



## Discrete structures

Cezary Kaliszyk

Raoul Schikora

**Vincent van Oostrom**

<http://cl-informatik.uibk.ac.at>

## Summary last week

- function  $f : \mathbb{N} \rightarrow \mathbb{N}$  **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**:  $\mu$ -recursion,  $\lambda$ -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)

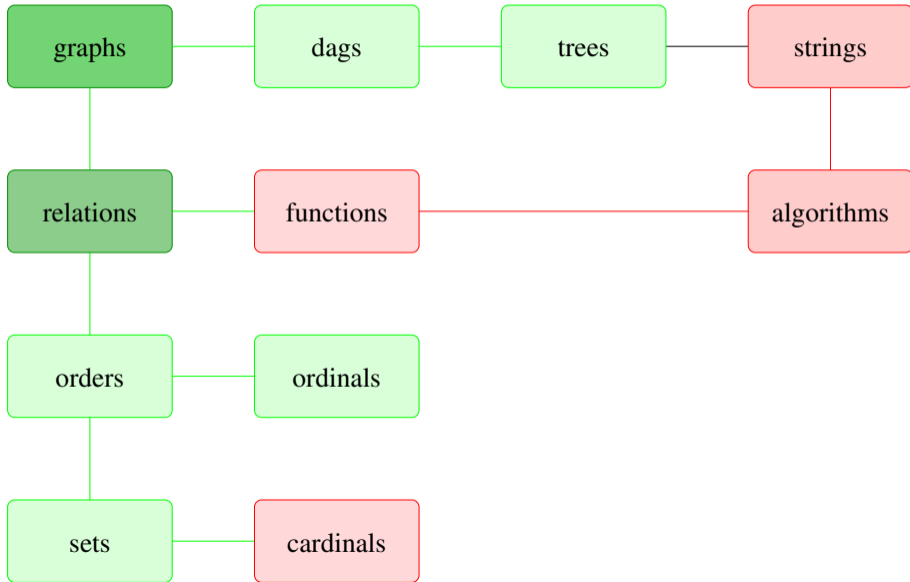
# Summary last week

- function  $f : \mathbb{N} \rightarrow \mathbb{N}$  **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**:  $\mu$ -recursion,  $\lambda$ -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  $\mathbb{N} \rightarrow \mathbb{N} \Rightarrow$  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**  $\bar{d}$  of  $d$  **distinct** from all TM behaviours  $\Rightarrow$  not a TM behaviour
- 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  $L = L(M)$   
i.e.  $L$  is the set of strings **accepted** by  $M$
- **recursive**, if there exists a **total** TM  $M$ , such that  $L = L(M)$   
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)

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  $\circlearrowleft$  (loops).

## Proof of Theorem.

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

- 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  $M_\epsilon, M_0, M_1, M_{00}, \dots$ , their behaviours can be depicted as:

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



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

## Proof.

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

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

diagonal behaviour  $d = ! \quad \circ \quad \circ \quad ! \quad \circ \quad \circ \quad \dots$

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

## Proof.

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

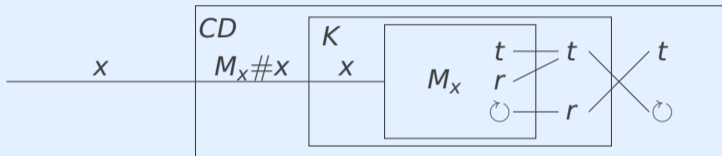
	$\epsilon$	0	1	00	01	10	11	000	001	010	...
$M_\epsilon$	○	○	○	!	!	○	!	○	!	!	
$M_0$	○	!	!	!	○	!	!	○	○	!	
$M_1$	○	!	!	!	○	!	!	○	○	!	
$M_{00}$	!	○	○	○	!	!	!	○	○	!	
$M_{01}$	!	!	!	!	!	○	○	!	!	○	...
$M_{10}$	!	!	○	!	!	!	!	!	○	!	
$\vdots$						$\vdots$					$\ddots$

complement diagonal  $cd = \text{○ ! ! ○ ! ! ...}$  not exhibited by any TM

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

**Proof.**

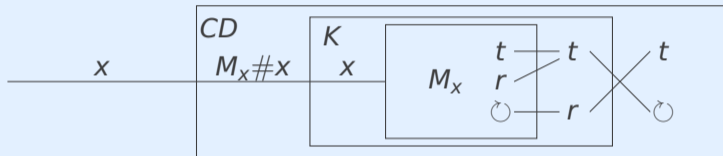
Suppose  $K$  were a **total TM**  $K$  such that  $L(K) = HP$ . Construct  $CD$ :



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

### Proof.

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



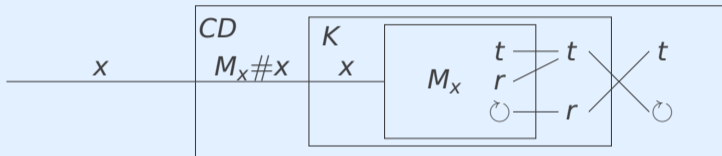
in words:

- 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  $L(K) = \text{HP}$ . Construct  $CD$ :



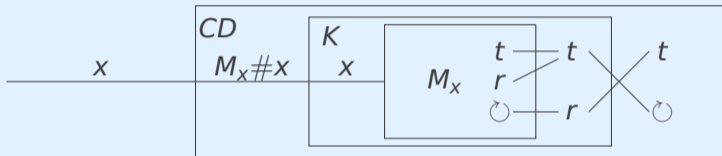
in words:

- 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  $L(K) = \text{HP}$ . Construct  $CD$ :



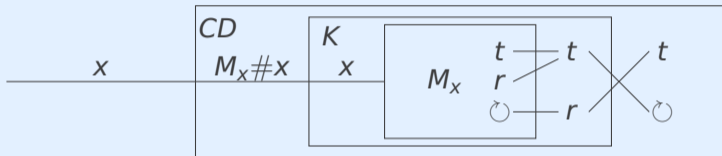
in words:

- 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  $L(K) = \text{HP}$ . 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  $t$ )

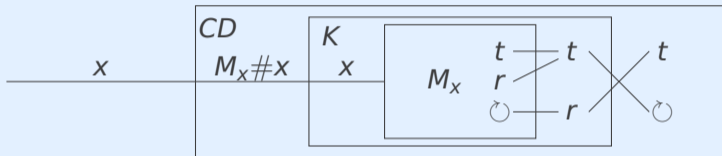




## (2) using $K$ we could construct a TM $CD$ exhibiting behaviour $cd$

### Proof.

Suppose  $K$  were a **total TM**  $K$  such that  $L(K) = \text{HP}$ . 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,  $\sim\text{HP}$  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,  $\sim\text{HP}$  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. ■

## 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)

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

## Theorem

*MP is not recursive*

## Proof.

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

- 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  $M_\epsilon, M_0, M_1, M_{00}, \dots$ , their languages can be depicted as:

	$\epsilon$	0	1	00	01	10	11	000	001	010	...
$M_\epsilon$	×	×	×	✓	✓	×	×	×	×	✓	
$M_0$	×	×	✓	✓	×	✓	✓	×	×	✓	
$M_1$	×	×	×	✓	×	×	✓	×	×	✓	
$M_{00}$	✓	×	×	✓	✓	✓	✓	×	×	✓	
$M_{01}$	×	✓	×	✓	×	×	×	✓	✓	×	...
$M_{10}$	✓	✓	×	✓	×	×	×	×	×	×	
⋮						⋮					⋮



# (1) there is a language $d$ not accepted by any TM

## Proof.

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

	$\epsilon$	0	1	00	01	10	11	000	001	010	...
$M_\epsilon$	×	×	×	✓	✓	×	×	×	×	✓	
$M_0$	×	×	✓	✓	×	✓	✓	×	×	✓	
$M_1$	×	×	×	✓	×	×	✓	×	×	✓	
$M_{00}$	✓	×	×	✓	✓	✓	✓	×	×	✓	
$M_{01}$	×	✓	×	✓	×	×	×	✓	✓	×	...
$M_{10}$	✓	✓	×	✓	×	×	×	×	×	×	
$\vdots$						$\vdots$					$\ddots$

diagonal language  $d = \{00, \dots\}$

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

## Proof.

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

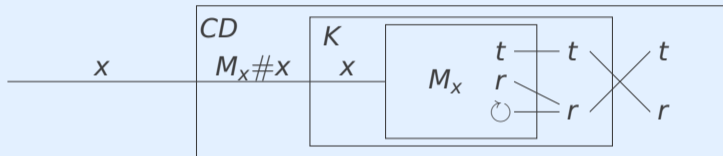
	$\epsilon$	0	1	00	01	10	11	000	001	010	...
$M_\epsilon$	✓	×	×	✓	✓	×	×	×	×	✓	
$M_0$	×	✓	✓	✓	×	✓	✓	×	×	✓	
$M_1$	×	×	✓	✓	×	×	✓	×	×	✓	
$M_{00}$	✓	×	×	×	✓	✓	✓	×	×	✓	
$M_{01}$	×	✓	×	✓	✓	×	×	✓	✓	×	...
$M_{10}$	✓	✓	×	✓	×	✓	×	×	×	×	
$\vdots$						$\vdots$					$\ddots$

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

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

**Proof.**

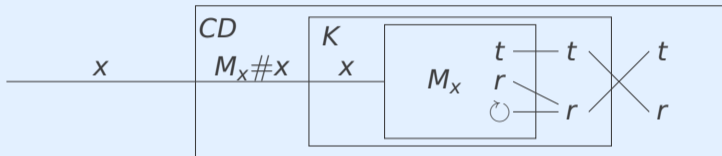
Suppose  $K$  were a **total TM**  $K$  such that  $L(K) = MP$ . Construct  $CD$ :



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

### Proof.

Suppose  $K$  were a **total TM**  $K$  such that  $L(K) = MP$ . Construct  $CD$ :



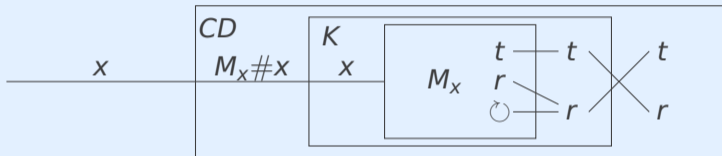
in words:

- 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  $L(K) = MP$ . Construct  $CD$ :



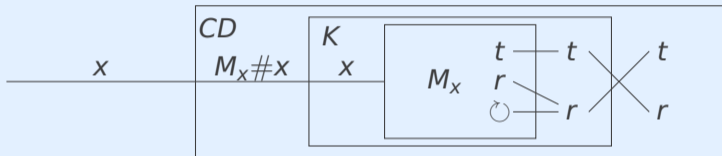
in words:

- 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  $L(K) = MP$ . Construct  $CD$ :



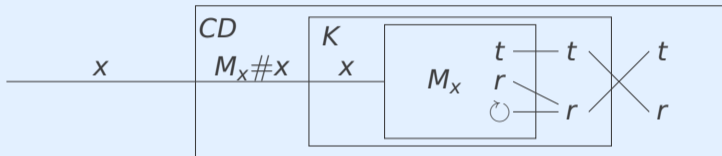
in words:

- 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  $L(K) = MP$ . Construct  $CD$ :



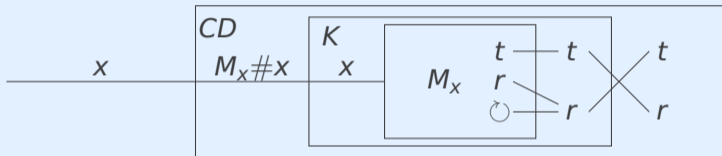
in words:

- 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  $L(K) = MP$ . 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  $t$ )

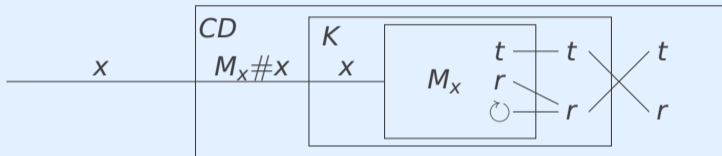
$M_x$  accepts/does not accept  $x$  iff  $K$  accepts/rejects  $M_x \# x$  iff  $CD$  rejects/accepts  $x$



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

### Proof.

Suppose  $K$  were a **total TM**  $K$  such that  $L(K) = MP$ . 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  $\supseteq \{0, 1, \#\}$

- $\emptyset \subseteq HP \subseteq A$

# 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  $\supseteq \{0, 1, \#\}$

- $\emptyset \subseteq HP \subseteq A$
- 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  $\supseteq \{0, 1, \#\}$

- $\emptyset \subseteq HP \subseteq A$
- 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  $\supseteq \{0, 1, \#\}$

- $\emptyset \subseteq HP \subseteq A$
- although both  $\emptyset, A$  recursive,  $HP$  is not recursive
- sub/superset of recursive or recursively enumerable language need not be so
- may have  $L_0 \subset L_1 \subset L_2 \subset L_3 \subset \dots$  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)

$f: \Sigma^* \rightarrow \Sigma^*$  is **computable** if  $\exists$  a total TM  $\mathcal{T}$  with input alphabet  $\Sigma$ , such that on input  $x \in \Sigma^*$ ,  $\mathcal{T}$  writes  $f(x)$  on the tape

# Formal definition of reduction

## Definition (recall from before)

$f: \Sigma^* \rightarrow \Sigma^*$  is **computable** if  $\exists$  a total TM  $T$  with input alphabet  $\Sigma$ , 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  $f: \Sigma^* \rightarrow \Sigma^*$  such that  $x \in L \Leftrightarrow f(x) \in L'$

# Formal definition of reduction

## Definition (recall from before)

$f: \Sigma^* \rightarrow \Sigma^*$  is **computable** if  $\exists$  a total TM  $T$  with input alphabet  $\Sigma$ , 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  $f: \Sigma^* \rightarrow \Sigma^*$  such that  $x \in L \Leftrightarrow f(x) \in L'$

## 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

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

## Proof.

$M\#x \in \text{HP} \Rightarrow M$  halts on  $x \Rightarrow M'$  accepts  $x \Rightarrow M'\#x \in \text{MP}$

$M\#x \notin \text{HP} \Rightarrow M$  does not halt on  $x \Rightarrow M'$  does not halt on  $x \Rightarrow M'\#x \notin \text{MP}$  ■

# Reducing HP to MP

## Definition

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

## Lemma

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

## Proof.

$M\#x \in \text{HP} \Rightarrow M$  halts on  $x \Rightarrow M'$  accepts  $x \Rightarrow M'\#x \in \text{MP}$

$M\#x \notin \text{HP} \Rightarrow M$  does not halt on  $x \Rightarrow M'$  does not halt on  $x \Rightarrow M'\#x \notin \text{MP}$  ■

## Corollary

*MP is not recursive*

## Proof.

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

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

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

## Proof.

$M\#x \in \text{MP} \Rightarrow M$  accepts  $x \Rightarrow M'$  accepts  $x \Rightarrow M'\#x \in \text{HP}$

$M\#x \notin \text{MP} \Rightarrow M$  does not accept  $x \Rightarrow M'$  does not halt on  $x \Rightarrow M'\#x \notin \text{HP}$  ■

# Reducing MP to HP

## Definition

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

## Lemma

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

## Proof.

$M\#x \in \text{MP} \Rightarrow M$  accepts  $x \Rightarrow M'$  accepts  $x \Rightarrow M'\#x \in \text{HP}$

$M\#x \notin \text{MP} \Rightarrow M$  does not accept  $x \Rightarrow M'$  does not halt on  $x \Rightarrow M'\#x \notin \text{HP}$  ■

## Corollary

*HP is not recursive*

## Proof.

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



## 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) ■

## More on reducibility

### Theorem

$\leq$  is reflexive and transitive

### Proof.

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

### 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

## 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

# Other classes of languages

## Question

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

## Regular languages

Accepted by **finite automata**.

## 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)

# Other classes of languages

## Question

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

## Context-free languages

Accepted by **push-down automata**.

## relevance of context-free languages

- programming languages are typically context-free (at first approximation); variations on the language of **matching parentheses** (Dyck language) having  $((()))$  as element, but not  $(())()$ . **block** structure of programs

Many other classes of languages (class useful based on having good closure properties). See the Chomsky hierarchy.

Not in this course, but in Master. Typical questions: parsing? complexity? closure properties?

# Some follow-up courses in Ba

- 1 Logic (Middeldorp)
- 2 Program Verification (Thiemann)
- 3 Term Rewriting (Middeldorp)

in general: concepts used in many courses