Summary last week

- function $f : \mathbb{N} \to \mathbb{N}$ computable if exists effective procedure computing $f(x)$ on x
- effective procedure for f if exists TM M that leaves output $f(x)$ on tape on input x;
- equivalently defined via other models of computation: μ-recursion, λ-calculus,...
- language L recursive(ly enumerable) if exists (total) TM M accepting L ($L = L(M)$)
- property P (semi-)decidable if $\{x \mid P(x)\}$ is recursive(ly enumerable)

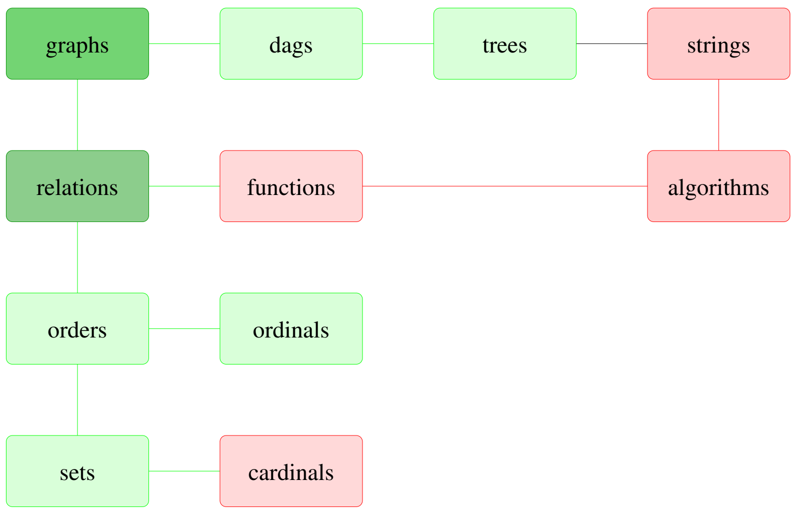
Summary last week

- function $f : \mathbb{N} \to \mathbb{N}$ computable if exists effective procedure computing $f(x)$ on x
- effective procedure for f if exists TM M that leaves output $f(x)$ on tape on input x;
- equivalently defined via other models of computation: μ-recursion, λ-calculus,...
- language L recursive(ly enumerable) if exists (total) TM M accepting L ($L = L(M)$)
- property P (semi-)decidable if $\{x \mid P(x)\}$ is recursive(ly enumerable)
- TM encoded by some $x \in \{0,1\}^*$ (program as bit-string) $\Rightarrow$ countably many TMs
- uncountably many functions $\mathbb{N} \to \mathbb{N}$ $\Rightarrow$ some (most) functions not computable
- exist TM U that is universal: U on $x\#y$ simulates TM M_x (TM having code x) on y
- diagonal d is behaviour exhibited when running M_x on x (itself) for each input x
- complement cd of d distinct from all TM behaviours $\Rightarrow$ not a TM behaviour
- halting problem HP $:= \{M\#x \mid M$ halts for input $x\}$ not recursive
- if L and $\sim L$ recursively enumerable, then (both) recursive

Course themes

- directed and undirected graphs
- relations and functions
- orders and induction
- trees and dags
- finite and infinite counting
- elementary number theory
- Turing machines, algorithms, and complexity
- decidable and undecidable problem

Discrete structures



Recursive/recursively enumerable languages

Definition

A language L (or, more generally, a set) is

- recursively enumerable, if there exists a TM M such that $L = \mathrm{L}(M)$
  i.e. L is the set of strings accepted by M
- recursive, if there exists a total TM M, such that $L = \mathrm{L}(M)$
  i.e. M is required to halt (accept or reject) on all strings

Theorem

- *every recursive set is recursively enumerable;*
- *if a set and its complement are recursively enumerable, they are recursive;*

A non-recursive language HP (recapitulation)

Definition (Halting Problem)

$\mathrm{HP} := \{M\#x \mid M \text{ halts on input } x\}$

Theorem

HP *is not recursive*

Definition

behaviour is a map from input words $x \in \{0,1\}^*$ to either ! (halts) or ↻ (loops).

Proof of Theorem.

Suppose HP were recursive, i.e. there is a total TM K such that $L(K) = \mathrm{HP}$.

1. there is a behaviour *cd* not exhibited by any TM;
2. using K we could construct a TM *CD* exhibiting behaviour *cd*.

Contradiction, so HP is not recursive. ∎

(1) there is a behaviour *cd* not exhibited by any TM

Proof.

Enumerating all TMs as $M_\epsilon, M_0, M_1, M_{00}, \ldots$, their behaviours can be depicted as:

	ϵ	0	1	00	01	10	11	000	001	010	...
M_ϵ	!	↻	↻	!	!	↻	!	↻	!	!	
M_0	↻	↻	!	!	↻	!	!	↻	↻	!	
M_1	↻	!	↻	!	↻	!	!	↻	↻	!	
M_{00}	!	↻	↻	!	!	!	!	↻	↻	!	
M_{01}	!	!	!	!	↻	↻	↻	!	!	↻	...
M_{10}	!	!	↻	!	!	↻	!	!	↻	!	
$\vdots$						$\vdots$					$\ddots$

(1) there is a behaviour *cd* not exhibited by any TM

Proof.

Enumerating all TMs as $M_\epsilon, M_0, M_1, M_{00}, \ldots$, their behaviours can be depicted as:

	ϵ	0	1	00	01	10	11	000	001	010	...
M_ϵ	!	↻	↻	!	!	↻	!	↻	!	!	
M_0	↻	↻	!	!	↻	!	!	↻	↻	!	
M_1	↻	!	↻	!	↻	!	!	↻	↻	!	
M_{00}	!	↻	↻	!	!	!	!	↻	↻	!	
M_{01}	!	!	!	!	↻	↻	↻	!	!	↻	...
M_{10}	!	!	↻	!	!	↻	!	!	↻	!	
$\vdots$						$\vdots$					$\ddots$

diagonal behaviour $d =$! ↻ ↻ ! ↻ ↻ ... ∎

(1) there is a behaviour *cd* not exhibited by any TM

Proof.

Enumerating all TMs as $M_\epsilon, M_0, M_1, M_{00}, \ldots$, their behaviours can be depicted as:

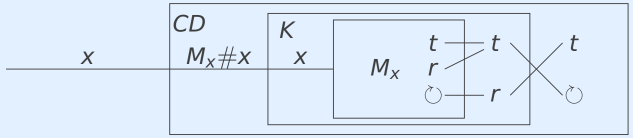
	ϵ	0	1	00	01	10	11	000	001	010	$\ldots$
M_ϵ	↻	↻	↻	!	!	↻	!	↻	!	!	
M_0	↻	!	!	!	↻	!	!	↻	↻	!	
M_1	↻	!	!	!	↻	!	!	↻	↻	!	
M_{00}	!	↻	↻	↻	!	!	!	↻	↻	!	
M_{01}	!	!	!	!	!	↻	↻	!	!	↻	$\cdots$
M_{10}	!	!	↻	!	!	!	!	!	↻	!	
$\vdots$						$\vdots$					$\ddots$

complement diagonal cd = ↻ ! ! ↻ ! ! . . . not exhibited by any TM ■

(2) using K we could construct a TM CD exhibiting behaviour cd

Proof.

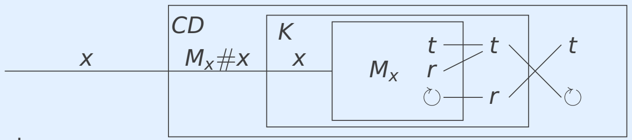
Suppose K were a total TM K such that $L(K) = \mathrm{HP}$. Construct CD:



(2) using K we could construct a TM CD exhibiting behaviour cd

Proof.

Suppose K were a total TM K such that $L(K) = \mathrm{HP}$. Construct CD:



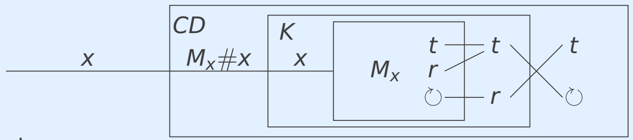
in words:

- input string x is first transformed into the string $M_x\#x$

(2) using K we could construct a TM CD exhibiting behaviour cd

Proof.

Suppose K were a total TM K such that $L(K) = \mathrm{HP}$. Construct CD:



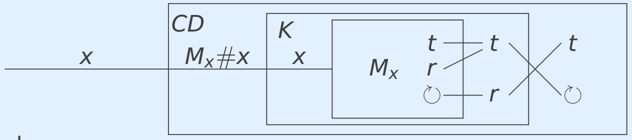
in words:

- input string x is first transformed into the string $M_x\#x$
- then run K on $M_x\#x$

(2) using K we could construct a TM CD exhibiting behaviour cd

Proof.

Suppose K were a total TM K such that $L(K) = \mathrm{HP}$. Construct CD:



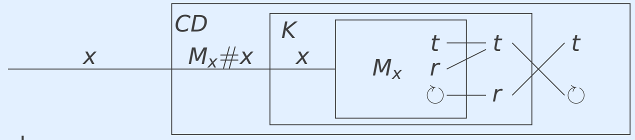
in words:

- input string x is first transformed into the string $M_x\#x$
- then run K on $M_x\#x$
- yields t if M_x halts on x, otherwise r (no looping; K total)

(2) using K we could construct a TM CD exhibiting behaviour cd

Proof.

Suppose K were a total TM K such that $L(K) = \mathrm{HP}$. Construct CD:



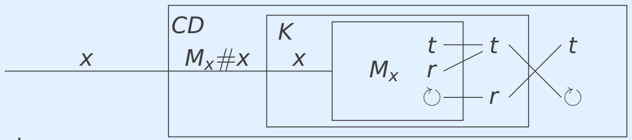
in words:

- input string x is first transformed into the string $M_x\#x$
- then run K on $M_x\#x$
- yields t if M_x halts on x, otherwise r (no looping; K total)
- if K accepts then we loop, otherwise K rejects and we accept (return t)

(2) using K we could construct a TM CD exhibiting behaviour cd

Proof.

Suppose K were a total TM K such that $L(K) = \mathrm{HP}$. Construct CD:



in words:

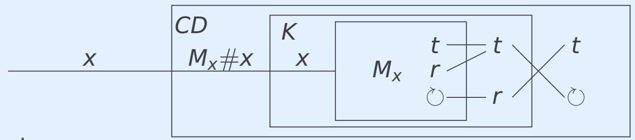
- input string x is first transformed into the string $M_x\#x$
- then run K on $M_x\#x$
- yields t if M_x halts on x, otherwise r (no looping; K total)
- if K accepts then we loop, otherwise K rejects and we accept (return t)

M_x halts/loops on x iff K accepts/rejects $M_x\#x$ iff CD loops/halts on x

(2) using K we could construct a TM CD exhibiting behaviour cd

Proof.

Suppose K were a total TM K such that $L(K) = \mathrm{HP}$. Construct CD:



in words:

- input string x is first transformed into the string $M_x\#x$
- then run K on $M_x\#x$
- yields t if M_x halts on x, otherwise r (no looping; K total)
- if K accepts then we loop, otherwise K rejects and we accept (return t)

M_x halts/loops on x iff K accepts/rejects $M_x\#x$ iff CD loops/halts on x

CD exhibits behaviour cd ∎

Complements

Lemma

if L recursive, then so is $\sim L$

Proof.

If $L = L(M)$ for total TM M, then $\sim L = L(M')$ for M' as M but swapping accept, reject ∎

Complements

Lemma

if L recursive, then so is $\sim L$

Proof.

If $L = L(M)$ for total TM M, then $\sim L = L(M')$ for M' as M but swapping accept, reject ∎

Theorem

$\sim$HP is not recursively enumerable (although HP is)

Proof.

Suppose $\sim$HP were recursively enumerable.

- then both HP (previous lecture) and $\sim$HP would be recursively enumerable;
- so both HP and $\sim$HP would in fact be recursive (previous lecture);
- but that would contradict that HP is not recursive (previous lecture).

So $\sim$HP is not recursively enumerable. ∎

Closure properties of recursively enumerable languages

Theorem

recursively enumerable languages are closed under union and intersection, but not under complement or difference

Proof.

- if $L_1 = L(M_1)$, $L_2 = L(M_2)$, running both M_1, M_2 on a given input x, and accepting if at least one/both of them accepts, shows closure under union/intersection.

Closure properties of recursively enumerable languages

Theorem

recursively enumerable languages are closed under union and intersection, but not under complement or difference

Proof.

- if $L_1 = L(M_1)$, $L_2 = L(M_2)$, running both M_1, M_2 on a given input x, and accepting if at least one/both of them accepts, shows closure under union/intersection.
- closure under complement fails as shown by the previous theorem. in particular, $\sim$HP is not r.e., although HP is. from this it follows that closure under difference fails since $\sim\text{HP} = \{0,1\}^* - \text{HP}$ and $\{0,1\}^*$ is trivially r.e. ∎

Closure properties of recursive languages

Theorem

recursively enumerable languages are closed under union and intersection, but not under complement or difference

Proof.

- if $L_1 = L(M_1)$, $L_2 = L(M_2)$, running both M_1, M_2 on a given input x, and accepting if at least one/both of them accepts, shows closure under union/intersection.
- closure under complement fails as shown by the previous theorem. in particular, $\sim$HP is not r.e., although HP is. from this it follows that closure under difference fails since $\sim \mathrm{HP} = \{0,1\}^* - \mathrm{HP}$ and $\{0,1\}^*$ is trivially r.e. ■

Corollary

recursive languages are closed under union, intersection, complement, and difference

by above closure properties and using De Morgan; $\sim(L_1 \cup L_2) = (\sim L_1) \cap (\sim L_2)$

Another non-recursive language MP

Definition (Membership Problem)

$\mathrm{MP} := \{M\#x \mid M \text{ accepts input } x\}$

Theorem

MP *is not recursive*

Proof.

Suppose MP were recursive, i.e. there is a total TM K such that $L(K) = \mathrm{MP}$.

1. there is a language *cd* not accepted by any TM;
2. using K we could construct a TM *CD* accepting *cd*.

Contradiction, so MP is not recursive. ■

(1) there is a language *cd* not accepted by any TM

Proof.

Enumerating all TMs as M_ϵ, M_0, M_1, M_{00}, ..., their languages can be depicted as:

	ϵ	0	1	00	01	10	11	000	001	010	...
M_ϵ	×	×	×	✓	✓	×	×	×	×	✓	
M_0	×	×	✓	✓	×	✓	✓	×	×	✓	
M_1	×	×	×	✓	×	×	✓	×	×	✓	
M_{00}	✓	×	×	✓	✓	✓	✓	×	×	✓	
M_{01}	×	✓	×	✓	×	×	×	✓	✓	×	...
M_{10}	✓	✓	×	✓	×	×	×	×	×	×	
⋮						⋮					⋱

(1) there is a language *cd* not accepted by any TM

Proof.

Enumerating all TMs as M_ϵ, M_0, M_1, M_{00}, ..., their languages can be depicted as:

	ϵ	0	1	00	01	10	11	000	001	010	...
M_ϵ	×	×	×	✓	✓	×	×	×	×	✓	
M_0	×	×	✓	✓	×	✓	✓	×	×	✓	
M_1	×	×	×	✓	×	×	✓	×	×	✓	
M_{00}	✓	×	×	✓	✓	✓	✓	×	×	✓	
M_{01}	×	✓	×	✓	×	×	×	✓	✓	×	...
M_{10}	✓	✓	×	✓	×	×	×	×	×	×	
⋮						⋮					⋱

diagonal language $\quad d = \{00, \ldots\}$ ■

(1) there is a language *cd* not accepted by any TM

Proof.

Enumerating all TMs as M_ϵ, M_0, M_1, M_{00}, ..., their languages can be depicted as:

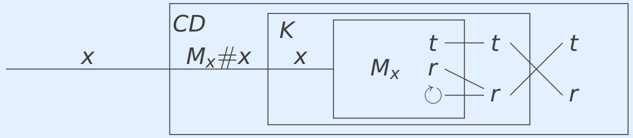
	ϵ	0	1	00	01	10	11	000	001	010	...
M_ϵ	$\checkmark$	$\times$	$\times$	$\checkmark$	$\checkmark$	$\times$	$\times$	$\times$	$\times$	$\checkmark$	
M_0	$\times$	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	$\checkmark$	$\checkmark$	$\times$	$\times$	$\checkmark$	
M_1	$\times$	$\times$	$\checkmark$	$\checkmark$	$\times$	$\times$	$\checkmark$	$\times$	$\times$	$\checkmark$	
M_{00}	$\checkmark$	$\times$	$\times$	$\times$	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	$\times$	$\checkmark$	
M_{01}	$\times$	$\checkmark$	$\times$	$\checkmark$	$\checkmark$	$\times$	$\times$	$\checkmark$	$\checkmark$	$\times$	...
M_{10}	$\checkmark$	$\checkmark$	$\times$	$\checkmark$	$\times$	$\checkmark$	$\times$	$\times$	$\times$	$\times$	
$\vdots$					$\vdots$						$\ddots$

complement diagonal $cd = \{\epsilon, 0, 1, 01, 10, 11, \ldots\}$ not accepted by any TM ∎

(2) using K we could construct a TM CD accepting *cd*

Proof.

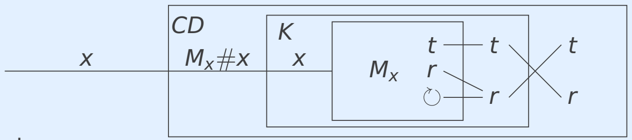
Suppose K were a total TM K such that $L(K) = \mathrm{MP}$. Construct CD:



(2) using K we could construct a TM CD accepting *cd*

Proof.

Suppose K were a total TM K such that $L(K) = \mathrm{MP}$. Construct CD:



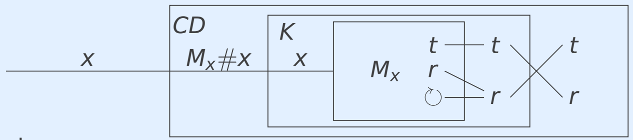
in words:

- input string x is first transformed into the string $M_x\#x$

(2) using K we could construct a TM CD accepting *cd*

Proof.

Suppose K were a total TM K such that $L(K) = \mathrm{MP}$. Construct CD:



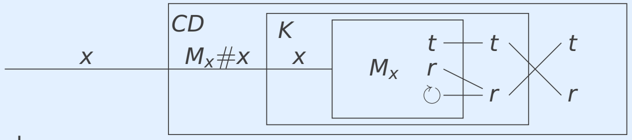
in words:

- input string x is first transformed into the string $M_x\#x$
- then run K on $M_x\#x$

(2) using K we could construct a TM CD accepting cd

Proof.

Suppose K were a total TM K such that $L(K) = \mathrm{MP}$. Construct CD:



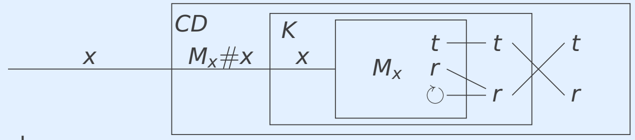
in words:

- input string x is first transformed into the string $M_x\#x$
- then run K on $M_x\#x$
- yields t if M_x accepts x, otherwise r (no looping; K total)

(2) using K we could construct a TM CD accepting cd

Proof.

Suppose K were a total TM K such that $L(K) = \mathrm{MP}$. Construct CD:



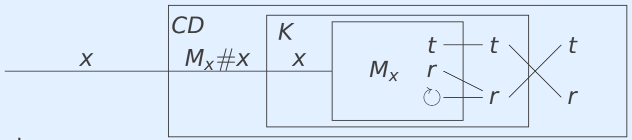
in words:

- input string x is first transformed into the string $M_x\#x$
- then run K on $M_x\#x$
- yields t if M_x accepts x, otherwise r (no looping; K total)
- if K accepts then we reject, otherwise K rejects and we accept (return t)

(2) using K we could construct a TM CD accepting cd

Proof.

Suppose K were a total TM K such that $L(K) = \mathrm{MP}$. Construct CD:



in words:

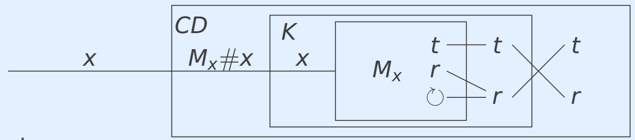
- input string x is first transformed into the string $M_x\#x$
- then run K on $M_x\#x$
- yields t if M_x accepts x, otherwise r (no looping; K total)
- if K accepts then we reject, otherwise K rejects and we accept (return t)

M_x accepts/does not accept x iff K accepts/rejects $M_x\#x$ iff CD rejects/accepts x

(2) using K we could construct a TM CD accepting cd

Proof.

Suppose K were a total TM K such that $L(K) = \mathrm{MP}$. Construct CD:



in words:

- input string x is first transformed into the string $M_x\#x$
- then run K on $M_x\#x$
- yields t if M_x accepts x, otherwise r (no looping; K total)
- if K accepts then we reject, otherwise K rejects and we accept (return t)

M_x accepts/does not accept x iff K accepts/rejects $M_x\#x$ iff CD rejects/accepts x

CD accepts cd ∎

Border between in/outside of language

Remark

- language L being recursively enumerable (recursive) depends on its border
- how 'difficult' it is to decide that string x is in L (and not in L)
- does not depend on cardinality of L
- but note every finite language recursive (finite case-distinction)

Border between in/outside of language

Remark

- language L being recursively enumerable (recursive) depends on its border
- how 'difficult' it is to decide that string x is in L (and not in L)
- does not depend on cardinality of L
- but note every finite language recursive

Example

for A set of all strings over alphabet $\supseteq \{0, 1, \#\}$

- $\emptyset \subseteq \text{HP} \subseteq A$

Border between in/outside of language

Remark

- language L being recursively enumerable (recursive) depends on its border
- how 'difficult' it is to decide that string x is in L (and not in L)
- does not depend on cardinality of L
- but note every finite language recursive

Example

for A set of all strings over alphabet $\supseteq \{0, 1, \#\}$

- $\emptyset \subseteq \text{HP} \subseteq A$
- although both $\emptyset, A$ recursive, HP is not recursive

Border between in/outside of language

Remark

- language L being recursively enumerable (recursive) depends on its border
- how 'difficult' it is to decide that string x is in L (and not in L)
- does not depend on cardinality of L
- but note every finite language recursive

Example

for A set of all strings over alphabet $\supseteq \{0, 1, \#\}$

- $\emptyset \subseteq \text{HP} \subseteq A$
- although both $\emptyset, A$ recursive, HP is not recursive
- sub/superset of recursive or recursively enumerable language need not be so

Border between in/outside of language

Remark

- language L being recursively enumerable (recursive) depends on its border
- how 'difficult' it is to decide that string x is in L (and not in L)
- does not depend on cardinality of L
- but note every finite language recursive

Example

for A set of all strings over alphabet $\supseteq \{0, 1, \#\}$

- $\emptyset \subseteq \mathrm{HP} \subseteq A$
- although both $\emptyset, A$ recursive, HP is not recursive
- sub/superset of recursive or recursively enumerable language need not be so
- may have $L_0 \subset L_1 \subset L_2 \subset L_3 \subset \ldots$ such that L_{2i} recursive, L_{2i+1} not recursive

Relating non-recursiveness of HP and MP

similarity between proofs

suppose $L = L(K)$ for some total TM K

- determine cd distinct from $L(M)$ for every TM M by diagonalising away
- show that using K we could construct TM CD with $L(CD) = cd$

so supposition must be false

avoid redoing diagonalisation?

Relating non-recursiveness of HP and MP

similarity between proofs

suppose $L = L(K)$ for some total TM K

- determine cd distinct from $L(M)$ for every TM M by diagonalising away
- show that using K we could construct TM CD with $L(CD) = cd$

so supposition must be false

reducing problems to each other

- solve problem L using solution for problem L' as sub-routine

Relating non-recursiveness of HP and MP

similarity between proofs

suppose $L = L(K)$ for some total TM K

- determine cd distinct from $L(M)$ for every TM M by diagonalising away
- show that using K we could construct TM CD with $L(CD) = cd$

so supposition must be false

reducing problems to each other

- solve problem L using solution for problem L' as sub-routine
- say: reduce L to L' (reduction from L to L')

Relating non-recursiveness of HP and MP

similarity between proofs

suppose $L = L(K)$ for some total TM K

- determine cd distinct from $L(M)$ for every TM M by diagonalising away
- show that using K we could construct TM CD with $L(CD) = cd$

so supposition must be false

reducing problems to each other

- solve problem L using solution for problem L' as sub-routine
- say: reduce L to L'
- $x \in L$ iff $f(x) \in L'$ using (computable!) function f

Relating non-recursiveness of HP and MP

similarity between proofs

suppose $L = L(K)$ for some total TM K

- determine cd distinct from $L(M)$ for every TM M by diagonalising away
- show that using K we could construct TM CD with $L(CD) = cd$

so supposition must be false

reducing problems to each other

- solve problem L using solution for problem L' as sub-routine
- say: reduce L to L'
- $x \in L$ iff $f(x) \in L'$ using function f
- L is not more complex than L' (may be strictly less so; may be simpler subroutine)

Relating non-recursiveness of HP and MP

similarity between proofs

suppose $L = L(K)$ for some total TM K

- determine cd distinct from $L(M)$ for every TM M by diagonalising away
- show that using K we could construct TM CD with $L(CD) = cd$

so supposition must be false

reducing problems to each other

- solve problem L using solution for problem L' as sub-routine
- say: reduce L to L'
- $x \in L$ iff $f(x) \in L'$ using function f
- L is not more complex than L'
- write: $L \leq L'$ (beware: $L \subseteq L'$ need not imply $L \leq L'$!)

Relating non-recursiveness of HP and MP

similarity between proofs

suppose $L = L(K)$ for some total TM K

- determine cd distinct from $L(M)$ for every TM M by diagonalising away
- show that using K we could construct TM CD with $L(CD) = cd$

so supposition must be false

reducing problems to each other

- solve problem L using solution for problem L' as sub-routine
- say: reduce L to L'
- $x \in L$ iff $f(x) \in L'$ using function f
- L is not more complex than L'
- write: $L \leq L'$
- if L not recursive and $L \leq L'$, then L' not recursive

Formal definition of reduction

Definition (recall from before)

$f\colon \Sigma^* \to \Sigma^*$ is computable if $\exists$ a total TM T with input alphabet Σ, such that on input $x \in \Sigma^*$, T writes $f(x)$ on the tape

Formal definition of reduction

Definition (recall from before)

$f\colon \Sigma^* \to \Sigma^*$ is computable if $\exists$ a total TM T with input alphabet Σ, such that on input $x \in \Sigma^*$, T writes $f(x)$ on the tape

Definition

For $L, L' \subseteq \Sigma^*$, L is reducible to L'; denoted by $L \leq L'$ if $\exists$ a computable $f\colon \Sigma^* \to \Sigma^*$ such that $x \in L \Leftrightarrow f(x) \in M$

Formal definition of reduction

Definition (recall from before)

$f\colon \Sigma^* \to \Sigma^*$ is computable if $\exists$ a total TM T with input alphabet Σ, such that on input $x \in \Sigma^*$, T writes $f(x)$ on the tape

Definition

For $L, L' \subseteq \Sigma^*$, L is reducible to L'; denoted by $L \leq L'$ if $\exists$ a computable $f\colon \Sigma^* \to \Sigma^*$ such that $x \in L \Leftrightarrow f(x) \in M$

Theorem

if L not recursive and $L \leq L'$, then L' is not recursive

Proof.

supposing $L \leq L'$ by function f by TM T and L' were recursive by total TM K, then $x \in L$ iff $f(x) \in L'$ iff K accepts $f(x)$. that is, L is recursive using TM that first transforms x into $f(x)$ by running T and then runs K on $f(x)$. contradiction. so L' is not recursive. ∎

Reducing HP to MP ?

Reducing HP to MP

Definition

$f(M\#x) = M'\#x$ where M' is obtained from M by changing reject into accept

Reducing HP to MP

Definition

$f(M\#x) = M'\#x$ where M' is obtained from M by changing reject into accept

Lemma

$M\#x \in$ HP *iff* $f(M\#x) \in$ MP

Proof.

$M\#x \in$ HP $\Rightarrow$ M halts on x $\Rightarrow$ M' accepts x $\Rightarrow$ $M'\#x \in$ MP
$M\#x \notin$ HP $\Rightarrow$ M does not halt on x $\Rightarrow$ M' does not halt on x $\Rightarrow$ $M'\#x \notin$ MP ∎

Reducing HP to MP

Definition

$f(M\#x) = M'\#x$ where M' is obtained from M by changing reject into accept

Lemma

$M\#x \in$ HP *iff* $f(M\#x) \in$ MP

Proof.

$M\#x \in$ HP $\Rightarrow$ M halts on x $\Rightarrow$ M' accepts x $\Rightarrow$ $M'\#x \in$ MP
$M\#x \notin$ HP $\Rightarrow$ M does not halt on x $\Rightarrow$ M' does not halt on x $\Rightarrow$ $M'\#x \notin$ MP ∎

Corollary

MP *is not recursive*

Proof.

since HP is not recursive (assumed known) and HP $\leq$ MP ∎

Reducing MP to HP ?

Reducing MP to HP

Definition

$f(M\#x) = M'\#x$ where M' is obtained from M by changing reject into looping

Reducing MP to HP

Definition

$f(M\#x) = M'\#x$ where M' is obtained from M by changing reject into looping

Lemma

$M\#x \in \mathrm{MP}$ *iff* $f(M\#x) \in \mathrm{HP}$

Proof.

$M\#x \in \mathrm{MP} \Rightarrow M$ accepts $x \Rightarrow M'$ accepts $x \Rightarrow M'\#x \in \mathrm{HP}$
$M\#x \notin \mathrm{MP} \Rightarrow M$ does not accept $x \Rightarrow M'$ does not halt on $x \Rightarrow M'\#x \notin \mathrm{HP}$ ∎

Reducing MP to HP

Definition

$f(M\#x) = M'\#x$ where M' is obtained from M by changing reject into looping

Lemma

$M\#x \in \mathrm{MP}$ *iff* $f(M\#x) \in \mathrm{HP}$

Proof.

$M\#x \in \mathrm{MP} \Rightarrow M$ accepts $x \Rightarrow M'$ accepts $x \Rightarrow M'\#x \in \mathrm{HP}$
$M\#x \notin \mathrm{MP} \Rightarrow M$ does not accept $x \Rightarrow M'$ does not halt on $x \Rightarrow M'\#x \notin \mathrm{HP}$ ∎

Corollary

HP *is not recursive*

Proof.

since MP is not recursive (assumed known) and $\mathrm{MP} \leq \mathrm{HP}$ ∎

More on reducibility

Theorem

$\leq$ *is reflexive and transitive*

More on reducibility

Theorem

$\leq$ is reflexive and transitive

Proof.

- to show reflexivity, take $f(x) = x$ (the identity function)

More on reducibility

Theorem

$\leq$ is reflexive and transitive

Proof.

- to show reflexivity, take $f(x) = x$
- to show transitivity, compose the functions (run the TMs consecutively) $\blacksquare$

More on reducibility

Theorem

$\leq$ is reflexive and transitive

Proof.

- to show reflexivity, take $f(x) = x$
- to show transitivity, compose the functions $\blacksquare$

Lemma

there is no program testing whether a given program is a "hello world"-program (prints "hello world" and then accepts)

More on reducibility

Theorem

$\leq$ is reflexive and transitive

Proof.

- to show reflexivity, take $f(x) = x$
- to show transitivity, compose the functions $\blacksquare$

Lemma

there is no program testing whether a given program is a "hello world"-program

Proof.

show HP $\leq$ "hello world"-program. let $f(M\#x)$ be $M'\#x$ with M' a TM that first runs M on its input, if that halts overwrites tape with "hello world" and then accepts:
$M\#x \in$ HP $\Rightarrow$ M halts on x $\Rightarrow$ M' is a "hello world"-program
$M\#x \notin$ HP $\Rightarrow$ M does not halt on x $\Rightarrow$ M' is not a "hello world"-program $\blacksquare$

Regular languages

Question

What languages can be accepted for machines more restricted than TMs?

Regular languages

We consider finite automata. These accept regular languages, and will show these are recursive, but not necessarily the other way around,

relevance of regular languages

- software for designing and testing of digital circuits
- software components of compiler, e.g. for lexical analysis:
- software for searching in long texts
- software to verify all kinds of systems having a finite number of states
- components of computer games (computer-controlled non-player-character)

Deterministic finite automata (DFAs)

Example

$\emptyset$ and the set of all strings are regular, as are all finite languages.

Deterministic finite automata (DFAs)

Example

$\emptyset$ and the set of all strings are regular, as are all finite languages.

Definition

A DFA is a 5-tuple $A = (Q, \Sigma, \delta, s, F)$ with

1. Q a finite set of states
2. Σ a finite set of input symbols, (Σ is called the input alphabet)
3. $\delta\colon Q \times \Sigma \to Q$ the transition function
4. $s \in Q$, the start or initial state
5. $F \subseteq Q$ a finite set of accepting or final states

Deterministic finite automata (DFAs)

Example

$\emptyset$ and the set of all strings are regular, as are all finite languages.

Definition

A DFA is a 5-tuple $A = (Q, \Sigma, \delta, s, F)$ with

1. Q a finite set of states
2. Σ a finite set of input symbols, (Σ is called the input alphabet)
3. $\delta\colon Q \times \Sigma \to Q$ the transition function
4. $s \in Q$, the start or initial state
5. $F \subseteq Q$ a finite set of accepting or final states

Beware: δ must be defined, for all possible inputs

Transition table

	$a_1 \in \Sigma$	$a_2 \in \Sigma$	$\cdots$
$q_1 \in Q$	$\delta(q_1, a_1)$	$\delta(q_1, a_2)$	$\cdots$
$q_2 \in Q$	$\delta(q_2, a_1)$		
$\vdots$	$\vdots$		

Transition table

	$a_1 \in \Sigma$	$a_2 \in \Sigma$	$\cdots$
$q_1 \in Q$	$\delta(q_1, a_1)$	$\delta(q_1, a_2)$	$\cdots$
$q_2 \in Q$	$\delta(q_2, a_1)$		
$\vdots$	$\vdots$		

Transition graph

For a DFA $A = (Q, \Sigma, \delta, s, F)$, its (directed) transition graph with initial state d and final states F where:

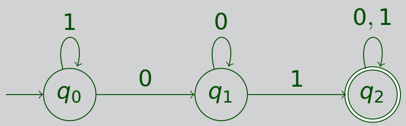
1. the states are the nodes
2. the edges E are
$$(p, q) \qquad p, q \in Q \text{ and } \exists a \in \Sigma \text{ with } \delta(p, a) = q$$
3. the edges are labelled by symbols by a function $b \colon E \to \Sigma$ defined by
$$(p, q) \mapsto a \qquad \text{if } \delta(p, a) = q$$

Example

The DFA $A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$ with transition table

	0	1
$\to q_0$	q_1	q_0
q_1	q_1	q_2
$*q_2$	q_2	q_2

has the following transition graph:



Definition (extending the transition function)

Let δ be a transition function. The extended transition function $\hat{\delta} \colon Q \times \Sigma^* \to Q$ is inductively defined by:

$$\hat{\delta}(q, \epsilon) := q$$
$$\hat{\delta}(q, xa) := \delta(\hat{\delta}(q, x), a) \qquad x \in \Sigma^*, \ a \in \Sigma$$

Definition (extending the transition function)

Let δ be a transition function. The extended transition function $\hat{\delta}\colon Q \times \Sigma^* \to Q$ is inductively defined by:

$$\hat{\delta}(q, \epsilon) := q$$
$$\hat{\delta}(q, xa) := \delta(\hat{\delta}(q, x), a) \qquad x \in \Sigma^*,\ a \in \Sigma$$

Definition

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA; the language $\mathsf{L}(A)$ accepted by A is:

$$\mathsf{L}(A) := \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F\}$$

Definition (extending the transition function)

Let δ be a transition function. The extended transition function $\hat{\delta}\colon Q \times \Sigma^* \to Q$ is inductively defined by:

$$\hat{\delta}(q, \epsilon) := q$$
$$\hat{\delta}(q, xa) := \delta(\hat{\delta}(q, x), a) \qquad x \in \Sigma^*,\ a \in \Sigma$$

Definition

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA; the language $\mathsf{L}(A)$ accepted by A is:

$$\mathsf{L}(A) := \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F\}$$

Example

For the DFA A above, $\hat{\delta}(q_0, 0010) = q_2$
$\hat{\delta}(q_0, 0010)$ is computed recursively as follows:

- $\hat{\delta}(q_0, 0010) = \delta(\hat{\delta}(q_0, 001), 0) = \delta(q_2, 0) = q_2$
- $\hat{\delta}(q_0, 001) = \delta(\hat{\delta}(q_0, 00), 1) = \delta(q_1, 1) = q_2$
- $\hat{\delta}(q_0, 00) = \delta(\hat{\delta}(q_0, 0), 0) = \delta(q_1, 0) = q_1$
- $\hat{\delta}(q_0, 0) = \delta(\hat{\delta}(q_0, \epsilon), 0) = \delta(q_0, 0) = q_1$

Example

For the DFA A above, $\hat{\delta}(q_0, 0010) = q_2$
$\hat{\delta}(q_0, 0010)$ is computed recursively as follows:

- $\hat{\delta}(q_0, 0010) = \delta(\hat{\delta}(q_0, 001), 0) = \delta(q_2, 0) = q_2$
- $\hat{\delta}(q_0, 001) = \delta(\hat{\delta}(q_0, 00), 1) = \delta(q_1, 1) = q_2$
- $\hat{\delta}(q_0, 00) = \delta(\hat{\delta}(q_0, 0), 0) = \delta(q_1, 0) = q_1$
- $\hat{\delta}(q_0, 0) = \delta(\hat{\delta}(q_0, \epsilon), 0) = \delta(q_0, 0) = q_1$

Example

For the DFA A, we have $\mathsf{L}(A) = \{x01y \mid x, y \in \Sigma^*\}$. The language $\mathsf{L}(A)$ is the set of all words in which 01 occurs somewhere (or rather of words not of the form: a number of 1s followed by a number of 0s)

Example

For the DFA A above, $\hat{\delta}(q_0, 0010) = q_2$
$\hat{\delta}(q_0, 0010)$ is computed recursively as follows:

- $\hat{\delta}(q_0, 0010) = \delta(\hat{\delta}(q_0, 001), 0) = \delta(q_2, 0) = q_2$
- $\hat{\delta}(q_0, 001) = \delta(\hat{\delta}(q_0, 00), 1) = \delta(q_1, 1) = q_2$
- $\hat{\delta}(q_0, 00) = \delta(\hat{\delta}(q_0, 0), 0) = \delta(q_1, 0) = q_1$
- $\hat{\delta}(q_0, 0) = \delta(\hat{\delta}(q_0, \epsilon), 0) = \delta(q_0, 0) = q_1$

Example

For the DFA A, we have $\mathrm{L}(A) = \{x01y \mid x, y \in \Sigma^*\}$. The language $\mathrm{L}(A)$ is the set of all words in which 01 occurs somewhere (or rather of words not of the form: a number of 1s followed by a number of 0s)

Definition

A formal language L is regular, if $\exists$ DFA A, such that $\mathrm{L}(A) = L$