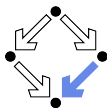


Turing Machines

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

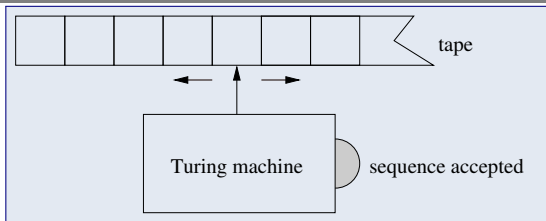
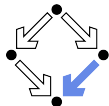
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
<http://www.risc.jku.at>





-
- 1. Turing Machines**
 2. Recognizing Languages
 3. Generating Languages
 4. Computing Functions
 5. The Church-Turing Thesis

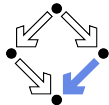
Turing Machine Model



- The machine is always in one of a **finite set of states**.
 - The machine starts its execution in a fixed start state.
- An **infinite tape** holds at its beginning the input word.
 - Tape is read and written and arbitrarily moved by the machine.
- The machine proceeds in a sequence of state **transitions**.
 - Machine reads symbol, overwrites it, and moves tape head left or right.
 - The symbol read and the current state determine the symbol written, the move direction, and the next state.
- If the machine cannot make another transition, it **terminates**.
 - The machine signals whether it is in an accepting state.

If the machine terminates in an accepting state, the word is accepted.

Turing Machines

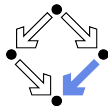


Turing Machine $M = (Q, \Gamma, \sqcup, \Sigma, \delta, q_0, F)$:

- The **state set** Q , a finite set of **states**.
- A **tape alphabet** Γ , a finite set of **tape symbols**.
- The **blank symbol** $\sqcup \in \Gamma$.
- An **input alphabet** $\Sigma \subseteq \Gamma \setminus \{\sqcup\}$.
- The (partial) **transition function** $\delta : Q \times \Gamma \rightarrow_p Q \times \Gamma \times \{ 'L', 'R' \}$,
 - $\delta(q, x) = (q', x', 'L'/'R')$... M reads in state q symbol x , goes to state q' , writes symbol x' , and moves the tape head left/right.
- The **start state** $q_0 \in Q$
- A set of **accepting states (final states)** $F \subseteq Q$.

The crucial difference to an automaton is the infinite tape that can be arbitrarily moved and written.

Example



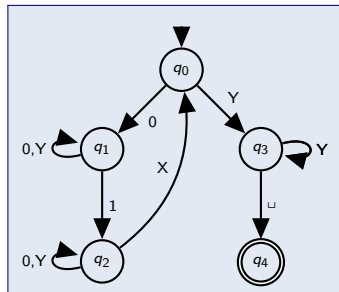
$$M = (Q, \Gamma, \sqcup, \Sigma, \delta, q_0, F)$$

$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$

$$\Gamma = \{\sqcup, 0, 1, X, Y\}$$

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

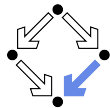
$$F = \{q_4\}$$



δ	\sqcup	0	1	X	Y
q_0	—	(q_1, X, R)	—	—	(q_3, Y, R)
q_1	—	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)
q_2	—	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)
q_3	(q_4, \sqcup, R)	—	—	—	(q_3, Y, R)
q_4	—	—	—	—	—

Machine accepts every word of form $0^n 1^n$ (replacing it by $X^n Y^n$).

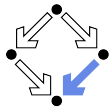
Turing Machine Simulators



For instance, <http://math.hws.edu/TMCM/java/xTuringMachine>.

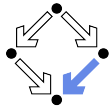
In State	Reading	Write	Move	New State
0	#	1	R	1
0	0	1	R	1
0	1	0	L	0
1	#	#	L	0
1	0	0	R	1
1	1	1	R	1

Generalized Turing Machines



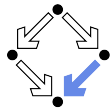
- Infinite tape in **both directions**.
 - Can be simulated by a machine whose tape is infinite in one direction.
- **Multiple** tapes.
 - Can be simulated by a machine with a single tape.
- **Nondeterministic** transitions.
 - We can simulate a nondeterministic M by a deterministic M' .
 - Let r be the maximum number of “choices” that M can make.
 - M' operates with 3 tapes.
 - Tape 1 holds the input (tape is only read).
 - M' writes to tape 2 all finite sequences of numbers $1, \dots, r$.
 - First all sequences of length 1, then all of length 2, etc.
 - After writing sequence $s_1 s_2 \dots s_n$ to tape 2, M' simulates M on tape 3.
 - M' copies the input to tape 3 and performs at most n transitions.
 - In transition i , M attempts to perform choice s_i .
 - If choice i is not possible or M terminates after n transitions in a non-accepting state, M' continues with next sequence.
 - If M terminates in accepting state, M' accepts the input.

Every generalized Turing machine can be simulated by the core form.



-
1. Turing Machines
 - 2. Recognizing Languages**
 3. Generating Languages
 4. Computing Functions
 5. The Church-Turing Thesis

Turing Machine Configurations



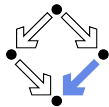
- **Configuration** $a_1 \dots a_k q a_{k+1} \dots a_m$:
 - q : the current state of M .
 - a_{k+1} : the symbol currently under the tape head.
 - $a_1 \dots a_k$: the portion of the tape left to the tape head.
 - $a_{k+2} \dots a_m$: the portion right to the head (followed by $\sqcup \dots$).
- **Move relation**: $a_1 \dots a_k q a_{k+1} \dots a_m \vdash b_1 \dots b_l p b_{l+1} \dots b_m$
If M is a situation described by the left configuration, it can make a transition to the situation described by the right configuration.
 - $a_i = b_i$ for all $i \neq k+1$ and one of the following:
 - $l = k+1$ and $\delta(q, a_{k+1}) = (p, b_l, R)$,
 - $l = k-1$ and $\delta(q, a_{k+1}) = (p, b_{l+2}, L)$.
- **Extended move relation**: $c_1 \vdash^* c_2$
 M can make in an arbitrary number of moves a transition from the situation described by configuration c_1 to the one described by c_2 .

$$c_1 \vdash^0 c_2 : \Leftrightarrow c_1 = c_2$$

$$c_1 \vdash^{i+1} c_2 : \Leftrightarrow \exists c : c_1 \vdash^i c \wedge c \vdash c_2$$

$$c_1 \vdash^* c_2 : \Leftrightarrow \exists i \in \mathbb{N} : c_1 \vdash^i c_2$$

The Language of a Turing Machine



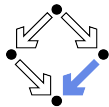
- The **language** $L(M)$ of Turing machine $M = (Q, \Gamma, \sqcup, \Sigma, \delta, q_0, F)$:
The set of all inputs that drive M from its initial configuration to a configuration with an accepting state such that from this configuration no further move is possible:

$$L(M) := \left\{ w \in \Sigma^* \mid \begin{array}{l} \exists a, b \in \Gamma^*, q \in Q : q_0 w \vdash^* a q b \wedge q \in F \\ \wedge \neg \exists a', b' \in \Gamma^*, q' \in Q : a q b \vdash a' q' b' \end{array} \right\}$$

- L is a **recursively enumerable language**:
 - There exists a Turing machine M such that $L = L(M)$.
- L is a **recursive language**:
 - There exists a Turing machine M such that $L = L(M)$ and M terminates for every possible input.

Every recursive language is recursively enumerable; as we will see, the converse does not hold.

Recursiv. Enumerable/Recursive Languages

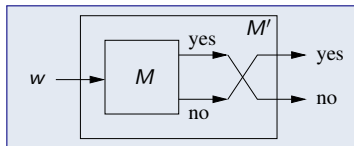


Theorem: L is recursive, if and only if both L and its complement \bar{L} are recursively enumerable.

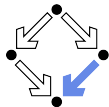
Proof \Rightarrow : Let L be a recursive. Since by definition L is recursively enumerable, it remains to be shown that also \bar{L} is recursively enumerable.

Since L is recursive, there exists a Turing machine M such that M halts for every input w : if $w \in L$, then M accepts w ; if $w \notin L$, then M does not accept w . With the help of M , we can construct the following M' with $L(M') = \bar{L}$:

```
function  $M'(w)$ :  
  case  $M(w)$  of  
    yes: return no  
    no:  return yes  
  end case  
end function
```

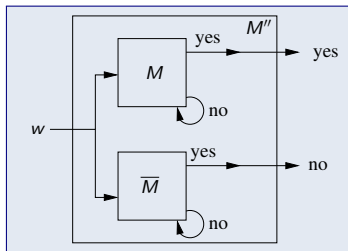


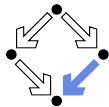
Recursiv. Enumerable/Recursive Languages



Proof \Leftarrow : Let L be such that both L and \bar{L} are recursively enumerable. We show that L is recursive. Since L is r.e., there exists M such that $L = L(M)$ and M halts for $w \in L$ with $M(w) = \text{yes}$. Since \bar{L} is r.e., there exists \bar{M} with $\bar{L} = L(\bar{M})$ and \bar{M} halts for $w \in \bar{L}$ with $\bar{M}(w) = \text{yes}$. We can thus construct M'' with $L(M'') = L$ that always halts:

```
function  $M''(w)$ :  
  parallel  
    begin  
      if  $M(w) = \text{yes}$  then  
        return yes  
      end if  
      loop forever  
    end  
    begin  
      if  $\bar{M}(w) = \text{yes}$  then  
        return no  
      end if  
      loop forever  
    end  
  end parallel  
end function
```





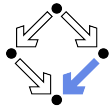
Closure of Recursive Languages

Let L, L_1, L_2 be recursive languages. Then also

- the complement \bar{L} ,
- the union $L_1 \cup L_2$,
- the intersection $L_1 \cap L_2$

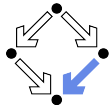
are recursive languages.

Proof by construction of the corresponding Turing machines.



-
1. Turing Machines
 2. Recognizing Languages
 - 3. Generating Languages**
 4. Computing Functions
 5. The Church-Turing Thesis

Enumerators

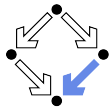


Turing machine $M = (Q, \Gamma, \sqcup, \emptyset, \delta, q_0, F)$ with special symbol $\# \in \Gamma$.

- M is an **enumerator**, if M has an additional **output tape** on which
 - M moves its tape head only to the right, and
 - M writes only symbols different from \sqcup .
- The **generated language** $Gen(M)$ of enumerator M is the set of all words that M eventually writes on its output tape.
 - The end of each word is marked by a trailing $\#$.

M may run forever and thus $Gen(M)$ may be infinite.

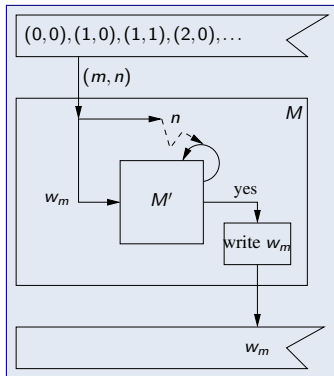
Recognizing versus Generating Languages



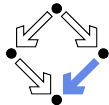
Theorem: L is recursively enumerable, if and only if there exists some enumerator M such that $L = \text{Gen}(M)$.

Proof \Rightarrow : Let L be recursively enumerable, i.e., $L = L(M')$ for some M' . We construct enumerator M such that $L = \text{Gen}(M)$.

```
procedure M:
  loop
    produce next  $(m, n)$  on working tape
    if  $M'(w_m) = \text{yes}$  in at most  $n$  steps then
      write  $w_m$  to output tape
    end if
  end loop
end procedure
```

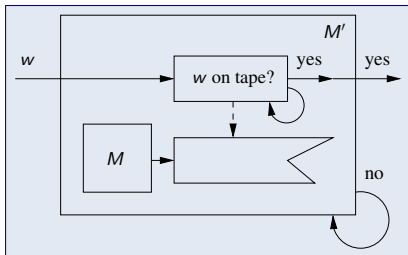


Recognizing versus Generating Languages



Proof \Leftarrow : Let L be such that $L = \text{Gen}(M)$ for some enumerator M . We show that there exists some Turing machine M' such that $L = L(M')$.

```
function  $M'(w)$ :  
  while  $M$  is not terminated do  
     $M$  writes next word  $w'$   
    if  $w = w'$  then  
      return yes  
    end if  
  end while  
  return no  
end function
```

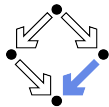


Recognizing is possible, if and only if generating is possible.



-
1. Turing Machines
 2. Recognizing Languages
 3. Generating Languages
 - 4. Computing Functions**
 5. The Church-Turing Thesis

Functions



Take binary relation $f \subseteq A \times B$.

- $f : A \rightarrow B$: f is a **total function** from A to B .
 - For every $a \in A$, there is **exactly one** $b \in B$ such that $(a, b) \in f$.
- $f : A \rightarrow_p B$: f is a **partial function** from A to B .
 - For every $a \in A$, there is **at most one** $b \in B$ such that $(a, b) \in f$.
- Auxiliary notions:

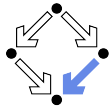
$$\text{domain}(f) := \{a \mid \exists b : (a, b) \in f\}$$

$$\text{range}(f) := \{b \mid \exists a : (a, b) \in f\}$$

$$f(a) := \text{such } b : (a, b) \in f$$

Every total function $f : A \rightarrow B$ is a partial function $f : A \rightarrow_p B$; every partial function $f : A \rightarrow_p B$ is a total function $f : \text{domain}(f) \rightarrow B$.

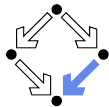
Functions



- Let $f : \Sigma^* \rightarrow_p \Gamma^*$ where $\sqcup \notin \Sigma \cup \Gamma$.
 - f is a function over words in some alphabets.
- f is **Turing computable**, if there exists a Turing machine M such that
 - for input w (i.e. initial tape content $w \sqcup \dots$), M terminates in an accepting state, if and only if $w \in \text{domain}(f)$;
 - for input w , M terminates in an accepting state with output w' (i.e. final tape content $w' \sqcup \dots$), if and only if $w' = f(w)$.
- **Not every function $f : \Sigma^* \rightarrow_p \Gamma^*$ is Turing computable:**
 - The set of all Turing machines is countably infinite: all machines can be ordered in a single list (in the alphabetic order of their definitions).
 - The set of all functions $\Sigma^* \rightarrow_p \Gamma^*$ is more than countably infinite (Cantor's diagonalization argument).
 - Consequently, there are more functions than Turing machines.

M computes f , if M terminates for arguments in the domain of f with output $f(a)$ and does not terminate for arguments outside the domain.

Example



We show that natural number subtraction is Turing computable.

- Subtraction \ominus on \mathbb{N} :

$$m \ominus n := \begin{cases} m - n & \text{if } m \geq n \\ 0 & \text{else} \end{cases}$$

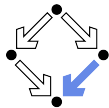
- Unary representation of $n \in \mathbb{N}$:

$$\underbrace{000\dots 0}_{n \text{ times}} \in L(0^*)$$

- Input $00 \sqcup 0$ shall lead to output 0 .
 - $2 \ominus 1 = 1$.

Idea: replace every pair of 0 in m and n by \sqcup .

Example (Contd)

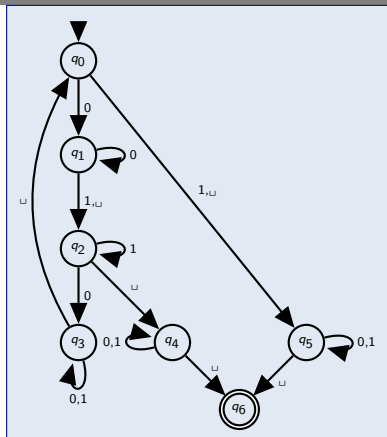


$$M = (Q, \Gamma, \sqcup, \Sigma, \delta, q_0, F)$$

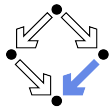
$$Q = \{q_0, \dots, q_6\}$$

$$\Sigma = \{0\}, \Gamma = \{0, 1, \sqcup\}, F = \{q_6\}$$

δ	0	1	\sqcup
q_0	(q_1, \sqcup, R)	(q_5, \sqcup, R)	(q_5, \sqcup, R)
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	$(q_2, 1, R)$
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, \sqcup, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, \sqcup, R)
q_4	$(q_4, 0, L)$	(q_4, \sqcup, L)	$(q_6, 0, R)$
q_5	(q_5, \sqcup, R)	(q_5, \sqcup, R)	(q_6, \sqcup, R)
q_6	—	—	—



- In q_0 , the leading 0 is replaced by \sqcup .
- In q_1 , M searches for the next \sqcup and replaces it by a 1.
- In q_2 , M searches for the next 0 and replaces it by 1, then moves left.
- In q_3 , M searches for previous \sqcup , moves right and starts from begin.
- In q_4 , M has found a \sqcup instead of 0 and replaces all previous 1 by \sqcup .
- In q_5 , n is (has become) 0; the rest of the tape is erased.
- In q_6 , the computation successfully terminates.



Example (Contd)

■ $2 \ominus 1 = 1$:

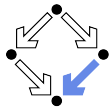
$q_0 0 \sqcup 0 \vdash \sqcup q_1 0 \sqcup 0 \vdash \sqcup 0 q_1 \sqcup 0 \vdash \sqcup 0 1 q_2 0$
 $\vdash \sqcup 0 q_3 1 1 \vdash \sqcup q_3 0 1 1 \vdash q_3 \sqcup 0 1 1 \vdash \sqcup q_0 0 1 1$
 $\vdash \sqcup \sqcup q_1 1 1 \vdash \sqcup \sqcup 1 q_2 1 \vdash \sqcup \sqcup 1 1 q_2 \vdash \sqcup \sqcup 1 q_4 1$
 $\vdash \sqcup \sqcup q_4 1 \vdash \sqcup q_4 \vdash \sqcup 0 q_6$

■ $1 \ominus 2 = 0$:

$q_0 0 \sqcup 0 0 \vdash \sqcup q_1 \sqcup 0 0 \vdash \sqcup 1 q_2 0 0 \vdash \sqcup q_3 1 1 0$
 $\vdash q_3 \sqcup 1 1 0 \vdash \sqcup q_0 1 1 0 \vdash \sqcup \sqcup q_5 1 0 \vdash \sqcup \sqcup \sqcup q_5 0$
 $\vdash \sqcup \sqcup \sqcup \sqcup q_5 \vdash \sqcup \sqcup \sqcup \sqcup \sqcup q_6$.

For $m > n$, leading blanks still have to be removed.

Turing Computability



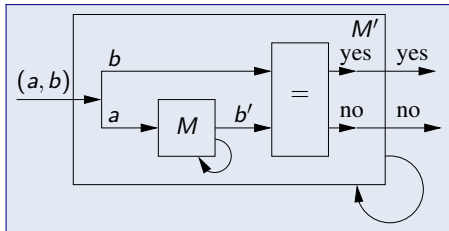
Theorem: $f : \Sigma^* \rightarrow_p \Gamma^*$ is Turing computable, if and only if

$$L_f := \{(a, b) \in \Sigma^* \times \Gamma^* \mid a \in \text{domain}(f) \wedge b = f(a)\}$$

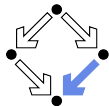
is recursively enumerable.

Proof \Rightarrow : Since $f : \Sigma^* \rightarrow_p \Gamma^*$ is Turing computable, there exists a Turing machine M which computes f . To show that L_f is r.e., we construct M' with $L(M') = L_f$:

```
function  $M'(a, b)$ :  
   $b' \leftarrow M(a)$   
  if  $b' = b$  then  
    return yes  
  else  
    return no  
  end if  
end function
```

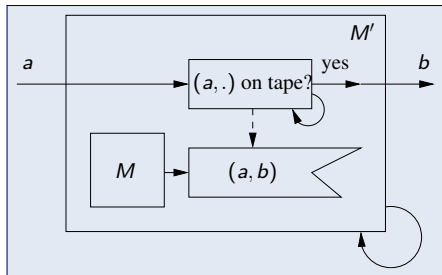


Turing Computability



Proof \Leftarrow : Since L_f is recursively enumerable, there exists an enumerator M with $\text{Gen}(M) = L_f$. We construct the following Turing machine M' which computes f :

```
function  $M'(a)$ :  
  while  $M$  is not terminated do  
     $M$  writes  $(a', b')$  to tape  
    if  $a = a'$  then  
      return  $b'$   
    end if  
  end while  
  loop forever  
end function
```

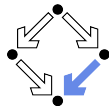


Computing is possible, if and only if recognizing is possible.



-
1. Turing Machines
 2. Recognizing Languages
 3. Generating Languages
 4. Computing Functions
 - 5. The Church-Turing Thesis**

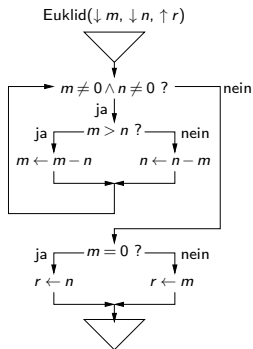
Algorithms



Computer science is based on algorithms.

Compute as follows the greatest common divisor of two natural numbers m, n that are not both 0:

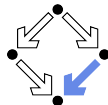
1. If $m = 0$, the result is n .
2. If $n = 0$, the result is m .
3. If $m > n$, subtract n from m and continue with step 1.
4. Otherwise subtract m from n and continue with step 1.



```
Euklid( $\downarrow m, \downarrow n, \uparrow r$ ):  
  while  $m \neq 0 \wedge n \neq 0$  do  
    if  $m > n$   
      then  $m \leftarrow m - n$   
      else  $n \leftarrow n - m$   
  if  $m = 0$   
    then  $r \leftarrow n$   
    else  $r \leftarrow m$   
end Euklid.
```

What is an “algorithm” and what is computable by an algorithm?

The Church-Turing Thesis



Church-Turing Thesis: Every problem that is solvable by an algorithm (in an intuitive sense) is solvable by a Turing machine. Thus the set of intuitively computable functions is identical with the set of Turing computable functions.

- Replaces fuzzy notion “algorithm” by precise notion “Turing machine”.
- Unprovable thesis, exactly because the notion “algorithm” is fuzzy.
- Substantially validated, because many different computational models have no more computational power than Turing machines.
 - Random access machines, loop programs, recursive functions, goto programs, λ -calculus, rewriting systems, grammars, ...

Turing machines represent the most powerful computational model known, but there are many other equally powerful (“Turing complete”) models.