E}$(# tries) ≤ 3, so $\mathbb{E}(\text{time}) \in O(t(n))$.

Theorem: ZPP = RP ∩ co-RP.

Proof: ① ZPP ⊆ RP: use worst-case def. of ZPP, translate MAYBE to NO.
② ZPP ⊆ co-RP: the same, but MAYBE → YES.
③ RP ∩ co-RP ⊆ ZPP: let M_1 be the machine witnessing L∈RP, M_2 for L∈co-RP.
Run both. If they agree, use their answer. Otherwise return MAYBE.
(!) both cannot be wrong simultaneously → if they agree, the answer is right.
$P(\text{MAYBE}) \leq \frac{1}{3}$ (if $\alpha \in L$, M_2 cannot fail, if $\alpha \notin L$, M_1 cannot fail)

Exercise: What happens if we modify def. of BPP or RP to use expected time and/or MAYBE?

Inclusions:



Excercise: Define class PP: $L \in PP \equiv \exists M$ PTM running in w.c. time $O(poly(n))$ s.t. $P(L(\alpha) = M(\alpha)) > 1/2$ for all α.

beware: amplification doesn't work for PP (not exponentially!)

Show that: $NP \subseteq PP$, $co\text{-}NP \subseteq PP$, $BPP \subseteq PP$, $PP \subseteq PSPACE$.

Theorem: $BPP \subseteq P/poly$.

Proof: For inputs of size n: iterate $O(n)$ times to get $P(\text{error}) \le \frac{1}{2} \cdot 2^{-n}$ ← amplify

Let $L \in BPP$

Let $r :=$ max. # random bits used by the machine (certificate size)

For a fixed input α: $P_{\beta \in \{0,1\}^r}(V(\langle\alpha,\beta\rangle) \neq L(\alpha)) \le \frac{1}{2} \cdot 2^{-n} \Rightarrow$ #"bad" certs for which this happens $\le 2^r \cdot \frac{1}{2} \cdot 2^{-n}$

↳ taking union over all α: # bad certs $\le 2^r \cdot \frac{1}{2} \cdot 2^{-n} \cdot 2^n = \frac{1}{2} \cdot 2^r < 2^r$

$\Rightarrow$ there exists a certificate which is good for all inputs: this will be the advice.

So our algorithm just calls V on $\langle$input, advice$\rangle$. This implies $L \in P/poly$.

Notes: It is known that $BPP \subseteq \Sigma_2^P \cap \Pi_2^P$ (Sipser-Gács theorem) ← this is stronger than $BPP \subseteq PSPACE$.

There are no known BPP-complete problems nor hierarchy theorems. ← BPP is a "semantic" class, so diagonalization doesn't work

It's believed that BPP = P (otherwise hard-to-believe things happen)

REGULAR LANGUAGES

Df: Deterministic Finite-state Automaton (DFA) consists of:

- Q – a finite non-empty set of states
- Σ – a finite non-empty alphabet
- $\delta: Q \times \Sigma \to Q$ – transition function
- $q_0 \in Q$ – initial state
- $F \subseteq Q$ – a set of accepting states

Df:
- Computation of a DFA over an input string $\alpha \in \Sigma^*$ is a sequence of states $s_0, s_1, \dots, s_{|\alpha|}$ ← uniquely determined such that $s_0 = q_0$ and $\forall i\ s_{i+1} = \delta(s_i, \alpha[i])$.
- The input is accepted $\equiv s_{|\alpha|} \in F$.
- $L(A)$ :: the language of all words accepted by the automaton A.

alternatively:
- DFA is a multi-graph with labelled edges (by Σ)
- computation is a walk in the graph starting in q_0 and labelled by the input α.

<u>Df:</u> <u>Extended transition function</u> $\delta^*: Q \times \Sigma^* \to Q$ ← $\delta^*(s,\alpha)$ is the final state of a computation on α starting in state s.

s.t. $\delta^*(s,\varepsilon) := s$

$\delta^*(s,\alpha x) := \delta(\delta^*(s,\alpha), x)$

α is accepted $\Leftrightarrow \delta^*(q_0,\alpha) \in F$.

<u>Df:</u> Language L is <u>regular</u> $\equiv \exists$ DFA A: $L(A) = A$.

<u>Example:</u> $\{\alpha \in \{0,1\}^* \mid \#1 \text{ in } \alpha \text{ is even}\}$ is regular (2 states: #1 mod 2)

<u>Example:</u> Every finite language is regular (states: prefixes of words in the language)

<u>Example:</u> $\{0^n1^n \mid n \geq 0\}$ is <u>not</u> regular. If there existed a DFA accepting it: set $t := |Q|$, consider $s_0 — s_t$, where $s_i := \delta^*(q_0, 0^i)$. By Pigeon-hole principle, there is $i<j$ s.t. $s_i = s_j$. Now $\delta^*(q_0, 0^i1^i) = \delta^*(q_0, 0^j1^i)$, so 0^i1^i is accepted $\Leftrightarrow$ 0^j1^i is. ↯

<u>Lemma (Pumping lemma for regular languages):</u>

For every regular language L, there exists $n \geq 0$ such that:

Every $w \in L$, $|w| \geq n$ can be decomposed as $w = \alpha\beta\gamma$, where:

① $\forall t \geq 0$ $\alpha\beta^t\gamma \in L$ (including $i=0$)

② $\beta \neq \varepsilon$

③ $|\alpha\beta| \leq n$.

<u>Proof:</u> Consider an automaton accepting L. Set $n := |Q|$.

Given $w \in L$, $|w| \geq n$, define $s_0 — s_m$: $s_i := \delta^*(q_0, w[:i])$

↑ let $m := |w|$

Since $m \geq n$, there is $i<j\leq n$ s.t. $s_i = s_j$.

Now set $\alpha := w[:i]$, $\beta := w[i:j]$, $\gamma := w[j:]$. ... this implies ② and ③

① $\delta^*(q_0,\alpha) = s_i = s_j = \delta^*(q_0,\alpha\beta)$ ... so $\delta^*(s_i,\beta) = s_i$, hence $\forall t \geq 0$ $\delta^*(q_0,\alpha\beta^t) = s_i$, so $\forall t$ $\delta^*(q_0,\alpha\beta^t\gamma)$ is always the same. For $t=1$, $\alpha\beta^t\gamma \in L$, so all $\alpha\beta^t\gamma \in L$.

<u>Example:</u> 0^n1^n again ... If it were regular, use 0^n1^n with n from the lemma.

Both α, β must consist purely from 0s, so $\alpha\beta^t\gamma$ is 0^i1^j and we can increase i, while staying inside the language. ↯

<u>Lemma:</u> Intersection of two regular languages (over a common alphabet) is regular.

<u>Proof:</u> Let L_1, L_2 be regular, DFA $A_1 = (Q_1, \Sigma, q_{01}, F_1)$ accepting L_1 and DFA $A_2 = (Q_2, \Sigma, q_{02}, F_2)$ accepting L_2.

Construct a <u>product</u> of A_1 and A_2:

$Q := Q_1 \times Q_2$

$\delta((s_1,s_2),x) := (\delta_1(s_1,x), \delta_2(s_2,x))$

$q_0 := (q_{01}, q_{02})$

$F := F_1 \times F_2$

we have $\delta^*((s_1,s_2),\alpha) = (\delta_1^*(s_1,\alpha), \delta_2^*(s_2,\alpha))$

so $\alpha \in L(A) \Leftrightarrow \alpha \in L_1 \cap L_2$.

Intuition: Run A_1, A_2 in parallel, accept iff both accepted.

Exercise: Regular languages are also closed under complement and union.

Df: Non-deterministic Finite-state Automaton (NFA)

Like DFA, but $\delta: Q \times \Sigma \to \mathcal{P}(Q)$ - we have multiple possible instructions to execute
and $Q_0 \subseteq Q$ replaces q_0 - multiple initial states

What changes: Computation requires $s_{i+1} \in \delta(s_i, \alpha[i])$, $s_0 \in Q_0$
There can be multiple computations for a given input, or perhaps none.
α is accepted $\equiv$ there exists a computation ending in an accepting state.

Df: $\delta^*: \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ defined as:
$\delta^*(S, \varepsilon) := S$, $\delta^*(S, \alpha x) := \bigcup_{t \in \delta^*(S,\alpha)} \delta(t, x)$.
} again: α is accepted $\Leftrightarrow \delta^*(Q_0, \alpha) \cap F \neq \emptyset$.

Thm: If L is accepted by an NFA, then it is regular. ← so non-determinism doesn't increase computing power of FAs

Proof: Construct a DFA $A'(Q', \Sigma, \delta', q_0', F')$ which simulates δ^* of the original NFA $A(Q, \Sigma, \delta, Q_0, F)$.

Let $Q' := \mathcal{P}(Q)$
$\delta'(s, x) := \delta^*(s, x)$ → Then $\delta'^*(q_0', \alpha) = \delta^*(Q_0, \alpha)$,
$q_0' := Q_0$ so $\alpha \in L(A') \Leftrightarrow \alpha \in L(A)$.
$F' := \{s \in Q \mid s \cap F \neq \emptyset\}$

Nicer generalization: ε-NFA, which adds ε-edges: these can be traversed without reading a symbol from the input

Df: Extend $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$.

👁 Computation = walk from $q_0 \in Q_0$ s.t. concatenated edge labels yield input string.

Df: ε-closure $U_\varepsilon(s)$ of a state s := set of all states reachable from s using only ε-edges.
$U_\varepsilon(S)$ of $S \subseteq Q$:= $\bigcup_{s \in S} U_\varepsilon(s)$.

↳ extendending δ^* to ε-NFAs: $\delta^*(S, \varepsilon) := U_\varepsilon(S)$
$\delta^*(S, \alpha x) := U_\varepsilon\left(\bigcup_{t \in \delta^*(S,\alpha)} \delta(t, x)\right)$

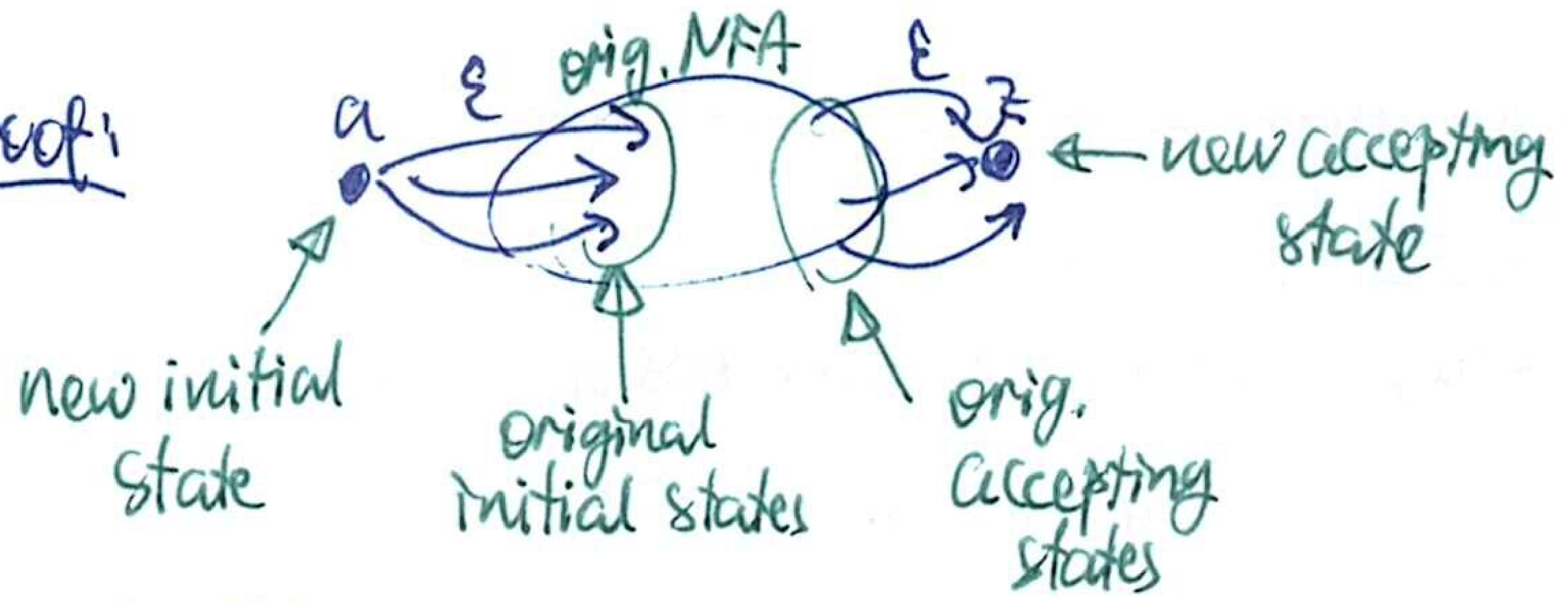
Thm: For every ε-NFA $A(Q, \Sigma, \delta, Q_0, F)$, there is a NFA $A' = (Q', \Sigma, \delta', Q_0', F')$ accepting the same language.

Proof: Just add ε-closure: $Q' := Q$
$Q_0' := U_\varepsilon(Q_0)$
$\delta'(S, x) := U_\varepsilon(\delta(S, x))$
$F' := F$
] so $\delta'^*(S, x) = \delta^*(S, x)$, hence $L(A) = L(A')$.

💡 ε-NFAs accept still the same regular languages, but they are easier to construct.

Lemma: For every ε-NFA, there is an equivalent ε-NFA (accepting the same language) which has a unique initial state (with no incoming edges) and unique accepting state (with no outgoing edges)

Proof:



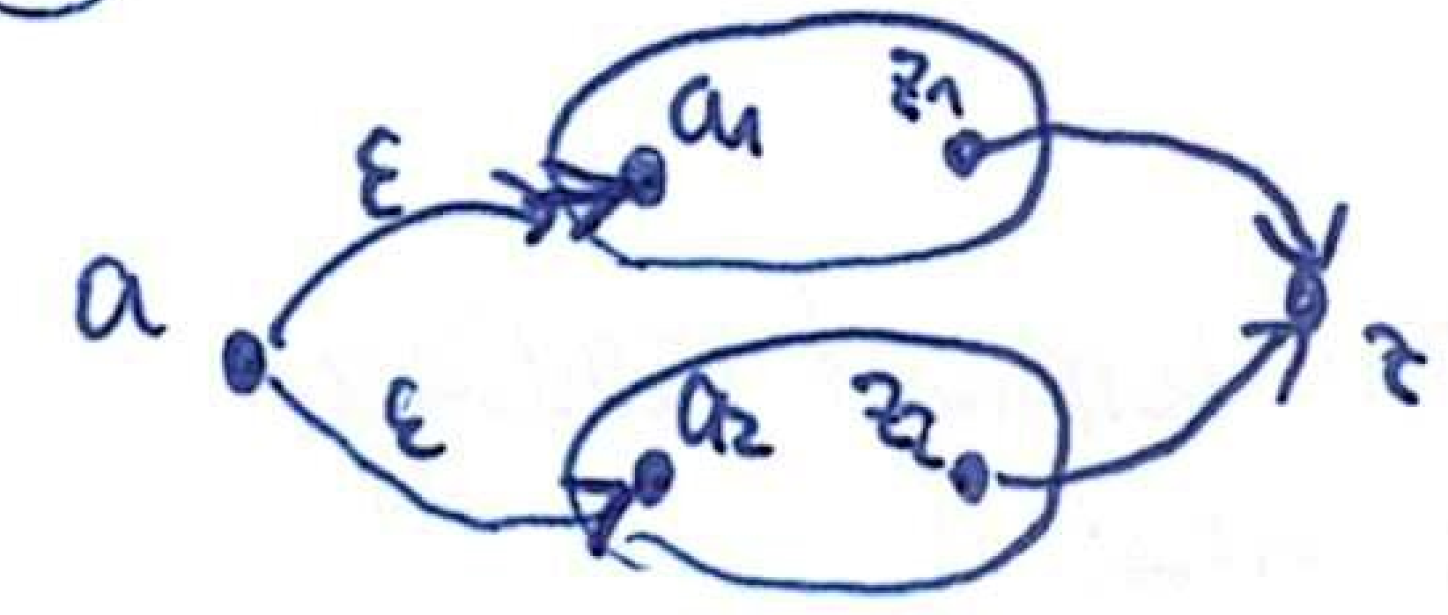
Theorem: The following operations with languages preserve regularity:

- $\overline{L}$ complement
- $L_1 \cap L_2$ intersection
- $L_1 \cup L_2$ union
- $L_1 \cdot L_2 := \{\alpha \cdot \beta \mid \alpha \in L_1, \beta \in L_2\}$ concatenation (associative)
- L^k: $L^0 := \{\varepsilon\}$, $L^{t+1} := L^t \cdot L$ power
- $L^* := \bigcup_{t \geq 0} L^t$ iteration
- $L^+ := \bigcup_{t > 0} L^t$ positive iteration
- $L^R := \{\alpha^R \mid \alpha \in L\}$ reversal

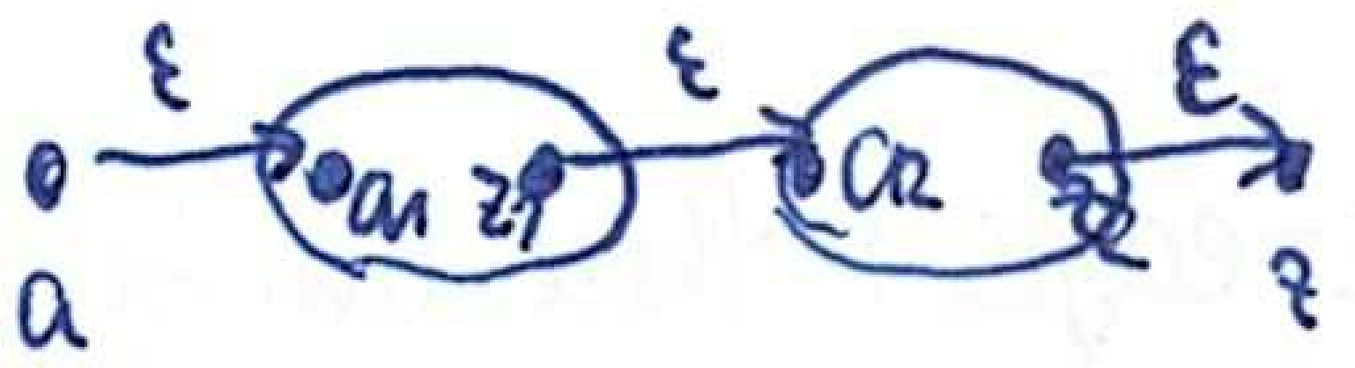
word written backwards

Proof: For $\overline{L}$ and $L_1 \cap L_2$, we already have the proof. Otherwise use ε-NFAs with unique init/acc. state.

① Union

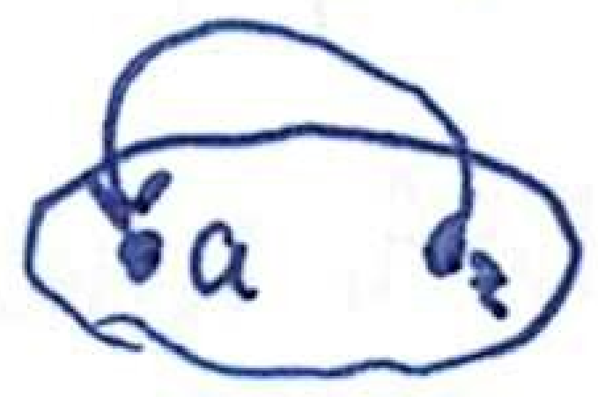


② Concatenation



& hence also powers

③ Positive iteration



④ Iteration: add union with $\{\varepsilon\}$

⑤ reversal: swap role of a, z, switch orientation of all edges.

this is an equivalent definition of regularity which does not use automata

Theorem (Kleene): L is regular $\Leftrightarrow$ L can be constructed from $\emptyset$, $\{\varepsilon\}$, $\{x\}$ for $x \in \Sigma$, using finitely many unions, concatenations and iterations.

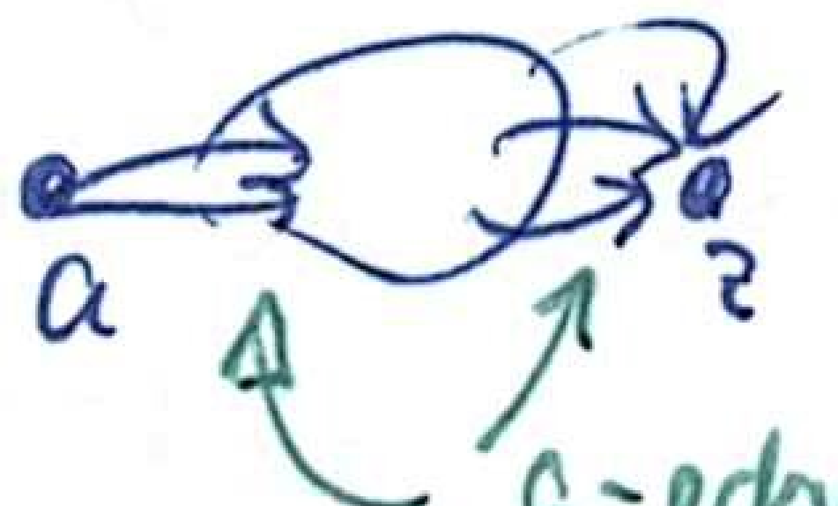
Proof: $\Leftarrow$ follows from the previous thm.

Prove $\Rightarrow$ using even more generalized NFAs, where each edge is labelled by a language and we can traverse the edge if we read a word in that language from the input.

Consider a DFA accepting L. We will transform it gradually to $a \xrightarrow{L} z$, ~~who~~ always preserving the accepted language & making sure that languages on the edges can be constructed in the required way (using $\cup, \cdot, *$).

Steps:

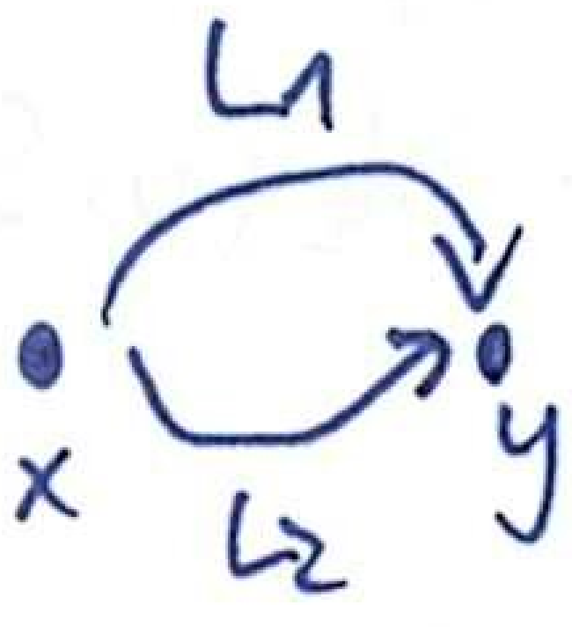
⓪ Initialization: add unique init. & acc. states:



ε-edges (labelled by language $\{\varepsilon\}$)

repeat while there are parallel edges

① elimination of parallel edges: replace



by $x \xrightarrow{L_1 \cup L_2} y$ (we can have $x=y$ here)

② elimination of states: remove a state $s \neq a, z$, routing around it:

a) if s has no loops: replace all $x \xrightarrow{L_1} s \xrightarrow{L_2} y$ by $x \xrightarrow{L_1 \cdot L_2} y$

b) if s has a loop: replace all $x \xrightarrow{L_1} \overset{\circlearrowleft L_2}{s} \xrightarrow{L_3} y$ by $x \xrightarrow{L_1 \cdot L_2^* \cdot L_3} y$

repeat until only a, z remain

Theorem: DSPACE(1) = NSPACE(1) = class of all regular languages.

Building the proof: Deterministic machines first.

① TMs with just the input tape, which is read only & head doesn't move left
… this is equivalent to a DFA (technical detail: how do we accept/reject?)

② Allow moving left. Tech. detail: delimit the input as $\langle\alpha\rangle$. On $\langle$, the TM must ~~move right~~ not move left, on $\rangle$, it must ~~move left~~, not move right

[This is called the bi-directional DFA. We will prove that these accept just regular languages.
(Infinite loop / divergence is interpreted as rejecting the input.)

③ Allow work tapes of constant size: their contents & head positions can be moved inside machine state → this is equivalent to ②.

④ Non-deterministic TMs: ① becomes NFA, so also regular
② will need a generalized proof
③ still reduces to ②.

↳ Need to prove: If L is accepted by a bi-dir. DFA, then L is regular.

Consider computation of the bi-dir. DFA on suffixes of a given input α:
For suffix $\alpha[i:]$:

- we start on $\alpha[i]$ in some state s
- we let the computation run until
  - it stops in q^+ or q^-
  - it diverges (equivalent to q^-)
  - it leaves the suffix $\alpha[i:]$ by moving left from position i.
- we can describe this behavior by a function $f_i : Q \setminus \{q^0, q^-\} \to Q$

The f_i's can be constructed backwards…

- $f_{|\alpha|}$ is trivial (↓ the TM stands at $\rangle$, so the TM must not move right, so iterate δ until it moves left/stops/diverges)
- $f_{i+1} \to f_i$: for $f_i(s)$, construct a sequence of states:

$s_0 = s$

$s_j \to s_{j+1}$: if $s_j = q^+/q^-$, stop & define $f_i(s) := s_j$

otherwise evaluate $\delta(s_j, \alpha[i]) \to (s_j', \text{movement})$
- if $s_j \in \{s_+, s_-\}$:
- if movement $= \leftarrow$: stop & define $f_i(s) := s_j'$
- if movement $= \bullet$: $s_{j+1} := s_j'$ & continue
- if movement $= \rightarrow$: $s_{j+1} := f_{i+1}(s_j')$ & continue

if $s_{j+1} = s_i$ for $i \le j$, the machine diverged, so $f_i(s) := q^-$ & stop.

⇒ f_i is a function of f_{i+1} and $\alpha[i]$.

So there is a DFA processing α^R, whose states are the f_i's.
α^R is accepted ⇔ $f_{\cancel{0}\,-1}(q_0) = q^+$

states are all functions from Q to Q, states visited during computation are the f_i's.

so L^R is regular, therefore L is also regular.

minor technicality: We let the TM start on $\langle$ instead of the first char. of α

← one more technicality: the DFA cannot make one more step to get f_{-1}, because $\langle$ is not a part of input. But this condition can be answered from f_0, too.

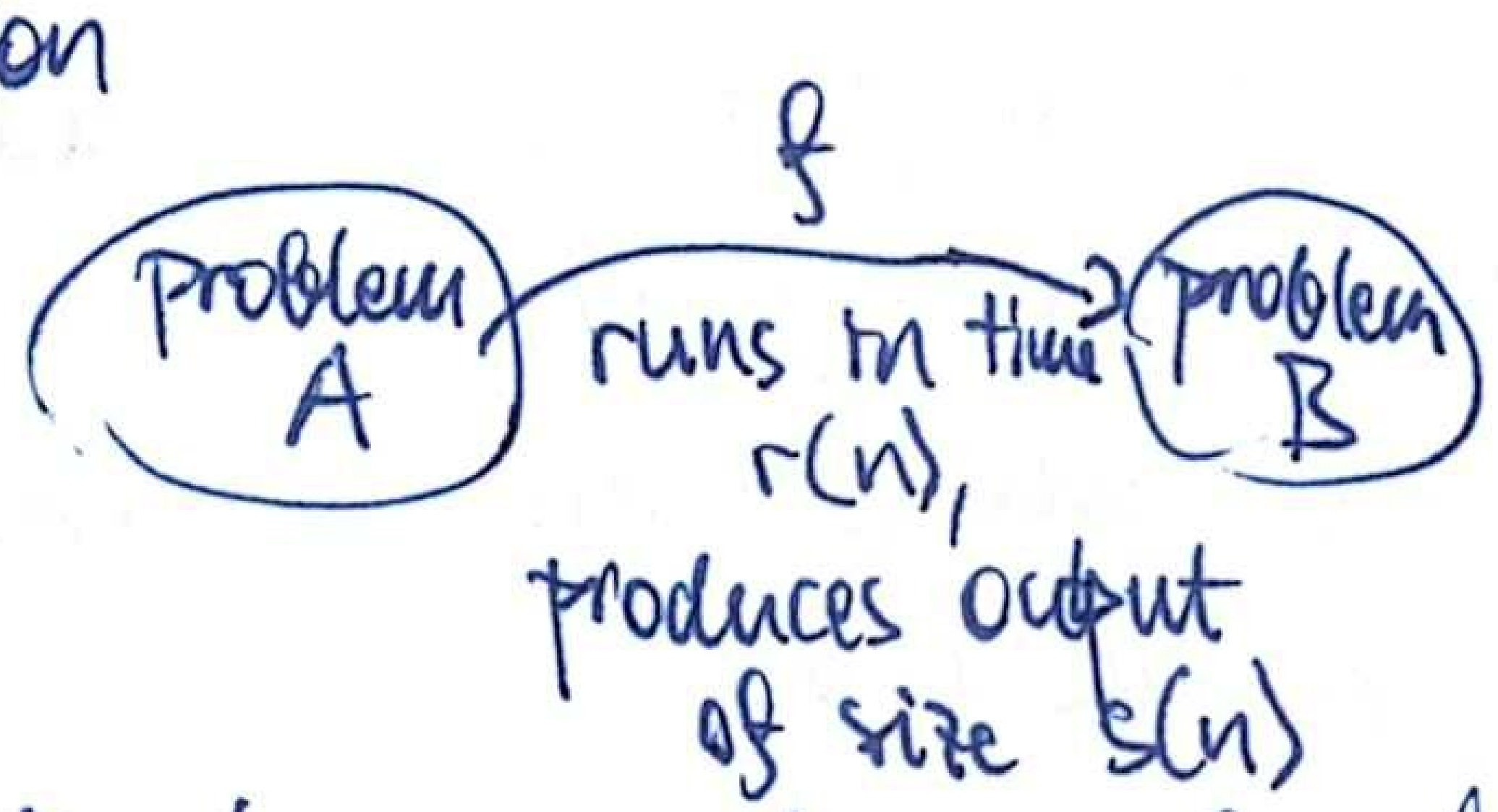
Exercise: Modify the proof to work for non-deterministic TMs.

FINE-GRAINED COMPLEXITY

Goal: Finer results than "polynomial vs. exponential"
E.g., prove that a $\Theta(n^2)$-time alg. is optimal.

Caveat: This will be model-dependent. We will assume RAM here.

Tool: Fine-grained reduction



If B can be solved in time $T(n)$, then A can be solved in time $O(r(n) + T(s(n)))$

covers copying of input/output of f

Upper bounds for B imply upper bounds for A.
Lower bounds for A imply lower bounds for B.

Orthogonal Vectors Problem (OV):

Input: two sets of vectors $A, B \subseteq \{0,1\}^d$, $|A|, |B| \leq n$

Question: are there $a \in A$, $b \in B$ s.t. $\langle a, b \rangle = 0$? ← i.e., bitwise AND is everywhere zero

Baseline algorithms: $O(n^2 d)$ trivial, $O(nd \cdot 2^d)$ ← for each $a \in A$, construct all orthogonal vectors and look them up in a suitable data structure for B (e.g., a trie)

Hypothesis (OVH): For no $\varepsilon > 0$, there is an algorithm solving OV in time $O(n^{2-\varepsilon} \cdot \text{poly}(d))$.

NFA Acceptance problem (NFAA):

Input: Non-deterministic finite-state automaton M of size $|M| = \#\text{states} + \#\text{transitions}$, string α. The alphabet is $\{0,1\}$.

Query: Does M accept α?

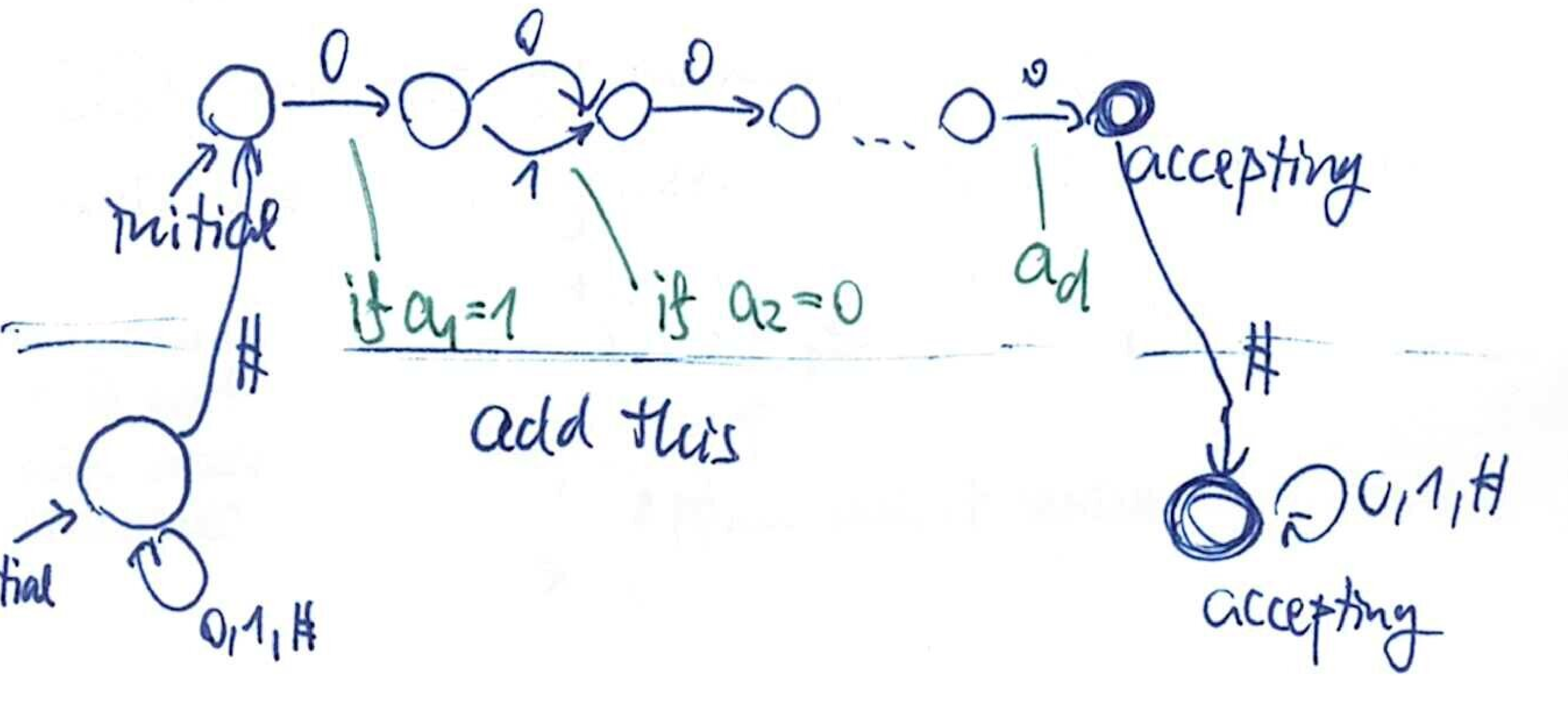
Baseline: $O(|M| \cdot |\alpha|)$ by computing δ^* (see previous lecture)
$O(2^{|M|} + |\alpha|)$ by reducing to a DFA first.

Theorem: Assuming OVH, there is no $\varepsilon > 0$ s.t. NFAA can be solved in time $O((|M| \cdot |\alpha|)^{1-\varepsilon})$.

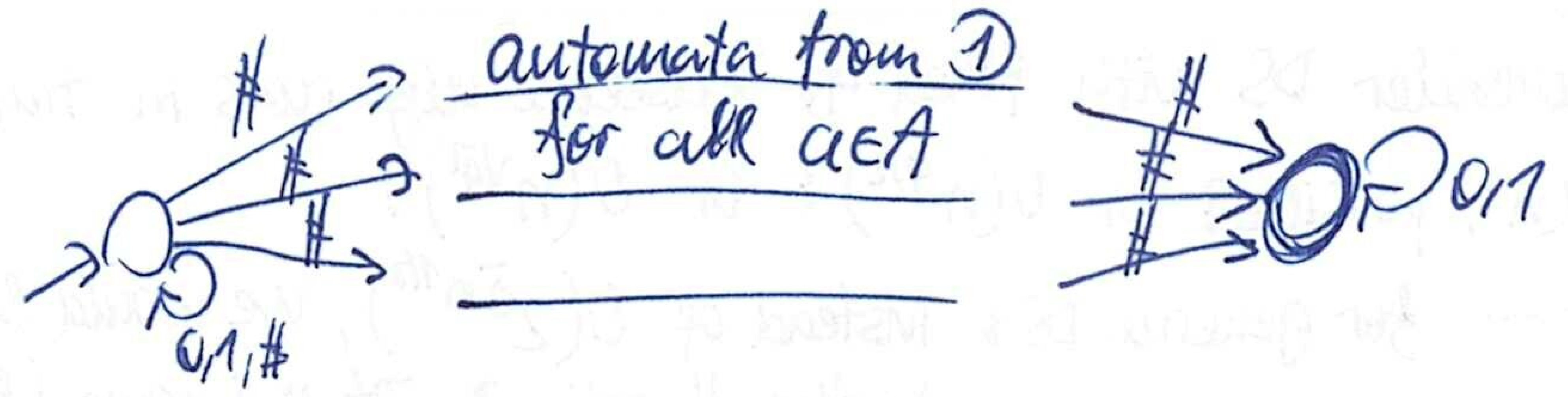
Proof: Will show reduction OV → NFAA running in time $O(nd)$, producing $|M| \in O(nd)$, $|\alpha| \in O(nd)$.
If NFAA can be solved in $O((|M| \cdot |\alpha|)^{1-\varepsilon})$ time for some $\varepsilon > 0$,
then OV can be solved in $O((n^2 d^2)^{1-\varepsilon}) = O(n^{2-2\varepsilon} \cdot d^{2-2\varepsilon})$ time, contradicting OVH.
Now the reduction:

① given $a \in A$, construct NFA which accepts $b \Leftrightarrow \langle a, b \rangle = 0$.

② given a, construct NFA accepting $\#b^1\#b^2\#\ldots\#b^n\#$ $\Leftrightarrow \exists i : \langle a, b^i \rangle = 0$



③ Add a choice of $a \in A$:



Surprisingly, fine-grained bounds are connected with the "big world" of P vs. NP.

Exponential Time Hypothesis (ETH): $\exists \varepsilon > 0$ s.t. 3-SAT can't be solved in $O(2^{\varepsilon N})$ time.

└ justification: baseline alg. is $O(2^N \cdot poly(M,N))$-time
state-of-the-art alg. is $O(\underbrace{1.3280^N}_{\text{improvements don't seem to converge towards 1}} \cdot poly(M,N))$-time.

for k-SAT: $N :=$ # variables, $M :=$ # clauses

Obviously, ETH implies P≠NP.

Strong ETH (SETH): $\forall \varepsilon > 0\ \exists k$ s.t. k-SAT cannot be solved in time $O(2^{\varepsilon N})$.
└ justification: state-of-the-art kSAT algs are slower for higher k, converging towards 2^N.

dependence on M is not important as $M \leq \binom{N}{k} \cdot 2^k$, which is polynomial for fixed k

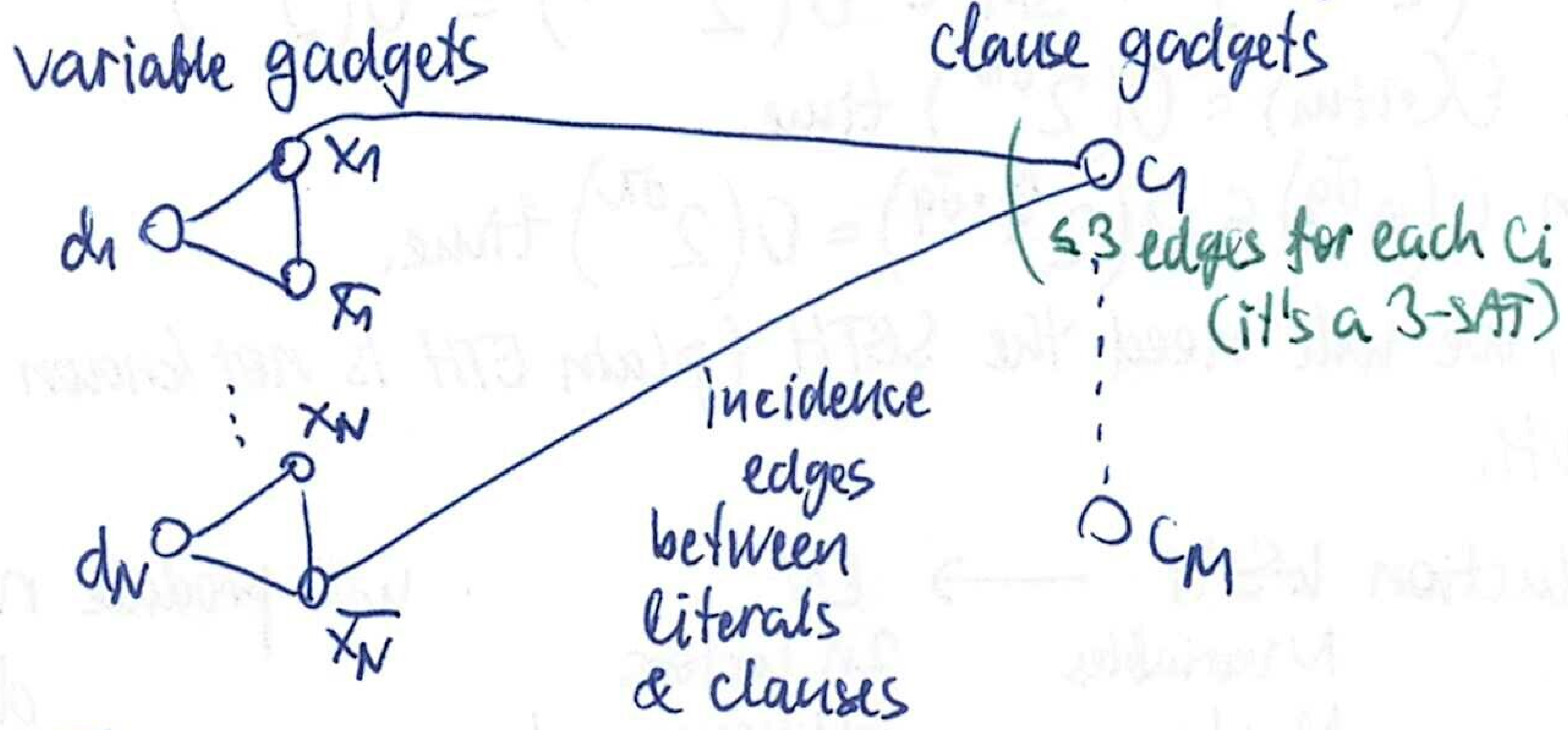
It's known that SETH ⇒ ETH.

Dominating Set problem (DS)

Input: undirected graph with n vertices, $q > 0$
Question: is there a dominating set D of size q?
← for $G = (V,E)$: $D \subseteq V$, $\forall u \in V\ \exists v \in D$: $\underbrace{(u = v \text{ or } \{u,v\} \in E)}_{u \text{ dominated by } v}$

Theorem: DS is NP-complete.

Proof: DS∈NP is trivial, will show NP-hardness by reducing 3-SAT to DS.



Will set $q = N$. This implies that a dom. set must use exactly 1 vertex from each var. gadget.
Formula satisfiable ⇒ choose dom. set according to assignment, check that all clause vertices are dominated.

Dom. set exists ⇒ choose assignment according to literals used in D ($d_i \in D$ ⇒ choose x_i arbitrarily), check that the formula is satisfied.

Theorem: ETH ⇒ $\exists \delta > 0$ s.t. DS cannot be solved in time $O(2^{\delta \cdot n^{1/3}})$.

Proof: Finer analysis of the same reduction.
Graph has n vertices, m edges for $n = 3N + M$, $m \leq 3n$] we have $M \leq \binom{N}{3} \cdot 2^3 \leq 2N^3$, so $n \leq 3N^3$ for N large enough, $m \in O(n)$
Reduction runs in $O(n+m) = O(N^3)$ time.
If DS can be solved in $O(2^{\delta \cdot n^{1/3}})$ time, then 3SAT can in $O(2^{3^{1/3} \cdot \delta \cdot N})$.
For δ small enough, this contradicts the ETH.

Now consider DS with fixed q. Baseline alg. runs in time $O(\binom{n}{q} \cdot qn) \leq O(n^{q+1})$.
Is $O(n^q)$ possible? Or $O(n^{q/2})$? Or $O(n^{\sqrt{q}})$?

for general DS: Instead of $O(2^{\delta n^{1/3}})$, we would like $O(2^{\delta n})$ ... how to get rid of N^3 in the #vertices? It is known (but we won't proved it here) that 3-SAT is hard even for sparse formulas ($M \in O(N)$).

<u>Theorem</u>: If ETH holds, then $\exists \delta > 0\ \forall^* q$: DS cannot be solved in $O(n^{\delta q})$ time.

<u>Proof</u>: We will show that a $O(n^{\delta q})$-time alg. for DS with $\boxed{q \geq \frac{2}{\delta}}$ (*)
implies a $O(2^{\delta N})$ alg. for 3-SAT. So for δ small enough, this would contradict the ETH.
Modify the previous reduction of 3-SAT to DS:

- ~~divide~~ partition variables to q groups per N/q variables
- variable gadgets: for each group, create vertices for all partial assignments setting variables in the group $\rightarrow 2^{N/q}$ vertices
  + add an extra d_i vertex
  edges form a clique
- clause gadgets: for each clause, add vertex c_i connected to all partial assignments which satisfy this clause

Again, a DS of size q selects 1 vertex from each var. gadget.
This either selects one partial assignment to vars in the group or d_i = "pick any".
Graph size: $n = q(2^{N/q}+1) + M \in O(2^{N/q})$ for fixed q

$$m = (2^{N/q}+1)^2 + 3M \in O(2^{2N/q}) \overset{\text{because of (*)}}{\subseteq} O(2^{\delta N})$$

Reduction takes $O(n+m) = O(2^{\delta N})$ time.
SAT is solved in $O(n^{\delta q}) \leq O(2^{\frac{N}{q} \cdot \delta q}) = O(2^{\delta N})$ time.

- For hardness of OV, we will need the SETH (plain ETH is not known to suffice)

<u>Theorem</u>: SETH $\Rightarrow$ OVH.

<u>Proof</u>: Will show a reduction k-SAT $\longrightarrow$ OV
N variables → 2n vectors
M clauses → dimension d

will produce $n = 2^{N/2}$
$d = M$
in time $O(nd)$, assuming k fixed.

So a $O(n^{2-\varepsilon} d)$ time alg. for OV implies a $O(2^{\frac{2-\varepsilon}{2} N} \cdot M)$-time alg. for k-SAT, contradicting SETH for k large enough.

Reduction: Split variables to 2 groups X, Y of size $N/2$.
Construct A using X:

- vectors correspond to partial assignments to X] $2^{N/2}$ vectors
- coordinates correspond to clauses] M coordinates
- "0" means that the clause is satisfied by the partial assgmt.

Similarly, construct B using Y.

$\langle a, b \rangle = 0 \Leftrightarrow$ all clauses are satisfied by a union of the two part. assgmts.

Problem: Longest Common Subsequence (LCS) — define $L(\alpha,\beta) :=$ max. length of a common sub-sequence of α,β

↑ Obtained by deleting elements while preserving order (i.e., not a subword)

Input: Strings $\alpha,\beta \in \Sigma^*$, $|\alpha|, |\beta| \leq n$; $c > 0$

↑ WLOG it's $= n$ (we can pad the shorter of the two strings)

Output: Is $L(\alpha,\beta) > c$?

Theorem: LCS can be solved in time $O(n^2)$ independent of $|\Sigma|$.

Proof: Let $A[i,j] := L(\alpha[:i], \beta[:j])$.

We have $A[0,j] = A[i,0] = 0$,

$$A[i,j] = \min \begin{cases} A[i-1,j] \\ A[i,j-1] \\ A[i-1,j-1]+1 \end{cases} \text{only if } \alpha[i-1] = \beta[j-1]$$

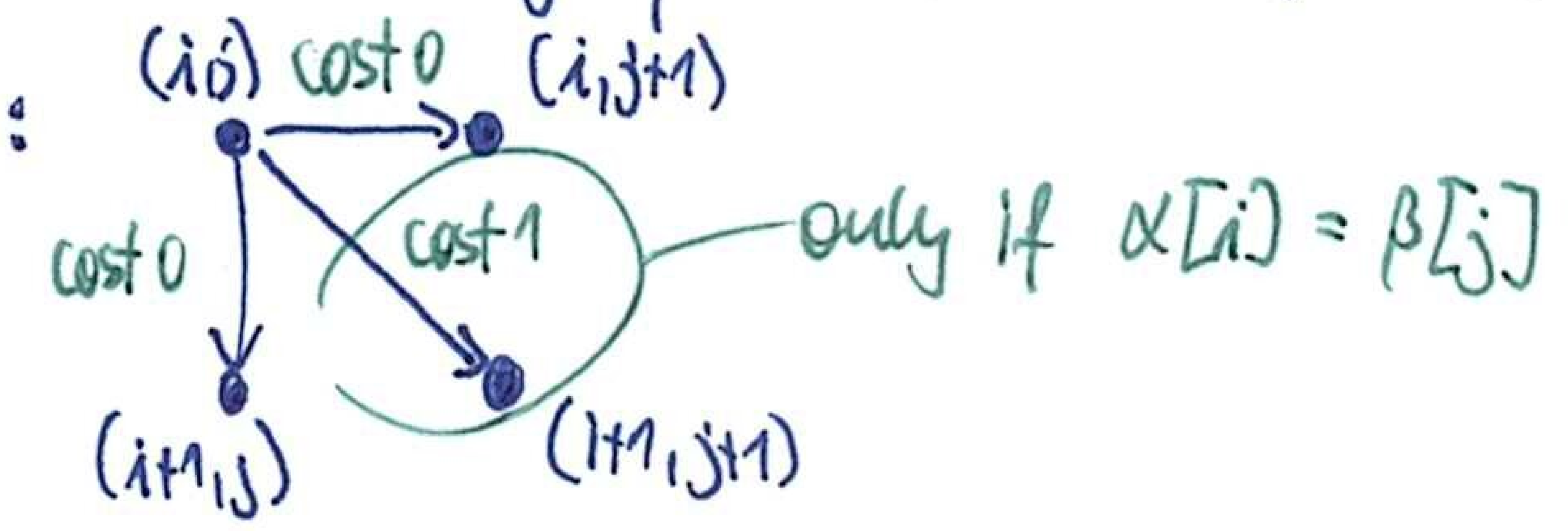
for $i,j > 0$

Using this, we can fill in the table of $A[i,j]$'s row by row in time $O(n^2)$.

Then $A[n,n] = L(\alpha,\beta)$.

Alternative proof: Define a directed graph with $V = \{0 - |\alpha|\} \times \{0 - |\beta|\}$, $|V| \in O(n^2)$

Edges:



$|E| \leq 3|V| \in O(n^2)$

Path from $(0,0)$ to $(|\alpha|, |\beta|)$ of cost c corresponds to a common subseq. of length c.

↳ LCS = cost of the longest path ... but since the graph is acyclic, this can be computed using the same recurrence as in the previous proof. ⇒ the same $O(n^2)$-time alg.

Theorem: Assuming OVH, for no $\varepsilon > 0$ there is a $O(n^{2-\varepsilon})$-time alg. for LCS.

↑ even for just the binary alphabet

Proof: Omitted, see the lecture notes by Karl Bringmann.

That's all, thanks for your attention! ☺