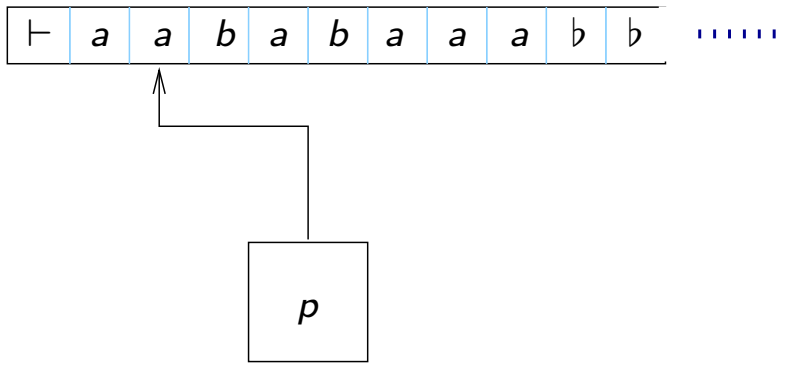
Topics to be covered in Chapters 8 - 9 include:

- Turing machines,
- Recursive and recursively enumerable languages,
- Equivalent models of TMs,
- The halting problem,
- Reduction and more undecidability results,
- Rice's theorem,
- The Post correspondence problem,
- Basic recursive function theory, ...

How a Turing machine works



- Finite control
- Tape infinite to the right
- Each step: In current state p, read current symbol under the tape head, say a: Change state to q, replace current symbol by b, and move head left or right.

$$(p, a) \to (q, b, L/R).$$

How a Turing machine works

- Special designated accept state t and reject state r. These states are assumed to be "sink" states.
- TM accepts its input by entering state t.
- TM rejects its input by entering state r.
- TM never falls off the left end of the tape (i.e it always moves right on seeing '$\vdash$').

Turning machines more formally

A Turing machine is a structure of the form

$$M = (Q, A, \Gamma, s, \delta, \vdash, \flat, t, r)$$

where

- Q is a finite set of states,
- A is the input alphabet,
- Γ is the tape alphabet which contains A,
- $s \in Q$ is the start state,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the (deterministic) transition relation,
- $\vdash \in \Gamma$ is the left-end marker.
- $\flat \in \Gamma$ is the blank tape symbol.
- $t \in Q$ is the accept state.
- $r \in Q$ is the reject state.

Configurations, runs, etc. of a Turing machine

- A configuration of M is of the form $(p, y\flat^\omega, n) \in Q \times \Gamma^\omega \times \mathbb{N}$, which says "$M$ is in state p, with "non-blank" tape contents y, and read head positioned at the n-th cell of the tape.
- Initial configuration of M on input w is $(s, \vdash w\flat^\omega, 0)$.
- 1-step transition of M: If $(p, a) \to (q, b, L)$ is a transition in δ, and $z(n) = a$: then

$$(p, z, n) \overset{1}{\Rightarrow} (q, s_b^n(z), n-1).$$

- Similarly, if $(p, a) \to (q, b, R)$ is a transition in δ, and $z(n) = a$: then

$$(p, z, n) \overset{1}{\Rightarrow} (q, s_b^n(z), n+1).$$

- M accepts w if $(s, \vdash w\flat^\omega, 0) \overset{*}{\Rightarrow} (t, z, i)$, for some z and i.
- M rejects w if $(s, \vdash w\flat^\omega, 0) \overset{*}{\Rightarrow} (r, z, i)$, for some z and i.

Language accepted by a Turing machine

- The Turing machine M is said to halt on an input if it eventually gets into state t or r on the input.
- Note that M may not get into either state t or r on a particular input w. In that case we say M loops on w.
- The language accepted by M is denoted $L(M)$ and is the set of strings accepted by M.
- A language $L \subseteq A^*$ is called recursively enumerable if it is accepted by some Turing machine M.
- A language $L \subseteq A^*$ is called recursive if it is accepted by some Turing machine M which halts on all inputs.

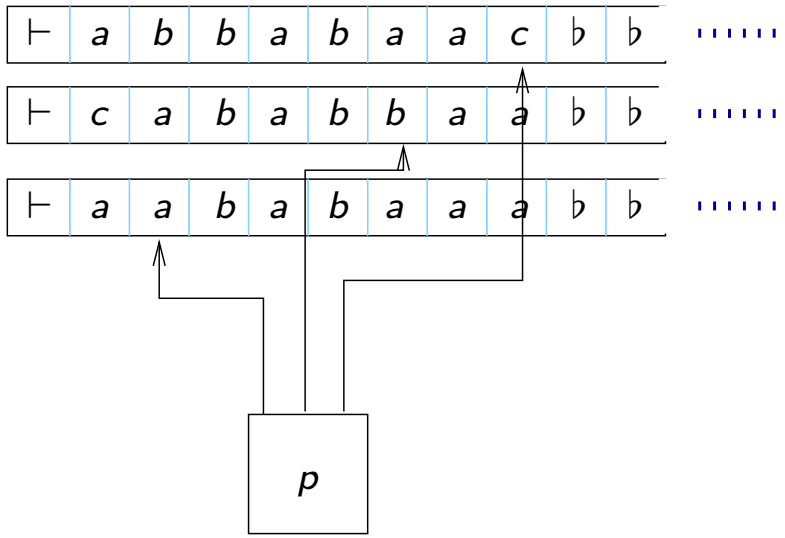
Computability and languages

- Notion of a function $f : \mathbb{N} \to \mathbb{N}$ being "computable" (informally if we can give a "finite recipe" or "algorithm" to compute $f(n)$ for a given n.)
- We say f is computable if we have a TM M that given $\vdash 0^n$ as input, outputs $0^{f(n)}$ on its tape, and halts.
- View f as a language

$$L_f = \{(n, f(n)) \mid n \in \mathbb{N}\}.$$

- Then f is computable iff L_f is recursive.

TM with multiple tapes



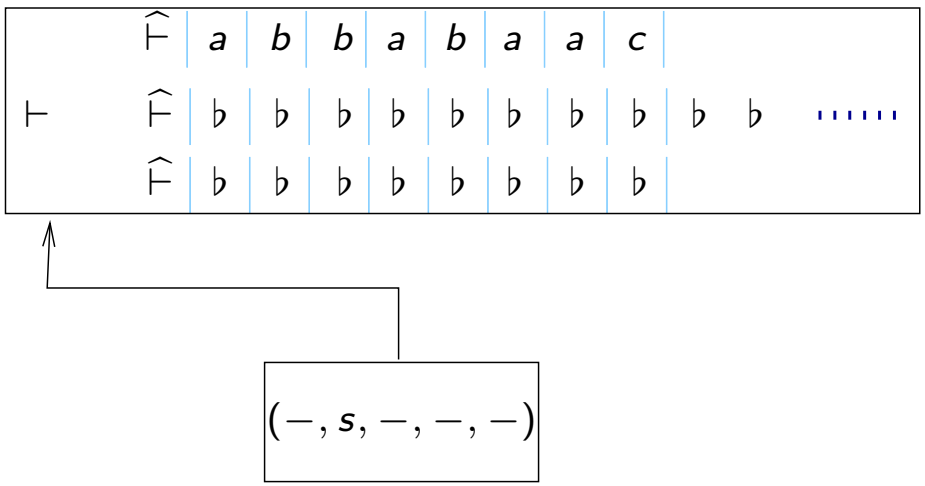
- Finite control
- Multiple tapes (say 3), each with its own read head.
- Each step: In current state p, read current symbols under the tape heads, say a, b, c: Change state to q, replace current symbols by a', b', c', and move each head left or right.

$$(p, a, b, c) \to (q, a', b', c', L/R, L/R, L/R).$$

How a TM can simulate a multi-tape TM

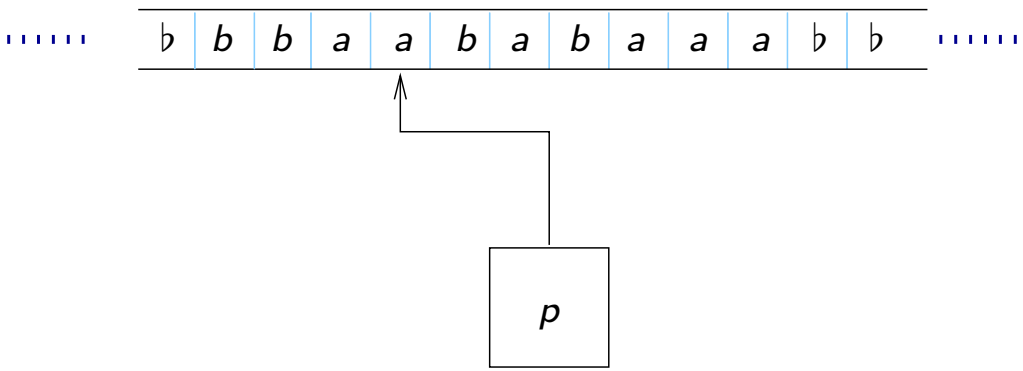
Let M be given multi-tape TM. Define a TM M' that:

- Given input w on its tape, first changes it to configuration:



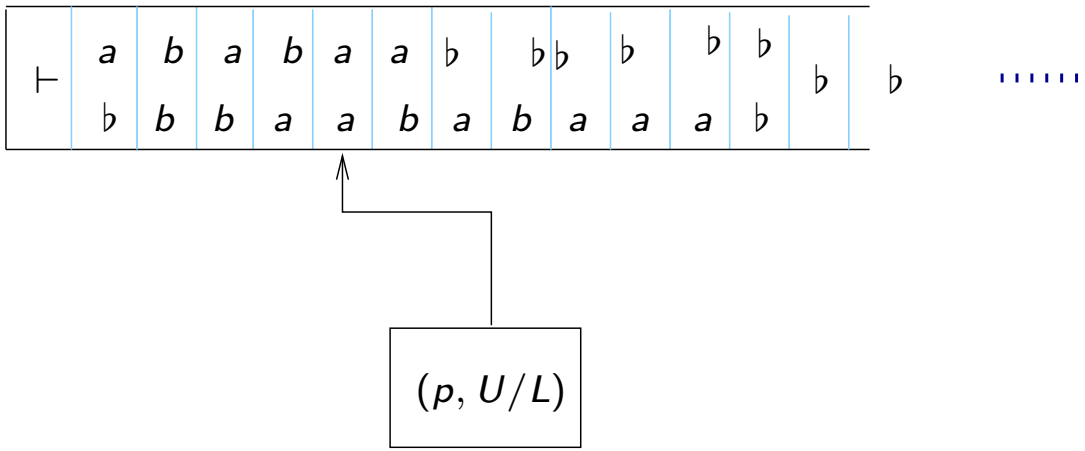
- Simulates a single step $(p, a, b, c) \to (q, a', b', c', L/R, L/R, L/R)$ of M by:
  - Scan top track to find $\widehat{a}$, and remember it in its finite control
  - Similarly scan tracks 2 and 3 and rembember $\widehat{b}$ and $\widehat{c}$ in its finite control.
  - Now change to state $(-, q, a', b', c')$.
  - Scan track 1 to find $\widehat{a}$, replace it by a', move head L/R.
  - Similarly for tracks 2 and 3.

TM with two-way infinite tape



- Finite control
- Single two-way infinite tape.

Simulating a two-way infinite tape



- To simulate, imagine the tape is folded to the right at some point, and simulate with a tape alphabet $\Gamma \times \Gamma \cup \{\vdash, \flat\}$.

The Church-Turing Thesis

Church-Turing Thesis

The definition of computability based on Turing machines, captures the "right" notion of computability.

Turing computability coincides with several other notions of computability proposed based on different models, in the 1930's:

- Post systems (Emil Post)
- μ-recursive functions (Gödel, Herbrand)
- λ-calculus (Church, Kleene)
- Combinatory logic (Curry, Schönfinkel)

Non-deterministic TM

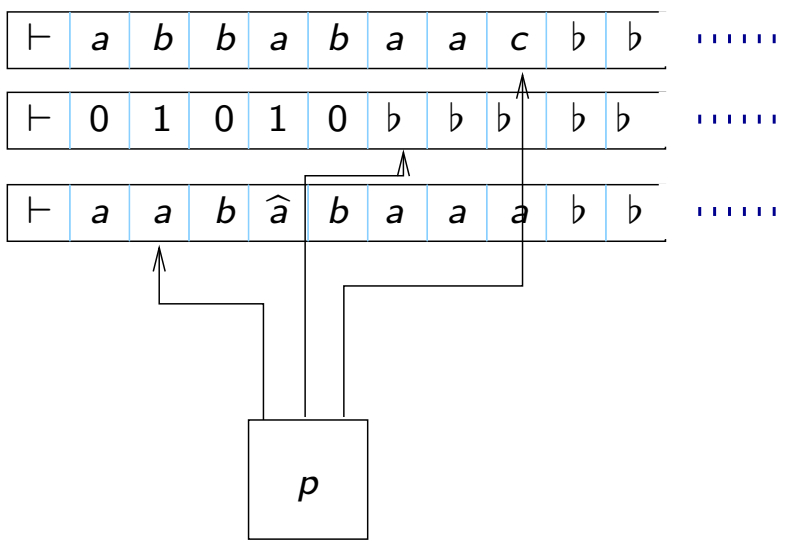
- Similar to TM already defined, except that moves can be non-deterministic.
- M accepts an input w if

$$(s, \vdash w\flat^{\omega}, 0) \stackrel{*}{\Rightarrow} (t, z, i),$$

for some z and i.

Simulating a non-deterministic TM by a det TM

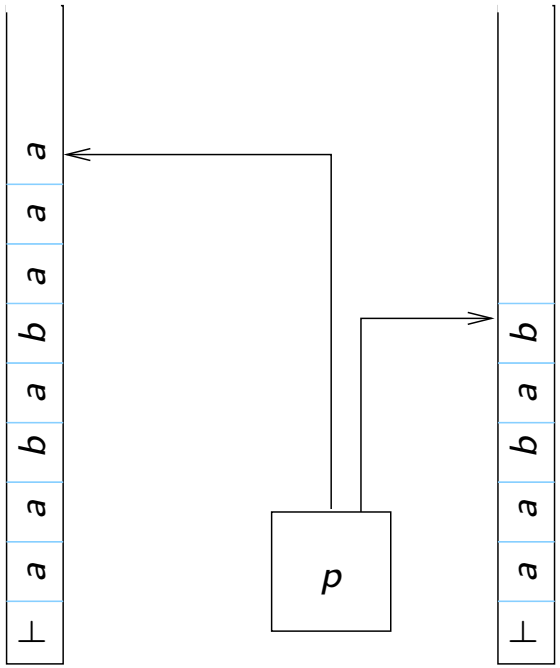
- Let M be a given non-det TM.
- Define a deterministic TM M' that accepts the same language as M.
- M' uses 3 tapes: for given input, guessed binary string, work tape to simulate run of M.



Simulating a non-deterministic TM by a det TM

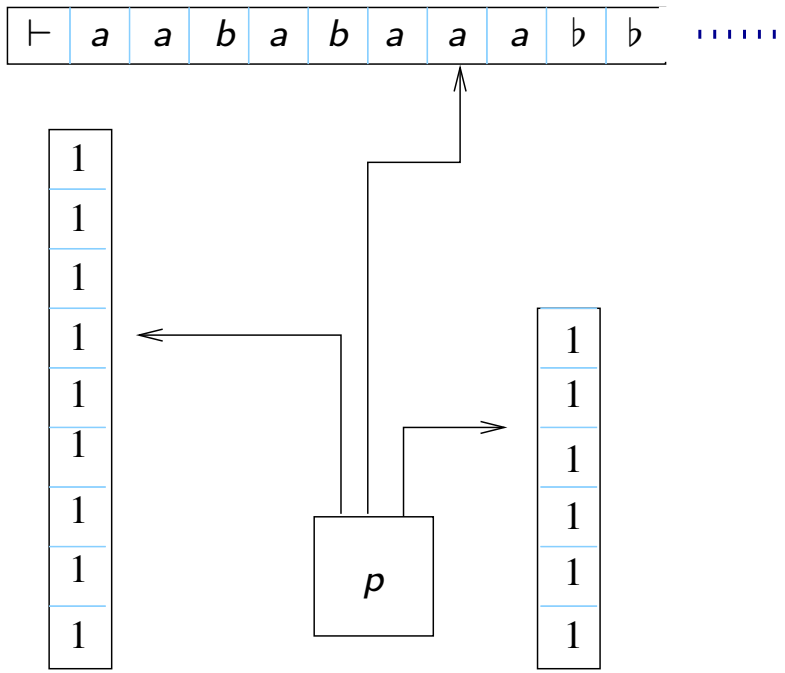
- Deterministic TM M' searches the tree representing the run of M on w, by doing a BFS on the tree.
- Enumerates binary strings in lexicographic order on tape 2.
- Simulate M on input along the path in tape 2, on tape 3.
- Accept (Enter state t) if M' enters t in the simulation.
- Guaranteed to capture all paths and accept (reject) if the given non-deterministic TM accepts (rejects).

2-stack PDA



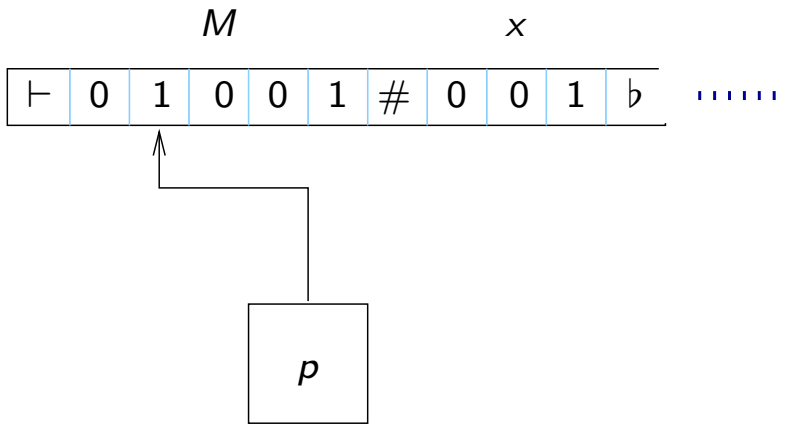
- The PDA can read the 2 top of stacks, and make a push/pop move independently on each stack.
- A 2-tape TM can easily simulate such a PDA.
- Conversely, a 2-stack PDA can simulate a TM.

Counter Machines



- A 2-counter machine can read the values of its 2 counters, say c and d, and then make a move (increment/decrement c/d, overwrite current symbol, move L/R, and change state).
- A TM can easily simulate a 2-counter machine.
- Conversely, a 4-counter machine can simulate a 2-stack PDA (and hence a TM).

Universal Turing machine



- We can construct a TM U that takes the encoding of a TM M and its input x, and "interprets" M on the input x.
- U accepts if M accepts x, rejects if M rejects x, and loops if M loops on x.

Encoding a TM as a $\{0,1\}$-string

$$0^n10^m10^k10^s10^t10^r10^u10^v\ 1\ 0^p10^a10^q10^b10\ 1\ 0^{p'}10^{a'}10^{q'}10^{b'}100\ \cdots 1\ 0^{p''}10^{a''}10^{q''}10^{b''}10.$$

represents a TM M with

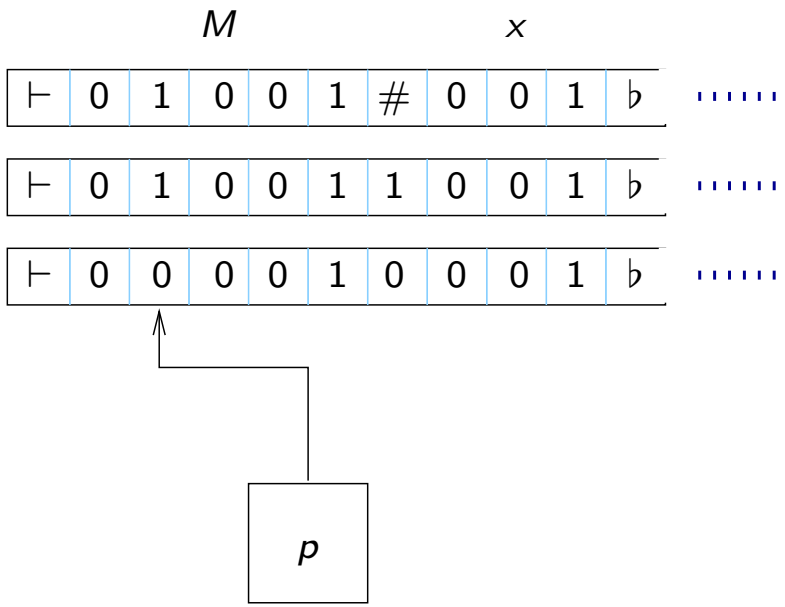
- states $\{1,2,\ldots,n\}$.
- Tape alphabet $\{1,2,\ldots,m\}$.
- Input alphabet $\{1,2,\ldots,k\}$ (with $k<m$).
- Start state $s\in\{1,2,\ldots,n\}$.
- Accept state $t\in\{1,2,\ldots,n\}$.
- Reject state $r\in\{1,2,\ldots,n\}$.
- Left-end marker symbol $u\in\{k+1,\ldots,m\}$.
- Blank symbol $v\in\{k+1,\ldots,m\}$.
- Each string $0^p10^a10^q10^b10$ represents the transition $(p,a)\to(q,b,L)$.

Example encoding of TM and its input

Input is encoded as $0^a10^b10^c$ etc.
What does the following TM do on input 001010?

000100001001010010001000100100 1 01000101000100 1 0100100100100 1 010101010.

How the universal Turing machine works



- Use 3 tapes: for input $M\#x$, for current configuration, and for current state and position of head.
- Repeat:
  - Execute the transition of M applicable in the current config.
- Accept if M gets into t state, Reject if M gets into r state.

Halting Problem for Turing machines

- Fix an encoding *enc* of TM's as above.
- Define the language

$$\mathrm{HP} = \{enc(M)\#enc(x) \mid M \text{ halts on } x\}.$$

Undecidability of HP

Theorem (Turing)

The language HP *is not recursive.*

Proving undecidability of HP

Assume that we have a Turing machine M which decides HP. Then we can compute the entries of the table below:

	ϵ	0	1	00	01	10	11	000	001	010	011	111	$\cdots$
M_ϵ	L	H	L	L	L	H	H	L	L	L	L	L	$\cdots$
M_0	L	L	L	L	L	L	L	L	L	L	L	L	$\cdots$
M_1	H	H	L	H	L	H	H	L	L	H	L	H	$\cdots$
M_{00}	L	L	L	L	L	L	L	L	L	L	L	L	$\cdots$
M_{01}	L	H	L	L	L	H	H	L	L	L	L	L	$\cdots$
M_{10}	H	H	L	H	L	H	H	L	L	H	L	H	$\cdots$
M_{11}	L	H	L	L	L	H	H	L	L	L	L	L	$\cdots$
M_{000}	L	L	L	L	L	L	H	L	L	L	H	L	$\cdots$
$\vdots$													

- For each $x \in \{0,1\}^*$ let M_x denote the TM
  - M, if x is the encoding of TM M with input alphabet $0, 1$.
  - M_{loop} otherwise, where M_{loop} is a one-state Turing machine that loops on all its inputs.

A TM N that behaves differently from all TM's

Let us assume we have a TM M that decides HP. Then we can define a TM N as follows: Given input $x \in \{0,1\}^*$, it

- runs as M on $x\#x$.
- If M accepts (i.e. M_x halts on x), goes to a new "looping" state l and loops there.
- If M rejects (i.e. M_x loops on x), goes to the accept state t'.

N essentially "complements the diagonal" of the table: Given input $x \in \{0,1\}^*$ it halts iff M_x loops on x.
Consider $y = enc(N)$. Then y cannot occur as any row of the table since the behaviour of N differs from all rows in the table. This is a contradiction.

Complement of HP is not r.e.

Fact 1: If L and $\overline{L}$ are both r.e. then L (and $\overline{L}$) must be recursive.

- Let M accept L and M' accept $\overline{L}$.
- We can construct a total TM that simulates M and M' on given input, one step at a time.
- Accept if M accepts, Reject if M' accepts.

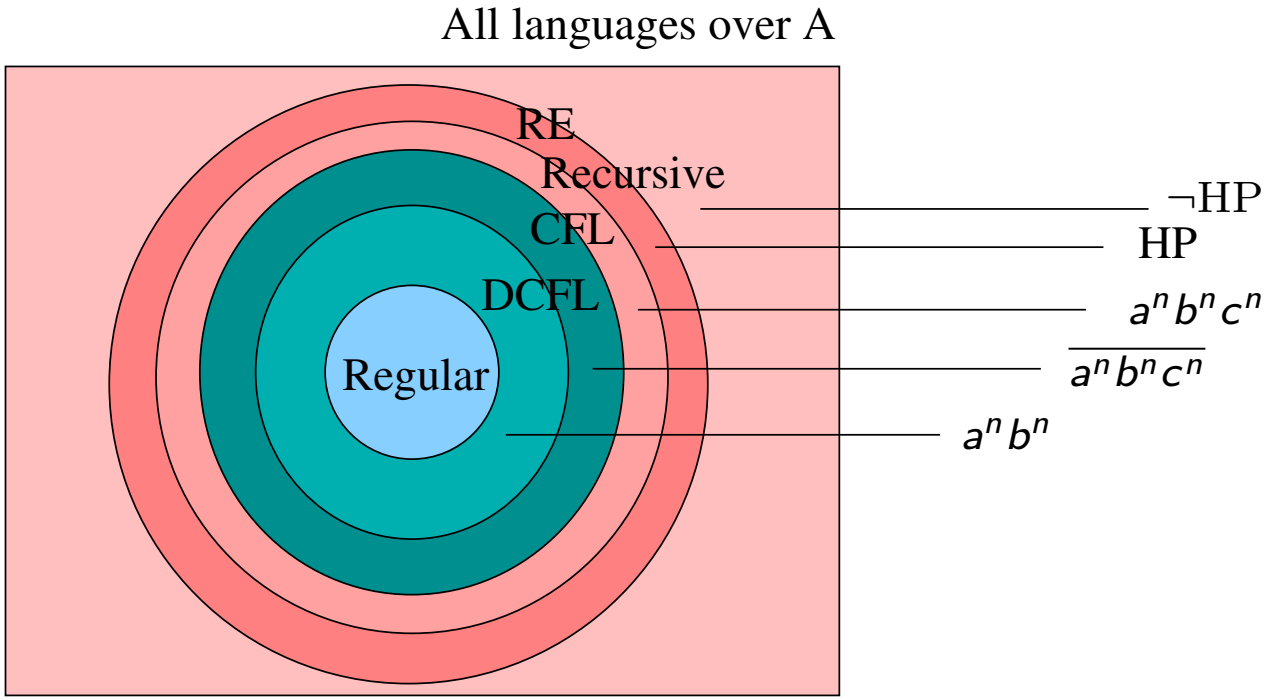
Fact 2: HP is recursively enumerable.

- Just run the universal TM U on input $M\#x$; accept iff U halts (i.e. M accepts or rejects x).

Corollary

The language $\neg\mathrm{HP}$ *is not even recursively enumerable.*

Where HP lies



More decidable/undecidable problems

Problem (a)

Is it decidable whether a given Turing machine has at least 481 states? Assume that the TM is given using the encoding below:

$$0^n 1 0^m 1 0^k 1 0^s 1 0^t 1 0^r 1 0^u 1 0^v \; 1 \; 0^p 1 0^a 1 0^q 1 0^b 1 0 \; 1 \; 0^{p'} 1 0^{a'} 1 0^{q'} 1 0^{b'} 1 0 0 \; \cdots 1 \; 0^{p''} 1 0^{a''} 1 0^{q''} 1 0^{b''} 1 0.$$

$$0001000010010100100010001000 0 \; 1 \; 01000101000100 \; 1 \; 0100100100100 \; 1 \; 010101010.$$

Yes, it is.

We can give a TM N which given $enc(M)$

- Counts the number of states in M upto 481.
- Accepts if it reaches 481, rejects otherwise.

More decidable/undecidable problems

Problem (b)

Is it decidable whether a given Turing machine takes more than 481 steps on input ϵ without halting?

000100001001010010001000100010000 1 01000101000100 1 0100100100100 1 010101010.

Yes, it is.

We can give a TM N which given $enc(M)$

- Uses 4 tapes: On the 4th tape it writes 481 0's.
- Uses the first 3 tapes to simulate M on input ϵ, like the universal TM U.
- Blanks out a 0 from 4th tape for each 1-step simulation done by U.
- Rejects if M halts before all 0's are blanked out on 4th tape, accepts otherwise.

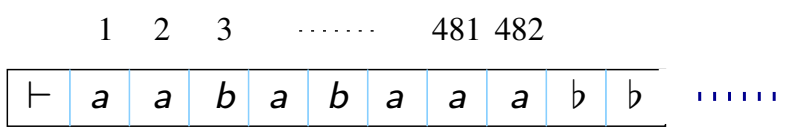
More decidable/undecidable problems

Problem (c)

Is it decidable whether a given Turing machine takes more than 481 steps on *some* input without halting?

000100001001010010001000100100000 1 01000101000100 1 0100100100100 1 010101010.

Yes, it is.
Check if M runs for more than 481 steps on each input x of length upto 481. If so accept, else reject.



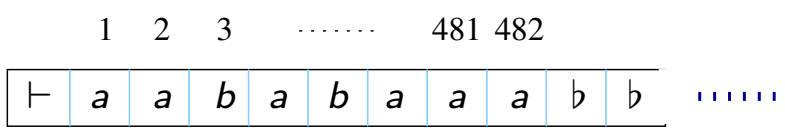
More decidable/undecidable problems

Problem (d)

Is it decidable whether a given Turing machine takes more than 481 steps on *all* inputs without halting?

000100001001010010001000100100000 1 01000101000100 1 0100100100100 1 010101010.

Yes, it is.
Check if M runs for more than 481 steps on each input x of length upto 481. If so accept, else reject.



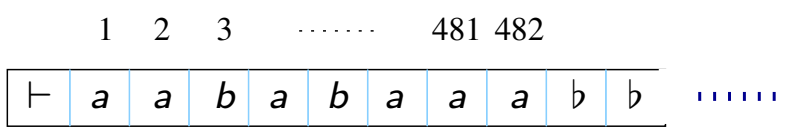
More decidable/undecidable problems

Problem (e)

Is it decidable whether a given Turing machine moves its head more than 481 cells away from the left-end marker, on input ϵ?

000100001001010010001000100000 1 01000101000100 1 0100100100100 1 010101010.

Yes, it is.
Simulate M on ϵ for upto $m^{481} \cdot 482 \cdot k$ steps. If M visits the 482nd cell, accept, else reject.



More decidable/undecidable problems

Problem (f)

Is it decidable whether a given Turing machine accepts the null-string ϵ?

No.
If it were decidable, say by a TM N, then we could use N to decide HP as follows: Define a new machine N' which given input $M\#x$, outputs the description of a machine M' which:

- erases its input
- writes x on its input tape
- Behaves like M on x
- Accepts if M halts on x.

N' then calls N with input M'.
N accepts M' iff M' accepts ϵ iff M halts on x.

Turing machine M' for Problem (f)

$$L(M') = \begin{cases} A^* & \text{if } M \text{ halts on } x \\ \emptyset & \text{if } M \text{ does not halt on } x. \end{cases}$$

More decidable/undecidable problems

Problem (g)

Is it decidable whether a given Turing machine accepts any string at all? That is, is $L(M) \neq \emptyset$?

More decidable/undecidable problems

Problem (h)

Is it decidable whether a given Turing machine accepts all strings? That is, is $L(M) = A^*$?

More decidable/undecidable problems

Problem (i)

Is it decidable whether a given Turing machine accepts a finite set?

More decidable/undecidable problems

Problem (j)

Is it decidable whether a given Turing machine accepts a regular set?

Given M and x, build a new machine M' that behaves as follows:

1. Saves its input y on tape 2.
2. writes x on tape 1.
3. runs as M on x.
4. if M gets into a halting state, then
   - M' takes back control,
   - Runs as M_R on y,
   - (Here M_R is any TM that accepts a non-regular language R, say $R = \{a^n b^n \mid n \geq 0\}$).
   - M' accepts iff M_R accepts.

Turing machine M' for Problem (j)

$$L(M') = \begin{cases} R & \text{if } M \text{ halts on } x \\ \emptyset & \text{if } M \text{ does not halt on } x. \end{cases}$$

More decidable/undecidable problems

Problem (k)

Is it decidable whether a given Turing machine accepts a CFL?

More decidable/undecidable problems

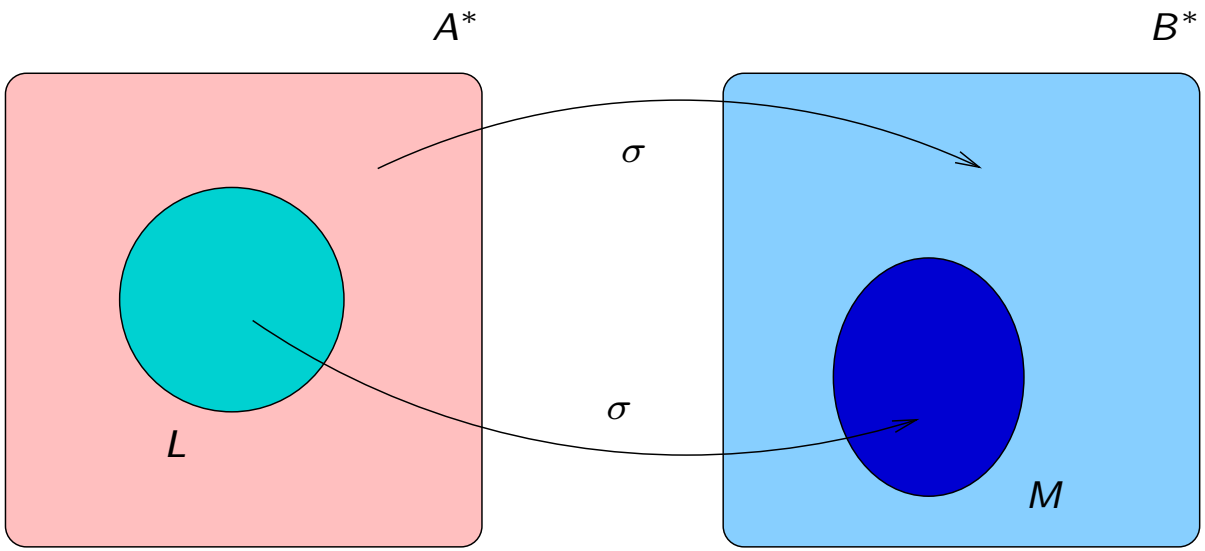
Problem (I)

Is it decidable whether a given Turing machine accepts a recursive set?

Reductions

Let $L \subseteq A^*$ and $M \subseteq B^*$ be two languages. We say L reduces to M and write $L \leq M$ iff there exists a computable map $\sigma : A^* \to B^*$ such that

$$w \in L \text{ iff } \sigma(w) \in M.$$



Reductions and recursive/re-ness

Theorem

If $L \leq M$ then:

1. *If M is r.e. then so is L.*
2. *If M is recursive then so is L.*

Or to put it differently:

Theorem

If $L \leq M$ then:

1. *If L is not r.e. then neither is M.*
2. *If L is not recursive then neither is M.*

Examples of reductions

Let L be the language $\{M \mid M \text{ accepts } \epsilon\}$. Then

$$\mathrm{HP} \leq L.$$

- Describe a computable map σ which witnesses the reduction.

Hence, since HP is undecidable (i.e. not recursive) so is L.

Examples of reductions

Let L be the language $\{M \mid M \text{ accepts a regular language}\}$. Then

$$\neg\mathrm{HP} \leq L.$$

- Describe a computable map σ which witnesses the reduction.
- Hence, since $\neg\mathrm{HP}$ is undecidable (i.e. not recursive) so is L.
- In fact, since $\neg\mathrm{HP}$ is not r.e., we can say that L is not r.e..

Rice's theorem

Theorem (Rice)

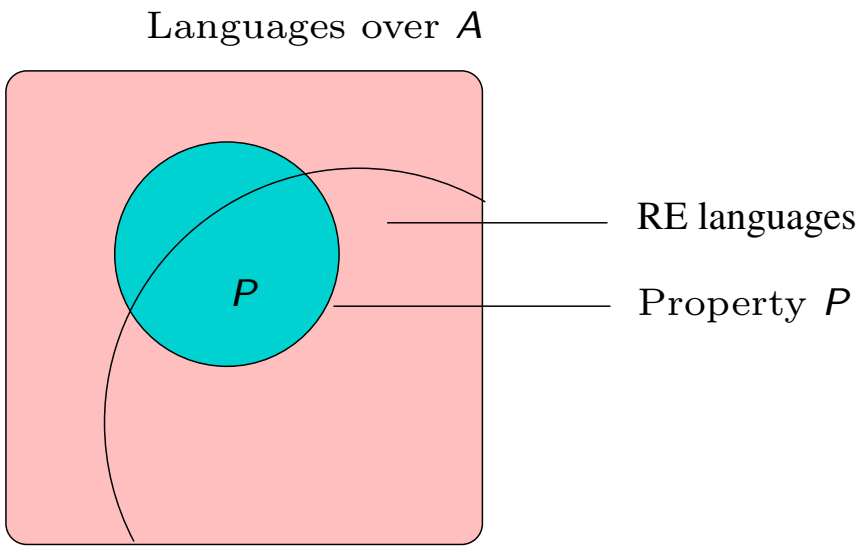
Any non-trivial property of r.e. languages is undecidable.

Theorem (Rice)

Any non-monotone property of r.e. languages is not even recursively enumerable.

Properties of languages

A property P of languages over an alphabet A is a subset of languages over A.



Non-trivial and montone properties

- A property P is a non-trivial property of r.e. languages, if there is at least one r.e. language L satisfying P, and another L' not satisfying P.
  - E.g. "is empty" is non-trivial
  - "is not accepted by a TM" is trivial.
- A property P of languages is monotone (w.r.t r.e. languages) if for all r.e. sets A and B, whenever $A \subseteq B$ and $P(A)$, we have $P(B)$.
- IOW, P is monotone if whenever a set has the property, then all supersets of that set have it as well.
  - "is infinite" is monotone,
  - "$L(M)$ is finite" is not monotone.

Rice's theorems

For a property P, we define

$$L_P = \{M \mid L(M) \text{ satisfies } P\}.$$

Theorem (Rice)

Any non-trivial property of r.e. languages is undecidable. That is, if P is a non-trivial property of r.e. languages, then the language L_P is not recursive.

Theorem (Rice)

Any non-monotone property of r.e. languages is not even recursively enumerable. That is, if P is a non-monotone property of r.e. languages, then the language L_P is not even recursively enumerable.

Proof of Rice's Theorem 1

- Let P be a non-trivial property of r.e. languages. Then there are TM's K and T such $L(K)$ satisfies P and $L(T)$ does not satisfy P.
- We show that $L_P = \{M \mid L(M) \text{ satisfies } P\}$ is not recursive.
- Case 1: If $\emptyset$ satisfies P. We reduce HP to L_P.
- Given $M\#x$, construct a machine $M' = \sigma(M\#x)$ that on input y
  - saves y on a separate track
  - writes x on its tape
  - runs as M on input x
  - if M halts on x, M' runs as K on y and accepts iff K accepts.

$$L(M') = \begin{cases} L(K) & \text{if } M \text{ halts on } x \\ \emptyset & \text{if } M \text{ does not halt on } x. \end{cases}$$

Proof of Rice's Theorem 2

- Let P be a non-monotone property of r.e. sets.
- Then there are TM's K and T such that $L(K) \subseteq L(T)$ and $L(K)$ satisifes P but $L(T)$ does not.
- We show $\neg\mathrm{HP} \leq L_P$.
- Given $M\#x$ output the description of M' that
  - Given input y on Tape 1.
  - Copies y on Tape 2, writes x on Tape 3
  - Run (in an interleaved fashion) as M on x, K on y, and T on y.
  - accept iff either
    - K accepts y, or,
    - M halts on x and T accepts y.

Proof of Rice's Theorem 2

Notice that:

$$L(M') = \begin{cases} L(K) & \text{if } M \text{ does not halt on } x \\ L(T) & \text{if } M \text{ halts on } x. \end{cases}$$